Non-Transferable Anonymous Tokens by Secret Binding

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

Abdullah Talayhan  
abdullah.talayhan@epfl.ch  
EPFL  
Switzerland

Laurane Marco  
laurane.marco@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 formalize 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) \[2\], private access to services \[2\], private statistics aggregation \[3\], or even for private contact tracing \[4\].

These protocols are usually designed in a client-server setting. 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:

1.1 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 \((sk_c, pk_c)\) 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, which 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), unforgeability 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 un-linkability. 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: \(Attr = \{\text{state} = \text{NY}, \text{age} = 34\}\)). 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, \(\text{state} = \text{NY} \land \text{age} \geq 18\)) (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. Baldimtsi 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 is 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.

\(^\text{4}\)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 \(1^\lambda\) denotes the security parameter \(\lambda\) 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 \(\{G, (Y_i = x^i \cdot G)_{i \in \{1,...,n}\}\}\), 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 redemption; 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

\textbf{Definition 2.1 (Anonymous Tokens with Interactive Redemption ).}

An Anonymous Token with Interactive Redemption is a tuple of algorithms defined in the following way:

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

1. Setup$(\lambda^2) \rightarrow pp$
2. Server.KeyGen(pp) $\rightarrow (sk_c, pk_c)$
3. Client.KeyGen(pp) $\rightarrow (sk_c, pk_c)$
4. if $\neg$KeyVer(pp, $sk_c$, $pk_c$) then return 0
5. Client.Query(pp, $sk_c$, $pk_c$) $\rightarrow$ (query, st)
6. Server.Issue(pp, $sk_c$, $pk_c$, query) $\rightarrow$ resp
7. Client.Final(st, resp) $\rightarrow$ ($\sigma$, $\omega$)
8. (Client.Prove(pp, $\sigma$, $\omega$), $sk_c$, $pk_c$), Server.Verify(pp, $\sigma$, $sk_c$) $\rightarrow$ 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

- Server.Issue(pp, $sk_c$, $pk_c$, query) $\rightarrow$ resp/⊥: 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 ⊥.

- Client.Final(st, resp) $\rightarrow$ ($\sigma$, $\omega$): 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 $\sigma$ (which is essentially a blind commitment to the client’s secret key), as well as decommitment information $\omega$ to the token.

- (Client.Prove(pp, $\sigma$, $\omega$, $sk_c$, $pk_c$), Server.Verify(pp, $\sigma$, $sk_c$)) $\rightarrow$ 0/1: Given a token $\sigma$, the redemption protocol consists of an interactive proof of knowledge of ($\omega$, $sk_c$) between the client (prover) and the server (verifier) to ensure that the client knows the secret key $sk_c$ and the decommitment information $\omega$.

Client.Prove and Server.Verify are two interactive algorithms. For instance, Server.Verify initially takes input (pp, $\sigma$, $sk_c$) 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 from the client. It continues until the new state is ⊥ (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 redeemption unless mentioned otherwise.

Definition 2.2 (Correctness). An AT scheme is correct if for all $\lambda$ 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.

---

1. 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 $\sigma$.
2. We also consider variants of AT where Server.Verify does not use $sk_c$, i.e., the server’s secret is not needed in redeem. In Appendix D.1 we propose such variant.
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 $\omega_j$). However, our forthcoming soundness property states that we can extract a consistent ($\omega_j, sk_c, pk_c$) 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 $pk_c$ and $sk_c$ and also change it at will.

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

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

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

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

$\text{OReveal}$ now runs sessions of the redeem protocol with the client. The session sid defines the inner state $s[t][sid]$ 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, $A$ can launch the redeem of the same tokens (of issuance session $j$) many times (with several redeem sessions sid), and possibly with different $pk_c$. For the acceptance

UNLINK$_2(A)$

1. Setup($1^\lambda$) → $pp$
2. Server.KeyGen($pp$) → ($sk_a, pk_a$)
3. $\text{Query}(j)$ → $M_{query}$
4. $\text{Final}(M_{query})$ → $M_{final}$
5. $\text{Redeem}(M_{final})$ → $M_{redeem}$
6. $\text{Query}(j)$ → $M_{query}$
7. $\text{Final}(M_{query})$ → $M_{final}$
8. $\text{Redeem}(M_{final})$ → $M_{redeem}$
9. $\text{Query}(j)$ → $M_{query}$
10. $\text{Final}(M_{query})$ → $M_{final}$
11. $\text{Redeem}(M_{final})$ → $M_{redeem}$
12. $\text{Query}(j)$ → $M_{query}$
13. $\text{Final}(M_{query})$ → $M_{final}$
14. $\text{Redeem}(M_{final})$ → $M_{redeem}$
15. $\text{Query}(j)$ → $M_{query}$
16. $\text{Final}(M_{query})$ → $M_{final}$
17. $\text{Redeem}(M_{final})$ → $M_{redeem}$
18. $\text{Query}(j)$ → $M_{query}$
19. $\text{Final}(M_{query})$ → $M_{final}$
20. $\text{Redeem}(M_{final})$ → $M_{redeem}$
21. $\text{Query}(j)$ → $M_{query}$
22. $\text{Final}(M_{query})$ → $M_{final}$
23. $\text{Redeem}(M_{final})$ → $M_{redeem}$
24. $\text{Query}(j)$ → $M_{query}$
25. $\text{Final}(M_{query})$ → $M_{final}$
26. $\text{Redeem}(M_{final})$ → $M_{redeem}$
27. $\text{Query}(j)$ → $M_{query}$
28. $\text{Final}(M_{query})$ → $M_{final}$
29. $\text{Redeem}(M_{final})$ → $M_{redeem}$
30. $\text{Query}(j)$ → $M_{query}$
31. $\text{Final}(M_{query})$ → $M_{final}$
32. $\text{Redeem}(M_{final})$ → $M_{redeem}$
33. $\text{Query}(j)$ → $M_{query}$
34. $\text{Final}(M_{query})$ → $M_{final}$
35. $\text{Redeem}(M_{final})$ → $M_{redeem}$

Figure 3: Request unforgeability game (Definition 2.5).

Figure 4: Unlinkability game (Definition 2.6).

of the forged query, the adversary must provide for which $sk_3$ 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). I.e., we assume that the server is set up with a key pair generated using Server.KeyGen.5

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 $A$, we have

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

where the UNLINK$_2(A)$ 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 $A_2$ is able to redeem a token $\sigma$ after being given information from an

---

5We need this assumption in the proof of Theorem 4.3 in $I^*$ on page 17.

6To allow malicious key generation, we would need an extra key registration process where the server would prove the knowledge of $sk_c$, 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 $A_1$, $A_2$ must know some witness $\omega$ and some key pair $(sk_\omega, pk_\omega)$ associated to $\omega$. The "must know" part of it is formalized by the notion of extractor.

Then, we define the uniqueness property of $(\omega, sk_\omega, pk_\omega)$ to formalize that $\omega$ is binding towards a unique $(sk_\omega, pk_\omega)$, thus to a unique $(sk_\omega, pk_\omega)$. Hence, transfer implies giving this pair.

**Soundness.** The game starts with an adversary $A_1$ (who could possibly get the secret $sk_\omega$ of the server, without loss of generality) who returns a state as well as a token $\sigma$.

We adapt the notion of knowledge soundness of proofs of knowledge to our setting. Now, we define the uniqueness property of $(\omega, sk_\omega, pk_\omega)$ which is limited to $K(p_{rnd}, \lambda)$ on average. The complexity of $E$ is $K(p_{rnd}, \lambda) \times Poly(\lambda)$. The number of oracle calls by $E$ is also limited to $K(p_{rnd}, \lambda)$ times the bound on the number of oracle calls by $A_2$.

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 $E$ succeeds to extract by running $A_2$ twice on average. In our results, we use $K(x, \lambda) = \frac{2}{\sqrt{x(\epsilon(x))}}$ for some negligible function $\epsilon(\lambda)$. Hence, for $p_{rnd}$ larger than a positive constant, we have $K = O(1)$.

When calling $A_2$ in a black box manner (which is counted as one unit of complexity for $E$), the extractor simulates the responses from the oracle queries made by $A_2$. This implies that the execution of $A_2$ is stalled upon an oracle query and the execution resumes when the extractor provides the answer to the query. By calling $A_2$, $E$ controls the random coins of $A_1$ and can easily rewind $A_2$ by replaying the same input and coins. Note that $E$ must keep the same capabilities as $A_2$: if $A_2$ needs to make new tokens to succeed to redeem, $E$ also needs to call $OlsueTok$ to extract. $A_2$ works without this oracle, extraction does not need it either.

