Anonymous Counting Tokens

Fabrice Benhamouda[1]*, Mariana Raykova[2], and Karn Seth[2]

[1] Amazon Web Services
[2] Google

Abstract. We introduce a new primitive called *anonymous counting tokens* (ACTs) which allows clients to obtain blind signatures or MACs (aka tokens) on messages of their choice, while at the same time enabling issuers to enforce rate limits on the number of tokens that a client can obtain for each message. Our constructions enforce that each client will be able to obtain only one token per message and we show a generic transformation to support other rate limiting as well. We achieve this new property while maintaining the unforgeability and unlinkability properties required for anonymous tokens schemes. We present four ACT constructions with various trade-offs for their efficiency and underlying security assumptions. One construction uses factorization-based primitives and a cyclic group. It is secure in the random oracle model under the q-DDHI assumption (in a cyclic group) and the DCR assumption. Our three other constructions use bilinear maps: one is secure in the standard model under q-DDHI and SXDH, one is secure in the random oracle model under SXDH, and the most efficient of the three is secure in the random oracle model and generic bilinear group model.

1 Introduction

Counting unique users can be a useful signal for different applications to measure service usage and user interest. In many contexts, however, the content for which we want to measure interest may be sensitive, so we would like to guarantee anonymity for the user while still providing accurate counts. The anonymity property becomes challenging when untrustworthy users may try to inflate the counts. As a concrete example, we consider the k-anonymity server developed in the context of Privacy Sandbox [Gra22]. The goal of this server is to count how many users have joined different user interest groups. Users should not be linkable to any specific interest groups. At the same time, it is important to obtain an accurate count of the number of users in each interest group. In particular, each user should not be counted multiple times. In addition, users should be allowed to join as many interest groups as they wish.

There is seemingly a tension between the desirable anonymity that does not allow mapping the count contribution to the user identity and the ability to bound contributions from each user. Multiparty computation (MPC) [Ode09] and in particular secure aggregation constructions [BIK$^+$17, BBG$^+$20] enable computing aggregates over user inputs while maintaining privacy for concrete contributions. However, these solutions do not allow users to be anonymous while at the same time limiting the rate of their input contributions.

Anonymous credential tools such as blind signatures [Cha82] and anonymous tokens [DGS$^+$18, KLOR20, SS22] provide capabilities to convey trust across different contexts while providing anonymity. In the setting of anonymous tokens, during token issuance, the user identity is known to an issuer who can provide a token that encodes a limited amount of information. The token is associated with a user-provided message or a random message that the issuer should not learn. In our initial example of counting the number of users in each interest group, the message would be the interest group name. The token can later be redeemed in a different context where the user is anonymous. At redemption, the message is revealed. In order to be able to use these tokens to count the number of users in each interest group, it is crucial that each user can only contribute a single token. In other words, anonymous tokens redemption (or issuance) should be restricted (or rate limited) to a single token per user and per message. More generally, anonymous tokens allowing redemption of a small fixed number of tokens per user and per message can be considered.

A recent IETF draft proposed by four large tech companies (Google, Apple, Cloudflare, Fastly) [HIP$^+$22] is highlighting two other applications of such rate-limited or counting tokens: rate-limiting anonymous tokens

* Work done while employed at Algorand Foundation, prior to joining Amazon.

per website (or "origin") to avoid abuse, and implementing metered paywall for a given website. In both cases, anonymous tokens need to be associated with the website, and rate limits need to be applied per message. The IETF draft proposes a solution that relies on two non-colluding servers: an attester and an issuer. The attester sees the information used for rate limiting in the clear (that is the "origin" or website in the applications above). Instead, if we were able to design an anonymous token scheme restricting each user to only receive a small number of tokens per (hidden, underlying) message, we could provide a solution for the IETF draft applications without the need for two separate servers.

The challenge in designing the rate-limiting capability on the private message authenticated in anonymous tokens lies in the following fact: users should be able to obtain blind tokens for many messages (e.g., be able to contribute to the counts for many different user interest groups in the first example, or visiting many different paywall-metered websites in the last example). And all these messages should remain hidden from the issuer. Only violations of the rate-limiting rules should be detectable before such tokens are redeemed. At the same time, different users should be able to obtain anonymous tokens for the same message, as many users will join the same interest group or visit the same website. Therefore tokens for the same message from different users need to be distinct. In particular, tokens cannot computed deterministically from the message (as it is the case in Privacy Pass [DGS+18]).

We note that anonymous tokens with public or private metadata such as [SS22, CDV23] do not help building the applications above. Indeed, in these schemes the metadata needs to be revealed to the issuer in order for the issuer to be able to apply rate limits. In the interest group example, this means the issuer would know which interest group a user is joining.

Contributions. In this paper we propose a new notion of anonymous tokens which we call *anonymous counting tokens* (ACTs). This primitive offers an additional rate-limiting property that guarantees that no user will be able to redeem with the same verifier more than one token for the same message. Conceptually there are two approaches to enforcing the rate-limiting property in the anonymous token functionality. This can be done either at issuance by enabling the issuer to detect repeated token requests for the same message from the same user, or at redemption by enabling the verifier to identify if two tokens for the same message were issued to the same user. With the first approach, the challenge is to preserve the blind property of the requests as long as there are no repeating message requests, and to reveal only the one bit information whether a message in a request has been queried before. With the second approach, the challenge is to enable the verifier to detect when the two tokens for the same message were issued to the same user while preserving the unlinkability property.

We present two conceptual approaches for building ACTs. One enables rate limiting at issuance and one enables it at redemption. Both of them assume that each user registers a public key with the issuer and this public key enables the rate limiting of one token per message per user. Recall that at issuance, users identify themselves to the issuer and can thus be associated to their registered public keys. Note that such registration is necessary: if there was no registration mechanism, tokens would information theoretically be completely independent of the user identity and it would be impossible to ensure a given user does not create and redeem two tokens for the same message (unless tokens are deterministic functions of messages in which case the issuer could know when two different users ask the same message, which in turn would break unlinkability).

Our first construction uses a PRF evaluation as the token issuance mechanism. This mechanism has been leveraged in previous anonymous token constructions [DGS+18, KLOR20, SS22]. Our first construction is in the random oracle model (ROM) and relies on the q-Decisional Diffie-Hellman Inversion assumption (q-DDHI) assumption in a group of prime order.

Our second set of constructions leverages the notion of equivalence class signatures (EQS) [FG18, FHS19] to construct an ACT scheme. Existing EQS schemes rely on bilinear maps. We present three instantiations of our EQS-based ACT construction. The first one is in the standard model and assumes the SXDH and q-DDHI assumptions over bilinear groups to support short $O(\log \lambda)$ messages (λ is the security parameter). The second instantiation is proven in the ROM under just the SXDH assumption and supports any length of message. The last instantiation uses much stronger security assumptions: it is only proven in the ROM and generic bilinear group model (GBGM) but achieves significantly shorter tokens. Our three instantiations are based on two generic transforms of EQS into ACT, however, our third instantiation is an optimization whose security is proven directly in the ROM and GBGM and does not directly follow from the security of the generic transform.

Tables 1 and 2 summarize the communication costs and assumptions trade-offs of our constructions. Our four constructions are the four first constructions of anonymous counting tokens: there are no prior such constructions. We provide four different constructions as parties implementing anonymous counting tokens may have different preferences for cryptographic assumptions and tools used in the constructions. For example, in enterprise products (targeted by the IETF draft [HIP+22]), adding pairing-based libraries can be quite challenging due to non-technical reasons (e.g., audit requirements, complex approval process, etc).

Tables 1 and 2 also contain comparisons to selected related works which do *not* achieve the anonymous *counting* tokens property. We include them for an informative comparison of what it takes to add the additional properties we need. Privacy Pass [DGS+18] achieves unforgeability and unlinkability, but has no notion of an underlying message on which rate limiting can be done. [TCR+22] extends Privacy Pass to support public metadata, which can be viewed as a message. However, the public metadata is revealed to both the issuer and the verifier and thus cannot be used in our context. [FHS19] enables multi-show anonymous credentials, a very different primitive that allows a user to get a credential on a set of messages, but without blind issuance. The credential can be redeemed multiple times while remaining unlinkable with the issuance and the other redemptions. Several of our constructions can be seen as extensions of [FHS19] to support blind issuance and a throttling mechanism, so we include it to give a sense of the extra cost incurred for these additional properties. We note that the costs in Table 2 for [FHS19] are for a single message or attribute, but their scheme supports vectors of messages/attributes.

Table 1: Summary of our constructions and selected previous work

Cons.	PubV	Assumptions	$\|\mathsf{msg}\|$	Extra
4.3	×	DCR + q-DDHI	any	**counting**
5.1	✓	SXDH + $2^{\|\mathsf{msg}\|}$-DDHI$_{\mathbb{G}_1}$	$\mathcal{O}(\log(\lambda))$	**counting**
6.3	✓	ROM + SXDH	any	**counting**
8.1	✓	ROM + GBGM	any	**counting**
[DGS+18]	×	ElGamal-OMD	n/a	n/a
[TCR+22]	×	ROM + OM-Gap-SDHI	any	public metadata
[FHS19]	✓	ROM + GBGM	any	multi-show

Extra properties are properties beyond anonymous token base properties. Our constructions are the only anonymous counting tokens. See text for detail. PubV refers to public verifiability (i.e., whether anyone can verify a token from the public parameters).
Assumptions: DCR = decisional composite residuosity assumption, SXDH = symmetric external decisional Diffie-Hellman assumption, q-DDHI = decisional Diffie-Hellman inversion assumption (q is the number of signature queries made by the adversary), ROM = random oracle model, GBGM = generic bilinear group model, all of our constructions but the first one use pairings. ElGamal-OMD = ElGamal One-More-Decryption assumption. OM-Gap-SDHI = (m,n) One-More-Gap-Strong Diffie-Hellman Inversion assumption.

1.1 Technical Approach

Next, we overview the main technical challenges and ideas for our constructions.

ACT from PRF. We start with our first construction. It follows the idea of previous anonymous tokens schemes to make the tokens be PRF evaluations under the issuer's secret key. A first construction attempt might be to make the anonymous tokens deterministic. This is what Privacy Pass [DGS+18] does, where tokens are PRF evaluations of the users' messages. This, however, is possible in Privacy Pass only because the tokens there do not correspond to messages that have meaning for the application and instead are sampled at random for every token issuance. For our ACT scheme, we need the user to be able to choose the message. If the tokens are a deterministic function of the message alone (and not of the user identity), then the issuer will know when two users ask for the same message which would violate the unlinkability. Thus, we need to have a randomized issuance algorithm.

Table 2: Performance of our constructions and selected previous work

Cons.	$\lvert\mathsf{blindRequest}\rvert$	$\lvert\mathsf{blindToken}\rvert$	$\lvert\mathsf{tok}\rvert$
4.3	$5 \times \mathsf{Ped} + 4 \times \mathsf{CS} + 12 \times \mathbb{Z}_p + 11 \times \mathbb{Z}_N$	$1 \times \mathsf{Ped} + 1 \times \mathbb{Z}_p + 3 \times \mathbb{Z}_N$	$1 \times \mathbb{G}$
5.1	$7 \times \mathbb{G}_1 + 6 \times \mathbb{G}_2$	$5 \times \mathbb{G}_1 + 1 \times \mathbb{G}_2$	$23 \times \mathbb{G}_1 + 12 \times \mathbb{G}_2$
6.3	$2 \times \mathbb{Z}_p + 3 \times \mathbb{G}_1$	$5 \times \mathbb{G}_1 + 1 \times \mathbb{G}_2$	$23 \times \mathbb{G}_1 + 12 \times \mathbb{G}_2$
8.1	$2 \times \mathbb{Z}_p + 2 \times \mathbb{G}_1$	$2 \times \mathbb{G}_1 + 1 \times \mathbb{G}_2$	$3 \times \mathbb{G}_1 + 1 \times \mathbb{G}_2$
[DGS+18]	$1 \times \mathbb{G}$	$1 \times \mathbb{G} + 2 \times \mathbb{Z}_p$	λ
[TCR+22]	$1 \times \mathbb{G} + 1 \times \mathbb{Z}_p$	$1 \times \mathbb{G} + 2 \times \mathbb{Z}_p$	$1 \times \mathbb{G}$
[FHS19]	$\mathbb{Z}_p + \mathbb{G}_1$	$2 \times \mathbb{G}_1 + \mathbb{G}_2$	$3 \times \mathbb{G}_1 + \mathbb{G}_2 + 2 \times \mathbb{Z}_p$

Only our four constructions are anonymous **counting** tokens. The other constructions are just for reference and do not achieve the counting property. See text for detail.
Constructions 5.1 and 6.3 (long version) are instantiated using the EQS construction from 7.1, and the element M'_1 and M'_3 are not included in tok as they can be recomputed.
$\mathbb{G}$ denotes a cyclic group (and by extension a group element from $\mathbb{G}$) with order p, $\mathbb{G}_1$ and $\mathbb{G}_2$ are asymmetric bilinear maps groups, CS is a Camenish-Shoup ciphertext, Ped is a Pedersen commitment on a strong RSA group.
Using Edwards25519 [BDL+12] for $\mathbb{G}$, $1 \times \mathbb{G} = 32$ bytes. Using BLS12-381 [Bow17] as bilinear group, $1 \times \mathbb{G}_1 = 48$ bytes, $1 \times \mathbb{G}_2 = 96$ bytes. For both Edwards25519 and BLS12-381, $1 \times \mathbb{Z}_p = 32$ bytes. Using the NIST recommendation [Bar16] for 128-bit safe-RSA modulus (that is 3072 bits), $1 \times \mathsf{Ped} = 384$ bytes and $1 \times \mathsf{CS} = 1,536$ bytes. N refers to this RSA modulus, and $\log(N) = 3072$ bits.

To give insight into our construction, we start with an overview of some unsuccessful ideas for building anonymous counting tokens from the randomized version of the Okamoto-Schnorr Privacy Pass tokens introduced by Kreuter et al. [KLOR20]. Contrary to Privacy Pass, tokens generated by these schemes are randomized (and not deterministic). Like in Privacy Pass, the client sends the following blinded request to the issuer: $\mathsf{r} \cdot \mathsf{H}(\mathsf{msg})$, where r is chosen a random in $\mathbb{Z}_p$ by the client and where H is a hash function into a cyclic group $\mathbb{G}$ of order p (where DDH is hard). H is modeled as a random oracle in the proof and we use the additive notation for $\mathbb{G}$.

A first idea to construct an ACT is to have each client always use the same fixed randomness r to blind their requests. This would allow the issuer to detect when the same user makes two requests for the same message. We would argue the unlinkability of the requests by remarking those can also be seen as PRF evaluations using the key r, which is only known by the client. The client could provide the issuer with a commitment to this PRF key r (as part of the registration process). And the client would then prove the correctness of the message included in each blinded request with respect to the committed key, using a zero-knowledge proof.

This idea would actually work. Unfortunately, the resulting protocol would be quite inefficient as the zero-knowledge proof made by the client requires proving the correct evaluation of the hash function H. We note that the Pedersen hash ($\mathsf{H}(m) = m_1 G_1 + \cdots + m_n G_n$, where m_i is the i-th bit of the message m and G_i is a generator of $\mathbb{G}$) has good algebraic properties which may allow for an efficient zero-knowledge proof. However, it cannot be used in this setting since its linear properties enable an attack on the rate limiting as follows: $\mathsf{H}(1||0||\ldots||0) + \mathsf{H}(0||1||0||\ldots||0) = \mathsf{H}(1||1||0||\ldots||0)$, which allows to get a fresh token on $1||1||0||\ldots||0$ by querying messages $1||0||\ldots||0$ and $0||1||\ldots||0$.

Another option could be to use SNARK/STARK/Bulletproofs-friendly hash functions such as MiMC [AGR+16] and Poseidon [GKR+21]. However, this would first introduce the new non-standard assumptions that are used for the security of these hash functions. Additionally, depending on the proof system used, there will be a need for more assumptions (for example non-falsifiable assumptions for SNARKs). Furthermore, those proof techniques are generic and use circuits. This significantly increases the prover's complexity (which is used in the token request that is computed by weak user devices, in many cases). Furthermore, the proof needs to prove not only correct hash evaluation, but also mapping to the elliptic curve, and exponentiation/scalar multiplication. The Poseidon costs in Section 6 of [GKR+21] report around 40ms for a SNARK just proving correct Poseidon hash evaluation: the subsequent scalar multiplication would at least double this time to $>$ 80ms. These costs appeared to be much higherthan our approach. In addition, without a trusted setup, only STARKs and Bulletproofs can be used and they are about 10x more expensive than SNARKs for Poseidon evaluation.

Yet another tempting way to get an efficient scheme would be to only require the client to prove the blinded request is of the form $\mathsf{r} \cdot \mathsf{T}$ for some group element T, and not proving knowledge of msg such that

$\mathsf{T} = \mathsf{H}(\mathsf{msg})$. We may think we can argue unforgeability since the client will only be able to extract a valid signature if they know a correct hash preimage of T (due to the form of the tokens in [KLOR20]).

