Anonymous Tokens with Stronger Metadata Bit Hiding from Algebraic MACs

Melissa Chase[1], F. Betül Durak[1], and Serge Vaudenay[2]

[1] Microsoft Research
[2] EPFL

Abstract. On the one hand, the web needs to be secured from malicious activities such as bots or DoS attacks; on the other hand, such needs ideally should not justify services tracking people's activities on the web. Anonymous tokens provide a nice tradeoff between allowing an issuer to ensure that a user has been vetted and protecting the users' privacy. However, in some cases, whether or not a token is issued reveals a lot of information to an adversary about the strategies used to distinguish honest users from bots or attackers.

In this work, we focus on designing an anonymous token protocol between a client and an issuer (also a verifier) that enables the issuer to support its fraud detection mechanisms while preserving users' privacy. This is done by allowing the issuer to embed a hidden (from the client) metadata bit into the tokens. We first study an existing protocol from CRYPTO 2020 which is an extension of Privacy Pass from PoPETs 2018; that protocol aimed to provide support for a hidden metadata bit, but provided a somewhat restricted security notion. We demonstrate a new attack, showing that this is a weakness of the protocol, not just the definition. In particular, the metadata bit hiding is weak in the setting where the attacker can redeem some tokens and get feedback on whether the bit extraction succeeded.

We then revisit the formalism of anonymous tokens with private metadata bit, consider the more natural notion, and design a scheme which achieves it. In order to design this new secure protocol, we base our construction on algebraic MACs instead of PRFs. Our security definitions capture a realistic threat model where adversaries could, through direct feedback or side channels, learn the embedded bit when the token is redeemed. Finally, we compare our protocol with one of the CRYPTO 2020 protocols. We obtain 20% more efficient performance.

1 Introduction

There has been significant industry interest recently in anonymous tokens, including Google's Trust Tokens (TT) [25] and Cloudflare's Privacy Pass (PP) [13]. These protocols are used to transfer trust signals without compromising users' privacy. Anonymous tokens define a protocol between three types of parties: a client, an issuer, and a redeemer. The client wishes to obtain tokens from an issuer and then present them to a redeemer. The issuer determines the trustworthiness of clients and issues tokens; and the redeemer (a.k.a. the verifier) verifies the tokens. These systems are anonymous in that the token issuance and redemption are unlinkable, in the sense that the issuer and redeemer cannot tell which of the issued tokens was used in any given redemption. In the above systems, we consider an issuer and a redeemer which are controlled by the same entity, so these two parties are assumed to share a secret key.

PP was specifically designed in the context of CDNs to assess the trustworthiness of a client at the edge before the client is granted (or denied) access to a web server. In the typical use case, the client is required to solve a CAPTCHA before it accesses a web server for the first time. If the CAPTCHA is successfully solved, the client is given a set of tokens to redeem the next time it visits the web server. At subsequent visits the server checks that the user's token is valid and has not previously been used, and if so allows the client to bypass the CAPTCHA and directly access the content. This allows for a better experience for the user since they only have to complete the CAPTCHA once, even if they are accessing the webserver over Tor or a VPN. Because of the unlinkability, it does so without helping the web server to track the user across different visits.

Even though CAPTCHA is one way to detect bad actors, there are more advanced techniques to assess how trustworthy the client is, for example based on machine learning algorithms for fraud detection. Typically, such fraud detection algorithms are run on the issuer side; when these algorithms determine that a client is likely to be malicious, the issuer should refuse to issue it any tokens. However, such feedback allows a fraudulent client to improve their methods to bypass the fraud detection. Ideally the issuer would instead embed a bit (to indicate if the client is malicious or trusted) inside the token which would be hidden from the client and only recovered by the redeemer during the redemption. That way, the malicious client would not find out that its fraudulent activity has been detected until it tries to redeem the token. This would make this type of attack on the fraud detection algorithms significantly more cumbersome.

Behind anonymous tokens with private metadata bit, there are three desired security and privacy requirements: unforgeability (to prevent malicious clients from forging valid tokens), unlinkability (to prevent a malicious server and redeemer from linking the tokens they issued with those that are redeemed), and privacy of embedded metadata bit (to prevent the clients from learning immediately if they were identified as malicious actors). More specifically, the metadata bit is a covert channel between issuer and redeemer, who as described above will share a key. If this channel were allowed to convey unlimited information, unlinkability would be meaningless, so the protocols must ensure that the embedded signal is only a single bit and no more. Such covert channel with one bit is used to communicate whether or not the issued token should be accepted without revealing the decision to the client until it attempts a redemption.

The initial proposal for including a private metadata bit built directly on the PP protocol where the client picks an arbitrary message t and masks it to hide it from the issuer (i.e., the issuer blindly "signs" the t), and then the client unmasks the signature on t. This idea naturally extends to support private metadata bits with PP: the issuer would choose two PP issuing keys and generate a token under one key or the other depending on the bit it wished to encode.

Because PP is based on an oblivious pseudo random function, a token generated under either key was indistinguishable from random so the client was unable to determine the bit from looking at a correctly generated token. However, this protocol had significant weaknesses; a malicious client can easily make malformed token requests (e.g. keep using the same message t), and then tell from the responses whether the tokens issued encode the same metadata bit. This means that the attacker has to make one request for which he can predict the resulting bit (e.g. by using a genuine user device or by behaving badly enough that it will be guaranteed to be detected as fraudulent), and then it can make an incorrectly formed token request and directly tell from the (invalid) token it receives whether the attacker was attempting to issue a token with metadata bit 0 or 1. This problem essentially comes from the fact that the PP allows the client to pick the messages to be signed arbitrarily. In our design, messages are jointly generated by the client and the issuer so we can ensure they are random.

Two recent papers [16,15] and [22] aim to address this problem and to formally define and construct anonymous tokens with private metadata bit. The latter is a more generic version of the former where the protocols can accept public metadata as well as the private metadata bit. The authors in these works identify the problem as being that the tokens are deterministic; they propose new randomized protocols. These proposals address the issues above and guarantee that an adversary who can maliciously interact with the token issuing server cannot learn anything about the private metadata bits encoded in the tokens.

However, these schemes still have some counter intuitive properties. In particular, in their protocol, there are two notions for a *token validity*: "validity from verification", where the adversary gets feedback on whether the token verifies correctly, and "validity from extraction", where the adversary gets feedback on whether there is an embedded bit or not without revealing the bit (if it exists). Their definition of privacy for the metadata bit allows the adversary to learn at redemption whether a token is "valid from verification", but not following the other notion. Contrarily, the unforgeability notion is based on the existence of an embedded bit. And this is not just a property of the definition: we will show that the proposed schemes have the property that if the redemption service reveals whether a bit is embedded during the token redemption, the adversary can use only a few malicious interactions with the issuing and redemption service to learn information about the hidden bits embedded in a large batch of tokens.

This separation may make sense in some contexts, where we can guarantee that the adversary gets no feedback at all from the redemption server on whether its token was accepted and what the included bit was. In other cases however, this seems to be a nontrivial weakness. For example, in the CDN application above, the adversary will clearly get feedback on the bit if it is used to determine whether it will be allowed to access the web content. A private metadata bit protocol with this type of weakness would allow the adversary to make many attempts to bypass the fraud detection and thus acquire many tokens, then make a few redemption requests and identify exactly which of the remaining unredeemed tokens successfully avoided the fraud detection. Those tokens could then be collected and, for example, used to mount a DDOS attack.

Thus, it is clear that in at least some settings, we would like a token system which provides stronger guarantees. Moreover, identifying all sources of feedback is challenging. It seems likely that if a new primitive for anonymous tokens is released, it will at some point be used in settings where the adversary can get feedback on the encoded bit, whether or not the security definition allows for that. Thus, the best solution would be the one that provides the most natural security guarantees - an adversary interacting with an issuer and a redeemer may learn the bits encoded *in the tokens it redeems*, but nothing else.

That leaves us with the following questions: *Is such a definition efficiently realizable? What is the overhead as compared to the solutions described above?*

Our Contributions. In the rest of the paper, we begin by summarizing the private metadata bit proposals PMBT of [16,15], describing the known weakness in detail, and explaining an additional attack. Then we present our more natural security definitions. Finally, we present ATHM, a *new construction for anonymous tokens* with private metadata bit which we show satisfies our definitions. We analyze the efficiency of ATHM in section 5 and show that, surprisingly, ATHM is faster than PMBT (1.3 ms vs 1.6 ms). Finally, we show the *flexibility of our approach* by demonstrating that it extends easily to allow for tokens including *public metadata* visible to both issuer/redeemer and client.

Our Techniques. As mentioned above, the initial privacy pass protocol was based around oblivious pseudorandom functions (OPRFs). The proposals of [16,15,22], as described above, identify the problem as that OPRFs are deterministic, and attempt to address it by making the protocol randomized. However, once we recognize that we do not in fact need a deterministic function, we can ask whether it makes sense to base this primitive around OPRFs, whose defining characteristic is that they are deterministic. Moreover, the obliviousness property of OPRFs turns out to be *not* a very good fit for hiding the metadata bit.

This begs the question: *Is there a better primitive to start from if we want to encode hidden data in anonymous tokens?*

For this, we turn to authentication primitives like MACs. Since we need privacy, we look at anonymous credentials, which allow issuers to certify attributes which can later be presented unlinkably [10,3]. Specifically, we borrow from keyed verification anonymous credentials (KVAC) [6,7], where the credential issuer and verifier share a secret key, and from constructions based on algebraic MACs.

KVAC directly gives us a protocol for blindly issuing credentials, in which an issuer issues a MAC on a set of attributes, some of which are only known to the client. At a high level we can apply this as follows: in token issuance, the client chooses a random nonce, and the issuer uses blind issuance to give a MAC on a pair of messages consisting of the nonce and the hidden bit. If we use the first construction from [6], MACGGM, that gives a construction in elliptic curve groups where tokens consist of the nonce and two additional group elements, which is roughly comparable with PMBT, and only twice as long as PP. But, if we consider the blind issuance protocol from [6], it would be roughly twice as expensive as the issuance for PMBT, and 4 times the cost of PP.[3] It also does not directly provide metadata privacy as that is not a property generally considered in the credentials setting. Technically then, we are left with two questions: 1) *Can we optimize the MACGGM based issuance protocol to the point where it is competitive with PP or PMBT?* and 2) *If the client does not know one of the attributes in the credential, do the protocols still work? Can we prove that this attribute will not be leaked to the client?*

We address the first of these questions with a very careful optimization of the blind issuance protocol to get a result with comparable cost to PMBT. If we wanted to directly reduce to MAC security, we would need the request to include something from which we could extract the nonce in $\mathbb{Z}_p$ which will be the message for the MAC. Extracting messages in $\mathbb{Z}_p$ is extremely expensive.[4] Moreover, the blind issuance protocol in [6] has the client form an ElGamal encryption of the message to be signed, which would result in a client-to-server message twice as long as in PP or PMBT, even before the proofs are added. Instead, we design an optimized blind issuance protocol and prove in the generic group model that the resulting token scheme is unforgeable.

Unlinkability follows in a straightforward way from the privacy of the KVAC scheme. Privacy of the metadata bit is more challenging, as there is no analog in the KVAC context. First, we note that the blind issuance protocol works even if the client does not know one of the messages, and the verifier (the issuance server) can simply verify with both possible bit messages, and output the bit for which the MAC verifies. Intuitively, we also might hope to get privacy for the metadata bit because MACGGM has some pseudorandomness properties: the basic MAC on a message pair (m_0, m_1) is $(U, (x + ym_0 + zm_1)U)$, where U is a random group element, and (x, y, z) are the secret key. DDH then guarantees that this will look like a random pair of messages. However, proving that this satisfies the metadata bit privacy property, where the adversary can interact maliciously with issuance and redemption oracles is significantly more challenging. Here, we again prove security in the generic group model.

Related Work. Beyond the works on anonymous tokens mentioned above, the most closely related work is in anonymous credentials. While there are also works based on RSA groups (beginning with [3,4]) and based on pairings, we focus here on works that can be implemented in prime order elliptic curve groups, since those provide the best efficiency. In that setting, besides MACGGM, there is one other proposal for a MAC based anonymous credential scheme [8] which is more expensive than MACGGM.

[3] We provide a more precise analysis in Appendix C.

[4] [6] addresses this by making non-standard assumptions about the extraction properties of Fiat-Shamir.

In addition, there are several anonymous credential constructions in elliptic curve groups that take a blind-signature based approach [18,1,24]. It is not clear how to add private issuer values (like the metadata bit) in these schemes, but even if we consider the simpler setting where the bit is known to the user, these schemes have several downsides: First, token redemption is significantly more expensive (4x for [18] or [24], and 8x for [1]). In a setting like the CDN application where we will be using tokens to decrease spam/prevent DDOS attacks, we want the cost to verify tokens to be as low as possible. Secondly, they all require a multi-round issuance protocol, with two round trips between the user and issuer. This requires that the token issuance server to be stateful, and in fact if the issuer can be tricked into completing a protocol in two different ways, then his secret key will be leaked. This means that implementing a token issuer requires careful state management, including storage per client session. In a setting where there are many many clients, many of which may be untrusted or on flaky connections, this can be quite expensive.

2 PMBT: a Case Study

We begin with [16] (and its full version [15]) and investigate the shortcomings as well as a new attack. There are two main constructions in these works called Private Metadata Bit Tokens (PMBT) and CMBT. For PMBT, the authors acknowledge that Verify returning always true is not meaningful and then announced their new protocol named CMBT in the full version of their paper [15]. Before the description of protocols, we borrow the interface of anonymous tokens (AT) with private metadata bit and its security and privacy requirements.

2.1 AT Interface and Security

- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{AT.Setup}(1^\lambda)$ sets up a common reference string crs and a trapdoor td with security parameter λ.
- $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{AT.KeyGen}(\mathsf{crs})$ generates the public parameters pp and a secret key sk from crs.
- $\{\sigma, \perp\} \leftarrow \langle \mathsf{AT.Client}(\mathsf{pp}, t), \mathsf{AT.IssueToken}(\mathsf{sk}, b)\rangle$ is an interactive token generation protocol between a *client* (also called a *user*) and the issuer. The client inputs are a string t along with the public parameters, and the issuer inputs are the secret key sk and a *metadata bit b*. The protocol outputs a token (also referred to as signature) σ for the client or $\perp$.
- bool $\leftarrow \mathsf{AT.Verify}(\mathsf{sk}, t, \sigma)$ (run by the *redeemer*) verifies a token σ with tag t and returns a boolean value to indicate if the token was valid.
- $\mathsf{ind} \leftarrow \mathsf{AT.ReadBit}(\mathsf{sk}, t, \sigma)$ (run by the *redeemer*) extracts the metadata bit b from a token (t, σ) and returns b if it succeeds or $\perp$ if it fails.

The issuance protocol is interactive between the user and the issuer. In [16], it is assumed to be a user-to-issuer-to-user protocol (2-move, user-initiated). The security properties of an AT scheme are unforgeability, unlinkability, and privacy of the metadata bit. They are formally (re)defined in section 3. Unforgeability ensures that no adversary can forge "valid" (in the sense that AT.ReadBit does not give an error) tokens. Privacy ensures that no adversary can read a hidden bit, even with access to a "validity" (in the sense of AT.Verify) oracle.

```
PMBT.Client($X_0, X_1, t$)                                  PMBT.IssueToken($(x_0, y_0), (x_1, y_1), b$)
input: $(X_0, X_1, t)$                                      input: $(x_0, y_0), (x_1, y_1), b$
output: $\sigma$                                            output: $\{\}$
$r \leftarrow\!\!\$\ \mathbb{Z}_p^*$
$T := H_t(t)$
$T' := r^{-1}T$
                          $\xrightarrow{T'}$
                                                            $s \leftarrow\!\!\$\ \{0,1\}^\lambda$
                                                            $S' := H_s(T', s)$
                                                            $W' := x_b T' + y_b S'$
                                                  $\pi \leftarrow \Pi.\mathsf{Prove}((X_0, X_1, T', S', W'), (x_b, y_b))$
                          $\xleftarrow{(s, W', \pi)}$
$S' := H_s(T', s)$
if not $\Pi.\mathsf{Verify}((X_0, X_1, T', S', W'), \pi)$ then return $\perp$
$S := rS'$
$W := rW'$
output: $\sigma := (S, W)$
```

Fig. 1. PMBT token issuance protocol as given in [16, Fig 8, p 325]

We recall the PMBT issuance protocol in Figure 1. As specified in [16], Verify always returns true and ReadBit returns $b \in \{0,1\}$ such that $W = x_b H_t(t) + y_b S$ when it exists or $\perp$ otherwise.

2.2 Potential Attack for PMBT

In this section, we describe new attacks on PMBT. The authors of PMBT [16] already acknowledge that, given (S_i, W_i) tokens (at least two) which are generated with the same tag $t_i = t$ and the same bit $b_i = b$, the client can generate many other tokens (S, W) with the same tag t by making a weighed average: for scalars α_i such that $\sum_i \alpha_i = 1$, compute $W = \sum_i \alpha_i W_i$ and $S = \sum_i \alpha_i S_i$ to get $W = x_b H_t(t) + y_b S$. This is not considered to be a forgery because tokens with the same tag are considered as the same one. When the b_i's are not necessarily the same, the obtained token is valid (in the sense that ReadBit would read a bit) if and only if all b_i's are the same. So, a validity oracle could be used to check hidden bits equality. This is not considered to be a metadata bit privacy attack as the adversary has no access to this type of validity oracle. However, it can be considered as a side-channel attack.

New attack. In a new attack, we consider an adversary who gets one token (t_1, S_1, W_1) with a known bit b_1. Then, the adversary selects a fresh t^* and makes a challenge query by using $H_t(t^*) - H_t(t_1)$ in the place of T in the protocol with an unknown challenge bit b^*. The adversary would get (S^*, W^*) satisfying

$$W^* = x_{b^*}(H_t(t^*) - H_t(t_1)) + y_{b^*}S^*$$

Then, setting $W_2 = W^* + W_1$ and $S_2 = S^* + S_1$ gives

$$W_2 = x_{b^*}H_t(t^*) + y_{b^*}S_2 + (x_{b_1} - x_{b^*})H_t(t_1) + (y_{b_1} - y_{b^*})S_1$$

(S_2, W_2) with tag t^* would encode a bit (which would be b^*) if and only if $b_1 = b^*$ (except with the negligible probability that $(x_{b_1} - x_{b^*})H_t(t_1) + (y_{b_1} - y_{b^*})S_1 = 0$). Hence, the ability to learn whether $\mathsf{ReadBit}$ runs successfully on a chosen token breaks privacy of the hidden bit. Note that the two tokens use different tags.

This attacks generalizes, applies to the CMBT fix, and to other schemes as discussed in Appendix A.

3 Anonymous Tokens Revisited

3.1 AT Interface

We revisit the interface of AT and update the security notions as follows. We deviate from the previous definitions in three ways: first of all, there is a unique $\mathsf{AT.ReadBit}$ algorithm (and no extra $\mathsf{AT.Verify}$) which returns the hidden bit or $\perp$ if invalid. Second, the client no longer chooses the t input in the issuing protocol. Instead, a unique nonce t is returned to the client. Finally, we added, for completeness, the optional *public metadata* m attribute. For protocols not allowing it, input m is ignored in algorithms and games.[5]

- $\mathsf{cpp} \leftarrow \mathsf{AT.Setup}(1^\lambda)$ sets up the common public parameters cpp from the security parameter λ. cpp typically contains a group description, its order, and a generator.[6] cpp is input to all other algorithms and omitted for more readability.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{AT.KeyGen}(\mathsf{cpp})$ generates the public key pk and the secret key sk of the token *issuer*, with cpp as input.
- $\langle t, \sigma, \perp\rangle \leftarrow \langle \mathsf{AT.Client}(\mathsf{pk}, m), \mathsf{AT.IssueToken}(\mathsf{sk}, b, m)\rangle$ is the interactive token issuance protocol between a *client* and an *issuer*. The client inputs the issuer's public key pk and the *public metadata* m. The issuer inputs their secret key sk, a *metadata bit* $b \in \{0, 1\}$, and m. (Both participants are assumed to agree on m.) The protocol outputs a token (t, σ) composed of a tag t and a token σ for the client and nothing $(\perp)$ for the issuer.
  The elements m, b, and t are called *attributes* and can be optionally offered by the protocol. The attribute t is a *nonce*; b is the issuer's private metadata bit; m is a public metadata (on which both participants must agree).

[5] A protocol with this m option is available in Appendix B.1.
[6] In the common reference string model, the CRS cpp comes with a trapdoor td.

As we focus on a round-trip protocol which is initiated by the client (2-move, client-initiated), we can specify it by three algorithms:

  - $\mathsf{AT.ClientQuery}(\mathsf{pk}, m) \to (\mathsf{query}, \mathsf{st})$ //Client sends query to issuer
  - $\mathsf{AT.IssueToken}(\mathsf{sk}, b, m, \mathsf{query}) \to \mathsf{resp}$ //Issuer replies with resp
  - $\mathsf{AT.ClientFinal}(\mathsf{st}, \mathsf{resp}) \to (t, \sigma)$ //Client locally computes token

- $\mathsf{ind} \leftarrow \mathsf{AT.ReadBit}(\mathsf{sk}, m, t, \sigma)$ extracts a bit b from a token σ with attributes (m, t). It outputs b if it succeeds and an error $\perp$ if it fails, in which case we say the token is *invalid*.

When we run these algorithms in the order they have been introduced and we obtain $\mathsf{ind} = b$, we say that the protocol is *correct*.

The security properties of an AT scheme are unforgeability, unlinkability, and privacy of metadata bit. **Unforgeability** implies that an adversary cannot create valid tokens with modified attributes on an existing token. More precisely, if the issuer is invoked $n_{b,m}$ times for each attribute (b, m), then, for no (b, m) the adversary can exhibit $n_{b,m}+1$ valid tokens with pairwise different tags t. The adversary has access to a $\mathsf{ReadBit}(\mathsf{sk}, \cdot, \cdot)$ oracle and can choose the bit b to be hidden in the token by the issuer. **Unlinkability** implies that a malicious issuer cannot link a redeemed token with one of the issuing sessions. The malicious issuer can maliciously set up the public parameters. **Privacy** of metadata means that a malicious client cannot guess the metadata bit hidden during a issuing session, even with access to an oracle for checking if a token is valid (but without access to an oracle which extracts the bit).

3.2 Unforgeability

The one-more unforgeability game (OMUF) is defined in Figure 2. This is the same as in Kreuter et al. [16] except for a modification in the quantifiers[7] and for the modification in the interface: having the token verification and the bit extraction in the same algorithm and oracle. Notably, (t, σ) making $\mathsf{AT.Verify}$ return true and $\mathsf{AT.ReadBit}$ return $\perp$ would not exist any more as there is no $\mathsf{AT.Verify}$. Those differences do not change the security notion.

Definition 1. *In the* OMUF *game*[8] *in Figure 2, we define the advantage of an adversary* $\mathcal{A}$ *by*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{OMUF}}(\lambda) = \Pr[\mathsf{win}]$$

We say that AT *is* OMUF*-secure if for any PPT adversary* $\mathcal{A}$*, the advantage is a negligible function.*

[7] In their OMUF security definition, the first condition says that both $q_0 \leq \ell$ AND $q_1 \leq \ell$ where q_b is the number of oracle queries with bit b. Suppose, the adversary made 10 queries with $b = 0$ ($q_0 = 10$) and 1000 queries with $b = 1$ ($q_1 = 1000$). If the adversary forges 11 tokens with $b = 0$, for this to succeed as a forgery, ℓ must be at least 1000. If it is not, this does not succeed.

[8] For protocols with no public metadata m, the variables $n_{b,m}$ in the game shall be changed to n_b and input m shall be removed from AT algorithms.

```
Game $\mathsf{OMUF}(1^\lambda)$:
1: $\mathsf{AT.Setup}(1^\lambda) \to \mathsf{cpp}$
2: $\mathsf{AT.KeyGen}(\mathsf{cpp}) \to (\mathsf{pk}, \mathsf{sk})$
3: initialize $n_{b,m} \leftarrow 0$ for all $(b, m)$
4: $\mathcal{A}^{\mathcal{O}_{\mathsf{sign}}, \mathcal{O}_{\mathsf{read}}}(\mathsf{cpp}, \mathsf{pk}) \to b, m, (t_i, \sigma_i)$
5: if $\#\{t_i\} \leq n_{b,m}$ then abort    ▷ this counts the number of pairwise different $t_i$ values
6: for each $i$ do
7:     if $\mathsf{AT.ReadBit}(\mathsf{sk}, t_i, m, \sigma_i) \neq b$ then abort
8: end for
9: adversary wins

Oracle $\mathcal{O}_{\mathsf{sign}}(b, m, \mathsf{query})$:
10: increment $n_{b,m}$
11: $\mathsf{AT.IssueToken}(\mathsf{sk}, b, m, \mathsf{query}) \to \mathsf{resp}$
12: return resp

Oracle $\mathcal{O}_{\mathsf{read}}(m, t, \sigma)$
13: return $\mathsf{AT.ReadBit}(\mathsf{sk}, m, t, \sigma)$
```

Fig. 2. One-More UnForgeability Game with Public Metadata m

3.3 Unlinkability

The unlinkability game (UNLINK) is defined in Figure 3. This is the same as in Kreuter et al. [16] except for the modification in the interface: the tag t is output instead of being an arbitrarily selected input by the client and the public metadata m must be the same for all challenge tokens.

Definition 2. *In the* UNLINK *game in Figure 3, we define the advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *for parameter* n *by*

$$\mathsf{Adv}^{\mathsf{UNLINK}}_{\mathcal{A},n}(\lambda) = \Pr[\mathsf{win}]$$

We say that AT *is* κ-UNLINK*-secure if for any PPT adversary* $\mathcal{A}$ *and any integer* n*, the advantage bounded by* $\frac{\kappa}{n}$ *plus a negligible function.*

The issuer knows which bit is hidden during an issuing session and can extract the hidden bit during redeem. Hence, we should only consider unlinkability when the bits are the same.