The non-transferability notion by Hanzlick and Slaamig [8] is much simpler and falsifiable. This comes from the fact that they assume that the device holding $sk_\omega$ is trusted, so they do not have to deal with a malicious $A_1$, misusing it. Essentially, their non-transferability notion is a form of unforgeability.

**Uniqueness.** Finally, we define the uniqueness notion. We say $E$ is non-transferable if it is both sound and ensures uniqueness.

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

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

where the UNIQ($A$) 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 $E$ that we can use as follows when we have a scheme with non-interactive redeem. We set $A_1$ to the honest client setting up their keys and issuing a token then set state to the non-interactive redeem message. Then, $A_2$ is set to a dummy algorithm just sending state: essentially, $A_2$ sets its output equal to its input. Due to correctness, $p_{rnd}$ is always 1, meaning $K(p_{rnd}, \lambda)$ is assumed to be small. Clearly,
The ability to extract $\sigma$ playing $K(p_{nd}, \lambda)$ times with $A_2$ learns nothing in this interaction. Thus, it reduces to an extractor not using $A_2$ at all and still being able to extract. Soundness implies that $E$ can extract $sk_c$ from state by playing with the issuance and redeem oreacles. Assuming uniqueness is satisfied, this must be the correct $sk_c$. The crucial point is that the input state to $A_2$ 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 $\pi_c$ by the client, the non-interactive proof $\pi_b$ 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.

3.1 Our NTAT scheme

Our protocol uses random oracles which are denoted differently as $H_1$ to enforce domain separation. In practice, we could have a single random oracle $H$ and define $H_i(input) = H(i, input)$.

Setup. Our protocol sets up pp and keys by the algorithms on Figure 7. We have $pp = (q, G, G_1, G_2, G_3, G_4)$ 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 $sk_c = x, pk_c = xG_1$ (denoted $pk_c = X$) and $sk_b = y, pk_b = yG_2$ (denoted $pk_b = Y$) respectively.

### Figure 7: Setup and key generation.

**Setup(1)**
1. Generate $\mathbb{G}$ of order $q$
2. Select random generators $G_1, \ldots, G_4$
3. $pp = (q, G, G_1, \ldots, G_4)$
4. return $pp$

**Client.KeyGen(pp)**
5. $pp = (q, G, G_1, \ldots, G_4)$
6. $x \leftarrow \mathcal{R}_{\mathbb{Z}_q}$
7. $X \leftarrow xG_1$
8. $(sk_c, pk_c) \leftarrow (x, X)$
9. return $(sk_c, pk_c)$

**Server.KeyGen(pp)**
10: $pp = (q, G, G_1, \ldots, G_4)$
11: $y \leftarrow \mathcal{R}_{\mathbb{Z}_q}$
12: $Y \leftarrow yG_2$
13: $(sk_b, pk_b) \leftarrow (y, Y)$
14: return $(sk_b, pk_b)$

ClientQuery$(pp, sk_c, pk_c)$
1: $(q, G, G_1, G_2, G_3, G_4) \leftarrow pp$
2: $x \leftarrow sk_c; X \leftarrow xG_1$
3: $r \leftarrow \mathcal{R}_{\mathbb{Z}_q}$
4: $\delta \leftarrow \mathcal{R}_\mathbb{G}$
5: $T \leftarrow \delta \cdot (X + rG_1 + G_4)$
6: $\pi_c \leftarrow \Pi_{\text{REPS.Prove}}(pp, X, T, \delta, r)$
7: query $\leftarrow (T, \pi_c)$
8: $st \leftarrow (pp, pk_c, r, \delta, T)$
9: return $(query, st)$

ClientFinal$(st, resp)$
1: $(pp, pk_c, r, \delta, T) \leftarrow st$
2: $(q, G, G_1, G_2, G_3, G_4) \leftarrow pp$
3: $Y \leftarrow pk_c; (x, \pi_c) \leftarrow resp$
4: if not $\Pi_{\text{REPS.Verify}}(pp, Y, T, s, \pi_c)$
5: return $\bot$
6: $\sigma = \frac{1}{\sqrt{q}}(xG_1 + rG_1 + G_4)$
7: return $\sigma$

ServerIssue$(pp, sk_b, pk_b, query)$
1: $(q, G, G_1, G_2, G_3, G_4) \leftarrow pp$
2: $(T, \pi_c) \leftarrow query$
3: $y \leftarrow sk_b; X \leftarrow pk_c$
4: if not $\Pi_{\text{REPS.Verify}}(pp, X, T, \pi_c)$
5: return $\bot$
6: $S = \frac{1}{\sqrt{q}}(xG_1 + rG_1 + G_4)$
7: $\pi_s \leftarrow \Pi_{\text{UNLINK.Prove}}(pp, pk_b, S, T, s, y)$
8: resp $\leftarrow (s, \pi_s)$
9: return resp

### Figure 8: Token issuance.

signature on $sk_c$ in a group which does not necessarily have a pairing (hence without public verifiability) [27, 28].

First, the client commits to its 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 $\delta$) 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 server also proves knowledge of $X$ during this proof $\pi_c$. This gives RUF-security at the same time.

After receiving $T$ and verifying $\pi_c$, the server (with secret key $y$) samples random $s$ and computes $S$. These values are specifically constructed so that the client can compute $\sigma = \frac{1}{\sqrt{q}}(xG_1 + rG_1 + G_4)$ using the values $s, \delta, T$ 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 $s, S, \pi_s$, verifying that $S$ was computed properly and computes $\sigma$.

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

$$x \cdot \frac{G_1}{G_1} + r \cdot \frac{0}{G_3} - \frac{1}{\delta} \cdot \frac{T}{G_4} = T$$

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 $\pi_s$**. We present in Figure 10 how the $\Pi_{\text{UNLINK}}$ (see Appendix B) proof will be used in our protocol with its notations to make a proof $\pi_s$ issued by the server. We use it to prove the knowledge of $y$ satisfying

$$y \cdot \frac{G_2}{S} = \left( \begin{array}{c} X \\ T - s \end{array} \right)$$
The protocol uses a random oracle which is denoted by $\sigma$. Pedersen commitment and in the end we reveal part of the BBS.

In our protocol, we use a multiplicative mask $\sigma$ known in the client sending their token $\sigma\Pi$ active protocol $\Pi$ Redeem protocol.

Full version, May 8th, 2024 F. Betül Durak, Laurane Marco, Abdullah Talayhan, and Serge Vaudenay

![Figure 9: \(\Pi_{\text{REPS}}\): Client proof $\pi_c$.](image)

![Figure 9: \(\Pi_{\text{REPS}}\): Client proof $\pi_c$.](image)

![Figure 10: \(\Pi_{\text{DLEQ}}\): Server proof $\pi_s$.](image)

![Figure 10: \(\Pi_{\text{DLEQ}}\): Server proof $\pi_s$.](image)

![Figure 11: Redemption.](image)

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

**Redeem protocol.** Suppose a token $\sigma$ was issued to a client with public key $X = xG_1$. To ensure non-transferability, during redemption we require the client to prove the knowledge of the 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 $\Pi = ZKPoK(x, s, r: \sigma = \frac{r}{2^s}(xG_1 + rG_2 + yG_3))$ by revealing $\sigma' = y\sigma$. We construct in Figure 11 such a protocol $\Pi$, 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 $\sigma$ to the server to be verified. With some rearranging, the client wants to convince the server that it knows $(x, s, r)$ such that $\sigma' - G_4 = xG_1 + rG_3 - s\sigma$. II is based on the generalized Schnorr protocol [29].