While the above reasoning does guarantee the regular unforgeability property, it fails to protect against the adversary being able to obtain two tokens for the same value, which is required by ACT. The issue stems from the fact that the resulting tokens are of the form $x\mathsf{H}(\mathsf{msg}) + yS$ where (x, y) is the secret key of the issuer and S is a random element that comes with the token. Now, we can observe that the token is additively homomorphic with respect to the hash of the message. With this observation, an attacker can obtain a token for message msg without directly asking for it, by additively sharing $\mathsf{H}(\mathsf{msg})$ as $A + B = \mathsf{H}(\mathsf{msg})$. Then the client can request two tokens, sending blind requests $\mathsf{r}A$ and $\mathsf{r}B$. The client can prove correctness for its requests as long as it does not have to prove knowledge of hash preimages of A and B. Then, using the additive properties of the tokens, it can recover a token for $\mathsf{H}(\mathsf{msg})$ which will be different from any previous token issued directly for that value.

Thus, we adopt a different pseudorandom function which has a structure that facilitates composition with sigma protocols for proof of correctness of evaluation (in particular, it does not involve a hash function). This is the Dodis-Yampolskiy verifiable pseudorandom function [DY05]. We instantiate it in a single group as a PRF without public verifiability, as in the work of Miao et al. [MPR$^+$20]. However, this on its own does not solve the question of the randomized issuance algorithm. One option is to add to the message a random value, which changes for every issuance, and make the token the PRF evaluation under the issuer's key on the message plus randomness. To ensure the rate limiting we will use a different PRF which the user evaluates only on the message under its registered key and provides this to the issuer during issuance to prove non-repeating message requests.

The way we choose to combine message and randomness as input for the PRF evaluation (to generate the token) leverages the function $\mathsf{F}(\mathsf{sk} = (\mathsf{u}, \mathsf{y}), \mathsf{msg}; \mathsf{r}) = (\mathsf{msg} + \mathsf{u} + \mathsf{r} \cdot \mathsf{y})$, which is used by Boneh and Boyen to construct short signatures without random oracle [BB04]. In the proof of their construction, they show how this function no longer has the limitation to short messages of the Dodis-Yampolskiy's variant as long as r is chosen at random and can be controlled by the reduction.

We further observe that the randomness r for the issuance needs to be chosen jointly by the client and the issuer. If the client can choose the randomness on its own, then it can force homomorphism of the tokens, which could create forgery issues similar to the one discussed above. If the issuer controls the randomness, then this becomes an easy fingerprinting mechanism, which violates unlinkability. While we can generate the randomness with an interactive coin-tossing protocol, we observe that the issuer's randomness does not need to be private with respect to the client since we just want to enforce that the randomness is chosen honestly. Thus, we apply the Fiat-Shamir transformation to generate the issuer's randomness in a non-interactive manner [FS87, AFK22].

ACT from EQS. The second general construction approach for ACT that we present views the tokens as signatures with certain homomorphic properties which allow the client to adapt a signature for a message $\vec{\mathsf{M}}$ to a signature of a transformed message $\vec{\mathsf{M}}' = f(\vec{\mathsf{M}})$. The set of allowed transformation f is limited and fixed. In particular, equivalence class (EQS) signatures [FG18, FHS19] enable the client to sign vectors of messages and the adaptation functionality allows the client to transform the signature into a signature of a new vector that is in the linear span of the signed message. This transformation is used as part of the blinding and unblinding operations in previous anonymous tokens and blind signatures constructions.

Taking this approach, of course, creates challenges for the rate-limiting property. Seemingly a client might be able to create multiple tokens from the same initial signature. To prevent this we need to embed the rate-limiting check but this time during redemption. This can be achieved similarly to the above construction using a PRF evaluation on the message with a key that each user commits with the issuer. The challenge when doing this check during redemption is to remove the link to the client identity while maintaining the ability to verify that the PRF value was generated by a key registered by a real client.

Our approach to satisfy the above requirements is to have the client embed a PRF evaluation on the message under their registered key in the blind signature request. We show how we can do this using two different PRF constructions $\mathsf{PRF}(\mathsf{u}, \mathsf{msg}) = \mathsf{u} \cdot \mathsf{H}(\mathsf{msg})$ and $\mathsf{PRF}(\mathsf{u}, \mathsf{msg}) = (\mathsf{msg} + \mathsf{u})^{-1} \cdot \mathsf{G}$ (the latter being the Dodis-Yampolskiy PRF). In the first case, unlike our first PRF-based ACT construction, we are able to use a random-oracle-based PRF without having to proof correct evaluation (in zero-knowledge) of the hash function H. Concretely, the EQS construction allows us to sign vectors of messages. And we sign a vector of the form $\mu \cdot (\mathsf{G}, \mathsf{H}(\mathsf{msg}), \mathsf{u} \cdot \mathsf{H}(\mathsf{msg}))$. Thus, the client just needs to prove the DDH relation between $(\mathsf{H}(\mathsf{msg}), \mathsf{u} \cdot \mathsf{H}(\mathsf{msg}))$ and the registered client key $(\mathsf{G}, \mathsf{u} \cdot \mathsf{G})$. Combining the unforgeability of EQS (for messages that are not a multiple of a signed message) with a check by the verifier that the first message in the redeemed token signature is G, we can guarantee that the client can create only one valid token from each blinded response it gets from the issuer. In the second case (i.e., the Dodis-Yampolskiy PRF case), the client can directly efficiently prove that the message in its blinded request is of the form $\mu \cdot (\mathsf{G}, (\mathsf{msg} + \mathsf{u}) \cdot \mathsf{G}, \mathsf{msg} \cdot \mathsf{G})$.

The above two constructions are generic transformations from any EQS to ACT. Instantiating them yields multiple concrete efficient ACT constructions under various security assumptions. In particular, we obtain an ACT construction with security in the standard model based on the SXDH and q-DDHI assumption, using the Dodis-Yampolskiy PRF construction together with the following EQS construction. The EQS signature is a normal signature. The adaptation of the signature of a message $\vec{\mathsf{M}}$ to a message $\vec{\mathsf{M}}' = \rho\vec{\mathsf{M}}$ is a ZK proof of knowledge of a valid signature on $\vec{\mathsf{M}}$ and of a scalar ρ such that $\vec{\mathsf{M}}' = \rho\vec{\mathsf{M}}$. For the concrete efficient instantiation of this EQS construction we use the efficient Jutla-Roy structure-preserving signatures [JR17] together with Groth-Sahai zero-knowledge proofs [GS08, EG14]. This construction has a restriction that it can support only short messages of length $O(\log \lambda)$ because the Dodis-Yampolskiy function is an adaptively secure PRF only over polynomial-size domains. The message length restriction can be solved by hashing the message using a hash function modeled as a random oracle. Hashing the message this way makes Dodis-Yampolskiy adaptively secure because, instead of having to guess the message forged by the adversary, the reduction just needs to guess which random oracle query will be used for the forgery. However, if we are willing to use the random oracle model, our second generic transformation (based on the random-oracle-based PRF $\mathsf{PRF}(\mathsf{u}, \mathsf{msg}) = \mathsf{u} \cdot \mathsf{H}(\mathsf{msg})$ instead of the Dodis-Yampolskiy PRF) is more efficient.

Finally, we present an optimized ACT construction with security in the generic bilinear group model (GBGM) and random oracle model (ROM). Conceptually this construction can be viewed as an optimization of the instantiation of our ACT from EQS which relies on $\mathsf{u} \cdot \mathsf{H}(\mathsf{msg})$ as PRF and uses the EQS from Fuchsbauer et al. [FHS19] and Fiat-Shamir transforms of Sigma protocols as ZK proofs. We prove the resulting scheme directly in the ROM and GBGM.

Other Related Work. We remark that the EQS scheme from [FHS19] has been used in [HS21] to construct anonymous credentials that are also "tag-based" and "aggregatable". However, we do not know how to use these extra properties to construct anonymous counting tokens, because, like metadata, the tag has to be known by the issuer, which would break unlinkability.

1.2 Organization of the Paper

After recalling preliminaries in Section 2, we define formally the notion of ACT in Section 3. We then present our construction of anonymous counting tokens (ACT) from Oblivious PRF in Section 4 and our two generic transforms of ACT from equivalence-class signature schemes (EQS) in Section 5. Combined with the EQS schemes in the full version of the paper [BRS23], these two generic transforms yield our concrete constructions of ACT from EQS that do not rely on the generic bilinear group model (GBGM). In Section 8 we show that an optimization of the second transform from Section 5 can be instantiated very efficiently in the GBGM. We conclude with a generic transformation that enables ACTs with different rate limits in Section 9.

2 Preliminaries

We denote by λ the security parameter. PPT means probabilistic polynomial time. $\mathsf{negl}(\lambda)$ indicates a quantity negligible in the security parameter, that is, for any positive integer k and for any large enough λ, $\mathsf{negl}(\lambda) \leq 1/\lambda^k$.

2.1 Cyclic Groups, Bilinear Groups, and Associated Assumptions

Our constructions make use of cyclic groups and bilinear groups. We denote by G the generator of a cyclic group $\mathbb{G}$ of prime order p. We use additive notation. We denote by $\mathbb{G}^*$ the set $\mathbb{G} \setminus \{0\}$.

A bilinear group is a set of three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, all of order p with generators $(\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_T)$, so that there exists an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ (called a pairing) such that $e(\mathsf{G}_1, \mathsf{G}_2) = \mathsf{G}_T$. The target group $\mathbb{G}_T$ is also denoted additively and we use $\bullet$ to denote the pairing operation: $e(\mathsf{G}_1, \mathsf{G}_2) = \mathsf{G}_1 \bullet \mathsf{G}_2$.

The symmetric external Diffie-Hellman (SXDH) assumption in a bilinear group $(\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_T)$ states that the decisional Diffie-Hellman (DDH) assumption holds in G_1 and G_2. The DDH assumption in $\mathbb{G}$ states that PPT adversaries $\mathcal{A}$:

$$\left| \Pr[x, y \leftarrow \mathbb{Z}_p,\ \mathcal{A}(\mathsf{G},\ x\mathsf{G},\ y\mathsf{G},\ (xy) \cdot \mathsf{G}) = 1] - \Pr[x, y, z \leftarrow \mathbb{Z}_p,\ \mathcal{A}(\mathsf{G},\ x\mathsf{G},\ y\mathsf{G},\ z \cdot \mathsf{G}) = 1] \right| \leq \mathsf{negl}(\lambda).$$

The q-decisional Diffie-Hellman inversion (q-DDHI) assumption in the group $\mathbb{G}$ states that for any PPT adversaries $\mathcal{A}$:

$$\left| \Pr[x \leftarrow \mathbb{Z}_p,\ \mathcal{A}(\mathsf{G},\ x\mathsf{G}, \dots,\ x^q\mathsf{G},\ (1/x) \cdot \mathsf{G}) = 1] - \Pr[x, y \leftarrow \mathbb{Z}_p,\ \mathcal{A}(\mathsf{G},\ x\mathsf{G}, \dots,\ x^q\mathsf{G},\ y \cdot \mathsf{G}) = 1] \right| \leq \mathsf{negl}(\lambda).$$

2.2 Pseudorandom Function

A pseudorandom function $\mathsf{PRF} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a function such that

$$\left| \Pr\left[\mathsf{K} \leftarrow\!\!\$\ \mathcal{K},\ \mathcal{A}^{\mathsf{PRF}(\mathsf{K},\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{O}(\cdot)}(1^\lambda) = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{O} : \mathcal{X} \to \mathcal{Y}$ is a random oracle.

We need to consider a stronger definition where $\mathcal{A}$ is also given some public information pk_K derived from $\mathsf{K} \leftarrow\!\!\$\ \mathcal{K}$, e.g., $\mathsf{pk}_\mathsf{K} = \mathsf{K} \cdot \mathsf{G}$ where G is a generator of a cyclic group:

$$\left| \Pr\left[\mathsf{K} \leftarrow\!\!\$\ \mathcal{K},\ \mathcal{A}^{\mathsf{PRF}(\mathsf{K},\cdot)}(\mathsf{pk}_\mathsf{K}) = 1\right] - \Pr\left[\mathsf{K} \leftarrow\!\!\$\ \mathcal{K},\ \mathcal{A}^{\mathsf{O}(\cdot)}(\mathsf{pk}_\mathsf{K}) = 1\right] \right| \leq \mathsf{negl}(\lambda) \tag{1}$$

Finally, we also consider a *selective* version where $\mathcal{A}$ must make all its query to its oracle PRF/O before receiving any answer and before seeing pk_K.

Dodis-Yampolskiy Pseudorandom Function. The Dodis-Yampolskiy function [BB04, DY05] is defined by $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{msg}) = \frac{1}{\mathsf{u}+\mathsf{msg}}\mathsf{G}$, with key $\mathsf{K} = \mathsf{u}$. We recall the following two lemmas that follow from the proof of weakly unforgeable signature scheme in Boneh-Boyen [BB04] and the pseudorandomness of the VRF in Dodis-Yampolskiy [DY05].[3]

Lemma 2.1. *If the q-DDHI assumption holds in group $\mathbb{G}$ with generator G, the function $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{msg}) = (\mathsf{u} + \mathsf{msg})^{-1} \cdot \mathsf{G}$ is a* selectively *pseudorandom function (when the adversary can make up to q queries), even when the adversaries sees $\mathsf{pk}_\mathsf{u} = \mathsf{u} \cdot \mathsf{G}$ after its selective queries.*

Using the same idea as in [DY05], we also get the following lemma:

Lemma 2.2. *If the 2^α-DDHI assumption holds in group $\mathbb{G}$ with generator G, the function $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{msg}) = (\mathsf{u} + \mathsf{msg})^{-1} \cdot \mathsf{G}$ is pseudorandom function when $\mathsf{msg} \in \{0,1\}^\alpha$, even when the adversaries sees $\mathsf{pk}_\mathsf{u} = \mathsf{u} \cdot \mathsf{G}$ (see Eq. (1)).*

In particular, the Dodis-Yampolskiy PRF is pseudorandom under a standard assumption for input message sizes that are logarithmic in the security parameter.

[3] Contrary to [BB04], we use a decisional assumption instead of the computational q-SDH because we want pseudorandomness and not unpredictability. Contrary to [DY05], we have the PRF value in $\mathbb{G}_1$ instead of $\mathbb{G}_T$ and our assumption is thus q-DDHI instead of q-DBDHI, and we do not need to have a bilinear map. Appendix A of Miao et al. [MPR+20] shows the proof under q-DDHI. The only difference with our case is that we allow the adversary to see $\mathsf{pk} = \mathsf{u} \cdot \mathsf{G}$, which can easily be simulated the same way as in [DY05]. Simulating $\mathsf{pk} = \mathsf{u} \cdot \mathsf{G}$ is why we rely on q-DDHI instead of just $(q-1)$-DDHI as would [MPR+20] require.

2.3 Camenisch-Shoup Encryption

The homomorphic encryption introduced by Camenisch and Shoup [CS03] is an additively homomorphic encryption. It additionally supports verifiable decryption, which enables a party holding the decryption key to prove the correctness of the decryption of a given ciphertext. Here, we define the encryption and decryption algorithms. We use the verifiable decryption proofs implicitly in our constructions. We use the additive notation for the CS algorithms except for decryption. That is, we write the multiplicative group $\mathbb{Z}^*_{N^2}$ additively, except in the description of the decryption algorithm. Later, in our constructions, we will refer only to the decryption algorithm by name and never use the multiplicative notation.

– $\mathsf{CS.Gen}(1^\lambda)$: Generate two ℓ-bit primes p' and q' such that $p = 2p' + 1$ and $q = 2q' + 1$ are primes and set $N = pq$. Choose random $\mathsf{R} \leftarrow\!\$\ \mathbb{Z}^*_{N^2}$ and set $\mathsf{G} = 2N\mathsf{R}$ be a $2N$-th residue.[4] Choose random $x \leftarrow\!\$\ \mathbb{Z}_{\lfloor N/4 \rfloor}$ and set $\mathsf{Y} = x\mathsf{G}$. Set $\mathsf{H} = 1 + N \mod N^2$, $\mathsf{PK} \leftarrow (N, \mathsf{G}, \mathsf{Y}, \mathsf{H})$ and $\mathsf{SK} \leftarrow x$. Remark that H is a generator of the subgroup of order N of $\mathbb{Z}^*_{N^2}$.
– $\mathsf{CS.Enc}(\mathsf{PK}, m \in \mathbb{Z}_n)$: Output $(r\mathsf{G}, m\mathsf{H} + r\mathsf{Y}) \in \mathbb{Z}^*_{N^2} \times \mathbb{Z}^*_{N^2}$ where $r \leftarrow\!\$\ \mathbb{Z}_{\lfloor N/4 \rfloor}$.
– $\mathsf{CS.Dec}(\mathsf{SK}, \mathsf{ct} = (u, e))$: Output $m = \frac{(\frac{e}{u^x} - 1) \mod N^2}{N}$ (in multiplicative notation).

2.4 Non-Interactive Zero-Knowledge Argument of Knowledge

A (non-interactive) zero-knowledge argument has the following algorithms:

– $\mathsf{crs} \leftarrow \mathsf{ZK.Setup}(\mathcal{R})$: generates public parameters (common random string) $\mathsf{ZK.crs}$ for the prove relationship $\mathcal{R}$. (We assume $\mathcal{R}$ implicitly defines the security parameter λ).
– $\pi \leftarrow \mathsf{ZK.Prove}(\mathcal{R}, \mathsf{crs}, \phi, w)$: generates a proof π that the prover knows a witness w such that the input statement ϕ satisfies the relation $\mathcal{R}(\phi, w)$.
– **false**/**true** $\leftarrow \mathsf{ZK.Verify}(\mathcal{R}, \mathsf{crs}, \phi, \pi)$: verifies the correctness of the proof for a statement ϕ.

