Non-Transferable Anonymous Tokens by Secret Binding

F. Betül Durak
betul.durak@microsoft.com
Microsoft Research
United States

Laurane Marco
laurane.marco@epfl.ch
EPFL
Switzerland

Abdullah Talayhan
abdullah.talayhan@epfl.ch
EPFL
Switzerland

Serge Vaudenay
serge.vaudenay@epfl.ch
EPFL
Switzerland

ABSTRACT

Non-transferability (NT) is a security notion which ensures that credentials are only used by their intended owners. Despite its importance, it has *not* been formally treated in the context of anonymous tokens (AT) which are lightweight anonymous credentials. In this work, we consider a client who "buys" access tokens which are forbidden to be transferred although anonymously redeemed. We extensively study the trade-offs between privacy (obtained through anonymity) and security in AT through the notion of non-transferability. We formalise new security notions, design a suite of protocols with various flavors of NT, prove their security, and implement the protocols to assess their efficiency. Finally, we study the existing anonymous credentials which offer NT, and show that they cannot automatically be used as AT without security and complexity implications.

1 INTRODUCTION

The notion of anonymity has been introduced and proposed to be used as a mechanism to protect the privacy of consumers in almost any system. On the one hand, it provides a strong tool to protect honest users from being tracked, on the other hand, it creates an opportunity for misbehaviour and fraud, thus creating a security threat. In recent years, there has been a growing interest in designing one-time anonymous tokens for applications such as private browsing with Privacy Pass (PP) [1] and Trust Tokens (TT)[1], private access to services[2] [2], private statistics aggregation [3], or even for private contact tracing [4].

These protocols are usually designed in a client-server setting[3]. They first require to establish some form of "trust" (that we keep informal for now) to the client before issuing any anonymity-preserving access token. In PP, the "trust" is established with CAPTCHA [5] to prove human-hood. In other settings, the system integrates the client-authenticity verification with a trusted attestor; or it may introduce an identity manager (IdM) as a proxy to authenticate the client with an account before issuing anonymous tokens as in [3].

Upon establishing the "trust", the protocol operates in two phases: issuance and redemption. During the issuance phase, the client obtains a token through a "signature" by the server on a random (and masked) message given by the client. During the redemption phase, the client redeems the unmasked token to the server who either accepts or rejects it. The protocol must satisfy two properties: (1) the security: the client cannot produce valid tokens without the help of the server (one-more unforgeability), and (2) privacy: the server must not be able to link the issuance and redemption sessions of a token (unlinkability).

[1]https://web.dev/trust-tokens/
[2]https://developer.apple.com/news/?id=huqjyh7k
[3]It is not unusual to involve more parties such as a trusted third party as an attestor.

When tokens have a form of value which is somehow charged to the client, "trust" must become strongly enforced. This implies that a token cannot be charged without the client's consent (what we call *accountability*) and sometimes that only the charged client can redeem the token (what we call *non-transferability*). Still, redeem should remain anonymous.

In this work, we use digital identities (defined by a key pair) as a way to establish trust and specifically issue the tokens to the identities defined by a public key. The client holds a corresponding secret key. When the client attempts to redeem, they will be asked to prove (in zero-knowledge) the knowledge of the secrets associated to their identity. Although token issuance with trust establishment is based on a public key, redeem with ownership proof remains anonymous. Of course, this assumes that the network traffic does not compromise privacy.

1.1 Non-Transferability

Anonymous token protocols implicitly assume that the client redeeming the token corresponds to the one to which it was issued. Ensuring this property is crucial as we detail next and discuss more extensively in Appendix C.

Token sharing and transferring. As explored in the context of anonymous credentials [6–8], sharing or unauthorized transfer of anonymous credentials is a threat to the systems. One time anonymous tokens are the optimized lightweight version of anonymous credentials with a randomly selected message being the attribute. Hence, unlike anonymous credentials which carry a value in the form of identification data with privacy protection, one-time tokens do not embed any meaningful data. Nevertheless, they cannot be issued to clients without a trust establishment, specifically when they are used for access control. For example, CAPTCHAs were introduced as proof-of-humanhood and solving them is considered as a trust signal. Because asking to solve CAPTCHAs was costly for honest users, token issuance were processed in batches in Privacy Pass [1]. Even though the tokens do not embed a valuable information, it is essential to maintain the initial trust establishment due to the economy it creates for malicious actors with "human bots". Since tokens being transferable would imply transferring said established trust with them, without prevention of transferability of anonymous tokens, demanded privacy from users can easily be abused. This problem cannot be prevented by enforcing a login information during the issuance.

This security threat highlight the importance of ensuring *non-transferability*: *A token should only be allowed to be redeemed by its assigned owner.*

Transferring can happen when tokens have no intrinsic value for the client once the client owns the token. Assuming that the client's secret key has some intrinsic value and that redeem implies using the secret key, the client may become more reluctant to transfer the token. This is essentially our approach: to bind the token to a valuable secret and to make redeem impossible without the secret.

Our Setting. In this work, we consider a setting where we don't separate the user from their device in the sense that a client refers to both the user and the device and a (possibly malicious) client can program the device however they want. This is a stronger setting than assuming a trusted hardware acting as an interface between the user and the protocols [8]. We assume that each client owns a key pair $(\mathsf{sk}_c, \mathsf{pk}_c)$ on the client side. We wish to obtain the following properties for a non-transferable anonymous token protocol: a client cannot request a token made for a specific client public key pk_c, without holding the corresponding secret key sk_c. When redeeming their token, the client must stay anonymous, but they should not be able to redeem a token without proving knowledge of the secret key sk_c associated to the public key pk_c for which the token was issued.

1.2 Our Results

In this work, we formalise the notion of a non-transferable one-time anonymous token (NTAT), we give an algebraic MAC-based construction and analyse its security and performance. More precisely, we formalize the NTAT framework via a key pair on the client side which allows to tie a token to a client's identity, and makes the redemption procedure interactive to require the client to prove knowledge of the secret key associated to the token they are redeeming.

We formalise the security of NTAT, which we study under three notions: *accountability* ensures that every issued token is approved by the server and accounted to a given client public key pk_c (it includes one-more unforgeability and request unforgeability); *unlinkability*, which is kept from the original anonymous tokens settings; and *non-transferability*, which is composed of the soundness and uniqueness notions, and captures the fact that a client who can successfully redeem the token must own the unique client secret key on which the token was issued.

We provide an NTAT construction, and a variant that achieves public verifiability of tokens through the use of pairings (App. D.1). We analyze their complexity and communication cost in comparison to some state of the art protocols. We provide additional variants of our construction in Appendix D. One with more attributes, one based on an honestly used trusted hardware to defeat attacks based on a maliciously used trusted hardware, one where client authentication is kept separate from the issuance protocol for a complexity advantage and one that binds the client's secret key to an external valuable secret.

We then proceed to prove their security. One-more unforgeability (OMUF) relies on the Algebraic Group Model (AGM) [9], the random oracle model (ROM), and the hardness of the discrete logarithm problem (DLog), unlinkability relies on DLog and ROM, request unforgeability and soundness are based on information theory, and uniqueness requires the hardness of the DLog. We use the AGM [9] (only for OMUF) as it helps proving security without making protocols more complex than necessary.

Design choices We introduce and justify a few design choices.

Firstly, we only consider anonymous tokens with interactive redemption, which we formally define in Theorem 2.1. Indeed, if redemption is non-interactive, then non-transferability is incompatible with the other security notions of accountability and unlinkability. We discuss this more extensively in subsection 2.4.

Then, we give a construction based on algebraic MACs since their algebraic structure allows for efficient (non-interactive) zero-knowledge proofs. Alternative construction based on oPRFs (oblivious pseudo-random functions) such as [10] were considered but they require to prove knowledge of the pre-image of a hash function, which is expensive.

Finally, our construction requires the client to authenticate during issuance, in the form of proving the knowledge of their secret key, since it otherwise leads to an attack on the one-more unforgeability security as highlighted in Appendix E.

1.3 Related Work

Non-transferability is a known and studied problem in the context of anonymous credentials [6, 8, 11]. In this section, we summarize the related and previous work and the security notion of non-transferability.

Anonymous Credentials (AC). AC protocols (between an issuer and a verifier) were introduced in the form of blind signatures by Chaum [12, 13]. In AC, a client wants to show a type of "proof" to a verifier to prove that they own a set of attributes (for instance: $\mathsf{Attr} = [\mathsf{state} = \text{“NY”}, \mathsf{age} = 34]$). The attributes should satisfy two conditions: (1) they have been verified and used to issue a credential by the issuer (issuance phase) and (2) they verify a public predicate (for instance, $\mathsf{state} = \text{“NY”} \wedge \mathsf{age} \geq 18$) (show phase). Anonymity holds in the sense that no more than verified predicates is revealed about the attributes, and no linkable information between issuance and show sessions is given.

There exist many variations of the anonymity notion. Attributes could be commitments to a hidden value where the issuer would see only a commitment to them or they could be the plain attributes shown clearly to the issuer. The show phase could reveal a rerandomized commitment while verifying the predicate anonymously, or it could simply reveal a given subset of plain attributes.

In 2001, Camenish and Lysyanskaya [6] introduced a variant of anonymous credentials, that they call non-transferable, based on RSA assumptions. Baldimsti and Lysyanskaya [14], give another variant of anonymous credentials that is still based on blind signatures. Their scheme was allegedly affected by attacks on ROS [15–17], but recent work [18] has dis-proven this claim.

U-Prove [11] is an implementation of AC based on Brands [19]. In U-Prove, the issuer must see the plain attributes which are put in credentials. The client can separate the attributes as those which are shown to the verifier, those which are revealed as a commitment, or those which remain hidden (though their existence is not hidden).

The issuing phase is a 3-move protocol and the show phase is non-interactive. The security of U-Prove is not formally proven.

U-Prove also describes an option to add the public key of a device as an attribute and to show ownership without revealing it during the show phase. This option is added to make the credentials device-protected. To achieve device-bound, it is suggested to embed the secret key of the device as an additional attribute and making the device prove that it knows the attribute (i.e. the secret key in a zero-knowledge way) during the show phase.

Keyed-Verification Anonymous Credentials (KVAC). In AC, if the verifier has to use the secret of the issuer during the show phase, the scheme becomes a KVAC [20]. Doing so offers performance advantage. However, both AC and KVAC can be used simultaneously: an upper AC provides credentials, and proving credentials to a specific verifier (with their own key) would transform in a KVAC that could be reused to the same verifier.

Barki et al. [21] describe a KVAC protocol which issues credentials on a commitment to the attributes. Showing credentials only requires the commitment and the credentials so does not provide non-transferability. They build a KVAC scheme without pairing and also transform it into an AC scheme using pairing. Their protocol relies on an algebraic MAC which we will use as a core primitive for our protocol.

Trusted Hardware as a Solution. Hanzlick and Slamanig [8] propose a non-transferable AC protocol called CHAC. In their setting, a client has their secret key kept by a trusted hardware such as a SIM card and that using the secret would not be possible without its help. The client is composed of a "core" (trusted hardware) and a "helper" (smartphone/computer). There are interaction between the two. The role of the core is to keep the secret key safe and the role of the helper is to minimize the tasks of the core.

They rely on the discrete logarithm assumption in bilinear groups. The issuance and the show protocols have 3 moves and are initiated by the server (to send a nonce). Furthermore, it is a multi-show system: the server cannot detect if a client uses their credentials several times.[4]

Anonymous Tokens. Enforcing a credential to be used only once is done by adding an extra attribute t (called a tag), serving as a unique identifier of the credential which is revealed at the show phase. Anonymous tokens are lightweight single-show credentials in which the number of attributes is restricted.

There exists various proposals to extend the tokens with more functionalities such as embedding a bit [22–24], or private vs public verifiability [4], or embedding more attributes [23, 24], making them look closer to anonymous credentials [13]. Each of these functionalities may require different underlying primitives such as blind signatures [12] (providing public verifiability), oblivious PRFs (oPRFs) [25] (first used in Privacy Pass [1] for efficiency and private verifiability) or algebraic MACs [23, 24, 26]. In this work, we based our constructions on (algebraic) MACs.

[4]The paper [8, Footnote 4] claims that it is easy to transform a multi-show protocol into a single-show protocol by adding a tag as an attribute and showing it. However, given that it must be shown at issuance as well, unlinkability would be broken.

1.4 Notations

We introduce some notation used throughout the paper. We write $1_{\text{predicate}}$ to refer to the indicator function which outputs 1 if predicate is true and 0 otherwise. We use the superscript oracles for convenience to denote oracle access to all the oracles of a given security game, whose names are denoted with a starting "O". The notation 1^λ denotes the security parameter λ in unary form.

We use two assumptions in our proofs: DLog and n-DLog as follows. Given an additive group $\mathbb{G}$ of a prime order q with a random generator G, DLog assumption states that for a given random $Y \in \mathbb{G}$, the adversary cannot find x such that $Y = x \cdot G$. And, n-DLog assumption states that given a set of $\left\{G, \left(Y_i = x^i \cdot G\right)_{i \in \{1,\dots,n\}}\right\}$, the adversary cannot find x.

2 NON-TRANSFERABLE ANONYMOUS TOKENS

The goal of non-transferability is to make sure that the redeemer of an anonymous token is the same client who requested the token in order to prevent stealing of tokens as well as unauthorized transfer of tokens.

More precisely, we want to have the following properties : each token must be *accounted to one public-key* pk_c and each issuance should be *authorized by the corresponding client*; each token is *issued obliviously* and *redeemed anonymously*, hence the public key pk_c cannot be revealed at redeem ; the client should *not be able to redeem a token without proving knowledge of the secret key* sk_c associated to the public key pk_c for which the token was issued.

We give the formal framework below. We focus on a two-move issuance protocol. We separate it with three algorithms Client.Query, Server.Issue, Client.Final instead of presenting it as a single interactive protocol. We then formalize the security notions of one time anonymous tokens. We enrich the standard one-more unforgeability and unlinkability notions to fit our new formalism, and we introduce a new notion of non-transferability.

2.1 A framework for non-transferable anonymous tokens

Definition 2.1 (Anonymous Tokens with Interactive Redemption). An Anonymous Token with Interactive Redemption is a tuple of algorithms defined in the following way:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: a probabilistic polynomial time algorithm that takes as input the security parameter λ and returns public parameters pp.
- $\mathsf{Server.KeyGen}(\mathsf{pp}) \to (\mathsf{sk}_s, \mathsf{pk}_s)$: a probabilistic polynomial time algorithm that takes as input pp and generates a private-public key pair for the server.
- $\mathsf{Client.KeyGen}(\mathsf{pp}) \to (\mathsf{sk}_c, \mathsf{pk}_c)$: a probabilistic polynomial time algorithm that takes as input pp and generates a private-public key pair for the client. Additionally, we use a predicate $\mathsf{KeyVer}(\mathsf{pp}, \mathsf{sk}_c, \mathsf{pk}_c)$ to tell whether a key pair is valid. (The predicate must be easy to evaluate.)
- $\mathsf{Client.Query}(\mathsf{pp}, \mathsf{sk}_c, \mathsf{pk}_s) \to (\mathsf{query}, \mathsf{st})$: the first step of an interactive issuance algorithm between client and server.

```
Correctness(λ)
1: $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$
2: $\mathsf{Server.KeyGen}(\mathsf{pp}) \to (\mathsf{sk}_s, \mathsf{pk}_s)$
3: $\mathsf{Client.KeyGen}(\mathsf{pp}) \to (\mathsf{sk}_c, \mathsf{pk}_c)$
4: if $\neg\mathsf{KeyVer}(\mathsf{pp}, \mathsf{sk}_c, \mathsf{pk}_c)$ then return 0
5: $\mathsf{Client.Query}(\mathsf{pp}, \mathsf{sk}_c, \mathsf{pk}_s) \to (\mathsf{query}, \mathsf{st})$
6: $\mathsf{Server.Issue}(\mathsf{pp}, \mathsf{sk}_s, \mathsf{pk}_c, \mathsf{query}) \to \mathsf{resp}$
7: $\mathsf{Client.Final}(\mathsf{st}, \mathsf{resp}) \to (\sigma, \omega)$
8: $\langle \mathsf{Client.Prove}(\mathsf{pp}, (\sigma, \omega), \mathsf{sk}_c, \mathsf{pk}_s), \mathsf{Server.Verify}(\mathsf{pp}, \sigma, \mathsf{sk}_s) \rangle \to b$
9: return $b$
```

Figure 1: Correctness game (Definition 2.2).

The client uses their secret key and pp to generate a request query for a token and saves their state st for future steps.[5]

- $\mathsf{Server.Issue}(\mathsf{pp}, \mathsf{sk}_s, \mathsf{pk}_c, \mathsf{query}) \to \mathsf{resp}/\perp$: the second step of issuance, where the server takes the client query, client public key, and server secret key and generates a response. If the query is not valid, the server returns $\perp$.
- $\mathsf{Client.Final}(\mathsf{st}, \mathsf{resp}) \to (\sigma, \omega)$: the third and final step of issuance, where the client takes the server's response from the previous step, as well as the client's previous state, and generates a token σ (which is essentially a blind commitment to the client's secret key), as well as decommitment information ω to the token.
- $\langle \mathsf{Client.Prove}(\mathsf{pp}, \sigma, \omega, \mathsf{sk}_c, \mathsf{pk}_s), \mathsf{Server.Verify}(\mathsf{pp}, \sigma, \mathsf{sk}_s) \rangle \to 0/1$: Given a token σ, the redemption protocol consists of an interactive proof of knowledge of (ω, sk_c) between the client (prover) and the server (verifier) to ensure that the client knows the secret key sk_c and the decommitment information ω.
  Client.Prove and Server.Verify are two interactive algorithms. For instance, Server.Verify initially takes input $(\mathsf{pp}, \sigma, \mathsf{sk}_s)$ and returns a state for itself. Then, it inputs the state and a message from the client and returns a new state for itself and a response for the client. It continues until the new state is $\perp$ (ideally one round-trip communication should happen), in which case the new response is the final output of the protocol.[6]

For the rest of this paper, when we refer to anonymous tokens (AT) we will assume interactive redemption unless mentioned otherwise.

Definition 2.2 (Correctness). An AT scheme is correct if for all λ and for all randomness used within the algorithms Correctness game (Figure 1) returns 1.

Definition 2.3 (Non-Transferable Anonymous Tokens). We say that an AT is a Non-Transferable Anonymous Token (NTAT) if it has the following security properties : it is accountable (Section 2.2), unlinkable (Section 2.3) and non-transferable (Section 2.4).

We define the aforementioned security notions in the following sections.

[5]In previous works, this algorithm also takes as input from the client a nonce t which is used to prevent double-spending. In our protocol, double spending is prevented by using a unique σ

[6]We also consider variants of AT where Server.Verify does not use sk_s. I.e., the server's secret is not needed in redeem. In Appendix D.1 we propose such variant.

```
OMUF(𝒜)
1: $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$
2: $\mathsf{Server.KeyGen}(\mathsf{pp}) \to (\mathsf{sk}_s, \mathsf{pk}_s)$
3: $q[\cdot] \leftarrow 0$;  $\mathsf{st}[\cdot] \leftarrow \emptyset$
4: $((\mathsf{sk}_c, \mathsf{pk}_c), (\sigma_i, \omega_i)_{i \in \{1,\ldots,q\}}) \leftarrow \mathcal{A}^{\mathsf{OIssueTok},\mathsf{ORedeem}}(\mathsf{pp}, \mathsf{pk}_s)$
5: if $q \le q[\mathsf{pk}_c]$ then return 0
6: if $\exists i \neq j$ s.t. $\sigma_i = \sigma_j$ then return 0
7: if $\neg\mathsf{KeyVer}(\mathsf{pp}, \mathsf{sk}_c, \mathsf{pk}_c)$ then return 0
8: for $i = 1$ to $q$ do
9:     $\langle \mathsf{Client.Prove}(\mathsf{pp}, \sigma_i, \omega_i, \mathsf{sk}_c, \mathsf{pk}_s), \mathsf{Server.Verify}(\mathsf{pp}, \sigma_i, \mathsf{sk}_s) \rangle \to \mathsf{out}$
10:    if out = 0 then return 0
11: return 1

OIssueTok(pk_c, query)
12: $q[\mathsf{pk}_c] \leftarrow q[\mathsf{pk}_c] + 1$
13: $\mathsf{resp} \leftarrow \mathsf{Server.Issue}(\mathsf{pp}, \mathsf{sk}_s, \mathsf{pk}_c, \mathsf{query})$
14: return resp

ORedeem(sid, input)
15: if st[sid] does not exist then
16:     $\mathsf{st}[\mathsf{sid}] \leftarrow (\mathsf{pp}, \mathsf{input}, \mathsf{sk}_s)$
17:     return
18: $\mathsf{Server.Verify}(\mathsf{st}[\mathsf{sid}], \mathsf{input}) \to (\mathsf{st}', \mathsf{output})$
19: $\mathsf{st}[\mathsf{sid}] \leftarrow \mathsf{st}'$
20: return output
```

Figure 2: One-More Unforgeability game (Definition 2.4).

2.2 Accountability Notion

We define the notion saying that every issued token must have been approved by the server and somehow "charged" to a specific public key pk_c upon request by its owner. For that, the main ingredient is the notion of one-more unforgeability (OMUF) to which we add the notion of request unforgeability (RUF). The former notion ensures that an adversary cannot forge q valid tokens without making q issuance requests to the server. This captures the case where the server is honest and all the clients are malicious by default. However, this alone is not enough to provide full security with accountability. With the RUF notion, an adversary cannot query the server to charge a public key without knowing the corresponding secret key. For this, one client is honest and other participants are malicious by default. These two notions make the system accountably secure.

One-More UnForgeability. We formalize the idea that an adversary cannot make $q + 1$ tokens when it requests the issuance of q tokens. Compared to the standard notion of OMUF, in our game, we have to count issuance sessions for the same pk_c and focus on the adversary not being able to make more tokens for the target pk_c.

Definition 2.4 (One-more unforgeability). An AT is one-more unforgeable if for any PPT adversary $\mathcal{A}$ we have

$$\mathsf{Adv}^{\mathsf{OMUF}}(\mathcal{A}) = \Pr[\mathsf{OMUF}(\mathcal{A}) \to 1] = \mathsf{negl}(\lambda)$$

where the $\mathsf{OMUF}(\mathcal{A})$ game is defined in Figure 2.

In OIssueTok, $\mathcal{A}$ can request the issuance of a token to be charged on key pk_c. The $q[\mathsf{pk}_c]$ counter is the number of issued tokens which have been charged to pk_c. Essentially, $\mathcal{A}$ tries to get more than $q[\mathsf{pk}_c]$ tokens associated to pk_c.

In ORedeem, $\mathcal{A}$ can launch a redeem session sid with the server by setting the token σ in input. This first call will define the inner state of the server for this redeem session. The last call to ORedeem sends the last message from the client. It sets the inner state of the server to $\perp$ and defines the final output of the protocol $\mathsf{out}_{\mathsf{sid}}$.