**Discussion.** In our protocol, we use a multiplicative mask $\delta$ on the Pedersen commitment and in the end we reveal part of the BBS signature, i.e. the group element $\sigma$. Thanks to the multiplicative masking, $\sigma$ 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 $\sigma$ 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 $\circ$ the pairing operation. We focus on these two operations to estimate the complexity. 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 KVAs) 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![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 $(2n^2 + 4n^2 + 4n)$ for the client and $(1n^2 + 1n^2 + 4n)$ for the server, and redeem takes $(1n^2 + 4n)$ for the client and $(1n^2 + 2n)$ for the server.]
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 1: Complexity comparison

<table>
<thead>
<tr>
<th></th>
<th>Issuance</th>
<th>Redemption</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td></td>
</tr>
<tr>
<td>Our NTAT</td>
<td>11×</td>
<td>8×</td>
<td>28×</td>
</tr>
<tr>
<td>Our Pairing NTAT</td>
<td>7×, 3×</td>
<td>6×, 4×</td>
<td>21×, 4×</td>
</tr>
<tr>
<td>CHAC [8]</td>
<td>5×</td>
<td>5×, 3×</td>
<td>10×</td>
</tr>
<tr>
<td>CHAC(^{+}) [8]</td>
<td>3×</td>
<td>5×, 4×</td>
<td>13×</td>
</tr>
<tr>
<td>BBDT [21]</td>
<td>3×, 2×</td>
<td>11×, 10×</td>
<td>17×</td>
</tr>
<tr>
<td>Pairing BBDT [21]</td>
<td>5×, 2×</td>
<td>11×</td>
<td>16×, 2×</td>
</tr>
<tr>
<td>PP [1]</td>
<td>2×</td>
<td>7×</td>
<td>10×</td>
</tr>
<tr>
<td>ATHM [23]</td>
<td>17×</td>
<td>11×</td>
<td>29×</td>
</tr>
</tbody>
</table>

### Table 2: Communication properties comparison

We denote \(\mathcal{G}\) the group elements, by \(q\) the scalars, by \(b\) the extra bits, and by \(\ell_1\) and \(\ell_2\) the length of \(p\) and \(H\). Note that BBDT does not specify explicit proofs for the server and client during issuance hence we denote their generic cost by \(\Pi_{\text{REP2}}, \Pi_{\text{DLEQ}}\).

<table>
<thead>
<tr>
<th></th>
<th>Issuance</th>
<th>Redemption</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td></td>
</tr>
<tr>
<td>Our NTAT</td>
<td>2(\mathcal{G}) + 7q</td>
<td>1(\mathcal{G}) + 4q +</td>
<td>3(\mathcal{G}) + 14q +</td>
</tr>
<tr>
<td>Our Pairing NTAT</td>
<td>2(\mathcal{G}) + 5q</td>
<td>1(\mathcal{G}) + 4q +</td>
<td>3(\mathcal{G}) + 9q +</td>
</tr>
<tr>
<td>U-Prove [11]</td>
<td>3(\mathcal{G}) + 2q</td>
<td>3(\mathcal{G}) + 5q</td>
<td>6(\mathcal{G}) + 7q</td>
</tr>
<tr>
<td>CHAC [8]</td>
<td>8(\mathcal{G}) + 1q</td>
<td>6(\mathcal{G}) + 1q</td>
<td>16(\mathcal{G}) + 2q</td>
</tr>
<tr>
<td>BBDT [21]</td>
<td>2(\mathcal{G}) + 2q</td>
<td>3(\mathcal{G}) + 8q +</td>
<td>5(\mathcal{G}) + 10q +</td>
</tr>
<tr>
<td>PP [21]</td>
<td>2(\mathcal{G}) + 1q +</td>
<td>1(\mathcal{G}) + 1b</td>
<td>3(\mathcal{G}) + 1q + 2b</td>
</tr>
<tr>
<td>ATHM [23]</td>
<td>4(\mathcal{G}) + 8q</td>
<td>2(\mathcal{G}) + 1q</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3: Functionality and properties comparison

<table>
<thead>
<tr>
<th></th>
<th>Type</th>
<th>Non-transferable</th>
<th>OMUF</th>
<th>Secretly redeem</th>
<th>Stateless issue</th>
<th>Single or Multi show</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our NTAT</td>
<td>AT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single</td>
</tr>
<tr>
<td>Our Pairing NTAT</td>
<td>AT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single</td>
</tr>
<tr>
<td>CHAC [8]</td>
<td>KVAC</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>Multi</td>
</tr>
<tr>
<td>CHAC(^{+}) [8]</td>
<td>KVAC</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>Multi</td>
</tr>
<tr>
<td>BBDT [21]</td>
<td>KVAC</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>Both</td>
</tr>
<tr>
<td>Pairing BBDT [21]</td>
<td>KVAC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Both</td>
</tr>
<tr>
<td>PP [21]</td>
<td>AT</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single</td>
</tr>
<tr>
<td>ATHM [23]</td>
<td>AT</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th></th>
<th>Issuance</th>
<th>Redemption</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td></td>
</tr>
<tr>
<td>NTAT scheme</td>
<td>651.94µs</td>
<td>517.31µs</td>
<td>191.24µs</td>
</tr>
<tr>
<td>Pairing NTAT</td>
<td>5.47ms</td>
<td>3.13ms</td>
<td>985.93µs</td>
</tr>
<tr>
<td>U-Prove</td>
<td>44.25µs</td>
<td>149.68µs</td>
<td>99.02µs</td>
</tr>
<tr>
<td>CHAC</td>
<td>1.28ms</td>
<td>10.74ms</td>
<td>4.02ms</td>
</tr>
</tbody>
</table>

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

\[
\text{Adv}_Q^{\text{OMUF}} \leq \frac{n m_H}{q} + \frac{n(n+5)}{2} \text{Adv}_Q^{\text{DLOG}} + \text{Adv}_Q^{\text{OMUF}} + \frac{1}{q}
\]

where \(q\) is the group order, \(n\) is the total number of ObserveTok queries, \(\text{Adv}_Q^{\text{DLOG}}\) (respectively \(\text{Adv}_Q^{\text{OMUF}}\)) 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 \(\pi_c\): an oracle to evaluate \(H_2\) by using lazy sampling. There are random oracles \(H_1\) and \(H_2\), too. The total number of oracle calls to \(H_1\) by \(\mathcal{A}\) is denoted by \(m_{H_1}\).

\(\Gamma^1\): We let \(((s_k_c, p_k_c), (\sigma_i, \omega_i))_{i \in \{1, \ldots, q\}}\) be the output of \(\mathcal{A}\), and we denote \(s_k = x, s_k = y, p_k = X, \) and \(\omega_i = (r_i, s_i)\). We further denote \(\sigma'_i = xG_i + r_iG_1 + G_k - s_i\sigma_i\). Our first step is to reduce OMUF to a game \(\Gamma^1\) in which the steps Step 7 and following are replaced by the following winning conditions: \(X = xG_1\) and \(\sigma_1 = \frac{1}{yG_1} (xG_1 + r_iG_1 + G_k)\) 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 \(\sigma'_1 = yG\) 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 \(\sigma'_1\) and passes thanks to the perfect completeness of the protocol. Clearly, those changes do not modify the outcome of the game.

\[
\operatorname{Adv}^{\text{OMUF}} = \operatorname{Adv}^{\Gamma^1}.
\]

\(\Gamma^2\): We define a DDH oracle specialized for the \(Y = yG_2\) key on input \((A, B)\), the \(\operatorname{ODDH}_y\) oracle answers whether \(yA = B\).

\[
\operatorname{ODDH}_y(A, B) = \begin{cases} 1 \text{ return } 1_{yA = B} \end{cases}
\]

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

We reduce \(\Gamma^1\) to another game \(\Gamma^2\) where we replace the \(\operatorname{ODEmdeem}\) oracle using an \(\operatorname{ODDH}_y\) oracle and transform \(\mathcal{A}\) accordingly. Note that the random oracle \(H_2\) is now only queried by \(\mathcal{A}\). So, we can change \(\mathcal{A}\) to simulate \(H_2\) by lazy sampling and make \(H_3\) disappear from the game.

We have

\[
\operatorname{Adv}^{\Gamma^1} = \operatorname{Adv}^{\Gamma^2}.
\]

\(\Gamma^3\): In the next step, we replace in a game \(\Gamma^3\) the \(\text{IssueTok}\) oracle by another oracle which only returns \(s_i'\) and \(S_j\) (for the \(j^{th}\) query) but not \(\pi_c\). This can be done by adding in \(\mathcal{A}\) a simulator for the missing proof \(\pi_c\). At the same time, we remove the random oracle calls to \(H_2\) which appear in \(\pi_c\) (namely, \(H_2(pp, Y, S, T) = s_j, \text{comm}_j', \text{comm}_j\)) as only \(\mathcal{A}\) is using it, so can simulate it.