Relation $\mathcal{R}$ and CRS $\mathsf{ZK.crs}$ are omitted when clear from the context (or not used). To simplify notation, we also often write:

$$\mathsf{ZK}\{\exists w : \phi\} \qquad \text{or} \qquad \mathsf{ZK}\{\reflectbox{K} w : \phi\}$$

instead of $\mathsf{ZK.Prove}(\mathcal{R}, \mathsf{crs}, \phi, w)$. "$\reflectbox{K}$" is used instead of "$\exists$" when the ZK argument is an argument of knowledge and satisfies computational knowledge soundness. We abuse notation and do not explicitly include as part of the witness random coins of algorithms in the statement. For example, we may write:

$$\mathsf{ZK}\{\exists x : c = \mathsf{Enc}(\mathsf{pk}, x), \mathsf{com} = \mathsf{Commit}(\mathsf{prm}, x)\}$$

without making explicit the randomness used by the encryption and commitment algorithms.

For some of the ZK arguments in this work, completeness holds for a smaller language than soundness. In that case, the notation above corresponds to the soundness language. The language for completeness is implicitly defined by the way the statement is constructed.

2.5 Zero Knowledge

ZK has the following properties.

Perfect Completeness. This property guarantees that an honestly generated proof for a true statement passes verification. For all relations $\mathcal{R}$ and $(\phi, w) \in \mathcal{R}$, for all $\mathsf{crs} \leftarrow \mathsf{ZK.Setup}(\mathcal{R})$ and honestly generated proofs $\pi \leftarrow \mathsf{ZK.Prove}(\mathcal{R}, \mathsf{crs}, \phi, w)$, the verification passes:

$$\mathbf{true} = \mathsf{ZK.Verify}(\mathcal{R}, \mathsf{crs}, \phi, \pi).$$

[4] Recall this is using additive notation for $\mathbb{Z}^*_{N^2}$. In usual multiplicative notation, this corresponds to: $\mathsf{G} = \mathsf{R}^{2N} \mod N^2$.

Computational Zero-knowledge. This property refers to the fact that the proof does not reveal any additional information about the witness apart from the correctness of the statement. For all relations $\mathcal{R}$ and all PPT adversaries $\mathcal{A}$, there exists a simulator Sim that can generate public parameters together with a trapdoor trap which can enable it to generate verifying proofs without knowing a witness.

$$\Big| \Pr\Big[\mathsf{crs} \leftarrow \mathsf{ZK.Setup}(\mathcal{R}),\ \mathcal{A}^{\mathsf{ZK.Prove}(\mathcal{R},\mathsf{crs},\cdot,\mathsf{cot})}(\mathsf{crs}) = 1\Big] \\ - \Pr\Big[(\mathsf{crs},\mathsf{trap}) \leftarrow \mathsf{Sim}(\mathcal{R}),\ \mathcal{A}^{\mathsf{Sim}'(\mathcal{R},\mathsf{trap},\cdot,\cdot)}(\mathsf{crs}) = 1\Big]\Big| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{Sim}'(\mathcal{R},\mathsf{trap},\phi,w)$ ignores its last argument w and returns $\mathsf{Sim}(\mathcal{R},\mathsf{trap},\phi)$.

Computational Soundness. This property captures the idea that no adversary can generate a proof on a false statement. For all relationships $\mathcal{R}$ and for every PPT adversary $\mathcal{A}$:

$$\Pr[\mathsf{crs} \leftarrow \mathsf{ZK.Setup}(\mathcal{R}),\ (\phi,\pi) \leftarrow \mathcal{A}(\mathsf{crs}) : \\ \mathbf{true} = \mathsf{ZK.Verify}(\mathcal{R},\mathsf{crs},\phi,\pi)\ \mathbf{and}\ \nexists w,\ (\phi,w) \in \mathcal{R}] \leq \mathsf{negl}(\lambda). \quad (2)$$

Computational Knowledge Soundness. This property is stronger than computational soundness. It says that for all relationships $\mathcal{R}$ and for every PPT adversary $\mathcal{A}$ there exists an extractor $\mathsf{Extract}$ such that for every valid proof that the adversary generates, the extractor can extract a valid witness (potentially by running the adversary multiple times) and the extractor-generated CRS is indistinguishable from an honestly generated CRS. The definition is adapted from [BPW12] removing the access to a simulation oracle, simplifying it by allowing the extractor to select the random coins of the adversary and allowing the use of a CRS (the latter is to allow constructions in the standard model). Concretely, we have:

$$\Pr\Big[(\mathsf{crs},\mathsf{trap}) \leftarrow \mathsf{Extract}(\mathcal{R}),\ \rho \leftarrow \mathsf{rnd}_{\mathcal{A}},\ (\phi,\pi) \leftarrow \mathcal{A}(\mathsf{crs};\rho), \\ w \leftarrow \mathsf{Extract}^{\mathcal{A}}(\mathsf{trap},\phi,\pi,\rho) : \\ \mathbf{true} = \mathsf{ZK.Verify}(\mathcal{R},\mathsf{crs},\phi,\pi)\ \mathbf{and}\ (\phi,w) \notin \mathcal{R}\Big] \leq \mathsf{negl}(\lambda)$$

(extractability) where $\mathsf{rnd}_{\mathcal{A}}$ is the distribution of random coins of the adversary, and for every arbitrary adversary $\mathcal{B}$ (even non-polynomial time):

$$\Big| \Pr[\mathsf{crs} \leftarrow \mathsf{ZK.Setup}(\mathcal{R}) : 1 = \mathcal{B}(\mathsf{crs})] \\ - \Pr[(\mathsf{crs},\mathsf{trap}) \leftarrow \mathsf{Extract}(\mathcal{R}) : 1 = \mathcal{B}(\mathsf{crs})]\Big| \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

(statistical setup indistinguishability).

Note that setup indistinguishability is only needed for constructions with a CRS and is trivial for construction like Fiat-Shamir. We require statistical setup indistinguishability because in our equivalence-class signature construction, setup indistinguishability is used between two non-polynomial-time games.

In the random oracle model, the extractor sees all random oracle queries made by $\mathcal{A}$ in the first step and controls the random oracle in the second step. Note that we do not require the existence of a straightline extractor, so we can use Fiat-Shamir arguments based on Sigma protocols in the random oracle model (without assuming algebraic adversaries or generic group model). However, this means that our security reductions can only extract one (or a logarithmic number of) adversarially-generated proofs at a time.

We also assume that ZK arguments for different relationships $\mathcal{R}$ use different random oracles. In practice, this just means prefixing hash queries for different relationships by different values (i.e., domain separation).

Construction of ZK Arguments. We now recall the two constructions we will be using in this paper: one from Sigma protocol and one based on Groth-Sahai proofs [GS08, EG14].

Sigma Protocols. A Sigma protocol [Cra97, CDS94] for a relationship $\mathcal{R}$ is a 3-move *public coin* protocol between a prover and a verifier both of which know an input ϕ and the prover knows w such that $\mathcal{R}(\phi, w)$. The prover sends the first message, which most often is some type of a commitment to a random value, the verifier provides a challenge as a second message and the third message is the prover's response computed based on the first two messages and the witness w.

Sigma protocols provide knowledge soundness as long as the prover can answer with verifying proofs more than one different challenge for the same first message. There exists an extractor that, given two transcripts with a common first message extracts a witness.

The Fiat-Shamir transform [FS87] can be applied to obtain a non-interactive version of any public coin protocol including the Sigma protocols. When analyzed in the random oracle (RO) model this transformation preserves the knowledge soundness property. The knowledge extractor needs to rewind the adversary and knowledge soundness is proven using the forking lemma [PS96, BPW12]. While interactive Sigma protocols are only proven to be zero-knowledge for honest verifiers, their non-interactive versions with Fiat-Shamir provide regular zero-knowledge.

Sigma protocols allow proving knowledge of committed or encrypted values as well as some classes of relations between those values. We will use several instances of Sigma protocols in our constructions.

Groth-Sahai Proofs. Groth-Sahai proofs are (non-interactive) ZK arguments invented by Groth and Sahai in [GS08] and fine-tuned by Escala and Groth in [EG14]. They allow to efficiently prove any set of pairing equations over a bilinear group. Contrary to Fiat-Shamir proofs, they are in the standard model with a common reference string. In addition, they are arguments of knowledge (with straightline extractor) when witnesses are restricted to be group elements in $\mathbb{G}_1$ or $\mathbb{G}_2$ (as opposed to scalars in $\mathbb{Z}_p$).

2.6 Signature Schemes

A signature scheme is defined as follows:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$ generates a public/secret key pair.
- $\rho \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$ outputs a signature ρ for message msg.
- **false**/**true** $\leftarrow \mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \rho)$: verifies the signature ρ.

Correctness ensures that for any key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$, for any message msg, for any signature $\rho \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$, the signature passes the verification: $\mathbf{true} = \mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \rho)$.

A signature scheme is said existentially unforgeable under chosen-message attacks (EUF-CMA) if for any PPT adversary $\mathcal{A}$:

$$\Big| \Pr[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda),\ (\mathsf{msg}, \rho) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{pk}) : \\ \mathbf{true} = \mathsf{Verify}(\mathsf{pk}, \rho, \mathsf{msg}) \textbf{ and } \mathsf{msg} \notin Q_{\mathsf{Sign}}] \Big| \leq \mathsf{negl}(\lambda)$$

where Q_{Sign} is the set of all messages queried to the oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$.

2.7 Commitment Schemes

We define commitment schemes as a pair of two algorithms $\mathsf{COM} = (\mathsf{COM.Setup}, \mathsf{Commit})$ where $\mathsf{Setup}(1^\lambda)$ outputs (public) commitment parameters prm, and $\mathsf{Commit}(\mathsf{prm}, \mathsf{msg}; t)$ returns a commitment com of message msg using randomness t. Public parameters are often omitted when clear from the context.

Pedersen Commitments. We will use the Pedersen commitment scheme when we need binding and hiding properties. We use Pedersen commitment over a group $\mathbb{G}$ where the Strong RSA assumption [BP97, FO97] holds. This will be needed since, in some cases, the committed values come from groups of different orders. The parameters for the commitment are group generators $\mathsf{G}, \mathsf{H} \in \mathbb{Z}_N^*$, where N is a Strong RSA modulus. As for Camenisch-Shoup (Section 2.3), we use the additive notation for $\mathbb{Z}_N^*$ instead of the multiplication notation. The commitment of a value x is of the form $\mathsf{Commit}(x; r) = x\mathsf{G} + r\mathsf{H}$ where $r \leftarrow\!\!\$\ \mathbb{Z}_{\lfloor N/4 \rfloor}$. The binding property of the commitment scheme requires that the prover does not know the discrete log relation between the generators G and H.

Extractable Commitments. These are commitments that have an extractable mode in which the commitment parameters are generated together with a trapdoor trap. There exists an extractor $\mathcal{E}$ which can extract the committed value $m \leftarrow \mathcal{E}(\mathsf{trap}, \mathsf{Commit}(m))$ using the trapdoor trap. We will use the Camenisch-Shoup encryption as an extractable commitment where in the normal mode, the secret key (i.e. the discrete log of Y) is not known, while in the trapdoor mode, the secret key is the trapdoor.

2.8 Equivalence-Class Signature Schemes (EQS)

Equivalence class signatures (EQS) [FHS19] are signatures for equivalence classes where a signature for a representative of the equivalence class can be transformed into a signature for any other representative in the same class using only public parameters. The EQS schemes that we use allow signing messages that are vectors of group elements $\vec{\mathsf{M}} \in \mathbb{G}_1^{*\ell}$ and provide the following signature adaptation property: a signature for $\vec{\mathsf{M}}$ can be adapted into signatures of any multiple $\mu\vec{\mathsf{M}}$, for $\mu \in \mathbb{Z}_p^*$. We exclude the 0 element for all coordinates of $\vec{\mathsf{M}}$ as well as for μ to match [FHS19].

As in [FG18], we use a slightly weaker definition than the original EQS notion: we allow the adapted signatures to be of a different format than the original signatures. The original signatures are called pre-signatures. We also only require computational signature adaptation instead of perfect signature adaptation: an adversary cannot computationally distinguish an (adapted) signature on the same message computed from two different pre-signatures, even if the adversary generated the secret key. We also allow for a common reference string (that is generated by a trusted party).

EQS. An equivalence class signature scheme consists of the following algorithms:

– $\mathsf{crs} \leftarrow \mathsf{EQS.Setup}(\mathcal{PG})$: on input a bilinear group $\mathcal{PG}$, generate a CRS crs.
– $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{EQS.KGen}(\mathsf{crs})$: on input a CRS crs generates secret and public keys which define pre-signature space $\mathcal{R}$ and signature space $\mathcal{S}$.
– $\rho \leftarrow \mathsf{EQS.Sign}(\mathsf{crs}, \mathsf{sk}, \vec{\mathsf{M}} \in \mathbb{G}_1^{*\ell})$: generates a pre-signature ρ for the representative $\vec{\mathsf{M}} = \vec{m}G_1 \in \mathbb{G}_1^{*\ell}$ of the class $\mathsf{Span}(\vec{\mathsf{M}}) = \mathsf{Span}(\vec{m}) \cdot G_1$.
– $\sigma \leftarrow \mathsf{EQS.Adapt}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}} \in \mathbb{G}_1^{*\ell}, \rho \in \mathcal{R}, \mu \in \mathbb{Z}_p^*)$: transforms a pre-signature ρ for a representative $\vec{\mathsf{M}}$ into a signature for $\vec{\mathsf{M}}' = \mu \cdot \vec{\mathsf{M}}$.
– **false**/**true** $\leftarrow \mathsf{EQS.Verify}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}' \in \mathbb{G}_1^{*\ell}, \sigma \in \mathcal{S})$: verifies signature σ for representative $\vec{\mathsf{M}}'$ using the public key pk.

When clear from the context, crs is omitted. Compared with [FG18], $\mathsf{EQS.Adapt}$ also takes as input $\vec{\mathsf{M}}$ (wlog since $\vec{\mathsf{M}}$ could also be included in ρ).

Perfect Correctness. An EQS is correct if, for any honestly generated pre-signature, any resulting adapted signature verifies. That is, for any $\vec{\mathsf{M}} \in \mathbb{G}_1^{*\ell}$ and $\mu \in \mathbb{Z}_p^*$:

$$\mathsf{crs} \leftarrow \mathsf{EQS.Setup}(\mathcal{PG}), \qquad (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{EQS.KGen}(\mathsf{crs}),$$
$$\rho \leftarrow \mathsf{EQS.Sign}(\mathsf{crs}, \mathsf{sk}, \vec{\mathsf{M}}), \qquad \sigma \leftarrow \mathsf{EQS.Adapt}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}, \rho, \mu)$$

we have:

$$\mathbf{true} = \mathsf{EQS.Verify}(\mathsf{crs}, \mathsf{pk}, \mu \cdot \vec{\mathsf{M}}, \sigma).$$

```
Game EUF-CMA$_{\mathcal{A}}(\lambda)$
$\mathcal{PG} \leftarrow \mathsf{GGen}(\lambda)$
$(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{EQS.Setup}(\mathcal{PG})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{EQS.KGen}(\mathsf{crs})$
$\mathcal{Q}_{\mathsf{Sign}} := \emptyset$
$(\vec{\mathsf{M}}'^{*} \in \mathbb{G}_1^{*\ell}, \sigma^* \in \mathcal{S}) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot)}(\mathsf{crs}, \mathsf{pk})$
return true $= \mathsf{Verify}(\mathsf{pk}, \vec{\mathsf{M}}'^{*}, \sigma^*)$ and
  $\forall \vec{\mathsf{M}} \in \mathcal{Q}_{\mathsf{Sign}}, \vec{\mathsf{M}}'^{*} \notin \mathsf{Span}(\vec{\mathsf{M}})$

Oracle $\mathsf{Sign}(\vec{\mathsf{M}} \in \mathbb{G}_1^{*\ell})$
$\mathcal{Q}_{\mathsf{Sign}} := \mathcal{Q}_{\mathsf{Sign}} \cup \{\vec{\mathsf{M}}\}$
$\rho \leftarrow \mathsf{EQS.Sign}(\mathsf{crs}, \mathsf{sk}, \vec{\mathsf{M}})$
return $\rho$
```

```
Game SIG-ADP$_{\mathcal{A}}(\lambda)$
$\mathcal{PG} \leftarrow \mathsf{GGen}(\lambda)$
$\mathsf{crs} \leftarrow \mathsf{EQS.Setup}(\mathcal{PG})$
$(\mathsf{pk}, \vec{\mathsf{M}}, \rho, \mu, \rho', \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{crs})$
$\mathsf{b_{chl}} \leftarrow_{\$} \{0,1\}$
$\sigma_0 \leftarrow \mathsf{EQS.Adapt}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}, \rho, \mu)$
$\sigma_1 \leftarrow \mathsf{EQS.Adapt}(\mathsf{crs}, \mathsf{pk}, \mu\vec{\mathsf{M}}, \rho', 1)$
abort if $\sigma_0 = \perp$ or $\sigma_1 = \perp$
$\mathsf{b_{guess}} \leftarrow \mathcal{A}(\mathsf{state}, \sigma_{\mathsf{b_{chl}}})$
return $(\mathsf{b_{chl}} == \mathsf{b_{guess}})$
```

Fig. 1: EUF-CMA and signature adaptation security game for EQS

Existential Unforgeability. We recall the notion of existential unforgeability under chosen-message attacks from [FHS19].