$\mathcal{A}$ is providing a key pair. The key sk_c is needed to check that the tokens are valid. The key pk_c is needed to check how many

```
RUF(𝒜)
1: Setup(1^λ) → pp
2: Q_query, Q_final ← ∅
3: ClientKeyGen(pp) → (sk_c, pk_c)
4: (query, sk_s) ← 𝒜^{ORequest,OFinal,ORedeem}(pp, pk_c)
5: if ∃j query = query_j then abort
6: resp ← Server.Issue(pp, sk_s, pk_c, query)
7: if resp = ⊥ then abort
8: return 1

ORequest(j, pk_s)
9: if j ∈ Q_query then abort
10: insert j in Q_query
11: Client.Query(pp, sk_c, pk_s) → (query_j, st_j)
12: return query_j

OFinal(j, resp)
13: if j ∈ Q_final or j ∉ Q_query then abort
14: insert j in Q_final
15: resp_j ← resp
16: Client.Final(pp, st_j, resp_j) → (σ_j, ω_j)
17: out_j ← 1_{σ_j ≠ ⊥}
18: return out_j

ORedeem(sid, input)
19: if st[sid] does not exist then
20:     (j, pk_s) ← input
21:     if j ∉ Q_final then abort
22:     st[sid] ← (pp, σ_j, ω_j, sk_c, pk_s)
23:     return σ_j
24: Client.Prove(st[sid], input) → (st′, output)
25: st[sid] ← st′
26: return output
```

Figure 3: Request unforgeability game (Definition 2.5).

```
UNLINK_2(𝒜)
1: Setup(1^λ) → pp
2: Server.KeyGen(pp) → (sk_s, pk_s)
3: Q_query, Q_final, Q_redeem ← ∅
4: i ← 0; b ←$ {0, 1}
5: (j_0, j_1, state) ← 𝒜_1^{oracles}(pp, sk_s, pk_s)
6: if {j_0, j_1} ⊈ Q_final − Q_redeem then abort
7: if out_{j_0} = 0 or out_{j_1} = 0 then abort
8: if b = 1 then
9:     swap (usr_{j_0}, σ_{j_0}, ω_{j_0}) and (usr_{j_1}, σ_{j_1}, ω_{j_1})
10: b′ ← 𝒜_2^{oracles}(state)
11: return 1_{b=b′}

ORequest(j, usr)
12: if j ∈ Q_query then abort
13: insert j in Q_query
14: usr_j ← usr
15: Client.Query(pp, sk_{c,usr}, pk_s) → (query_j, st_j)
16: return query_j

OFinal(j, resp)
17: if j ∈ Q_final or j ∉ Q_query then abort
18: insert j in Q_final
19: resp_j ← resp
20: Client.Final(pp, st_j, resp_j) → (σ_j, ω_j)
21: out_j ← 1_{σ_j ≠ ⊥}
22: return out_j

ORedeem(sid, input)
23: if st[sid] does not exist then
24:     j ← input
25:     if j ∉ Q_final then abort
26:     insert j in Q_redeem
27:     usr ← usr_j
28:     st[sid] ← (pp, σ_j, ω_j, sk_{c,usr}, pk_s)
29:     return σ_j
30: Client.Prove(st[sid], input) → (st′, output)
31: st[sid] ← st′
32: return output

OClientKeyGen()
33: increment i
34: ClientKeyGen(pp) → (sk_{c,i}, pk_{c,i})
35: return (sk_{c,i}, pk_{c,i})
```

Figure 4: Unlinkability game (Definition 2.6).

tokens have been accounted. The two keys must satisfy the KeyVer predicate. However, both keys can be maliciously generated.

We could have formalized a stronger notion in which redeem is done by a malicious prover, without having to produce a consistent sk_c (nor a ω_i). However, our forthcoming soundness property states that we can extract a consistent $(\omega_i, \mathsf{sk}_c, \mathsf{pk}_c)$ out from a successful prover. If the malicious redeemer makes no further issuance query, neither does the extractor in Theorem 4.4. So, we keep an honest prover in our OMUF definition and argue that it is enough, together with soundness, to capture OMUF with a malicious prover.

Unforgeability of requests. This notion says that if a query by a client to issue a token is accepted by the server with public key pk_c, it is guaranteed that the query was not forged, even if the adversary interacts with the client using sk_c. In this model, one client is honest and all other participants (other clients and the server) are malicious by default. This implies that the adversary can choose pk_s and sk_s and also change it at will.

Definition 2.5. An anonymous token with interactive redemption scheme is request-unforgeable if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{RUF}}(\mathcal{A}) = \Pr[\mathsf{RUF}(\mathcal{A}) \rightarrow 1] = \mathsf{negl}(\lambda)$$

where the $\mathsf{RUF}(\mathcal{A})$ game is defined in Figure 3.

In RUF, the client is honest and simulated by the oracles. ORequest makes the client request the token of index j. OFinal finalizes the issuance of the token of index j by giving the response from the (possibly malicious) server. ORedeem makes the client redeem the token of index j. The adversary tries to forge an issuance query which has not been made by ORequest.

ORedeem now runs sessions of the redeem protocol with the client. The session sid defines the inner state st[sid] for the client. The very first call with this session identifier defines the state of the session. By convention, the input defines the index j of the token to be redeemed. Interestingly, $\mathcal{A}$ can launch the redeem of the same tokens (of issuance session j) many times (with several redeem sessions sid), and possibly with different pk_s. For the acceptance of the forged query, the adversary must provide for which sk_s the query is accepted by honestly running the server.

2.3 Unlinkability Notion

This notion captures the fact that a malicious server should not be able to link the issuance and redemption phase of a given token. We assume an honest key generation in the setup of the server (line 2 in the game).[7] I.e., we assume that the server is set up with a key pair generated using Server.KeyGen.[8]

Our notion is equivalent to the one in previous work [22, 23], with adaptation to our NTAT notion (but for honest key setup on the server).

Definition 2.6 (Unlinkability). An AT scheme is unlinkable if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{UNLINK}_2}(\mathcal{A}) = \Pr[\mathsf{UNLINK}_2(\mathcal{A}) \rightarrow 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where the $\mathsf{UNLINK}_2(\mathcal{A})$ game is defined in Figure 4.

2.4 Non-Transferability Notion

We now define the non-transferability notion. In this definition, the goal is to capture the fact that a client who does not possess the client secret key on which the token was issued should not be able to successfully redeem the token. For that, we first define the notion of soundness for redeem which is similar to the knowledge soundness of interactive proofs of knowledge: if an adversary $\mathcal{A}_2$ is able to redeem a token σ after being given information from an

[7] We need this assumption in the proof of Theorem 4.3 (in Γ^4 on page 17).

[8] To allow malicious key generation, we would need an extra key registration process where the server would prove the knowledge of sk_s in an extractable manner. We choose the simpler approach with honest setup. Alternatively, we could improve the proof of Theorem 4.3 by adding extra assumptions such as AGM.

adversary $\mathcal{A}_1, \mathcal{A}_2$ must know some witness ω and some key pair $(\mathsf{sk}_c, \mathsf{pk}_c)$ associated to σ. The "must know" part of it is formalized by the notion of extractor.

Then, we define the uniqueness property of $(\omega, \mathsf{sk}_c, \mathsf{pk}_c)$ to formalize that σ is binding towards a unique $(\omega, \mathsf{sk}_c, \mathsf{pk}_c)$, thus to a unique $(\mathsf{sk}_c, \mathsf{pk}_c)$. Hence, transfer implies giving this pair.

Soundness. The game starts with an adversary $\mathcal{A}_1$ (who could possibly get the secret sk_s of the server, without loss of generality) who returns a state as well as a token σ and a session identifier sid^* to be used by the ORedeem oracle to redeem σ. Then, an adversary $\mathcal{A}_2$ fed with the state plays with the server through OIssueTok and ORedeem. $\mathcal{A}_2$ is supposed to successfully redeem σ (to be checked by $\mathsf{out}_{\mathsf{sid}^*} = 1$). If this is the case, an extractor $\mathcal{E}$ playing with $\mathcal{A}_2$ in a blackbox manner and with the state should extract the input $(\omega, \mathsf{sk}_c, \mathsf{pk}_c)$ to a successful honest prover to redeem σ, and satisfying $\mathsf{KeyVer}(\mathsf{pp}, \mathsf{sk}_c, \mathsf{pk}_c)$. Our idea is that if $\mathcal{A}_1$ with a registered key pk_c issues a token σ and transfers it to $\mathcal{A}_2$, for $\mathcal{A}_2$ to redeem it, $\mathcal{A}_1$ must provide sk_c as well. We further make sure that the pk_c of $\mathcal{A}_1$ and the extracted pk_c are the same by the next notion of uniqueness.

We adapt the notion of knowledge soundness of proofs of knowledge to our setting.

```
SOUND(A_1, A_2, E)
 1: Setup(1^λ) → pp
 2: Server.KeyGen(pp) → (sk_s, pk_s)
 3: (σ, state, sid*) ← A_1(pp, sk_s, pk_s)
 4: ORedeem(sid*, (σ, sk_s))                          ▷ initializes st[sid*]
 5: A_2^{OIssueTok,ORedeem}(state)                    ▷ indirect output in out_{sid*}
 6: if out_{sid*} undefined or out_{sid*} = 0 then return 0      ▷ redeem failed
 7: E^{A_2,OIssueTok,ORedeem}(pp, pk_s, σ, sid*, state) → (ω, sk_c, pk_c)
 8: run ⟨Client.Prove(pp, σ, ω, sk_c, pk_s), Server.Verify(pp, σ, sk_s)⟩ → out
 9: if out = 1 and KeyVer(pp, sk_c, pk_c) then
10:     return 1                                      ▷ extraction succeeded
11: return 2

OIssueTok(pk_c, query)
 1: resp ← Server.Issue(pp, sk_s, pk_c, query)
 2: return resp

ORedeem(sid, input)
 1: if st[sid] does not exist then
 2:     st[sid] ← (pp, input, sk_s)
 3:     return
 4: Server.Verify(st[sid], input) → (st', output)
 5: st[sid] ← st'
 6: if st' = ⊥ then out_sid ← output
 7: return output
```

Figure 5: Soundness game (Definition 2.7).

Definition 2.7 (Soundness). An AT scheme is K-sound if there exists an extractor $\mathcal{E}$ of bounded complexity with limit to oracle accesses (as detailed below) such that for any $\mathcal{A}_1$ and $\mathcal{A}_2$, the SOUND game outputs 2 with negligible probability.

$$\mathsf{Adv}^{\mathsf{SOUND}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{E}) = \Pr[\mathsf{SOUND}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{E}) \rightarrow 2] \leq \mathsf{negl}(\lambda)$$

where the SOUND game is defined in Figure 5. Given the random coins rnd which are used in the first three steps of the SOUND game (for setup, key generation, and $\mathcal{A}_1$), we define $p_{\mathsf{rnd}} = 1 - \Pr[\mathsf{SOUND}(\mathcal{A}_1, \mathcal{A}_2, .) \rightarrow 0 | \mathsf{rnd}]$ as the probability, over the coins by $\mathcal{A}_2$, that $\mathcal{A}_2$ succeeds to redeem the transferred token σ. The extractor $\mathcal{E}$ can run $\mathcal{A}_2$ in a black-box manner a number of times which is limited to $K(p_{\mathsf{rnd}}, \lambda)$ on average. The complexity of $\mathcal{E}$ is $K(p_{\mathsf{rnd}}, \lambda) \times \mathsf{Poly}(\lambda)$. The number of oracle calls by $\mathcal{E}$ is also limited to $K(p_{\mathsf{rnd}}, \lambda)$ times the bound on the number of oracle calls by $\mathcal{A}_2$.

```
UNIQ(A)
 1: Setup(1^λ) → pp
 2: A(pp) → (σ, ω, sk_c, pk_c, ω', sk'_c, pk'_c, sk_s, pk_s)
 3: run ⟨Client.Prove(pp, σ, ω, sk_c, pk_s), Server.Verify(pp, σ, sk_s)⟩ → out
 4: run ⟨Client.Prove(pp, σ, ω', sk'_c, pk_s), Server.Verify(pp, σ, sk_s)⟩ → out'
 5: if ¬KeyVer(pp, sk_c, pk_c) or out = 0 then return 0
 6: if ¬KeyVer(pp, sk'_c, pk'_c) or out' = 0 then return 0
 7: if (ω, sk_c, pk_c) = (ω', sk'_c, pk'_c) then return 0
 8: return 1
```

Figure 6: Uniqueness game (Definition 2.8).

Essentially, if we want to transfer a token to be redeemed with high probability p_{rnd}, we have an extraction with small complexity, where "high" and "small" are defined by the function K. Typically, we will have $K(1, \lambda) \approx 2$ so $\mathcal{E}$ succeeds to extract by running $\mathcal{A}_2$ twice on average. In our results, we use $K(x, \lambda) = \frac{2}{x(x - \varepsilon(\lambda))}$ for some negligible function $\varepsilon(\lambda)$. Hence, for p_{rnd} larger than a positive constant, we have $K = O(1)$.

When calling $\mathcal{A}_2$ in a black box manner (which is counted as one unit of complexity for $\mathcal{E}$), the extractor simulates the responses from the oracle queries made by $\mathcal{A}_2$. This implies that the execution of $\mathcal{A}_2$ is stalled upon an oracle query and the execution resumes when the extractor provides the answer to the query. By calling $\mathcal{A}_2$, $\mathcal{E}$ controls the random coins of $\mathcal{A}_2$ and can easily rewind $\mathcal{A}_2$ by replaying the same input and coins. Note that $\mathcal{E}$ must keep the same capabilities as $\mathcal{A}_2$: if $\mathcal{A}_2$ needs to make new tokens to succeed to redeem, $\mathcal{E}$ also needs to call OIssueTok to extract. If $\mathcal{A}_2$ works without this oracle, extraction does not need it either.

The non-transferability notion by Hanzlick and Slamanig [8] is much simpler and falsifiable. This comes from the fact that they assume that the device holding sk_c is trusted, so they do not have to deal with a malicious $\mathcal{A}_1$ misusing it. Essentially, their non-transferability notion is a form of unforgeability.

Uniqueness. Finally, we define the uniqueness notion. We say redeem is non-transferable if it is both sound and enforces uniqueness.

Definition 2.8 (Uniqueness). An AT scheme enforces uniqueness if for any PPT adversary $\mathcal{A}$ we have

$$\mathsf{Adv}^{\mathsf{UNIQ}}(\mathcal{A}) = \Pr[\mathsf{UNIQ}(\mathcal{A}) \rightarrow 1] = \mathsf{negl}(\lambda)$$

where the $\mathsf{UNIQ}(\mathcal{A})$ game is defined in Figure 6.

Redeem cannot be non-interactive. Non-transferability means soundness and uniqueness together. An intuitive observation is that, when redeem is non-interactive, soundness is compatible with neither accountability (more specifically with RUF security) nor unlinkability. Indeed, assuming soundness, there exists some $\mathcal{E}$ that we can use as follows when we have a scheme with non-interactive redeem. We set $\mathcal{A}_1$ to the honest client setting up their keys and issuing a token then set state to the non-interactive redeem message. Then, $\mathcal{A}_2$ is set to a dummy algorithm just sending state: essentially, $\mathcal{A}_2$ sets its output equal to its input. Due to correctness, p_{rnd} is always 1, meaning $K(p_{\mathsf{rnd}}, \lambda)$ is assumed to be small. Clearly,

$\mathcal{E}$ playing $K(p_{\mathsf{rnd}}, \lambda)$ times with $\mathcal{A}_2$ learns nothing in this interaction. Thus, it reduces to an extractor not using $\mathcal{A}_2$ at all and still being able to extract. Soundness implies that $\mathcal{E}$ can extract sk_c from state by playing with the issuance and redeem oracles. Assuming uniqueness is satisfied, this must be the correct sk_c. The crucial point is that the input state to $\mathcal{A}_2$ can be obtained when an honest client is redeeming a token, because of the non-interactive redeem. The ability to extract sk_c from redeem allows us to build adversaries to break the RUF and UNLINK security notions. This justifies our definition of an AT. In this argument, the non-interaction is essential.

3 OUR NON-TRANSFERABLE ANONYMOUS TOKENS PROTOCOL

In this section, we present our non-transferable anonymous tokens scheme following our interface of Theorem 2.1 and Theorem 2.3. We have the non-interactive proof π_c by the client, the non-interactive proof π_s by the server, the issuance protocol defined with three algorithms ClientQuery, ServerIssue, ClientFinal, and the redeem interactive protocol defined by Client.Prove and Server.Verify. We also provide in Appendix D variants of our protocol based on pairings (ensuring publicly verifiability) and trusted hardware (with explicit and implicit authentication). The correctness of our schemes is straightforward and omitted.

3.1 Our NTAT scheme

Our protocol uses random oracles which are denoted differently as H_i to enforce domain separation. In practice, we could have a single random oracle H and define $H_i(\mathsf{input}) = H(i, \mathsf{input})$.

Setup. Our protocol sets up pp and keys by the algorithms on Figure 7. We have $\mathsf{pp} = (q, \mathbb{G}, G_1, G_2, G_3, G_4)$ to define an additive group $\mathbb{G}$ of prime order q with random generators G_1, G_2, G_3, G_4. For accountability, we require $\mathbb{G}$ to be a generic group. In particular, for uniqueness, the discrete logarithm problem must be hard. The client and the server have key $\mathsf{sk}_c = x$, $\mathsf{pk}_c = xG_1$ (denoted $\mathsf{pk}_c = X$) and $\mathsf{sk}_s = y$, $\mathsf{pk}_s = yG_2$ (denoted $\mathsf{pk}_s = Y$) respectively.

```
Setup(1^λ)
  Generate $\mathbb{G}$ of order q
  Select random generators $G_1, \dots, G_4$ of $\mathbb{G}$
  $\mathsf{pp} \leftarrow (q, \mathbb{G}, G_1, \dots, G_4)$
  return pp

Client.KeyGen(pp)
  $\mathsf{pp} \leftarrow (q, \mathbb{G}, G_1, \dots, G_4)$
  $x \leftarrow\!\$\ \mathbb{Z}_q$
  $X \leftarrow xG_1$
  $(\mathsf{sk}_c, \mathsf{pk}_c) \leftarrow (x, X)$
  return $(\mathsf{sk}_c, \mathsf{pk}_c)$

KeyVer(pp, sk_c, pk_c)
  $\mathsf{pp} \leftarrow (q, \mathbb{G}, G_1, \dots, G_4)$
  $(x, X) \leftarrow (\mathsf{sk}_c, \mathsf{pk}_c)$
  return $1_{X = xG_1}$

Server.KeyGen(pp)
  $\mathsf{pp} \leftarrow (q, \mathbb{G}, G_1, \dots, G_4)$
  $y \leftarrow\!\$\ \mathbb{Z}_q$
  $Y \leftarrow yG_2$
  $(\mathsf{sk}_s, \mathsf{pk}_s) \leftarrow (y, Y)$
  return $(\mathsf{sk}_s, \mathsf{pk}_s)$
```

Figure 7: Set-up and key generation.

Issuance protocol.

The core of the protocol described in Figure 8 uses the algebraic MAC construction from Barki et al. [21]. The (σ, r, s) triplet, with $\sigma = \frac{1}{y+s}(xG_1 + rG_3 + G_4)$, is precisely $\mathsf{MAC}_{BB}(X)$ under secret key y with generators G_1, G_3, G_4. This can also be seen as a BBS+ signature on sk_c in a group which does not necessarily have a pairing (hence without public verifiability) [27, 28].

```
ClientQuery(pp, sk_c, pk_s)
  $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
  $x \leftarrow \mathsf{sk}_c; X \leftarrow xG_1$
  $r \leftarrow\!\$\ \mathbb{Z}_q$,
  $\delta \leftarrow\!\$\ \mathbb{Z}_q^*$
  $T \leftarrow \delta \cdot (X + rG_3 + G_4)$
  $\pi_c \leftarrow \Pi_{\mathrm{REP3}}.\mathsf{Prove}(\mathsf{pp}, X, T, x, \delta, r)$
  $\mathsf{query} \leftarrow (T, \pi_c)$
  $\mathsf{st} \leftarrow (\mathsf{pp}, \mathsf{pk}_s, r, \delta, T)$
  return (query, st)

ClientFinal(st, resp)
  $(\mathsf{pp}, \mathsf{pk}_s, r, \delta, T) \leftarrow \mathsf{st}$
  $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
  $Y \leftarrow \mathsf{pk}_s;\ (s, S, \pi_s) \leftarrow \mathsf{resp}$
  if not $\Pi_{\mathrm{DLEQ}}.\mathsf{Verify}(\mathsf{pp}, Y, S, T, s, \pi_s)$
    return ⊥
  $\sigma \leftarrow \frac{1}{\delta}.S$    ▷ $\sigma = \frac{1}{y+s}(xG_1 + rG_3 + G_4)$
  return $(\sigma, \omega = (r, s))$

ServerIssue(pp, sk_s, pk_c, query)
  $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
  $(T, \pi_c) \leftarrow \mathsf{query}$
  $y \leftarrow \mathsf{sk}_s;\ X \leftarrow \mathsf{pk}_c$
  if not $\Pi_{\mathrm{REP3}}.\mathsf{Verify}(\mathsf{pp}, X, T, \pi_c)$
    return ⊥
  $s \leftarrow\!\$\ \mathbb{Z}_q - \{-y\}$
  $S \leftarrow \frac{1}{y+s} T$
  $\pi_s \leftarrow \Pi_{\mathrm{DLEQ}}.\mathsf{Prove}(\mathsf{pp}, yG_2, S, T, s, y)$
    ▷ PoK of $y$ s.t. $Y = yG_2 \wedge yS = T - sS$
  $\mathsf{resp} \leftarrow (s, S, \pi_s)$
  return resp
```

Figure 8: Token issuance.

First, the client commits to their public key $X = xG_1$ via $C = X + rG_3 + G_4$. This commitment will be embedded in the token, so that later the client can prove knowledge of the corresponding secret key x to enable non-transferability.

The client constructs T by randomly masking (with a factor δ) the commit value $X + rG_3 + G_4$. This will be used by the server to construct resp. The random masking ensures unlinkability. The client provides a proof of knowledge attesting the fact that T was computed properly. This ensures that the resulting token is actually tied to the client's public key X. As discussed in Appendix E, this proof is not enough to obtain OMUF-security. The client also proves knowledge of x during this proof π_c. This gives RUF-security at the same time.

After receiving T and verifying π_c, the server (with secret key y) samples random s and computes S. These values are specifically constructed so that the client can compute $\sigma = \frac{1}{y+s}(xG_1 + rG_3 + G_4)$ using the values S, s, δ in the final part of the issuance protocol. The server also proves that S was constructed properly, which is required to ensure unlinkability.

Finally, the client receives s, S, π_s, verifies that S was computed properly and computes σ.

Client proof π_c. We present in Figure 9 how we use Π_{REP3} (see Appendix B) over the group $\mathbb{G}^2$ in our protocol to make a proof π_c issued by the client. We use it to prove the knowledge of x, δ, and r satisfying

$$x \begin{pmatrix} G_1 \\ G_1 \end{pmatrix} + r \begin{pmatrix} 0 \\ G_3 \end{pmatrix} - \frac{1}{\delta} \begin{pmatrix} 0 \\ T \end{pmatrix} = \begin{pmatrix} X \\ -G_4 \end{pmatrix}$$

The protocol uses a random oracle which is denoted by H_1. It is based on a generalized Schnorr proof [29] with Fiat-Shamir transform [30].