To simulate \(\pi_c\), the adversary picks \(ch\) and \(\text{resp}\) at random then computes \(\text{comm}_j', \text{comm}_j\). Due to the distribution of \(ch\) and \(\text{resp}\), 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 \(\frac{m_{H_2}}{q}\). If it happens, the simulation fails. Otherwise, the adversary can program the query to \(H_2\) in its own simulation for verification to return \(\text{ch}\). By doing this over all the \(n\) issuance queries, we obtain

\[
\operatorname{Adv}^{\Gamma^3} \leq \operatorname{Adv}^{\Gamma^4} + \frac{m_{H_2}}{q}.
\]

\(\Gamma^4\): Next, we add in a game \(\Gamma^4\) the extra winning condition that no collision on the output \(s_i'\) of the \(\text{IssueTok}\) oracle occurs. The advantage loss is bounded by \(\frac{n(m-1)}{2q}\) where \(n = \sum q_i[p_k_c]\) is the total number of \(\text{IssueTok}\) queries. In the new game, we further replace the sampling method of \(s_i'\) by a uniform sampling in \(Z_q\) and add the extra winning condition that \(s_j' \neq -y\) for every \(j\). The advantage loss is bounded by \(\frac{n}{q}\).

\[
\operatorname{Adv}^{\Gamma^3} \leq \operatorname{Adv}^{\Gamma^4} + \frac{n(n+1)}{2q}.
\]

\(\Gamma^5\): In \(\Gamma^5\), we first change \(\mathcal{A}\) so that it does not query \(\text{IssueTok}\) 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 \(\text{IssueTok}\) oracle that \(H_2(pp, X, T, \text{comm}_1, \text{comm}_2)\) has been queried before in the game. When this is not the case, \(ch = ch'\) with probability \(1/q\). We fully write the \(\Gamma^5\) game in Figure 12.

\[
\operatorname{Adv}^{\Gamma^3} \leq \operatorname{Adv}^{\Gamma^5} + \frac{n}{q}.
\]

**Figure 12:** \(\Gamma^5\) game for OMUF security.

**AGM:** In the algebraic group model, \(\mathcal{A}\) must provide an expression of every group element it receives (namely, the queries \(X_j\) and \(T_j\) to \(\text{IssueTok}\), the queries \(\text{comm}_1\) and \(\text{comm}_2\) to \(H_1\), and the final output \(X\) and \(s_i\)) as a linear combination of the group elements it received (namely, \(G_1, \ldots, G_4, Y\), and the \(S_i\) returned by \(\text{IssueTok}\)). Hence, each group element \(Z\) which is produced by the adversary comes with a vector \(\text{vec}_Z = (\alpha_1, \ldots, \alpha_4, b_1, b_2, b_3, c_1, \ldots, c_{n}) \in \mathbb{Z}_q^{n+5}\) s.t.

\[
Z = a_1G_1 + \cdots + a_4G_4 + bY + c_1S_1 + \cdots + c_nS_n.
\]

We write base = \(\left(G_1, G_2, G_3, Y, S_1, \ldots, S_n\right)\) with a scalar product \(Z = (\text{vec}_Z, \text{base})\).

In our protocol, given that \(S_j = \frac{1}{yG_1} T_j\), the adversary can form a 4-dimensional vector \(\text{vec'}_Z = (\vec{y}, \vec{y'}, \vec{y''}, \vec{y'''})\) where coefficients are rational functions in terms of a variable \(\vec{y}\) and s.t. \(Z\) is equal to \(\text{vec'}_Z, \text{base'}\) after evaluation at \(\vec{y} = y\) with base = \(\left(G_1, \ldots, G_4\right)\). We define

\[
\phi(a_1, \ldots, c_n) = (a_1, a_2 + bY, a_3, a_4) + \frac{c_1}{\vec{y}} \text{vec'}_T + \cdots + \frac{c_n}{\vec{y}} \text{vec'}_n\text{ by recursion with vec'}_Z = \phi(\text{vec}_Z).
\]
The \( s'_j \) are poles of the rational functions which are randomly selected by OlsuseTok. Recall that they must be pairwise different in the winning cases. Let \( Q(y) = (y + s'_1) \cdots (y + s'_n) \). By induction on \( n \), it follows that \( Q(y) \cdot \text{vec}'(Z) \) always has polynomial coefficients, with degrees bounded by \( n + 1 \).

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

If the adversary finds a vector \( v \) such that \( \langle \phi, v \rangle = 0 \), then either [case1] \( v = 0 \), or [case2] \( v \neq 0 \) (and thus \( \phi(v) \neq 0 \) thanks to \( \phi \) being injective) and \( y \) is a root of \( \langle \phi, v \rangle \). 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 \( \langle \phi, v \rangle = 0 \) 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 \( P(y) = Q(y) \cdot \phi(v) \). \( P \) is a vector of polynomials such that \( \langle P(y), base' \rangle = 0 \) and \( P \neq 0 \). The adversary tries to solve \( P(y) = 0 \) 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 \( \langle P(y), base' \rangle = 0 \) as a non-trivial relation over \( base' \).

\( \Gamma^6 \): The verification of \( r_2 \) consists in checking that
\[
\begin{align*}
\text{comm}_1 &= \text{resp}_1 \cdot \text{ch}_1 \cdot X_j \\
\text{comm}_2 &= \text{resp}_2 \cdot \text{ch}_2 \cdot T_j - \text{ch}_3 \cdot X_j,
\end{align*}
\]
with \( \text{resp}_1 = \text{ch}_1 \cdot \text{vec}_1 \cdot \text{ch}_1 \cdot \text{vec}_1 \). We obtain two equations of interest (for instance, the first equation is with \( v = \text{vec}_{\text{comm}_1} - \text{resp}_1 \cdot \text{vec}_1 \cdot \text{ch}_1 \cdot \text{vec}_1 \) and apply the reduction. Taking these equations of interest, when both are in [case1] we have
\[
\text{vec}_{\text{comm}_1} = \text{vec}_1 \cdot \text{ch}_1 \cdot \text{vec}_1
\]
\( \text{vec}_{\text{comm}_2} = \text{vec}_1 \cdot \text{ch}_1 \cdot \text{vec}_1 \).

We wonder for how many \( \gamma \) values there exist some scalars \( \text{resp}_1, \text{resp}_2, \text{resp}_3 \) satisfying the two equations. Clearly, if either \( \text{vec}_1 \) is not proportional to \( \text{vec}_2 \) or \( \text{vec}_3 \) is not in the linear span of \( \text{vec}_1 \) and \( \text{vec}_2 \), the number of possible \( \gamma \) values is equal to 0 and verification fails with probability bounded by \( q^{-1} \).

We reduce to a game \( \Gamma^6 \) where for each OlsuseTok query, either \( \text{vec}_2 \) is of form \( \text{vec}_1 \cdot \text{ch}_1 \cdot \text{vec}_1 \), or the oracle returns \( \perp \) or \( \mathcal{A} \) returns a polynomial \( P \) such that \( \langle P(y), base' \rangle = 0 \) and \( P \neq 0 \) [case2]. We have
\[
\text{Adv}^\mathcal{A}_6 \leq \text{Adv}^\mathcal{A}_6 + \frac{n}{q}
\]
\( \Gamma^7 \): In \( \Gamma^6 \), we can in [case1] extract \( X_j, \delta_j \) and \( r_j \) such that \( X_j = x_j \cdot G_1 \) and \( T_j = \delta_j (X + r_j \cdot G_1 + G_4) \) 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 \( \Gamma^7 \) (Figure 13) in those cases. We also define a companion adversary \( \mathcal{A}' \) who does the same but aborts in the winning cases of \( \Gamma^7 \) and rather focuses on the leftover case where \( \langle P(y), base' \rangle = 0 \).

It is such that \( \langle P(y), base' \rangle = 0 \). 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 \( \langle w, base' \rangle = 0 \). This solves the discrete logarithm game.

\[
\text{Adv}^\mathcal{A}_7 \leq \text{Adv}^\mathcal{A}_7 + \text{Adv}^{\text{DLog}}
\]

\( \Gamma^8 \): We now simulate OlsuseTok \((x, \delta, r)\) with a BBS signature oracle which upon a queried message \((x, r)\) would answer with a random \( s \) and \( \sigma = \frac{1}{q(s)} \cdot (xG_1 + rG_4 + G_4) \). The simulation consists of multiplying \( \sigma \) by \( \delta \) 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].

\textbf{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 \( Z_q - \{y\} \). The colliding probability is \( \delta = 0 \). Based on the BBS security [28, Th. 2], we obtain

\[
\text{Adv}^\mathcal{A}_8 \leq \text{Adv}^{\text{DLog}} + \text{Adv}^{\text{DLog}} + \frac{1}{q}
\]

Note that the cited BBS result holds in AGM with a pairing group. The pairing allows to simulate the \( \text{ODDH}_4 \) oracle which becomes redundant in the game. In our context, we have no pairing but we keep the \( \text{ODDH}_4 \) oracle. We easily check that the result holds in this context as well.