If an adversary (1) puts n_b tokens with bit b for each b in $\mathcal{Q}$ with total of $n = \sum_b n_b$ tokens; (2) it draws and reads the bit b in out_{i^*}; (3) it outputs i at random among indices where it put the token with $b = b^*$, then the adversary wins with probability $\sum_b \frac{n_b}{n} \times \frac{1}{n_b} = \frac{2}{n}$. So, we focus on 2-UNLINK security.

3.4 Privacy of the Metadata Bit

The privacy of the metadata bit game (PMB) is defined in Figure 4 with a challenge bit b^*. This is the same as in Kreuter et al. [16] except for the modification in the interface: the verify oracle is implemented by checking if AT.ReadBit does not return $\perp$. We also modified to have a single access to $\mathcal{O}_{\mathsf{chal}}$ and to give access to $\mathcal{O}_{\mathsf{read}}$ until the challenge is released. In the case of [16], the separation between AT.Verify and AT.ReadBit allowed $\mathcal{O}_{\mathsf{verify}}$ to return true although

```
Game $\mathsf{UNLINK}_n(1^\lambda)$:
1: $\mathsf{AT.Setup}(1^\lambda) \to \mathsf{cpp}$
2: initialize $\mathcal{Q}_{\mathsf{query}}, \mathcal{Q}_{\mathsf{final}} \leftarrow \emptyset$
3: $\mathcal{A}_1(\mathsf{cpp}) \to (\mathsf{pk}, \mathsf{state}_1)$
4: $\mathcal{A}_2^{\mathcal{O}_{\mathsf{query}}, \mathcal{O}_{\mathsf{final}}}(\mathsf{state}_1) \to$
        $(\mathcal{Q}, (\mathsf{resp}_i)_{i \in \mathcal{Q}}), \mathsf{state}_2)$
5: if $\mathcal{Q} \not\subseteq \mathcal{Q}_{\mathsf{query}} - \mathcal{Q}_{\mathsf{final}}$ then abort
6: if $\#\mathcal{Q} < n$ then abort
7: for all $i \in \mathcal{Q}$ do
8:     $\mathsf{out}_i \leftarrow \mathsf{AT.ClientFinal}(\mathsf{st}_i, \mathsf{resp}_i)$
9:     if $\mathsf{out}_i = \perp$ then abort
10: end for
11: if $\#\{m_i; i \in \mathcal{Q}\} > 1$ then abort    ▷ the $m_i$ must all be the same
12: $i^* \leftarrow\!\!\$\; \mathcal{Q}$
13: pick a random permutation $\varphi$ of $\mathcal{Q}$
14: $\mathcal{A}_3(\mathsf{state}_2, \mathsf{out}_{i^*}, (\mathsf{out}_{\varphi(i)})_{i \in \mathcal{Q}}) \to i$
15: win iff $i = i^*$

Oracle $\mathcal{O}_{\mathsf{query}}(i, m)$:
16: if $i \in \mathcal{Q}_{\mathsf{query}}$ then return
17: insert $i$ in $\mathcal{Q}_{\mathsf{query}}$
18: $m_i \leftarrow m$
19: $\mathsf{AT.ClientQuery}(\mathsf{pk}, m_i) \to \mathsf{query}_i, \mathsf{st}_i$
20: return $\mathsf{query}_i$

Oracle $\mathcal{O}_{\mathsf{final}}(i, \mathsf{resp})$:
21: if $i \in \mathcal{Q}_{\mathsf{final}}$ or $i \notin \mathcal{Q}_{\mathsf{query}}$ then return
22: insert $i$ in $\mathcal{Q}_{\mathsf{final}}$
23: return $\mathsf{AT.ClientFinal}(\mathsf{st}_i, \mathsf{resp})$
```

Fig. 3. Unlinkability Game

AT.ReadBit would return $\perp$. Our interface does not allow it any more so the adversary has more information.

Definition 3. *In the* PMB *game in Figure 4, we define the advantage of an adversary $\mathcal{A}$ by*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PMB}}(\lambda) = \Pr[\mathsf{PMB}_1 \to 1] - \Pr[\mathsf{PMB}_0 \to 1]$$

We say that AT *is* PMB*-secure if for any PPT adversary $\mathcal{A}$, the advantage is a negligible function.*

4 ATHM: Anonymous Tokens with Hidden Metadata

Instead of relying on a deterministic PRF, we construct a protocol which is based on a randomized algebraic MAC, like in keyed-verification anonymous credentials [6,7] and Signal's private group management in group chats [8,9]. A valid MAC for an input b with a nonce t and a secret key (x, y, z) is a pair $\sigma = (P, Q)$ such that $Q = (x + by + tz)P$. In our scheme, a token with a hidden bit b will be a pair (t, σ) such that σ is a valid MAC for the attributes (b, t) with a secret key (x, y, z). In Appendix B.1, we present variants and extensions of ATHM to introduce public metadata m or other MAC algorithms.

4.1 The ATHM Components

Our scheme uses as a building block a simulatable non-interactive proof Π_2.

```
Game $\mathsf{PMB}_{b^*}(1^\lambda)$:
 1: $\mathsf{AT.Setup}(1^\lambda) \to \mathsf{cpp}$
 2: $\mathsf{AT.KeyGen}(\mathsf{cpp}) \to (\mathsf{pk}, \mathsf{sk})$
 3: $\mathsf{flag} \leftarrow \mathsf{false}$
 4: return $\mathcal{A}^{\mathcal{O}_{\mathsf{sign}}, \mathcal{O}_{\mathsf{chal}}, \mathcal{O}_{\mathsf{read}}, \mathcal{O}_{\mathsf{valid}}}(\mathsf{cpp}, \mathsf{pk})$

Oracle $\mathcal{O}_{\mathsf{read}}(m, t, \sigma)$:
 5: if $\mathsf{flag}$ and $m = m^*$ then return $\perp$
 6: return $\mathsf{AT.ReadBit}(\mathsf{sk}, m, t, \sigma)$

Oracle $\mathcal{O}_{\mathsf{valid}}(m, t, \sigma)$
 7: $\mathsf{bool} \leftarrow \mathsf{AT.ReadBit}(\mathsf{sk}, m, t, \sigma) \neq \perp$
 8: return $\mathsf{bool}$

Oracle $\mathcal{O}_{\mathsf{sign}}(b, m, \mathsf{query})$:
 9: $\mathsf{AT.IssueToken}(\mathsf{sk}, b, m, \mathsf{query}) \to \mathsf{resp}$
10: return $\mathsf{resp}$

Oracle $\mathcal{O}_{\mathsf{chal}}(m, \mathsf{query})$:
11: if $\mathsf{flag}$ then return $\perp$
12: $\mathsf{flag} \leftarrow \mathsf{true}$
13: $m^* \leftarrow m$
14: $\mathsf{AT.IssueToken}(\mathsf{sk}, b^*, m^*, \mathsf{query}) \to$
    $\mathsf{resp}$
15: return $\mathsf{resp}$
```

Fig. 4. Privacy of the Metadata Bit Game

Setup *algorithm.* Setup is composed of two phases. The Setup_0 algorithm generates an (additive) group, of prime order p, and a generator G. The Setup_2 algorithm selects common parameters cpp_2 for Π_2.

```
$\mathsf{Setup}(1^\lambda)$:
 1: $\mathsf{Setup}_0(1^\lambda) \to (\mathsf{gp}, p, G)$            ▷ group setup
 2: $\mathsf{Setup}_2(\mathsf{gp}, p, G) \to \mathsf{cpp}_2$             ▷ $\Pi_2$ setup
 3: $\mathsf{cpp} \leftarrow (\mathsf{gp}, p, G, \mathsf{cpp}_2)$
```

Our proposed Π_2 scheme requires Setup_2 to select uniformly a random non-zero group element $\mathsf{cpp}_2 = H$. (See subsection 4.3.)

The KeyGen *algorithm.* Key generation is composed of several phases.

```
$\mathsf{KeyGen}(\mathsf{cpp})$:
 1: $\mathsf{KeyGen}_0(\mathsf{cpp}) \to (\mathsf{pk}_0, \mathsf{sk}_0)$
 2: $\mathsf{KeyGen}_2(\mathsf{cpp}, \mathsf{pk}_0, \mathsf{sk}_0) \to (\mathsf{pk}_2, \mathsf{sk}_2)$      ▷ $\Pi_2$ key generation
 3: $\mathsf{pk} \leftarrow (\mathsf{pk}_0, \mathsf{pk}_2)$
 4: $\mathsf{sk} \leftarrow (\mathsf{sk}_0, \mathsf{sk}_2)$
```

In KeyGen_0, the issuer selects three secrets $\mathsf{sk}_0 = (x, y, z) \in \mathbb{Z}_p \times (\mathbb{Z}_p^*)^2$ uniformly ($y, z \neq 0$) and sets $\mathsf{pk}_0 = Z = zG$.

Our proposed Π_2 scheme requires KeyGen_2 to add in pk_2 Pedersen commitments [19] $C_x = xG + r_xH$, $C_y = yG + r_yH$, together with a Schnorr proof of knowledge [20] of z such that $Z = zG$. Clients must verify this proof before starting the issuance protocol, but this is done once for all. (See subsection 4.3.)

The token issuance protocol. The user has public parameters. The server's input is the secret (x, y, z) and a bit b to hide inside the token. The protocol works as depicted in Figure 5: the client selects a random tag share t_C and a random mask $r \in \mathbb{Z}_p$ and sends $T = t_C Z + rG$ to the issuer. The issuer selects a random tag share t_S and generates a pair (U, V) such that $(U, V - rU)$ is a valid MAC for tag $t = t_C + t_S$ and metadata b with key (x, y, z). For this, the issuer selects $U = dG$ for a random $d \in \mathbb{Z}_p^*$ and $V = d(xG + byG + t_S zG + T)$. Π_2 proves that the

(U, V, t_S) triplet was correctly generated and that $b \in \{0, 1\}$. Then, (U, V, t_S, π) is returned. The client computes $(U, V - rU)$. To make it unlinkable, the pair is multiplied by a random mask c to obtain another pair $\sigma = (P, Q)$.

```
ATHM.Client$(G, Z)$                                             ATHM.IssueToken$((x, y, z), b)$
input: $(G, Z)$                                                 input: $(x, y, z), b$
$t_C, r \leftarrow\!\!\$\ \mathbb{Z}_p$
$T := t_C Z + rG$
                                     $\xrightarrow{T}$
                                                                $t_S \leftarrow\!\!\$\ \mathbb{Z}_p$
                                                                $d \leftarrow\!\!\$\ \mathbb{Z}_p^*$
                                                                $U := dG$
                                                                $V := d(xG + byG + t_S zG + T)$
                                     $\pi \leftarrow \Pi_2$.Prove$(b, d, \mathsf{sk}_0, \mathsf{sk}_2; U, V, t_S, T, \mathsf{cpp}, \mathsf{pk})$
                                     $\xleftarrow{U, V, t_S, \pi}$
if not $\Pi_2$.Verify$(\pi, U, V, t_S, T, \mathsf{cpp}, \mathsf{pk})$ then return $\bot$
if $U = 0$ then return $\bot$
$c \leftarrow\!\!\$\ \mathbb{Z}_p^*$
$P := cU$
$Q := c(V - rU)$
$t := t_C + t_S$
$\sigma := (P, Q)$
output: $(t, \sigma)$
```

Fig. 5. ATHM token issuance protocol.

The proof Π_2 is a Fiat-Shamir transform of an OR proof of two Schnorr proofs for $b = 0$ and $b = 1$. It is specified in subsection 4.3.

The ReadBit *algorithm.* The redemption of (t, P, Q) with (x, y, z) checks $P \neq 0$ and looks for which $b \in \{0, 1\}$, the equality $Q = (x + by + tz)P$ is satisfied. It returns that bit (it must be unique) or an error if there is none.

Rationales. MACGGM (see subsection 4.2) ensures OMUF security. It uses two attributes to carry the tag t and hidden bit b. We observe that it is necessary that the issuer contributes to t. If the client could decide $t = t_C$, then PMB security could be broken by getting a challenge with tag t^* then issuing a token with same tag $t = t^*$ and taking the weighted average of both. The obtained token is valid if and only if b^* is equal to the bit b put in the second token.

We can also observe that it is necessary to have t_C, r, and c. Without any of them, the malicious issuer can easily break UNLINK security by linking an issued token with a redeemed token.

The same goes with the proof Π_2 which plays two roles. First, it proves that the used sk corresponds to pk. Without Π_2, the issuer could use a set of different sk's and make a selection of sk in this set which would play the role of a marker in the token. Second, Π_2 proves that either $b = 0$ or $b = 1$, thus b only contains one bit. Without Π_2, the issuer could use more than one bit inside b and use this as a marker to link tokens to redeem to clients requesting a token. So, Π_2 is necessary for UNLINK security. The client must also verify $U \neq 0$, because $U = 0$ could be used by the issuer to mark a token.

Beware of double spending. Note that for this protocol, tag t needs to be a nonce as in other protocols [15], [16] i.e. the redeemer should check against double-spending of a token with the same t. Otherwise, it is easy to transform a valid token with tag t into another valid token with the same tag t: let $\sigma = (P, Q)$ be a signature on t. Then the client can forge another signature $\sigma' = (c'P, c'Q)$ on t for a random c'.

4.2 The MAC Building Block

Our security results will be based on the security of an algebraic MAC. The simplest one is the MACGGM algorithm [6] defined as follows: given a secret $(x, y, z) \in \mathbb{Z}_p^3$, a valid authentication for $(b, t) \in \mathbb{Z}_p^2$ is a pair $\sigma = (P, Q)$ such that $Q = (x + by + tz)P$. For this MAC to be secure, it is important that no adversary can find any linear relation between the random values of P. Hence, P is selected at random by the issuer.

The security of MACGGM was proven in the generic group model (GGM) [6]. So, we use the same model to prove the security of ATHM. However, our construction generalizes to other MAC algorithms which can be proven in the standard model, and we use non-GGM security for this generalization, as shown in Appendix B.2.

4.3 The Simulatable Proof Building Block

As already mentioned, setup and key generation for the π proof is specified in Figure 6. The setup essentially sets up an additional generator H to make a Pedersen commitment [19]. Key generation computes Pedersen commitments C_x on x and C_y on y together with a Schnorr proof [20] with Fiat-Shamir transform [14] for the knowledge of z such that $Z = zG$. Clients are assumed to verify that pk is correct by running once the Verify_2 algorithm.

On a high level, the proof π first commits to b by releasing a Pedersen commitment [19] $C = bC_y + \mu H$, then performs an OR proof [11]: a proof of knowledge for μ such that either $C = \mu H$ or $C = C_y + \mu H$. Finally, it performs a proof of knowledge for (d', ρ, w) such that $-G = d'U$, $-(C_x + C + t_S Z + T) = d'V + \rho H$, and $-T = d'V + wG$. The link with d is that $d' = -\frac{1}{d}$. Hence

$$\exists(d', \rho, w) \quad d' \begin{pmatrix} U \\ V \\ V \end{pmatrix} + \rho \begin{pmatrix} 0 \\ H \\ 0 \end{pmatrix} + w \begin{pmatrix} 0 \\ 0 \\ G \end{pmatrix} = - \begin{pmatrix} G \\ C_x + C + t_S Z + T \\ T \end{pmatrix} \tag{1}$$

```
Setup_2(gp, p, G):
 1: $\mathsf{td}_2 \leftarrow\!\$\ \mathbb{Z}_p^*$
 2: $H \leftarrow \mathsf{td}_2.G$
 3: $\mathsf{cpp}_2 \leftarrow H$
 4: return $\mathsf{cpp}_2$

KeyGen_2(cpp, pk_0, sk_0):
 5: $(\mathsf{gp}, p, G, H) \leftarrow \mathsf{cpp}$
 6: $Z \leftarrow \mathsf{pk}_0$
 7: $(x, y, z) \leftarrow \mathsf{sk}_0$
 8: $r_x, r_y \leftarrow\!\$\ \mathbb{Z}_p$
 9: $C_x \leftarrow xG + r_xH$
10: $C_y \leftarrow yG + r_yH$
11: $\rho_z \leftarrow\!\$\ \mathbb{Z}_p$
12: $\Gamma_z \leftarrow \rho_z G$
13: $\varepsilon \leftarrow \mathsf{Hash}(G, H, Z, \Gamma_z)$
14: $a_z \leftarrow \rho_z + \varepsilon z\mathrm{g}$
15: $\mathsf{sk}_2 \leftarrow (r_x, r_y)$
16: $\mathsf{pk}_2 \leftarrow (C_x, C_y, \varepsilon, a_z)$
17: return $(\mathsf{pk}_2, \mathsf{sk}_2)$

Verify_2(cpp, pk):
18: $(\mathsf{gp}, p, G, H) \leftarrow \mathsf{cpp}$
19: $(Z, C_x, C_y, \varepsilon, a_z) \leftarrow \mathsf{pk}$
20: $\Gamma_z \leftarrow a_zG - \varepsilon Z$
21: return $1_{\varepsilon = \mathsf{Hash}(G,H,Z,\Gamma_z)}$
```

Fig. 6. Initialization of Π_2.

where $\rho = -(r_x + br_y + \mu)$ and $w = x + by + t_S z$. The proof follows standard NIZK techniques, with (generalized) Schnorr proof [20], OR proof [11], and Fiat-Shamir transform [14].

We will use two properties of Π_2: that we can simulate the proof on any entry by programming the random oracle, and that we have a straightline extractor for (x, y, z, b, d) in the generic group model. We will prove these properties in the security analysis.

To construct Prove, we merge this OR proof with the AND proofs with statement given in Equation 1 and we transform into a non-interactive proof. We formally define the algorithms in Π_2 below.

Prove. The algorithm $\mathsf{Prove}(b, d, \mathsf{sk}_0, \mathsf{sk}_2; U, V, t_S, T, \mathsf{cpp}, \mathsf{pk})$ parses different elements, picks μ, sets $C = bC_y + \mu H$, $d' = -\frac{1}{d}$, $\rho = -(r_x + br_y + \mu)$, and $w = x + by + t_S z$.

The issuer (prover) picks e_{1-b}, a_{1-b}, r_μ, r_d, r_ρ, r_w at random and computes $C_b = r_\mu H$, $C_{1-b} = a_{1-b}H - e_{1-b}(C - (1-b)C_y)$,

$$\begin{pmatrix} C_d \\ C_\rho \\ C_w \end{pmatrix} = r_d \begin{pmatrix} U \\ V \\ V \end{pmatrix} + r_\rho \begin{pmatrix} 0 \\ H \\ 0 \end{pmatrix} + r_w \begin{pmatrix} 0 \\ 0 \\ G \end{pmatrix}$$

$e = \mathsf{Hash}(G, H, C_x, C_y, Z, U, V, t_S, T, C, C_0, C_1, C_d, C_\rho, C_w)$, $e_b = e - e_{1-b}$, $a_b = r_\mu + e_b\mu$, and $(a_d, a_\rho, a_w) = (r_d, r_\rho, r_w) + e(d', \rho, w)$. Finally, the output is

$$\pi = (C, e_0, e_1, a_0, a_1, a_d, a_\rho, a_w)$$

Verify. The algorithm $\mathsf{Verify}(\pi, U, V, t_S, T, \mathsf{cpp}, \mathsf{pk})$ parses π, cpp, and pk, computes $C_0 = a_0H - e_0C$, $C_1 = a_1H - e_1(C - C_y)$, $e = e_0 + e_1$,

$$\begin{pmatrix} C_d \\ C_\rho \\ C_w \end{pmatrix} = a_d \begin{pmatrix} U \\ V \\ V \end{pmatrix} + a_\rho \begin{pmatrix} 0 \\ H \\ 0 \end{pmatrix} + a_w \begin{pmatrix} 0 \\ 0 \\ G \end{pmatrix} + e \begin{pmatrix} G \\ C_x + C + t_S Z + T \\ T \end{pmatrix}$$

then verifies $e = \mathsf{Hash}(G, H, C_x, C_y, Z, U, V, t_S, T, C, C_0, C_1, C_d, C_\rho, C_w)$.

5 Performance

Implementation. We implemented our construction as given in Figure 5 in Rust (version 1.66.0). We use the Ristretto group using curve25519-dalek library. We use RistrettoBasePointTable struct (which is a precomputed table for multiplications with the group generator G) to accelerate the scalar multiplications (PMBT implementation does *not* use these tables). We use two of these tables: one for G and one for H which requires 60 KB of memory for constant time cryptography and up to 4 times speed up. For OR proofs and verification, we did *not* rely on any external library, meaning we implemented it in pure Rust. We did *not* use multi-scalar multiplication. It is available at `https://github.com/Microsoft/MacTok`.

We benchmarked the implementation on a machine with Intel(R) i7-1185G7 3.00GHz CPU. Our benchmarks excludes the key generation (because it is generated only once for all). They include client blinded message generation, server's computation of MACs (blindly, along with the proof $\Pi_2.\mathsf{Prove}$), client's unblinding (along with the $\Pi_2.\mathsf{Verify}_2$), and server's redemption. It takes 1.3 ms whereas PMBT takes 1.6 ms for the same operations [9]. We note that we disabled SIMD optimizations (which allows curve25519-dalek to run faster curve operations) in both ATHM and PMBT due to the unstable version (1.66.0-nightly) of the Rust compiler that does not allow building PMBT. When we run our ATHM protocol with SIMD optimization, we get 0.9 ms of running time.

Theoretical Complexity. We also computed the number of scalar multiplications to compare ATHM with PMBT and observed that ATHM computes 29 scalar multiplication whereas PMBT computes 31 multiplications in total, for benchmarked operations (client and server side computations including redemption along with the proof and verification).

In ATHM, the issuer computes 11 scalar multiplications during issuance (one for C, 7 for the proof, and 3 for the ATHM protocol). The client computes 17 scalar multiplications (one for $t_S Z$, 11 for the proof, and 5 for ATHM). The total is 28 multiplications. [10] Furthermore, the number of transmission is of 4 group elements (C in the proof and (T, U, V) in the proof) and 8 integers (t_S in ATHM and the 7 elements of π). The total is 12.[11] For redeem, the number of multiplications is 1 and the token length is 3 (t, P, and Q). For key generation, the issuer computes 2 multiplications (1 for Z and 1 for pk_2) and the client computes 2 multiplications for the Π_2 verification. The public key contains 5 elements (1 for Z and 4 for pk_2).

[9] PMBT code is available at `https://github.com/mmaker/anonymous-tokens`

[10] In this count, we took the computation of $(1-b)C_y$ as free. Furthermore, the computation of $r_d V$ and $a_d V$ are done twice but count for a single operation.

[11] By setting $t_S = \mathsf{Hash}(U)$, the issuer would not have to send t_S any longer and save the transmission of one $\mathbb{Z}_p$ element.

As a comparison, in PMBT, for issuance, the issuer computes 12 multiplications in total and the client computes 15 multiplications. In total, the number of multiplications is 27 for issuance. The number of transmissions is 2 group elements and 7 scalars. The total is 9. The redemption needs 4 multiplication with a token length 2 group elements. For key generation, the issuer computes 4 multiplication. The public key contains 4 elements.

We provide a complexity comparison in Table 1. We compute the number of scalar multiplications for both participants during issuance, and the amount of communication between them ($\mathbb{G}$ stands for group elements, $\mathbb{Z}_p$ stands for integers, and h stands for hashes), and the same for redeem. We added KVAC reduced to two attributes t and b, as presented in Appendix C. Note that PP does not have hidden bit metadata, PMBT uses a weaker PMB security notion, and KVAC does not ensure PMB security.

Table 1. Complexity comparison

	Issuance			**Redemption**		**Total**	
	Client	Server	Comm.	Server	Comm.	Comp.	Comm.
ATHM	17×	11×	$4\mathbb{G} + 7\mathbb{Z}_p$	1×	$2\mathbb{G} + 1\mathbb{Z}_p$	29×	$6\mathbb{G} + 8\mathbb{Z}_p$
PP	2×	7×	$2\mathbb{G} + 1\mathbb{Z}_p + 1h$	1×	$1\mathbb{G} + 1h$	10×	$3\mathbb{G} + 1\mathbb{Z}_p + 2h$
PMBT	15×	12×	$2\mathbb{G} + 7\mathbb{Z}_p$	4×	$2\mathbb{G} + 1\mathbb{Z}_p$	31×	$4\mathbb{G} + 8\mathbb{Z}_p$
KVAC	35×	30×	$7\mathbb{G} + 12\mathbb{Z}_p$	1×	$2\mathbb{G} + 1\mathbb{Z}_p$	66×	$9\mathbb{G} + 13\mathbb{Z}_p$
BLOR	11×	7×	$5\mathbb{G} + 6\mathbb{Z}_p$	6×	$3\mathbb{G} + 4\mathbb{Z}_p$	24×	$8\mathbb{G} + 10\mathbb{Z}_p$

We also include BLOR [2] (that we named from the initials of the authors). This protocol has a different model: public verifiability but hidden metadata. This means that anyone can check whether a token hides a hidden bit but only the issuer can determine which bit is hidden. Besides, the issuance protocol has an additional move. So the protocol implies that the issuer is a stateful server.

6 Security Proof for ATHM in the Generic Group Model

We consider ATHM as specified in section 4, i.e. with MACGGM, no T_{ext}, and with the nonce t and private metadata bit b attributes. (See Appendix B for T_{ext} and other options.) Considering more attributes would work the same. We prove OMUF, 2-UNLINK, and PMB security in the generic group and random oracle models. The security of MACGGM without generic groups is an open problem, so the GGM seems to be unavoidable here. It also helps to build a straightline extractor for Π_2 (in UNLINK). The random oracle is used in the simulation of Π_2 (in OMUF and PMB) and in the soundness of Π_2 (in UNLINK).