Server proof π_s. We present in Figure 10 how the Π_{DLEQ} (see Appendix B) proof will be used in our protocol with its notations to make a proof π_s issued by the server. We use it to prove the knowledge of y satisfying

$$y \begin{pmatrix} G_2 \\ S \end{pmatrix} = \begin{pmatrix} Y \\ T - sS \end{pmatrix}$$

```
Prover(pp, X, T, x, δ, r)
1: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
2: $(a, b, c) \leftarrow\$ \mathbb{Z}_\mathsf{q}^3$
3: $\mathsf{comm}_1 \leftarrow a.G_1$
4: $\mathsf{comm}_2 \leftarrow a.G_1 + bG_3 + cT$
5: $\mathsf{ch} \leftarrow H_1(\mathsf{pp}, X, T, \mathsf{comm}_1, \mathsf{comm}_2)$
6: $\mathsf{resp}_1 \leftarrow a - \mathsf{ch}.x \bmod \mathsf{q}$
7: $\mathsf{resp}_2 \leftarrow b - \mathsf{ch}.r \bmod \mathsf{q}$
8: $\mathsf{resp}_3 \leftarrow c + \mathsf{ch}.\frac{1}{\delta} \bmod \mathsf{q}$
9: return $(\mathsf{ch}, \mathsf{resp}_1, \mathsf{resp}_2, \mathsf{resp}_3)$
```

```
Verifier(pp, X, T, ch, resp_1, resp_2, resp_3)
1: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
2: $\mathsf{comm}_1' \leftarrow \mathsf{resp}_1.G_1 + \mathsf{ch}.X$
3: $\mathsf{comm}_2' \leftarrow \mathsf{resp}_1.G_1 + \mathsf{resp}_2.G_3 + \mathsf{resp}_3.T - \mathsf{ch}.G_4$
4: $\mathsf{ch}' \leftarrow H_1(\mathsf{pp}, X, T, \mathsf{comm}_1', \mathsf{comm}_2')$
5: return $(\mathsf{ch} == \mathsf{ch}')$
```

Figure 9: Π_{REP3}: Client proof π_c.

```
Prover(pp, Y, S, T, s, y)
1: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
2: $a \leftarrow\$ \mathbb{Z}_\mathsf{q}$
3: $\mathsf{comm}_1 \leftarrow a.G_2$
4: $\mathsf{comm}_2 \leftarrow a.S$
5: $\mathsf{ch} \leftarrow H_2(\mathsf{pp}, Y, S, T - sS, \mathsf{comm}_1, \mathsf{comm}_2)$
6: $\mathsf{resp} \leftarrow a + \mathsf{ch}.y \bmod \mathsf{q}$
7: return $(\mathsf{ch}, \mathsf{resp})$
```

```
Verifier(pp, Y, S, T, s, ch, resp)
1: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
2: $\mathsf{comm}_1' \leftarrow \mathsf{resp}.G_2 - \mathsf{ch}.Y$
3: $\mathsf{comm}_2' \leftarrow \mathsf{resp}.S - \mathsf{ch}.(T - sS)$
4: $\mathsf{ch}' \leftarrow H_2(\mathsf{pp}, Y, S, T - sS, \mathsf{comm}_1', \mathsf{comm}_2')$
5: return $(\mathsf{ch} == \mathsf{ch}')$
```

Figure 10: Π_{DLEQ}: Server proof π_s.

```
Client.Prove(pp, σ, ω, sk_c, pk_s)
1: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
2: $(r, s) \leftarrow \omega$; $x \leftarrow \mathsf{sk}_c$; $Y \leftarrow \mathsf{pk}_s$
3: $\sigma' \leftarrow xG_1 + rG_3 + G_4 - s\sigma$
4: $\alpha, \beta, \gamma \leftarrow\$ \mathbb{Z}_\mathsf{q}$
5: $Q \leftarrow \alpha G_1 + \beta G_3 + \gamma\sigma$
6: $\rho \leftarrow\$ \{0,1\}^{\ell_\rho}$
7: $\mathsf{comm} \leftarrow H_3(\rho, Q)$
8: Send $(\sigma', \mathsf{comm})$ to the server
9: Receive $c$
10: $(v_0, v_1, v_2) \leftarrow (\alpha + cx, \beta + cr, \gamma - cs)$
11: Send$(v_0, v_1, v_2, \rho)$ to the server
```

```
Server.Verify(pp, σ, sk_s)
1: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
2: $y \leftarrow \mathsf{sk}_s$
3: Receive $\sigma', \mathsf{comm}$ from the client
4: if $\sigma' \neq y\sigma$ then abort
5: $c \leftarrow\$ \mathbb{Z}_\mathsf{q}$
6: Send $c$ to the client
7: Receive $(v_0, v_1, v_2, \rho)$ from the client
8: $Q' \leftarrow v_0 G_1 + v_1 G_3 + v_2\sigma$
9: $Q^* \leftarrow Q' - c(\sigma' - G_4)$
10: $\mathsf{comm}^* \leftarrow H_3(\rho, Q^*)$
11: return $(\mathsf{comm} == \mathsf{comm}^*)$
```

Figure 11: Redemption.

The protocol uses a random oracle which is denoted by H_2. Like π_c, it is based on a generalized Schnorr proof [29] with Fiat-Shamir transform [30].

Redeem protocol. Suppose a token σ was issued to a client with public key $X = xG_1$. To ensure non-transferability, during redemption we require the client to prove knowledge of that secret key x, as well as of the randomness $\omega = (r, s)$ used within the token. To preserve unlinkability, the client cannot reveal their public key X. Thus to redeem their token, the client must engage in an interactive protocol $\Pi = \mathsf{ZKPoK}(x, s, r : \sigma = \frac{1}{y+s}(xG_1 + rG_3 + G_4))$ by revealing $\sigma' = y\sigma$. We construct in Figure 11 such a protocol Π, assuming the client and server are already both in possession of the token, i.e. we do not picture the first interaction which consists in the client sending their token σ to the server to be verified. With some rearranging, the client wants to convince the server that it knows (x, s, r) such that $\sigma' - G_4 = xG_1 + rG_3 - s\sigma$. Π is based on the generalized Schnorr protocol [29].

Discussion. In our protocol, we use a multiplicative mask δ on the Pedersen commitment and in the end we reveal part of the BBS signature, i.e. the group element σ. Thanks to the multiplicative masking, σ is unlinkable. The scalar s and the Pedersen nonce r remain private. For the redeem protocol, the client proves that they know the scalars and that it makes a full valid signature to the verifier who owns the signing key y.

We could have used additive masking instead as in the standard [31]. However, additive masking implies that we must keep σ secret as well as s as they are not modified by the additive mask. Moreover, we would need a token tag t as an extra attribute to be revealed to prevent double-spending. Finally, the client must prove the knowledge of a valid signature without even knowing how to verify it without pairing. It makes proofs more expensive and complex without pairing.

We give the detailed protocols in Appendix A.

3.2 Performance

We start by evaluating the complexity of our protocol and its pairing variant. We denote the client by C, by S the server, by $\times$ the scalar multiplication operation in $\mathbb{G}$ and by e the pairing operation. We focus on these two operations to estimate the complexity.[9] We do not count group additions, operations on integers, or hashing for simplicity.

The complexity comparison with the pairing variant (Appendix D.1) is a bit delicate because the group is different and there is a new pairing operation. Although the number of group operations is lower, their complexity is higher, as confirmed by experiment. The advantage may rather be in the functionality to have a redeem server who does not need the secret y.

We also compare the two versions of our NTAT first with non-transferable ACs with limited attribute and with other state-of-the-art protocols. Namely, we compare it to U-Prove (with device-bound version) [11] and CHAC (with one attribute) [8]. We also add to the comparisons Privacy Pass (PP) [1], the construction from Barki et al. which we denote by BBDT (from the initials of the authors) and BBDT's pairing variant (note that the communication is the same for the 2 variants hence we only detail the complexity and functionality), and ATHM [23] which is based on an algebraic MAC construction. We summarize the complexity comparison in Table 1. We also count the amount of communication, and compare it to the same protocols in Table 2.

We stress that those protocols do not have the same functionality, and we highlight some differences in Table 3. For instance, PP, ATHM and BBDT do not offer non-transferability. PP and ATHM are anonymous tokens with the notion of one-more unforgeability (OMUF). ATHM was made to offer private bit metadata while others do not. Other protocols are ACs (or KVACs) and one-more unforgeability is not really the point. (As a matter of fact, U-Prove is *not* OMUF-secure and it is not clear how to make it so, as discussed in Appendix F.) Issuance creates credentials which may be used multiple times. U-Prove may offer non-transferability in our sense, when using the device binding version. We discuss it in Appendix F. CHAC is a KVAC which splits the client into two parts, one being a trusted hardware. CHAC is done to minimize the work of the trusted hardware and to treat many attributes by aggregation. Their non-transferability notion relies on the trusted hardware

[9] If we want to regroup a sum of n scalar multiplications as an optimized n-multi-scalar multiplication, the figures for NTAT break down as follows: issuance takes $(2\times^3 + 1\times^2 + 4\times^1)$ for the client and $(1\times^4 + 1\times^2 + 3\times^1)$ for the server, and redeem takes $(1\times^3 + 1\times^1)$ for the client and $(1\times^3 + 2\times^1)$ for the server.

Table 1: Complexity comparison

	Issuance		Redemption		Total
	Client	Server	Client	Server	
Our NTAT scheme	11×	8×	4×	5×	28×
Our Pairing NTAT	7×, 2e	6×	4×	4×, 2e	21×, 4e
U-Prove [11]	9×	3×	3×	7×	22×
CHAC [8]	5×	5×, 5e	17×	12e	27×, 17e
CHAC* [8]	3×	5×, 5e	10×	12e	18×, 17e
BBDT [21]	3×, Π_{REP2}.Prove, Π_{DLEQ}.Verify	2×, Π_{REP2}.Verify, Π_{DLEQ}.Prove	11×	10×	17×, Π_{REP2}, Π_{DLEQ}
Pairing BBDT [21]	3×, Π_{REP2}.Prove, Π_{DLEQ}.Verify	2×, Π_{REP2}.Verify, Π_{DLEQ}.Prove	11×	9×, 2e	16×, 2e, Π_{REP2}, Π_{DLEQ}
PP [1]	2×	7×	0×	1×	10×
ATHM [23]	17×	11×	0×	1×	29×

assumption and is much weaker than ours. However, in our comparison we merged the two parts of the client and simplified the protocol for fair comparison. We discuss it in Appendix G. For a fair comparison with NTAT, we consider U-Prove with no attribute and device binding, and CHAC with one attribute. Regarding CHAC, we also considered a variant CHAC* with a merged client (which would not offer non-transferability).

We detail our criteria from Table 3 as follows:

- Protocols indicating one form of non-transferability are marked as such but it could mean different things. For instance, CHAC relies on a trusted hardware to have a weaker notion of non-transferability. U-Prove also claims device binding by using a trusted hardware and suggests redeem to be interactive, although it is not fully specified, not formalized, and not proven. NTAT has a formal notion based on soundness and proven security.
- Anonymous tokens are marked as offering OMUF security. Others are marked when we can just add an additional attribute (to play the role of a nonce) which is private at issuance and revealed at redeem.
- Having a “secretless redeem” (a redeem server without the secret of the issuing server) allows to separate the issuing server and the redeem server, which could be useful in some applications.
- Having a stateless issuer is a major advantage as it allows to implement issuers as a REST and use multiple backends servers with load balancing.
- Protocols which make redeem reveal a unique identifier of the token/credential are indicated as “single-show”. Protocols allowing to have a unique identifier as an extra attribute without breaking unlinkability are marked as “both”, except if they do not have multi-show unlinkability.

We implemented[10] our NTAT scheme, U-Prove and CHAC in Rust with Ristretto group using curve25519-dalek library (SIMD and BasePointTable optimizations disabled) and the Pairing NTAT scheme with Bls12-381 curve using the arkworks library [32]. We use SHA256 for H_1, H_2, H_3. The benchmarks have been obtained using the Criterion.rs statistical benchmarking suite [33] on a laptop with 2.3 GHz Intel Core i9-9880H. We report the benchmark results in Table 4.

[10]Source code is available at https://zenodo.org/records/11001946.

Table 2: Communication comparison. We denote by $\mathbb{G}$ the group elements, by q the scalars, by b the extra bits, and by ℓ_ρ and ℓ_H the length of ρ and H. Note that BBDT does not specify explicit proofs for the server and client during issuance hence we denote their generic cost by $\Pi_{\text{REP2}}, \Pi_{\text{DLEQ}}$.

	Issuance	Redemption	Total
Our NTAT scheme	$2\mathbb{G} + 7\text{q}$	$1\mathbb{G} + 4\text{q} + (\ell_\rho + \ell_H)b$	$3\mathbb{G} + 11\text{q} + (\ell_\rho + \ell_H)b$
Our Pairing NTAT	$2\mathbb{G} + 5\text{q}$	$1\mathbb{G} + 4\text{q} + (\ell_\rho + \ell_H)b$	$3\mathbb{G} + 9\text{q} + (\ell_\rho + \ell_H)b$
U-Prove [11]	$3\mathbb{G} + 2\text{q}$	$3\mathbb{G} + 5\text{q}$	$6\mathbb{G} + 7\text{q}$
CHAC [8]	$8\mathbb{G} + 1\text{q}$	$8\mathbb{G} + 1\text{q}$	$16\mathbb{G} + 2\text{q}$
BBDT[21]	$2\mathbb{G} + 2\text{q} + \Pi_{REP2} + \Pi_{\text{DLEQ}}$	$3\mathbb{G} + 8\text{q} + \ell_H b$	$5\mathbb{G} + 10\text{q} + \ell_H b + \Pi_{REP2} + \Pi_{\text{DLEQ}}$
PP [21]	$2\mathbb{G} + 1\text{q} + \ell_H b$	$1\mathbb{G} + \ell_H b$	$3\mathbb{G} + 1\text{q} + 2\ell_H b$
ATHM [23]	$4\mathbb{G} + 8\text{q}$	$2\mathbb{G} + 1\text{q}$	$6\mathbb{G} + 9\text{q}$

Table 3: Functionality and properties comparison

	Type	Non-transferable	OMUF	Secretless redeem	Stateless issuer	Single or Multi show
Our NTAT	AT	✓	✓	×	✓	Single
Our Pairing NTAT	AT	✓	✓	✓	✓	Single
U-Prove [11]	AC	✓	×	✓	×	Single
CHAC [8]	KVAC	✓	×	✓	×	Multi
CHAC* [8]	KVAC	×	×	✓	×	Multi
BBDT [21]	KVAC	×	✓	×	✓	Both
Pairing BBDT [21]	KVAC	×	✓	✓	✓	Both
PP[1]	AT	×	✓	×	✓	Single
ATHM[23]	AT	×	✓	×	✓	Single

Table 4: Benchmark Results. For U-Prove (Fig. 26, Fig. 27) and CHAC (Fig. 30) we merge the core and helper procedures as a single client.

	Issuance		Redemption		Total
	Client	Server	Client	Server	
NTAT scheme	651.94μs	517.31μs	191.24μs	190.80μs	1.55ms
Pairing NTAT	5.47ms	3.13ms	985.93μs	3.82ms	13.41ms
U-Prove	444.25μs	149.68μs	99.02μs	307.737μs	1.00ms
CHAC	1.28ms	10.74ms	4.02ms	19.80ms	35.84ms

4 SECURITY

4.1 OMUF Security

This security relies on the algebraic group model and random oracle model.

THEOREM 4.1 (OMUF). *The NTAT scheme described in Figures 7–11 is* OMUF*-secure in AGM and ROM. More precisely, for any adversary, we have*

$$\mathsf{Adv}^{\mathsf{OMUF}} \le n\frac{m_{H_2}}{\mathsf{q}} + \frac{n(n+5)}{2\mathsf{q}} + 2\mathsf{Adv}^{DLog} + \mathsf{Adv}^{n\text{-}DLog} + \frac{1}{\mathsf{q}}$$

where q *is the group order, n is the total number of* OIssueTok *queries,* Adv^{DLog} *(respectively* $\mathsf{Adv}^{n\text{-}DLog}$*) is the advantage of an adversary* *solving the DLog (respectively n-Dlog) problem with similar complexity, and m_{H_i} is the number of H_i queries in Random Oracle Model (ROM).*

Proof. We consider an adversary $\mathcal{A}$ playing the OMUF game in Figure 2.

We use the ROM in π_c: an oracle call to evaluate H_2 by using lazy sampling. There are random oracles H_1 and H_3, too. The total number of oracle calls to H_i by $\mathcal{A}$ is denoted by m_{H_i}.

Γ^1: We let $((\mathsf{sk}_c, \mathsf{pk}_c), (\sigma_i, \omega_i)_{i\in\{1,\ldots,q\}})$ be the output of $\mathcal{A}$, and we denote $\mathsf{sk}_c = x$, $\mathsf{sk}_s = y$, $\mathsf{pk}_c = X$, and $\omega_i = (r_i, s_i)$. We further denote $\sigma'_i = xG_1 + r_iG_3 + G_4 - s_i\sigma_i$. Our first step is to reduce OMUF to a game Γ^1 in which the steps Step 7 and following are replaced by the following winning conditions: $X = xG_1$ and $\sigma_i = \frac{1}{y+s_i}(xG_1 + r_iG_3 + G_4)$ for $i = 1, \ldots, q$. Clearly, Step 7 of the OMUF game aborts if and only if $X \neq xG_1$ so this condition does not modify the outcome of the game. The second condition is equivalent to $\sigma'_i = y\sigma_i$ which is precisely what is verified in the first message of the redeem protocol in Step 9 of the OMUF game. The rest of the redeem protocol is based on the representation of σ'_i and passes thanks to the perfect completeness of the protocol. Clearly, those changes do not modify the outcome of the game.

$$\mathsf{Adv}^{\mathsf{OMUF}} = \mathsf{Adv}^{\Gamma^1}$$

Γ^2: We define a DDH oracle specialized for the $Y = yG_2$ key: on input (A, B), the ODDH_y oracle answers whether $yA = B$.

```
ODDH_y(A, B)
1: return $1_{yA=B}$
```

Clearly, the ORedeem oracle can be simulated by an adversary who would have access to ODDH_y by simply calling $\mathsf{ODDH}_y(\sigma, \sigma')$ in the first operations by the server. Indeed, the other operations on the server side need no secret information to be done. Conversely, the ODDH_y oracle can be simulated by an adversary who would have access to ORedeem by just setting a dummy session sid with input A and first message (B, comm), with a dummy comm. The rest of the interaction would be dropped. Hence, access to ORedeem and ODDH_y are equivalent.

We reduce Γ^1 to another game Γ^2 where we replace the ORedeem oracle using an ODDH_y oracle and transform $\mathcal{A}$ accordingly. Note that the random oracle H_3 is now only queried by $\mathcal{A}$. So, we can change $\mathcal{A}$ to simulate H_3 by lazy sampling and make H_3 disappear from the game.

We have

$$\mathsf{Adv}^{\Gamma^1} = \mathsf{Adv}^{\Gamma^2}$$

Γ^3: In the next step, we replace in a game Γ^3 the OIssueTok oracle by another oracle which only returns s'_j and S_j (for the j^{th} query) but not π_s. This can be done by adding in $\mathcal{A}$ a simulator for the missing proof π_s. At the same time, we remove the random oracle calls to H_2 which appear in π_s (namely, $H_2(\mathsf{pp}, Y, S, T - sS, \mathsf{comm}'_1, \mathsf{comm}'_2)$) as only $\mathcal{A}$ is using it, so can simulate it.

To simulate π_s, the adversary picks ch and resp at random then computes $(\mathsf{comm}'_1, \mathsf{comm}'_2)$. Due to the distribution of ch and resp, this pair is uniform in the pairs proportional to (G_2, S). So, the probability that H_2 was called on this input before is bounded by $\frac{m_{H_2}}{\mathsf{q}}$. If it happens, the simulation fails. Otherwise, the adversary can program the query to H_2 in its own simulation for verification to return ch. By doing this over all the n issuance queries, we obtain

$$\mathsf{Adv}^{\Gamma^2} \leq \mathsf{Adv}^{\Gamma^3} + n\frac{m_{H_2}}{\mathsf{q}}$$

Γ^4: Next, we add in a game Γ^4 the extra winning condition that no collision on the output s'_i of the OIssueTok oracle occurs. The advantage loss is bounded by $\frac{n(n-1)}{2\mathsf{q}}$ where $n = \sum q[\mathsf{pk}_c]$ is the total number of OIssueTok queries. In the new game, we further replace the sampling method of s'_j by a uniform sampling in $\mathbb{Z}_\mathsf{q}$ and add the extra winning condition that $s'_j \neq -y$ for every j. The advantage loss is bounded by $\frac{n}{\mathsf{q}}$.

$$\mathsf{Adv}^{\Gamma^3} \leq \mathsf{Adv}^{\Gamma^4} + \frac{n(n+1)}{2\mathsf{q}}$$

Γ^5: In Γ^5, we first change $\mathcal{A}$ so that it does not query OIssueTok with an incorrect proof. This is possible as $\mathcal{A}$ can check the proof before submitting and simulate what happens when the proof is incorrect. Next, we make sure in the OIssueTok oracle that $H_1(\mathsf{pp}, X, T, \mathsf{comm}_1, \mathsf{comm}_2)$ has been queried before in the game. When this is not the case, $\mathsf{ch} = \mathsf{ch}'$ with probability $1/\mathsf{q}$. We fully write the Γ^5 game in Figure 12.

$$\mathsf{Adv}^{\Gamma^4} \leq \mathsf{Adv}^{\Gamma^5} + \frac{n}{\mathsf{q}}$$

```
$\Gamma^5(\mathcal{A})$
1: $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$
2: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
3: $\mathsf{Server.KeyGen}(\mathsf{pp}) \rightarrow (y, Y)$
4: $q[\cdot] \leftarrow 0, j \leftarrow 0$
5: $((x, X), (\sigma_i, r_i, s_i)_{i\in\{1,\ldots,q\}}) \leftarrow$
       $\mathcal{A}^{\mathsf{OIssueTok},\mathsf{ODDH}_y}(\mathsf{pp}, Y)$
6: if $q \leq q[X]$ then abort
7: if $\exists i \neq j \quad \sigma_i = \sigma_j$ then abort
8: if $\exists i \neq j \quad s'_i = s'_j$ then abort
9: if $X \neq xG_1$ then abort
10: if $\exists i \quad \sigma_i \neq \frac{1}{y+s_i}(xG_1 + r_iG_3 + G_4)$
11: then abort
12: return 1

$\mathsf{ODDH}_y(A, B)$
1: return $1_{yA=B}$

$\mathsf{OIssueTok}(X, T, \mathsf{ch}, \mathsf{resp}_1, \mathsf{resp}_2, \mathsf{resp}_3)$
13: increment $j$
14: $(X_j, T_j, \mathsf{ch}_j, \mathsf{resp}_{j1}, \mathsf{resp}_{j2}, \mathsf{resp}_{j3}) \leftarrow$
    $(X, T, \mathsf{ch}, \mathsf{resp}_1, \mathsf{resp}_2, \mathsf{resp}_3)$
15: $q[X_j] \leftarrow q[X_j] + 1$
16: $\mathsf{comm}_1 \leftarrow \mathsf{resp}_1.G_1 + \mathsf{ch}.X$
17: $\mathsf{comm}_2 \leftarrow \mathsf{resp}_1.G_1 + \mathsf{resp}_2.G_3 +$
    $\mathsf{resp}_3.T - \mathsf{ch}.G_4$
18: if $H_1(\mathsf{pp}, X, T, \mathsf{comm}_1, \mathsf{comm}_2)$
    not queried before
19: then abort
20: $\mathsf{ch}' \leftarrow$
    $H_1(\mathsf{pp}, X, T, \mathsf{comm}_1, \mathsf{comm}_2)$
21: if $\mathsf{ch} \neq \mathsf{ch}'$ then abort
22: $s'_j \leftarrow\$ \mathbb{Z}_q$
23: if $s'_j = -y$ then abort
24: $S_j \leftarrow \frac{1}{y+s'_j}T$
25: return $(s'_j, S_j)$
```

Figure 12: Γ^5 game for OMUF security.