Definition 2.3. *An EQS scheme* $\mathsf{EQS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Adapt}, \mathsf{Verify})$ *satisfies existential unforgeability under chosen-message attacks (EUF-CMA) if for all PPT adversaries* $\mathcal{A}$*:*

$$\mathsf{Adv}^{euf\text{-}cma}_{\mathsf{EQS},\mathcal{A}}(\lambda) := \Pr[\text{EUF-CMA}_{\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda),$$

where EUF-CMA$_{\mathcal{A}}(\lambda)$ *is defined in Fig. 1.*

Fuchsbauer and Gay introduced a weaker EUF-CoMA notion in [FG18]. This notion requires the adversary in the security game to provide the discrete logarithms of all group elements. In our first construction of ACT from EQS (Construction 5.1), we could use this weak EUF-CoMA definition if we add a ZK proof of knowledge of the discrete logarithms of the message elements. However, such proof is very expensive (unless using Fiat-Shamir in the generic group model or the algebraic group model, but such proofs are much harder since extraction in the GGM or the AGM requires careful consideration of how the proof of the full scheme works).

Signature Adaptation. An EQS satisfies signature adaptation if a malicious signer cannot distinguish between two signatures on the same message $\vec{\mathsf{M}}' \in \mathbb{G}_1^{*\ell}$ adapted from two pre-signatures on two potentially different messages. Contrary to [FHS19], we allow signature adaptation to hold only computationally. We also implicitly assume that $\mathsf{EQS.Adapt}$ fails if the pre-signature ρ is invalid, which is why we don't have a verification algorithm for ρ. More formally, we define signature adaptation as follows.

Definition 2.4. *An EQS scheme* $\mathsf{EQS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Adapt}, \mathsf{Verify})$ *satisfies signature adaptation if for all PPT adversaries* $\mathcal{A}$*:*

$$\mathsf{Adv}^{sig\text{-}adp}_{\mathsf{EQS},\mathcal{A}}(\lambda) := |\Pr[\text{SIG-ADP}_{\mathcal{A}}(\lambda) = 1] - 1/2| = \mathsf{negl}(\lambda),$$

where the game SIG-ADP$_{\mathcal{A}}(\lambda)$ *is defined in Fig. 1.*

3 Definitions

In this section, we define *anonymous counting tokens* (ACTs) with public (respectively private) key verifiability. The private key verifiability version includes the grey-background text, while the public verifiability version does not.

Definition 3.1 (ACT). *An anonymous counting token (ACT) scheme with private key verifiability consists of the following algorithms:*

- $(\mathsf{pprm_S}, \mathsf{privprm_S}) \leftarrow \mathsf{ACT.GenParam}(1^\lambda)$*: generates parameters for the ACT scheme. These are parameters that will be reused throughout the execution of token issuance. Outputs private parameters* $\mathsf{privprm_S}$ *for the token issuer and public parameters* $\mathsf{pprm_S}$ *for the ACT scheme.*
- $(\mathsf{pprm_C}, \mathsf{privprm_C}) \leftarrow \mathsf{ACT.ClientRegister}(\mathsf{pprm_S})$*: on input the public parameters for the ACT scheme, this algorithm generates private parameters* $\mathsf{privprm_C}$ *for the client and public parameters* $\mathsf{pprm_C}$*.*
- $(\mathsf{blindRequest}, \mathsf{rand_{msg}}) \leftarrow \mathsf{ACT.Request}(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{msg})$*: on input the public parameters* $\mathsf{pprm_S}$ *for the ACT scheme, the private parameters for a client* $\mathsf{pprm_C}$ *and a message* msg*, generate a blinded token issuance request* $\mathsf{blindRequest}$ *and state information* $\mathsf{rand_{msg}}$*.*
- $(\mathsf{blindToken}, \boxed{\mathit{tag}}\ ) \leftarrow \mathsf{ACT.Sign}(\mathsf{privprm_S}, \mathsf{pprm_C}, \mathsf{blindRequest})$ *on input the private parameters for the issuer server* $\mathsf{privprm_S}$*, the public parameters for the client* $\mathsf{pprm_C}$ *and the blinded request* $\mathsf{blindRequest}$*, generate a blinded token. There is an optional output* tag *which the issuer can use for throttling one token per message per client.*
- $(\mathsf{msg}, \mathsf{tok}) \leftarrow \mathsf{ACT.Unblind}(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{blindToken}, \mathsf{rand_{msg}})$*: on inputs the public parameters* $\mathsf{pprm_S}$ *for the ACT scheme and the private parameters for a client* $\mathsf{pprm_C}$ *and a blind token* $\mathsf{blindToken}$ *and randomness* $\mathsf{rand_{msg}}$ *used to blind the request for the message, generate the unblinded token* tok *for message* msg)*.*
- $(\mathsf{bit}, \boxed{\mathit{tag}}\ ) \leftarrow \mathsf{ACT.Verify}(\mathsf{vrfyprm}, \mathsf{msg}, \mathsf{tok})$*: on input the verification parameters for the ACT scheme* $\mathsf{vrfyprm} := (\mathsf{pprm_S}, \mathsf{privprm_S})$*, which consist of the public parameters* $\mathsf{pprm_S}$ *for the ACT scheme, the private parameter for the issuer server* $\mathsf{privprm_S}$*, a message* msg *and a token* tok*, output verification bit* bit*. There is an optional output* tag *which the issuer can use for throttling one token per message per client.*

Figure 2 presents the interactions between a client and an issuer server during token issuance and verification using the algorithms of the ACT scheme. The client has the public parameters of the scheme and the server has the public keys $\mathcal{C}$ registered by clients as well as a set of tags $\mathcal{T}$ which it uses to throttle issuance at a single token per message per client. In order for the server to be able to enforce that each client gets at most one token per message, the server will obtain a tag that allows it to detect when the same client tries to obtain more than one token per message. This tag will be related to the message and the client's registered key but will only reveal whether more than one token per message is obtained/used by the same client. An ACT construction may enforce the throttling property either at issuance (i.e., the client cannot obtain a second token for the same message) or during verification where a client cannot redeem more than one token for the same message. For each of our constructions, we will specify which of the two functionalities it provides.

ACT Correctness. An ACT scheme is correct if any honestly generated token verifies. That is for any sets of issuer's and client's parameters

$$\begin{aligned}(\mathsf{pprm_S}, \mathsf{privprm_S}) &\leftarrow \mathsf{ACT.GenParam}(\lambda),\\ (\mathsf{pprm_C}, \mathsf{privprm_C}) &\leftarrow \mathsf{ACT.ClientRegister}(\mathsf{pprm_S}),\end{aligned}$$

and any message msg, the following holds

$$\begin{aligned}&(\mathsf{blindRequest}, \mathsf{rand_{msg}}) \leftarrow \mathsf{ACT.Request}(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{msg})\\ &\mathsf{blindToken} \leftarrow \mathsf{ACT.Sign}(\mathsf{privprm_S}, \mathsf{pprm_C}, \mathsf{blindRequest})\\ &(\mathsf{msg}, \mathsf{tok}) \leftarrow \mathsf{ACT.Unblind}(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{blindToken}, \mathsf{rand_{msg}})\\ &\mathbf{true} \leftarrow \mathsf{ACT.Verify}(\mathsf{vrfyprm}, \mathsf{msg}, \mathsf{tok}).\end{aligned}$$

$\mathsf{Client}(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{msg})$ $\qquad\qquad$ $\mathsf{Issuer}(\mathsf{privprm_S}, \mathcal{C}, \mathcal{T})$

$(\mathsf{bR}, \mathsf{r}) \leftarrow \mathsf{ACT.Request}(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{msg})$

$\mathsf{state} \leftarrow \mathsf{state} \cup (\mathsf{msg}, \mathsf{r})$

$\xrightarrow{\mathsf{bR}}$

$(\mathsf{bT}, \boxed{\mathsf{tag}})$

$\quad \leftarrow \mathsf{ACT.Sign}(\mathsf{privprm_S}, \mathsf{pprm_C}, \mathsf{bR})$

$\textbf{if } \exists\ \mathsf{tag},$

$\quad \textbf{if } \mathsf{tag} \in \mathcal{T},\ \textbf{abort}, \textbf{else } \mathcal{T} = \mathcal{T} \cup \mathsf{tag}$

$\xleftarrow{\mathsf{bT}}$

$(\mathsf{msg}, \mathsf{t}) \leftarrow \mathsf{ACT.Unblind}(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{bT}, \mathsf{r})$

$\xrightarrow{\mathsf{msg}, \mathsf{t}}$

$(\mathsf{bit}, \boxed{\mathsf{tag}}) \leftarrow \mathsf{ACT.Verify}(\mathsf{vrfyprm}, \mathsf{msg}, \mathsf{t})$

$\textbf{if } \exists\ \mathsf{tag},$

$\quad \textbf{if } \mathsf{tag} \in \mathcal{T}, ; \textbf{abort}, \textbf{else } \mathcal{T} = \mathcal{T} \cup \mathsf{tag}$

$\textbf{if } \mathsf{bit} = \textbf{false}, \textbf{abort}$

Fig. 2: Token issuance and verification for ACT (Definition 3.1).

3.1 Security Properties

Unforgeability. The first security property is unforgeability, which guarantees that an adversary cannot generate tokens for more messages than the ones it has requested signatures and it also cannot generate more than one signature for a message per registered client key. This holds even when the adversary can register public parameters for many clients.

Definition 3.2 (Unforgeability). *An anonymous counting token scheme* ACT *is* unforgeable *if for any* PPT *adversary $\mathcal{A}$ and any* $\max(\mathsf{T}) \geq 0$, $\max(\mathsf{R}) \geq 0$ *(the maximum number of queries):*

$$\mathsf{Adv}^{\mathsf{omuf}}_{\mathsf{ACT},\mathcal{A}}(\lambda) := \Pr\big[\mathrm{OMUF}_{\mathsf{ACT},\mathcal{A}}(\lambda) = 1\big] = \mathsf{negl}(\lambda).$$

where $\mathrm{OMUF}_{\mathsf{ACT},\mathcal{A}}(\lambda)$ *is defined in Figure 3.*

Unlinkability. The next ACT property is *unlinkability* which guarantees that even the issuer cannot link client token requests with redeemed tokens, except if it can trivially do so. Definition relies on $\mathrm{UNLINK}_{\mathsf{ACT},\mathcal{A}}(\lambda)$ is defined in Figure 4.

The high-level idea of the game is the following. The adversary plays the role of the issuer, can register as many clients as it wants in via the GetPrm oracle, and can ask those clients to generate blind token requests for messages of its choice via the Request oracle. It needs to be distinguish blind token requests bT for two different client/message pairs (oracle $\mathsf{Chl_{issue}}$); or it needs to distinguish redeemed/unblind tokens for the same message but two different issuance sessions (oracle $\mathsf{Chl_{redeem}}$). As the adversary can always provide wrong blindToken (as issuer), in that latter, we request that ACT.Unblind succeeds on both the blind tokens provided by the adversary.

Our unlinkability notion assumes that the issuer parameters $\mathsf{pprm_S}$ and $\mathsf{privprm_S}$ are honestly generated. We informally discuss how to remove this requirement in each of our constructions.

Definition 3.3 (Unlinkability). *An anonymous token scheme* ACT *is* unlinkable *if for any* PPT *adversary $\mathcal{A}$:*

$$\mathsf{Adv}^{\mathrm{unlink}}_{\mathsf{ACT},\mathcal{A}}(\lambda) := |2\Pr\big[\mathrm{UNLINK}_{\mathsf{ACT},\mathcal{A}}(\lambda) = 1\big] - 1| = \mathsf{negl}(\lambda),$$

where $\mathrm{UNLINK}_{\mathsf{ACT},\mathcal{A}}(\lambda)$ *is defined in Figure 4.*

```
Game $\mathrm{OMUF}_{\mathsf{ACT},\mathcal{A}}(\lambda)$
$(\mathsf{pprm}_\mathsf{S}, \mathsf{privprm}_\mathsf{S}) \leftarrow \mathsf{ACT.GenParam}(1^\lambda)$
$\mathsf{R} \leftarrow 0,\ \mathsf{T} \leftarrow 0,\ \mathsf{state} \leftarrow \{\},\ \mathsf{tags} \leftarrow \{\}$
$\mathsf{output} \leftarrow \mathcal{A}^{\mathsf{Register}(\cdot),\mathsf{Sign}(\cdot),\mathsf{Verify}(\cdot)}(\mathsf{pprm}_\mathsf{S})$
$(\mathsf{msg}_i, \mathsf{tok}_i)_{i\in[\mathsf{T}+1]} := \mathsf{output}$
return
    $\forall\ i \neq j \in [\mathsf{T}+1], (\mathsf{msg}_i, \mathsf{tok}_i) \neq (\mathsf{msg}_j, \mathsf{tok}_j)$
      and
    $\Big($      // Type 1 forgery:
$\Big($ $\forall i \in [\mathsf{T}+1]$,
    $\mathsf{ACT.Verify}(\mathsf{vrfyprm}, \mathsf{msg}_i, \mathsf{tok}_i) = (\mathbf{true}, \boxed{\mathsf{tag}_i})$,
    $\boxed{\forall\ i \neq j \in [\mathsf{T}+1], (\mathsf{msg}_i, \mathsf{tag}_i) \neq (\mathsf{msg}_j, \mathsf{tag}_j)}\Big)$
          or
        // Type 2 forgery:
$\Big($ $\exists\ S \subset [\mathsf{T}], \exists \mathsf{msg} :\ |S| > \mathsf{R}$ and
    $\forall i \in S, \mathsf{msg}_i = \mathsf{msg}$ and
    $\forall i \in S, \mathsf{ACT.Verify}(\mathsf{vrfyprm}, \mathsf{msg}_i, \mathsf{tok}_i) = \mathbf{true}\Big)$
    $\Big)$

Oracle $\mathsf{Register}(\mathsf{pprm}_\mathsf{C})$
$\mathsf{state} \leftarrow \mathsf{state} \cup \mathsf{pprm}_\mathsf{C}$
$\mathsf{R} \leftarrow \mathsf{R} + 1$

Oracle $\mathsf{Sign}(\mathsf{pprm}_\mathsf{C}, \mathsf{bR})$
if $\mathsf{pprm}_\mathsf{C} \notin \mathsf{state}$, return $\perp$
$\mathsf{T} = \mathsf{T} + 1$
$(\mathsf{bT}, \boxed{\mathsf{tag}})$
    $\leftarrow \mathsf{ACT.Sign}(\mathsf{privprm}_\mathsf{S}, \mathsf{pprm}_\mathsf{C}, \mathsf{bR})$
$\boxed{\text{if } \mathsf{tag} \in \mathsf{tags}, \text{return } \perp}$
$\boxed{\mathsf{tags} \leftarrow \mathsf{tags} \cup \mathsf{tag}}$
return $\mathsf{bT}$

Oracle $\mathsf{Verify}(\mathsf{msg}, \mathsf{tok})$
$\mathsf{bit} \leftarrow \mathsf{ACT.Verify}(\mathsf{vrfyprm}, \mathsf{msg}, \mathsf{tok})$
return $\mathsf{bit}$
```

Fig. 3: Unforgeability game for an ACT scheme. The appropriate $\boxed{\text{boxed}}$ instructions are included depending on whether rate limiting is done at issuance (boxed instructions in Sign) or at redemption (boxed instruction in main game).

4 Anonymous Counting Tokens from Oblivious PRF

In this section, we present our first anonymous counting tokens construction which leverages oblivious pseudorandom functions. We make use of the extended Boneh-Boyen PRF function $\mathcal{F}(\mathsf{sk} = (\mathsf{u}, \mathsf{y}), \mathsf{msg}, \mathsf{r}) = (\mathsf{msg} + \mathsf{u} + \mathsf{r} \cdot \mathsf{y})^{-1} \cdot \mathsf{G}$ where G is a generator of a group $\mathbb{G}$, which was used by Boneh and Boyen [BB04] to construct short signatures without oracles. The Dodis-Yampolskiy function [DY05] $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{msg}) = (\mathsf{msg} + \mathsf{u})^{-1} \cdot \mathsf{G}$ can be viewed as a special case of this function where the key y is set to zero. For our construction and proofs, we need the property that $\mathcal{F}$ is pseudorandom when evaluated on adversarially chosen messages and on randomness that is sampled uniformly at random.

4.1 Extended Boneh-Boyen Pseudorandom Function

In this section, we prove the pseudorandom properties we will use for the Boneh-Boyen PRF function $\mathcal{F}(\mathsf{sk} = (\mathsf{u}, \mathsf{y}), \mathsf{msg}, \mathsf{r}) = (\mathsf{msg} + \mathsf{u} + \mathsf{r} \cdot \mathsf{y})^{-1} \cdot \mathsf{G}$. Concretely, we prove the following lemma:

Lemma 4.1. *If $\mathcal{F}_{\mathsf{DY}}(\mathsf{msg}, \mathsf{u}) = (\mathsf{msg} + \mathsf{u})^{-1} \cdot \mathsf{G}$ is a selectively pseudorandom function over the group $\mathbb{G}$ with generator G, the function $\mathcal{F}(\mathsf{sk} = (\mathsf{u}, \mathsf{y}), \mathsf{msg}, \mathsf{r}) = (\mathsf{u} + \mathsf{msg} + \mathsf{r} \cdot \mathsf{y})^{-1} \cdot \mathsf{G}$ is an extended pseudorandom function, that is:*

$$\mathsf{Adv}^{eprf}_{\mathsf{PRF},\mathcal{A}}(\lambda) := \Pr[\mathrm{EPRF}_{\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda),$$

where $\mathrm{EPRF}_{\mathcal{A}}(\lambda)$ is defined in Fig. 5.