We first discuss about the Generic Group Model (GGM). We adopt the model by Maurer [17] which models a lower-level interface to the generic group. In this model, all group operations are outsourced to an external oracle which also keeps the group element values in registers which cannot be read by the adversary. Initially, the setup $\mathsf{GGMSetup}$ sets a register $\mathsf{Mem}[1]$ set to a "base" group element from the setup: the generator G. Other registers $\mathsf{Mem}[\cdot]$ are initialized to 0. These registers are given an address that the adversary or the game can use. The adversary uses addresses as references when a group operation is requested but never sees the element value itself. Actually, the oracle only computes subtractions GGMSub (from which we can do additions, scalar multiplications, inversion, and get the neutral element) and comparisons GGMCmp (whether or not the group elements referred to by two addresses are equal):

```
Oracle $\mathsf{GGMSetup}(\lambda)$:
 1: $\mathsf{Setup}_0(1^\lambda) \to (\mathsf{gp}, p, G)$
 2: initialize $\mathsf{Mem}[\cdot] = 0$
 3: $\mathsf{Mem}[1] \leftarrow G$
 4: return $p$

Oracle $\mathsf{GGMSub}(i, j, k)$:
 5: $\mathsf{Mem}[k] \leftarrow \mathsf{Mem}[i] - \mathsf{Mem}[j]$
 6: return

Oracle $\mathsf{GGMCmp}(i, j)$:
 7: return $1_{\mathsf{Mem}[i]=\mathsf{Mem}[j]}$
```

Clearly, the only oracle leaking information about group element values is the comparison oracle GGMCmp. The main task of the proof is to simulate this oracle to reduce it to a trivial game.

The Maurer generic group model [17] does not allow to efficiently hash group elements or to build dictionaries with group elements as a key, which is essential in generic algorithms such as Pollard Rho or Baby-Step Giant-Step. Managing to do so essentially reduces to simulating the original Shoup generic group model [21]. In the Shoup model, the adversary has access to the *encryption* of the group element values and can interact with an oracle to subtract encrypted elements. An ideal deterministic encryption is set up at the beginning of the game. To simulate the Shoup model with q_{sub} calls to the subtraction oracle, we need $q_{\mathsf{GGMCmp}} = \frac{q_{\mathsf{sub}}(q_{\mathsf{sub}}-1)}{2}$ calls to GGMCmp.

Overview of our proofs in GGM. In our security proofs, we need to distinguish registers which are visible by the adversary from others which are used by the game or other oracles. For that, we imagine an interface between the GGM oracles and a querier who is either the game/oracles or the adversary. The interface controls access privileges, depending on whether the query comes from the adversary or not. For convenience, we partition the memory Mem into three arrays of registers: base elements $\mathsf{Base}[\cdot]$, working registers $R[\cdot]$ for the adversary, and private registers $\mathsf{Rpriv}[\cdot]$ for the game/oracles. $\mathsf{GGMSub}(i, j, k)$ queries which are from the game or oracles should use a k address pointing to the Rpriv array. The interface would make the i address of $\mathsf{Mem}[i]$ correspond to a register address i' in one of the three arrays, depending on i, and likewise for $\mathsf{Mem}[j]$. For instance, the partition could be defined by address i corresponding to the register at address $\lfloor \frac{i}{3} \rfloor$ in one of the three arrays, depending on $i \bmod 3$.

Elements which are new for the adversary (in the sense that they are provided by the game or oracles) are stored in the Base array. Namely, setup stores

$\mathsf{Base}[1] = G$ at the beginning of the game. A variable dim keeps the number of assigned base elements (i.e. $\mathsf{dim} = 1$ after setup). The Rpriv array is not accessible by the adversary (i.e., the interface does not allow a $\mathsf{GGMSub}(i, j, k)$ or $\mathsf{GGMCmp}(i, j)$ query from the adversary with any input address i, j, k corresponding to Rpriv). The Base array is a write-once array which is readable but not writable except by a new $\mathsf{Reveal}(i)$ system call (i.e. the inputs i and j in any $\mathsf{GGMSub}(i, j, k)$ or $\mathsf{GGMCmp}(i, j)$ query can point to the Base array, but not k). The new system call $\mathsf{Reveal}(i)$, which is *not* accessible by the adversary, increments the value of dim and assigns $\mathsf{Base}[\mathsf{dim}]$ to $\mathsf{Mem}[i]$. The idea is that $\mathsf{Mem}[i]$ corresponds to a $\mathsf{Rpriv}[i']$ result from a computation by the game/oracles which should be returned to the adversary. Hence, Reveal is the only access which is writing inside Base. It is write-once: we make sure that elements are not overwritten. So we have dim "base" elements in the Base array. Actually, "base" should be understood in the sense of linear algebra: it is intended that every element in Base are "linearly free". Any vanishing linear combination would imply solving a discrete logarithm problem.

The adversary can work in the $R[\cdot]$ array and can read in the $\mathsf{Base}[\cdot]$ array. Given an adversary $\mathcal{A}$ in this model, we can define another adversary $\mathcal{B}$ who does the same as $\mathcal{A}$ but follows step by step the oracles queries made by $\mathcal{A}$ to GGMSub in order to express every $R[i']$ as a known linear combination of the base elements. Initially, $\mathcal{B}$ defines vectors $\mathsf{Vec}[\cdot]$ from $\mathbb{Z}^\infty$ (i.e., sequences of eventually null integers, but only the first dim coordinates can be nonzero) which are initialized to the zero vector. Upon a query $\mathsf{GGMSub}(i, j, k)$ by $\mathcal{A}$, $\mathcal{B}$ forwards the query and does an additional task: first, it defines vectors v_i and v_j corresponding to input i and j. If address i is pointing to $R[i']$, we define $v_i = \mathsf{Vec}[i']$. If address i is pointing to $\mathsf{Base}[i']$, we define $v_i = (0, \ldots, 0, 1, 0, \ldots)$ with 1 at coordinate i'. The same is done with v_j. Note that k must point to some $R[k']$. Hence, $\mathcal{B}$ affects $\mathsf{Vec}[k'] \leftarrow (v_i - v_j) \bmod p$. With these operations, we easily prove by induction that at every step of the game, we have

$$R[i'] = \sum_{\ell=1}^{\mathsf{dim}} \mathsf{Vec}[i']_\ell \cdot \mathsf{Base}[\ell]$$

So, from now on, we assume without loss of generality that all adversaries follow this approach to express group elements as a linear combination of "base" elements. We call *linearization* the transform of $\mathcal{A}$ to $\mathcal{B}$.

When the server is modelled by the game, the base elements consist of G and H from setup, Z, C_x, and C_y from key generation, and every (U, V) pair returned by the issuer, as well as a C value from π.

In the next transform, which we call *algebraic transform*, each secret scalar value such as td_2, x, y, z, r_x, r_y, d_i, etc are associated to a formal variable $\bar{\mathsf{td}}_2$, $\bar{x}$ and so on. Let Var be the tuple of variables and Val be the corresponding tuple of scalar values. Each base element $\mathsf{Base}[\ell]$ will be associated to a multivariate scalar function $P_\ell^{\mathsf{Base}}(\mathsf{Var})$ satisfying the fundamental property that $\mathsf{Base}[\ell] = P_\ell^{\mathsf{Base}}(\mathsf{Val}).G$. The function will be known by $\mathcal{B}$ (only Val remains unknown).

Then, $\mathcal{B}$ will define a formal multivariate function

$$P^R_{i'}(\mathsf{Var}) = \sum_{\ell=1}^{\mathsf{dim}} \mathsf{Vec}[i']_\ell \cdot P^{\mathsf{Base}}_\ell(\mathsf{Var})$$

and extend the fundamental property to $R[i'] = P^R_{i'}(\mathsf{Val}).G$ thanks to the linear expression. Those functions will appear to be low-degree polynomials. The goal of this transformation is to be able to simulate OCmp by simply comparing the polynomials. The simulation is not perfect because some polynomials may be different and still evaluate to the same scalar, but this will occur with negligible probability thanks to the Schwartz-Zippel lemma.

6.1 OMUF Security in GGM and ROM

Theorem 1. *For every $\mathcal{A}$ playing* OMUF *in the generic group and random oracle models and making n oracle calls to $\mathcal{O}_{\mathsf{sign}}$, if Π_2 is perfectly simulatable, we have*

$$\mathsf{Adv}^{\mathsf{OMUF}} \leq (2(n+1+q_{\mathcal{O}_{\mathsf{read}}}) + q_{\mathsf{GGMCmp}} + 1) \times \frac{n+2}{p} + \frac{n}{p} + \frac{q_{\mathsf{Hash}}}{p^6}$$

where n, $q_{\mathcal{O}_{\mathsf{read}}}$, q_{GGMCmp}, and q_{Hash} are the number of queries to $\mathcal{O}_{\mathsf{sign}}$, $\mathcal{O}_{\mathsf{read}}$, GGMCmp*, and to the random oracle.*

By doing q_{GGMCmp} queries, the adversary can compute the discrete logarithm z of Z with success probability q_{GGMCmp}/p. Then, with $n = 1$ query, the adversary gets one valid token (b, t, P, Q). For any δ, the token $(b, t + \delta, P, Q + \delta z P)$ is valid too. Thus, the OMUF game succeeds with advantage q_{GGMCmp}/p. Hence, our bound is tight.

Proof. We consider an adversary $\mathcal{A}$ playing the OMUF game. Without loss of generality, we assume that $\mathcal{A}$ either aborts, or returns (b_j, t_j, P_j, Q_j) tuples which are all valid, with pairwise different t_j, same $b_j = b$, and the total number equals to $n_b + 1$.

Eliminating π. Before applying the GGM transforms, we first reduce to a game Γ_1 where $\mathcal{O}_{\mathsf{sign}}$ no longer computes a proof π. Instead, a proof

$$\pi = (C, e_0, e_1, a_0, a_1, a_d, a_\rho, a_w)$$

is simulated by the adversary. As only the adversary needs to query the random oracle, this oracle can be simulated by the adversary by lazy sampling. This gives the opportunity to program the oracle too.

To simulate the proof, the adversary picks $\mathcal{V} = (C, e_0, e_1, a_0, a_d, a_1, a_\rho, a_w)$ in $\mathbb{G} \times \mathbb{Z}_p^7$ uniformly at random. After selecting π, the adversary can follow the $\Pi_2.\mathsf{Verify}$ algorithm to compute $C_0, C_1, e, C_d, C_\rho, C_w$. Then, the adversary forms the input to the random oracle

$$q = (G, H, C_x, C_y, Z, U, V, t_S, T, C, C_0, C_1, C_d, C_\rho, C_w)$$

If q is already queried to Hash, the simulation fails and the game aborts. However, since π was randomly picked, this should happen with probability limited to $\frac{q_{\mathsf{Hash}}}{p^6}$.

In the way the normal proof is generated (see subsection 4.3), what is randomly selected is $\mathcal{W} = (\mu, e_{1-b}, a_{1-b}, r_\mu, r_d, r_\rho, r_w, e)$ where $e = \mathsf{Hash}(q)$, but there is a one-to-one mapping between $\mathcal{V}$ and $\mathcal{W}$. So, π is well distributed but the link with the random oracle is missing. Since $\mathsf{Hash}(q)$ was not queried before, the adversary can program the random oracle with $\mathsf{Hash}(q) = e$. Hence, the proof becomes valid. Except in the failure case, the proof is perfect.

$$\mathsf{Adv}^{\mathsf{OMUF}} \leq \mathsf{Adv}^{\Gamma_1} + \frac{q_{\mathsf{Hash}}}{p^6}$$

Finally, oracles no longer use H, C_x, C_y. We can reduce to a game where Setup_2 and KeyGen_2 are skipped. The adversary can run them and select H, C_x, and C_y randomly. This simulation is perfect. Hence we can now assume that the remaining Γ^1 game applies to a variant of ATHM with no H, C_x, C_y, and π.

In the ith query to $\mathcal{O}_{\mathsf{sign}}$, we let $(b, \mathsf{query}) = (b_i, T_i)$ denote the input and $(U_i, V_i, t_{S,i})$ denote the output.

Uniform secret values. Our next transform consists of modifying the game Γ_1 into a game Γ_2 in which y and z in KeyGen and every d in $\mathcal{O}_{\mathsf{sign}}$ queries are uniformly selected in $\mathbb{Z}_p$ instead of $\mathbb{Z}_p^*$. The failure case is when one random selection draws zero. By using the difference lemma, we obtain

$$\mathsf{Adv}^{\Gamma_1} \leq \mathsf{Adv}^{\Gamma_2} + \frac{n+2}{p}$$

GGM transforms. We apply the linearization transform in the GGM with base elements $(G, Z, (U_i, V_i)_i)$. For each group element $A = R[i']$, the adversary gets a vector $\mathsf{Vec}[i'] = (a_G, a_Z, (a_{U_i}, a_{V_i})_i, 0, 0, \ldots)$ such that

$$A = a_G.G + a_Z.Z + \sum_i (a_{U_i}.U_i + a_{V_i}.V_i)$$

The adversary has access to group elements in $R[\cdot]$ and $\mathsf{Base}[\cdot]$. For each of those element, the adversary knows a corresponding vector which we denote by Vec_A.

We then apply the algebraic transform in the GGM with the secret scalar values $\mathsf{Val} = (x, y, z, (d_i)_i)$ corresponding to the formal variables $\mathsf{Var} = (\bar{x}, \bar{y}, \bar{z}, (\bar{d}_i)_i)$. We recursively define the polynomials $P_\ell^{\mathsf{Base}}(\mathsf{Var})$ and $P_{i'}^{R}(\mathsf{Var})$. For convenience, we denote $\mathsf{Pol}_A(\mathsf{Var})$ the polynomial associated to each group element A for which the adversary has access. Clearly, we can set $\mathsf{Pol}_G(\mathsf{Var}) = 1$, $\mathsf{Pol}_Z(\mathsf{Var}) = \bar{z}$, $\mathsf{Pol}_{U_i}(\mathsf{Var}) = \bar{d}_i$ to ensure the fundamental property for these base elements. To define $\mathsf{Pol}_{V_i}(\mathsf{Var})$, we assume that $\mathsf{Pol}_{T_i}(\mathsf{Var})$ is defined (it is by linear combination of previous base elements) and we define

$$\mathsf{Pol}_{V_i}(\mathsf{Var}) = \bar{d}_i(\bar{x} + b_i\bar{y} + t_{S,i}\bar{z}) + \bar{d}_i.\mathsf{Pol}_{T_i}(\mathsf{Var})$$

By induction, for every group element A which is accessible by the adversary, we have $A = \mathsf{Pol}_A(\mathsf{Val}).G$. The formal polynomial Pol_A is known by the adversary but Val remains secret.

We prove by induction the following fact.

Fact 1 *For each A accessible by the adversary after q queries to $\mathcal{O}_{\mathsf{sign}}$, the Pol_A polynomial has total degree bounded by $q+1$. Furthermore, every partial degree is bounded by 1. Monomials are square-free.*

Indeed, after $q = 0$ queries, the largest degree is for $\mathsf{Pol}_Z(\mathsf{Var}) = \bar{z}$. Making a new query to $\mathcal{O}_{\mathsf{sign}}$ multiplies $\mathsf{Pol}_{T_i}(\mathsf{Var})$ by a fresh $\bar{d}_i$ and adds a degree-2 polynomial $\bar{d}_i.(\bar{x} + b_i\bar{y} + t_{S,i}\bar{z})$.

We can now use out GGM transforms to start the simulation of the GGMCmp oracle. First observe that queries to this oracle are only made by the adversary, and by the game calling $< AT.\mathsf{ReadBit}$ either in the $\mathcal{O}_{\mathsf{read}}$ or in the final verification of the forged tokens. By abuse of notation we define the polynomials associated to A and B in the query (A, B) to the oracle by the game. For a token (P, Q), which is provided by the adversary, we have $\mathsf{Pol}_A(\mathsf{Var}) = \mathsf{Pol}_Q(\mathsf{Var})$ and $\mathsf{Pol}_B(\mathsf{Var}) = (\bar{x} + b\bar{y} + t\bar{z})\mathsf{Pol}_P(\mathsf{Var})$, with b and t provided by the adversary.

If the call to the comparison oracle is made by the adversary, the polynomials have degree bounded by $q + 1$ so we obtain the result. Otherwise, the call must come from the usage of $\mathsf{AT.ReadBit}$ in either the game of the $\mathcal{O}_{\mathsf{read}}$ oracle, which multiply a degree-$(q+1)$-bounded polynomial by a degree-1 polynomial $\bar{x}+b\bar{y}+t\bar{z}$. So, the degree is bounded by $q + 2$. The values in Val are uniform in $\mathbb{Z}_p$. By using the Schwartz-Zippel lemma and the bound on the degree of polynomials, we have the following fact.

Fact 2 *Let q be the number of $\mathcal{O}_{\mathsf{sign}}$ queries before an input (A, B) is presented for the first time to a comparison oracle. Except with a probability bounded by $\frac{q+2}{p}$, we have $A = B$ if and only if $\mathsf{Pol}_A = \mathsf{Pol}_B$.*

The adversary, game, or $\mathcal{O}_{\mathsf{read}}$ oracle can simulate that comparison oracle by checking equality between $\mathsf{Pol}_A = \mathsf{Pol}_B$. Hence, by induction, using hybrids, we reduce to a game Γ_3 where no access to the comparison oracle is made, and for every final (b, t, P, Q) output, we have $\mathsf{Pol}_Q(\mathsf{Var}) = (\bar{x} + b\bar{y} + t\bar{z})\mathsf{Pol}_P(\mathsf{Var})$ (in winning cases). The total number of comparisons is bounded by $2(n + 1 + q_{\mathcal{O}_{\mathsf{read}}}) + q_{\mathsf{GGMCmp}}$, where $q_{\mathcal{O}_{\mathsf{read}}}$ is the number of calls to $\mathcal{O}_{\mathsf{read}}$ by the adversary and q_{GGMCmp} is the number of calls to the comparison by the adversary. We obtain

$$|\mathsf{Adv}^{\Gamma_2} - \mathsf{Adv}^{\Gamma_3}| \leq (2(n + 1 + q_{\mathcal{O}_{\mathsf{read}}}) + q_{\mathsf{GGMCmp}}) \times \frac{n+2}{p}$$

In Γ_3, no query to GGMCmp is made. So no information about Mem leaks. Hence, secret values become useless and the game becomes linear.

Analyzing the linear game. We now focus on one of the n_b+1 final (b, t, P, Q) produced by $\mathcal{A}$ in winning cases. Since the tokens are assumed to be valid and thanks to our previous transforms, we know that $\mathsf{Pol}_Q(\mathsf{Var}) = (\bar{x}+b\bar{y}+t\bar{z})\mathsf{Pol}_P(\mathsf{Var})$. $\mathcal{A}$ is only making scalar linear combinations of G, Z, and the U_i and V_i. We write

$$\begin{aligned} P &= a_G.G + a_Z.Z + \sum_i (a_{U_i}.U_i + a_{V_i}.V_i) \\ Q &= b_G.G + b_Z.Z + \sum_i (b_{U_i}.U_i + b_{V_i}.V_i) \\ &= (x + by + tz)P \end{aligned}$$

Thanks to Fact 1, the partial degree of Pol_Q in $\bar{z}$ is bounded by 1. We can see from the last equation $\mathsf{Pol}_Q = \mathsf{Pol}_P \times (\bar{x} + b\bar{y} + t\bar{z}))$ that $a_Z = 0$.

Pol_P has constant term a_G. From the last equation, Pol_Q has a_G as a coefficient of monomial $\bar{x}$. However, no monomial $\bar{x}$ can appear in the linear expression of Pol_Q. (For instance, Pol_{V_i} is always a multiple of $\bar{d}_i$.) Hence, $a_G = 0$.

Similarly, $\bar{x} + b\bar{y} + t\bar{z}$ has no constant term so we must have $b_G = 0$.

By inspecting the monomial $\bar{z}$ we now obtain that $b_Z = 0$.

Hence,

$$\begin{aligned} P &= \sum_i (a_{U_i}.U_i + a_{V_i}.V_i) \\ Q &= \sum_i (b_{U_i}.U_i + b_{V_i}.V_i) \\ &= (x + by + tz)P \end{aligned}$$

Given a polynomial and a variable $\bar{u}$, we say that $\bar{u}$ appears if the partial degree in $\bar{u}$ is at least 1. We have the following fact.

Fact 3 *$\bar{d}_i$ appears in Pol_P if and only if it appears in Pol_Q.*

Since the partial degree of Pol_Q in $\bar{x}$ is at most 1 due to Fact 1, the partial degree of Pol_P in $\bar{x}$ must be zero. Hence, a monomial μ is in Pol_P if and only if the monomial $\bar{x}\mu$ is in Pol_Q.

In the ith oracle call to $\mathcal{O}_{\mathsf{sign}}$, we have

$$\mathsf{Pol}_{V_i} = \bar{d}_i(\bar{x} + b_i\bar{y}) + t_{S,i}\bar{d}_i\bar{z} + \bar{d}_i \times \mathsf{Pol}_{T_i}$$

with $t_{S,i}$ sampled as a fresh uniform scalar. Note that $\bar{d}_i$ cannot appear in Pol_{T_i} because it is formed before sampling d_i. Pol_{T_i} may have the monomial $\bar{z}$ (it is actually supposed to) but Pol_{T_i} is set before sampling $t_{S,i}$. Hence, except with probability $\frac{1}{p}$, the $\bar{d}_i\bar{z}$ monomial is present in Pol_{V_i}. We deduce what follows.

Fact 4 *For any i, the monomial $\bar{d}_i\bar{z}$ has a nonzero coefficient in Pol_{V_i}, except with probability $\frac{1}{p}$.*

We reduce to a game Γ_4 where $\bar{d}_i\bar{z}$ never has a zero coefficient in Pol_{V_i} for every i. We have

$$\mathsf{Adv}^{\Gamma_3} \leq \mathsf{Adv}^{\Gamma_4} + \frac{n}{p}$$

Fact 5 *In Γ_4, among all group elements given to $\mathcal{A}$, the monomial $\bar{d}_i\bar{z}$ has a nonzero coefficient in Pol_{V_i} and only in Pol_{V_i}.*

Indeed, even though it could be put in a Pol_{T_j} for $j > i$, the monomial would be multiplied by $\bar{d}_j$ and thus $\bar{d}_i\bar{z}$ would not appear as a standalone monomial in Pol_{V_j}. Coming back to the representation of a final P and Q, we deduce that $a_{V_i} = 0$ for every i (as we cannot have $\bar{d}_i\bar{z}^2$ in Pol_Q). By writing $\mathsf{Pol}_{U_i} = \bar{d}_i$, Pol_P is linear in every $\bar{d}_i$. This implies that no monomial in a final Pol_Q can be divisible by any $\bar{d}_i\bar{d}_{i'}$.

We have $\mathsf{Pol}_{V_i} = (\bar{x} + b_i\bar{y} + t_{S,i}\bar{z})\bar{d}_i + \bar{d}_i\mathsf{Pol}_{T_i}$. For every i such that $b_{V_i} \neq 0$, we deduce that no $\bar{d}_{i'}$ appear in Pol_{T_i}. Hence, for every i such that $b_{V_i} \neq 0$, we have that T_i must be a known linear combination of G and Z. We write $T_i = t_{C,i}Z + r_iG$. Hence, $\mathsf{Pol}_{V_i} = \bar{d}_i(\bar{x} + b_i\bar{y} + (t_{C,i} + t_{S,i})\bar{z})$. By writing $t_i = t_{C,i} + t_{S,i}$, we have

$$\begin{aligned}
\mathsf{Pol}_P &= \sum_i a_{U_i}.\bar{d}_i \\
\mathsf{Pol}_Q &= \sum_i (b_{U_i} + b_{V_i}(\bar{x} + b_i\bar{y} + t_i\bar{z} + r_i))\bar{d}_i \\
&= \mathsf{Pol}_P \times (\bar{x} + b\bar{y} + t\bar{z}))
\end{aligned}$$

By inspecting $\bar{d}_i$ we can further see that we must have $b_{U_i} = -b_{V_i}r_i$ and $a_{U_i} = b_{V_i}$. Hence, $P = \sum_i a_{U_i}U_i$ and $Q = \sum_i a_{U_i}(V_i - r_iU_i)$.

We say that the ith query is well formed if Pol_{T_i} is a linear combination of Pol_G and Pol_Z (i.e. that Pol_{T_i} is a polynomial in $\bar{z}$ with degree bounded by 1: $\mathsf{Pol}_{T_i} = t_{C,i}\bar{z} + r_i$). For each well formed query we can define $t_i = t_{C,i} + t_{S,i}$. It follows that for every i such that $a_{U_i} \neq 0$, we have that the ith query is well formed and that $b_i = b$ and $t_i = t$. This proves that for any valid (b, t, P, Q), (P, Q) is a known linear combination of all $(U_i, V_i - r_iU_i)$ for well-formed queries satisfying $(b, t) = (b_i, t_i)$. Since P is nonzero, there exists i such that the ith query is well formed and $(b, t) = (b_i, t_i)$. Hence, the number of pairwise different t cannot exceed n_b. We deduce there is no winning case in Γ_4.

We can wrap up by collecting all Adv overheads to get an upper bound for $\mathsf{Adv}^{\mathsf{OMUF}}$. □

6.2 UNLINK Security in GGM and ROM

Theorem 2. ATHM *is* 2-UNLINK*-secure in the generic group and random oracle models. More precisely, for any n, given an* UNLINK_n*-adversary $\mathcal{A}$, we have*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{UNLINK}} \leq \frac{2}{n} + \frac{3 + n + q_{\mathsf{GGMCmp}} + 3q_{\mathsf{Hash}}}{p}$$

where q_{GGMCmp} and q_{Hash} are the number of queries to the GGMCmp *oracle and the random oracle.*

Perfect 2-UNLINK security is when the advantage is bounded by $\frac{2}{n}$. By doing q_{GGMCmp} queries, the adversary can compute the discrete logarithm td_2 of H with success probability q_{GGMCmp}/p. If it succeeds, the adversary can select n pairwise different secrets x_i and use the secret (x_i, y, z) in the ith query. Thanks to td_2, the Pedersen commitment C_x can equivocate to a commitment to x_i for each i. So, the proof Π_2 can be made and verified. Then, each token can be uniquely recognized and the game succeeds with advantage 1. If the discrete logarithm fails, hiding the bit $b = i \bmod 2$ and a random guess works with advantage $\frac{2}{n}$. So, the UNLINK_n game succeeds with advantage roughly $\frac{2}{n} + q_{\mathsf{GGMCmp}}/p$. Hence, our bound is tight.

Incidentally, if the setup is maliciously done and the server gets td_2, it is enough to run the above UNLINK attack. So unlinkability relies on a trusted setup. On the other hand, a malicious setup is only useful for a malicious server: it only harms unlinkability.