AGM: In the algebraic group model, $\mathcal{A}$ must provide an expression of every group elements it provides (namely, the queries X_j and T_j to OIssueTok, the queries comm_1 and comm_e to H_1, and the final output X and σ_i) as a linear combination of the group elements it received (namely, $G_1, \ldots, G_4, Y$, and the S_j returned by OIssueTok). Hence, each group element Z which is produced by the adversary comes with a vector $\mathsf{vec}_Z = (a_1, \ldots, a_4, b, c_1, \ldots, c_n) \in \mathbb{Z}_\mathsf{q}^{n+5}$ s.t.

$$Z = a_1G_1 + \cdots + a_4G_4 + bY + c_1S_1 + \cdots + c_nS_n$$

We write base $= (G_1, \ldots, G_4, Y, S_1, \ldots, S_n)$ with a scalar product $Z = \langle \mathsf{vec}_Z, \mathsf{base}\rangle$.

In our protocol, given that $S_j = \frac{1}{y+s'_j}T_j$, the adversary can form a 4-dimensional vector $\mathsf{vec}'_Z \in \mathbb{Z}_\mathsf{q}(\bar{y})^4$ where coefficients are rational functions in terms of a variable $\bar{y}$, and s.t. Z is equal to $\langle \mathsf{vec}'_Z, \mathsf{base}'\rangle$ after evaluation to $\bar{y} \leftarrow y$, with $\mathsf{base}' = (G_1, \ldots, G_4)$. We define

$$\phi(a_1, \ldots, c_n) = (a_1, a_2 + b\bar{y}, a_3, a_4) + \frac{c_1}{\bar{y} + s'_1}\mathsf{vec}'_{T_1} + \cdots + \frac{c_n}{\bar{y} + s'_n}\mathsf{vec}'_{T_n}$$

by recursion with $\mathsf{vec}'_Z = \phi(\mathsf{vec}_Z)$.

The $-s'_j$ are poles of the rational functions which are randomly selected by $\mathsf{OIssueTok}$. Recall that they must be pairwise different in the winning cases. Let $Q(\bar{y}) = (\bar{y} + s'_1) \cdots (\bar{y} + s'_n)$. By induction on n, it follows that $Q(\bar{y}).\mathsf{vec}'(Z)$ always has polynomial coefficients, with degrees bounded by $n + 1$.

Given the way ϕ is constructed and thanks to the non-collision of the poles $-s'_j$, we can see that $\phi(v) = 0$ implies $v = 0$. So, ϕ is injective.

If the adversary finds a vector v such that $\langle v, \mathsf{base}\rangle = 0$, then either [case1] $v = 0$, or [case2] $v \neq 0$ (and thus $\phi(v) \neq 0$ thanks to ϕ being injective) and y is a root of $\langle \phi(v), \mathsf{base}'\rangle$. That is, y is a root of $\phi(v)$ (that the adversary can find by solving a polynomial equation) or the game finds a non-trivial relation over base'.

In the next reduction, we spot some $\langle v, \mathsf{base}\rangle = 0$ equations of interest. For each equation, we check in which case we are. If we are in [case1], we proceed in the game reduction. Otherwise, the adversary computes $P(\bar{y}) = Q(\bar{y})\phi(v)$. P is a vector of polynomials such that $\langle P(y), \mathsf{base}'\rangle = 0$ and $P \neq 0$. The adversary tries to solve $P(\bar{y}) = 0$ and checks if it gets y. If this is the case, the adversary continues like in [case1] to make forgeries. Otherwise, the adversary stops and yields P, and the game finds $\langle P(y), \mathsf{base}'\rangle = 0$ as a non-trivial relation over base'.

Γ^6**:** The verification of π_c consists in checking that
$$\mathsf{ch}_j = H_1(\mathsf{pp}, X_j, T_j, \mathsf{comm}_{j1}, \mathsf{comm}_{j2})$$
with $\mathsf{comm}_{j1} = \mathsf{resp}_{j1}G_1 + \mathsf{ch}_jX_j$ and $\mathsf{comm}_{j2} = \mathsf{resp}_{j1}G_1 + \mathsf{resp}_{j2}G_3 + \mathsf{resp}_{j3}T_j - \mathsf{ch}_jG_4$, which we write $\mathsf{comm}_{j2} - \mathsf{comm}_{j1} = \mathsf{resp}_{j2}G_3 + \mathsf{resp}_{j3}T_j - \mathsf{ch}_j(X_j + G_4)$. The first time this hash is computed, the value of ch is not known but the input vectors are already set. We are interested in $\mathsf{vec}_{\mathsf{comm}_{j1}}$ and $\mathsf{vec}_{\mathsf{comm}_{j2}}$. We obtain two equations of interest (for instance, the first equation is with $v = \mathsf{vec}_{\mathsf{comm}_{j1}} - \mathsf{resp}_{j1}\mathsf{vec}_{G_1} - \mathsf{ch}_j\mathsf{vec}_{X_j}$) and apply the reduction. Taking these equations of interest, when both are in [case1] we have
$$\begin{aligned}\mathsf{vec}_{\mathsf{comm}_{j1}} &= \mathsf{resp}_{j1}\mathsf{vec}_{G_1} + \mathsf{ch}_j\mathsf{vec}_{X_j}\\ \mathsf{vec}_{\mathsf{comm}_{j2}} - \mathsf{vec}_{\mathsf{comm}_{j1}} &= \mathsf{resp}_{j2}\mathsf{vec}_{G_3} + \mathsf{resp}_{j3}\mathsf{vec}_{T_j}\\ &\quad - \mathsf{ch}_j(\mathsf{vec}_{X_j} + \mathsf{vec}_{G_4})\end{aligned}$$
We wonder for how many ch_j values there exist some scalars resp_{j1}, resp_{j2}, resp_{j3} satisfying the two equations. Clearly, if either vec_{X_j} is not proportional to vec_{G_1} or vec_{T_j} is not in the linear span of vec_{G_3} and $\mathsf{vec}_{X_j} + \mathsf{vec}_{G_4}$, the number of possible ch_j is 0 or 1 and verification passes with probability bounded by $\frac{1}{\mathsf{q}}$.

We reduce to a game Γ^6 where for each $\mathsf{OIssueTok}$ query, either vec_{X_j} is of form $x_j\mathsf{vec}_{G_1}$ and vec_{T_j} is in the linear span of vec_{G_3} and $\mathsf{vec}_{X_j} + \mathsf{vec}_{G_4}$, or the oracle returns $\perp$ or $\mathcal{A}$ returns a polynomial P such that $\langle P(y), \mathsf{base}'\rangle = 0$ and $P \neq 0$ [case2]. We have
$$\mathsf{Adv}^{\Gamma^5} \leq \mathsf{Adv}^{\Gamma^6} + \frac{n}{\mathsf{q}}$$
Γ^7**:** In Γ^6, we can in [case1] extract x_j, δ_j, and r_j such that $X_j = x_jG_1$ and $T_j = \delta_j(X + r_jG_3 + G_4)$ by solving a linear system with vector representation. If this fails, $\mathcal{A}$ can compute the nonzero polynomial P. The adversary can find the roots of P and check if one of them is equal to y. If this is the case, $\mathcal{A}$ can use y to forge a token for a fresh X to return it. We obtain an adversary who wins in the forgery game Γ^7 (Figure 13) in those cases. We also define a companion adversary $\mathcal{A}'$ who does the same but aborts in the winning cases of Γ^7 and rather focuses on the leftover case where $P(y) \neq 0$. It is such that $\langle P(y), \mathsf{base}'\rangle = 0$. The game and $\mathcal{A}'$ boil down in a simple game which sets up pp, gives it to the adversary who finds a nonzero vector w such that $\langle w, \mathsf{base}'\rangle = 0$. This solves the discrete logarithm game.
$$\mathsf{Adv}^{\Gamma^6} \leq \mathsf{Adv}^{\Gamma^7} + \mathsf{Adv}^{\mathsf{DLog}}$$

```
$\Gamma^7(\mathcal{A})$
 1: $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$
 2: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
 3: $\mathsf{Server.KeyGen}(\mathsf{pp}) \to (y, Y)$
 4: $q[\cdot] \leftarrow 0, j \leftarrow 0$
 5: $((x, X), (\sigma_i, r_i, s_i)_{i\in\{1,\dots,q\}}) \leftarrow$
        $\mathcal{A}^{\mathsf{OIssueTok}}(\mathsf{pp}, Y)$
 6: **if** $q \leq q[X]$ **then abort**
 7: **if** $\exists i \neq j \quad \mathsf{vec}(\sigma_i) = \sigma_j$ **then abort**
 8: **if** $\exists i \neq j \quad s'_i = s'_j$ **then abort**
 9: **if** $X \neq xG_1$ **then abort**
10: **if** $\exists i \quad \sigma_i \neq \frac{1}{\bar{y}+s_i}(xG_1 + r_iG_3 + G_4)$
    **then abort**
11: **return** 1

$\mathsf{OIssueTok}(x, \delta, r)$
12: increment $j$
13: $(x_j, \delta_j, r'_j) \leftarrow (x, \delta, r)$
14: $q[x_j.G_1] \leftarrow q[x_j.G_1] + 1$
15: $s'_j \leftarrow\$ \mathbb{Z}_q$
16: **if** $s'_j = -y$ **then abort**
17: $S_j \leftarrow \frac{\delta_j}{y+s'_j}(xG_1 + r'_jG_3 + G_4)$
18: **return** $(s'_j, S_j)$

$\mathsf{ODDH}_y(A, B)$
19: **return** $1_{yA=B}$
```

Figure 13: Γ^7 **game for** OMUF **security.**

Γ^8**:** We now simulate $\mathsf{OIssueTok}(x, \delta, r)$ with a BBS signature oracle which upon a queried message (x, r) would answer with a random s and $\sigma = \frac{1}{y+s}(xG_1 + rG_3 + G_4)$. The simulation consists of multiplying σ by δ before returning the result. In the winning case, $\mathcal{A}$ forges more message/signature pairs with same x as it queries the signing oracle with that x. Hence, there must be one pair $((x, r), (s, \sigma))$ which does not match any sign query. This if a BBS forgery in the sense of SUF+ [28].

SUF+ security. We now use the SUF+ security of BBS [28]. To follow the terminology of the authors, we generate non-colliding s'_j in $\mathbb{Z}_\mathsf{q} - \{y\}$. The colliding probability is $\delta = 0$. Based on the BBS security [28, Th. 2], we obtain
$$\mathsf{Adv}^{\Gamma^8} \leq \mathsf{Adv}^{\mathsf{DLog}} + \mathsf{Adv}^{n\text{-}\mathsf{DLog}} + \frac{1}{\mathsf{q}}$$
Note that the cited BBS result holds in AGM with a pairing group. The pairing allows to simulate the ODDH_y oracle which becomes redundant in the game. In our context, we have no pairing but we keep the ODDH_y oracle. We easily check that the result holds in this context as well. □

4.2 RUF Security

This security is based on the discrete logarithm assumption and the random oracle model. We leave the proof to Appendix **??**.

Theorem 4.2 (RUF). *The NTAT scheme described in Figures 7–11 is* RUF*-secure in ROM, assuming that the discrete logarithm problem is hard. More precisely, for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ of complexity essentially twice the complexity of $\mathcal{A}$, and solving the discrete logarithm problem such that we have*
$$\mathsf{Adv}^{\mathsf{RUF}}(\mathcal{A}) \leq m_{\mathsf{ORedeem}}\frac{m_{H_3}}{\mathsf{q}2^{\ell_\rho}} + \frac{m_{H_1}(m_{H_1}+1)}{2\mathsf{q}} + \sqrt{2m_{H_3}.\mathsf{Adv}^{\mathsf{DLog}}(\mathcal{B})}$$
where q *is the group order, m_{ORedeem} is the total number of* ORedeem *queries,* Adv^{DLog} *is the advantage of an adversary solving the DLog problem with similar complexity, and m_{H_i} is the number of H_i queries in ROM.*

Proof. We consider an adversary $\mathcal{A}$ playing the RUF game.

Γ^1: We change the ORequest to return st_j together with (T, π_c) in query_j (i.e. to give the additional information of r and δ to $\mathcal{A}$) and we change $\mathcal{A}$ so that it receives it but doing the same computation as before (i.e. ignoring the extra information). In the new game, the adversary can compute $\sigma = \frac{S}{\delta}$, $\omega = (r, s)$, and $\sigma' = X + rG_3 + G_4 - s\sigma$.

$$\mathsf{Adv}^{\mathsf{RUF}} = \mathsf{Adv}^{\Gamma^1}$$

Γ^2: We change the adversary so that it simulates the ORedeem oracle using the zero-knowledge simulator and the programming technique of H_3 which was already used in the proof of OMUF-security. We reduce to a game Γ^2 with no ORedeem oracle. To do so, $\mathcal{A}$ would generate a dummy comm as the missing part of the output. Upon query with the challenge c, $\mathcal{A}$ would generate (v_0, v_1, v_2, ρ) at random, compute Q', Q^*, then program H_3 so that $\mathsf{comm} = H_3(\rho, Q^*)$. This fails with probability bounded by $\frac{m_{H_3}}{\mathsf{q}2^{\ell_\rho}}$, the probability that (ρ, Q^*) was ever queried to H_3, for each of the ORedeem sessions.

$$\mathsf{Adv}^{\Gamma^1} \leq \mathsf{Adv}^{\Gamma^2} + m_{\mathsf{ORedeem}} \frac{m_{H_3}}{\mathsf{q}2^{\ell_\rho}}$$

Furthermore, the adversary can now fully simulate OFinal and H_2. The oracles OFinal, ORedeem, H_2, and H_3 have now been removed.

Γ^3: In Γ^3, we have no oracle any more except H_1 and ORequest which returns $T = \delta(X + rG_3 + G_4)$ together with π_c, r, and δ. Clearly, this latter oracle can be perfectly simulated and we have only H_1 remaining.

$$\mathsf{Adv}^{\Gamma^2} = \mathsf{Adv}^{\Gamma^3}$$

We get an adversary which is given (pp, X) and which returns $\mathsf{query} = (T, \pi_c, y)$ with H_1 as only oracle. The adversary wins if $\Pi_{\mathsf{REP3}}.\mathsf{Verify}(\mathsf{pp}, X, T, \pi_c)$ accepts. Acceptance implies a verification that $H_1(\mathsf{pp}, X, T, \mathsf{comm}'_1, \mathsf{comm}'_2) = \mathsf{ch}$.

Γ^4: We reduce to a game in which we require the final H_1 query to the random oracle to have been made before by $\mathcal{A}$ as an additional winning condition. If this does not hold, the previous game wins with probability $\frac{1}{\mathsf{q}}$. We further require H_1 to show no collision during the game as an additional condition, which normally occurs with probability bounded by $\frac{m_{H_1}(m_{H_1}-1)}{2\mathsf{q}}$. Hence,

$$\mathsf{Adv}^{\Gamma^3} \leq \mathsf{Adv}^{\Gamma^4} + \frac{m_{H_1}(m_{H_1}+1)}{2\mathsf{q}}$$

Γ^5: Thanks to the new requirements, ch from π_c defines a unique $(\mathsf{comm}_1, \mathsf{comm}_2)$ in the table of H_1 queries which must match $(\mathsf{comm}'_1, \mathsf{comm}'_2)$ in the end. Hence, the protocol is equivalent to having $\mathcal{A}$ to return $(\mathsf{comm}_1, \mathsf{comm}_2)$ instead of ch in the final π_c and the verification to be that $(\mathsf{comm}'_1, \mathsf{comm}'_2) = (\mathsf{comm}_1, \mathsf{comm}_2)$ with $(\mathsf{comm}'_1, \mathsf{comm}'_2)$ computed from $\mathsf{ch} = H_1(\mathsf{pp}, X, T, \mathsf{comm}_1, \mathsf{comm}_2)$.

We obtain a regular unforgeability security for the Fiat-Shamir transform of a regular Σ-protocol. We can now use the forking lemma ([34], [35]) to obtain two successful executions with the same input (pp, X), same comm_1, different ch, and the corresponding different resp_1. This gives

$$\mathsf{resp}_1.G_1 + \mathsf{ch}.X = \mathsf{resp}^*_1.G_1 + \mathsf{ch}^*.X$$

which solves the discrete logarithm of X. This defines an adversary $\mathcal{B}$ solving the discrete logarithm problem and such that

$$\mathsf{Adv}^{\Gamma^5} \leq \sqrt{2m_{H_3}.\mathsf{Adv}^{\mathsf{DLog}}(\mathcal{B})}$$

□

4.3 UNLINK Security

This security is based on information theory. Only the number of oracle accesses is bounded. We leave the proof to Appendix ??.

Theorem 4.3 (Unlinkability). *The NTAT scheme described in Figures 7–11 is* UNLINK$_2$*-secure in ROM. More precisely, for any adversary, we have*

$$\mathsf{Adv}_2^{\mathsf{UNLINK}} \leq \frac{1}{2} + \frac{n_{q_r}(n_{H_1}+1) + n_{H_2} + m_{\mathsf{ORedeem}} n_{H_3} 2^{-\ell_\rho}}{\mathsf{q}}$$

where q *is the group order,* ℓ_ρ *is the bitlength of* ρ*,* n_{q_r} *is the total number of* ORequest *queries,* m_{ORedeem} *is the total number of* ORedeem *queries,* n_{H_i} *is the number of* H_i *queries in ROM.*

Proof. Recall that we assume honest key generation on the server side. We consider the UNLINK$_2$ game from Figure 4 with adversary $(\mathcal{A}_1, \mathcal{A}_2)$.

Γ^1: We remove the computation of π_c inside ORequest and make $\mathcal{A}$ simulate it, using the zero-knowledge of the proof π_c. More precisely, $\mathcal{A}$ pick ch and $(\mathsf{resp}_1, \mathsf{resp}_2, \mathsf{resp}_3)$ then deduce comm'_1 and comm'_2 and the input to H_1. Except with probability $\frac{n_{H_1}}{\mathsf{q}}$, this query is fresh and we can program H_1. Hence we get:

$$\mathsf{Adv}^{\mathsf{UNLINK}_2} \leq \mathsf{Adv}^{\Gamma^1} + \frac{n_{q_r} n_{H_1}}{\mathsf{q}}$$

where n_{q_r} is the number of request to ORequest. There is no H_1 oracle any more as it is fully simulated by $\mathcal{A}$.

Γ^2: In the next step, we simplify the ORedeem oracle which would only return σ_j and $\sigma'_j = x_jG_1 + r_jG_3 + G_4 - s_j\sigma$ and the rest would be simulated by the adversary. To do so, $\mathcal{A}$ would generate a dummy comm as the missing part of the output. Upon query with the challenge c, $\mathcal{A}$ would generate (v_0, v_1, v_2, ρ) at random, compute Q', Q^*, then program H_3 so that $\mathsf{comm} = H_3(\rho, Q^*)$. This fails with probability bounded by $\frac{n_{H_3}}{\mathsf{q}2^{\ell_\rho}}$, the probability that (ρ, Q^*) was ever queried to H_3, for each of the ORedeem sessions.

$$\mathsf{Adv}^{\Gamma^1} \leq \mathsf{Adv}^{\Gamma^2} + m_{\mathsf{ORedeem}} \frac{n_{H_3}}{\mathsf{q}2^{\ell_\rho}}$$

There is no H_3 oracle any more as it is fully simulated by $\mathcal{A}$.

Γ^3: We notice that $\sigma' = \frac{1}{\delta}(T - sS)$. We change ORedeem so that it would compute $\sigma_j = \frac{1}{\delta_j}S_j$ and $\sigma'_j = \frac{1}{\delta_j}(T_j - s_jS_j)$. Note that the swap in UNLINK would need to hold on the values of δ_j, T_j, S_j to be consistent.

$$\mathsf{Adv}^{\Gamma^2} = \mathsf{Adv}^{\Gamma^3}$$

Γ^4: In $\mathsf{OFinal}(j, \mathsf{resp})$, we parse $\mathsf{resp} = (s, S, \pi_s)$ and $\mathsf{query}_j = T$ (as there is no π_c any more). In Γ^4 we change OFinal to output out_j by directly checking whether $S = \frac{1}{y+s}T$.[11] The difference between the two games is bounded by the soundness of π_s, i.e. the probability that $\mathcal{A}$ can issue some (Y, S, T, s, π_s) such that verification passes but the value y such that $Y = yG_2$ does not satisfy $yS = T - sS$.