\section{4.2 RUF Security}

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

\textbf{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 \( B \) of complexity essentially twice the complexity of \( \mathcal{A} \), and solving the discrete logarithm problem such that we have

\[
\text{Adv}^{\text{RUF}}(\mathcal{A}) \leq m_{\text{ORedeem}} \cdot \frac{m_H}{q^{2\epsilon^2} + \frac{m_H(m_H+1)}{2q^2} + \sqrt{4m_H \cdot \text{Adv}^{\text{DLog}}(B)}}
\]

where \( q \) is the group order, \( m_{\text{ORedeem}} \) is the total number of ORedeem queries, \( \text{Adv}^{\text{DLog}} \) is the advantage of an adversary solving the DLog problem with similar complexity, and \( m_H \) is the number of \( H_1 \) queries in ROM.
We get an adversary \( \mathcal{A} \) playing the RUF game. 
\( \Gamma^1 \): We change the ORedeeem oracle using the zero-knowledge simulator and the programming technique of \( H_3 \) which was already used in the proof of OMUF-secuity. We reduce to a game \( \Gamma^2 \) with no ORedeeem 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 \((v_0,v_1,v_2,\rho)\) at random, compute \( Q^r, Q^s \), then program \( H_3 \) so that comm = \( H_3(\rho, Q^r) \). This fails with probability bounded by \( \frac{m_{H_3}}{q^{2^p}} \), the probability that \((\rho, Q^s)\) was ever queried to \( H_3 \), for each of the ORedeeem sessions.

\[
\text{Adv}_{RUF}^{\Gamma^1} \leq \text{Adv}_{RUF}^{\Gamma^2} + m_{H_3} \frac{m_{H_3}}{q^{2^p}}.
\]

Furthermore, this adversary can now fully simulate OFinal and \( H_2 \). The oracles OFinal, ORedeeem, \( H_2 \), and \( H_3 \) have now been removed. 
\( \Gamma^3 \): In \( \Gamma^3 \), \( \mathcal{A} \) needs no oracle except \( H_1 \) and ORedeeem which returns \( T = \delta(X + rG_3 + G_4) \) together with \( \pi_c, \tau, \text{ and } \delta. \) Clearly, this latter oracle can be perfectly simulated and we have only \( H_1 \) remaining.

\[
\text{Adv}_{RUF}^{\Gamma^2} = \text{Adv}_{RUF}^{\Gamma^3}
\]

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

\( \Gamma^4 \): 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 \( \frac{1}{2} \). We further require \( H_1 \) to show no collision during the game as an additional condition, which normally occurs with probability bounded by \( \frac{m_{H_3}(m_{H_3} - 1)}{2q} \). Hence,

\[
\text{Adv}_{RUF}^{\Gamma^3} \leq \text{Adv}_{RUF}^{\Gamma^4} + \frac{m_{H_3}(m_{H_3} + 1)}{2q}.
\]

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

We obtain a regular unforgeability security for the Fiat-Shamir transform of a regular \( \Sigma \)-protocol. We can now use the forking lemma ([34], [35]) to obtain two successful executions with the same input \((pp, X)\), same \( \text{comm}_1 \), different \( ch \), and the corresponding different \( \text{resp}_1 \). This gives

\[
\text{resp}_1, G_1 + ch, X = \text{resp}_1', G_1 + ch', X
\]

which solves the discrete logarithm of \( X \). This defines an adversary \( B \) solving the discrete logarithm problem and such that

\[
\text{Adv}^{\Gamma^5} \leq \sqrt{2m_{H_3}}.\text{Adv}^{\text{DLog}}(B)
\]

\( \square \)

### 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

\[
\text{Adv}^{\text{UNLINK}}_{\mathcal{A}} \leq 1 \frac{1}{2} + n_q(n_{H_3} + 1) + n_{H_3} + m_{\text{ORedeeem}} n_{H_3} 2^{-\ell p},
\]

where \( q \) is the group order, \( \ell p \) is the bitlength of \( \rho, n_q \) is the total number of ORedeeem queries, \( m_{\text{ORedeeem}} \) is the total number of ORedeeem queries, \( n_{H_3} \) is the number of \( H_3 \) 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 \((\mathcal{A}_1, \mathcal{A}_2)\).

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

\[
\text{Adv}^{\text{UNLINK}} \leq \text{Adv}^{\Gamma^1} + \frac{n_q n_{H_3}}{q},
\]

where \( n_q \) is the number of request to ORedeeem. There is no \( H_3 \) oracle any more as it is fully simulated by \( \mathcal{A} \).

\( \Gamma^2 \): In the next step, we simplify the ORedeeem oracle which would only return \( \sigma_j \) and \( \sigma'_j \) at random, compute \( Q^r, Q^s \), then program \( H_3 \) so that comm = \( H_3(\rho, Q^r) \). This fails with probability bounded by \( \frac{n_q}{q} \), the probability that \((\rho, Q^s)\) was ever queried to \( H_3 \), for each of the ORedeeem sessions.

\[
\text{Adv}^{\Gamma^1} \leq \text{Adv}^{\Gamma^2} + m_{\text{ORedeeem}} n_{H_3} 2^{q},
\]

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

\( \Gamma^3 \): We notice that \( \sigma' = \frac{1}{2}(T - sS) \). We change ORedeeem so that it would compute \( \sigma_j = \frac{1}{2} S_j \) and \( \sigma'_j = \frac{1}{2} (T_j - s_j S_j) \). Note that the swap in UNLINK would need to hold on the values of \( \delta_j, T_j, S_j \) to be consistent.

\[
\text{Adv}^{\Gamma^2} = \text{Adv}^{\Gamma^3}
\]

\( \Gamma^4 \): In OFinal, \((j, \text{resp})\), we parse \( x = (s, S, \pi_x) \) and query \( j = T \) as there is no \( \pi_c \) any more. In \( \Gamma^4 \) we change OFinal to output out \( j \), by directly checking whether \( S = \frac{1}{2} x \). The difference between the two games is bounded by the soundness of \( \pi_x \), i.e. the probability that \( \mathcal{A} \) can issue some \((Y, S, T, s, \pi_x)\) such that verification passes but the value \( y \) such that \( y G_2 \) does not satisfy \( y S = T - S \).

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

\[11\] Here we use \( y \) from \( sk_x \), which comes from the honest key generation of \( pk_x \).
value of $ch = H_2(pp, Y, S, T - sS, comm_1, comm_2)$ is uniformly selected. If we let $z$ be such that $T - sS = zS$, the chances that there exists $res$ such that $\pi_3 = (ch, res)$ passes for $(Y, S, T, s)$ is $\frac{1}{q}$ if $y = z$ and $\frac{1}{q}$ 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 $\pi_3$ is $\frac{1}{q}$. Hence, the soundness advantage is bounded by $\frac{m_H \cdot 1}{q}$. We deduce

$$\text{Adv}^{\text{Sp}} \leq \text{Adv}^{\text{F}} + \frac{m_H \cdot 1}{q}$$

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

$$\text{Adv}^{\text{Sp}} \leq \text{Adv}^{\text{F}} + \frac{n\cdot r}{q}$$

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

$$\Gamma^5(\mathcal{A}_1)$$

1: Setup$(1^\lambda) \rightarrow pp$
2: $(q_1, G_1, G_2, G_3, \ldots, G_n) \leftarrow pp$
3: $y \leftarrow \mathcal{U}_{\mathbb{Z}_q}$
4: $Y \leftarrow yG_2$
5: $Q_{\text{query}}, G_{\text{compare}}, \text{oracle} \rightarrow \emptyset$
6: $i \leftarrow 0, b \leftarrow \{0, 1\}$
7: $(j_0, j_1, \text{state}) \leftarrow \mathcal{A}_1(\text{oracle}, pp, y, Y)$
8: if $(j_0 = j_1) \not\in Q_{\text{final}} = Q_{\text{Redeem}}$ then abort
9: if $out_{j_0} = 0$ or $out_{j_1} = 0$ then abort
10: if $b = 1$ then
11: swap $(s_{j_0}, s_{j_1}, T_b)$ and $(s_{j_1}, s_{j_0}, T_b)$
12: $b' \leftarrow \mathcal{A}_1^{\text{oracle}}(\text{state})$
13: return $T_{b'}$

OClientKeyGen() 14: increment $i$
15: $x_i \leftarrow \mathcal{U}_{\mathbb{Z}_q}$
16: $X_i \leftarrow x_iG_1$
17: return $(x_i, X_i)$

ORequest$(j, i, \text{usr})$
18: if $j \in Q_{\text{query}}$, then abort
19: insert $j$ in $Q_{\text{query}}$
20: $(s_j, X_j) \leftarrow (s_{k_{\text{usr}}} \cdot pk_{\text{usr}}, x_j)$
21: $T_j \leftarrow \emptyset - \{0\}$
22: return $T_j$

Final$(j, x, S)$
23: if $i \in Q_{\text{final}}$ or $j \notin Q_{\text{query}}$ then abort
24: insert $j$ in $Q_{\text{final}}$
25: return $(x_j, S_j) \leftarrow (x_j, S)$
26: $out_{j0} \leftarrow \mathcal{U}_{\mathbb{Z}_q}$
27: return out$j_0$

ORedeem$(j)$ 28: if $j \notin Q_{\text{final}}$ then abort
29: insert $j$ in $Q_{\text{redem}}$
30: if $out_{j} = 0$ then return
31: $d_j \leftarrow \mathcal{U}_{\mathbb{Z}_q}$
32: $\sigma_j \leftarrow (T_j - s_jS_j)$
33: $\sigma_j' \leftarrow \frac{1}{\sqrt{2H}}(T_j - s_jS_j)$
34: return $(\sigma_j, \sigma_j')$ 29

Figure 14: Game $\Gamma^5$ in unlikeliness.