Proof. We start with an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ playing the UNLINK_n game. To win the game, there must exist $\mathcal{O}_{\mathsf{sign}}$ queries. We assume that $\mathsf{AT.ClientQuery}$ verifies pk during at least one query and aborts if not valid. Hence, we could add after the selection of pk by $\mathcal{A}_1$ an explicit verification of pk with abort if not valid. So, we can assume without loss of generality that pk is valid.

ROM. We first reduce to a game Γ_1 in which for every verification of pk (right after $\mathcal{A}_1$ produces pk) and every final verification of π (in every $\mathsf{AT.ClientFinal}$ instance at the end of the game, i.e. excluding those in $\mathcal{O}_{\mathsf{final}}$), the Hash query which is made to compute the challenge ε or e was done by the adversary before the verification. Clearly, if this is not the case, the probability that the corresponding verification succeeds is exactly $\frac{1}{p}$. Hence,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{UNLINK}} \leq \mathsf{Adv}_{\mathcal{A}}^{\Gamma_1} + \frac{1+n}{p}$$

Hence, we will be able to discuss on *when* the query was done by the first time by the adversary, when the challenge was unknown and the input to the hash were committed.

GGM. We modify the $\mathcal{O}_{\mathsf{final}}$ oracle so that it would return t_C, r, and c for the issuance session which is finalized and the token revealed. As it gives more information, the winning probability does not decrease. It further helps in the GGM transform by removing from the base elements the group elements provided by the client in the revealed sessions. Hence, we only take as base elements the values of G, H, and T_i from the $\mathcal{O}_{\mathsf{query}}$ responses which have not been finalized. Note that the basis contains independent and uniform elements (with nonzero G and H). We apply the linearization with the remaining basis.

Next, we reduce to a game Γ_2 in which every group equation check in every verification of pk (right after $\mathcal{A}_1$ produces pk) and every group equation check in the final verifications of π (in every $\mathsf{AT.ClientFinal}$ instance at the end of the game) are replaced by their corresponding vectorial check. A failure case would give a non-trivial linear relation between base elements, which would solve the discrete logarithm problem. Such success happens with probability bounded by $\frac{q_{\mathsf{GGMCmp}}+1}{p}$ in GGM [17].

$$\mathsf{Adv}_{\mathcal{A}}^{\Gamma_1} \leq \mathsf{Adv}_{\mathcal{A}}^{\Gamma_2} + \frac{1}{p} + \frac{q_{\mathsf{GGMCmp}}+1}{p}$$

Extraction of the secret. We first extract x, r_x, y, r_y, z from $\mathcal{A}_1$ as follows. We use the generic group model and the linearization transform. In GGM, the base elements for $\mathcal{A}_1$ are G and H. So, the elements made by $\mathcal{A}_2$ such as C_x, C_y, and

Z have a linear expression in G and H. Such linearization step clearly extracts x, r_x, y, r_y, z, r_z such that $C_x = xG + r_xH$, $C_y = yG + r_yH$, and $Z = zG + r_zH$. We only have to prove that $r_z = 0$ except with negligible probability. When $r_z = 0$ we say that Z is *proportional* to G by abuse of language.

Thanks to the verification condition, we have $\varepsilon = \mathsf{Hash}(\mathsf{query})$ with $\mathsf{query} = (G, H, Z, \Gamma_z)$ and $\Gamma_z + \varepsilon Z = a_zG$. Due to the Γ_2 reduction, we deduce that $\Gamma_z + \varepsilon Z$ is proportional to G in the vectorial sense.

We reduce to a game Γ_3 in which for every hash query of form (G, H, A, B) by $\mathcal{A}_1$ with $B + \mathsf{Hash}(G, H, A, B)A$ proportional to G, then A and B are also proportional to G. We analyze the probability of the failure case. The first time (G, H, A, B) is queried to the random oracle, A and B are committed and the hash $\mathsf{Hash}(G, H, A, B)$ is random and uniform. Hence, the probability of the failure case is bounded by $\frac{1}{p}$. Hence

$$\mathsf{Adv}_{\mathcal{A}}^{\Gamma_2} \leq \mathsf{Adv}_{\mathcal{A}}^{\Gamma_3} + \frac{q_{\mathsf{Hash}}}{p}$$

In our case, $\Gamma_z + \mathsf{Hash}(G, H, Z, \Gamma_z)Z$ is proportional to G. Hence, both Z and Γ_z are proportional to G in Γ_3. In Γ_3, we can now assume that sk is extracted and that $(\mathsf{pk}, \mathsf{sk})$ is valid.

Extraction of b. For every $\mathsf{AT.ClientFinal}$ instance, the input resp from $\mathcal{A}_2$ parses as $\mathsf{resp} = (U, V, t_S, \pi)$ and $\pi_i = (C, e_0, e_1, a_0, a_1, a_d, a_\rho, a_w)$. Following the verification procedure of π defines $C_0, C_1, e, C_d, C_\rho, C_w$ and the query

$$q = (G, H, C_x, C_y, Z, U, V, t_S, T, C, C_0, C_1, C_d, C_\rho, C_w)$$

to Hash. We want to extract b and d from it.

Thanks to the verification condition, we have $e = \mathsf{Hash}(q)$ with $C_0 = a_0H - e_0C$ and $C_1 = a_1H - e_1(C - C_y)$. Due to the Γ_2 reduction, these last two relations hold in a vectorial sense.

We reduce to a game Γ_4 in which for every hash query of the form of q by $\mathcal{A}_2$ with neither C nor $C - C_y$ being proportional to H, with C_0 being a linear combination of H and C, and with C_1 being a linear combination of H and $C - C_y$, we have $\mathsf{Hash}(q) \neq e'_0 + e'_1$ where $C_0 = a'_0H - e'_0C$ and $C_1 = a'_1H - e'_1(C - C_y)$ are the unique linear relations. Clearly, the failure case has probability bounded by $\frac{1}{p}$ for each Hash query. Hence

$$\mathsf{Adv}_{\mathcal{A}}^{\Gamma_3} \leq \mathsf{Adv}_{\mathcal{A}}^{\Gamma_4} + \frac{q_{\mathsf{Hash}}}{p}$$

In our case, we have $e = \mathsf{Hash}(q)$ with $C_0 = a_0H - e_0C$ and $C_1 = a_1H - e_1(C - C_y)$ and the query $\mathsf{Hash}(q)$ was made. So, either C or $C - C_y$ is proportional to H, which gives b. (If both C and $C - C_y$ are proportional to H, we select b arbitrarily.) In Γ_4, we can now assume that b and μ such that $C = bC_y + \mu H$ are extracted for each final $\mathsf{AT.ClientFinal}$ instance.

Extraction of (d', ρ, w). We continue the analysis of the $\mathsf{AT.ClientFinal}$ instance by observing that we must have

$$\begin{pmatrix} C_d \\ C_\rho \\ C_w \end{pmatrix} - e \begin{pmatrix} G \\ C_x + C + t_S Z + T \\ T \end{pmatrix} \in \left\langle \begin{pmatrix} U \\ V \\ V \end{pmatrix}, \begin{pmatrix} 0 \\ H \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ G \end{pmatrix} \right\rangle$$

in a vectorial sense, which we write $\mathcal{V}_1 - e\mathcal{V}_2 \in \langle \mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5 \rangle$ for short, where $\langle \mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5 \rangle$ denotes the linear span of $\mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5$. We reduce to a game Γ_5 in which for every hash query of the form of q by $\mathcal{A}_2$ with $\mathcal{V}_1 - \mathsf{Hash}(q)\mathcal{V}_2 \in \langle \mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5 \rangle$, then $\mathcal{V}_2 \in \langle \mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5 \rangle$. We analyze the probability of the failure case. Again, the probability of the failure case is bounded by $\frac{1}{p}$. Hence

$$\mathsf{Adv}_{\mathcal{A}}^{\Gamma_4} \leq \mathsf{Adv}_{\mathcal{A}}^{\Gamma_5} + \frac{q_{\mathsf{Hash}}}{p}$$

Since $\mathcal{V}_1 - \mathsf{Hash}(q)\mathcal{V}_2 \in \langle \mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5 \rangle$ and $\mathsf{Hash}(q)$ was queried before verification, we deduce that $\mathcal{V}_2 \in \langle \mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5 \rangle$. Hence we can write $-\mathcal{V}_2 = d'\mathcal{V}_3 + \rho\mathcal{V}_4 + w\mathcal{V}_5$, which is Equation 1. We easily deduce d such that $U = dG$ and $V = d((x + by + t_S z)G + T)$.

Information theoretic argument. The rest of the proof is an information theoretic argument for which complexities do not matter. Given $(x, y, z, T_i, U_i, V_i, t_{i,S}, \pi_i)$ we can uniquely determine b_i. The variables $t_{i,C}$ and r_i are uniform but linked by $T_i = t_{i,C} Z + r_i G$ and c_i is still independent and uniform. Since $t_i = t_{i,C} + t_{i,S}$ and $P_i = c_i U_i$, we have that $(t_i, P_i) | (x, y, z, T_i, U_i, V_i, t_{i,S}, \pi_i)$ is uniformly distributed as a pair composed of a scalar and a nonzero group element. Hence, whenever $\mathcal{A}_2$ returns $\mathcal{Q}$ and the list of resp_i, it determines the values of the b_i but the (t_i, P_i) to be released are still uniform. After permutation, $(t_{\sigma(i)}, P_{\sigma(i)}, Q_{\sigma(i)})$ has a value of $Q_{\sigma(i)}$ which is imposed by $Q_{\sigma(i)} = (x + b_{\sigma(i)} y + t_{\sigma(i)} z) P_{\sigma(i)}$ so brings $b_{\sigma(i)}$ as only information.

This reduces to the following game: the adversary chooses a list of bits $(b_i)_{i \in \mathcal{Q}}$ with $\#\mathcal{Q} \geq n$, the game selects a random i^* and a random permutation σ then provides b_{i^*} and $(b_{\sigma(i)})_{i \in \mathcal{Q}}$ to the adversary, and the adversary finally makes a guess i and win if and only if $i = i^*$. If the adversary puts n_0 zeros and n_1 ones, the adversary can only win with probability $\frac{1}{n_0}$ when it is a zero (which happens with probability $\frac{n_0}{n_0+n_1}$), and with probability $\frac{1}{n_1}$ when it is a one (which happens with probability $\frac{n_1}{n_0+n_1}$). Overall, the adversary wins with probability $\frac{2}{n_0+n_1}$ which is at most $\frac{2}{n}$ since $n_0 + n_1 = \#\mathcal{Q} \geq n$. □

6.3 PMB Security in GGM and ROM

Theorem 3. *For every $\mathcal{A}$ playing* PMB *in the generic group model and making n oracle calls to $\mathcal{O}_{\mathsf{sign}}$, if Π_2 is perfectly simulatable, we have*

$$\mathsf{Adv}^{\mathsf{PMB}} \leq 2(2(n + 1 + q_{\mathcal{O}_{\mathsf{read}}}) + q_{\mathsf{GGMCmp}} + 1) \times \frac{n+2}{p} + \frac{q_{\mathsf{Hash}}}{p^6}$$

where $q_{\mathcal{O}_{\mathsf{read}}}$, q_{GGMCmp}, and q_{Hash} are the number of queries to $\mathcal{O}_{\mathsf{read}}$, GGMCmp, and to the random oracle.

Again, the bound is tight. By computing the logarithm of Z, an adversary can modify an existing valid token to make it a valid token with a chosen t. Two tokens with same t can be combined into a single token which is valid if and only if the hidden bits are equal. Then we have a similar attack as the one we presented on PMBT.

Proof. We now consider an adversary $\mathcal{A}$ playing the PMB_{b^*} game.

The only new element in the generic group model treatment is that there is a new variable $\bar{b}^*$ appearing in polynomials. This variable appears as soon as $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{chal}}$. It appears as a term of form $\mathsf{Pol}_{V_{i^*}} = (\bar{x} + \bar{b}^*\bar{y} + t_{S,i^*}\bar{z} + \mathsf{Pol}_{T_{i^*}})\bar{d}_{i^*}$, where i^* is the index number of the $\mathcal{O}$ query for the $\mathcal{O}_{\mathsf{chal}}$ query.

We treat the variable $\bar{b}^*$ differently than others from Var (as $\bar{b}^*$ takes random values in $\{0,1\}$ instead of random values in $\mathbb{Z}_p$ like other variables.) For each polynomial $\mathsf{Pol}_A(\bar{b}^*, \mathsf{Var})$ in which $\bar{b}^*$ appears, we can make two partial evaluations $\mathsf{Pol}_A(0, \mathsf{Var})$ and $\mathsf{Pol}_A(1, \mathsf{Var})$ corresponding to $\bar{b}^* = 0$ and $\bar{b}^* = 1$. We obtain two polynomials with no $\bar{b}^*$ variable. Hence, this increase the number of polynomials by a factor at most 2.

We first proceed like for OMUF security with games Γ_1 to have no π in the return from the issuer and Γ_2 to sample secrets in $\mathbb{Z}_p$. The transition to Γ_3 to get rid of the GGMCmp oracle (and thus of the $\mathcal{O}_{\mathsf{read}}$ and $\mathcal{O}_{\mathsf{valid}}$ oracles) is also using hybrid arguments but is more complicated. Oracles are simulated in the order they are called. For the simulation of GGMCmp and $\mathcal{O}_{\mathsf{read}}$ until the challenge is made, it works like for OMUF security. After the challenge is made, the $\mathsf{GGMCmp}(A, B)$ made by the adversary are simulated by answering 1 if and only if there exists $\beta \in \{0,1\}$ such that $\mathsf{Pol}_A - \mathsf{Pol}_B$ vanishes in the partial evaluation $\bar{b}^* = \beta$. To simulate $\mathcal{O}_{\mathsf{valid}}(t, P, Q)$ (which is the last one to use GGMCmp), the answer is 1 if and only if there exists $b, \beta \in \{0,1\}$ such that $\mathsf{Pol}_Q - (\bar{x} + b\bar{y} + t\bar{z}) \times \mathsf{Pol}_P$ vanishes in the partial evaluation $\bar{b}^* = \beta$. In the first case, we prove the following variant of Fact 2.

Fact 6 *Let q is the number of $\mathcal{O}_{\mathsf{sign}}$ queries (including $\mathcal{O}_{\mathsf{chal}}$) before an input (A, B) is presented for the first time to the GGMCmp oracle. We assume that the call is made by the adversary. Except with a probability bounded by $\frac{q+2}{p}$, we have $A = B$ if and only if $\mathsf{Pol}_A(0, \mathsf{Var}) = \mathsf{Pol}_B(0, \mathsf{Var})$ or $\mathsf{Pol}_A(1, \mathsf{Var}) = \mathsf{Pol}_B(1, \mathsf{Var})$.*

The direct implication works like in Fact 2 as the match happens for $\bar{b}^* = b^*$. We now want to show the converse implication: if $\mathsf{Pol}_A - \mathsf{Pol}_B$ vanishes for $\bar{b}^* = \beta$ with $\beta \in \{0,1\}$, then we want to prove $A = B$.

Thanks to the generic group model, the adversary knows a linear combination of provided group elements to obtain $A - B$. Let λ_i and μ_i be the coefficients of U_i and V_i respectively. We let i be the largest index such that $(\lambda_i, \mu_i) \neq (0,0)$. The partial derivative of Pol_{A-B} with respect to $\bar{d}_i$ is $\lambda_i + \mu_i(\bar{x} + b_i\bar{y} + t_{S,i}\bar{z} + \mathsf{Pol}_{T_i})$ (with b_i replaced by $\bar{b}^*$ in the $i = i^*$ case). Since $\mathsf{Pol}_{A-B}(\beta, \mathsf{Var}) = 0$, the partial derivative vanishes for $\bar{b}^* = \beta$ too:

$$\lambda_i + \mu_i(\bar{x} + b_i\bar{y} + t_{S,i}\bar{z} + \mathsf{Pol}_{T_i}(\beta, \mathsf{Var})) = 0$$

The adversary knows how to express T_i as a linear combination of provided group elements. We notice that the monomials $\bar{x}$ and $\bar{b}^*\bar{x}$ are in no polynomial of any group element. So, there is no way to make a T_i such that $\mathsf{Pol}_{T_i}(\beta, \mathsf{Var})$ has a monomial $\bar{x}$. Hence, it cannot cancel $\bar{x}$. This implies $\mu_i = 0$ then $\lambda_i = 0$ which contradicts $(\lambda_i, \mu_i) \neq (0, 0)$. We deduce that $A - B$ has no U_i and V_i as a component in the linear combination. Hence, it does not depend on b^* and the result is trivial: if $\mathsf{Pol}_A - \mathsf{Pol}_B$ vanishes for $\bar{b}^* = \beta$, it vanishes with the partial evaluation $\bar{b}^* = b^*$ too. Hence, we can apply Fact 2 and conclude. We can simulate the first GGMCmp oracle call, if it is made by the adversary. We now look at what happens it it is made by $\mathcal{O}_{\mathsf{valid}}$.

To simulate $\mathcal{O}_{\mathsf{valid}}$, we use the following fact.

Fact 7 *Let q is the number of $\mathcal{O}_{\mathsf{sign}}$ queries (including $\mathcal{O}_{\mathsf{chal}}$) before a $\mathcal{O}_{\mathsf{valid}}(t, P, Q)$ call is made for the first time. We assume there is no GGMCmp call before. Except with a probability bounded by $\frac{q+2}{p}$, the oracle returns 1 if and only if there exists $b, \beta \in \{0, 1\}$ such that $\mathsf{Pol}_Q(\beta, \mathsf{Var}) = (\bar{x} + b\bar{y} + t\bar{z})\mathsf{Pol}_P(\beta, \mathsf{Var})$.*

The direct implication uses Fact 2 with $\beta = b^*$ and the right b which makes (b, t, P, Q) valid. For the converse implication, we assume that the polynomial equality is verified for a given (b, β) pair of bits.

We let $a_{U_i}, a_{V_i}, b_{U_i}, b_{V_i}$ be the coefficients of U_i and V_i for P and of U_i and V_i for Q. Let $\lambda_i = b_{U_i} - (\bar{x} + b\bar{y} + t\bar{z})a_{U_i}$ and $\mu_i = b_{V_i} - (\bar{x} + b\bar{y} + t\bar{z})a_{V_i}$. Like in the previous case, let i be the largest index such that (λ_i, μ_i) is nonzero. The partial derivative of $\mathsf{Pol}_Q - (\bar{x} + b\bar{y} + t\bar{z})\mathsf{Pol}_P$ in terms of $\bar{d}_i$ is again $\lambda_i + \mu_i(\bar{x} + b_i\bar{y} + t_{S,i}\bar{z} + \mathsf{Pol}_{T_i})$ (with b_i replaced by $\bar{b}^*$ in the $i = i^*$ case). We know it vanishes when evaluated on $\bar{b}^* = \beta$. Clearly, a_{V_i} must be zero (otherwise, $\bar{x}^2$ appears and cannot vanish). The same argument about the monomial $\bar{x}$ and also about the monomial $\bar{y}$ implies that $(\bar{x} + b\bar{y})a_{U_i} = (\bar{x} + b_i\bar{y})b_{V_i}$. Hence,

$$\frac{\partial \mathsf{Pol}_{\Delta}}{\partial \bar{d}_i}(\bar{b}^*, \mathsf{Var}) = b_{U_i} + b_{V_i}\left((b_i - b)\bar{y} + (t_{S,i} - t)\bar{z} + \mathsf{Pol}_{T_i}(\bar{b}^*, \mathsf{Var})\right)$$

for $\Delta = Q - (x + by + tz)P$. The adversary also knows a linear combination of T_i in terms of the provided group elements. Let λ'_j and μ'_j be the coefficients of U_j and V_j and let j be the largest index for which they are nonzero. The second partial derivative gives

$$\frac{\partial^2 \mathsf{Pol}_{A-B}}{\partial \bar{d}_j\, \partial \bar{d}_i}(\bar{b}^*, \mathsf{Var}) = b_{U_i}\left(\lambda_j + \mu_j(\bar{x} + b_j\bar{y} + t_j\bar{z} + \mathsf{Pol}_{T_j}(\bar{b}^*, \mathsf{Var}))\right)$$

which should vanish for $\bar{b}^* = \beta$. The same argument than before shows that $\bar{x}$ cannot disappear. Hence, T_i cannot have components in U_j or V_j and does not depend on b^*.

Knowing that T_i has no component is any previous U_i of V_i, we can go to the second largest i such that (λ_i, μ_i) is nonzero and look at the partial derivative with respect to $\bar{d}_i$ (which we know does not appear in a subsequent $V_{i'}$). We obtain the same result that T_i does not have any component in U_j or V_j. We analyze like this all components in every (U_i, V_i) of $\mathsf{Pol}_Q - (\bar{x} + b\bar{y} + t\bar{z})\mathsf{Pol}_P$. Since it vanished on $\bar{b}^* = \beta$, we cannot have any other component. Hence

$$\mathsf{Pol}_Q - (\bar{x}+b\bar{y}+t\bar{z})\mathsf{Pol}_P = \sum_i \left(b_{U_i} + b_{V_i}\left((b_i - b)\bar{y} + (t_{S,i} - t)\bar{z} + \mathsf{Pol}_{T_i}(\bar{b}^*, \mathsf{Var})\right)\right)\bar{d}_i$$

with b_{i^*} to be replaced by $\bar{b}^*$. For all indices i of nonzero terms, we deduce that $b = b_i$ and T_i is or form $T_i = t_{C,i}Z + r_iG$. This boils down to

$$\mathsf{Pol}_Q - (\bar{x} + b\bar{y} + t\bar{z})\mathsf{Pol}_P = b_{V_{i^*}}(\bar{b}^* - b)\bar{y}\bar{d}_{i^*} + \sum_{i \neq i^*} b_{V_i}(b_i - b)\bar{y}\bar{d}_i$$

If $b_{V_{i^*}} = 0$, this vanishing for $\bar{b}^* = \beta$ implies vanishing for $\bar{b}^* = b^*$ too, so we can apply Fact 2 to deduce that $Q = (x + by + tz)P$ most of the cases. If $b_{V_{i^*}} \neq 0$, this vanishing for $\bar{b}^* = \beta$ implies $b = \beta$. We can then see that it vanishes for $\bar{b}^* = b^*$ and $b = b^*$ too. We apply Fact 2 to deduce that $Q = (x + \beta y + tz)P$ most of the cases.

Using hybrids, we obtain

$$|\Pr[\mathsf{PMB}_{b^*} \to 1] - \Pr[\Gamma_{3,b^*} \to 1]| \leq (2(n+1+q_{\mathcal{O}_{\mathsf{read}}}) + q_{\mathsf{GGMCmp}} + 1) \times \frac{n+2}{p} + \frac{q_{\mathsf{Hash}}}{p^6}$$

In the game Γ_3, the oracles $\mathcal{O}_{\mathsf{verify}}$, $\mathcal{O}_{\mathsf{read}}$, and GGMCmp are not used any more.

After getting rid of $\mathcal{O}_{\mathsf{verify}}$ and $\mathcal{O}_{\mathsf{read}}$, we obtain a game in which no information about b^* is given to the adversary. Thus,

$$\Pr[\Gamma_{3,0} \to 1] = \Pr[\Gamma_{3,1} \to 1]$$

□

7 Conclusion

In our work, we studied the anonymous tokens with hidden metadata bit from issuer to the verifier. We started our studies with a security weakness in a protocol from CRYPTO 2020 which is an extension of Privacy Pass from PoPETs 2018. The protocol is based on oblivious PRFs. We discussed the real-world implications of the security notions defined in their protocol and showed that such problems can be overcome with a new token protocol from algebraic MACs. We defined the security with more natural and strong notions.

We believe algebraic MACs suit better in anonymous tokens with hidden bit as a primitive. However, it is an open question to understand if there are other OPRFs or another new primitive that would help designing anonymous tokens with strong security and privacy guarantees.

Acknowledgements. We heartily thank Greg Zaverucha, Michele Orrù, and Nirvan Tyagi for the insightful discussions and fruitful comments on the early version of the paper. We are also very thankful to Kim Laine, Radames Cruz Moreno, and Wei Dai for their valuable time and help with the implementation of the protocols and benchmarks in Rust.

References

1. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous Credentials Light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, pages 1087–1098. ACM, 2013.
2. Fabrice Benhamouda, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Publicly verifiable anonymous tokens with private metadata bit. Cryptology ePrint Archive, Paper 2022/004, 2022. `https://eprint.iacr.org/2022/004`.
3. Jan Camenisch and Anna Lysyanskaya. An Efficient System for Non-Transferable Anonymous Credentials with Optional Anonymity Revocation. In *EUROCRYPT*. Springer International Publishing, 2001.
4. Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, pages 268–289. Springer Berlin Heidelberg, 2003.
5. Pyrros Chaidos and Jens Groth. Making sigma-protocols non-interactive without random oracles. In *Public-Key Cryptography – PKC*, pages 650–670. Springer International Publishing, 2015.
6. Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and Keyed-Verification Anonymous Credentials. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 1205–1216. Association for Computing Machinery, 2014.
7. Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and Keyed-Verification Anonymous Credentials. https://eprint.iacr.org/2013/516, 2014.
8. Melissa Chase, Trevor Perrin, and Greg Zaverucha. *The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption*, page 1445–1459. Association for Computing Machinery, 2020.
9. Melissa Chase, Trevor Perrin, and Greg Zaverucha. The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption. https://eprint.iacr.org/2019/1416, 2020.
10. David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology – CRYPTO*. Springer International Publishing, 1982.
11. Ivan Damgård. On Σ-protocol, 2010.
12. Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 445–456, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
13. Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy Pass: Bypassing Internet Challenges Anonymously. In *PoPETs*, pages 164–180, 2018.
14. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.
15. Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous Tokens with Private Metadata Bit. https://eprint.iacr.org/2020/072.
16. Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous Tokens with Private Metadata Bit. In *Advances in Cryptology – CRYPTO*, pages 308–336. Springer International Publishing, 2020.