To bound this probability, we look at every query to H_2 by the game. Each query defines $(Y, S, T, T - sS, \mathsf{comm}_1, \mathsf{comm}_2)$ and the

[11]Here, we use y from sk_s which comes from the honest key generation of pk_s.

value of $\mathsf{ch} = H_2(\mathsf{pp}, Y, S, T - sS, \mathsf{comm}_1, \mathsf{comm}_2)$ is uniformly selected. If we let z be such that $T - sS = zS$, the chances that there exists resp such that $\pi_s = (\mathsf{ch}, \mathsf{resp})$ passes for (Y, S, T, s) is 1 if $y = z$ and $\frac{1}{\mathsf{q}}$ if $y \neq z$. Furthermore, for any (Y, S, T, s) which matches no query to H_2, the probability that the verification passes for any π_s is $\frac{1}{\mathsf{q}}$. Hence, the soundness advantage is bounded by $\frac{n_{H_2}+1}{\mathsf{q}}$. We deduce

$$\mathsf{Adv}^{\Gamma^3} \leq \mathsf{Adv}^{\Gamma^4} + \frac{n_{H_2+1}}{\mathsf{q}}$$

Γ^5**:** Thanks to the previous reduction, r_j is only used to compute T_j in ORequest. In Client.Query, instead of sampling (r, δ) uniformly, the new game Γ^5 samples (T, δ) uniformly but with $T \neq 0$. The distribution is unchanged, except in the $T = 0$ case which occurs with probability $\frac{1}{\mathsf{q}}$.

$$\mathsf{Adv}^{\Gamma^4} \leq \mathsf{Adv}^{\Gamma^5} + \frac{n_{q_r}}{\mathsf{q}}$$

Next, since δ is not used in ORequest any more, we postpone the random selection of δ to where it is used: in ORedeem.

```
Γ^5(𝒜)
1: $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$
2: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
3: $y \leftarrow\$ \mathbb{Z}_\mathsf{q}$
4: $Y \leftarrow yG_2$
5: $Q_{\mathsf{query}}, Q_{\mathsf{final}}, Q_{\mathsf{redeem}} \leftarrow \emptyset$
6: $i \leftarrow 0; b \leftarrow\$ \{0,1\}$
7: $(j_0, j_1, \mathsf{state}) \leftarrow \mathcal{A}_1^{\mathsf{oracles}}(\mathsf{pp}, y, Y)$
8: **if** $\{j_0, j_1\} \nsubseteq Q_{\mathsf{final}} - Q_{\mathsf{redeem}}$ **then abort**
9: **if** $\mathsf{out}_{j_0} = 0$ or $\mathsf{out}_{j_1} = 0$ **then abort**
10: **if** $b = 1$ **then**
11:     swap $(s_{j_0}, S_{j_0}, T_{j_0})$ and $(s_{j_1}, S_{j_1}, T_{j_1})$
12: $b' \leftarrow \mathcal{A}_2^{\mathsf{oracles}}(\mathsf{state})$
13: **return** $1_{b=b'}$

OClientKeyGen()
14: increment $i$
15: $x_i \leftarrow\$ \mathbb{Z}_\mathsf{q}$
16: $X_i \leftarrow x_i G_1$
17: **return** $(x_i, X_i)$

ORequest(j, usr)
18: **if** $j \in Q_{\mathsf{query}}$ **then abort**
19: insert $j$ in $Q_{\mathsf{query}}$
20: $(x_j, X_j) \leftarrow (\mathsf{sk}_{c,\mathsf{usr}}, \mathsf{pk}_{c,\mathsf{usr}})$
21: $T_j \leftarrow\$ \mathbb{G} - \{0\}$
22: **return** $T_j$

OFinal(j, s, S)
23: **if** $j \in Q_{\mathsf{final}}$ or $j \notin Q_{\mathsf{query}}$ **then abort**
24: insert $j$ in $Q_{\mathsf{final}}$
25: $(s_j, S_j) \leftarrow (s, S)$
26: $\mathsf{out}_j \leftarrow 1_{S=\frac{1}{y+s}T_j}$
27: **return** $\mathsf{out}_j$

ORedeem(j)
28: **if** $j \notin Q_{\mathsf{final}}$ **then abort**
29: insert $j$ in $Q_{\mathsf{redeem}}$
30: **if** $\mathsf{out}_j = \perp$ **then return**
31: $\delta_j \leftarrow\$ \mathbb{Z}_\mathsf{q}^*$
32: $\sigma_j \leftarrow \frac{1}{\delta_j} S_j$
33: $\sigma'_j \leftarrow \frac{1}{\delta_j}(T_j - s_j S_j)$
34: **return** $(\sigma_j, \sigma'_j)$
```

Figure 14: Game Γ^5 in unlinkability.

Γ^5 **unlinkability.** We show what is left in Γ^5 on Figure 14. When ORedeem goes on, we have $\mathsf{out}_j = 1$ so $S_j = \frac{1}{y+s_j}T_j$. This implies $T_j - s_j S_j = yS_j$ hence $\sigma'_j = y\sigma_j$. The random selection of δ_j makes σ_j independent from (s_j, S_j, T_j). Hence, it is clear that no information is revealed about b. Therefore, we get $\mathsf{Adv}^{\Gamma^5} = \frac{1}{2}$ □

4.4 SOUND Security

This security is based on information theory in ROM. However, ROM is only needed to limit the capability of the adversary in terms of access to H_3 and to compute the probability of a collision. We do not program a random oracle nor exploit access information to it.

Theorem 4.4 (Soundness). *The NTAT scheme described in Figures 7–11 is K-SOUND-secure in ROM with $K(x) = \frac{2}{x(x-1/\mathsf{q})}$ and*

$$\mathsf{Adv}^{\mathsf{SOUND}} \leq \frac{4m_{H_3}^2}{2^{\ell_H}}$$

where m_{H_3} is the maximal number of H_3 calls made by $\mathcal{A}_1$ and $\mathcal{A}_2$, and ℓ_H is the bitlength of H_3 outputs.

For instance, a transfer of probability $p_{\mathsf{rnd}} = 100\%$ would yield $K(1) \approx 2$ executions of $\mathcal{A}_2$ to extract the secret.

Proof. We want to show that for any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists an extractor $\mathcal{E}$ that outputs a valid witness.

The general idea is to build an extractor $\mathcal{E}$ like in the usual extractors in Σ protocols: $\mathcal{E}$ runs twice the redeem protocol, session sid^*, with same commitment until the answer from two different challenges is obtained, then extract a witness from two correct responses to the two challenges.

We consider the SOUND game with $\mathcal{A}_1$, $\mathcal{A}_2$, and our $\mathcal{E}$ which is defined below.

When $\mathcal{E}$ runs $\mathcal{A}_2$, $\mathcal{A}_2$ may call the ORedeem oracle which is stateful. Except calls of type $\mathsf{ORedeem}(\mathsf{sid}, \sigma', \mathsf{comm})$ following a $\mathsf{ORedeem}(\mathsf{sid}, \mathsf{pp}, \sigma)$, which answer whether $\sigma' = y\sigma$, these calls can be perfectly simulated by $\mathcal{E}$. To simulate the remaining $\mathsf{ORedeem}(\mathsf{sid}, \sigma', \mathsf{comm})$, we just make $\mathcal{E}$ query ORedeem with a dummy fresh sid' to extract the ODDH_y output. By changing sid to this dummy fresh sid', the oracle used during the extraction phase can be considered as stateless. This eliminates troubles to rewind $\mathcal{A}_2$.

The extractor $\mathcal{E}$ iterates a loop of independent rounds until the extraction works. At each round, $\mathcal{E}$ runs $\mathcal{A}_2$ twice with same coin flips until the $\mathsf{ORedeem}(\mathsf{sid}^*, \sigma', \mathsf{comm})$ oracle call is made by $\mathcal{A}_2$, at the point where Server.Verify selects the challenge c. Hence, $\mathcal{E}$ can easily run two executions of $\mathcal{A}_2$ which fork during the selection of c.

In the two executions, if session sid^* does not succeed in either execution, the round fails and $\mathcal{E}$ iterates to the next round. Otherwise, we let σ, σ', comm be the common values in the sessions sid^* of ORedeem during the two executions. The two executions give $(c, v_0, v_1, v_2, \rho, Q)$ and $(c', v'_0, v'_1, v'_2, \rho', Q')$ such that $H_3(\rho, Q) = H_3(\rho', Q') = \mathsf{comm}$ and

$$\begin{aligned} Q &= v_0G_1 + v_1G_3 + v_2\sigma - c(\sigma' - G_4) \\ Q' &= v'_0G_1 + v'_1G_3 + v'_2\sigma - c'(\sigma' - G_4) \end{aligned}$$

If $c = c'$, the round fails and $\mathcal{E}$ iterates. Otherwise, $\mathcal{E}$ ends the loop.

Next, if $Q \neq Q'$, $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{E})$ found a collision on H_3 during $2m_{H_3}$ calls and aborts. In this case, the SOUND game returns 2. This happens with complexity bounded by $\frac{4m_{H_3}^2}{2^{\ell_H}}$. Otherwise, we have $Q = Q'$ and $c \neq c'$. Hence, $\mathcal{E}$ can set $x = \frac{v'_0 - v_0}{c' - c}$, $r = \frac{v'_1 - v_1}{c' - c}$, and $s = -\frac{v'_2 - v_2}{c' - c}$ to get $\sigma' = xG_1 + rG_3 + G_4 - s\sigma$. The extractor further sets $X = xG_1$ so that (x, X) is a a possible output of Client.KeyGen. Finally, $\mathcal{E}$ outputs (r, s, x, X) which makes the redeem protocol return 1 so the SOUND game returns 1.

We let E be the event that $\mathsf{out}_{\mathsf{sid}^*} = 1$ (i.e. that the transfer of the token to $\mathcal{A}_2$ succeeds). We let rnd be the set of random coins of the game in the first three steps and rnd' be the set of random coins which are used by the entire game to run $\mathcal{A}_2(\mathsf{state})$ until just before the time that the challenge c is picked. In one round of $\mathcal{E}$, once rnd' is set, the two executions are independent. We let $\varepsilon(\mathsf{rnd}') = \Pr[E|\mathsf{rnd}']$ which would be the probability that one execution makes session sid^* succeed. We let E' be the event that the the round succeeds. Due to the independence of the two executions, we have

$$\Pr[E'|\mathsf{rnd}'] \geq \varepsilon(\mathsf{rnd}')\left(\varepsilon(\mathsf{rnd}') - \frac{1}{\mathsf{q}}\right)$$

The average of $\varepsilon(\mathsf{rnd}')$ over rnd' is $\Pr[E|\mathsf{rnd}] = p_{\mathsf{rnd}}$. We apply the Jensen inequality on $x \mapsto x\left(x - \frac{1}{\mathsf{q}}\right)$ to deduce $\Pr[E'] \geq p_{\mathsf{rnd}}\left(p_{\mathsf{rnd}} - \frac{1}{\mathsf{q}}\right)$. The average number of rounds is bounded by $1/\Pr[E']$. Since we have two executions of $\mathcal{A}_2$ per round, we obtain $K(x) = \frac{2}{x(x-1/\mathsf{q})}$. □

4.5 UNIQ Security

Theorem 4.5 (UNIQ). *The NTAT scheme described in Figures 7–11 is* UNIQ*-secure, assuming that the discrete logarithm problem is hard. More precisely, for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ of complexity essentially similar to $\mathcal{A}$, and solving the discrete logarithm problem such that we have*

$$\mathsf{Adv}^{\mathsf{UNIQ}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{DLog}}(\mathcal{B}) + \frac{5}{\mathsf{q}}$$

where q *is the group order and* Adv^{DLog} *is the advantage of an adversary solving the DLog problem with similar complexity.*

Proof. We consider an adversary $\mathcal{A}$ playing the UNIQ game. We define a discrete logarithm solver as follows.

```
$\mathcal{B}(\mathsf{q}, \mathbb{G}, G, Z)$
1: $\alpha_1, \beta_1, \ldots, \alpha_4, \beta_4 \leftarrow\$ \mathbb{Z}_\mathsf{q}^8$
2: $G_i \leftarrow \alpha_i G + \beta_i Z$ for $i = 1, \ldots, 4$
3: $\mathsf{pp} \leftarrow (\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4)$
4: if $\exists i \quad G_i = 0$ then abort
5: $\mathcal{A}(\mathsf{pp}) \to (\sigma, r, s, x, X, r', s', x', X', y, Y)$
6: $a \leftarrow (y+s')(x\alpha_1 + r\alpha_3 + \alpha_4) - (y+s)(x'\alpha_1 + r'\alpha_3 + \alpha_4)$
7: $b \leftarrow (y+s')(x\beta_1 + r\beta_3 + \beta_4) - (y+s)(x'\beta_1 + r'\beta_3 + \beta_4)$
8: if $b = 0$ then abort
9: return $-a/b$
```

The $G_i = 0$ failure cases occur with probability bounded by $\frac{4}{\mathsf{q}}$. Except in those cases, $\mathcal{B}$ provides to $\mathcal{A}$ some pp with correct distribution. Hence, $\mathcal{A}$ gets pp and outputs some values making UNIQ win with probability $\mathsf{Adv}^{\mathsf{UNIQ}}(\mathcal{A}) - \frac{4}{\mathsf{q}}$.

In the UNIQ winning cases, we have $X = xG_1$, $X' = x'G_1$, and $(r, s, x) \neq (r', s', x')$. Furthermore, the redeem protocol accept with the key pair (y, Y). This implies $y\sigma = xG_1 + rG_3 + G_4 - s\sigma$ and $y\sigma = x'G_1 + r'G_3 + G_4 - s'\sigma$ which leads us to $aG + bZ = 0$.

Let z be such that $Z = zG$ and $\gamma_i = \alpha_i + z\beta_i$ be such that $G_i = \gamma_i G$ for $i = 1, \ldots, 4$. The adversary $\mathcal{A}$ has information on the γ_i but β_i is independent from γ_i. We have $b = ((y+s')x - (y+s)x')\beta_1 + ((y+s')r - (y+s)r')\beta_3 + (s'-s)\beta_4$ Assuming that $s \neq s'$ or $x \neq x'$ or $r \neq r'$, the distribution of b conditioned to pp being fixed is uniform, so $\Pr[b = 0|\mathsf{pp}] = \frac{1}{\mathsf{q}}$. Hence, $\mathcal{B}$ succeeds with probability at least $\mathsf{Adv}^{\mathsf{UNIQ}}(\mathcal{A}) - \frac{5}{\mathsf{q}}$. □

5 CONCLUSION

In conclusion, we formalised the key security notion of *non-transferability* for anonymous tokens. We provided a concrete construction with proven security as well as an efficient implementation. We also discussed several variants to extend the basic functionality to include public verifiability, more attributes or binding to an external valuable secret.

Our treatment of the non-transferability notion is the first formal one in the literature with complete security definitions and their respective proofs. Our main construction is resistant to token stealing attacks. Moreover, we provide a variant which also prevents identity leasing.

Acknowledgements. We are greatly thankful to our anonymous reviewers for suggesting important improvements to the paper. We also thank Greg Zaverucha and Christian Paquin for their discussions.

REFERENCES

[1] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proceedings on Privacy Enhancing Technologies*, pages 164–180, 2018.
[2] Michael Z. Lee, Alan M. Dunn, Jonathan Katz, Brent Waters, and Emmett Witchel. Anon-Pass: Practical Anonymous Subscriptions. *IEEE Security & Privacy*, 12:20–27, 2014.
[3] Sharon Huang, Subodh Iyengar, Sundar Jeyaraman, Shiv Kushwah, Chen-Kuei Lee Zutian Luo, Payman Mohassel, Ananth Raghunathan, Shaahid Shaikh, Yen-Chieh, and Sung Albert Zhang. DIT: De-Identified Authenticated Telemetry at Scale, 2021. https://research.fb.com/privatestats.
[4] Tjerand Silde and Martin Strand. Anonymous Tokens with Public Metadata and Applications to Private Contact Tracing. Cryptology ePrint Archive, Report 2021/203, 2021. https://ia.cr/2021/203.
[5] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using Hard AI Problems for Security. In *Advances in Cryptology — EUROCRYPT 2003*, pages 294–311. Springer Berlin Heidelberg, 2003.
[6] Jan Camenisch and Anna Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *Advances in Cryptology — EUROCRYPT 2001*, pages 93–118, 2001.
[7] Sebastian Pape. A Survey on Non-transferable Anonymous Credentials. In *The Future of Identity in the Information Society*, pages 107–118. Springer Berlin Heidelberg, 2009.
[8] Lucjan Hanzlik and Daniel Slamanig. With a Little Help from My Friends: Constructing Practical Anonymous Credentials. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, page 2004–2023. Association for Computing Machinery, 2021.
[9] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The Algebraic Group Model and its Applications. In *Advances in Cryptology – CRYPTO 2018*, pages 33–62. Springer International Publishing, 2018.
[10] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. Cryptology ePrint Archive, Report 2014/650, 2014. https://ia.cr/2014/650.
[11] Christian Paquin and Greg Zaverucha. U-Prove Cryptographic Specification V1.1 Revision 4. Microsoft Technical Report, 2022. https://github.com/microsoft/uprove-node-reference/raw/main/doc/U-Prove%20Cryptographic%20Specification%20V1.1%20Revision%204.pdf.
[12] David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, pages 199–203. Plenum, 1982.
[13] David Chaum. Security without Identification: Transaction Systems to Make Big Brother Obsolete. *COmmunications of ACM*, page 1030–1044, 1985.
[14] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous Credentials Light. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &amp; Communications Security*, page 1087–1098. Association for Computing Machinery, 2013.
[15] David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 288–304, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
[16] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1084–1101, 2019.
[17] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In *Advances in Cryptology: Proceedings of EUROCRYPT*, 2021.
[18] Julia Kastner, Julian Loss, and Omar Renawi. Concurrent security of anonymous credentials light, revisited. Cryptology ePrint Archive, Paper 2023/707, 2023. https://eprint.iacr.org/2023/707.
[19] Stefan Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology — CRYPTO' 93*, pages 302–318. Springer Berlin Heidelberg, 1994.
[20] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and Keyed-Verification Anonymous Credentials. In *ACM CCS 2014*, 2014.
[21] Amira Barki, Solenn Brunet, Nicolas Desmoulins, and Jacques Traoré. Improved Algebraic MACs and Practical Keyed-Verification Anonymous Credentials. In *Selected Areas in Cryptography – SAC 2016*, pages 360–380, 2017.
[22] Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous Tokens with Private Metadata Bit. In *Advances in Cryptology – CRYPTO 2020*, pages 308–336, 2020.

[23] Melissa Chase, F. Betül Durak, and Serge Vaudenay. Anonymous Tokens with Stronger Metadata Bit Hiding from Algebraic MACs. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part II*, page 418–449, Berlin, Heidelberg, 2023. Springer-Verlag.

[24] Melissa Chase, F. Betül Durak, and Serge Vaudenay. Anonymous tokens with hidden metadata bit from algebraic macs. Cryptology ePrint Archive, Paper 2022/1622, 2022. https://eprint.iacr.org/2022/1622.

[25] M. Naor and O. Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 458–467, 1997.

[26] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message Authentication, Revisited. In *Advances in Cryptology – EUROCRYPT 2012*, pages 355–374, 2012.

[27] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks*, pages 111–125, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[28] Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 691–721, Cham, 2023. Springer Nature Switzerland.

[29] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, 1990.

[30] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology — CRYPTO' 86*, volume 263, 03 1999.

[31] M. Lodder, T. Looker, and A. Whitehead. Blind Signatures Extension of the BBS Signature Scheme, 2023. https://identity.foundation/bbs-signature/draft-blind-bbs-signatures.html.

[32] arkworks contributors. `arkworks` zksnark ecosystem, 2022.

[33] `Criterion.rs` statistics-driven microbenchmarking in rust, 2023.

[34] David Pointcheval and Jacques Stern. Security Proofs for Signature Schemes. In *Advances in Cryptology — EUROCRYPT '96*, pages 387–398, 1996.

[35] David Pointcheval and Jacques Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 2001.

[36] Jan Camenisch and Markus Stadler. Proof Systems for General Statements about Discrete Logarithms. Technical report, ETH Zurich, Department of Computer Science, 1997.

[37] Paulo Mateus and Serge Vaudenay. On Tamper-Resistance from a Theoretical Viewpoint. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, Proceedings*, pages 411–428, 2009.

[38] Ivan Puddu, Daniele Lain, Moritz Schneider, Elizaveta Tretiakova, Sinisa Matetic, and Srdjan Capkun. TEEvil: Identity Lease via Trusted Execution Environments. arxiv, 2019. https://arxiv.org/abs/1903.00449.

[39] Stefan Dziembowski, Sebastian Faust, and Tomasz Lizurej. Individual Cryptography. Cryptology ePrint Archive, Report 2023/088, 2023. https://eprint.iacr.org/2023/088.pdf.

[40] Y Desmedt. Major Security Problems with the "Unforgeable" (Feige-)Fiat-Shamir Proofs of Identity and How to Overcome Them. In *Congress on Computer and Communication Security and Protection Securicom'88*, pages 147–159. SEDEP Paris France, 1988.

[41] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Universally Composable Multiparty Computation with Partially Isolated Parties. In *Theory of Cryptography*, pages 315–331, 2009.

[42] Ioana Boureanu and Serge Vaudenay. Input-Aware Equivocable Commitments and UC-secure Commitments with Atomic Exchanges. In *Provable Security*, pages 121–138, 2013.

[43] Nirvan Tyagi, Sofía Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A. Wood. A Fast and Simple Partially Oblivious PRF, with Applications. In *Advances in Cryptology — EUROCRYPT 2021*. Springer-Verlag, 2021.

[44] Craig Gentry, Zulfikar Ramzan, and Stuart Stubblebine. Secure Distributed Human Computation. In *Financial Cryptography and Data Security*, pages 328–332, 2005.

[45] Serge Vaudenay. E-Passport Threats. *IEEE Security and Privacy*, page 61–64, 2007.

[46] Serge Vaudenay and Martin Vuagnoux. About Machine-Readable Travel Documents. *Journal of Physics: Conference Series*, 2007.

[47] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Universal Designated Verifier Signature Proof (or How to Efficiently Prove Knowledge of a Signature). In *Advances in Cryptology - ASIACRYPT 2005*, pages 644–661, 2005.

[48] Jean Monnerat, Serge Vaudenay, and Martin Vuagnoux. About Machine-Readable Travel Documents Privacy Enhancement Using (Weakly) Non-Transferable Data Authentication. In *International Conference on RFID Security*, pages 13–26, 2007.

[49] Jean Monnerat, Sylvain Pasini, and Serge Vaudenay. Efficient Deniable Authentication for Signatures. In *Applied Cryptography and Network Security*, pages 172–291, 2009.

[50] Fatih Balli, F. Betül Durak, and Serge Vaudenay. BioID: A Privacy-Friendly Identity Document. In *Security and Trust Management*, pages 53–70, 2019.

[51] Rafael Pass. On Deniability in the Common Reference String and Random Oracle Model. In *Advances in Cryptology - CRYPTO 2003*, pages 316–337, 2003.

[52] Paulo Mateus and Serge Vaudenay. On Tamper-Resistance from a Theoretical Viewpoint. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 411–428, 2009.

[53] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to Leak a Secret. In *Advances in Cryptology — ASIACRYPT 2001*, pages 552–565, 2001.

[54] Thomas Ristenpart and Scott Yilek. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In *Advances in Cryptology - EUROCRYPT 2007*, pages 228–245, 2007.

[55] David Chaum and Hans van Antwerpen. Undeniable Signatures. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 212–216, 1990.

[56] Yvo Desmedt and Moti Yung. Weaknesses of Undeniable Signature Schemes. In *Advances in Cryptology — EUROCRYPT '91*, pages 205–220, 1991.

[57] Markus Jakobsson. Blackmailing using Undeniable Signatures. In *Advances in Cryptology — EUROCRYPT'94*, pages 425–427, 1995.

[58] Jan Camenisch and Markus Michels. Confirmer Signature Schemes Secure against Adaptive Adversaries. In *Advances in Cryptology — EUROCRYPT 2000*, pages 243–258, 2000.

[59] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated Verifier Proofs and Their Applications. In *Advances in Cryptology — EUROCRYPT '96*, pages 143–154, 1996.

[60] Popov A, M.Nystroem, D.Balfanz, and J.Hodges. The Token Binding Protocol Version 1.0. IETF RFC 8471, 2018. https://www.rfc-editor.org/info/rfc8471.