$\Gamma^5$ unlikeliness. We show what is left in $\Gamma^5$ on Figure 14. When ORedeem goes on, we have $out_{j_0} = 1$ so $S_j = \frac{1}{\sqrt{2H}} T_j$. This implies $T_j - s_jS_j = yS_j$ hence $\sigma_j' = y\sigma_j$. The random selection of $\delta_j$ makes $\sigma_j$ independent from $(s_j, S_j, T_j)$. Hence, it is clear that no information is revealed about $h$. Therefore, we get $\text{Adv}^{\text{S}} = \frac{1}{2}$.

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 $K(x) = \frac{2}{x(x-1)/q}$ and

$$\text{Adv}^{\text{SOUND}} \leq \frac{4m_H^2}{2H}$$

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

For instance, a transfer of probability $p_{\text{rand}} = 100\%$ would yield $K(1) = 2$ 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 $\Sigma$ 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 ORedeem(sid, $\sigma'$, comm) following a ORedeem(sid, pp, $\sigma$), which answer whether $\sigma' = y\sigma$, these calls can be perfectly simulated by $\mathcal{E}$. To simulate the remaining ORedeem(sid, $\sigma'$, comm), we just make $\mathcal{E}$ query ORedeem with a dummy fresh sid` to extract the ODDH 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 rewrite $\mathcal{A}_2$.

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 ORedeem(sid*, $\sigma'$, 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 $\sigma$, $\sigma'$, comm be the common values in the sessions sid* of ORedeem during the two executions. The two executions give $(c, v_0, v_1, v_2, p, Q)$ and $(\epsilon', \epsilon''_0, \epsilon''_1, \epsilon''_2, \epsilon''_3)$ such that $H_3(p, Q) = H_3(\epsilon', \epsilon''_3)$ and

$$Q = v_0G_1 + v_1G_2 + v_2 - c\sigma' - G_4$$

$$Q' = \epsilon''_0G_1 + \epsilon''_1G_2 + \epsilon''_2 - c\sigma' - G_4$$

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

Next, if $Q + Q' \neq H_3(\epsilon', \epsilon''_3$, $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{E})$ found a collision on $H_3$ during $2m_H$ calls and aborts. In this case, the SOUND game returns 2.

This happens with complexity bounded by $\frac{4m_H^2}{2H}$. Otherwise, we have $Q = Q'$ and $c \neq \epsilon'$. Hence, $\mathcal{E}$ can set $x = \frac{v_0 - v_1}{\epsilon''_0 - \epsilon''_1}$, $r = \frac{v_2 - \epsilon''_3}{\epsilon''_2 - \epsilon''_3}$, and $s = \frac{\epsilon''_3 - \epsilon''_0}{\epsilon''_2 - \epsilon''_3}$ to get $\sigma' = xG_1 + rG_2 + sG_4 - \sigma$. 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 $\mathcal{E}$ be the event that out$id = 1$ (i.e., 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 $\mathcal{A}_2$(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 $\epsilon (\text{rnd}^*) = \Pr[E(\text{rnd}^*)]$ 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
\[ \Pr[V'] \geq \epsilon(\text{rnd'}) \geq (\epsilon(\text{rnd'}) - 1) / q. \]
The average of \( \epsilon(\text{rnd'}) \) over \( \text{rnd'} \) is \( \Pr[V|\text{rnd}] = p_{\text{rnd}} \). We apply the Jensen inequality on \( x \mapsto x - x^2 / 2 \) to deduce \( \Pr[V'] \geq p_{\text{rnd}} (p_{\text{rnd}} - 1) / q \). The average number of rounds is bounded by 1/\( \Pr[V'] \). Since we have two executions of \( A_2 \) per round, we obtain \( K(x) = x/2 - 1/q \).

\section{UNIQ Security}

\textbf{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 \( A \), there exists an adversary \( B \) of complexity essentially similar to \( A \), and solving the discrete logarithm problem such that we have
\[ \text{Adv}^{\text{UNIQ}}(A) \leq \text{Adv}^{\text{DLog}}(B) + 5 \]
where \( q \) is the group order and \( \text{Adv}^{\text{DLog}} \) is the advantage of an adversary solving the DLog problem with similar complexity.

\textbf{Proof.} We consider an adversary \( A \) playing the UNIQ game. We define a discrete logarithm solver as follows.
\[ B(q, G, Z) = \frac{\text{Adv}^{\text{UNIQ}}(A) - 5}{q} \]
Except in those cases, \( B \) provides to \( A \) some pp with correct distribution. Hence, \( A \) gets pp and outputs some values making UNIQ win with probability \( \text{Adv}^{\text{UNIQ}}(A) - 5/q \).

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

Let \( z \) be such that \( Z = ZG_1 + y \) be such that \( G_1 = y'G \) for \( i = 1, \ldots, 4 \). The adversary \( A \) has information on the \( y_i \) but \( G_1 \) is independent from \( y_i \). We have \( b = ((y + s')x - (y + s)x')/aG_1 + ((y + s')r - (y + s)r')/bG_3 + (s' - s')/bG_4 \). 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 \( \Pr[b = 0] = q \). Hence, \( B \) succeeds with probability at least \( \text{Adv}^{\text{UNIQ}}(A) - 5/q \).

\section{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 leaching.

\textbf{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.

\section{References}


The protocol is in Figure 17.

**Figure 17: **Π⁺LEQ: Server proof \( \pi_s \) with additive masking.

**Figure 18: **Redemption with additive masking.

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

\[
G_4 + tG_5 = -sxG_1 - rG_3 + s'G_4 + A + \mu B
\]

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 \( \Pi_{REP3} \), and 6 in \( \Pi_{LEQ} \)). The computation of \( s\sigma \) and \( r\sigma'G_3 \) in ClientFinal being already done for the verification of \( \pi_s \) and 10 for the server (2 in ServerIssue, 5 in \( \Pi_{REP3} \), and 3 in \( \Pi_{LEQ} \)). 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 \( \pi_s \) is replaced by the verification of \( e(S, Y + sG_2) = e(T + r\sigma'G_3, G_2) \). 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]).

\[
\Pi_{\text{DLEQ}} = \text{NIZK} \{ (x) : X = xG \land W = xT \}
\]

\[
\Pi_{\text{REP}} = \text{NIZK} \{ (x_1, x_2, \ldots, x_n) : X = x_1G_1 + x_2G_2 + \cdots + x_nG_n \}
\]

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

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

**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 consists 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 \( P_2^* \) 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 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 [33]. 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 Yílek [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: confirmers signatures and designated verifier signatures [58, 59].

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 \( π_2 \) 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 \( (T, π_2) \) 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 \( e(S, Y + sG_2) = e(\frac{1}{\sqrt{3}} T, (y+s)G_2) = e(T, G_2) \). If this check is passed, they compute \( σ \) as before, otherwise they return \( ⊥ \).

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 \( σ' = σy \) on the server side by a pairing check \( e(σ, Y) = e(σ', G_2) \) 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.
We can easily add attributes and \( n \) complexity cost is \( n \). For instance, there is even an extra blinding unknown in the extra commitment. For type-2 attributes, this is enough. However, type-3 and type-4 require to prove that the attribute is the logarithm of an extra provided group element, and (type-4) same as in an extra commitment to this attribute which is known.