Recall that Lemma 2.1 shows that $\mathcal{F}_{\mathsf{DY}}$ satisfies the premise.

```
Game $\mathrm{UNLINK}_{\mathsf{ACT},\mathcal{A}}(\lambda)$
$(\mathsf{pprm_S}, \mathsf{privprm_S}) \leftarrow \mathsf{ACT.GenParam}(1^\lambda)$
$\mathsf{b_{chl}} \leftarrow\!\!\$\ \{0,1\}$
$\mathsf{b_{guess}} \leftarrow \mathcal{A}^{O}(\mathsf{pprm_S}, \mathsf{privprm_S})$
return $(\mathsf{b_{chl}} == \mathsf{b_{guess}})$

$\mathsf{GetPrm}()$
$(\mathsf{pprm_C}, \mathsf{privprm_C})$
    $\leftarrow \mathsf{ACT.ClientRegister}(\mathsf{pprm_S})$
return $\mathsf{pprm_C}$

$\mathsf{Request}(\mathsf{pprm_C}, \mathsf{msg})$
abort if $\mathsf{pprm}_C$ not from $\mathsf{GetPrm}$
abort if $(\mathsf{pprm_C}, \mathsf{msg}) \in \mathcal{Q}'$
$(\mathsf{bR}, \mathsf{r})$
    $\leftarrow \mathsf{ACT.Request}(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{msg})$
$\mathcal{Q} = \mathcal{Q} \cup (\mathsf{pprm_C}, \mathsf{msg}, \mathsf{bR}, \mathsf{r})$
$\mathcal{Q}' = \mathcal{Q}' \cup (\mathsf{pprm_C}, \mathsf{msg})$
return $\mathsf{bR}$

$\mathsf{Chl_{issue}}(\mathsf{pprm_C}, \mathsf{msg}_0, \mathsf{msg}_1)$
abort if $\mathsf{pprm_C}$ not from $\mathsf{GetPrm}$
abort if $(\mathsf{pprm_C}, \mathsf{msg}_0)$ or $(\mathsf{pprm_C}, \mathsf{msg}_1) \in \mathcal{Q}'$
abort if $\mathsf{msg}_0 = \mathsf{msg}_1$
$(\mathsf{bR}, \mathsf{r}) \leftarrow \mathsf{ACT.Request}(\mathsf{privprm_C}, \mathsf{msg_{b_{chl}}})$
$\mathcal{Q}' = \mathcal{Q}' \cup (\mathsf{pprm_C}, \mathsf{msg}_0) \cup (\mathsf{pprm_C}, \mathsf{msg}_1)$
return $\mathsf{bR}$

$\mathsf{Chl_{redeem}}(\mathsf{pprm}_0, \mathsf{pprm}_1, \mathsf{bR}_0, \mathsf{bR}_1, \mathsf{bT}_0, \mathsf{bT}_1)$
abort if $(\mathsf{pprm}_0, \star, \mathsf{bR}_0, \star) \notin \mathcal{Q}$
abort if $(\mathsf{pprm}_1, \star, \mathsf{bR}_1, \star) \notin \mathcal{Q}$
Find $(\mathsf{pprm}_b, \mathsf{msg}_b, \mathsf{bR}_b, \mathsf{r_b}) \in \mathcal{Q}$
    and delete them (for $b \in \{0,1\}$)
$(\mathsf{msg}'_0, \mathsf{tok}_0) \leftarrow$
    $\mathsf{ACT.Unblind}(\mathsf{pprm_S}, \mathsf{privprm}_0, \mathsf{bT}_0, \mathsf{r}_0)$
$(\mathsf{msg}'_1, \mathsf{tok}_1) \leftarrow$
    $\mathsf{ACT.Unblind}(\mathsf{pprm_S}, \mathsf{privprm}_1, \mathsf{bT}_1, \mathsf{r}_1)$
abort if $\mathsf{msg}_0 \neq \mathsf{msg}_1$
abort if $\mathsf{msg}'_0 \neq \mathsf{msg}_0$ or $\mathsf{msg}'_1 \neq \mathsf{msg}_1$
return $\mathsf{tok_{b_{chl}}}$
```

Fig. 4: Unlinkability game for an ACT scheme, where $\mathcal{A}$ has access to oracles $O = \mathsf{GetPrm}(\cdot), \mathsf{Request}(\cdot), \mathsf{Chl_{issue}}(\cdot), \mathsf{Chl_{redeem}}(\cdot)$

Proof. Assuming there exists an adversary $\mathcal{A}$ that distinguishes $\mathcal{F}$ from random with T evaluations and R challenge queries, we build and adversary $\mathcal{B}$ that distinguishes $\mathcal{F}_{\mathsf{DY}}$ with the same number of selective evaluation and challenge queries as follows. Here, we consider a variant of the selective version of Eq. (1), where $\mathcal{B}$ also is given access to an evaluation oracle $\mathsf{Eval}(\mathsf{msg})$ (instead of a single $\mathsf{Challenge}$ oracle that either matches $\mathsf{PRF} = \mathcal{F}$ or O in Eq. (1)). Usual hybrid techniques can reduce this variant to the selection version of Eq. (1).

$\mathcal{B}$ generates a random $\mathsf{y} \in \mathbb{Z}_p$ and provides $\mathsf{y} \cdot \mathsf{G}$ to $\mathcal{A}$. $\mathcal{B}$ makes T evaluation queries on random messages $(\mathsf{msg}_i)_{i\in[\mathsf{T}]}$ to obtains $(\mathsf{F}_i)_{i\in[\mathsf{T}]}$ and R challenge queries on random messages $(\mathsf{msg}'_j)_{j\in[\mathsf{R}]}$ to obtain $(\mathsf{E}_j)_{j\in[\mathsf{R}]}$. On the i-th evaluation query m_i from $\mathcal{A}$, $\mathcal{B}$ chooses $\mathsf{r}_i = \mathsf{y}^{-1} \cdot (\mathsf{msg}_i - m_i)$, and returns $(\mathsf{r}_i, \mathsf{F}_i)$. Similarly, on the j-th challenge query m'_j from $\mathcal{A}$, $\mathcal{B}$ chooses $\mathsf{r}'_j = \mathsf{y}^{-1} \cdot (\mathsf{msg}'_j - m'_j)$, and returns $(\mathsf{r}'_i, \mathsf{E}_i)$. $\mathcal{B}$ returns the same guess as the guess of $\mathcal{A}$.

The view of $\mathcal{A}$ is identical to the one in experiment $\mathrm{EPRF}_{\mathcal{A}}(\lambda)$ in Lemma 4.1 since all values r_i and r'_j are distributed uniformly at random since the messages msg_i, msg'_j are random. Therefore, the probability of success of $\mathcal{A}$ is bounded by the probability of the selective adversary against the Dodis-Yampolskiy pseudorandom function.

□

4.2 Verifiable Oblivious Pseudorandom Function

We will need to evaluate obliviously the function $\mathcal{F}(\mathsf{sk} = (\mathsf{u}, \mathsf{y}), \mathsf{msg}, \mathsf{r}) = (\mathsf{msg} + \mathsf{u} + \mathsf{r} \cdot \mathsf{y})^{-1} \cdot \mathsf{G}$: one party has the secret key sk while the other party has a message $(\mathsf{msg}, \mathsf{r})$ as input. We call the resulting protocol a verifiable oblivious pseudorandom function (VOPRF).

We do not prove separate properties for the VOPRF and we prove everything the ACT security properties. Informally, the security property that we will be proving is that given public parameters and committed input,

```
Game EPRF_$\mathcal{A}$($\lambda$)
    $(\mathsf{u}, \mathsf{y}) \leftarrow\!\!\$\ \mathbb{Z}_p^2$
    $\mathsf{sk} \leftarrow (\mathsf{u}, \mathsf{y})$
    $\mathsf{b_{chl}} \leftarrow\!\!\$\ \{0, 1\}$
    $\mathsf{b_{guess}} \leftarrow \mathcal{A}^{\mathsf{Eval}(\cdot),\mathsf{Challenge}(\cdot)}(\mathsf{u} \cdot \mathsf{G}, \mathsf{y} \cdot \mathsf{G})$
    return $(\mathsf{b_{chl}} == \mathsf{b_{guess}})$

Oracle Eval(msg)
    $\mathsf{r} \leftarrow\!\!\$\ \mathcal{R}$
    return $\mathcal{F}(\mathsf{sk}, \mathsf{msg}, \mathsf{r})$

Oracle Challenge(msg)
    $\mathsf{r} \leftarrow\!\!\$\ \mathcal{R}$
    $\mathsf{E}_0 \leftarrow \mathsf{F}(\mathsf{sk}, \mathsf{msg}, \mathsf{r})$
    $\mathsf{E}_1 \leftarrow\!\!\$\ \mathcal{V}$
    return $\mathsf{E}_{\mathsf{b_{chl}}}$
```

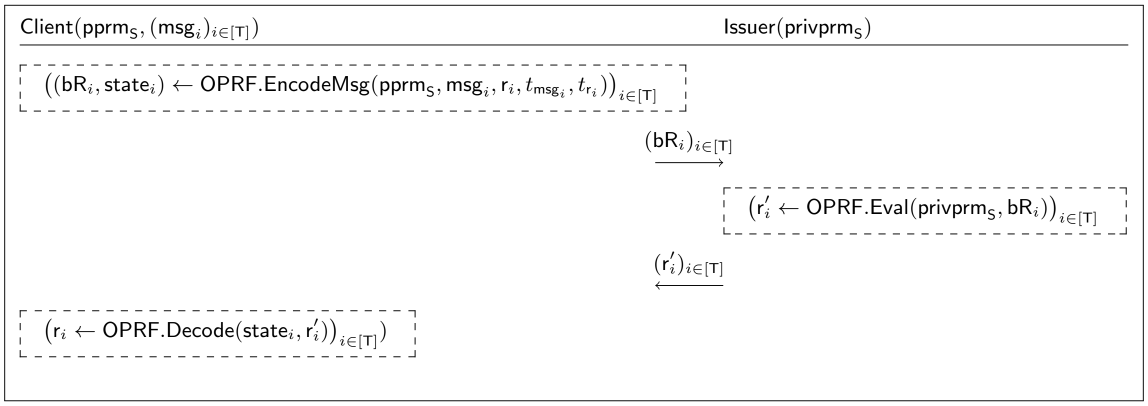
Fig. 5: Extended pseudorandom function

the protocol that consists of the steps: the client runs $\mathsf{EncodeMsg}$, the server runs Eval and the clients runs Decode to obtain its output, is a malicious secure computation protocol where the clients receives output $\mathcal{F}(\mathsf{msg}, \mathsf{r})$ and the server learns nothing.

The protocol is used as part of the final ACT protocol, where we implicitly assume inputs and keys to be previously committed.

Definition 4.2 (Verifiable Oblivious Pseudorandom Function). *A verifiable oblivious pseudorandom function (VOPRF) for the function $\mathcal{F}$ defined in Section 4.1 consists of the following algorithms* ($\mathsf{VOPRF.GenParam}$, $\mathsf{VOPRF.EncodeMsg}$, $\mathsf{VOPRF.Eval}$, $\mathsf{OPRF.Decode}$)*:*

- $(\mathsf{pprm}, \mathsf{privprm}) \leftarrow \mathsf{VOPRF.GenParam}(1^\lambda)$ *takes an input the security parameter and generates public and private parameters*
- $(\mathsf{digest}, \mathsf{state}) \leftarrow \mathsf{VOPRF.EncodeMsg}(\mathsf{pprm}, (\mathsf{msg}, \mathsf{r}), (t_{\mathsf{msg}}, t_{\mathsf{r}}))$ *takes as input a message, randomness r, as well as randomness t_{msg}, t_r used to commit* msg *and r (defined in this way for composition with other protocols), and outputs a digest and a state. The digest includes a proof of correct evaluation with respect to committed input and public parameters.*
- $\mathsf{blindPRF} \leftarrow \mathsf{VOPRF.Eval}(\mathsf{privprm}, \mathsf{digest})$ *takes a digest with a proof of correctness and PRF private parameters, and outputs a value* $\mathsf{blindPRF}$*. The value* $\mathsf{blindPRF}$ *includes a proof of correct evaluation.*
- $\tau \leftarrow \mathsf{VOPRF.Decode}(\mathsf{state}, \mathsf{blindPRF})$ *takes a value* $\mathsf{blindPRF}$*, and outputs the value* $\tau = \mathcal{F}(\mathsf{sk}, \mathsf{msg}, \mathsf{r})$ *(at least if everything was generated honestly).*



$\mathsf{Client}(\mathsf{pprm_S}, (\mathsf{msg}_i)_{i\in[\mathsf{T}]})$
$\mathsf{Issuer}(\mathsf{privprm_S})$
$\big((\mathsf{bR}_i, \mathsf{state}_i) \leftarrow \mathsf{OPRF.EncodeMsg}(\mathsf{pprm_S}, \mathsf{msg}_i, \mathsf{r}_i, t_{\mathsf{msg}_i}, t_{\mathsf{r}_i})\big)_{i\in[\mathsf{T}]}$
$(\mathsf{bR}_i)_{i\in[\mathsf{T}]}$
$\big(\mathsf{r}'_i \leftarrow \mathsf{OPRF.Eval}(\mathsf{privprm_S}, \mathsf{bR}_i)\big)_{i\in[\mathsf{T}]}$
$(\mathsf{r}'_i)_{i\in[\mathsf{T}]}$
$\big(\mathsf{r}_i \leftarrow \mathsf{OPRF.Decode}(\mathsf{state}_i, \mathsf{r}'_i)\big)_{i\in[\mathsf{T}]}\big)$

Fig. 6: Oblivious Evaluation of PRF.

4.3 ACT Construction

Assuming we have a VOPRF (with the right properties) for the extended Boneh-Boyen PRF function $\mathcal{F}(\mathsf{sk} = (\mathsf{u}, \mathsf{y}), \mathsf{msg}, \mathsf{r}) = (\mathsf{msg} + \mathsf{u} + \mathsf{r} \cdot \mathsf{y})^{-1} \cdot \mathsf{G}$, we construct an ACT from this VOPRF.

Construction 4.3 (ACT from VOPRF). Let $\mathbb{G}$ be a cyclic group of prime order p with generator G, $\mathsf{VOPRF} = (\mathsf{VOPRF.GenParam}, \mathsf{VOPRF.EncodeMsg}, \mathsf{VOPRF.Eval}, \mathsf{VOPRF.Decode})$ be the verifiable oblivious pseudorandom function defined in Construction 4.4, $\mathsf{COM}_{\mathsf{Ped}} = (\mathsf{COM}_{\mathsf{Ped}}.\mathsf{Setup}, \mathsf{Commit}_{\mathsf{Ped}})$ be the Pedersen commitment scheme over a strong-RSA group (a hiding and binding commitment scheme), $\mathsf{COM}_{\mathsf{Ext}} = (\mathsf{COM}_{\mathsf{Ext}}.\mathsf{Setup}, \mathsf{Commit}_{\mathsf{Ext}})$ be an extractable commitment scheme defined as the CS encryption scheme (see Section 2.7), $\mathcal{F}_{\mathsf{DY}}$ be the Dodis-Yampolskiy (selective) PRF over $\mathbb{G}$,[5] and ZK be a sound zero-knowledge argument scheme. We construct an anonymous counting token scheme ACT as follows:

ACT.GenParam(1^λ): Generate

1. $(\mathsf{PK}_{\mathsf{VOPRF}}, \mathsf{SK}_{\mathsf{VOPRF}}) \leftarrow \mathsf{VOPRF.GenParam}(1^\lambda)$. Note that $\mathsf{PK}_{\mathsf{VOPRF}}$ contains (public) parameters $\mathsf{prm}_{\mathsf{Ext}}$ for the extractable commitment scheme and $\mathsf{prm}_{\mathsf{Ped}}$ for the Perdersen hiding and binding commitment.

$$\textbf{Output:}\quad \begin{aligned} &\mathsf{pprm}_{\mathsf{S}} \leftarrow \mathsf{PK}_{\mathsf{VOPRF}} \\ &\mathsf{privprm}_{\mathsf{S}} \leftarrow \mathsf{SK}_{\mathsf{VOPRF}} \end{aligned}$$

ACT.ClientRegister$(\mathsf{pprm}_{\mathsf{S}})$: Generate

1. a Dodis-Yampolskiy PRF key $\mathsf{u}_{\mathsf{C}} \leftarrow\!\!\$\ \mathbb{Z}_p$,
2. a commitment $\mathsf{com}_{\mathsf{u}_{\mathsf{C}}} \leftarrow \mathsf{Commit}_{\mathsf{Ped}}(\mathsf{u}_{\mathsf{C}}; t_{\mathsf{u}_{\mathsf{C}}})$

$$\textbf{Output:}\quad \begin{aligned} &\mathsf{pprm}_{\mathsf{C}} \leftarrow \mathsf{com}_{\mathsf{u}_{\mathsf{C}}} \\ &\mathsf{privprm}_{\mathsf{C}} \leftarrow \mathsf{u}_{\mathsf{C}} \end{aligned}$$

ACT.TokenRequest$(\mathsf{pprm}_{\mathsf{S}}, \mathsf{privprm}_{\mathsf{C}}, \mathsf{msg})$:

1. Compute a commitment to the message $\mathsf{com}_{\mathsf{msg}} \leftarrow \mathsf{Commit}_{\mathsf{Ext}}(\mathsf{H}(\mathsf{msg}); r_{\mathsf{msg}})$.
2. Compute
   - Dodis-Yampolskiy PRF evaluation $\mathsf{v} \leftarrow \mathcal{F}_{\mathsf{DY}}(\mathsf{u}_{\mathsf{C}}, \mathsf{H}(\mathsf{msg}))$.
   - Proof of correct PRF evaluation

$$\pi_{\mathsf{v}} : \mathsf{ZK}\{\exists h, \mathsf{u}_{\mathsf{C}}, t_{\mathsf{msg}}, t_{\mathsf{u}_{\mathsf{C}}} : \mathsf{v} = \mathcal{F}_{\mathsf{DY}}(\mathsf{u}_{\mathsf{C}}, h), \\ \mathsf{com}_{\mathsf{msg}} = \mathsf{Commit}_{\mathsf{Ext}}(h; t_{\mathsf{msg}}),\ \mathsf{com}_{\mathsf{u}_{\mathsf{C}}} = \mathsf{Commit}_{\mathsf{Ped}}(\mathsf{u}_{\mathsf{C}}; t_{\mathsf{u}_{\mathsf{C}}})\}$$

3. Generate a random r_{C} and commitment $\mathsf{com}_{\mathsf{r}_{\mathsf{C}}} \leftarrow \mathsf{Commit}_{\mathsf{Ext}}(\mathsf{r}_{\mathsf{C}}; t_{\mathsf{r}_{\mathsf{C}}})$.
4. Hash the transcript to get random value $\mathsf{r}_{\mathsf{S}} \leftarrow \mathsf{H}(\mathsf{trnc})$ where

$$\mathsf{trnc} = (\mathsf{pprm}_{\mathsf{S}}, \mathsf{com}_{\mathsf{u}_{\mathsf{C}}}, \mathsf{com}_{\mathsf{msg}}, \mathsf{v}, \mathsf{com}_{\mathsf{r}_{\mathsf{C}}}).$$

5. Compute $\mathsf{r} \leftarrow \mathsf{r}_{\mathsf{C}} + \mathsf{r}_{\mathsf{S}}$, commit $\mathsf{com}_{\mathsf{r}} \leftarrow \mathsf{Commit}_{\mathsf{Ped}}(\mathsf{r}; t_{\mathsf{r}})$ and generate a proof:

$$\pi_{\mathsf{r}} : \mathsf{ZK}\{\exists\ \mathsf{r}_{\mathsf{C}}, t_{\mathsf{r}_{\mathsf{C}}}, t_{\mathsf{r}} :\ \mathsf{r} = \mathsf{r}_{\mathsf{C}} + \mathsf{r}_{\mathsf{S}}, \\ \mathsf{com}_{\mathsf{r}_{\mathsf{C}}} = \mathsf{Commit}_{\mathsf{Ext}}(\mathsf{r}_{\mathsf{C}}, t_{\mathsf{r}_{\mathsf{C}}}), \mathsf{com}_{\mathsf{r}} = \mathsf{Commit}_{\mathsf{Ped}}(\mathsf{r}; t_{\mathsf{r}})\}.$$

(The above 3 steps are used to create a random value r that neither the issuer nor the client control as explained in Section 1.1.)

[5] This PRF is used for the rate limitation of the client. VOPRF does not evaluate this PRF but rather evaluates $\mathcal{F}$ defined in Section 4.1.

6. Compute first OPRF message on input msg with randomness r

$$(\mathsf{VOPRF.state}, \mathsf{VOPRF.digest}) \leftarrow \mathsf{VOPRF.EncodeMsg}(\mathsf{PK}_{\mathsf{VOPRF}}, (\mathsf{H}(\mathsf{msg}), \mathsf{r}), (t_{\mathsf{msg}}, t_{\mathsf{r}})).$$

Output: $\mathsf{blindRequest} \leftarrow (\mathsf{com}_{\mathsf{msg}}, \mathsf{v}, \pi_{\mathsf{v}}, \mathsf{com}_{\mathsf{r_C}}, \mathsf{com}_{\mathsf{r}}, \pi_{\mathsf{r}}, \mathsf{VOPRF.digest})$
$\mathsf{rand}_{\mathsf{msg}} \leftarrow (\mathsf{msg}, \mathsf{r}, \mathsf{VOPRF.state})$

ACT.Sign$(\mathsf{privprm}_{\mathsf{S}}, \mathsf{pprm}_{\mathsf{C}}, \mathsf{blindRequest})$

1. Parse $\mathsf{blindRequest} = (\mathsf{com}_{\mathsf{msg}}, \mathsf{v}, \pi_{\mathsf{v}}, \mathsf{com}_{\mathsf{r_C}}, \mathsf{com}_{\mathsf{r}}, \pi_{\mathsf{r}}, \mathsf{VOPRF.digest})$.
2. Parse $\mathsf{privprm}_{\mathsf{S}} = \mathsf{sk}_{\mathsf{OPRF}}$.
3. Compute $\mathsf{r_S} \leftarrow \mathsf{H}(\mathsf{trnc})$ as in $\mathsf{ACT.Request}$.
4. Verify the proofs π_{v} and π_{r} and abort if any of them doesn't verify.
5. Compute the second message of the VOPRF evaluation

$$\mathsf{blindPRF} \leftarrow \mathsf{VOPRF.Eval}(\mathsf{sk}_{\mathsf{VOPRF}}, \mathsf{VOPRF.digest}).$$

Output: $\mathsf{blindToken} \leftarrow \mathsf{blindPRF}$
$\mathsf{tag} \leftarrow \mathsf{v}$

ACT.Unblind$(\mathsf{pprm}_{\mathsf{S}}, \mathsf{privprm}_{\mathsf{C}}, \mathsf{blindToken}, \mathsf{rand}_{\mathsf{msg}})$

1. Parse $\mathsf{blindToken} = \mathsf{blindPRF}$.
2. Parse $\mathsf{rand}_{\mathsf{msg}} = (\mathsf{msg}, \mathsf{r}, \mathsf{VOPRF.state})$.
3. Decode the returned token (implicitly verifying its correctness):

$$\tau \leftarrow \mathsf{VOPRF.Decode}(\mathsf{VOPRF.state}, \mathsf{blindPRF}).$$

4. Set the signature $\mathsf{tok} \leftarrow (\mathsf{r}, \tau)$

Output: $(\mathsf{msg}, \mathsf{tok})$

ACT.Verify$(\mathsf{pprm}_{\mathsf{S}}, \mathsf{privprm}, \mathsf{msg}, \mathsf{tok})$

1. Parse $\mathsf{privprm}_{\mathsf{S}} = \mathsf{sk}_{\mathsf{VOPRF}}$ and $\mathsf{tok} = (\mathsf{r}, \tau)$. Set $\mathsf{bit} \leftarrow$ **false**.
2. If $\mathcal{F}(\mathsf{sk}_{\mathsf{VOPRF}}, \mathsf{H}(\mathsf{msg}), \mathsf{r}) = \tau$, set $\mathsf{bit} \leftarrow$ **true**.

Output: bit

4.4 Verifiable Oblivious Pseudorandom Function Construction

Next, we present our construction of a verifiable oblivious pseudorandom function which closely follows the construction of distributed oblivious PRF of Miao et al. [MPR+20]. The main difference is that [MPR+20] relies on the selective pseudorandom property of the Dodis-Yampolskiy function while we use the extended Boneh-Boyen PRF function $\mathcal{F}(\mathsf{sk} = (\mathsf{u}, \mathsf{y}), \mathsf{msg}, \mathsf{r}) = (\mathsf{msg} + \mathsf{u} + \mathsf{r} \cdot \mathsf{y})^{-1} \cdot \mathsf{G}$.

Construction 4.4. Let $\mathsf{COM}_{\mathsf{Ped}}$ be the Pedersen commitment scheme, CS be the Camenisch-Shoup encryption scheme, and $\mathsf{COM}_{\mathsf{Ext}}$ be extractable commitment instantiated as CS encryption (but with a different public key, whose secret key is known by the issuer, contrary to the secret key of $\mathsf{COM}_{\mathsf{Ext}}$).We assume for simplicity that we use the same modulus N for CS, $\mathsf{COM}_{\mathsf{Ext}}$, and $\mathsf{COM}_{\mathsf{Ped}}$. We construct an VOPRF ($\mathsf{VOPRF.GenParam}$, $\mathsf{VOPRF.EncodeMsg}$, $\mathsf{VOPRF.Eval}$, $\mathsf{VOPRF.Decode}$) as follows:

VOPRF.GenParam(1^λ):

1. Generate CS parameters $(\mathsf{pk}_{\mathsf{CS}} \leftarrow (N, \mathsf{G}_{\mathsf{CS}}, \mathsf{Y}_{\mathsf{CS}}, \mathsf{H}_{\mathsf{CS}}), \mathsf{sk}_{\mathsf{CS}} \leftarrow x) \leftarrow \mathsf{CS.Gen}(1^\lambda)$.
2. Generate extractable commitment parameters $\mathsf{prm}_{\mathsf{Ext}} \leftarrow \mathsf{COM}_{\mathsf{Ext}}.\mathsf{Setup}(1^\lambda)$.
3. Generate Pedersen commitment parameters $\mathsf{prm}_{\mathsf{Ped}} \leftarrow \mathsf{COM}_{\mathsf{Ped}}.\mathsf{Setup}(1^\lambda)$. We use the same modulus N for the two commitment schemes and the CS scheme above.
4. Sample random keys $\mathsf{u}, \mathsf{y} \leftarrow\!\!\$\ \mathbb{Z}_{|\mathbb{G}|}$ for the function $\mathcal{F}$.
5. Encrypt $\mathsf{ct_u} \leftarrow \mathsf{CS.Enc}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{u}) = (r_{\mathsf{u}} \cdot \mathsf{G}_{\mathsf{CS}}, \mathsf{u} \cdot \mathsf{H}_{\mathsf{CS}} + r_{\mathsf{u}} \cdot \mathsf{Y}_{\mathsf{CS}})$ where $r_{\mathsf{u}} \leftarrow\!\!\$\ \mathbb{Z}_{\lfloor N/4 \rfloor}$.
6. Encrypt $\mathsf{ct_y} \leftarrow \mathsf{CS.Enc}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{y}) = (r_{\mathsf{y}} \cdot \mathsf{G}_{\mathsf{CS}}, \mathsf{y} \cdot \mathsf{H}_{\mathsf{CS}} + r_{\mathsf{y}} \cdot \mathsf{Y}_{\mathsf{CS}})$ where $r_{\mathsf{y}} \leftarrow\!\!\$\ \mathbb{Z}_{\lfloor N/4 \rfloor}$.

Output: $\mathsf{pprm_S} \leftarrow (\mathsf{pk}_{\mathsf{CS}}, \mathsf{prm}_{\mathsf{Ped}}, \mathsf{prm}_{\mathsf{Ext}}, \mathsf{ct_u}, \mathsf{ct_y})$
$\mathsf{privprm_S} \leftarrow (\mathsf{u}, \mathsf{y}, \mathsf{sk}_{\mathsf{CS}})$

VOPRF.EncodeMsg$(\mathsf{pprm_S},\ (\mathsf{msg}, r) \in \mathbb{Z}^2_{|\mathbb{G}|},\ (t_{\mathsf{msg}}, t_r))$:[6]

1. Commit $\mathsf{com_{msg}} \leftarrow \mathsf{Commit_{Ext}}(\mathsf{msg}; t_{\mathsf{msg}})$, $\mathsf{com}_r \leftarrow \mathsf{Commit_{Ped}}(r; t_r)$.
2. Sample $a \leftarrow\!\!\$\ \mathbb{Z}_p$ and $b \leftarrow\!\!\$\ \mathbb{Z}_{p^2 \cdot 2^\lambda}$.
3. Compute commitments with randomness $t_a, t_b, t_r \leftarrow\!\!\$\ \mathbb{Z}_{\lfloor N/4 \rfloor}$
   $\mathsf{com}_a \leftarrow \mathsf{Commit_{Ext}}(a; t_a), \mathsf{com}_b \leftarrow \mathsf{Commit_{Ped}}(b; t_b)$,
   $\mathsf{com}_r \leftarrow \mathsf{Commit_{Ped}}(r; t_r)$.
4. Let $\alpha = a \cdot \mathsf{msg},\ \ \gamma = a \cdot r$. Compute commitments:
$$\mathsf{com}_\alpha \leftarrow \mathsf{Commit_{Ped}}(\alpha, t_\alpha), \quad \mathsf{com}_\alpha \leftarrow \mathsf{Commit_{Ped}}(\gamma, t_\gamma)$$
5. Compute encryption of $\beta = a \cdot \mathsf{msg} + a \cdot (\mathsf{u} + r \cdot \mathsf{y}) + b \cdot p$ (implicitly defined):
$$\mathsf{ct}_\beta = \mathsf{CS.Enc}(\mathsf{pk}_{\mathsf{CS}}, \beta) = \mathsf{Enc}(\mathsf{pk}_{\mathsf{CS}}, a \cdot \mathsf{msg} + b \cdot p) + a \cdot \mathsf{ct_u} + a \cdot r \cdot \mathsf{ct_y}.$$
6. Generate a ZK proof
$$\begin{aligned}\pi = \mathsf{ZK}\{\exists\ & a, b, \mathsf{msg}, r, \alpha, \gamma, t_a, t_b, t_{\mathsf{msg}}, t_r, t_\alpha, t_\gamma \text{ s.t. :} \\ & \mathsf{ct}_\beta = \mathsf{Enc}(\mathsf{pk}_{\mathsf{CS}}, a \cdot \mathsf{msg} + b \cdot p) + a \cdot \mathsf{ct_u} + a \cdot r \cdot \mathsf{ct_y}, \\ & \mathsf{com}_a = \mathsf{Commit_{Ext}}(a; t_a),\ \mathsf{com}_b = \mathsf{Commit_{Ped}}(b; t_b), \\ & \mathsf{com}_r = \mathsf{Commit_{Ped}}(r; t_r),\ \mathsf{com_{msg}} = \mathsf{Commit_{Ext}}(\mathsf{msg}; t_{\mathsf{msg}}), \\ & \mathsf{com}_\alpha = \mathsf{Commit_{Ped}}(a \cdot \mathsf{msg}; t_\alpha),\ \mathsf{com}_\gamma = \mathsf{Commit_{Ped}}(a \cdot r; t_\gamma), \\ & a < p \cdot 2^{2\lambda+1}, \alpha < p \cdot 2^{2\lambda+1}, r < p \cdot 2^{2\lambda+1}, b < p^2 \cdot 2^{3\lambda+1}\}.\end{aligned}$$

Output: $\mathsf{digest} \leftarrow (\mathsf{ct}_\beta, \mathsf{com}_a, \mathsf{com}_b, \mathsf{com_{msg}}, \mathsf{com}_r, \mathsf{com}_\alpha, \mathsf{com}_\gamma, \pi)$
$\mathsf{state} \leftarrow (a, b)$

VOPRF.Eval$(\mathsf{privprm_S}, \mathsf{digest})$:

[6] Note that when called from the ACT, msg will actually be a hash of some message $\mathsf{H}(\mathsf{msg})$.

1. Parse $\mathsf{digest} \leftarrow (\mathsf{ct}, \pi)$. If π does not verify, abort.
2. Compute $\beta \leftarrow \mathsf{CS.Dec}(\mathsf{sk_{CS}}, \mathsf{ct})$. If $\beta \geq p^3 2^{\lambda+1}$, abort.
3. Compute commitment $\mathsf{com}_\beta = \mathsf{Commit_{Ped}}(\beta; r_\beta)$ where $r_\beta \leftarrow\$ \mathbb{Z}_{\lfloor N/4 \rfloor}$.
4. Set $\mathsf{F} = \beta^{-1} \cdot \mathsf{G}$.
5. Generate a ZK proof

$$\pi = \mathsf{ZK}\{\beta, r_\beta, \mathsf{sk_{CS}} \text{ s.t. : } \mathsf{ct} = (\mathsf{G}', \beta \cdot \mathsf{H_{CS}} + \mathsf{sk_{CS}} \cdot \mathsf{G}'), \mathsf{Y_{CS}} = \mathsf{sk_{CS}} \cdot \mathsf{G_{CS}}, \\ \mathsf{com}_\beta = \mathsf{Commit_{Ped}}(\beta; r_\beta), \mathsf{F} = \beta^{-1} \cdot \mathsf{G}, \beta < p^3 \cdot 2^{3\lambda+1}\}.$$

Output: $\mathsf{blindPRF} \leftarrow (\mathsf{F}, \mathsf{com}_\beta, \pi)$

VOPRF.Decode$(\mathsf{state}, \mathsf{blindPRF})$:

1. Parse $\mathsf{blindPRF} = (\mathsf{F}, \mathsf{com}_\beta, \pi), \mathsf{state} = (a, b)$.
2. If π does not verify, abort.
3. Set $\mathsf{F}' = a \cdot \mathsf{F}$.