[61] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 174–187, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

```
ClientQuery(pp, sk_c, pk_s)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
2: $x \leftarrow \mathsf{sk}_c; X \leftarrow xG_1$
3: $r' \leftarrow\$ \mathbb{Z}_q; t \leftarrow\$ \mathbb{Z}_q^*$
4: $T \leftarrow X + r'G_3 + G_4 + tG_5$
5: $\pi_c \leftarrow \Pi_{\mathrm{REP3}}.\mathsf{Prove}(\mathsf{pp}, X, T, x, r', t)$
6: query $\leftarrow (T, \pi_c)$
7: st $\leftarrow (\mathsf{pp}, \mathsf{pk}_s, r', t, T)$
8: return (query, st)

ClientFinal(st, resp)
1: $(\mathsf{pp}, \mathsf{pk}_s, r', t, T) \leftarrow$ st
2: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
3: $Y \leftarrow \mathsf{pk}_s$
4: $(s, r'', S, \pi_s) \leftarrow$ resp
5: if not $\Pi_{\mathrm{DLEQ}}.\mathsf{Verify}(\mathsf{pp}, Y, S, T, s, r'', \pi_s)$
   return $\perp$
6: $\sigma \leftarrow S$
7: $r \leftarrow r' + r''$   $\triangleright$ $\sigma = \frac{1}{y+s}(xG_1 + rG_3 + G_4 + tG_5)$
8: $\sigma' \leftarrow T + r''G_3 - s\sigma$ $\triangleright$ $\sigma' = y\sigma$
9: return $(t, \omega = (\sigma, \sigma', r, s))$

ServerIssue(pp, sk_s, pk_c, query)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
2: $(T, \pi_c) \leftarrow$ query
3: $y \leftarrow \mathsf{sk}_s;\ X \leftarrow \mathsf{pk}_c$
4: if not $\Pi_{\mathrm{REP3}}.\mathsf{Verify}(\mathsf{pp}, X, T, \pi_c)$
   return $\perp$
5: $r'' \leftarrow\$ \mathbb{Z}_q$
6: $s \leftarrow\$ \mathbb{Z}_q - \{-y\}$
7: $S \leftarrow \frac{1}{y+s}(T + r''G_3)$
8: $\pi_s \leftarrow \Pi_{\mathrm{DLEQ}}.\mathsf{Prove}(\mathsf{pp}, yG_2, S, T, s, r'', y)$
   $\triangleright$ PoK of $y$ s.t. $Y = yG_2 \wedge yS = T + r''G_3 - sS$
9: resp $\leftarrow (s, r'', S, \pi_s)$
10: return resp
```

Figure 15: Token issuance with additive masking.

ADDITIONAL MATERIAL

A NON-TRANSFERABILITY WITH ADDITIVE MASKING

In this section, we briefly define a non-transferable anonymous token from BBS signatures with additive masking (instead of multiplicative mask on a Pedersen commitment). The advantage is that it is closer to ongoing standards. The drawback is the complexity overhead.

Setup. Setup provides an additional group generator G_5 to be used for the new attribute t.

Issuance protocol. The client selects the token tag t, a random r', and sets $T = x.G_1 + r'.G_3 + G_4 + t.G_5$. Note that $T - G_4$ is a Pedersen commitment for (x, t). The client proves the knowledge of an opening (x, r', t) (proof π_c). The server adds another masking $r''.G_3$ and signs. It reveals (s, r'', σ) and a proof of correctness for σ (proof π_s). Finally, the client keeps the tag t and the witness $\omega = (\sigma, r, s)$ with $r = r' + r''$. The protocol is on Figure 15. We included the precomputation of $\sigma' = T + r''.G_3 - s.\sigma = x.G_1 + r.G_3 + G_4 + t.G_5 - s\sigma = y.\sigma$ which plays a role in redemption. It comes for almost free at this stage as the client already computed $r''G_3$ and $s\sigma$ to verify π_s.

Client proof π_c. π_c proves knowledge of (x, r', t) such that

$$x\begin{pmatrix} G_1 \\ G_1 \end{pmatrix} + r'\begin{pmatrix} 0 \\ G_3 \end{pmatrix} + t\begin{pmatrix} 0 \\ G_5 \end{pmatrix} = \begin{pmatrix} X \\ T - G_4 \end{pmatrix}$$

The protocol is in Figure 16.

Server proof π_s. π_s proves knowledge of y such that

$$y\begin{pmatrix} G_2 \\ S \end{pmatrix} = \begin{pmatrix} Y \\ T - s.S \end{pmatrix}$$

The protocol is in Figure 17.

```
Prover(pp, X, T, x, r', t)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
2: $(a, b, c) \leftarrow\$ \mathbb{Z}_q^3$
3: $\mathsf{comm}_1 \leftarrow a.G_1$
4: $\mathsf{comm}_2 \leftarrow a.G_1 + bG_3 + cG_5$
5: $\mathsf{ch} \leftarrow H_1(\mathsf{pp}, X, T, \mathsf{comm}_1, \mathsf{comm}_2)$
6: $\mathsf{resp}_1 \leftarrow a - \mathsf{ch}.x \bmod q$
7: $\mathsf{resp}_2 \leftarrow b - \mathsf{ch}.r' \bmod q$
8: $\mathsf{resp}_3 \leftarrow c - \mathsf{ch}.t \bmod q$
9: return resp $= (\mathsf{ch}, \mathsf{resp}_1, \mathsf{resp}_2, \mathsf{resp}_3)$

Verifier(pp, X, T, ch, resp_1, resp_2, resp_3)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
2: $\mathsf{comm}'_1 \leftarrow \mathsf{resp}_1.G_1 + \mathsf{ch}.X$
3: $\mathsf{comm}'_2 \leftarrow \mathsf{resp}_1.G_1 + \mathsf{resp}_2.G_3 + \mathsf{resp}_3.G_5 + \mathsf{ch}.(T - G_4)$
4: $\mathsf{ch}' \leftarrow H_1(\mathsf{pp}, X, T, \mathsf{comm}'_1, \mathsf{comm}'_2)$
5: return $(\mathsf{ch} == \mathsf{ch}')$
```

Figure 16: Π_{REP3}: Client proof π_c with additive masking.

```
Prover(pp, Y, S, T, s, r'', y)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
2: $a \leftarrow\$ \mathbb{Z}_q$
3: $\mathsf{comm}_1 \leftarrow a.G_2$
4: $\mathsf{comm}_2 \leftarrow a.S$
5: $\mathsf{ch} \leftarrow H_2(\mathsf{pp}, Y, S, T - sS, \mathsf{comm}_1, \mathsf{comm}_2)$
6: $\mathsf{resp} \leftarrow a + \mathsf{ch}.y \bmod q$
7: return (ch, resp)

Verifier(pp, Y, S, T, s, r'', ch, resp)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
2: $\mathsf{comm}'_1 \leftarrow \mathsf{resp}.G_2 - \mathsf{ch}.Y$
3: $\mathsf{comm}'_2 \leftarrow \mathsf{resp}.S - \mathsf{ch}.(T + r''G_3 - sS)$
4: $\mathsf{ch}' \leftarrow H_2(\mathsf{pp}, Y, S, T + r''G_3 - sS, \mathsf{comm}'_1, \mathsf{comm}'_2)$
5: return $(\mathsf{ch} == \mathsf{ch}')$
```

Figure 17: Π_{DLEQ}: Server proof π_s with additive masking.

```
Client.Prove(pp, t, ω, sk_c, pk_s)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
2: $(\sigma, \sigma', r, s) \leftarrow \omega;\ x \leftarrow \mathsf{sk}_c;\ Y \leftarrow \mathsf{pk}_s$
3: $\alpha, \beta, \gamma, \delta, \mu \leftarrow\$ \mathbb{Z}_q$
4: $A \leftarrow \frac{1}{\mu}\sigma, B \leftarrow \frac{1}{\mu}\sigma'$
5: $Q \leftarrow \alpha.G_1 + \beta.G_3 + \gamma.A + \delta.B$
6: $\rho \leftarrow\$ \{0, 1\}^{\ell_\rho}$
7: $\mathsf{comm} \leftarrow H_3(\rho, Q)$
8: Send $(A, B, \mathsf{comm})$ to the server
9: Receive $c$
10: $(v_0, v_1, v_2, v_3) \leftarrow (\alpha + cx, \beta + cr, \gamma - cs\mu, \delta - c\mu)$
11: Send$(v_0, \ldots, v_3, \rho)$ to the server

Server.Verify(pp, t, sk_s)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, G_5) \leftarrow$ pp
2: $y \leftarrow \mathsf{sk}_s$
3: Receive $A, B$, comm from the client
4: if $B \neq y.A$ then abort
5: $c \leftarrow\$ \mathbb{Z}_q$
6: Send $c$ to the client
7: Receive $(v_0, v_1, v_2, v_3, \rho)$ from the client
8: $Q' \leftarrow v_0.G_1 + v_1.G_3 + v_2.A + v_3.B + c.(G_4 + t.G_5)$
9: $\mathsf{comm}' \leftarrow H_3(\rho, Q')$
10: return $(\mathsf{comm} == \mathsf{comm}')$
```

Figure 18: Redemption with additive masking.

Redemption. To redeem a token, the client masks (σ, σ') with a random μ, reveals the tag t, $A = \frac{1}{\mu}\sigma$, $B = \frac{1}{\mu}\sigma'$, and proves knowledge of a valid tuple (x, r, s', μ) with $s' = s\mu$ such that

$$G_4 + t.G_5 = -x.G_1 - r.G_3 + s'.A + \mu.B$$

The protocol is in Figure 18.

Complexity. The number of multiplications during issuance by the client is 11 for the client (2 in ClientQuery, 3 in Π_{REP3}, and 6 in Π_{DLEQ}, the computation of $s\sigma$ and $r''G_3$ in ClientFinal being already done for the verification of π_s) and 10 for the server (2 in ServerIssue, 5 in Π_{REP3}, and 3 in Π_{DLEQ}). For redemption, we have 6 multiplications for the client and 7 for the server. The total is 31 multiplications which is worse than our NTAT.

Variant with pairing. When a pairing is available, the protocol simplifies as follows. The proof π_s is replaced by the verification of $e(S, Y + sG_2) = e(T + r''G_3, G_2)$. It becomes a standard blind BBS signature [31]. For redeem, we can now use the standard proof of knowledge of a BBS signature with opening of attribute t [28].

The complexity of issuance for the client becomes 7 multiplications and 2 pairings. It is of 7 multiplications for the server. For redemption, one multiplication (by y) for the verifier can be replaced by 2 pairings using no secret. The total is of 19 multiplications and 4 pairings which becomes slightly better than our Pairing NTAT.

B A NOTE ON PROOF SYSTEMS

Throughout the paper, we use and refer to the following well-known proof systems, written here using the notation of Camenisch-Stadler ([36]).

$$\Pi_{\mathrm{DLEQ}} = \mathrm{NIZK}\{(x) : X = xG \wedge W = xT\}$$
$$\Pi_{\mathrm{REP}n} = \mathrm{NIZK}\{(x_1, x_2, \dots, x_n) : X = x_1G_1 + x_2G_2 + \cdots + x_nG_n\}$$

More generally, we consider $\mathrm{NIZK}\{x : \varphi(x) = X\}$ for a group homomorphism $\varphi : \mathbb{Z}_q^n \to \mathbb{G}^m$, $x \in \mathbb{Z}_q^n$, $X \in \mathbb{G}^m$, which is a generalization of the Fiat-Shamir [30] transform of the Schnorr proof [29]. Namely, the prover picks $r \in \mathbb{Z}_q^n$, computes $\mathsf{comm} = \varphi(r)$, $\mathsf{ch} = H(\mathsf{pp}, \varphi, X, \mathsf{comm})$, $\mathsf{resp} = r - \mathsf{ch}.x$, and the proof is $\pi = (\mathsf{ch}, \mathsf{resp})$. To verify it, the verifier computes $\mathsf{comm}' = \varphi(\mathsf{resp}) + \mathsf{ch}.X$, then checks $\mathsf{ch} = H(\mathsf{pp}, \varphi, X, \mathsf{comm}')$.

Π_{DLEQ} can be seen as a proof with $n = 1$, $m = 2$, and $\varphi(x) = (xG, xT)$. $\Pi_{\mathrm{REP}n}$ can be seen as a proof with $m = 1$ and $\varphi(x_1, \dots, x_n) = x_1G_1 + \cdots + x_nG_n$.

C ADDITIONAL THREAT MODELS

C.1 Anonymity and Its Security Threats

In this section, we briefly describe security threats related to anonymity.

Trusted Hardware as a Threat. We can use trusted hardware to break security or privacy. For instance, Mateus and Vaudenay [37] showed how it can be used by a malicious citizen to sell ballots in e-voting (thus breaking receipt-freeness). Selling ballots is transferring credentials to vote.

Identity Leasing. Puddu et al. [38] construct a system TEEvil based on a trusted hardware (e.g. Intel SGX) by which an account owner would anonymously lease their account to a third party for a limited period of time without revealing their credentials and get anonymous rewards such as cryptocurrencies (e.g. ZCash). Such a system is a threat to digital society, e-voting, and social networks, not even mentioning the business model of content providers. TEEvil acts as a trusted third party in a fair exchange of an identity lease against payment.

Individual Cryptography. Dziembowski et al. [39] propose a protocol to prove that a secret S is stored as a whole in at least one physical device, thus making it *individual.* The motivation is to avoid the creation of a distributed account among participants who do not trust each other but would like to share the cost of creating an account, thus making the account non-individual. In this setting, S is the secret related to the account. It is known by the verifier V and should be known by the individual account owner. The technique to prove it consist of saying that to compute $H(S, N)$ for a random nonce N and a random oracle H requires to know S as a whole. In practice, H is a hash function and could be implemented with MPC at some cost. Hence, some $H(S, N)$ computations could escape from this assumption but not all if there are too many. This approach does not prevent S from being known by more than one device. It does not prevent either against an attack where $H(S, N)$ would be computed by a trusted hardware. Also, assuming that the verifier knows the secret S makes this approach inappropriate in our context, because the redeem server should not learn the secret of the client.

Fraud in distance bounding. The notion of *terrorist fraud* was proposed by Desmedt [40] as an attack where a malicious prover P_1^* helps another malicious prover P_2^* (a terrorist) to enter a country by participating to the identity proof with the honest immigration officer V. In this form of attack, P_1^* has credentials and tries to make P_2^* use them.

To give this attack a sense, we should imagine that exchange between P_2^* and V are purely digital, with no sensor, and we should exclude the trivial attack consisting for P_1^* in providing to P_2^* all their private input then going offline. The rationale is that this input would provide more privilege to P_2^* and, for instance, connect to the wallet of P_1^*. However, P_1^* expects a monetary reward for their participation instead of leaving out their purse. We should further consider the relay attack where P_2^* purely relays communication between P_1^* and V. This can be discarded by assuming that P_2^* is somehow isolated [41], or that interaction with P_2^* must be atomic [42]. For instance, we can assume that P_1^* is offline, or that communication to P_1^* would take too much time and be detected (as it is used for distance-bounding protocols).

Hoarding Attacks. Tyagi et al. studied the construction of a partially oblivious PRF as an extension to PP to prevent so-called hoarding attacks which are already a real-world security problem arising from anonymity[43]. In hoarding (a.k.a farming) attacks, the adversary accumulates anonymously issued tokens over time to mount DDoS attacks by redeeming them all at once. The efficient construction of a partially oblivious PRF allows the client and server to establish a public metadata such as an expiration timestamp so that malicious clients are not given time to hoard the tokens. However, this alone does not solve the problem that the adversary creates an army of malicious clients to collect the tokens to redeem them at once, falling back to the original problem. Adding a proof of humanhood such as a CAPTCHA would not be enough since a distributed human army could still defeat this protection [44]. This shows that protection against transfer of tokens is also necessary.

Token stealing. Kreuter et al. provided a possible solution to token hijacking in the presence of a man-in-the-middle adversary [22]. The adversary, in that setting, is so powerful that it intercepts the communication during the redemption phase and intercept the tokens from a secure channel between the client and the issuer. The aim of the adversary is to redeem the tokens in another web server. Since, the attack scenario is application specific, the countermeasure they propose does not cover the incidents in which the tokens, which may be stored in the cache of the client device, can be stolen even before redemption phase started.

C.2 4 Shades of Non-Transferability

In this section we discuss additional threat models related to non-transferability.

Transfer of Passive Authentication: In the biometric passport standard, an honest prover (the passport) passively proves the validity of an identity by disclosing a digital signature from an authority on a document which associates the picture, name, date of birth, gender, and citizenship of the person. This creates a privacy threat for the passport holder as the malicious verifier could later exhibit this signed document and reveal identity attributes in an undeniable manner [45, 46]. This threat could be mitigated by having the passport to prove knowledge of a valid signature instead of revealing it [47–50].

Transfer of interactive proofs: In a zero-knowledge interactive proof, a prover P proves to a verifier V that a predicate is true without revealing any more information. In the standard model, proofs are *deniable* in the sense that the verifier cannot later on prove to another verifier W that the predicate is true. The threat is that an honest P interacting with a malicious V^* would allow V^* to later prove to W^* that P participated in a successful proof. When moving to non-standard models such as the random oracle model or the common reference string model, deniability may fall down and extra care should be taken to make interactive proofs deniable [51]. However, deniability breaks down again when allowing trusted hardware [52].

Transfer of genuinity: A common way to defeat spam consists of requiring emails to be digitally signed by the sender to prove that it is genuine. This however creates a privacy threat against the honest sender as the malicious receiver could later show undeniable evidences that the sender sent that email. Ring signatures were proposed as a solution for that: making the sender sign on behalf of the ring made of the sender and the receiver. The malicious receiver would no longer be able to prove that the sender sent the email as the signature could have equally be made by the receiver [53]. Having a malicious ring/group member be able to prove that they did *not* produce a given valid signature is an anonymity loss and a privacy threat for other ring/group members. A malicious user could declare a rogue public key and later on prove that this public key was pseudorandomly generated, proving such the ignorance of the secret key and thus the impossibility to have produced the signature. The threat was raised by Ristenpart and Yilek [54] and defeated by requiring members to prove honest key generation during a key registration phase. However, a proof of ignorance is still possible when allowing trusted hardware [52].

Transfer in Undeniable Signatures: In *undeniable signature* [55], a digital signature cannot be universally verified (the signature is called *invisible*) for privacy reasons. It rather requires an interactive proof. Desmedt and Yung [56] showed that by having the verifier implemented a distributed system of malicious verifiers, the signature validity would be proved to all verifiers. The attack was extended by Jakobsson [57] as a way to blackmail the signer. At the same time, some variations of undeniable signatures were proposed to offer non-transferability: confirmer signatures and designated verifier signatures [58, 59].

```
ClientQuery(pp, sk_c, pk_s)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
2: $x \leftarrow \mathsf{sk}_c; X \leftarrow xG_1$
3: $r \leftarrow\!\!\$\ \mathbb{Z}_q$,
4: $\delta \leftarrow\!\!\$\ \mathbb{Z}_q^*$
5: $T \leftarrow \delta \cdot (X + rG_3 + G_4)$
6: $\pi_c \leftarrow \Pi_{\mathrm{REP3}}.\mathsf{Prove}(\mathsf{pp}, X, T, x, \delta, r)$
7: $\mathsf{query} \leftarrow (T, \pi_c)$
8: $\mathsf{st} \leftarrow (\mathsf{pp}, \mathsf{pk}_s, r, \delta, T)$
9: return $(\mathsf{query}, \mathsf{st})$
```

```
ClientFinal(st, resp)
1: $(\mathsf{pp}, \mathsf{pk}_s, r, \delta, T) \leftarrow \mathsf{st}$
2: $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
3: $Y \leftarrow \mathsf{pk}_s$; $(s, S) \leftarrow \mathsf{resp}$
4: if $e(S, Y + sG_2) \neq e(T, G_2)$ return $\bot$
5: $\sigma \leftarrow \frac{1}{\delta}.S$
6: return $(\sigma, \omega = (r, s))$
```

```
ServerIssue(pp, sk_s, pk_c, query)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow \mathsf{pp}$
2: $(T, \pi_c) \leftarrow \mathsf{query}$
3: $y \leftarrow \mathsf{sk}_s$; $X \leftarrow \mathsf{pk}_c$
4: if not $\Pi_{\mathrm{REP3}}.\mathsf{Verify}(\mathsf{pp}, X, T, \pi_c)$
     return $\bot$
5: $s \leftarrow\!\!\$\ \mathbb{Z}_q - \{-y\}$
6: $S \leftarrow \frac{1}{y+s} T$
7: $\mathsf{resp} \leftarrow (s, S)$
8: return resp
```

Figure 19: Token issuance with pairing.

D NTAT VARIANTS

D.1 A Universally-Verifiable (Pairing-Based) Variant

We provide a variant of our protocol based on pairings: it has less operations (namely, the π_s proof by the server is not needed any more) and has a redeem server who does not need the issuer's secret key anymore. For this reason, Server.Verify deviates from Theorem 2.1 by having pk_s as input instead of sk_s. We also briefly indicate how security proofs change for this variant.

Issuance protocol. In this version of the protocol, illustrated in Figure 19, we modify the issuance to ensure public verifiability, i.e., no secret information is needed to verify a token. In particular this means that the server issuing and the one verifying the token can be two distinct entities, and that during issuance the client can verify the validity of the server's response without a server-side proof. We achieve this by using pairings. We therefore implicitly assume that groups used in the set-up are in this case pairing friendly.

More precisely, the client computes (T, π_c) as before. The server returns (s, S) only, in particular, no proof is returned. However, in this version the client can verify the validity of the server's response via a pairing computation. It checks whether $e(S, Y + sG_2) = e(T, G_2)$. Indeed, if the server behaves honestly we have $e(S, Y+sG_2) = e(\frac{1}{y+s}T, (y+s)G_2) = e(T, G_2)$. If this check is passed, they compute σ as before, otherwise they return $\bot$.

Redeem protocol. We can achieve public verifiability by using pairings. This means that the redeemer does not need to know the server secret key y in order to verify the token anymore. The redemption procedure described in Figure 20. It is similar to the previous redemption protocol, except that we replace the check $\sigma' = y\sigma$ on the server side by a pairing check $e(\sigma, Y) = e(\sigma', G_2)$ such that the verifying server does not need to know the issuing server's secret key y anymore.

OMUF-**Security.** The security result is similar for this variant.