Private attributes can be sorted in three types: (type-2) attributes which are present but must remain hidden. They require \( n \) extra multiplications for the server during redemption. (type-3) attributes which are present but must remain hidden, and (type-4) 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, \( sk_c = x \) could be considered as a type-3 attribute.

The redemption protocol changes naturally. Type-i attributes are revealed. They require \( n_{pq} \) extra multiplications for the server, and \( n_{pq} \) communications. Other attributes generate 1 extra multiplications 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 \( n_d + 2n_c + 4n_q \) extra multiplications for the client, \( n_d + n_p + 2n_c + 3n_q \) extra multiplications for the server, and \( n_{pq} \) 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]. 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 $\pi_c$ 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].

We believe that such approach is implicitly made by U-Prove [11]. To remove $sk_c$ from issuance, we replace the proof $\pi_c$ in the protocol by a proof of knowledge $\Pi_{REP2}$ of $\delta$ 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 $F^8$ as the adversary must provide $sk_c$ to the Oissueto 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 $3G + 10q + (\ell_p + \ell_q)b$. 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 $R(sk'_c)$. 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 $(pk_c, \pi, C)$ 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 $\Pi_{KeyGen}$ proof. We want to prove the following relation: the existence of $x$ such that $x \cdot G_1 = X \land R(C - H(x))$, where $R$ 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 $X = x \cdot G_1$ we can use the Schnorr sigma protocol [29]. We can reasonably assume the existence of a sigma protocol for $R$. 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 $(\Pi_{KeyGen}, \text{Prove}, \Pi_{KeyGen}, \text{Verify})$ protocol.
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.

<table>
<thead>
<tr>
<th></th>
<th>Issuance</th>
<th>Redemption</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NTAT scheme</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTAT with attributes</td>
<td>((11 + n_1 + n_2 + 2n_3 + 3n_4) \times 8)</td>
<td>((4 + n_b + 2n_c + 4n_d) \times 5)</td>
<td>((28 + 2n_1 + 2n_2 + 4n_3 + 6n_4 + n_a + 2n_b + 4n_c + 7n_d) \times 5)</td>
</tr>
<tr>
<td>BBDT [21]</td>
<td>((n_2 + 2) \times \Pi_{n_2 \rightarrow \pi_2} . \text{Prove} + \Pi_{\text{OMUF}} . \text{Verify} + \Pi_{\text{USEQ}} . \text{Prove})</td>
<td>((n_b + 10) \times (n_b + 9))</td>
<td>((n_2 + 2n_b + 14) \times (n_2 + 2n_b + 14 + \Pi_{\text{OMUF}}))</td>
</tr>
<tr>
<td>U-Prove [11]</td>
<td>(9\times 3\times (3 + n_b + 4n_d)\times 5)</td>
<td>((7 + n_b + 3n_d)\times 5)</td>
<td>((22 + 2n_b + 7n_d)\times (26 + n_a)\times (15 + n_a))</td>
</tr>
<tr>
<td>CHAC [8]</td>
<td>(5\times (4 + n_1)\times 5)</td>
<td>(17\times )</td>
<td>((10 + n_a)\times e)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th></th>
<th>Issuance</th>
<th>Redemption</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NTAT scheme</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTAT with attributes</td>
<td>(2G + 7q)</td>
<td>(1G + 4q + (f_p + f_H) b)</td>
<td>(3G + 11q + (f_p + f_H) b)</td>
</tr>
<tr>
<td>BBDT [21]</td>
<td>((2 + n_1 + n_2) \times \Pi_{n_1 \rightarrow \pi_1} + \Pi_{\text{OMUF}})</td>
<td>((1 + n_d) \times (4 + n_a + n_b + n_c + 2n_d)\times q + (f_p + f_H) b)</td>
<td>((3 + n_1 + n_4 + n_d) \times (11 + n_2 + n_3 + 2n_4 + n_a + n_b + n_c + 2n_d)\times q + (f_p + f_H) b)</td>
</tr>
<tr>
<td>U-Prove [11]</td>
<td>(3G + 2q)</td>
<td>(8G + 1q)</td>
<td>(16G + 2q)</td>
</tr>
<tr>
<td>CHAC [8]</td>
<td>(2G + 2q + \Pi_{\text{OMUF}(\pi_1)} + \Pi_{\text{USEQ}})</td>
<td>(3G + \Pi_{\text{OMUF}(\pi_2) + \Pi_{\text{USEQ}}} + \Pi_{\text{USEQ}})</td>
<td>(5G + (\Pi_{\text{OMUF}(\pi_1) + \Pi_{\text{USEQ}}} + \Pi_{\text{USEQ}})</td>
</tr>
</tbody>
</table>

**E ON THE NECESSITY OF AUTHENTICATION IN NTAT**

The necessity to prove the knowledge of \( x \) in \( \pi_c \). 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 \( \pi_c \) 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 \( \sigma_1 = \frac{1}{q_{x_1}} (X_1 + r_1, G_2 + G_4) \).
2. Engage the issuance protocol with a fake public key \( X_2 = \sigma_1 \) and get \( \sigma_2 = \frac{1}{q_{x_2}} (X_2 + r_2, G_3 + G_4) \).
3. Set \( r_3 = \frac{1}{x_3 - x_2 - r_1 - r_2}, x_3 = x_3, G_1, \sigma_3 = r_3 - x_3, G_4 + \sigma_3 - \frac{1}{x_3 - x_2 - r_1 - r_2} \sigma_1, r_3 = r_3 - x_3, G_4 + \sigma_3 - \frac{1}{x_3 - x_2 - r_1 - r_2} \sigma_1, s_3 = s_2 \).

We do have \( \sigma_3 = \frac{1}{q_{x_3}} (x_3, G_1 + r_3, G_3 + G_4) \) 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 \( \pi_c \), the above attack does not work because the adversary does not know log \( x \).

Interestingly, this is not an attack on 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 \( (\log \sigma : L, r_2) \) 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**


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 with \( U \)-issuance in \( (1, 2, 3, 4, a, b, c, d) \) is the number of Type-\( i \) attributes.

The equivalent to redemption is called “presentation”. The presentation phase plays two role: authentication/integrity of shown attributes with \( U \)-issuance in \( (1, 2, 3, 4, a, b, c, d) \) is the number of Type-\( i \) attributes.

queries 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 proven information \( Pl \) in the token. \( Pl \) 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 redemption. A token may be used multiple times (including to open attributes differently) but in a linkable way. We consider it as a
### Figure 22: Issuance with trusted hardware

<table>
<thead>
<tr>
<th>Core</th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a \leftarrow s \mathbb{Z}_q^*)</td>
<td>(\text{comm}_1 \leftarrow a \cdot G_1)</td>
<td>(\text{comm}_1)</td>
</tr>
<tr>
<td></td>
<td>(r, b, c \leftarrow s \mathbb{Z}_q^<em>, \delta \leftarrow s \mathbb{Z}_q^</em>)</td>
<td>(T \leftarrow \delta \cdot (X + rG_3 + cG_1))</td>
</tr>
<tr>
<td></td>
<td>(\text{comm}_2 \leftarrow \text{comm}_1 + bG_3 + cT)</td>
<td>(\text{ch} \leftarrow H_1(\text{pp}, X, T, \text{comm}_1, \text{comm}_2))</td>
</tr>
<tr>
<td></td>
<td>(\text{resp}_1 \leftarrow a - \text{ch}.x)</td>
<td>(\text{resp}_1)</td>
</tr>
<tr>
<td></td>
<td>(\text{resp}_2 \leftarrow b - \text{ch}.r)</td>
<td>(\text{resp}_2)</td>
</tr>
<tr>
<td></td>
<td>(\text{resp}_3 \leftarrow c + \text{ch} \frac{1}{s})</td>
<td>(\text{resp}_3)</td>
</tr>
<tr>
<td></td>
<td>(T, \text{ch}, \text{resp}_1, \text{resp}_2, \text{resp}_3)</td>
<td>(\text{comm}'_1 \leftarrow \text{resp}_1.G_1 + \text{ch}.X)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{comm}'_2 \leftarrow \text{resp}_1.G_1 + \text{resp}_2.G_3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+ \text{resp}_3.T - \text{ch}.G_4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{ch} \leftarrow H_1(\text{pp}, X, T, \text{comm}'_1, \text{comm}'_2))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{if not} (\text{ch} == \text{ch}') \text{return } \bot)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(s \leftarrow s \mathbb{Z}_q - { -y }, S \leftarrow \frac{1}{y_3} T)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\pi_s \leftarrow \Pi_{\text{DLEQ}}.\text{Prove}(\text{pp}, y.G_2, S, T, s, y))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(s, S, \pi_s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{if not } \Pi_{\text{DLEQ}}.\text{Verify}(\text{pp}, Y, S, T, s, \pi_s))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{return } \bot)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\sigma \leftarrow \frac{2}{3})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>output: ((\sigma, \omega = (r, s)))</td>
</tr>
</tbody>
</table>

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 client 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, then it is the client’s turn with Client.Query, then Server.Issue and finally Client.Final. 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 \(\Gamma\) and \(\Sigma_2\) in common, in addition to \(pp\) and \(Y\). The forged tokens will have the same \(H\). A valid token is a tuple \((H, P_I, \Sigma_2, \sigma'_{c,i}, \sigma'_{r,i})\).
satisfying \( \sigma'_{c, i} \) with no \( \text{sk}_c \).