17. Ueli Maurer. Abstract Models of Computation in Cryptography. In Nigel P. Smart, editor, *Cryptography and Coding*, pages 1–12, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
18. Christian Paquin and Greg Zaverucha. U-Prove Cryptographic Specification v1.1 (Revision 3), December 2013. Released under the Open Specification Promise (http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx).
19. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1991.
20. Claus Peter Schnorr. Efficient identification and signatures for smart cards. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 239–252, New York, NY, USA, 1989. Springer-Verlag New York, Inc.
21. Victor Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1997.
22. Tjerand Silde and Martin Strand. Anonymous Tokens with Public Metadata and Applications to Private Contact Tracing. https://fc22.ifca.ai/preproceedings/40.pdf.
23. Akira Takahashi and Greg Zaverucha. Verifiable encryption from MPC-in-the-head. https://eprint.iacr.org/2021/1704, 2021.
24. Stefano Tessaro and Chenzhi Zhu. Short Pairing-Free Blind Signatures with Exponential Security. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022*, volume 13276 of *Lecture Notes in Computer Science*, pages 782–811. Springer, 2022.
25. Trust Tokens API. https://developer.chrome.com/docs/privacy-sandbox/trust-tokens/.

A More Attacks

We extend here the previously presented attack.

A.1 PMBT

The previous attack generalizes. We consider an adversary who collected n tokens with distinct t_i's, $(t_1, S_1, W_1), \ldots (t_n, S_n, W_n)$, satisfying $W_i = x_{b_i} H(t_i) + y_{b_i} S_i$. Assuming an oracle returning whether $\mathsf{AT.ReadBit}$ returns $\perp$ or not, we describe an attack allowing the adversary to check whether a subset of the collected tokens hide the same bit or not. Let I be a subset of $\{1, \ldots, n\}$. The adversary chooses a random t^* and runs the token request protocol one more time, but this time uses $T^* = H(t^*) - \sum_{i \in I} H(t_i)$ instead of $T^* = H(t^*)$, and obtains (S^*, W^*) such that $W^* = x_{b^*} T^* + y_{b^*} S^*$. Now, the adversary attempts to redeem token $(t^*, S^* + \sum S_i, W^* + \sum W_i)$. This will be a valid token iff all of the b_i for $i \in I$ are equal to b^*. If $\mathsf{AT.ReadBit}$ does not return $\perp$, the adversary deduces that all b_i for $i \in I$ are equal. If $\mathsf{AT.ReadBit}$ returns $\perp$, the adversary may try again, because it could still be the case that all b_i are equal but that b^* was not. Eventually, the adversary infers information and can apply a cut-and-choose strategy.

As conclusion, the attack method does not violate the unforgeability game by rule (that t's must be different) and does not violate the privacy game by rule (that there is no validity oracle). As acknowledged by the authors [16, Section 6.1], having Verify returning always true and pushing the validity to ReadBit not returning $\perp$ is meaningless. They propose a way to enable token verification in the CMBT scheme which is presented in the full version of the paper [15, Appendix J].

A.2 The CMBT Fix

The CMBT protocol is presented in the eprint version of the paper [15]. We summarize this protocol in Figure 7.

CMBT was updated[12] on April 21, 2022 [15] as shown in Figure 7. A token $(S, W, \tilde{W})$ is verified and reads bit b if $\tilde{W} = \tilde{x} H_t(t) + \tilde{y} S$ (verify part) and $W = x_b H_t(t) + y_b S \neq x_{1-b} H_t(t) + y_{1-b} S$ (read part). Unforgeability is enforced by the security of previous constructions. However, the privacy of metadata bit is still in question when we do not separate Verify and ReadBit as we will detail next.

Clearly, a linear combination attack with tokens sharing the same t will forge new tokens making $\mathsf{AT.Verify}$ true and $\mathsf{AT.ReadBit}$ returning the common bit to all tokens or $\perp$ if there is no unanimous bit b. Hence, privacy is broken with access to an oracle telling whether a token is well formed (in the sense of Verify).

[12] In the January 13, 2021 version (20210113:200918) of the paper, $\tilde{W}'$ was returned by the issuer without the $\tilde{\pi}$ proof, so a malicious issuer could hide a marker in it and break unlinkability. The authors corrected the protocol on April 21, 2022 after our disclosure.

```
$\mathsf{CMBT.Client}(\mathsf{pp}, t)$                                   $\mathsf{CMBT.IssueToken}(\mathsf{pp}, \mathsf{sk}, b)$
input: $(\mathsf{pp}, t)$                                              input: $\mathsf{pp}, \mathsf{sk}, b$
output: $\sigma$                                                       output: $\{\}$
$r \leftarrow\!\!\$\ \mathbb{Z}_p^*$
$T := H_t(t)$
$T' := r^{-1}T$
                                   $\xrightarrow{T'}$
                                                                       $(X_0, X_1, \tilde{X}) := \mathsf{pp}$
                                                                       $((x_0, y_0), (x_1, y_1), (\tilde{x}, \tilde{y})) := \mathsf{sk}$
                                                                       $s \leftarrow\!\!\$\ \{0,1\}^\lambda$, $S' := H_s(T', s)$
                                                                       $W' := x_b T' + y_b S'$
                                                                       $\tilde{W}' := \tilde{x}T' + \tilde{y}S'$
                                                     $\pi \leftarrow \mathit{\Pi}.\mathsf{Prove}((X_0, X_1, T', S', W'), (b, x_b, y_b))$
                                                           $\tilde{\pi} \leftarrow \mathit{\Pi}.\mathsf{Prove}((\tilde{X}, T', S', \tilde{W}'), (\tilde{x}, \tilde{y}))$
                                   $\xleftarrow{(s, W', \tilde{W}', \pi, \tilde{\pi})}$
$(X_0, X_1, \tilde{X}) := \mathsf{pp}$
$S' := H_s(T', s)$
if not $\mathit{\Pi}.\mathsf{Verify}((X_0, X_1, T', S', W'), \pi)$ or not $\mathit{\Pi}.\mathsf{Verify}((\tilde{X}, T', S', \tilde{W}'), \tilde{\pi})$ then return $\bot$
$S := rS'$
$W := rW'$, $\tilde{W} := r\tilde{W}'$
output: $\sigma := (S, W, \tilde{W})$
```

Fig. 7. CMBT token issuance protocol as given in [15, Fig 22, p 55] (current version 20220421:171853).

As another attack, if a client engages in a protocol to issue three tokens $(t_i, S_i, W_i, \tilde{W}_i)$, $i = 1, 2, 3$ with $t_1 = t_2$ and hidden metadata bits b_i, respectively, from the verification equations, the client can compute $\mathit{\Delta} S = S_2 - S_1$, $\mathit{\Delta}\tilde{W} = \tilde{W}_2 - \tilde{W}_1 = \tilde{y}\mathit{\Delta} S$, and $\mathit{\Delta} W = W_2 - W_1$. Let us now forge $(t_4, S_4, W_4, \tilde{W}_4) = (t_3, S_3 + \mathit{\Delta} S, W_3 + \mathit{\Delta} W, \tilde{W}_3 + \mathit{\Delta}\tilde{W})$. This is a token for which Verify returns true. Note that this does not violate the unforgeability result because the security game does not take as a valid forgery the creation of a new token with the same tag $t_3 = t_4$. However, ReadBit returns a bit if and only if $b_1 = b_2 = b_3$. In other words, either this new token is taken as valid but returns no bit, or it is fully valid and hides the same bit as all other tokens. Hence, having access to an oracle telling whether a bit is returned or not breaks the privacy of the metadata bit.

We can draw the following conclusion: if a side channel attack gives access to a *validity* oracle (i.e. whether AT.ReadBit returns $\bot$ or not once it is known that AT.Verify is true), we have an attack against the privacy of the metadata bit b.

A.3 Anonymous Tokens with Public Metadata

Silde and Strand [22] design a protocol to add *public* metadata in Privacy Pass while also extending it for a private metadata bit and for public verifiability of the token. They consider a case study with the reporting phase in digital contact tracing when a user reports proximity keys with an anonymous token from the authority who has the positive test. To avoid changing credential secret keys every day, they add the date in public metadata md of the token.

Their scheme with private metadata is adapted from CMBT. It has similar problems as the ones we explained in the previous section. Moreover, the redemption scheme verifies validity and reads the hidden bit at the same time. However, given two tokens (S_i, W_i) $(i = 1, 2)$ with the same (t, md, b), a combination of the two tokens is a valid token for (t, md, b) as well while two tokens with the same (t, md) and different b, a combination is a token which reads no bit. The PMB game for the privacy of the metadata bit is specified with a $\mathcal{O}_{\mathsf{verify}}$ oracle but it is not clear what this oracle answers for this scheme. If $\mathcal{O}_{\mathsf{verify}}$ always returns true, this attack does not contradict the security result. However, if $\mathcal{O}_{\mathsf{verify}}$ returns whether the token reads any bit (as it should do) then PMB security is broken.

B ATHM: Anonymous Token with Hidden Metadata

The full version of ATHM includes several options. First of all, we add the public metadata attribute m. It can, for instance, include a coarse expiration data. It should be used carefully as it degrates unlinkability. A token with a hidden bit b will be a tuple (t, m, σ) such that σ is a valid MAC for the attributes (b, m, t).

The objective of other options is to get rid of the dependency to the generic group model and possibly have provable security in the standard model. For that, one option is the choice of the MAC. A valid MAC for an input b, a public metadata m, a nonce t, and a secret key $(\vec{x}, \vec{y}, \vec{y'}, \vec{z})$ is a pair $\sigma = (P, \vec{Q})$ such that $\vec{Q} = (\vec{x} + b\vec{y} + m\vec{y'} + t\vec{z})P$. As detailed below, several options can be made for the MAC algorithm in which case the secrets $\vec{x}$, $\vec{y}$, $\vec{y'}$, and $\vec{z}$ will be vectors of various dimension and structure with elements in $\mathbb{Z}_p$.[13]

Finally, we also have options for the choice of proof schemes Π_1 and Π_2. The proof Π_1 is an extractable proof made by the client and Π_2 is the simulatable proof made by the issuer.

Options are summarized in Table 2.

B.1 The ATHM Components

Our scheme uses as a building block an extractable non-interactive proof Π_1 and a simulatable non-interactive proof Π_2. Several options for Π_1 and Π_2 exist but we focus on one in this section. They will be discussed in separate sections.

Setup *algorithm.* Setup is composed of three phases. The Setup_0 algorithm generates an (additive) group, which is cyclic, of prime order p, and a generator G. The crs string includes p, group parameters gp to be able to do operations in the group, and the generator G. It also includes parameters for a proof Π_1

[13] If we multiply a vector with elements in $\mathbb{Z}_p$ by a group element, we obtain a vector with elements in the group.

Table 2. Options for the ATHM Protocol. Non-standard models for provable security are indicated by either an ad-hoc assumption, or the random-oracle model (ROM), or the generic group model (GGM). In the Attributes menu, several options can be selected.

Attributes	**MAC**	$\mathit{\Pi}_1$ (T_{ext})	$\mathit{\Pi}_2$ (π)
– Issuer private metadata bit b – Public metadata m – Nonce t	– MAC-GGM – MAC-DDH	– no T_{ext} *(GGM)* – Knowledge-of-Exponent *(Assumption 3 or GGM)* – MPC-in-the-Head *(ROM)*	– Fiat-Shamir *(ROM)* – Homomorphic encryption

which are generated by Setup_1 together with a trapdoor td_1 for extraction and parameters for $\mathit{\Pi}_2$ which are generated by Setup_2 together with a trapdoor td_2.

```
Setup$(1^\lambda)$:
1: $\mathsf{Setup}_0(1^\lambda) \to (\mathsf{gp}, p, G)$          $\triangleright$ group setup
2: $\mathsf{Setup}_1(\mathsf{gp}, p, G) \to (\mathsf{crs}_1, \mathsf{td}_1)$          $\triangleright$ $\mathit{\Pi}_1$ setup
3: $\mathsf{Setup}_2(\mathsf{gp}, p, G) \to (\mathsf{crs}_2, \mathsf{td}_2)$          $\triangleright$ $\mathit{\Pi}_2$ setup
4: $\mathsf{crs} \leftarrow (\mathsf{gp}, p, G, \mathsf{crs}_1, \mathsf{crs}_2)$
5: $\mathsf{td} \leftarrow (\mathsf{td}_1, \mathsf{td}_2)$
```

The KeyGen *algorithm.* Key generation is composed of several phases.

```
KeyGen(crs):
1: $\mathsf{KeyGen}_0(\mathsf{crs}) \to (\mathsf{pk}_0, \mathsf{sk}_0)$
2: $\mathsf{KeyGen}_1(\mathsf{crs}, \mathsf{pk}_0) \to (\mathsf{pk}_1, \mathsf{sk}_1)$          $\triangleright$ $\mathit{\Pi}_1$ key generation
3: $\mathsf{KeyGen}_2(\mathsf{crs}, \mathsf{pk}_0, \mathsf{sk}_0) \to (\mathsf{pk}_2, \mathsf{sk}_2)$          $\triangleright$ $\mathit{\Pi}_2$ key generation
4: $\mathsf{pk} \leftarrow (\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{pk}_2)$
5: $\mathsf{sk} \leftarrow (\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{sk}_2)$
```

In KeyGen_0, the issuer selects four secrets $\mathsf{sk}_0 = (\vec{x}, \vec{y}, \vec{y'}, \vec{z})$ with $\vec{y}, \vec{y'}, \vec{z} \neq 0$, and sets $\mathsf{pk}_0 = \vec{Z} = \vec{z}G$. The structure of those secrets depends on the choice of the MAC algorithm. We denote by $\mathcal{E}_x$, $\mathcal{E}_y$, $\mathcal{E}_{y'}$, and $\mathcal{E}_z$ the respective domains of $\vec{x}$, $\vec{y}$, $\vec{y'}$, and z, depending on the choice of the MAC algorithm. We will also need the span $\bar{\mathcal{E}}_z$ of $\mathcal{E}_z$.

The token issuance protocol. The user has public parameters. The server's input is the secret $(\vec{x}, \vec{y}, \vec{y'}, \vec{z})$, a bit b to hide inside the token, and an agreed public metadata m to embed. The protocol works as depicted on Figure 8: the client selects a random tag share t_C and a random mask $\vec{r} \in \bar{\mathcal{E}}_z$ and sends $\vec{T} = t_C\vec{Z} + \vec{r}G$ to the issuer. The client must also send an extractable proof T_{ext} for $\vec{T}$. The issuer selects a random tag share t_S and generates a pair $(U, \vec{V})$ such that $(U, \vec{V} - \vec{r}U)$ is a valid MAC for tag $t = t_C + t_S$ and metadata (b, m) with key $(\vec{x}, \vec{y}, \vec{y'}, \vec{z})$. For this, the issuer selects $U = dG$ for a random $d \in \mathbb{Z}_p^*$ and $\vec{V} = d(\vec{x}G + b\vec{y}G + m\vec{y'}G + t_S\vec{z}G + \vec{T})$. A NIZK proof Π_2 must prove that the $(U, \vec{V}, t_S)$ triplet was correctly generated. Note that Π_2 must prove that b is a bit and that m was embedded. Both participants are assumed to have agreed on m (alternately, one participant decides and sends m to the other). Then, $(U, \vec{V}, t_S, \pi)$ is returned. The client computes $(U, \vec{V} - \vec{r}U)$. To make it unlinkable, the pair is multiplied by a random mask c to obtain another pair $\sigma = (P, \vec{Q})$.

```
$\mathsf{ATHM.Client}(G, \vec{Z}, m)$                                   $\mathsf{ATHM.IssueToken}((\vec{x}, \vec{y}, \vec{y'}, \vec{z}), b, m)$
$\mathsf{input}$: $(G, \vec{Z})$                                        $\mathsf{input}$: $(\vec{x}, \vec{y}, \vec{y'}, \vec{z}), b, m$
$t_C \leftarrow\!\!\$\ \mathbb{Z}_p, \quad \vec{r} \leftarrow\!\!\$\ \bar{\mathcal{E}}_z$
$\vec{T} := t_C\vec{Z} + \vec{r}G$
$\left[\vec{T}_{\mathsf{ext}} := \Pi_1.\mathsf{Prove}(t_C, \vec{r}; \mathsf{crs}, \mathsf{pk}, \vec{T})\right]$
                              $\xrightarrow{\vec{T}\left[, \vec{T}_{\mathsf{ext}}\right]}$
                              [**if not** $\Pi_1.\mathsf{Verify}(\vec{T}, \vec{T}_{\mathsf{ext}}, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_1)$ **then return** $\perp$]
                                                                        $t_S \leftarrow\!\!\$\ \mathbb{Z}_p, \quad d \leftarrow\!\!\$\ \mathbb{Z}_p^*$
                                                                        $U := dG$
                                                                        $\vec{V} := d(\vec{x}G + b\vec{y}G + m\vec{y'}G + t_S\vec{z}G + \vec{T})$
                                 $\pi \leftarrow \Pi_2.\mathsf{Prove}(b, d; U, \vec{V}, t_S, m, \vec{T}, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_2)$
                              $\xleftarrow{U, \vec{V}, t_S, \pi}$
**if not** $\Pi_2.\mathsf{Verify}(\pi, U, \vec{V}, t_S, m, \vec{T}, \mathsf{crs}, \mathsf{pk})$ **then return** $\perp$
**if** $U = 0$ **then return** $\perp$
$c \leftarrow\!\!\$\ \mathbb{Z}_p^*$
$P := cU$
$\vec{Q} := c(\vec{V} - \vec{r}U)$
$t := t_C + t_S$
$\sigma := (P, \vec{Q})$
$\mathsf{output}$: $(t, \sigma)$
```

Fig. 8. ATHM token issuance protocol.

About the extractable proof $\vec{T}_{\mathsf{ext}}$ following Π_1, there are several options which are discussed in a subsequent section. The role of $\vec{T}_{\mathsf{ext}}$ is to allow to formally prove the security of ATHM without relying on the generic group model.

The NIZK proof Π_2 is a Fiat-Shamir transform of an OR proof of two Schnorr proofs for $b = 0$ and $b = 1$. Variants without using a random oracle are possible too.

The ReadBit *algorithm.* The redemption of $(m, t, P, \vec{Q})$ with $(\vec{x}, \vec{y}, \vec{y'}, \vec{z})$ checks for which $b \in \{0, 1\}$, the equality $\vec{Q} = (\vec{x} + b\vec{y} + m\vec{y'} + t\vec{z})P$ is satisfied. Assuming $0 \notin \mathcal{E}_y$ ensures that b is unique. If b exists, it is returned. Otherwise, $\perp$ is returned.

Note that for this protocol, tag t needs to be a nonce as in other protocols [15], [16] i.e. the redeemer should check against double-spending of a token with the same t. Otherwise, it is easy to transform a valid token with attributes (b, m, t) into another valid token with the same attributes: let $\sigma = (P, \vec{Q})$. Then the client can forge another token $\sigma' = (P', \vec{Q'})$ by taking $\sigma' = c'.\sigma$ for any $c' \in \mathbb{Z}_p^*$.

B.2 The MAC Building Block

Our security results will be based on the security of an algebraic MAC. The simplest one is the MACGGM algorithm [6] defined as follows: given a secret $(x, y, y', z) \in \mathbb{Z}_p^4$, a valid authentication for $(b, m, t) \in \mathbb{Z}_p^3$ is a pair $\sigma = (P, Q)$ such that $Q = (x + by + my' + tz)P$. For this MAC to be secure, it is important that no adversary can find any linear relation between the random values of P. Hence, P is selected at random by the issuer.

In MACDDH [6], we consider $\vec{x}$, $\vec{y}$, $\vec{y'}$, and $\vec{z}$ as vectors of a particular form: $\vec{x} \in \mathcal{E}_x = \mathbb{Z}_p^3$, $\bar{\mathcal{E}}_z = \{(\alpha, \beta, 0); \alpha, \beta \in \mathbb{Z}_p\}$, and $\vec{y}, \vec{y'}, \vec{z} \in \mathcal{E}_y = \mathcal{E}_{y'} = \mathcal{E}_z = \bar{\mathcal{E}}_z - \{(0, 0, 0)\}$.

In general, we let $\vec{x} \in \mathcal{E}_x$, $\vec{y} \in \mathcal{E}_y$, $\vec{y'} \in \mathcal{E}_{y'}$, $\vec{z} \in \mathcal{E}_z$. A MAC $\sigma = (P, \vec{Q})$ for attributes (b, m, t) is valid if it satisfies $\vec{Q} = (\vec{x} + b\vec{y} + m\vec{y'} + t\vec{z})P$.

We will assume that no adversary can forge a valid σ for a new (b, t, m) tuple when given access to a MAC oracle and also that no adversary can distinguish $b \in \{b_0, b_1\}$ from a random valid tuple (t, m, σ) for this unknown b. The two respective games are depicted on Figure 9 and Figure 10.

```
Game EF-CMA:
 1: $\mathsf{Setup}_0(1^\lambda) \to (\mathsf{gp}, p, G)$
 2: pick $\vec{x} \leftarrow\$ \mathcal{E}_x$, $\vec{y} \leftarrow\$ \mathcal{E}_y$, $\vec{y'} \leftarrow\$ \mathcal{E}_{y'}$,
    $\vec{z} \leftarrow\$ \mathcal{E}_z$
 3: initialize $i \leftarrow 0$
 4: $\mathcal{A}^{\mathcal{O}_{\mathsf{MAC}}, \mathcal{O}_{\mathsf{verify}}}(\mathsf{gp}, p) \to (b, m, t, P, \vec{Q})$
 5: if $(b, m, t)$ is equal to some $(b_i, m_i, t_i)$
    then abort
 6: win iff $\mathcal{O}_{\mathsf{verify}}(b, m, t, P, \vec{Q})$

Oracle $\mathcal{O}_{\mathsf{MAC}}(b, m, t)$:
 7: increment $i$
 8: set $(b_i, m_i, t_i) \leftarrow (b, m, t)$
 9: pick a nonzero group element $P_i$
10: $\vec{Q}_i \leftarrow (\vec{x} + b_i\vec{y} + m_i\vec{y'} + t_i\vec{z})P_i$
11: **return** $(P_i, \vec{Q}_i)$

Oracle $\mathcal{O}_{\mathsf{verify}}(b, m, t, P, \vec{Q})$:
12: if $P = 0$ then **return** false
13: **return** whether $\vec{Q} = (\vec{x} + b\vec{y} + m\vec{y'} +$
    $t\vec{z})P$
```

Fig. 9. Forgeability Game for the MAC

We will make the two following assumptions. For MACGGM, they are proven in the generic group model [6]. For MACDDH, they are proven under the DDH assumption [6].

```
Game $\mathsf{IND\text{-}CMA}_{b^*}$:
 1: $\mathsf{flag} \leftarrow \mathsf{false}$
 2: initialize $i \leftarrow 0$
 3: $m^* \leftarrow \bot$
 4: $\mathsf{Setup}_0(1^\lambda) \to (\mathsf{gp}, p, G)$
 5: pick $\vec{x} \leftarrow\$ \mathcal{E}_x$, $\vec{y} \leftarrow\$ \mathcal{E}_y$, $\vec{y'} \leftarrow\$ \mathcal{E}_{y'}$,
    $\vec{z} \leftarrow\$ \mathcal{E}_z$
 6: return $\mathcal{A}^{\mathcal{O}_{\mathsf{MAC}}, \mathcal{O}_{\mathsf{chal}}, \mathcal{O}_{\mathsf{verify}}, \mathcal{O}_{\mathsf{valid}}}(\mathsf{gp}, p)$

Oracle $\mathcal{O}_{\mathsf{MAC}}(b, m, t)$:
 7: if $\mathsf{flag}$ and $(b, m, t) \in \{(b_0^*, m^*, t^*), (b_1^*, m^*, t^*)\}$ then return $\bot$
 8: increment $i$
 9: $(b_i, m_i, t_i) \leftarrow (b, m, t)$
10: pick a nonzero group element $P$
11: $\vec{Q} \leftarrow (\vec{x} + b\vec{y} + m\vec{y'} + t\vec{z})P$
12: return $(P, \vec{Q})$

Oracle $\mathcal{O}_{\mathsf{chal}}(b, b', m, t)$:
13: if $\mathsf{flag}$ or $\exists i\ (b_i, m_i, t_i) \in \{(b, m, t), (b', m, t)\}$ then return $\bot$
14: $(b_0^*, b_1^*, m^*, t^*) \leftarrow (b, b', m, t)$
15: $\mathsf{flag} \leftarrow \mathsf{true}$
16: pick a nonzero group element $P^*$
17: $\vec{Q}^* \leftarrow (\vec{x} + b_{b^*}^*\vec{y} + m^*\vec{y'} + t^*\vec{z})P^*$
18: return $(P^*, \vec{Q}^*)$

Oracle $\mathcal{O}_{\mathsf{verify}}(b, m, t, P, \vec{Q})$:
19: if $\mathsf{flag}$ and $b \in \{b_0^*, b_1^*\}$ and $(t, m) = (t^*, m^*)$ then return $\bot$
20: if $P = 0$ then return $\mathsf{false}$
21: return whether $\vec{Q} = (\vec{x} + b\vec{y} + m\vec{y'} + t\vec{z})P$

Oracle $\mathcal{O}_{\mathsf{valid}}(b_0, b_1, m, t, P, \vec{Q})$
22: if $\neg\mathsf{flag}$ then return $\bot$
23: if $(t, m) = (t^*, m^*)$ and $\{b_0, b_1\} \neq \{b_0^*, b_1^*\}$ then return $\bot$
24: if $P = 0$ then return $\mathsf{false}$
25: if $\vec{Q} = (\vec{x} + b_0\vec{y} + m\vec{y'} + t\vec{z})P$ then
    return $\mathsf{true}$
26: if $\vec{Q} = (\vec{x} + b_1\vec{y} + m\vec{y'} + t\vec{z})P$ then
    return $\mathsf{true}$
27: return $\mathsf{false}$
```

Fig. 10. Pseudorandomness Games for the MAC

Assumption 1 *In the* $\mathsf{EF\text{-}CMA}$ *game on Figure 9, we define the advantage of an adversary* $\mathcal{A}$ *by*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EF\text{-}CMA}}(\lambda) = \Pr[\mathsf{win}]$$

We assume that for any PPT adversary $\mathcal{A}$*, the advantage is a negligible function.*

Assumption 2 *In the* $\mathsf{IND\text{-}CMA}$ *game on Figure 10, we define the advantage of an adversary* $\mathcal{A}$ *by*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CMA}}(\lambda) = \Pr[\mathsf{IND\text{-}CMA}_1 \to 1] - \Pr[\mathsf{IND\text{-}CMA}_0 \to 1]$$

We assume that for any PPT adversary $\mathcal{A}$*, the advantage is a negligible function.*

B.3 The Extractable Proof Building Block

The $\mathit{\Pi}_1$ extractable proof is used in our analysis to formally prove security. We require a *straightline extraction* of t_C and $\vec{r}$ by using a trapdoor and a verification algorithm which does not require the trapdoor. Namely, a dedicated algorithm should be able to extract t_C and $\vec{r}$ from a valid proof and the view of the client and the issuer should be able to verify the proof. Furthermore, we need the proof to be simulatable in a computationally indistinguishable manner by using a trapdoor.