The AGM model is enriched as group elements may further be of three different types, depending on the group they live in. There are three types of groups so three types of linear expressions. Depending on the pairing, G_2 (and Y) may not be in the same group

```
Client.Prove(pp, $\sigma$, $\omega$, sk$_c$, pk$_s$)
1: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow$ pp
2: $(r, s) \leftarrow \omega$; $x \leftarrow \mathsf{sk}_c$; $Y \leftarrow \mathsf{pk}_s$
3: $\sigma' \leftarrow xG_1 + rG_3 + G_4 - s\sigma$
4: $\alpha, \beta, \gamma \leftarrow\$ \mathbb{Z}_\mathsf{q}$
5: $Q \leftarrow \alpha G_1 + \beta G_3 + \gamma\sigma$
6: $\rho \leftarrow\$ \{0,1\}^{\ell_\rho}$
7: comm $\leftarrow H_3(\rho, Q)$
8: Send $(\sigma', \mathsf{comm})$ to the server
9: Receive $c$
10: $(v_0, v_1, v_2) \leftarrow (\alpha + cx, \beta + cr, \gamma - cs)$
11: Send$(v_0, v_1, v_2, \rho)$ to the server
```

```
Server.Verify(pp, $\sigma$, pk$_s$)
1: $(\mathsf{q}, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow$ pp
2: $Y \leftarrow \mathsf{pk}_s$
3: Receive $\sigma'$, comm from the client
4: if $e(\sigma, Y) \neq e(\sigma', G_2)$ then abort
5: $c \leftarrow\$ \mathbb{Z}_\mathsf{q}$
6: Send $c$ to the client
7: Receive $(v_0, v_1, v_2, \rho)$ from the client
8: $Q' \leftarrow v_0G_1 + v_1G_3 + v_2\sigma$
9: $Q^* \leftarrow Q' - c(\sigma' - G_4)$
10: comm$^* \leftarrow H_3(\rho, Q^*)$
11: return (comm == comm$^*$)
```

Figure 20: Redemption with pairing.

of $(G_1, G_3, G_4, S_1, \ldots, S_n)$. Vectors for elements in the pairing range group are written in the basis consisting of $e(G_1, G_2)$, $e(G_3, G_2)$, $e(G_4, G_2)$, etc.

No ODDH_y oracle is needed to simulate ORedeem. Thus, the transition to Γ^2 does not introduce an extra oracle.

RUF-**Security.** The security result is the same for this variant.

UNLINK-**Security.** The security result is similar for this variant.

In the transition to Γ^4, the pairing verification is equivalent to the verification so we have no advantage overhead: $\mathsf{Adv}^{\Gamma^3} = \mathsf{Adv}^{\Gamma^4}$.

SOUND-**Security.** The security result is the same for this variant.

UNIQ-**Security.** The security result is the same for this variant.

D.2 Variant with More Attributes

We can easily add attributes $x_1, \ldots, x_n \in \mathbb{Z}_\mathsf{q}$ in NTAT. We assume new random generators $H_1, \ldots, H_n \in \mathbb{G}$ are provided in pp.

During issuance, we distinguish two kinds of attributes: (type-1) public attributes (which are known by the client and the server during issuance) and (client)-private attributes (which are known by the client only and for which the client must prove knowledge). For simplicity, we do not discuss server-private attributes which are put by the server and unknown by the client (e.g. private metadata bits [22, 23]). During issuance, adding more attributes does not affect the communication cost, but it may induce a complexity overhead. Private attributes can be sorted in three types: (type-2) attributes for which it is enough that the client proves its knowledge, (type-3) attributes for which it is required to prove that the attribute is the logarithm of an extra provided group element, and (type-4) attributes for which it is required to prove that the attribute is the same as in an extra commitment to this attribute which is known.

We add attributes by replacing X by $X + \sum_i x_iH_i$. Adding a type-1 attribute x_i is trivial as there is nothing more to do. Adding a private attribute x_i requires to prove the knowledge of x_i in a $\Pi_{\mathsf{REP}n}$ proof. For type-2 attributes, this is enough. However, type-3 and type-4 attributes require that the known value is the same as in an extra (provided) equation related to this attribute. For type-4 attributes, there is even an extra blinding unknown in the extra commitment. For instance, $\mathsf{sk}_c = x$ could be considered as a type-3 attribute.

If we count the computation of x_iH_i on both sides, the additional complexity cost is $n_1 + n_2 + 2n_3 + 3n_4$ multiplications for the client and $n_1 + n_2 + 2n_3 + 3n_4$ for the server, where n_i is the number of

```
ClientQuery(pp, sk$_c$, pk$_s$)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, H_1, \ldots, H_n) \leftarrow$ pp
2: $x \leftarrow \mathsf{sk}_c; X \leftarrow xG_1$
3: $(x_1, \ldots, x_n) \leftarrow$ att
4: $X_i \leftarrow x_iH_i \; i = 1, \ldots, n$
5: $r \leftarrow\$ \mathbb{Z}_q$,
6: $\delta \leftarrow\$ \mathbb{Z}_\mathsf{q}^*$
7: $T \leftarrow \delta \cdot (X + \sum_{i=1}^n x_iH_i + rG_3 + G_4)$
8: $\pi_c \leftarrow \Pi_{\mathsf{REP3}}$.Prove(pp, $X$, extra, $T$, $x$, $(x_i)_i$, $\delta$, $r$)
9: query $\leftarrow (T, \pi_c)$
10: (pp, pk$_s$, $r$, $\delta$, $T$) $\leftarrow$ st
11: return (query, st)
```

```
ClientFinal(st, resp)
1: (pp, pk$_s$, $r$, $\delta$, $T$) $\leftarrow$ st
2: $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow$ pp
3: $Y \leftarrow \mathsf{pk}_s$; $(s, S, \pi_s) \leftarrow$ resp
4: if not $\Pi_{\mathsf{DLEQ}}$.Verify(pp, $Y$, $S$, $T$, $s$, $\pi_s$)
   return $\perp$
5: $\sigma \leftarrow \frac{1}{\delta}.S$
6: return $(\sigma, \omega = (r, s))$
```

```
ServerIssue(pp, sk$_s$, pk$_c$, query)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4, H_1, \ldots., H_n) \leftarrow$ pp
2: $(T, \pi_c) \leftarrow$ query
3: $y \leftarrow \mathsf{sk}_s$; $X \leftarrow \mathsf{pk}_c$
4: if not $\Pi_{\mathsf{REP3}}$.Verify(pp, $X$, extra$T$, $\pi_c$)
   return $\perp$
5: $s \leftarrow\$ \mathbb{Z}_q - \{-y\}$
6: $S \leftarrow \frac{1}{y+s}T$
7: $\pi_s \leftarrow \Pi_{\mathsf{DLEQ}}$.Prove(pp, $yG_2$, $S$, $T$, $s$, $y$)
8: resp $\leftarrow (s, S, \pi_s)$
9: return resp
```

Figure 21: Token issuance with additional attributes.

type-i attributes. The additional communication cost is $(n_3 + n_4)\mathbb{G} + (n_2 + n_3 + 2n_4)\mathsf{q}$.

We illustrate the issuance in Figure 21.

During redemption, we distinguish four kinds of attributes (the type during redemption is independent from the type in issuance): (type-a) attributes which are revealed (the verifier can see the attribute), (type-b) attributes which are present but must remain hidden, and (type-c) attributes which are revealed in the form of an exponential, and (type-d) attributes which are revealed in the form of a commitment. For instance, $\mathsf{sk}_c = x$ could be considered as a type-3b attribute.

The redemption protocol changes naturally. Type-a attributes are revealed. They require n_a extra multiplications for the server, and $n_a\mathsf{q}$ communications. Other attributes generate 1 extra multiplication each for the client and for the server, and 1 extra scalar to transmit. If they are of type-c, the extra equation implies 1 extra multiplication on both sides. If they are of type-d, the extra equation in the proof implies 2 extra multiplications on both sides, 1 extra scalar to transmit, and the commitment implies 1 extra multiplication for the client and one extra group element to transmit. Overall, we need $n_b + 2n_c + 4n_d$ extra multiplications for the client, $n_a + n_b + 2n_c + 3n_d$ extra multiplications for the server, and $n_d\mathbb{G} + (n_a + n_b + n_c + 2n_d)\mathsf{q}$ extra transmission.

We compare it to the versions of BBDT, CHAC, U-Prove with several attributes. In the case of CHAC we consider only one device (i.e. $i = 1$ in the original paper). We consider an interactive version of redemption for all the protocols that we compare ours to. Observe that, while ours fully support all four kinds of attributes for issuance and for redemption distinctly , it is not the case of all other protocols. In fact, in BBDT we have only Type-1, Type-2, Type-a and Type-b attributes. In U-Prove, we have only Type-1, Type-a, Type-b, and Type-d attributes. Note that in U-Prove issuance is independent of the number of attributes since they consider the public attributes generation out of the scope of their protocol. In CHAC, we only have Type-1, Type-a and Type-b. In their model, the Type-b attributes are attributes for which tokens were issued, but that are kept secret and not involved during redemption. This differs slightly from our model, and justifies why n_b does not appear in our comparison. We summarize the comparison results in terms of complexity and communication in Tables 5, 6.

D.3 Variant with Clients using a Trusted Hardware

Our NTAT does not protect against an identity leasing attack [38]. A malicious client could upload their secret sk_c to a trusted platform (i.e. a trusted hardware) which would implement a secure transaction: a temporary usage of the identity against payment. The maliciously used trusted hardware would secure the transaction, making sure that sk_c does not leak and is used only in a limited way. It is already known that maliciously used trusted hardware can break some fragile cryptographic notions such as non-transferability [37]. It is also known that honestly used trusted hardware can be used to defeat maliciously used ones [37].[12] The approach consists of storing sk_c securely inside a trusted hardware and making it do some computations which use sk_c. The selection of those computations should be such that a malicious usage of the trusted hardware does not allow transferability, and be light enough. This is the approach of CHAC [8].

We propose a variant of NTAT in which we focus on securing the storage of sk_c and some computations using it. We therefore differ slightly from our client-server model, and introduce a third entity for these two steps, which we call core (to be consistent with [8, 11]), which consists of a trusted hardware. We present the modification in Figures 22–23. The core is involved during issuance and redeem. However, Section D.4 presents a variant in which the core would not be required for issuance any more.

If we merge the core and the client, we obtain a protocol which is clearly equivalent to our NTAT protocol, hence enjoying the same security. Then, focusing on the core protocol, it is clear that we essentially implement an honest prover for the Schnorr protocol [29]. This protocol is honest-verifier zero-knowledge. With a malicious verifier, it may only leak transferable evidences that the protocol happened but the evidence would not be usable with an honest server. Hence, the core must be involved during a redeem session with an honest server.

D.4 Variant with Client Authentication outside Token Issuance

Assuming that the client proved knowledge of sk_c before engaging the NTAT issuance protocol, the proof of knowledge of sk_c in π_c becomes redundant. Essentially, we assume that the client securely authenticates, with a proof of knowledge of sk_c, and that the server keeps pk_c as a state in a client-server communication session (hence becomes stateful). During the session, the client can request tokens once or multiple times. The authenticated status could further span in multiple secure communication sessions by using techniques such as token binding protocol [60][13]

We believe that such approach is implicitly made by U-Prove [11]

To remove sk_c from issuance, we replace the proof π_c in the protocol by a proof of knowledge Π_{REP2} of δ and r satisfying

$$rG_3 - \frac{1}{\delta}T = -G_4 - X$$

The protocol is in Figure 24.

We stress that the proof of knowledge of sk_c is needed in RUF-security (which is evident by definition of RUF), and also in OMUF-security: there is an attack when such a proof is not done (see Appendix E) and the knowledge of sk_c is used in Γ^8 as the adversary must provide sk_c to the OIssueTok oracle. Our point is not to suppress authentication but rather to avoid doing it redundantly, for instance if the client wants several tokens or if the protocol is such that authentication has already been made.

There are multiple advantages with this approach:

- Simplicity: the issuance protocol no longer needs sk_c and is simpler; the variant with secure hardware no longer needs the secure hardware during issuance; the authentication part of the client is kept separate from issuance.
- Complexity: it saves one multiplication on both sides during issuance and one scalar to communicate. This leads us to a total of 26 scalar multiplications and the transmission of $3\mathbb{G} + 10\mathrm{q} + (\ell_\rho + \ell_H)b$. One separate authentication can be used in several issuance requests.
- Privacy: the issuance protocol no longer gives transferable evidence of an issuance request and becomes deniable.

The drawback is that authentication is not integrated any more and may become more expensive in a separate protocol.

D.5 Binding to an External Valuable Secret

In this section, we explore possible solutions to the scenario where the key used in our NTAT protocol is not in itself valuable, and we wish to bind it to some external valuable secret. More precisely, suppose the user is in possession of an external valuable secret sk_c' satisfying a public predicate $R(\mathsf{sk}_c')$. We consider a hash function H together with some law + compatible with sk_c'.

We enrich the NTAT key generation as shown on Figure 25.

The triplet (pk_c, π, C) plays the role of the public key. Intuitively, C gives a one time pad encryption of sk_c' so that if sk_c is revealed, so will sk_c' be.

We now discuss the Π_{KeyGen} proof. We want to prove the following relation: the existence of x such that $x \cdot G_1 = X \wedge \mathcal{R}(C - H(x))$, where $\mathcal{R}$ is the predicate related to our external valuable public key, x is kept private and X and C are publicly known values. For proving the first part, i.e. proving $X = x \cdot \mathrm{G}_1$ we can use the Schnorr sigma protocol [29]. We can reasonably assume the existence of a sigma protocol for $\mathcal{R}$. Finally, one can combine these two sigma-protocols into an AND sigma-protocol ([61]) and then turn this into a non-interactive zero-knowledge proof using the Fiat-Shamir [30] transform, which gives us the $(\Pi_{\mathsf{KeyGen}}.\mathsf{Prove}, \Pi_{\mathsf{KeyGen}}.\mathsf{Verify})$ protocol.

[12] This arm race reasoning can be pushed even further with *nested trusted hardware*: putting a defensive trusted hardware inside an offensive trusted hardware to re-enable identity leasing. We do not consider it here.

[13] The Token Binding Protocol allows to bind several (TLS) sessions with a proof of possession of the same cryptographic key. When used for token issuance and token redeem, this makes sure that a token owned by an honest client cannot be stolen nor reused. The standard suggests using a trusted hardware to store the binding key. However, this does not protect export by a malicious client (so there is no non-transferability) and this does not offer unlinkability.

Table 5: Complexity comparison. When relevant, n_i, $i \in \{1, 2, 3, 4, a, b, c, d\}$ is the number of Type-i attributes.

	Issuance		Redemption		Total
	Client	Server	Client	Server	
NTAT scheme	$11\times$	$8\times$	$4\times$	$5\times$	$28\times$
NTAT with attributes	$(11 + n_1 + n_2 + 2n_3 + 3n_4)\times$	$(8+n_1+n_2+2n_3+3n_4)\times$	$(4 + n_b + 2n_c + 4n_d)\times$	$(5 + n_a + n_b + 2n_c + 3n_d)\times$	$(28 + 2n_1 + 2n_2 + 4n_3 + 6n_4 + n_a + 2n_b+4n_c+7n_d)\times$
BBDT [21]	$(n_2 + 2) \times +\Pi_{\mathrm{REP}(n_2+1)}$.Prove $+\Pi_{\mathrm{DLEQ}}$.Verify	$2 \times + \Pi_{\mathrm{REP}(n_2+1)}$.Verify $+ \Pi_{\mathrm{DLEQ}}$.Prove	$(n_b + 10)\times$	$(n_b + 9)\times$	$(n_2+2n_b+14)\times + \Pi_{\mathrm{REP}(n_2+1)} + \Pi_{\mathrm{DLEQ}}$
U-Prove [11]	$9\times$	$3\times$	$(3 + n_b + 4n_d)\times$	$(7 + n_b + 3n_d)\times$	$(22+2n_b+7n_d)\times$
CHAC [8]	$5\times$	$(4 + n_1)\times, 5e$	$17\times$	$(10 + n_a)e$	$(26 + n_a)\times,(15 + n_a)e$

Table 6: Communication comparison. When relevant, n_i, $i \in \{1, 2, 3, 4, a, b, c, d\}$ is the number of Type-i attributes.

	Issuance	Redemption	Total
NTAT scheme	$2\mathbb{G} + 7\mathrm{q}$	$1\mathbb{G} + 4\mathrm{q} + (\ell_\rho + \ell_H)b$	$3\mathbb{G} + 11\mathrm{q} + (\ell_\rho + \ell_H)b$
NTAT with attributes	$(2 + n_3 + n_4)\mathbb{G} + (7 + n_2 + n_3 + 2n_4)\mathrm{q}$	$(1 + n_d)\mathbb{G} + (4 + n_a + n_b + n_c + 2n_d)\mathrm{q} + (\ell_\rho + \ell_H)b$	$(3 + n_3 + n_4 + n_d)\mathbb{G} + (11 + n_2 + n_3 + 2n_4 + n_a + n_b + n_c + 2n_d)\mathrm{q} + (\ell_\rho + \ell_H)b$
U-Prove [11]	$3\mathbb{G} + 2\mathrm{q}$	$3\mathbb{G} + (5 + n_b + 3n_d + n_a)\mathrm{q}$	$6\mathbb{G} + (7 + n_b + 3n_d + n_a)\mathrm{q}$
CHAC [8]	$8\mathbb{G} + 1\mathrm{q}$	$8\mathbb{G} + 1\mathrm{q}$	$16\mathbb{G} + 2\mathrm{q}$
BBDT [21]	$2\mathbb{G} + 2\mathrm{q} + \Pi_{\mathrm{REP}(n_b+1)} + \Pi_{\mathrm{DLEQ}}$	$3\mathbb{G} + (n_a + n_b + 7)\mathrm{q} + \ell_H b$	$5\mathbb{G} + (n_a + n_b + 9)\mathrm{q} + \ell_H b + \Pi_{\mathrm{REP}(n_2+1)} + \Pi_{\mathrm{DLEQ}}$

E ON THE NECESSITY OF AUTHENTICATION IN NTAT

The necessity to prove the knowledge of x in π_c. RUF security needs to authenticate the key holder but we could have assumed it was done outside of the protocol. However, it seems necessary in OMUF too, although it is not intuitive. Here is an OMUF attack when π_c does not prove the knowledge of x but only proves commitment to X:

(1) Engage the issuance protocol normally with public key $X_1 = x_1.G_1$ and get $\sigma_1 = \frac{1}{y+s_1}(X_1 + r_1.G_3 + G_4)$.
(2) Engage the issuance protocol with a fake public key $X_2 = \sigma_1$ and get $\sigma_2 = \frac{1}{y+s_2}(X_2 + r_2.G_3 + G_4)$.
(3) Set $x_3 = \frac{1}{1+s_1-s_2}.x_1$, $X_3 = x_3.G_1$, $\sigma_3 = \frac{s_2-s_1}{s_2-s_1-1}\sigma_2 - \frac{1}{s_2-s_1-1}\sigma_1$, $r_3 = \frac{r_2.(s_2-s_1)-r_1}{s_2-s_1-1}$, $s_3 = s_2$.
(4) Yield $x_3, X_3, \sigma_3, r_3, s_3$.

We do have $\sigma_3 = \frac{1}{y+s_3}(x_3.G_1 + r_3.G_3 + G_4)$ which is a forgery for public key $x_3.G_1$ which was never queried to the issuer. If we prove knowledge of x in π_c, the above attack does not work because the adversary does not know $\log X_2$.

Interestingly, this is not an attack against the one-more unforgeability of BBS as the second query to the signing oracle does not yield a signature: it would be a signature on $(\log \sigma : 1, r_2)$ but the adversary cannot provide $\log \sigma_1$. So, this attack makes two sign queries and yield two signatures. It does not break one-more unforgeability for BBS. In our case, the accounting of queries and signatures is per-X values. The attack makes one query with X_1 and gets one signature, one query with X_2 and gets no signature, and no query with X_3 but gets one signature. This breaks our OMUF notion.

F U-Prove

U-Prove [11] is an anonymous credentials system. It uses no pairing but still offers public verifiability.

Issuance in U-Prove is 3-move, initiated by the server (which implies that the server must keep a state). During issuance, ownership verification is out of the scope of U-Prove: anyone can request a token on behalf of anyone with a proposed list of attributes. In particular, they have no RUF security.

The equivalent to redemption is called "presentation". The presentation phase plays two role: authentication/integrity of shown attributes and proof of holding some secrets. Clients can selectively reveal some attributes, or commitments to attributes, or keep them hidden (but their presence is revealed). A client can also insert some *prover information* PI in the token. PI is blindly signed during issuance, i.e., the issuer does not see it. However, it must be revealed during presentation.

Tokens have a unique identifier which must be revealed at redeem. A token may be used multiple times (including to open attributes differently) but in a linkable way. We consider it as a

Core		Client		Server
$\mathsf{pp}, c = x$		pp, X, Y		pp, X, Y
$a \leftarrow\$ \mathbb{Z}_q$				
$\mathsf{comm}_1 \leftarrow a \cdot G_1$				
	$\xrightarrow{\mathsf{comm}_1}$			
		$r, b, c \leftarrow\$ \mathbb{Z}_q,\ \delta \leftarrow\$ \mathbb{Z}_q^*$		
		$T \leftarrow \delta \cdot (X + rG_3 + G_4)$		
		$\mathsf{comm}_2 \leftarrow \mathsf{comm}_1 + bG_3 + cT$		
		$\mathsf{ch} \leftarrow H_1(\mathsf{pp}, X, T, \mathsf{comm}_1, \mathsf{comm}_2)$		
	$\xleftarrow{\mathsf{ch}}$			
$\mathsf{resp}_1 \leftarrow a - \mathsf{ch}.x$				
	$\xrightarrow{\mathsf{resp}_1}$			
		$\mathsf{resp}_2 \leftarrow b - \mathsf{ch}.r$		
		$\mathsf{resp}_3 \leftarrow c + \mathsf{ch}\frac{1}{\delta}$		
			$\xrightarrow{T, \mathsf{ch}, \mathsf{resp}_1, \mathsf{resp}_2, \mathsf{resp}_3}$	
				$\mathsf{comm}'_1 \leftarrow \mathsf{resp}_1.G_1 + \mathsf{ch}.X$
				$\mathsf{comm}'_2 \leftarrow \mathsf{resp}_1.G_1 + \mathsf{resp}_2.G_3 + \mathsf{resp}_3.T - \mathsf{ch}.G_4$
				$\mathsf{ch}' \leftarrow H_1(\mathsf{pp}, X, T, \mathsf{comm}'_1, \mathsf{comm}'_2)$
				if not $(\mathsf{ch} == \mathsf{ch}')$ **return** $\perp$
				$s \leftarrow\$ \mathbb{Z}_q - \{-y\},\ S \leftarrow \frac{1}{y+s}T$
				$\pi_s \leftarrow \Pi_{\mathrm{DLEQ}}.\mathsf{Prove}(\mathsf{pp},\ yG_2, S, T, s,\ y)$
			$\xleftarrow{s, S, \pi_s}$	
		if not $\Pi_{\mathrm{DLEQ}}.\mathsf{Verify}(\mathsf{pp}, Y, S, T, s, \pi_s)$		
		return $\perp$		
		$\sigma \leftarrow \frac{S}{\delta}$		
		output: $(\sigma, \omega = (r, s))$		

Figure 22: Issuance with trusted hardware

single-show system. In the U-Prove specifications, presentation is non-interactive, which makes it transferable. Thus, the informative part of the U-Prove document suggests to make presentation interactive.

For non-transferability, U-Prove suggests to disincentivize transfer by embedding a valuable secret in one additional attribute [11, Section 3.5]. This attribute value should not be a hash (otherwise, the transfer of the hash would still be possible). The idea is that knowing the attribute values is necessary for the client during presentation. However, the attribute value must be shared with the issuer so we do not see it as an appropriate solution (unless U-Prove is extended to allow attribute registration without sharing).