Output: $\tau \leftarrow \mathsf{F}'$

Security Proof. We start with the intuition for our security proof. The first observation is that in the case of a single client, an ACT forgery corresponds to generating a new evaluation of the PRF $\mathcal{F}$ on a message that has not been queried. To formalize this, we leverage the result of Miao et al. [MPR+20, Theorem B.1] which constructs a distributed oblivious PRF evaluation protocol with malicious security on committed inputs for the Dodis-Yampolskiy PRF. Their result essentially shows that the VOPRF Construction 4.4 with $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{msg}) = (\mathsf{u} + \mathsf{msg})^{-1} \cdot \mathsf{G}$ is a secure two-party computation protocol where the client obtains $\mathsf{PRF}(\mathsf{u}, \mathsf{msg})$ and the server has no output. This means that if we instantiated the ACT construction with this VOPRF construction, then the ACT scheme will be unforgeable for a single client who chooses its messages selectively. Then, we show how we can reduce the unforgeability of the ACT construction with PRF $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{y}, \mathsf{msg}, \mathsf{r}) = (\mathsf{u} + \mathsf{msg} + \mathsf{r} \cdot \mathsf{y})^{-1} \cdot \mathsf{G}$ to the single client unforgeability of the ACT with the Dodis-Yampolskiy PRF. This reduction will follow the ideas of the reduction from unforgeability to weak unforgeability for the Boneh-Boyen signatures [BB04].

The unlinkability of the scheme follows from the selective pseudorandom property of the Dodis-Yampolskiy PRF. It indeed shows that the unlinkability adversary only obtains pseudorandom tokens that do not reveal any information about the underlying input messages.

We start by recalling the theorem of Miao et al. [MPR+20].

Theorem 4.5 ([MPR+20, Theorem B.1]). *The constructions of Figure 6 instantiated with* $\mathcal{F}(\mathsf{u}, \mathsf{msg}) = \mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{msg}) = (\mathsf{u} + \mathsf{msg}) \cdot \mathsf{G}$ *(instead of the extended Boneh-Boyen function* $\mathcal{F}$ *as written in Construction 4.4) and the algorithms from Construction 4.4 for oblivious evaluation, is a secure two-party computation protocol for which there exist simulators* $\mathsf{Sim_C}$ *which simulates the view of the client and* $\mathsf{Sim_S}$ *which simulate the view of the server.*

We proceed to formalize the single client unforgeability security which the above theorem will enable us to prove. The main difference of this weaker unforgeability notion, apart from considering a single user, is the following: it requires that the user commits to all messages it will use to query for tokens before the PRF key is chosen.

Definition 4.6 (Single Client Unforgeability). *An anonymous counting token scheme* ACT *is single client unforgeable if for any* PPT *adversary* $\mathcal{A}$ *and any* $\mathsf{T} \geq 0$

$$\mathsf{Adv}^{\mathsf{sc-omuf}}_{\mathsf{ACT}, \mathcal{A}, \mathsf{T}}(\lambda) := \Pr\left[\text{SC-OMUF}_{\mathsf{ACT}, \mathcal{A}, \mathsf{T}}(\lambda) = 1\right] = \mathsf{negl}(\lambda).$$

where $\text{SC-OMUF}_{\mathsf{ACT}, \mathcal{A}, \mathsf{T}}(\lambda)$ *is defined in Figure 3.*

```
Game $\mathrm{SC\text{-}OMUF}_{\mathsf{ACT},\mathcal{A},\mathsf{T}}(\lambda)$
$\mathsf{prm}_{\mathsf{Ext}} \leftarrow \mathsf{Gen}_{\mathsf{Ext}}(1^{\lambda})$
$(\mathsf{msg}_i, \mathsf{Commit}_{\mathsf{Ext}}(\mathsf{msg}_i, t_{\mathsf{msg}_i}))_{i\in[\mathsf{T}]}, \mathsf{pprm}_{\mathsf{C}} \leftarrow \mathcal{A}(\mathsf{prm}_{\mathsf{Ext}})$
$(\mathsf{pprm}'_{\mathsf{S}}, \mathsf{privprm}_{\mathsf{S}}) \leftarrow \mathsf{ACT.GenParam}'(1^{\lambda})$
/Generates all ACT parameters except $\mathsf{prm}_{\mathsf{Ext}}$, which was generated above/
$\mathsf{pprm}_{\mathsf{S}} = \mathsf{pprm}'_{\mathsf{S}} \cup \mathsf{prm}_{\mathsf{Ext}}$
$(\mathsf{blindRequest}_i)_{i\in[\mathsf{T}]}, \mathsf{pprm}_{\mathsf{C}} \leftarrow \mathcal{A}(\mathsf{pprm})$ where $\mathsf{blindRequest}_i[1] = \mathsf{Commit}_{\mathsf{Ext}}(\mathsf{msg}_i)$
/Each $\mathsf{blindRequest}_i$ is generated with respect to one of the committed messages/
$\mathsf{output} \leftarrow \mathcal{A}\Big(\mathsf{pprm}_{\mathsf{S}}, (\mathsf{ACT.Sign}(\mathsf{privprm}_{\mathsf{S}}, \mathsf{pprm}_{\mathsf{C}}, \mathsf{blindRequest}_i))_{i\in[\mathsf{T}]}\Big)$
$(\mathsf{msg}'_i, \mathsf{tok}'_i)_{i\in[\mathsf{T}+1]} := \mathsf{output}$
return
    $\Big(\ \forall i \in [\mathsf{T}+1],\quad \mathsf{ACT.Verify}(\mathsf{vrfyprm}, \mathsf{msg}'_i, \mathsf{tok}'_i) = \mathbf{true}\ \Big)$
```

Fig. 7: (Selective) Single Client unforgeability game for an ACT scheme

We translate Theorem 4.5 into a statement about single client unforgeability in the next lemma.

Lemma 4.7. *The* ACT *scheme from Construction 4.3 instantiated with the VOPRF from Construction 4.4 with* $\mathsf{y} = 0$ *satisfies single client unforgeability.*

Proof. Assume there is an adversary $\mathcal{A}$ that wins the single client unforgeability game in Figure 3 where $\mathcal{A}$ declares all of its queries together non-adaptively, with non-negligible probability. We construct two PPT adversaries $\mathcal{B}_{\mathsf{ZK}}$, which breaks the soundness of the ZK scheme, and $\mathcal{B}_{\mathsf{OPRF}}$ which breaks the security of the protocol for the distributed oblivious PRF evaluation.

$\mathcal{B}_{\mathsf{OPRF}}$ interacts with its challenger to obtain PRF parameters which it sends to $\mathcal{A}$. It invokes the client simulator $\mathsf{Sim}_{\mathsf{C}}$ that exists from Theorem 4.5 to interact and answer the signing queries from $\mathcal{A}$. Let $\mathcal{A}$ returns output $(\mathsf{msg}_i, \tau_i)_{i\in[\mathsf{T}+1]}$, then $\mathcal{B}$ returns $(\mathsf{msg}_i, \tau_i)_{i\in[\mathsf{T}+1]}$ as a forgery for the PRF.

We analyze the probability of success for $\mathcal{B}_{\mathsf{OPRF}}$. Assuming the soundness of the ZK protocol, all requested messages msg_i must be different since corresponding $\mathsf{tag}_i = \mathcal{F}_{\mathsf{DY}}(\mathsf{u}_{\mathsf{C}}, \mathsf{H}(\mathsf{msg}))$ are different. Therefore, $(\mathsf{msg}_i, \tau_i)_{i\in[\mathsf{T}+1]}$ are valid forgeries for the PRF.

Thus, we can bound the success probability for $\mathcal{A}$ as follows:

$$\mathsf{Adv}^{\mathrm{omuf}}_{\mathsf{ACT},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathrm{omuf}}_{\mathsf{OPRF},\mathcal{A}}(\lambda) + \mathsf{T} \cdot \mathsf{Adv}^{\mathrm{snd}}_{\mathsf{ZK},\mathcal{B}_{\mathsf{ZK}}}(\lambda). \tag{3}$$

This concludes the proof. □

Next, we proceed to prove the ACT unforgeability using the single user unforgeability from above.

Theorem 4.8. *The* ACT *scheme from Construction 4.3 instantiated with the VOPRF from Construction 4.4 with* $\mathsf{y} \neq 0$ *satisfies unforgeability from Definition 3.2.*

Proof. Let us assume there is an adversary $\mathcal{A}$ against the unforgeability game for ACT, we show how we can construct an adversary $\mathcal{B}$ against the construction with single user unforgeability above (which sets $\mathsf{y} = 0$).

We note that as in Lemma 10 of [BB04], there are actually 2 classes of adversaries that require 2 different types of reductions to single user unforgeability. For simplicity of exposition, $\mathcal{B}$ will give a reduction to single-user unforgeability that addresses only the first type of adversary. We will point out where this reduction can fail, and then give a second reduction for adversaries that consistently cause the first reduction to fail.

We first assume, without loss of generality, that $\mathcal{A}$ only makes successful queries to the Verify oracle on tokens that it received as the output of Sign. An $\mathcal{A}$ that makes successful queries to Verify on tokens *not* received from Sign can directly be turned into one that creates a successful forgery of Type 1 or 2 by guessing which of $\mathcal{A}$'s calls to Verify would succeed and using the input to the call as a forgery together with T honest calls to Sign.

With this restriction in mind, we give our construction of $\mathcal{B}$ breaking single user unforgeability. $\mathcal{B}$ receives from its challenger extractable commitment parameters $\mathsf{prm}^{\mathcal{B}}_{\mathsf{Ext}}$. It generates new T random messages $(\mathsf{msg}_i)_{i\in[\mathsf{T}]}$ and commits to them $(\mathsf{com}_i = \mathsf{Commit}_{\mathsf{Ext}}(\mathsf{H}(\mathsf{msg}_i)))_{i\in[\mathsf{T}]}$. $\mathcal{B}$ receives the rest of the parameters pprm' for the ACT from its challenger and interacts with its challenger to obtain tokens $(\tau_i)_{i\in[\mathsf{T}]}$ for the committed messages. Let $(a_i)_{i\in[\mathsf{T}]}$ be the randomness $\mathcal{B}$ used to compute $\mathsf{Enc}_{\mathsf{CS}}(\mathsf{pk}_{\mathsf{CS}}, a_i(\mathsf{H}(\mathsf{msg}_i) + \mathsf{u}) + bq)$ in its requests and $(\mathsf{blindPRF}_i)_{i\in[\mathsf{T}]}$ be the responses it received from its challenger for the PRF evaluation on all messages.

$\mathcal{B}$ simulates public parameters for $\mathcal{A}$ as follows: it generates parameters for an extractable commitment together with a trapdoor $\mathsf{prm}^{\mathcal{A}}_{\mathsf{Ext}}, \mathsf{trap}^{\mathcal{A}}_{\mathsf{Ext}}$, generates $\mathsf{y} \leftarrow\!\!\$\ \mathbb{Z}_p$, and provides $\mathsf{pprm} \leftarrow (\mathsf{pprm}', \mathsf{prm}^{\mathcal{A}}_{\mathsf{Ext}}, \mathsf{ct}_{\mathsf{y}} = \mathsf{Enc}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{y}))$ to $\mathcal{A}$ as public parameters.

$\mathcal{B}$ initializes $\mathcal{Q} = \emptyset$ and answers the i-th random oracle query from $\mathcal{A}$ as follows: on input $\mathsf{trnc} = (\mathsf{pprm}, \mathsf{com}_{\mathsf{uc}}, \mathsf{com}_{\mathsf{msg}}, \mathsf{v}, \mathsf{com}_{\mathsf{rc}})$, extracts $\mathsf{H}(\mathsf{msg})$ (just the hash value, not the preimage msg) and r_{C} from the commitments $\mathsf{com}_{\mathsf{msg}}$ and $\mathsf{com}_{\mathsf{rc}}$, sets the RO answer $\mathsf{r}_{\mathsf{S}} = (\mathsf{H}(\mathsf{msg}_i) - \mathsf{H}(\mathsf{msg})) \cdot \mathsf{y}^{-1} - \mathsf{r}_{\mathsf{C}}$ (i.e., $\mathsf{msg}_i = \mathsf{msg} + (\mathsf{r}_{\mathsf{C}} + \mathsf{r}_{\mathsf{S}}) \cdot \mathsf{y}$). It adds $(\mathsf{com}_{\mathsf{msg}}, \mathsf{H}(\mathsf{msg}), \mathsf{H}(\mathsf{msg}_i))$ to $\mathcal{Q}$.

$\mathcal{B}$ answers signing queries from $\mathcal{A}$ as follows: it extracts $\mathsf{H}(\mathsf{msg})$ as above. It verifies the proof of correct evaluation of the PRF $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}_{\mathsf{C}}, \mathsf{H}(\mathsf{msg}))$, and if it fails or shows repeating messages, aborts. It extracts the value a from com_a. If $\exists\ (\mathsf{com}_{\mathsf{msg}}, \mathsf{H}(\mathsf{msg}), \mathsf{H}(\mathsf{msg}_i)) \in \mathcal{Q}$, return $a_i \cdot a^{-1} \cdot \mathsf{blindPRF}_i$. If $(\mathsf{com}_{\mathsf{msg}}, \mathsf{H}(\mathsf{msg}), \mathsf{H}(\mathsf{msg}_i)) \notin \mathcal{Q}$, $\mathcal{B}$ aborts. $\mathcal{B}$ uses a commitment to 0 as the commitment to β in the rseponse, and uses the ZK simulator to simulate the proof of response.

$\mathcal{B}$ answers verification queries from $\mathcal{A}$ as follows: on input $(\mathsf{msg}, \mathsf{r}, \tau)$, check whether $\exists\ (\mathsf{com}_{\mathsf{msg}}, \mathsf{H}(\mathsf{msg}), \mathsf{H}(\mathsf{msg}_i)) \in \mathcal{Q}$. If this does not hold, return **false**. If it holds, check whether $\mathsf{H}(\mathsf{msg}) + \mathsf{r} \cdot \mathsf{y} = \mathsf{H}(\mathsf{msg}_i)$. If the check fails, return **false**. Otherwise, if $\tau = \mathsf{tok}_i$, return **true**, else return **false**.

Let $(\mathsf{msg}'_i, \mathsf{r}'_i, \tau'_i)_{i\in[\mathsf{T}+1]}$ be a successful forgery returned by $\mathcal{A}$. $\mathcal{B}$ returns the same set to its challenger.

We argue that the view of $\mathcal{A}$ is indistinguishable from that in the security game $\mathrm{OMUF}_{\mathsf{ACT},\mathcal{A},\mathsf{R},\mathsf{T}}(\lambda)$ in Definition 3.2. For all signing queries for which $\mathcal{A}$ queries the random oracle with the correct transcript, the responses are the correct PRF evaluation. Assuming the soundness of the ZK, signing queries where the adversary has not queried the RO for the corresponding randomness value, cannot have valid ZK proofs in the query. Assuming the ZK soundness and the pseudorandom function properties, $\mathcal{A}$ will not be able to guess a correct token for a message if it has not queried the RO on the corresponding randomness and then the OPRF for the corresponding evaluation.

We now analyze the probability of success for $\mathcal{B}$. We first consider the case when all the values τ'_i output by $\mathcal{A}$ are distinct. This implies that the values $\mathsf{msg}'_i + \mathsf{y} \cdot \mathsf{r}'_i$ are also all distinct (except when y is 0, which happens with probability $\frac{1}{p}$). If $\mathcal{A}$ generated Type 1 forgery, then assuming ZK soundness all msg'_i are different and hence this is also a valid forgery for $\mathcal{B}$. Now we argue that if the ZK protocol is sound, then $\mathcal{A}$ could not have generated a forgery of Type 2, which is of the form $(\mathsf{msg}', \mathsf{r}'_i, \tau'_i)_{i\in[\mathsf{R}+1]}$. The ZK soundness guarantees that $\mathcal{B}$ would not have answered more than R requests for the same message. Then the only other option for forgery is if $\mathcal{A}$ made another query for message msg'_j and $\exists\ k \in [\mathsf{R}+1]$ made such that $\mathsf{H}(\mathsf{msg}'_k) + \mathsf{y} \cdot \mathsf{r}'_k = \mathsf{H}(\mathsf{msg}'_j) + \mathsf{y} \cdot \mathsf{r}'_j$, which we have, for the time being, assumed does not happen.

Thus, we can bound the success probability for $\mathcal{A}$ as:

$$\mathsf{Adv}^{\text{omuf}}_{\mathsf{ACT},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{sc-omuf}}_{\mathsf{ACT},\mathcal{A}}(\lambda) + \mathsf{T} \cdot \mathsf{Adv}^{\text{snd}}_{\mathsf{ZK},\mathcal{B}_{\mathsf{ZK}}}(\lambda) \tag{4}$$

We now consider the case when the values τ'_i output by $\mathcal{A}$ are *not* all distinct. Since this is a successful forgery, this implies $\exists i, j \in [\mathsf{T}+1]$ such that $(\mathsf{msg}'_i, \mathsf{r}'_i) \neq (\mathsf{msg}'_j, \mathsf{r}'_j)$ but $\mathsf{msg}'_i + \mathsf{y} \cdot \mathsf{r}'_i = \mathsf{msg}'_j + \mathsf{y} \cdot \mathsf{r}'_j$. If there exists an adversary $\mathcal{A}^\star$ that produces such a forgery with non-negligible probability, then we show how to construct an adversary $\mathcal{B}^\star$ that breaks single user unforgeability. As in Lemma 10 of [BB04], we will show that $\mathcal{A}^\star$ can learn something extra about y that it shouldn't learn.

We construct $\mathcal{B}^\star$ as follows. $\mathcal{B}^\star$ initally proceeds exactly as $\mathcal{B}$: $\mathcal{B}^\star$ receives extractable commitment parameters $\mathsf{prm}^{\mathcal{B}^\star}_{\mathsf{Ext}}$. It generates new T random messages $(\mathsf{msg}_i)_{i\in[\mathsf{T}]}$, receives the rest of the parameters pprm' for the ACT from its challenger and interacts with its challenger to obtain tokens $(\tau_i)_{i\in[\mathsf{T}]}$ for the committed messages. As earlier, let $(a_i)_{i\in[\mathsf{T}]}$ be the randomness $\mathcal{B}^\star$ used in its requests and $(\mathsf{blindPRF}_i)_{i\in[\mathsf{T}]}$ be the responses it received from its challenger for the PRF evaluation on all messages.