We propose several options for $\mathit{\Pi}_1$.

- We use a construction from techniques of verifiable encryption, which has a complexity cost but a security proven under standard assumptions. Namely, we should encrypt the values t_C and $\vec{r}$ with a public key, which can be decrypted using a trapdoor only, and in a way that we can publicly verify that it encrypts values satisfying $\vec{T} = t_C\vec{Z} + \vec{r}G$.
- We use a simple $\mathit{\Pi}_1$ scheme (i.e. the proof is computed and verified with a very low complexity overhead) but an ad-hoc (non-standard) extractability assumption.
- Use a trivial $\mathit{\Pi}_1$ scheme (i.e. with a void T_{ext} and an always true verification) and the generic group model.

Using verifiable encryption. Regarding the first approach, we can use the construction by Takahashi and Zaverucha [23]. Namely, t_C and $\vec{r}$ are shares among several participants who run a multiparty computation protocol to compute $[t_C]\vec{Z} + [\vec{r}]G$, reveal it, and compare it to $\vec{T}$. The execution is simulated. The views are committed with an extractable commitment (for instance, a public-key encryption scheme with a public key set up by Setup_1 in the common reference string crs_1, extractable from the secret key in td_1). Based on the commit values, a challenge is computed using a random oracle. This challenge gives the index of a participant whose view remains hidden but other commitments are opened. The $\vec{T}_{\mathsf{ext}}$ record consists of the commitments and the openings. Verification consists of checking that the openings match their respective commit values, that the opened views match and make their participant output true, and that the missing view is the one corresponding to the hash of the commit values through the random oracle.

To extract from $\vec{T}_{\mathsf{ext}}$, we use the trapdoor which allows to extract all views from the commit values. The views include all shares of t_C and $\vec{r}$ which can thus be reconstructed.

To forge $\vec{T}_{\mathsf{ext}}$, we can program the random oracle to predict the view not to be revealed and make the corresponding participant malicious so that others would output true.

The proposed verifiable encryption construction from [23] is secure from any MPC-in-the-head zero-knowledge proof and an IND-CPA cryptosystem which is binding. This can be instantiated with quite standard assumptions.

Using an ad-hoc knowledge-of exponent assumption. We consider a key generation KeyGen_1 which sets an additional secret $\mathsf{sk}_1 = \theta$ for the issuer and reveals $\mathsf{pk}_1 = (G_{\mathsf{ext}}, \vec{Z}_{\mathsf{ext}})$ to the client, with $G_{\mathsf{ext}} = \theta G$ and $\vec{Z}_{\mathsf{ext}} = \theta \vec{Z}$. Then, we set $\vec{T}_{\mathsf{ext}} = t_C \vec{Z}_{\mathsf{ext}} + \vec{r} G_{\mathsf{ext}}$ to be verified by $\vec{T}_{\mathsf{ext}} = \theta \vec{T}$.

Simulating $\vec{T}_{\mathsf{ext}}$ is trivial with the help of the trapdoor $\mathsf{sk}_1 = \theta$.

For extraction, we need a knowledge-of-exponent-like assumption [12]: that being able to make a valid $(\vec{T}, \vec{T}_{\mathsf{ext}})$ pair with the knowledge of the client would imply knowing a linear combination $\vec{T} = t_C \vec{Z} + \vec{r} G$. In the standard knowledge-of-exponent assumption, we do not have any oracle whereas in our case, we need them. However, some oracles may break the knowledge-of-exponent assumption as shown below. Thus, the difficulty is to show that our oracles do not break this assumption. The oracle calls may help to provide other $(\vec{T}, \vec{T}_{\mathsf{ext}})$ pairs to the client, without the client knowing the linear combination. Our assumption captures that the oracles somehow do not help for that. We formalize two types of oracles.

An oracle is of Type I if there exists vectors of low-degree polynomials $\vec{f}_1, \ldots, \vec{f}_n$ and scalars $\lambda_1, \ldots, \lambda_n$, for some integer n, such that the oracle is defined as follows:

Oracle $O(\vec{T}, \vec{T}_{\mathsf{ext}})$:
```
1: if $\vec{T}_{\mathsf{ext}} \neq \theta \vec{T}$ then **return** $\perp$
2: $\vec{A}_i \leftarrow \vec{f}_i(\mathsf{aux}) G + \lambda_i \vec{T}$ for $i = 1, \ldots, n$
3: pick $d \leftarrow\!\!\$\ \mathbb{Z}_p$ at random
4: **return** $(d\vec{A}_1, \ldots, d\vec{A}_n)$
```
The values θ and aux as well as the group parameters are used by the oracle. What is crucial is that O is stateless, does not use θ or $\vec{T}_{\mathsf{ext}}$ after Step 1, and that it randomizes the output with a fresh d. Once the random coins for t_S are fixed, the issuer producing $(U, \vec{V})$ (without π) can be seen as a Type I oracle which would represent $\mathcal{O}_{\mathsf{sign}}$.

An oracle is of Type II if there exists a deterministic polynomially bounded algorithm $\vec{f}$ returning a vector of scalars, such that the oracle is defined as follows:

Oracle $O(\mathsf{inp}, P, \vec{Q})$:

```
1: compute $\vec{s} \leftarrow \vec{f}(\mathsf{inp}, \mathsf{aux})$
2: **return** whether $\vec{Q} = \vec{s}P$
```

The value aux and the group parameters are used by the oracle. One crucial property is that the output of the oracle is independent from the group element representation. The $\mathcal{O}_{\mathsf{verify}}$ oracle can be seen as a Type II oracle.

We can now define the following game with an adversary $\mathcal{A}$ and an extractor $\mathsf{Extract}$:

```
1: $\mathsf{Setup}_0(1^\lambda) \rightarrow (\mathsf{gp}, p, G)$        ▷ set up the group parameters
2: pick $\mathsf{aux}_1, \mathsf{aux}_2, \ldots \leftarrow\!\!\$\ \mathbb{Z}_p$
3: $\vec{Z} \leftarrow G.\vec{L}(\mathsf{aux})$
4: pick $\theta \leftarrow\!\!\$\ \mathbb{Z}_p^*$
5: $G_{\mathsf{ext}} \leftarrow \theta G$, $\vec{Z}_{\mathsf{ext}} \leftarrow \theta \vec{Z}$
6: run $\mathcal{A}(\mathsf{gp}, p, G, G_{\mathsf{ext}}, \vec{Z}, \vec{Z}_{\mathsf{ext}}) \rightarrow (\vec{T}, \vec{T}_{\mathsf{ext}})$ with access to oracles of Type I and
   Type II
7: set $\mathsf{View}$ to the view of $\mathcal{A}$        ▷ including inputs, coins, and oracle answers
8: run $\mathsf{Extract}(\mathsf{View}) \rightarrow (t_C, \vec{r})$
9: **return** whether $\vec{T}_{\mathsf{ext}} = \theta \vec{T}$ and $t_C \vec{Z} + \vec{r} G \neq \vec{T}$
```

The game includes a uniform vector aux of secrets from $\mathbb{Z}_p$ and a linear function $\vec{L}$ to define $\vec{Z}$.

Our assumption for the extraction which we will prove shortly is as follows:

Assumption 3 *For every PPT $\mathcal{A}$, every linear $\vec{L}$, and every set of oracles of Type I and Type II, there exists a polynomially bounded extractor* $\mathsf{Extract}$ *such that the above game returns 1 with negligible probability.*

Hence, $\mathsf{Extract}$ can extract t_C and $\vec{r}$ on the fly from the view of $\mathcal{A}$ whenever $(\vec{T}, \vec{T}_{\mathsf{ext}})$ are valid pairs. With such an assumption, we can extract without rewinding and without the coins of the oracles. Note that the syntax for the extractor $\Pi_1.\mathsf{Extract}$ is a bit different from the CRS model here: instead of extracting with the help of a trapdoor td_1, we extract from the view View of the adversary.

To illustrate the difficulty, we stress that the assumption cannot generalize to *any* oracle. For instance, the oracle which uses more $\vec{T}_{\mathsf{ext}}$ by returning a random d multiplied by $(\vec{T}, \vec{T}_{\mathsf{ext}})$ would give to the client a valid pair for which he cannot know the scalar d. The variant of this counterexample which returns a secret x (from aux) multiplied by the $(\vec{T}, \vec{T}_{\mathsf{ext}})$ pair follows the same argument (unless x leaks). Consequently, the Boolean oracle defined by $O(i, P, \vec{Q})$ returning the ith bit of $(xP, x\vec{Q})$ with a secret x from aux makes the assumption invalid too.

So far, it is unknown how to prove Assumption 3 in the standard model. To convince ourselves that it is a reasonable assumption, we prove it in the generic model. However, our later security proofs will be in the standard model with the extra assumption that Assumption 3 is true.

Proof of Assumption 3 in the generic group model. We prove our assumption in the generic group model. We assume that all group operations that an adversary can make are only addition, subtractions, and comparison of previously obtained group elements. They are externalized with the values of group elements hidden.

In this model, an algorithm who knows the view of the adversary and who runs the adversary algorithm step by step to pay attention to every group operation is able to express any group element issued by the adversary as a linear combination of group elements which are in the view. These elements are G, G_{ext}, $\vec{Z}$, $\vec{Z}_{\mathsf{ext}}$, and the outputs from the Type I oracles.

We let Var be a set of formal variables which represent the values in aux. We represent each group element A produced by the adversary as a formal polynomial $\mathsf{Pol}_A(\bar{\theta}, \mathsf{Var}, \bar{d}_1, \ldots, \bar{d}_q)$ in terms of secrets θ, aux, and each d_i which is selected by the Type I oracles. The polynomial is such that $A = \mathsf{Pol}_A(\theta, \mathsf{aux}, d_1, \ldots, d_q)G$. Hence, $\mathsf{Pol}_G = 1$, $\mathsf{Pol}_{G_{\mathsf{ext}}} = \bar{\theta}$, for the ith components Z_i and $Z_{\mathsf{ext},i}$ of $\vec{Z}$ and $\vec{Z}_{\mathsf{ext}}$, we have $\mathsf{Pol}_{Z_{\mathsf{ext},i}} = \bar{\theta} \times \mathsf{Pol}_{Z_i}$. In the Type I oracles, we assume that the degree of each f_j is bounded by some δ. Each output A_j from the i-th oracle call can be expressed as a polynomial which is multiple of $\bar{d}_i$. For this, we take the f_j as polynomials in Var (of degree bounded by δ).

By induction, every appearing group element is represented by a polynomial of degree bounded by $\delta + q_I$, where q_I is the number of queries to a Type I oracle. Since aux is uniform, we notice that the probability that two group elements which are represented by two different polynomials will match with a probability bounded by $\frac{\delta+q_I}{p}$, due to the Schwartz-Zippel Lemma. As this is negligible, we assume it does not occur. Hence two appearing group elements are equal if and only if their polynomial representations are the same.

The first crucial observation is that the partial degree in θ of every polynomial representation of any produced group element is bounded by 1. This is because the known group elements are such that no f_j uses θ. Since the polynomial representation of $\vec{T}_{\mathsf{ext}}$ has no θ^2, to have the equation $\vec{T}_{\mathsf{ext}} = \theta\vec{T}$ satisfied, the polynomial representation of $\vec{T}$ cannot have any θ. We thus assume that neither the input $\vec{T}$ to oracle calls nor the final output $\vec{T}$ has any θ.

The second observation is that the outputs of the oracle calls have no θ (since neither $\vec{T}$ nor f_j has any) but are multiple of some d_i. The final $\vec{T}_{\mathsf{ext}}$ must be a linear combination of group elements in the view. Since none has any $d_i\theta$ monomial, this means $\vec{T}$ must have no d_i inside. We know it must have no θ as well. Hence, it is a linear combination of G and $\vec{Z}$ only. This linear combination is $\vec{T} = t_C\vec{Z} + \vec{r}G$.

By following step by step the group operations, and also doing the corresponding polynomial operation, we obtain t_C and $\vec{r}$. To lower the complexity of extraction, we can even do operations modulo the monomials d_i that we know will not appear in the end.

B.4 The Simulatable Proof Building Block

We adapt the Π_2 proof here.

Setup_2. First of all, $\mathsf{Setup}_2(\mathsf{gp}, p, G)$ first selects $\theta' \in \mathbb{Z}_p^*$ and sets $H = \theta'.G$, $\mathsf{crs}_2 = xH$, and $\mathsf{td}_2 = \theta'$. The role of H is to do a Pedersen commitment.

KeyGen_2. The algorithm $\mathsf{KeyGen}_2(\mathsf{crs}, \vec{Z}, \mathsf{sk}_0)$ parses $\mathsf{crs} = (\mathsf{gp}, p, G, ., H)$ and $\mathsf{sk}_0 = (\vec{x}, \vec{y}, \vec{y'}, \vec{z})$, picks $\vec{r}_x \in \mathcal{E}_x, \vec{r}_y \in \mathcal{E}_y, \vec{r}_{y'} \in \mathcal{E}_{y'}$, and computes $\vec{C}_x = \vec{x}G + \vec{r}_x H$, $\vec{C}_y = \vec{y}G + \vec{r}_y H$, and $\vec{C}_{y'} = \vec{y'}G + \vec{r}_{y'} H$.

Then, KeyGen_2 computes a proof of knowledge for $(\vec{x}, \vec{y}, \vec{z}, \vec{r}_x, \vec{r}_y)$. For this, it picks random $\vec{\rho}_x, \vec{\rho}_{r_x} \in \mathcal{E}_x$, $\vec{\rho}_y, \vec{\rho}_{r_y} \in \mathcal{E}_y$, $\vec{\rho}_{y'}, \vec{\rho}_{r_{y'}} \in \mathcal{E}_{y'}$, $\vec{\rho}_z \in \mathcal{E}_z$, computes $\vec{\Gamma}_x = \vec{\rho}_x G + \vec{\rho}_{r_x} H$, $\vec{\Gamma}_y = \vec{\rho}_y G + \vec{\rho}_{r_y} H$, $\vec{\Gamma}_{y'} = \vec{\rho}_{y'} G + \vec{\rho}_{r_{y'}} H$, $\vec{\Gamma}_z = \vec{\rho}_z G$,

$$\varepsilon = \mathsf{Hash}(G, H, \vec{C}_x, \vec{C}_y, \vec{C}_{y'}, \vec{Z}, \vec{\Gamma}_x, \vec{\Gamma}_y, \vec{\Gamma}_{y'}, \vec{\Gamma}_z)$$

$(\vec{a}_x, \vec{a}_{r_x}, \vec{a}_y, \vec{a}_{r_y}, \vec{a}_{y'}, \vec{a}_{r_{y'}}, \vec{a}_z) = (\vec{\rho}_x, \vec{\rho}_{r_y}, \vec{\rho}_y, \vec{\rho}_{r_y}, \vec{\rho}_{y'}, \vec{\rho}_{r_{y'}}, \vec{\rho}_z) + \varepsilon(\vec{x}, \vec{r}_x, \vec{y}, \vec{r}_y, \vec{y'}, \vec{r}_{y'}, \vec{z})$, and the proof is $(\varepsilon, \vec{a}_x, \vec{a}_{r_x}, \vec{a}_y, \vec{a}_{r_y}, \vec{a}_{y'}, \vec{a}_{r_{y'}}, \vec{a}_z)$. Finally,

$$\mathsf{pk}_2 = (\vec{C}_x, \vec{C}_y, \vec{C}_{y'}, \varepsilon, \vec{a}_x, \vec{a}_{r_x}, \vec{a}_y, \vec{a}_{r_y}, \vec{a}_{y'}, \vec{a}_{r_{y'}}, \vec{a}_z)$$

and $\mathsf{sk}_2 = (\vec{r}_x, \vec{r}_y, \vec{r}_{y'})$ are the output of KeyGen_2.

To verify the proof (this must be done at least once for all), the client computes $\vec{\Gamma}_x = \vec{a}_x G + \vec{a}_{r_x} H - \varepsilon \vec{C}_x$, $\vec{\Gamma}_y = \vec{a}_y G + \vec{a}_{r_y} H - \varepsilon \vec{C}_y$, $\vec{\Gamma}_{y'} = \vec{a}_{y'} G + \vec{a}_{r_{y'}} H - \varepsilon \vec{C}_{y'}$, $\vec{\Gamma}_z = \vec{a}_z G - \varepsilon \vec{Z}$, and checks $\varepsilon = \mathsf{Hash}(G, H, \vec{C}_x, \vec{C}_y, \vec{C}_{y'}, \vec{Z}, \vec{\Gamma}_x, \vec{\Gamma}_y, \vec{\Gamma}_{y'}, \vec{\Gamma}_z)$.

Prove. The algorithm $\mathsf{Prove}(b, d; U, \vec{V}, t_S, m, \vec{T}, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_2)$ parses different elements and picks $\vec{\mu}, \vec{a}_{1-b}, \vec{r}_\mu \in \bar{\mathcal{E}}_y$, $e_{1-b}, r_d \in \mathbb{Z}_p$, $\vec{r}_\rho, \vec{r}_w \in \mathcal{E}$ at random. Then, it sets $d' = -\frac{1}{d}$, $\vec{\rho} = -(\vec{r}_x + b\vec{r}_y + m\vec{r}_{y'} + \vec{\mu})$, $\vec{w} = \vec{x} + b\vec{y} + m\vec{y'} + t_S \vec{z}$ and computes $\vec{C} = b.\vec{C}_y + H.\vec{\mu}$, $\vec{C}_b = H.\vec{r}_\mu$, $\vec{C}_{1-b} = H.\vec{a}_{1-b} - e_{1-b}(\vec{C} - (1-b)\vec{C}_y)$,

$$\begin{pmatrix} C_d \\ \vec{C}_\rho \\ \vec{C}_w \end{pmatrix} = r_d \begin{pmatrix} U \\ \vec{V} \\ \vec{V} \end{pmatrix} + \begin{pmatrix} 0 \cdots 0 \\ H.\mathcal{I} \\ 0.\mathcal{I} \end{pmatrix} \vec{r}_\rho + \begin{pmatrix} 0 \cdots 0 \\ 0.\mathcal{I} \\ G.\mathcal{I} \end{pmatrix} \vec{r}_w$$

where $\mathcal{I}$ is the identity matrix,

$$e = \mathsf{Hash}(G, H, \vec{C}_x, \vec{C}_y, \vec{C}_{y'}, \vec{Z}, U, \vec{V}, t_S, m, \vec{C}, \vec{C}_0, \vec{C}_1, C_d, \vec{C}_\rho, \vec{C}_w)$$

$e_b = e - e_{1-b}$, $\vec{a}_b = \vec{r}_\mu + e_b \vec{\mu}$, and $(a_d, \vec{a}_\rho, \vec{a}_w) = (r_d, \vec{r}_\rho, \vec{r}_w) + e(d', \vec{\rho}, \vec{w})$. Finally,

$$\pi = (\vec{C}, e_0, e_1, \vec{a}_0, \vec{a}_1, a_d, \vec{a}_\rho, \vec{a}_w)$$

Verify. The algorithm $\mathsf{Verify}(\pi, U, \vec{V}, t_S, m, \vec{T}, \mathsf{crs}, \mathsf{pk})$ computes $\vec{C}_0 = H.\vec{a}_0 - e_0.\vec{C}$, $\vec{C}_1 = H.\vec{a}_1 - e_1.(\vec{C} - \vec{C}_y)$, $e = e_0 + e_1$,

$$\begin{pmatrix} C_d \\ \vec{C}_\rho \\ \vec{C}_w \end{pmatrix} = a_d \begin{pmatrix} U \\ \vec{V} \\ \vec{V} \end{pmatrix} + \begin{pmatrix} 0 \cdots 0 \\ H.\mathcal{I} \\ 0.\mathcal{I} \end{pmatrix} \vec{a}_\rho + \begin{pmatrix} 0 \cdots 0 \\ 0.\mathcal{I} \\ G.\mathcal{I} \end{pmatrix} \vec{a}_w + e \begin{pmatrix} G \\ \vec{C}_x + \vec{C} + m\vec{C}_{y'} + t_S.\vec{Z} + \vec{T} \\ \vec{T} \end{pmatrix}$$

then verifies the $e = \mathsf{Hash}(G, H, \vec{C}_x, \vec{C}_y, \vec{C}_{y'}, \vec{Z}, U, \vec{V}, t_S, m, \vec{C}, \vec{C}_0, \vec{C}_1, C_d, \vec{C}_\rho, \vec{C}_w)$.

$\mathsf{SimKeyGen}$. The algorithm $\mathsf{SimKeyGen}(\mathsf{crs}, \vec{Z})$ parses crs to retrieve G and H. It picks a fully random pk_2, computes $\vec{\varGamma}_x$, $\vec{\varGamma}_y$, $\vec{\varGamma}_{y'}$, $\vec{\varGamma}_z$ like in verification, then programs the random oracle to have a consistent Hash. The simulation is perfect.

$\mathsf{Simulate}$. The algorithm $\mathsf{Simulate}(U, \vec{V}, t_S, \vec{T}, \mathsf{crs}, \mathsf{pk}, \mathsf{td}_2)$ parses inputs. Then, it picks a random proof π with same distribution, i.e. $\vec{C} \in G.\bar{\mathcal{E}}_y$, $e_0, e_1, a_d \in \mathbb{Z}_p$, $\vec{a}_0, \vec{a}_1 \in \bar{\mathcal{E}}_y$ $\vec{a}_\rho, \vec{a}_w \in \mathcal{E}$ all uniform and independent. It can then proceed as in $\mathsf{Verify}(\pi, U, \vec{V}, t_S, \vec{T}, \mathsf{crs}, \mathsf{pk})$ to compute $\vec{C}_0, \vec{C}_1, C_d, \vec{C}_\rho, \vec{C}_w$, then the hash. Then, the random oracle is programmed to output the correct hash $e = e_0 + e_1$. Programming may fail with probability at most $\frac{q_{\mathsf{Hash}}}{p^6}$.

Special soundness of pk_2. We can extract from a process issuing a valid pk_2 a single $(\vec{C}_x, \vec{C}_y, \vec{C}_{y'}, \vec{\varGamma}_x, \vec{\varGamma}_y, \vec{\varGamma}_{y'})$ and the answers $(\vec{a}_x, \vec{a}_{r_x}, \vec{a}_y, \vec{a}_{r_y}, \vec{a}_{y'}, \vec{a}_{r_{y'}})$ to two different challenges ε and $\varepsilon + \Delta\varepsilon$. From the equations in verification, we can then set $\vec{x} = \frac{1}{\Delta\varepsilon}\Delta\vec{a}_x$, $\vec{r}_x = \frac{1}{\Delta\varepsilon}\Delta\vec{a}_{r_x}$, $\vec{y} = \frac{1}{\Delta\varepsilon}\Delta\vec{a}_y$, $\vec{r}_y = \frac{1}{\Delta\varepsilon}\Delta\vec{a}_{r_y}$, $\vec{y'} = \frac{1}{\Delta\varepsilon}\Delta\vec{a}_{y'}$, $\vec{r}_{y'} = \frac{1}{\Delta\varepsilon}\Delta\vec{a}_{r_{y'}}$, and obtain $\vec{C}_x = G.\vec{x} + H.\vec{r}_x$, $\vec{C}_y = G.\vec{y} + H.\vec{r}_y$, and $\vec{C}_{y'} = G.\vec{y'} + H.\vec{r}_{y'}$.

Ideally, we should have an adaptive extractable proof in pk_2 to extract the secrets. Since we only need this in unlinkability and we should extract from the initialization adversary $\mathcal{A}_1$ who is only setting up pk_2, we keep the proof as it is for simplicity. An adversary $\mathcal{A}_1$ succeeding to build a valid pk_2 without being extractable would have a probability of success that we denote by $\mathsf{Adv}^{\mathsf{EXTRACT}}_{\mathcal{A}_1}$.

Special soundness of π. We assume that the secrets $\vec{x}, \vec{y}, \vec{y'}, \vec{z}, \vec{r}_x, \vec{r}_y, \vec{r}_{y'}$ have been correctly extracted. We consider an issuer who responds with $(U, \vec{V}, t_S, \pi)$ to a request $\vec{T}$ with a proof π which verifies, with probability larger than $\frac{1}{p}$. Hence, there exists a single $(U, \vec{V}, t_S, \vec{C}, \vec{C}_0, \vec{C}_1, C_d, \vec{C}_\rho, \vec{C}_w)$ and the valid answer $(e_0, e_1, \vec{a}_0, \vec{a}_1, a_d, \vec{a}_\rho, \vec{a}_w)$ to two different challenges e and $e + \Delta e$.