U-Prove suggests another technique for non-transferability: device binding. This ends up in a similar setting as in CHAC [8] with a client split into a "core" and a "helper". A difference with CHAC is that there is no ownership verification during issuance - the core is not involved in issuance. Another difference is that in U-Prove, interaction with the core needs 4 moves instead of 2 during presentation. Device binding ensures that the secret of the device was used to make the presentation proof. However, the security of U-Prove is not formally proven.

We rewrite in Figure 26 and Figure 27 the U-Prove protocol in a simplified manner: using device binding, and with interactive presentation. The algorithms in issuance are run as follows : the server starts by running Server.Issue$_0$, then it is the client's turn with ClientQuery, then Server.Issue$_1$ and finally ClientFinal This is the "option" which would compare the best with NTAT. However, it is vulnerable to ROS attack which makes U-Prove *not* OMUF secure.

ROS Attack. Benhamouda et al. [17] presented the ROS attack which applies to U-Prove as follows. We consider a malicious client who engages with n issuance sessions in parallel with the same set of attributes. This implies that the sessions will have the values of Γ and Σ_z in common, in addition to pp and Y. The forged tokens will have the same H. A valid token is a tuple $(H, \mathsf{PI}_i, \Sigma'_z, \sigma'_{c,i}, \sigma'_{r,i})$

Core		Client		Server
$\mathsf{pp}, \mathsf{sk}_c = x$		$\mathsf{pp}, X, Y, \sigma, \omega,$		pp, σ, y
$\alpha \leftarrow\$ \mathbb{Z}_q$				
$\mathsf{comm}_1 \leftarrow \alpha G_1$				
	$\xrightarrow{\mathsf{comm}_1}$			
		$(r, s) \leftarrow \omega$		
		$\sigma' \leftarrow X + rG_3 + G_4 - s\sigma$		
		$\beta, \gamma \leftarrow\$ \mathbb{Z}_q^2$		
		$Q \leftarrow \mathsf{comm}_1 + \beta G_3 + \gamma\sigma$		
		$\rho \leftarrow\$ \{0,1\}^\ell$		
		$\mathsf{comm} \leftarrow H_3(\rho, Q)$		
			$\xrightarrow{\sigma', \mathsf{comm}}$	
				If $\sigma' \neq y\sigma$ **then** abort
				$c \leftarrow\$ \mathbb{Z}_q$
			$\xleftarrow{c}$	
	$\xleftarrow{c}$			
$v_0 \leftarrow \alpha + cx$				
	$\xrightarrow{v_0}$			
		$v_1 \leftarrow \beta + cr$		
		$v_2 \leftarrow \gamma - cs$		
			$\xrightarrow{\rho, v_0, v_1, v_2}$	
				$Q' \leftarrow v_0G_1 + v_1G_3 + v_2\sigma$
				$Q^* \leftarrow Q' - c(\sigma' - G_4)$
				$\mathsf{comm}^* \leftarrow H_3(\rho, Q^*)$
				output: $(\mathsf{comm} == \mathsf{comm}')$

Figure 23: Redemption with trusted hardware

```
Prover(pp, $X, T, \delta, r$)
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow pp$
2: $(b, c) \leftarrow\$ \mathbb{Z}_q^2$
3: $comm_2 \leftarrow bG_3 + cT$
4: $ch \leftarrow H_1(pp, X, T, comm_2)$
5: $resp_2 \leftarrow b - ch.r \bmod q$
6: $resp_3 \leftarrow c + ch.\frac{1}{\delta} \bmod q$ return ch, $resp_2$, $resp_3$
```

```
Verifier(pp, $X, T$, (ch, $resp_2$, $resp_3$))
1: $(q, \mathbb{G}, G_1, G_2, G_3, G_4) \leftarrow pp$
2: $comm'_2 \leftarrow resp_2.G_3 + resp_3.T - ch.G_4$
3: $ch' \leftarrow H_1(pp, X, T, comm'_2)$
   return $(ch == ch')$
```

Figure 24: Π_{REP2}**: Client proof** π_c **with no** sk_c**.**

```
Client.KeyGen(pp, $sk'_c$)
1: $pp \rightarrow (q, \mathbb{G}, G_1, \dots, G_4)$
2: $x \leftarrow\$ \mathbb{Z}_q$
3: $X \leftarrow xG_1$
4: $(sk_c, pk_c) \leftarrow (x, X)$
5: $C \leftarrow H(x) + sk'_c$
6: $\pi \leftarrow \Pi_{KeyGen}.Prove(pp, C, x, X)$
7: return $(sk_c, (pk_c, \pi, C))$
```

```
KeyVer(pp, $sk_c$, ($pk_c, \pi, C$))
1: $pp \rightarrow (q, \mathbb{G}, G_1, \dots, G_4)$
2: return $\Pi_{KeyGen}.Verify(pp, \pi, C, pk_c)$
```

Figure 25: Updated client key generation to bind to an external valuable secret

satisfying $\sigma'_{c,i} = \mathsf{Hash}(H, \mathsf{PI}_i, \Sigma'_z, \Sigma'_{a,i}, \Sigma'_{b,i})$, where

$$\begin{pmatrix} \Sigma'_{a,i} \\ \Sigma'_{b,i} \end{pmatrix} = \sigma'_{r,i} \begin{pmatrix} Y \\ H \end{pmatrix} - \sigma'_{c,i} \begin{pmatrix} G_0 \\ \Sigma'_z \end{pmatrix}$$

```
ClientQuery(pp, $sk_c, pk_s, (\Sigma_z, \Sigma_a, \Sigma_b)$)
1: pick $\alpha, \beta_1, \beta_2$
2: $H \leftarrow \alpha(G_{xt} + pk_c)$
3: $\Sigma'_z \leftarrow \alpha\Sigma_z$
4: $\Sigma'_a \leftarrow \beta_1 G_0 + \beta_2 Y + \Sigma_a$
5: $\Sigma'_b \leftarrow \beta_1 \Sigma'_z + \beta_2 H + \alpha\Sigma_b$
6: $\sigma'_c \leftarrow Hash(H, PI, \Sigma'_z, \Sigma'_a, \Sigma'_b)$
7: $\sigma_c \leftarrow \sigma'_c + \beta_1$
8: $st \leftarrow (pp, \beta_2, H, pk_s, \sigma'_c, \Sigma'_a, \Sigma'_b, \Sigma'_z)$
9: return $\sigma_c, st$
ClientFinal($st, \sigma_r$)
1: $\sigma'_r \leftarrow \sigma_r + \beta_2$
2:
3: if $\Sigma'_a + \Sigma'_b = \sigma'_r(H + Y) - \sigma'_c(G_0 + \Sigma'_z)$ then return token: $(H, PI, \Sigma'_z, \sigma'_c, \sigma'_r)$ and witness: $\alpha$
```

```
Server.Issue_0(pp, $pk_c, sk_s, pk_s$)
1: $\Gamma \leftarrow G_{xt} + pk_c$
2: $\Sigma_z \leftarrow y_0\Gamma$
3: pick $w$
4: $\Sigma_a \leftarrow wY; \Sigma_b \leftarrow w\Gamma$
5: $st \leftarrow (sk_s, w)$
6: return $(st, (\Sigma_z, \Sigma_a, \Sigma_b))$
Server.Issue_1(st, $\sigma_c$)
1: $st \leftarrow (sk_s, w)$
2: $\sigma_r \leftarrow y_0\sigma_c + w$
3: return $\sigma_r$
```

Figure 26: U-Prove **issuance with device binding**

By running n parallel token requests, with $n \geq \log_2 q$, the bitlength of the group order, the malicious client first collects Σ_z and n pairs of $(\Sigma_{a,i}, \Sigma_{b,i})$. The malicious prover selects two possible choices $\mathsf{PI}_{i,\mathrm{b}}$, $\mathrm{b} \in \{0,1\}$ for each i, computes $2n$ hashes to get $\sigma_{c,i,\mathrm{b}} = \mathsf{Hash}(H, \mathsf{PI}_{i,\mathrm{b}}, \Sigma_z, \Sigma_{a,i}, \Sigma_{b,i})$ (following the protocol as if $\alpha = 1$ and

$$\mathsf{pp} = (G_0, G_{xt}, G_d) \text{ (where, } G_{xt} = G_0 + x_t G_t\text{)}, (\mathsf{sk}_c, \mathsf{pk}_c) = (x_d, x_d G_d)$$

$$\begin{array}{lcccr}
\text{core} & & \text{helper} & & \text{verifier} \\
\mathsf{sk}_c = x_d & & H, \mathsf{PI}, \Sigma'_z, \sigma'_c, \sigma'_r, \alpha & & \mathsf{pk}_s = Y \\
\hline
\text{pick } w'_d & & & & \\
A_d \leftarrow w'_d G_d & & & & \\
 & \xrightarrow{A_d} & & & \\
 & & \text{pick } w_0, w_d & & \\
 & & \mathsf{comm} \leftarrow w_0 H + w_d G_d + A_d & & \\
 & & & \xrightarrow{H, \mathsf{PI}, \Sigma'_z, \sigma'_c, \sigma'_r, \mathsf{comm}} & \\
 & & & & \sigma'_c \overset{?}{=} \mathsf{Hash}(H, \mathsf{PI}, \Sigma'_z, \sigma'_r Y \\
 & & & & - \sigma'_c G_0, \sigma'_r H - \sigma'_c \Sigma'_z) \\
 & & & & \text{pick } a \\
 & & & \xleftarrow{a} & \\
 & & c_p \leftarrow \mathsf{Hash}(H, a) & & \\
 & \xleftarrow{c_p} & & & \\
c \leftarrow \mathsf{Hash}(c_p) & & & & \\
r'_d \leftarrow -c x_d + w'_d & & & & \\
 & \xrightarrow{r'_d} & & & \\
 & & c \leftarrow \mathsf{Hash}(c_p) & & \\
 & & r_0 \leftarrow \frac{c}{\alpha} + w_0 & & \\
 & & r_d \leftarrow r'_d + w_d & & \\
 & & & \xrightarrow{r_0, r_d} & \\
 & & & & c_p \leftarrow \mathsf{Hash}(H, a) \\
 & & & & c \leftarrow \mathsf{Hash}(c_p) \\
 & & & & \mathsf{comm} \overset{?}{=} -c G_{xt} + r_0 H + r_d G_d \\
\hline
\end{array}$$

Figure 27: U-Prove **interactive presentation with device binding**

$\beta_1 = \beta_2 = 0$), then makes a smart choice to select $\mathsf{b} = \mathsf{b}_i$ for each i so that by setting $\mathsf{PI}_i = \mathsf{PI}_{i,\mathsf{b}_i}$ and $\sigma_{c,i} = \sigma_{c,\mathsf{b}_i}$, the adversary sends $\sigma_{c,i}$ to wrap up the issuance, gets back $\sigma_{r,i}$, and is able to forge $n+1$ valid tokens. Therefore, U-Prove is not OMUF-secure: by making n issuance queries, the adversary gets $n+1$ valid tokens with pairwise different token identifiers. (A token identifier is the hash of the tuple $(H, \mathsf{PI}, \Sigma'_z, \sigma'_c, \sigma'_r)$.)

More precisely, we assume that the index i goes from 0 to $n-1$. The adversary sets $z_i = \frac{2^i}{\sigma_{c,i,1} - \sigma_{c,i,0}}$ and

$$z_n = -\sum_{i=0}^{n-1} z_i \sigma_{c,i,0}$$

This has the property that for any selection of $\mathsf{b}_0, \ldots, \mathsf{b}_{n-1}$, we have

$$z_0 \sigma_{c,0,\mathsf{b}_0} + \cdots + z_{n-1} \sigma_{c,n-1,\mathsf{b}_{n-1}} + z_n = \mathsf{b}_0 + 2\mathsf{b}_1 + \cdots + 2^{n-1} \mathsf{b}_{n-1}$$

Then, the adversary defines

$$\begin{pmatrix} \Sigma_{a,n} \\ \Sigma_{b,n} \end{pmatrix} = \sum_{i=0}^{n-1} z_i \begin{pmatrix} \Sigma_{a,i} \\ \Sigma_{b,i} \end{pmatrix} - z_n \begin{pmatrix} G_0 \\ \Sigma'_z \end{pmatrix}$$

Hence, by computing $\sigma_{c,n} = \mathsf{Hash}(H, \mathsf{PI}^*, \Sigma_{a,n}, \Sigma_{b,n}) \bmod \mathsf{q}$ for an arbitrary PI^* then setting the b_i as the binary representation of $\sigma_{c,n}$ (which is possible when $n \geq \log_2 \mathsf{q}$), sending the $\sigma_{c,i}$ to finish the issuances and getting the $\sigma_{r,i}$, we obtain the relation

$$\begin{pmatrix} \Sigma_{a,n} \\ \Sigma_{b,n} \end{pmatrix} = \left(\sum_{i=0}^{n-1} z_i \sigma_{r,i} \right) \begin{pmatrix} Y \\ H \end{pmatrix} - \sigma_{c,n} \begin{pmatrix} G_0 \\ \Sigma'_z \end{pmatrix}$$

so an extra valid token $(H, \mathsf{PI}^*, \Sigma'_z, \sigma_{c,n}, \sigma_{r,n})$.

The attack above uses $\alpha = 1$ and $\beta_1 = \beta_2 = 0$ for simplicity but could also use random choices for α, β_1, β_2.

There may be several ways to defeat the ROS attack. We could require PI to be constant. We could use H as a token identifier. Those fixes would require to revisit the security of U-Prove in all applications. The cleanest way would be to add an attribute which would play the role of the token identifier so that issuance requests for the same set of attributes would produce tokens which are considered as being the same. Such additional attribute would be revealed at presentation (so add one scalar to transmit and one multiplication for the server) but would be hidden at issuance. However, there is currently no such mechanism in U-Prove. So far, all attributes must be shared with the issuer.

One can argue that the ROS attack applies in corner use cases of U-Prove and can be defeated. However, it illustrates that using anonymous credentials as anonymous tokens and hope that they would automatically be OMUF-secure is a wrong belief.

G WITH A LITTLE HELP FROM MY FRIEND

In this section, we simplify the CHAC protocols for a single issuance [8]. It relies on pairing and the redeemer does not need the secret key. However, there is no variant without pairing.

CHAC separates the client into a "core" and a "helper". The core has low capabilities but is secure. The helper has computational power but is not secure. An essential part of CHAC is to put the minimum inside the core. In addition to this, the protocol offers an effective aggregation protocol so that having many attributes does not add too much complexity.

Their notion of non-transferability is called "dependability". It relies on the assumption that the core belonging to the user is a trusted hardware which would not deviate from its protocol. Hence, the malicious user cannot hold their own secret.

The protocol has an additional move in issuance. Furthermore, all attributes must be shared during issuance. We illustrate the protocol in Figure 28–29 as well as its merged client version in Figure 30.

$\mathsf{crs} = (Y_1, Y_2) = (g_1^{\delta}, g_2^{\delta})$, $(\mathsf{sk}, \mathsf{pk}) = (Y_1^{\alpha}, (g_1, g_1^{\alpha}))$, $(\mathsf{isk}, \mathsf{ipk}) = ((F, (x_{i,j})_{i,j}), (g_2^{x_{i,j}})_{i,j})$, $\mathsf{Attr} = (v_i)_i$, $j \in \{1, \ldots, \ell\}$, $\ell = 2$

core	helper	server
$\mathsf{sk} = Y_1^{\alpha}$		$\mathsf{isk} = (F, (x_{i,j})_{i,j})$

pick nonce

$\xleftarrow{\text{nonce}}$

$\mathsf{aid} \leftarrow H(\mathsf{Attr}, \mathsf{nonce})$

$\xleftarrow{\text{aid}}$

pick k, r
$U_1 \leftarrow g_1^k, U_2 \leftarrow g_2^k$
$\mathsf{Sig} \leftarrow \mathsf{sk} \cdot H(\mathsf{aid})^r$
$w \leftarrow r/k$

$\xrightarrow{U_1, U_2, \mathsf{Sig}, w}$

$S_1 \leftarrow U_1^w, S_2 \leftarrow U_2^w$

$\xrightarrow{\mathsf{pk}, \mathsf{Sig}, S_1, S_2}$

$\mathsf{aid} \leftarrow H(\mathsf{Attr}, \mathsf{nonce})$
$e(S_1, g_2) \stackrel{?}{=} e(g_1, S_2)$
$e(\mathsf{Sig}, g_2) \stackrel{?}{=} e(\mathsf{pk}, Y_2) e(H(\mathsf{aid}), S_2)$
$y \leftarrow F(\mathsf{pk})$
$Z_i \leftarrow (\prod_j \mathsf{pk}_j^{x_{i,j}})^y$
$W_1 \leftarrow g_1^{\frac{1}{y}}, W_2 \leftarrow g_2^{\frac{1}{y}}$
$V_i \leftarrow H((\mathsf{ipk}_{i,j})_j, v_i)^{\frac{1}{y}}$

$\xleftarrow{W_1, W_2, (Z_i, V_i)_i}$

Figure 28: CHAC **Issuance**

$\mathsf{crs} = (Y_1, Y_2) = (g_1^{\delta}, g_2^{\delta})$, $(\mathsf{sk}, \mathsf{pk}) = (Y_1^{\alpha}, (g_1, g_1^{\alpha}))$, $(\mathsf{isk}, \mathsf{ipk}) = ((F, (x_{i,j})_{i,j}), (g_2^{x_{i,j}})_{i,j})$, $\mathsf{Attr} = (v_i)_i$, $j \in \{1, \ldots, \ell\}$, $\ell = 2$

core: $\mathsf{sk} = Y_1^{\alpha}$ | helper | server: $\mathsf{isk} = (F, (x_{i,j})_{i,j})$

server: pick nonce

$\xleftarrow{\text{nonce}}$

helper: $\mathsf{aid} \leftarrow H(\mathsf{Attr}, \mathsf{nonce})$

$\xleftarrow{\mathsf{aid}}$

core:

pick k, r
$U_1 \leftarrow g_1^{k}, U_2 \leftarrow g_2^{k}$
$\mathsf{Sig} \leftarrow \mathsf{sk} \cdot H(\mathsf{aid})^{r}$
$w \leftarrow r/k$

$\xrightarrow{U_1, U_2, \mathsf{Sig}, w}$

helper:

$S_1 \leftarrow U_1^{w}, S_2 \leftarrow U_2^{w}$
$Z \leftarrow \prod_i Z_i, V \leftarrow \prod_i V_i$
pick r', k', ψ
$A' \leftarrow g_1^{r'}, B' \leftarrow \mathsf{pk}^{r'}$
$\mathsf{pk}' \leftarrow (A', B')$
$\mathsf{Sig}' \leftarrow \mathsf{Sig}^{r'} H(\mathsf{aid})^{k'}$
$S_1' \leftarrow S_1^{r'} g_1^{k'}, S_2' \leftarrow S_2^{r'} g_2^{k'}$
$Z' \leftarrow Z^{r'\psi}$
$W_1' \leftarrow W_1^{\frac{1}{\psi}}, W_2' \leftarrow W_2^{\frac{1}{\psi}}$
$V' \leftarrow V^{\frac{1}{\psi}}$
$\mathsf{msg} \leftarrow (\mathsf{pk}', \mathsf{Sig}', S_1', S_2', Z', W_1', W_2', V')$

$\xrightarrow{\mathsf{msg}}$

server:

$\mathsf{aid} \leftarrow H(\mathsf{Attr}, \mathsf{nonce})$
$e(S_1', g_2) \stackrel{?}{=} e(g_1, S_2')$
$e(\mathsf{Sig}', g_2) \stackrel{?}{=} e(\mathsf{pk}_2', Y_2) e(H(\mathsf{aid}), S_2')$
$\prod_j e(\mathsf{pk}_j', \prod_i \mathsf{ipk}_{i,j}) \stackrel{?}{=} e(Z', W_2')$
$e(W_1', g_2) \stackrel{?}{=} e(g_1, W_2')$
$e(W_1', \prod_i H((\mathsf{ipk}_{i,j})_j, v_i)) \stackrel{?}{=} e(g_1, V')$

Figure 29: CHAC **Redeem**

$\mathsf{crs} = (Y_1, Y_2) = (g_1^{\delta}, g_2^{\delta})$, $(\mathsf{sk}, \mathsf{pk}) = (Y_1^{\alpha}, (g_1, g_1^{\alpha}))$, $(\mathsf{isk}, \mathsf{ipk}) = ((F, x_1, x_2), (g_2^{x_1}, g_2^{x_2}))$

client		server
$\mathsf{sk} = Y_1^{\alpha}$		$\mathsf{isk} = (F, x_1, x_2)$
		pick nonce
	$\xleftarrow{\text{nonce}}$	
$h \leftarrow H(\text{nonce})$ pick r $S_1 \leftarrow g_1^r, S_2 \leftarrow g_2^r$ $\mathsf{Sig} \leftarrow \mathsf{sk} \cdot h^r$		
	$\xrightarrow{\mathsf{pk}, \mathsf{Sig}, S_1, S_2}$	
		$h \leftarrow H(\text{nonce})$ $e(S_1, g_2) \overset{?}{=} e(g_1, S_2)$ $e(\mathsf{Sig}, g_2) \overset{?}{=} e(\mathsf{pk}_2, Y_2) e(h, S_2)$ $y \leftarrow F(\mathsf{pk})$ $Z \leftarrow (\mathsf{pk}_1^{x_1} \mathsf{pk}_2^{x_2})^y$ $W_1 \leftarrow g_1^{\frac{1}{y}}, W_2 \leftarrow g_2^{\frac{1}{y}}$ $V \leftarrow H(\mathsf{ipk})^{\frac{1}{y}}$
	$\xleftarrow{W_1, W_2, Z, V}$	
		pick nonce
	$\xleftarrow{\text{nonce}}$	
$h \leftarrow H(\text{nonce})$ pick r', k'', ψ $(k'' = rr' + k')$ $S_1' \leftarrow g_1^{k''}, S_2 \leftarrow g_2^{k''}$ $\mathsf{Sig}' \leftarrow \mathsf{sk}^{r'} h^{k''}$ $A' \leftarrow g_1^{r'}, B' \leftarrow \mathsf{pk}_2^{r'}$ $\mathsf{pk}' \leftarrow (A', B')$ $Z' \leftarrow Z^{r'\psi}$ $W_1' \leftarrow W_1^{\frac{1}{\psi}}, W_2' \leftarrow W_2^{\frac{1}{\psi}}$ $V' \leftarrow V^{\frac{1}{\psi}}$ $\mathsf{msg} \leftarrow (\mathsf{pk}', \mathsf{Sig}', S_1', S_2', Z', W_1', W_2', V')$		
	$\xrightarrow{\mathsf{msg}}$	
		$h \leftarrow H(\text{nonce})$ $e(S_1', g_2) \overset{?}{=} e(g_1, S_2')$ $e(\mathsf{Sig}', g_2) \overset{?}{=} e(\mathsf{pk}_2', Y_2) e(h, S_2')$ $e(\mathsf{pk}_1', \mathsf{ipk}_1) e(\mathsf{pk}_2', \mathsf{ipk}_2) \overset{?}{=} e(Z', W_2')$ $e(W_1', g_2) \overset{?}{=} e(g_1, W_2')$ $e(W_1', H(\mathsf{ipk})) \overset{?}{=} e(g_1, V')$

Figure 30: CHAC **with a single (and constant) attribute and merged client**