Figure 23: Redemption with trusted hardware

Prover(pp, X, T, \( \delta, r \))
1. \((q, G, G_1, G_2, G_3, G_4) \leftarrow \text{pp} \)
2. \((b, c) \leftarrow \mathbb{Z}_q \)
3. \(\text{comm}_1 \leftarrow bG_3 + cT \)
4. \(c \leftarrow H_2(\text{pp}, X, T, \text{comm}_1) \)
5. \(\text{resp}_3 \leftarrow b - \text{ch}\ mod\ q \)
6. \(\text{return} \ (\text{ch}, \text{resp}_3) \)

Figure 24: \( \Pi_{\text{REP2}} \): Client proof \( \pi_c \) with no \( \text{sk}_c \).

Layer.KeyGen(pp, sk_)
1. \(\text{pp} \leftarrow (q, G_1, G_2, G_3, G_4) \)
2. \(x \leftarrow \mathbb{Z}_q \)
3. \(X \leftarrow xG_3 \)
4. \(\text{sk}_c, \text{pp}_c \leftarrow (x, X) \)
5. \(C \leftarrow H(x) + \text{sk}_c \)
6. \(\pi \leftarrow \Pi_{\text{KeyGen}}.\text{Prove}(\text{pp}, X, X) \)
7. \(\text{return} \ (\text{sk}_c, (\text{pp}_c, \pi, C)) \)

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

satisfying \( \sigma'_{c, i} \) with no \( \text{sk}_c \).

Client.KeyGen(pp, \( \text{pp}_c \))
1. \(\text{pp} \leftarrow (q, G_1, G_2, G_3, G_4) \)
2. \(x \leftarrow \mathbb{Z}_q \)
3. \(X \leftarrow xG_3 \)
4. \(\text{sk}_c, \text{pp}_c \leftarrow (x, X) \)
5. \(C \leftarrow H(x) + \text{sk}_c \)
6. \(\pi \leftarrow \Pi_{\text{KeyGen}}.\text{Prove}(\text{pp}, X, X) \)
7. \(\text{return} \ (\text{sk}_c, (\text{pp}_c, \pi, C)) \)

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 \( \Sigma_2 \) and \( n \) pairs of \((\Sigma_{a, i}, \Sigma_{b, i})\). The malicious prover selects two possible choices \( \text{pp}_{i, b} \), \( b \in \{0, 1\} \) for each \( i \), computes \( 2n \) hashes to get \( \sigma_{a, i, b} = \text{Hash}(H, \text{pp}_{i, b}, \Sigma_2, \Sigma_{a, i}, \Sigma_{b, i}) \) (following the protocol as if \( a = 1 \) and
\[ \beta_1 = \beta_2 = 0 \), then makes a smart choice to select \( b = b_i \) for each \( i \) so that by setting \( \Pi_i = \Pi_{b_i} \) and \( \sigma_{c,i} = \sigma_{c,b_i} \), 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 \( (H, \Pi, \Sigma_0, \sigma_0, \alpha) \).

More precisely, we assume that the index \( i \) goes from \( 0 \) to \( n-1 \).

The adversary sets \( z_i = \frac{\sigma_{c,i} - \sigma_{c,0}}{\sigma_{r,i}} \) and

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

This has the property that for any selection of \( b_0, \ldots, b_{n-1} \), we have

\[ z_0 \sigma_{c,0} b_0 + \cdots + z_{n-1} \sigma_{c, n-1} b_{n-1} + z_n = b_0 + 2b_1 + \cdots + 2^{n-1} b_{n-1} \]

Then, the adversary defines

\[ \begin{pmatrix} \Sigma_{a,n} \\ \Sigma_{b,n} \end{pmatrix} = \sum_{i=0}^{n-1} z_i \begin{pmatrix} \Sigma_{a,i} \\ \Sigma_{b,i} \end{pmatrix} - z_n \begin{pmatrix} G_0 \\ \Sigma_0 \end{pmatrix} \]

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

\[ \begin{pmatrix} \Sigma_{a,n} \\ \Sigma_{b,n} \end{pmatrix} = \sum_{i=0}^{n-1} z_i \sigma_{r,i} \begin{pmatrix} Y \\ H \end{pmatrix} = \sigma_{c,n} \begin{pmatrix} G_0 \\ \Sigma_0 \end{pmatrix} \]

so an extra valid token \( (H, \Pi, \Sigma_0, \sigma_{c,n}, \sigma_{r,n}) \).

The attack above uses \( \alpha = 1 \) and \( \beta_1 = \beta_2 = 0 \) for simplicity but could also use random choices for \( \alpha, \beta_1, \beta_2 \).

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.
Figure 28: CHAC Issuance
\[ crs = (Y_1, Y_2) = (g_Y^1, g_Y^2). \] (sk, pk) = (Y_1^\alpha, (g_1, g_1^\alpha)). \) (isk, ipk) = ((F, (x_{i,j}), (g_{x_{i,j}}^2)_{i,j}), Attr = (a_i), j \in \{1, \ldots, t\}, t = 2 \\

\[ \text{core} \quad sk = Y_1^\alpha \quad \text{helper} \quad (sk, pk) = (Y_1^\alpha, (g_1, g_1^\alpha)) \]

\[ \text{server} \quad isk = (F, (x_{i,j})) \]

---

\[ \text{pick nonce} \]

\[ \text{aid} \leftarrow H(Attr, \text{nonce}) \]

\[ \text{pick } k, r \]

\[ U_1 \leftarrow g_Y^k, U_2 \leftarrow g_Y^k \]

\[ \text{Sig} \leftarrow sk \cdot H(\text{aid})^r \]

\[ w \leftarrow r/k \]

\[ U_1, U_2, \text{Sig, w} \]

\[ S_1 \leftarrow U_1^w, S_2 \leftarrow U_2^w \]

\[ Z \leftarrow \prod V, V \leftarrow \prod V_i \]

\[ \text{pick } r', K', \psi \]

\[ A' \leftarrow g_1^{r'}, B' \leftarrow pk^{r'} \]

\[ pk' \leftarrow (A', B') \]

\[ \text{Sig} \leftarrow \text{Sig}^{r'} \cdot H(\text{aid})^{k'} \]

\[ S'_1 \leftarrow S_1^{g_1^{r'}}, S'_2 \leftarrow S_2^{g_1^{r'}} \]

\[ Z' \leftarrow Z^{r'} \cdot \psi \]

\[ W_1' \leftarrow W_1^{z_{i,j}}, W_2' \leftarrow W_2^{z_{i,j}} \]

\[ V' \leftarrow V^{z_{i,j}} \]

\[ \text{msg} \leftarrow (pk', \text{Sig}', S'_1, S'_2, Z', W_1', W_2', V') \]

\[ \text{msg} \]

\[ \text{aid} \leftarrow H(Attr, \text{nonce}) \]

\[ e(S'_1, g_2) \equiv e(g_1, S'_2) \]

\[ e(S'_1, g_2) \equiv e(g_1, S'_2) \]

\[ \prod e(pk'_j, \prod ipk_{i,j}) \equiv e(Z', W_2') \]

\[ e(W_1', g_2) \equiv e(g_1, W_2') \]

\[ e(W_1', \prod_{i} H((ipk_{i,j})_{j}, v_i)) \equiv e(g_1, V') \]

---

Figure 29: CHAC Redeem
\[
\text{crs} = (Y_1, Y_2) = (g_1^a, g_2^a), \quad \text{(sk, pk)} = (Y_1^a, (g_1^a, g_2^a)), \quad \text{(isk, ipk)} = ((F, x_1, x_2), (g_1^{x_1}, g_2^{x_2}))
\]

\[
\begin{align*}
\text{client} & \quad \text{sk} = Y_1^a \\
\text{server} & \quad \text{isk} = (F, x_1, x_2)
\end{align*}
\]

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