Thanks to the equations in verification, we obtain $H.\Delta\vec{a}_0 = \Delta e_0.\vec{C}$, $H.\Delta\vec{a}_1 = \Delta e_1.(\vec{C} - \vec{C}_y)$, so $\vec{C} = H.\frac{\Delta(\vec{a}_0 + \vec{a}_1)}{\Delta e} + \frac{\Delta e_1}{\Delta e}\vec{C}_y$. We also obtain $G = -\frac{\Delta a_d}{\Delta e}U$, $-(\vec{C}_x + \vec{C} + m\vec{C}_{y'} + t_S\vec{Z} + \vec{T}) = \frac{\Delta a_d}{\Delta e}\vec{V} + \frac{1}{\Delta e}H.\Delta\vec{a}_\rho$, $-\vec{T} = \frac{\Delta a_d}{\Delta e}\vec{V} + \frac{1}{\Delta e}G.\Delta\vec{a}_w$. As $G \neq 0$, we can invert and multiply the last three equations by $d = -\frac{\Delta e}{\Delta a_d}$. If $\frac{\Delta e_1}{\Delta e} \neq 1$, we have $\Delta e_0 \neq 0$ so we obtain $\vec{C} = \frac{1}{\Delta e_0}H.\Delta\vec{a}_0$. Otherwise, we obtain $\vec{C} = \vec{C}_y + \frac{1}{\Delta e}H.\Delta(\vec{a}_0 + \vec{a}_1)$. In any case, we obtain $b, \vec{\mu}, d, \vec{\rho}, \vec{w}$ such that $\vec{C} = b.\vec{C}_y + H.\vec{\mu}$, $U = d.G$, $\vec{V} = d(\vec{C}_x + \vec{C} + m\vec{C}_{y'} + t_S\vec{Z} + \vec{T}) + H.\vec{\rho} = G.\vec{w} + d.\vec{T}$.

Given that $\vec{C}_x = G.\vec{x} + H.\vec{r}_x$, $\vec{C}_y = G.\vec{y} + H.\vec{r}_y$, $\vec{C}_{y'} = G.\vec{y'} + H.\vec{r}_{y'}$, and $\vec{Z} = G.\vec{z}$, the equations in $\vec{V}$ give $(\vec{x} + b\vec{y} + m\vec{y'} + t_S\vec{z} - \frac{1}{d}.\vec{w})G + (\vec{r}_x + b\vec{r}_y + m\vec{r}_{y'} + \vec{\mu} + \frac{1}{d}.\vec{\rho})H = 0$. Unless the issuer can compute the discrete logarithm of H, this implies that we have $\frac{1}{d}\vec{w} = \vec{x} + b\vec{y} + m\vec{y'} + t_S\vec{z}$ and $-\vec{\mu} - \frac{1}{d}\vec{\rho} = \vec{r}_x + b\vec{r}_y + m\vec{r}_{y'} + t_S\vec{z'}$. These equations imply $U = d.G$ and $\vec{V} = d(G\vec{x} + bG\vec{y} + m\vec{y'} + t_SG\vec{z} + \vec{T})$. Thus, if no such (b, d) exist with $b \in \{0, 1\}$, the proof succeeds with probability bounded by $\frac{1}{p}$ plus the advantage of a discrete logarithm adversary.

The solution (b, d) such that $b \in \{0, 1\}$, $U = d.G$, and $\vec{V} = d(G\vec{x} + bG\vec{y} + m\vec{y'} + t_SG\vec{z} + \vec{T})$ is unique when it exists. We let $\mathsf{Adv}^{\mathsf{SOUND}}_{\mathcal{A}}$ be the probability that and adversary $\mathcal{A}$ succeeds to build a valid proof π when no solution exists. Using the forking lemma, we obtain the answer to an alternate challenge with probability $\frac{(\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SOUND}})^2}{4q_{\mathsf{Hash}}} - \frac{1}{p}$. Using the above extraction method, we deduce the discrete logarithm of H. This defines an adversary $\mathcal{B}$ such that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SOUND}} \leq 2\sqrt{q_{\mathsf{Hash}}}\sqrt{\mathsf{Adv}_{\mathcal{B}}^{\mathsf{DLOG}} + \frac{1}{p}}$$

Variant without ROM. The Fiat-Shamir relies on the random oracle model (ROM). We can get rid of this assumption by following usual techniques [5]. For instance, we can use an additive homomorphic encryption scheme. The client would generate a key pair and sends the encryption key pk' together with the encryption $\mathsf{Enc}_{\mathsf{pk}'}(e)$ of a random challenge e. Then, the server would do all operations homomorphically to return and encrypted π. The client would decrypt it.

B.5 Unforgeability

Theorem 4. *Under Assumption 1,* ATHM *is* OMUF*-secure. More precisely, given an* OMUF*-adversary* $\mathcal{A}$ *making* q *oracle calls to* $\mathcal{O}_{\mathsf{sign}}$ *and* q_{Hash} *calls to the random oracle, there exists an* $\mathsf{EF\text{-}CMA}$*-adversary* $\mathcal{B}$ *making* $q+3$ *oracle calls to* $\mathcal{O}_{\mathsf{MAC}}$ *such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{OMUF}} \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{EF\text{-}CMA}} + \frac{q+1}{p} + \frac{q_{\mathsf{Hash}}}{p^6}$$

This result needs $\Pi_1.\mathsf{Extract}$ to succeed if an only if $\Pi_1.\mathsf{Verify}$ returns true and Π_2 to be perfectly simulatable with a trapdoor using $\Pi_2.\mathsf{SimKeyGen}$ and $\Pi_2.\mathsf{Simulate}$.

Proof. We start with an adversary $\mathcal{A}$ playing the one-more unforgerability game against ATHM and we transform it into a sequence of adversaries playing other games with the same success.

First of all, we change the game so that the issuer no longer sends a proof π but rather make the adversary to set up Π_2, generate the keys, and simulate π perfectly. The simulation uses $\Pi_2.\mathsf{SimKeyGen}$ and $\Pi_2.\mathsf{Simulate}$. We obtain an adversary Adv_0 playing a game Γ_0 such that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{OMUF}} \leq \mathsf{Adv}_{\mathcal{A}_0}^{\Gamma_0} + \frac{q_{\mathsf{Hash}}}{p^6}$$

Then, we define a new adversary $\mathcal{A}_1$ who can use the extractable proof $\vec{T}_{\mathsf{ext}}$ in our scheme. Giving consistent $\vec{T}$ and $\vec{T}_{\mathsf{ext}}$ proves the knowledge of t_C and $\vec{r}$ with a way to extract them. With a trapdoor td_1 (in the CRS model), $\mathcal{A}_1$ can extract $(t_C, \vec{r}) = \Pi_1.\mathsf{Extract}(\mathsf{td}_1, \vec{T}_{\mathsf{ext}})$ (or $(t_C, \vec{r}) = \Pi_1.\mathsf{Extract}(\mathsf{View})$ with View being the view of $\mathcal{A}$ in the model based on Assumption 3). Then, $\mathcal{A}_1$ can call a MAC oracle to build $(t, P, \vec{Q})$ such that t and P are random and $\vec{Q} = (\vec{x} + b\vec{y} + m\vec{y}' + t\vec{z})P$.

Then, $\mathcal{A}_1$ sets $t_S = t - t_C$, $U = P$, and $\vec{V} = \vec{Q} + \vec{r}U$ to return to the client as written in Figure 11. Note that this requires to simulate the proof π based on the trapdoor θ' from td_2 (or programming the random oracle). This way, an attacker for our scheme transforms into a attacker in the Γ_1 game in Figure 11.

The extraction procedure $\Pi_1.\mathsf{Extract}(\mathsf{td}_1, \vec{T}_{\mathsf{ext}})$ fails iff $\Pi_1.\mathsf{Verify}(\vec{T}, \vec{T}_{\mathsf{ext}}, G, \vec{Z})$ is false. As forging π is perfectly simulatable, $\mathcal{A}_1$ playing Γ_1 perfectly simulates $\mathcal{A}$ playing OMUF. If $\mathcal{A}$ wins, there exists a bit b such that there are more t_j with $b_j = b$ than oracle calls to $\mathcal{O}_{\mathsf{sign}}$ with bit b. Hence, there must exist one for which t_j does not match any $\mathcal{O}$ call with the bit b. Therefore, $(b_j, m_j, t_j, \sigma_j)$ is a valid forgery. Therefore, we have

$$\mathsf{Adv}_{\mathcal{A}_0}^{\Gamma_0} \leq \mathsf{Adv}_{\mathcal{A}_1}^{\Gamma_1}$$

```
Adversary $\mathcal{A}_1^{\mathcal{O},\mathcal{O}_{\mathsf{verify}}}(\mathsf{gp}, p, G, \vec{Z})$:
 1: $\mathsf{Setup}_1(\mathsf{gp}, p, G) \to (\mathsf{crs}_1, \mathsf{td}_1)$
 2: $\mathsf{Setup}_2(\mathsf{gp}, p, G) \to (\mathsf{crs}_2, \mathsf{td}_2)$
 3: $\mathsf{crs} \leftarrow (\mathsf{gp}, p, G, \mathsf{crs}_1, \mathsf{crs}_2)$
 4: $\mathsf{KeyGen}_1(\mathsf{crs}, \vec{Z}) \to (\mathsf{pk}_1, \mathsf{sk}_1)$
 5: $\mathsf{SimKeyGen}(\mathsf{crs}, \vec{Z}) \to \mathsf{pk}_2$
 6: $\mathsf{pk} \leftarrow (\mathsf{crs}, \vec{Z}, \mathsf{pk}_1, \mathsf{pk}_2)$
 7: run $\mathcal{A}^{\mathcal{O}_{\mathsf{sign}},\mathcal{O}_{\mathsf{read}}}(\mathsf{crs}, \mathsf{pk}) \to$
    $(b, m, (t'_j, \sigma_j)_j)$
 8: find $j$ such that for all $i$, $(b, m, t'_j) \neq$
    $(b_i, m_i, t_i)$ (if none, return $\bot$)
 9: return $(b, m, t'_j, \sigma_j)$

Subroutine $\mathcal{O}_{\mathsf{sign}}(b, m, \mathsf{query})$:
10: parse $\mathsf{query} \to (\vec{T}, \vec{T}_{\mathsf{ext}})$
11: $\Pi_1.\mathsf{Extract}(\mathsf{td}_1, \mathsf{sk}_1, \vec{T}_{\mathsf{ext}}) \to (t_C, \vec{r})$
12: if extraction failed or $\vec{T} \neq t_C\vec{Z} + \vec{r}G$
    then return $\bot$
13: pick $t \leftarrow\$ \mathbb{Z}_p$
14: $\mathcal{O}(b, m, t) \to (P, \vec{Q})$
15: $t_S \leftarrow t - t_C$
16: $U \leftarrow P$
17: $\vec{V} \leftarrow \vec{Q} + \vec{r}U$
18: $\pi \leftarrow \Pi_2.\mathsf{Simulate}(U, \vec{V}, t_S, \mathsf{crs}, \mathsf{pk}, \mathsf{td}_2)$
19: return $(U, \vec{V}, t_S, \pi)$

Subroutine $\mathcal{O}_{\mathsf{read}}(m, t, \sigma)$:
20: if $\mathcal{O}_{\mathsf{verify}}(0, m, t, \sigma)$ then return 0
21: if $\mathcal{O}_{\mathsf{verify}}(1, m, t, \sigma)$ then return 1
22: return $\bot$
```

```
Game $\Gamma_1$:
 1: $\mathsf{Setup}_0(1^\lambda) \to (\mathsf{gp}, p, G)$
 2: pick $\vec{x} \leftarrow\$ \mathcal{E}_x$, $y \leftarrow\$ \mathcal{E}_y$, $y' \leftarrow\$ \mathcal{E}_{y'}$,
    $z \leftarrow\$ \mathcal{E}_z$
 3: set $\vec{Z} \leftarrow \vec{z}G$
 4: initialize $i \leftarrow 0$
 5: $\mathcal{A}_1^{\mathcal{O},\mathcal{O}_{\mathsf{verify}}}(\mathsf{gp}, p, G, \vec{Z}) \to (b, m, t, P, \vec{Q})$
 6: if $(b, m, t)$ is equal to some $(b_i, m_i, t_i)$
    then abort
 7: win iff $\mathcal{O}_{\mathsf{verify}}(b, m, t, P, \vec{Q})$

Oracle $\mathcal{O}(b, m, t)$:
 8: increment $i$
 9: $(b_i, m_i, t_i) \leftarrow (b, m, t)$
10: pick a nonzero group element $P_i$
11: $\vec{Q}_i \leftarrow (\vec{x} + b_i\vec{y} + m_i\vec{y'} + t_i\vec{z})P_i$
12: return $(P_i, \vec{Q}_i)$

Oracle $\mathcal{O}_{\mathsf{verify}}(b, m, t, P, \vec{Q})$:
13: if $P = 0$ then return false
14: return whether $\vec{Q} = (\vec{x} + b\vec{y} + m\vec{y'} +$
    $t\vec{z})P$
```

Fig. 11. Unforgeability Step 1: $\mathcal{A}_1$ playing Γ_1

We now get rid of the selection of G and of the input $\vec{Z}$ to the adversary by a change of variable $G = \mathsf{PP}_0$, $\vec{Z} = \vec{\mathsf{QQ}}_0$, $\vec{x} = \vec{\mathsf{xx}}$, $\vec{y} = \vec{\mathsf{yy}}$, $\vec{y'} = \vec{\mathsf{yy}'}$, $\vec{z} = \vec{\mathsf{xx}} + \mathsf{tt}_0.\vec{\mathsf{zz}}$, which uses an initial MAC $(\mathsf{bb}_0, \mathsf{mm}_0, \mathsf{tt}_0, \mathsf{PP}_0, \vec{\mathsf{QQ}}_0)$ with secret $(\vec{\mathsf{xx}}, \vec{\mathsf{yy}}, \vec{\mathsf{yy}'}, \vec{\mathsf{zz}})$ and input $\mathsf{bb}_0 = 0$, $\mathsf{mm}_0 = 0$, and $\mathsf{tt}_0 = 1$. We set $\mathsf{bb} = \frac{b}{1+t}$, $\mathsf{mm} = \frac{m}{1+t}$, $\mathsf{tt} = \frac{t.\mathsf{tt}_0}{1+t}$, $\mathsf{PP} = P$ and $\vec{\mathsf{QQ}} = (1+t)^{-1}\vec{Q}$ so that

$$\vec{\mathsf{xx}} + \mathsf{bb}.\vec{\mathsf{yy}} + \mathsf{mm}.\vec{\mathsf{yy}'} + \mathsf{tt}.\vec{\mathsf{zz}} = (1+t)^{-1}(\vec{x} + b\vec{y} + m\vec{y'} + t\vec{z})$$

Thus, an adversary $\mathcal{A}_1$ for Γ_1 transforms into another adversary $\mathcal{B}$ in the EF-CMA game as depicted in Figure 12.

```
Adversary $\mathcal{B}^{\mathcal{O}_{\mathsf{MAC}},\mathcal{O}_{\mathsf{verify}}}(\mathsf{gp}, p)$:
 1: $\mathsf{bb}_0 \leftarrow 0$
 2: $\mathsf{mm}_0 \leftarrow 0$
 3: $\mathsf{tt}_0 \leftarrow 1$
 4: $\mathcal{O}_{\mathsf{MAC}}(\mathsf{bb}_0, \mathsf{mm}_0, \mathsf{tt}_0) \rightarrow (G, \vec{Z})$
 5: run $\mathcal{A}_1^{\mathcal{O}',\mathcal{O}''}(\mathsf{gp}, p, G, \vec{Z}) \rightarrow (b, m, t, P, \vec{Q})$
 6: $\mathsf{bb} \leftarrow \frac{b}{1+t}$
 7: $\mathsf{mm} \leftarrow \frac{m}{1+t}$
 8: $\mathsf{tt} \leftarrow \frac{t.\mathsf{tt}_0}{1+t}$
 9: $\mathsf{PP} \leftarrow P$
10: $\vec{\mathsf{QQ}} \leftarrow (1+t)^{-1}\vec{Q}$
11: return $(\mathsf{bb}, \mathsf{mm}, \mathsf{tt}, \mathsf{PP}, \vec{\mathsf{QQ}})$

Subroutine $\mathcal{O}'(b, m, t)$:
12: $\mathsf{bb} \leftarrow \frac{b}{1+t}$
13: $\mathsf{mm} \leftarrow \frac{m}{1+t}$
14: $\mathsf{tt} \leftarrow \frac{t.\mathsf{tt}_0}{1+t}$
15: $\mathcal{O}_{\mathsf{MAC}}(\mathsf{bb}, \mathsf{mm}, \mathsf{tt}) \rightarrow (\mathsf{PP}, \vec{\mathsf{QQ}})$
16: $P \leftarrow \mathsf{PP}$
17: $\vec{Q} \leftarrow (1+t)\vec{\mathsf{QQ}}$
18: return $(P, \vec{Q})$

Subroutine $\mathcal{O}''(b, m, t, P, \vec{Q})$:
19: $\mathsf{bb} \leftarrow \frac{b}{1+t}$
20: $\mathsf{mm} \leftarrow \frac{m}{1+t}$
21: $\mathsf{tt} \leftarrow \frac{t.\mathsf{tt}_0}{1+t}$
22: $\mathsf{PP} \leftarrow P$
23: $\vec{\mathsf{QQ}} \leftarrow (1+t)^{-1}\vec{Q}$
24: return $\mathcal{O}_{\mathsf{verify}}(\mathsf{bb}, \mathsf{mm}, \mathsf{tt}, \mathsf{PP}, \vec{\mathsf{QQ}})$
```

```
Game EF-CMA:
 1: $\mathsf{Setup}_0(1^\lambda) \rightarrow (\mathsf{gp}, p, G_0)$
 2: pick $\vec{\mathsf{xx}} \leftarrow\!\!\$\ \mathcal{E}_x$, $\vec{\mathsf{yy}} \leftarrow\!\!\$\ \mathcal{E}_y$, $\vec{\mathsf{yy}'} \leftarrow\!\!\$\ \mathcal{E}_{y'}$, $\vec{\mathsf{zz}} \leftarrow\!\!\$\ \mathcal{E}_z$
 3: initialize $i \leftarrow 0$
 4: $\mathcal{B}^{\mathcal{O}_{\mathsf{MAC}},\mathcal{O}_{\mathsf{verify}}}(\mathsf{gp}, p) \rightarrow (\mathsf{bb}, \mathsf{mm}, \mathsf{tt}, \mathsf{PP}, \vec{\mathsf{QQ}})$
 5: if $(\mathsf{bb}, \mathsf{mm}, \mathsf{tt})$ is equal to some $(\mathsf{bb}_i, \mathsf{mm}_i, \mathsf{tt}_i)$ then abort
 6: win iff $\mathcal{O}_{\mathsf{verify}}(\mathsf{bb}, \mathsf{mm}, \mathsf{tt}, \mathsf{PP}, \vec{\mathsf{QQ}})$

Oracle $\mathcal{O}_{\mathsf{MAC}}(\mathsf{bb}, \mathsf{mm}, \mathsf{tt})$:
 7: increment $i$
 8: $(\mathsf{bb}_i, \mathsf{mm}_i, \mathsf{tt}_i) \leftarrow (\mathsf{bb}, \mathsf{mm}, \mathsf{tt})$
 9: pick a nonzero group element $\mathsf{PP}_i$
10: $\vec{\mathsf{QQ}}_i \leftarrow (\vec{\mathsf{xx}} + \mathsf{bb}_i.\vec{\mathsf{yy}} + \mathsf{mm}_i.\vec{\mathsf{yy}'} + \mathsf{tt}_i.\vec{\mathsf{zz}})\mathsf{PP}_i$
11: return $(\mathsf{PP}_i, \vec{\mathsf{QQ}}_i)$

Oracle $\mathcal{O}_{\mathsf{verify}}(\mathsf{bb}, \mathsf{mm}, \mathsf{tt}, \mathsf{PP}, \vec{\mathsf{QQ}})$:
12: if $\mathsf{PP} = 0$ then return false
13: return whether $\vec{\mathsf{QQ}} = (\vec{\mathsf{xx}} + \mathsf{bb}.\vec{\mathsf{yy}} + \mathsf{mm}.\vec{\mathsf{yy}'} + \mathsf{tt}.\vec{\mathsf{zz}})\mathsf{PP}$
```

Fig. 12. Unforgeability Step 2: $\mathcal{B}$ playing EF-CMA

Some rare failure event may happen: during the selection of tt_0, we may obtain $\vec{z} = \mathsf{xx} + \mathsf{tt}_0.\vec{\mathsf{zz}} = 0$ making an invalid $\vec{z}$. Furthermore, during the selection of any t, we may obtain $t = -1$ making the change of variable fail. However, by the difference Lemma, we have

$$\mathsf{Adv}_{\mathcal{A}_1}^{\Gamma_1} \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{EF\text{-}CMA}} + \frac{q+1}{p}$$

□

B.6 Unlinkability

The role of the NIZK is important as it makes sure that the issuer must use either $b = 0$ or $b = 1$ and therefore hides only one bit. Without the NIZK, the issuer could use more than one bit with b and use this as a marker to link tokens to redeem to clients requesting a token. So, the client must verify the NIZK proof for unlinkability. The client must also verify $U \neq 0$, because $U = 0$ could be used by the issuer to mark a token.

The issuer knows which bit is hidden during an issuing session and can extract the hidden bit during redeem. Hence, we should only consider unlinkability when the bits are the same.

Theorem 5. ATHM *is* 2-UNLINK*-secure. More precisely, given an* UNLINK*-adversary* $\mathcal{A}$ *making oracle calls to* $\mathcal{O}_{\mathsf{query}}$ *with index set* $\mathcal{Q}_{\mathsf{query}}$*, there exist an adversary* $\mathcal{A}'$ *making a valid* pk_2 *without being extractable,* $\#\mathcal{Q}_{\mathsf{query}}$ SOUND*-adversaries* $\mathcal{B}_i$ *against the NIZK and* $\#\mathcal{Q}_{\mathsf{query}}$ *distinguishers* $\mathcal{D}_i$ *on* T_{ext} *such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{UNLINK}} \leq \frac{2}{n} + \mathsf{Adv}_{\mathcal{A}'}^{\mathsf{EXTRACT}} + \sum_{i \in \mathcal{Q}_{\mathsf{query}}} \mathsf{Adv}_{\mathcal{B}_i}^{\mathsf{SOUND}} + \sum_{i \in \mathcal{Q}_{\mathsf{query}}} \mathsf{Adv}_{\mathcal{D}_i}^{\mathsf{IND}}$$

The proof uses the fact that Π_2 is sound and that T_{ext} can be simulated using a trapdoor in a computationally indistinguishable manner.

Proof. We start with an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ playing the UNLINK_n game.

We assume that AT.ClientQuery verifies pk, at least at the first time is it run. Using the soundness of the proof of $\vec{x}, \vec{r}_x, \vec{y}, \vec{r}_y, \vec{y'}, \vec{r}_{y'}$ in pk_2 (based on that $\mathcal{A}_1$ is only running this unique proof), there is an extractor which extracts from $\mathcal{A}'$ those secrets, except with negligible probability $\mathsf{Adv}_{\mathcal{A}'}^{\mathsf{EXTRACT}}$. We reduce to a game Γ_0 in which this extraction succeeds. We have

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{UNLINK}} \leq \mathsf{Adv}_{\mathcal{A}}^{\Gamma_0} + \mathsf{Adv}_{\mathcal{A}'}^{\mathsf{EXTRACT}}$$

Second, we use the simulator of Π_1 to simulate all $\vec{T}_{\mathsf{ext}}$ and reduce to a protocol using no $\vec{T}_{\mathsf{ext}}$. For that, we define distinguishers $\mathcal{D}_i$ between a valid $\vec{T}_{\mathsf{ext}}$ and a simulated one. We obtain a game Γ_1 with no Π_1 and an adversary $\mathcal{A}_0$ such that

$$\mathsf{Adv}_{\mathcal{A}}^{\Gamma_0} \leq \mathsf{Adv}_{\mathcal{A}_0}^{\Gamma_1} + \sum_{i \in \mathcal{Q}_{\mathsf{query}}} \mathsf{Adv}_{\mathcal{D}_i}^{\mathsf{IND}}$$

Then, we reduce to a game Γ in which for every i, the $(m_i, \mathsf{query}_i, \mathsf{resp}_i)$ triplets, parsed as $\mathsf{query}_i = \vec{T}_i$ and $\mathsf{resp}_i = (U_i, \vec{V}_i, t_{i,S}, \pi_i)$, is such that there exists (b_i, d_i) such that $b_i \in \{0,1\}$, $U_i = d_i G$, and $\vec{V}_i = d_i(G.\vec{x} + b_i G.\vec{y} + m_i.\vec{y'} + t_{i,S} G.\vec{z} + \vec{T}_i)$. Note that (b_i, d_i) is unique when it exists. Thanks to the soundness of Π_2, we have $\#\mathcal{Q}_{\mathsf{query}}$ adversaries $\mathcal{B}_i$ to build a valid proof when no (b_i, d_i) exists, such that

$$\mathsf{Adv}_{\mathcal{A}_0}^{\Gamma_1} \leq \mathsf{Adv}_{\mathcal{A}_0}^{\Gamma} + \sum_{i \in \mathcal{Q}_{\mathsf{query}}} \mathsf{Adv}_{\mathcal{B}_i}^{\mathsf{SOUND}}$$

We deduce that $\vec{Q}_i = (\vec{x} + b_i \vec{y} + m_i \vec{y'} + t_i \vec{z}) P_i$ with $b_i \in \{0,1\}$ in the game Γ. We also note that all m_i $(i \in \mathcal{Q})$ must be equal to a common m as required in the UNLINK game.

The rest of the proof is an information theoretic argument for which complexities do not matter. Given $(\vec{x}, \vec{y}, \vec{y'}, \vec{z}, \vec{T}_i, U_i, \vec{V}_i, t_{i,S}, m, \pi_i)$ we can uniquely determine b_i. We observe that $(t_i, P_i) | (\vec{x}, \vec{y}, \vec{y'}, \vec{z}, \vec{T}_i, U_i, \vec{V}_i, t_{i,S}, m, \pi_i)$ is uniformly distributed as a pair composed of a scalar and a nonzero group element. Hence, whenever $\mathcal{A}_2$ returns $\mathcal{Q}$ and the list of resp_i, it determines the values of the b_i but the (t_i, P_i) to be released are still uniform. After permutation, $(t_{\sigma(i)}, P_{\sigma(i)}, \vec{Q}_{\sigma(i)})$ has a value of $\vec{Q}_{\sigma(i)}$ which is imposed by $\vec{Q}_{\sigma(i)} = (\vec{x} + b_{\sigma(i)} \vec{y} + m\vec{y'} + t_{\sigma(i)} \vec{z}) P_{\sigma(i)}$ so brings $b_{\sigma(i)}$ as only information.

This reduces to the following game: the adversary chooses a list of bits $(b_i)_{i \in \mathcal{Q}}$ with $\#\mathcal{Q} \geq n$, the game selects a random i^* and a random permutation σ then provides b_{i^*} and $(b_{\sigma(i)})_{i \in \mathcal{Q}}$ to the adversary, and the adversary finally makes a guess i and win if and only if $i = i^*$. If the adversary puts n_0 zeros and n_1 ones, the adversary can only win with probability $\frac{1}{n_0}$ when it is a zero (which happens with probability $\frac{n_0}{n_0+n_1}$), and with probability $\frac{1}{n_1}$ when it is a one (which happens with probability $\frac{n_1}{n_0+n_1}$). Overall, the adversary wins with probability $\frac{2}{n_0+n_1}$ which is at most $\frac{2}{n}$ since $n_0 + n_1 = \#\mathcal{Q} \geq n$. □

B.7 Privacy of Metadata

Theorem 6. *Under Assumption 2,* ATHM *is* PMB*-secure. More precisely, given an* PMB*-adversary* $\mathcal{A}$ *making* q *oracle calls to* $\mathcal{O}_{\mathsf{sign}}$ *and* q_{Hash} *calls to the random oracle, there exists an* $\mathsf{IND\text{-}CMA}$*-adversary* $\mathcal{B}$ *making* $q+3$ *oracle calls to* $\mathcal{O}_{\mathsf{MAC}}$ *such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PMB}} \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{IND\text{-}CMA}} + 4\frac{q+1}{p} + 2\frac{q_{\mathsf{Hash}}}{p^6}$$

This result needs $\Pi_1.\mathsf{Extract}$ to be succeed if an only if $\Pi_1.\mathsf{Verify}$ returns true and Π_2 to be perfectly simulatable with a trapdoor using $\Pi_2.\mathsf{SimKeyGen}$ and $\Pi_2.\mathsf{Simulate}$.

Proof. We start with an adversary $\mathcal{A}$ playing the PMB game against ATHM and we transform it into a sequence of adversaries playing other games with same success.

We start by the same first step as in the proof of unforgeability. We define an adversary $\mathcal{A}_1$ who runs $\mathcal{A}$ but simulates π in $\mathit{\Pi}_2$. With a trapdoor td_1 (in the CRS model), $\mathcal{A}_1$ can also extract $(t_C, \vec{r}) = \mathit{\Pi}_1.\mathsf{Extract}(\mathsf{td}_1, \vec{T}_{\mathsf{ext}})$. Then, $\mathcal{A}_1$ can call a MAC oracle to build $(t, P, \vec{Q})$ such that t and P are random and $\vec{Q} = (\vec{x}+b\vec{y}+m\vec{y}'+t\vec{z})P$. Then, $\mathcal{A}_1$ can set $t_S = t - t_C$, $U = P$, and $\vec{V} = \vec{Q}+\vec{r}U$ to return to $\mathcal{A}$. More precisely, the adversary $\mathcal{A}_1$ is fully written in Figure 13. This way, an adversary $\mathcal{A}$ for our scheme transforms into an adversary $\mathcal{A}_1$ in the $\mathit{\Gamma}_{b^*}$ game in Figure 13.

The extraction procedure is such that $\mathit{\Pi}_1.\mathsf{Extract}(\mathsf{td}, \vec{T}_{\mathsf{ext}})$ fails is equivalent to $\mathit{\Pi}_1.\mathsf{Verify}(\vec{T}, \vec{T}_{\mathsf{ext}}, G, \vec{Z})$ is false. As forging π is perfect, $\mathcal{A}_1$ playing $\mathit{\Gamma}_{b^*}$ perfectly simulates $\mathcal{A}$ playing PMB_{b^*}.

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PMB}} \leq \mathsf{Adv}_{\mathcal{A}_1}^{\mathit{\Gamma}} + \frac{q_{\mathsf{Hash}}}{p^6}$$

In the next step, we remove G, $\vec{Z}$, and pk_2 as in the proof of unforgeability, with the same change of variable. We define an adversary $\mathcal{B}$ plyaing the $\mathsf{IND\text{-}CMA}$ game in Figure 14.

For $\mathcal{B}$ in $\mathsf{IND\text{-}CMA}_{b^*}$ to simulate $\mathcal{A}_1$ in $\mathit{\Gamma}_{b^*}$, there are a few things which can go wrong. Like in the previous proof, we can select the secrets such that $\vec{\mathsf{x}}\mathsf{x} + \mathsf{tt}_0.\vec{\mathsf{z}}\mathsf{z} = 0$ or any t equal to -1 which make the change of variable fail. This happens with probability at most $\frac{q+1}{p}$. We can also select $t^* = \mathsf{tt}_0$ (with probability $\frac{1}{p}$). Finally, any t in $\mathcal{O}_{\mathsf{MAC}}$ in $\mathit{\Gamma}_{b^*}$ can be selected equal to t^* which makes $\mathcal{O}'_{\mathsf{MAC}}$ abort. This happens with probability at most $\frac{q}{p}$. By the difference Lemma, we obtain

$$|\Pr[\mathit{\Gamma}_{b^*}(\mathcal{A}_1) \to 1] - \Pr[\mathsf{IND\text{-}CMA}_{b^*}(\mathcal{B}) \to 1]| \leq 2\frac{q+1}{p}$$

We deduce

$$\left|\mathsf{Adv}_{\mathcal{A}_1}^{\mathit{\Gamma}} - \mathsf{Adv}_{\mathcal{B}}^{\mathsf{IND\text{-}CMA}}\right| \leq 4\frac{q+1}{p}$$

□

B.8 Extensions of ATHM

We could add public verifiability of tokens by using a pairing. This can only be done with no private metadata (otherwise privacy is broken). We could also consider batching too like in PMBT, so that the issuer could make a single proof for a batch of responses. Clearly, algebraic MACs offer lots of flexibility for extensions. This is left as future work.

C Straightforward Scheme from KVAC

We apply the construction from Chase et al. [6,7] to build a KVAC with only two attributes:

```
Adversary $\mathcal{A}_1^{\mathcal{O}_{\mathsf{MAC}},\mathcal{O}_{\mathsf{chal}},\mathcal{O}_{\mathsf{verify}},\mathcal{O}_{\mathsf{valid}}}(\mathsf{gp}, p, G, Z)$:
 1: $\mathsf{Setup}_1(\mathsf{gp}, p, G) \to (\mathsf{crs}_1, \mathsf{td}_1)$
 2: $\mathsf{Setup}_2(\mathsf{gp}, p, G) \to (\mathsf{crs}_2, \mathsf{td}_2)$
 3: $\mathsf{crs} \leftarrow (\mathsf{gp}, p, G, \mathsf{crs}_1, \mathsf{crs}_2)$
 4: $\mathsf{KeyGen}_1(\mathsf{crs}, \vec{Z}) \to (\mathsf{pk}_1, \mathsf{sk}_1)$
 5: $\mathit{\Pi}_2.\mathsf{SimKeyGen}(\mathsf{crs}, \vec{Z}) \to \mathsf{pk}_2$
 6: $\mathsf{pk} \leftarrow (\mathsf{crs}, Z, \mathsf{pk}_1, \mathsf{pk}_2)$
 7: return $\mathcal{A}^{\mathcal{O}_{\mathsf{sign}},\mathcal{O}'_{\mathsf{sign}},\mathcal{O}_{\mathsf{read}},\mathcal{O}_{\mathsf{valid}}}(\mathsf{crs}, \mathsf{pk})$

Subroutine $\mathcal{O}_{\mathsf{sign}}(b, m, \mathsf{query})$:
 8: parse $\mathsf{query} = (\vec{T}, \vec{T}_{\mathsf{ext}})$
 9: $\mathit{\Pi}_1.\mathsf{Extract}(\mathsf{td}_1, \mathsf{sk}_1, \vec{T}_{\mathsf{ext}}) \to (t_C, \vec{r})$
10: if extraction failed or $\vec{T} \neq t_C\vec{Z} + \vec{r}G$
    then return $\perp$
11: pick $t \leftarrow\$ \mathbb{Z}_p$
12: $\mathcal{O}_{\mathsf{MAC}}(b, m, t) \to (P, \vec{Q})$
13: $t_S \leftarrow t - t_C$
14: $U \leftarrow P$
15: $\vec{V} \leftarrow \vec{Q} + \vec{r}U$
16: $\pi \leftarrow \mathit{\Pi}_2.\mathsf{Simulate}(U, \vec{V}, t_S, \mathsf{crs}, \mathsf{pk}, \mathsf{td}_2)$
17: return $(U, \vec{V}, t_S, \pi)$

Subroutine $\mathcal{O}'_{\mathsf{sign}}(m, \mathsf{query})$:
18: parse $\mathsf{query} = (\vec{T}, \vec{T}_{\mathsf{ext}})$
19: $\mathsf{Extract}(\mathsf{td}_1, \mathsf{sk}_1, \vec{T}_{\mathsf{ext}}) \to (t_C, \vec{r})$
20: if extraction failed or $\vec{T} \neq t_C\vec{Z} + \vec{r}G$
    then return $\perp$
21: pick $t \leftarrow\$ \mathbb{Z}_p$
22: $\mathcal{O}_{\mathsf{chal}}(m, t) \to (P, \vec{Q})$
23: $t_S \leftarrow t - t_C$
24: $U \leftarrow P$
25: $\vec{V} \leftarrow \vec{Q} + \vec{r}U$
26: $\pi \leftarrow \mathit{\Pi}_2.\mathsf{Simulate}(U, \vec{V}, t_S, \mathsf{crs}, \mathsf{pk}, \mathsf{td}_2)$
27: return $(U, \vec{V}, t_S, \pi)$

Subroutine $\mathcal{O}_{\mathsf{read}}(m, t, \sigma)$:
28: if flag then return $\perp$
29: if $\mathcal{O}_{\mathsf{verify}}(0, m, t, \sigma)$ then return 0
30: if $\mathcal{O}_{\mathsf{verify}}(1, m, t, \sigma)$ then return 1
31: return $\perp$
```

```
Game $\mathit{\Gamma}_{b^*}$:
 1: $\mathsf{Setup}_0(1^\lambda) \to (\mathsf{gp}, p, G)$
 2: pick $\vec{x} \leftarrow\$ \mathcal{E}_x$, $\vec{y} \leftarrow\$ \mathcal{E}_y$, $\vec{y'} \leftarrow\$ \mathcal{E}_{y'}$,
    $\vec{z} \leftarrow\$ \mathcal{E}_z$
 3: set $\vec{Z} \leftarrow \vec{z}G$
 4: initialize $i \leftarrow 0$, $\mathsf{flag} \leftarrow \mathsf{false}$
 5: return $\mathcal{A}_1^{\mathcal{O}_{\mathsf{MAC}},\mathcal{O}_{\mathsf{chal}},\mathcal{O}_{\mathsf{verify}},\mathcal{O}_{\mathsf{valid}}}$
    $(\mathsf{gp}, p, G, \vec{Z})$

Oracle $\mathcal{O}_{\mathsf{MAC}}(b, m, t)$:
 6: pick a nonzero group element $P$
 7: $\vec{Q} \leftarrow (\vec{x} + b\vec{y} + m\vec{y'} + t\vec{z})P$
 8: return $(P, \vec{Q})$

Oracle $\mathcal{O}_{\mathsf{chal}}(m, t)$:
 9: if flag then return $\perp$
10: $\mathsf{flag} \leftarrow \mathsf{true}$
11: $(m^*, t^*) \leftarrow (m, t)$
12: $\mathcal{O}_{\mathsf{MAC}}(b^*, m^*, t^*) \to (P^*, \vec{Q}^*)$
13: return $(P^*, \vec{Q}^*)$

Oracle $\mathcal{O}_{\mathsf{verify}}(b, m, t, P, \vec{Q})$:
14: if flag and $b \in \{0,1\}$ and $(t, m) =$
    $(t^*, m^*)$ then return $\perp$
15: if $P = 0$ then return false
16: return whether $\vec{Q} = (\vec{x} + b\vec{y} + m\vec{y'} +$
    $t\vec{z})P$

Oracle $\mathcal{O}_{\mathsf{valid}}(t, m, P, \vec{Q})$:
17: if $P = 0$ then return false
18: return whether $\exists b \in \{0,1\}$ $\vec{Q} =$
    $(\vec{x} + b\vec{y} + m\vec{y'} + t\vec{z})P$
```

Fig. 13. Privacy Step 1: $\mathcal{A}_1$ playing $\mathit{\Gamma}$

```
Adversary $\mathcal{B}^{\mathcal{O}'_{\mathsf{MAC}},\mathcal{O}'_{\mathsf{chal}},\mathcal{O}'_{\mathsf{verify}},\mathcal{O}'_{\mathsf{valid}}}(\mathsf{gp},p)$:
 1: $\mathsf{bb}_0 \leftarrow 0$
 2: $\mathsf{mm}_0 \leftarrow 0$
 3: $\mathsf{tt}_0 \leftarrow 1$
 4: $\mathcal{O}'_{\mathsf{MAC}}(\mathsf{bb}_0,\mathsf{mm}_0,\mathsf{tt}_0) \rightarrow (G,\vec{Z})$
 5: return $\mathcal{A}_1^{\mathcal{O}_{\mathsf{MAC}},\mathcal{O}_{\mathsf{chal}},\mathcal{O}_{\mathsf{verify}},\mathcal{O}_{\mathsf{valid}}}(\mathsf{gp},p,G,\vec{Z})$

Subroutine $\mathcal{O}_{\mathsf{verify}}(b,m,t,P,\vec{Q})$:
 6: $\mathsf{bb} \leftarrow \frac{b}{1+t}$, $\mathsf{mm} \leftarrow \frac{m}{1+t}$, $\mathsf{tt} \leftarrow \frac{t.\mathsf{tt}_0}{1+t}$
 7: $\mathsf{PP} \leftarrow P$, $\vec{\mathsf{QQ}} \leftarrow (1+t)^{-1}\vec{Q}$
 8: return $\mathcal{O}'_{\mathsf{verify}}(\mathsf{bb},\mathsf{mm},\mathsf{tt},\mathsf{PP},\vec{\mathsf{QQ}})$

Subroutine $\mathcal{O}_{\mathsf{valid}}(m,t,P,\vec{Q})$:
 9: $\mathsf{bb} \leftarrow 0$, $\mathsf{bb}' \leftarrow \frac{1}{1+t}$
10: $\mathsf{mm} \leftarrow \frac{m}{1+t}$, $\mathsf{tt} \leftarrow \frac{t.\mathsf{tt}_0}{1+t}$
11: $\mathsf{PP} \leftarrow P$, $\vec{\mathsf{QQ}} \leftarrow (1+t)^{-1}\vec{Q}$
12: return $\mathcal{O}'_{\mathsf{valid}}(\mathsf{bb},\mathsf{bb}',\mathsf{mm},\mathsf{tt},\mathsf{PP},\vec{\mathsf{QQ}})$

Subroutine $\mathcal{O}_{\mathsf{MAC}}(b,m,t)$:
13: $\mathsf{bb} \leftarrow \frac{b}{1+t}$, $\mathsf{mm} \leftarrow \frac{m}{1+t}$, $\mathsf{tt} \leftarrow \frac{t.\mathsf{tt}_0}{1+t}$
14: $\mathcal{O}'_{\mathsf{MAC}}(\mathsf{bb},\mathsf{mm},\mathsf{tt}) \rightarrow (\mathsf{PP},\vec{\mathsf{QQ}})$
15: $P \leftarrow \mathsf{PP}$, $\vec{Q} \leftarrow (1+t)\vec{\mathsf{QQ}}$
16: return $(P,\vec{Q})$

Subroutine $\mathcal{O}_{\mathsf{chal}}(m,t)$:
17: $\mathsf{bb} \leftarrow 0$, $\mathsf{bb}' \leftarrow \frac{1}{1+t}$
18: $\mathsf{mm} \leftarrow \frac{m}{1+t}$, $\mathsf{tt} \leftarrow \frac{t.\mathsf{tt}_0}{1+t}$
19: $\mathcal{O}'_{\mathsf{chal}}(\mathsf{bb},\mathsf{bb}',\mathsf{mm},\mathsf{tt}) \rightarrow (\mathsf{PP},\vec{\mathsf{QQ}})$
20: $P \leftarrow \mathsf{PP}$
21: $\vec{Q} \leftarrow (1+t)\vec{\mathsf{QQ}}$
22: return $(P,\vec{Q})$
```

```
Game $\mathsf{IND\text{-}CMA}_{b^*}$:
 1: $\mathsf{flag} \leftarrow \mathsf{false}$, $\mathsf{mm}^* \leftarrow \perp$
 2: $\mathsf{Setup}_0(1^\lambda) \rightarrow (\mathsf{gp},p,G_0)$
 3: pick $\vec{\mathsf{xx}} \leftarrow\!\$\ \mathcal{E}_x$, $\vec{yy} \leftarrow\!\$\ \mathcal{E}_y$, $\vec{\mathsf{yy}}' \leftarrow\!\$\ \mathcal{E}_{y'}$, $\vec{\mathsf{zz}} \leftarrow\!\$\ \mathcal{E}_z$
 4: return $\mathcal{B}^{\mathcal{O}'_{\mathsf{MAC}},\mathcal{O}'_{\mathsf{chal}},\mathcal{O}'_{\mathsf{verify}},\mathcal{O}'_{\mathsf{valid}}}(\mathsf{gp},p)$

Oracle $\mathcal{O}'_{\mathsf{verify}}(\mathsf{bb},\mathsf{mm},\mathsf{tt},\mathsf{PP},\vec{\mathsf{QQ}})$:
 5: if $\mathsf{flag}$ and $\mathsf{bb} \in \{\mathsf{bb}^*_0,\mathsf{bb}^*_1\}$ and $(\mathsf{mm},\mathsf{tt}) = (\mathsf{mm}^*,\mathsf{tt}^*)$ then return $\perp$
 6: if $\mathsf{PP} = 0$ then return $\mathsf{false}$
 7: return whether $\vec{\mathsf{QQ}} = (\vec{\mathsf{xx}} + \mathsf{bb}.\vec{\mathsf{yy}} + \mathsf{mm}.\vec{\mathsf{yy}}' + \mathsf{tt}.\vec{\mathsf{zz}})\mathsf{PP}$

Oracle $\mathcal{O}'_{\mathsf{valid}}(\mathsf{bb}_0,\mathsf{bb}_1,\mathsf{mm},\mathsf{tt},\mathsf{PP},\vec{\mathsf{QQ}})$:
 8: if $\neg\mathsf{flag}$ then return $\perp$
 9: if $(\mathsf{tt},\mathsf{mm}) = (\mathsf{tt}^*,\mathsf{mm}^*)$ and $\{\mathsf{bb}_0,\mathsf{bb}_1\} \neq \{\mathsf{bb}^*_0,\mathsf{bb}^*_1\}$ then return $\perp$
10: if $\mathsf{PP} = 0$ then return $\mathsf{false}$
11: if $\vec{\mathsf{QQ}} = (\vec{\mathsf{xx}} + \mathsf{bb}_0.\vec{\mathsf{yy}} + \mathsf{mm}.\vec{\mathsf{yy}}' + \mathsf{tt}.\vec{\mathsf{zz}})\mathsf{PP}$ then return $\mathsf{true}$
12: if $\vec{\mathsf{QQ}} = (\vec{\mathsf{xx}} + \mathsf{bb}_1.\vec{\mathsf{yy}} + \mathsf{mm}.\vec{\mathsf{yy}}' + \mathsf{tt}.\vec{\mathsf{zz}})\mathsf{PP}$ then return $\mathsf{true}$
13: return $\mathsf{false}$

Oracle $\mathcal{O}'_{\mathsf{MAC}}(\mathsf{bb},\mathsf{mm},\mathsf{tt})$:
14: if $\mathsf{flag}$ and $(\mathsf{bb},\mathsf{mm},\mathsf{tt}) \in \{(\mathsf{bb}_0,\mathsf{mm}^*,\mathsf{tt}^*),(\mathsf{bb}_1,\mathsf{mm}^*,\mathsf{tt}^*)\}$ then return $\perp$
15: increment $i$
16: $(\mathsf{bb}_i,\mathsf{mm}_i,\mathsf{tt}_i) \leftarrow (\mathsf{bb},\mathsf{mm},\mathsf{tt})$
17: pick a nonzero group element $\mathsf{PP}_i$
18: $\vec{\mathsf{QQ}}_i \leftarrow (\vec{\mathsf{xx}} + \mathsf{bb}_i.\vec{\mathsf{yy}} + \mathsf{mm}_i.\vec{\mathsf{yy}}' + \mathsf{tt}_i.\vec{\mathsf{zz}})\mathsf{PP}_i$
19: return $(\mathsf{PP}_i,\vec{\mathsf{QQ}}_i)$

Oracle $\mathcal{O}'_{\mathsf{chal}}(\mathsf{bb},\mathsf{bb}',\mathsf{mm},\mathsf{tt})$:
20: if $\mathsf{flag}$ or $\exists i\ (\mathsf{bb}_i,\mathsf{mm}_i,\mathsf{tt}_i) \in \{(\mathsf{bb},\mathsf{mm},\mathsf{tt}),(\mathsf{bb}',\mathsf{mm},\mathsf{tt})\}$ then return $\perp$
21: $(\mathsf{bb}^*_0,\mathsf{bb}^*_1,\mathsf{mm}^*,\mathsf{tt}^*) \leftarrow (\mathsf{bb},\mathsf{bb}',\mathsf{mm},\mathsf{tt})$
22: $\mathsf{flag} \leftarrow \mathsf{true}$
23: pick a nonzero group element $\mathsf{PP}^*$
24: $\vec{\mathsf{QQ}}^* \leftarrow (\vec{\mathsf{xx}} + \mathsf{bb}^*_{b^*}\vec{\mathsf{yy}} + \mathsf{mm}^*\vec{\mathsf{yy}}' + \mathsf{tt}^*\vec{\mathsf{zz}})\mathsf{PP}^*$
25: return $(\mathsf{PP}^*,\vec{\mathsf{QQ}}^*)$
```

Fig. 14. Privacy Step 2: $\mathcal{B}$ playing $\mathsf{IND\text{-}CMA}$

- t which should be known by the client only;
- b which should be known by the server only, and which should be a bit.

Note that the original KVAC construction does not consider server-private attributes but it looks fairly straightforward to add it, except when it comes to prove that they respect a specific format. Namely, the attribute b must be a bit.

The common group parameters include two generators G and H. Public parameters include $Y = yG$, $Z = zG$, and a commitment $C_x = xG + r_xH$.

Essentially, the client first selects an ElGamal key pair (k, K), $K = kG$. Then, the client encrypts tG into the ciphertext (T, T'). It must come with a proof for correct computation π_c. Then, the server massages the encryption (T, T') of tG to obtain an encryption (W, W') of $V = d(x + by + tz)G$ which is returned together with $U = dG$. It must come with a proof for correct computation π_s. The client then decrypts (W, W') to get V then rerandomizes (U, V) to obtain the tag (P, Q).

The proof π_c must prove the knowledge of (t, r) such that

$$t \begin{pmatrix} 0 \\ G \end{pmatrix} + r \begin{pmatrix} G \\ K \end{pmatrix} = \begin{pmatrix} T \\ T' \end{pmatrix}$$

We estimate it to 3 multiplications for the client, 4 multiplications for the server, and a proof of 3 scalars.

The proof π_s must prove $b \in \{0, 1\}$ and the knowledge of $(x, y, z, r_x, \frac{1}{d}, r')$ such that

$$x \begin{pmatrix} G \\ 0 \\ 0 \\ 0 \\ 0 \\ G \end{pmatrix} + y \begin{pmatrix} 0 \\ G \\ 0 \\ 0 \\ 0 \\ bG \end{pmatrix} + z \begin{pmatrix} 0 \\ 0 \\ G \\ 0 \\ T \\ T' \end{pmatrix} + r_x \begin{pmatrix} H \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{1}{d} \begin{pmatrix} 0 \\ 0 \\ 0 \\ U \\ -W \\ -W' \end{pmatrix} + r' \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ G \\ K \end{pmatrix} = \begin{pmatrix} C_x \\ Y \\ Z \\ G \\ 0 \\ 0 \end{pmatrix}$$

There may be ways to optimize this proof. If we proceed like our construction which releases $C = bG + \mu H$, do the OR proof, then use C, we obtain

- for the OR proof: 12 multiplications for the client, 8 multiplications for the server, and a proof including 1 group element and 4 scalars;
- for the rest of the proof: 13 multiplications for the client, 12 multiplications for the server, and a proof including 5 scalars.

Further note that PMB security is not guaranteed by the KVAC construction. On top of that, UNLINK security should also be revisited as we added a server-private attribute which is not considered in the KVAC construction.

KVAC.Client(G, H, C_x, Y, Z, t)		KVAC.IssueToken$(G, H, (x, y, z, r_x), b)$
private input: t		private input: $(x, y, z), b$
$k, r \leftarrow\$\ \mathbb{Z}_p$		
$K := kG$		
$T := rG$		
$T' := tG + rK$		
$\pi_c := \mathit{\Pi}_1.\mathsf{Prove}(t, r; \ldots)$		
	$\xrightarrow{K, T, T', \pi_c}$	
		$\mathit{\Pi}_1.\mathsf{Verify}(\pi_c, \ldots)$
		$d, r' \leftarrow\$\ \mathbb{Z}_p$
		$U := dG$
		$W := dzT + dr'G$
		$W' := d(x + by)G + dzT' + dr'K$
		$\pi_s := \mathit{\Pi}_2.\mathsf{Prove}(d, r', b; \ldots)$
	$\xleftarrow{U, W, W', \pi_s}$	
$\mathit{\Pi}_2.\mathsf{Verify}(\pi_s, \ldots)$		
$c \leftarrow\$\ \mathbb{Z}_p^*$		
$P := cU$		
$Q := c(W' - kW)$		
$\sigma := (P, Q)$		
output: (t, σ)		

Fig. 15. KVAC Construction with Private Attributes t and b.