As earlier, $\mathcal{B}^\star$ generates fresh parameters for the extractable commitment together with a trapdoor $\mathsf{prm}^{\mathcal{A}}_{\mathsf{Ext}}, \mathsf{trap}^{\mathcal{A}}_{\mathsf{Ext}}$, but instead of generating $\mathsf{y} \leftarrow\$ \mathbb{Z}_p$, $\mathcal{B}^\star$ instead uses the encryption $ct_{\mathsf{u}} = \mathsf{Enc}_{\mathsf{CS}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{u})$ received from its challenger as part of pprm' as y, and generates a fresh $\mathsf{u}' \leftarrow\$ \mathbb{Z}_p$. It forwards these parameters to $\mathcal{A}^\star$ as public parameters (that is, it provides the freshly generated $\mathsf{prm}^{\mathcal{A}}_{\mathsf{Ext}}$ as the extractable commitment parameters, provides ct_{u} received from the challenger as the encryption of y, and provides $\mathsf{ct}_{\mathsf{u}'} = \mathsf{Enc}_{\mathsf{CS}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{u}')$ as the encryption of u).

$\mathcal{B}^\star$ initializes $\mathcal{Q} = \emptyset$ and answers the i-th random oracle query from $\mathcal{A}^\star$ as follows: on input $\mathsf{trnc} = (\mathsf{pprm}, \mathsf{com}_{\mathsf{u_C}}, \mathsf{com}_{\mathsf{msg}}, \mathsf{v}, \mathsf{com}_{\mathsf{r_C}})$, extracts $\mathsf{H}(\mathsf{msg})$ and $\mathsf{r_C}$ from the commitments $\mathsf{com}_{\mathsf{msg}}$ and $\mathsf{com}_{\mathsf{r_C}}$. If $\mathsf{com}_{\mathsf{msg}} = -\mathsf{u}'$ or $\mathsf{H}(\mathsf{msg}_i) = 0$, $\mathcal{B}^\star$ aborts. Otherwise, it computes $\mathsf{r}_i = (\mathsf{H}(\mathsf{msg}) + \mathsf{u}') \cdot \mathsf{H}(\mathsf{msg}_i)^{-1}$ and sets the RO answer $\mathsf{r_S} = \mathsf{r}_i - \mathsf{r_C}$. Note that this makes it so that $\mathsf{H}(\mathsf{msg}) + \mathsf{u}' + \mathsf{r}_i \cdot \mathsf{u} = \mathsf{r}_i \cdot (\mathsf{H}(\mathsf{msg}_i) + \mathsf{u})$.

$\mathcal{B}^\star$ answers signing queries from $\mathcal{A}^\star$ as follows: it extracts $\mathsf{H}(\mathsf{msg})$ as above. It verifies the proof of correct evaluation of the PRF $\mathcal{F}_{\mathsf{DY}}(\mathsf{u_C}, \mathsf{H}(\mathsf{msg}))$, and if it fails or shows repeating messages, aborts. It extracts the value a from com_a. If $\exists\, (\mathsf{com}_{\mathsf{msg}}, \mathsf{H}(\mathsf{msg}), \mathsf{H}(\mathsf{msg}_i)) \in \mathcal{Q}$, return $a_i \cdot \mathsf{r}_i^{-1} \cdot a^{-1} \cdot \mathsf{blindPRF}_i$ in the VOPRF response, using a commitment to 0 as the commitment to β in the reponse, and using the ZK simulator to simulate the proof of response. If $(\mathsf{com}_{\mathsf{msg}}, \mathsf{H}(\mathsf{msg}), \mathsf{H}(\mathsf{msg}_i)) \notin \mathcal{Q}$, $\mathcal{B}^\star$ aborts.

We observe that in the returned value, $\mathsf{blindPRF}_i = a_i^{-1} \cdot (\mathsf{u} + \mathsf{H}(\mathsf{msg}_i))^{-1} \cdot \mathsf{G}$, so $a_i \cdot \mathsf{r}_i^{-1} \cdot a^{-1} \cdot \mathsf{blindPRF}_i = a^{-1} \cdot \mathsf{r}_i^{-1} \cdot (\mathsf{u} + \mathsf{H}(\mathsf{msg}_i))^{-1} \cdot \mathsf{G}$. Since we chose r_i so that $\mathsf{H}(\mathsf{msg}) + \mathsf{u}' + \mathsf{r}_i \cdot \mathsf{u} = \mathsf{r}_i \cdot (\mathsf{H}(\mathsf{msg}_i) + \mathsf{u})$, we have that the returned value corresponds to $a^{-1} \cdot (\mathsf{H}(\mathsf{msg}) + \mathsf{u}' + \mathsf{r}_i \cdot \mathsf{u})^{-1} \cdot \mathsf{G}$, which is of the correct form.

$\mathcal{B}^\star$ answers verification queries from $\mathcal{A}^\star$ as follows: on input $(\mathsf{msg}, \mathsf{r}, \tau)$, check whether $\exists\, (\mathsf{com}_{\mathsf{msg}}, \mathsf{H}(\mathsf{msg}), \mathsf{H}(\mathsf{msg}_i)) \in \mathcal{Q}$. If this does not hold, return **false**. If it holds, check if $\mathsf{r} = \mathsf{r}_i$ and if $\tau = \mathsf{tok}_i^{\mathsf{r}_i}$. If these both hold, return **true**, else return **false**.

Now, whenever $\mathcal{A}^\star$ outputs a forgery with $(\mathsf{msg}, \mathsf{r}, \tau)$ and $(\mathsf{msg}', \mathsf{r}', \tau)$ that are both correctly verifying tokens, this implies that $\mathsf{msg} + \mathsf{u}' + \mathsf{u} \cdot \mathsf{r} = \mathsf{msg}' + \mathsf{u}' + \mathsf{u} \cdot \mathsf{r}'$. $\mathcal{B}^\star$ can therefore solve for u and use it to create a forgery for the underlying single-user game.

We show that $\mathcal{B}^\star$ has a similar advantage in the SC-OMUF game as $\mathcal{A}^\star$ has in the OMUF game.

We do so by defining a sequence of hybrid games as below:

Hyb_1 The real execution of $\mathcal{A}^\star$ with the OMUF challenger.

Hyb_2 Same as above, but the OMUF challenger keeps track of the queries $\mathcal{Q}$ made to the RO, and aborts if $\mathcal{A}^\star$ makes a successful Sign call with a valid ZK proof, with $\mathsf{com}_{\mathsf{msg}}$ and $\mathsf{com}_{\mathsf{r_C}}$ that it has never queried to the RO.

Hyb_3 Same as above, but the OMUF challenger extracts all witnesses from $\mathcal{A}^\star$'s ZK proofs and checks them for soundness, aborting otherwise.

Hyb_4 Same as above, except the OMUF challenger uses the ZK simulator for the proofs it sends to $\mathcal{A}^\star$. (In particular, the proof for the VOPRF response in Sign)

Hyb_5 Same as above, except the OMUF challenger uses a commitment to 0 instead of a commitment to β in the VOPRF response (but continues simulating the proof with the correct PRF output)

Hyb_6 Same as above, except the OMUF challenger pre-selects T messages msg_i at the start of the interaction, and programs the RO to generate r as $\mathcal{B}^\star$ does. Namely, on the i'th RO query with passing proofs containing, say, $\mathsf{com}_{\mathsf{msg}}, \mathsf{com}_{\mathsf{r_C}}$, the OMUF challenger extracts $\mathsf{H}(\mathsf{msg})$ and $\mathsf{r_C}$ and computes $\mathsf{r}_i = (\mathsf{H}(\mathsf{msg}) + \mathsf{u}) \cdot \mathsf{H}(\mathsf{msg}_i)^{-1}$ and sets the RO answer $\mathsf{r_S} = \mathsf{r}_i - \mathsf{r_C}$.

Hyb_7 Same as above, except that the OMUF challenger precomputes $\mathsf{blindPRF}_i$ on the values msg_i at the start, and uses these in the responses to Sign as $\mathcal{B}^\star$ does. Namely, the OMUF challenger looks up the Sign query in the RO query list $\mathcal{Q}$ to find the correspond msg_i, $\mathsf{blindPRF}_i$ and r_i, extracts a from the query, and uses $a_i \cdot \mathsf{r}_i^{-1} \cdot a^{-1} \cdot \mathsf{blindPRF}_i$ in its Sign response.

Hyb_8 Same as above, except the OMUF uses an internally simulated SC-OMUF challenger to generate τ_i and $\mathsf{blindPRF}_i$ at the start of the interaction. The OMUF also forwards the $\mathsf{ct_u}$ from the internal SC-OMUF challenger to $\mathcal{A}^\star$ in the position of $\mathsf{ct_y}$, as $\mathcal{B}^\star$ does.

Hyb_9 The interaction of $\mathcal{B}^\star$ with the SC-OMUF challenger.

We can see that each sequence of hybrids above is indistinguishable. Hyb_1 and Hyb_2 are indistinguishable based on soundness of the ZK proof in Sign together with the negligible collision probability of the RO. The indistinguishability of Hyb_3 and Hyb_4 directly follow from the security of the corresponding ZK proofs. Hyb_5 depends on the hiding property of com. Hyb_6 is identical except when $\mathsf{H}(\mathsf{msg}) = \mathsf{u}$ which should happen with negligible probability by the exponential size of p and the security of ct_u. The remaining hybrids are identically distributed using our initial assumption that $\mathcal{A}^\star$ never makes successful Verify queries except on tokens that were received via Sign.

Putting these together, if $\mathcal{A}^\star$ interacting with the OMUF challenger can produce a forgery $(\mathsf{msg}'_i, \mathsf{r}'_i, \tau'_i)_{i\in[\mathsf{T}+1]}$ such that $\exists i, j \in [\mathsf{T}+1]$ such that $(\mathsf{msg}'_i, \mathsf{r}'_i) \neq (\mathsf{msg}'_j, \mathsf{r}'_j)$, then $\mathcal{B}^\star$ can produce a similar output with negligibly close probability while interacting with the SC-OMUF challenger, and can use this forgery to extract the u of the SC-OMUF challenger to win the SC-OMUF game.

This concludes the proof of unforgeability.

$\square$

Unlinkability. Next we prove the unlinkability of ACT Construction 4.3.

Theorem 4.9. *If* $\mathcal{F}_{\mathsf{DY}}(\mathsf{msg}, \mathsf{u}) = (\mathsf{msg} + \mathsf{u})^{-1}$ *is selectively pseudorandom,* ZK *is a zero-knowledge ZK argument, then the* ACT *scheme in Construction 5.1 satisfies unlinkability from Definition 3.3, when* H *is modelled as a random oracle.*

Recall that $\mathcal{F}_{\mathsf{DY}}$ is selectively pseudorandom. Note that we do not need $\mathcal{F}_{\mathsf{DY}}$ to be pseudorandom since it is always evaluated on hashes of messages and not messages themselves.

Proof. We consider the following hybrids

Hyb_1 This is the regular execution of the unlinkability game.

Hyb_2 In this game we start to simulate the ZK that the client provides for the honest evaluation of $\mathsf{v} = \mathcal{F}_{\mathsf{DY}}(\mathsf{u}_\mathsf{C}, h)$ in $\mathsf{Request}$ (with $h = \mathsf{H}(\mathsf{msg})$). This indistinguishability follows from the ZK properties.

Hyb_3 In this game instead of $\mathsf{v} = \mathcal{F}_{\mathsf{DY}}(\mathsf{u}_\mathsf{C}, \mathsf{H}(\mathsf{msg}))$, the client provides a truly random value. The indistinguishability follows from the selective pseudorandom property of $\mathcal{F}_{\mathsf{DY}}$ and the fact H is modeled as a random oracle. More precisely, using a classical hybrid argument, we just need to handle a single client C at a time. Let us show a reduction from selective pseudorandom of $\mathcal{F}_{\mathsf{DY}}$ to distinguishing when v is uniformly random or computed honestly for C. Assuming q_H requests are made to H in total, the reduction first generates q_H random scalars $h_1, \ldots, h_{q_\mathsf{H}}$ and ask the PRF oracle their PRF value. Then the reduction starts simulating the game. For each query to H on a new input of the form msg, the reduction answers with the next available h_i. This allows the reduction to use the PRF values output by the PRF oracle to compute the value v. If the PRF oracle actually evaluated $\mathcal{F}_{\mathsf{DY}}$, then the reduction would perfectly simulate the game when v are honestly computed for C. Otherwise, it would perfectly simulate the game when v is chosen uniformly at random for C.

Hyb_4 In this game we choose a set of fixed messages msg'_i and then program the random oracle H so that it determines the randomness r_i for each execution in such a way that for the i-th execution $(\mathsf{bR}_i, \mathsf{r}_i) \leftarrow \mathsf{ACT.Request}(\mathsf{pprm}_\mathsf{S}, \mathsf{privprm}_\mathsf{C}, \mathsf{msg}_i)$ (which includes the calls $\mathsf{ACT.Request}(\mathsf{pprm}_\mathsf{S}, \mathsf{privprm}_{\mathsf{b}_\mathsf{chl}}, \mathsf{msg}_{\mathsf{b}_\mathsf{chl}})$ from the challenge oracle) we have that $\mathsf{msg}'_i = \mathsf{H}(\mathsf{msg}) + \mathsf{r}_i \cdot \mathsf{y}$.

Hyb_5 In this game we use the issuer simulator from Theorem 4.5 to generate the response for the $\mathsf{ACT.Request}$ oracle calls and answer $\mathsf{ACT.Unblind}$ queries by returning $\mathcal{F}((\mathsf{u}, \mathsf{y}), \mathsf{H}(\mathsf{msg}), \mathsf{r}_i) = \mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{msg}'_i)$ the random value r_i associated with the respective execution. The indistinguishability follows from the security of the simulator.

The answers to the oracle queries in this hybrid are independent of the value bit (we have mapped the challenge queries to the same fixed messages). Therefore, the adversary succeeds with only negligible probability. $\square$

Parameter Generation for Unlinkability. Unlinkability assumes that the ACT parameters are generated honestly and the issuer only sees the secret key. These parameters include Camenish-Shoup encryption parameters (twice, one for the CS encryption for which the issuer knows the secret key, and one for the extractable commitment for which the issuer does not know the secret key and Pedersen commitment parameters over a group where the Strong RSA holds, as well as encryption of the issuer's private keys for Boneh-Boyen function. We can distribute the generation of these parameters across multiple parties. Both CS and Pedersen parameters require the generation of group where the Strong RSA assumption holds. This can be done in a distributed way leveraging proof techniques for showing that the modulus N is a product of two large safe primes [GMR98, CM99]. The public parameters for the CS encryption and extractable commitment can be generated by sampling in a distributed manner the CS private key (which for CS encryption will be given to the issuer) and of the extractable commitment secret key (which is destroyed), and then computing the shares of the corresponding public keys.

Instantiation. We recall that we use a single RSA modulus N for all the RSA-based primitives. We estimate the communications costs of $\mathsf{blindRequest}$ sent from the user to the server which consists of

– $\mathsf{VOPRF.digest}$ and the message and randomness commitments which amounts to 4 Pedersen commitments, 3 CS ciphertexts and 13 scalars of total $9\log(p) + 7\log(N) + 27\lambda$ bits (see Appendix A for description of the sigma protocol in Construction A.4 and the cost estimate.)
– $\mathsf{v}, \pi_{\mathsf{v}}$ which is 1 additional Pedersen commitment, and 4 scalars of total size $2\log(p) + 2\log(N) + 8\lambda$ (we note that we can optimize the Pedersen commitment and the scalars to be smaller here but we will count them as the ones above for simplicity). Alongside this, we also send 1 group element corresponding to the actual VRF output.
– $\mathsf{com}_{\mathsf{rc}}, \pi_{\mathsf{r}}$ which contains 1 additional extractable commitment, and 3 scalars of total size $\log(p) + 2\log(N) + 6\lambda$.

So we have a total for $\mathsf{blindRequest}$: 5 Pedersen commitments,4 CS ciphertexts, 1 group element and 20 scalars of total size $12\log(p) + 11\log(N) + 41\lambda$.

The communication for $\mathsf{blindToken}$ is the same as that for Eval in the underlying VOPRF. This consists of 1 group element of $\log(p)$ bits and a Sigma protocol. The Sigma protocol is the same as the proof in Figure 14, Section C.7.2 [MPR$^+$20]. Applying the optimization described in Appendix A, this sigma protocol has communication cost 1 Pedersen commitment and 3 scalars of total size $3\log(N) + 6\lambda$ bits. Therefore the total for Eval is 1 group element, 1 Pedersen commitment, and scalars of total size $3\log(N) + 6\lambda$.

5 ACTs from Equivalence-Class Signature

In this section, we present ACT constructions from equivalence-class signatures (EQS). Note that these constructions will have the functionality where the rate-limiting of a single token per message per client will be enforced during token redemption. In particular, the ACT verification will output the verification bit of the validity of the token and a tag that is a pseudorandom value derived from the client's key and the message. The issuer can compare this tag against its database of redeemed message tags and reject the token if this value occurs there. However, this latter rate-limiting step is not part of the ACT verification algorithm itself.

We present two ACT constructions which differ in the type of PRF used to enforce the rate-limiting property. The first one is based on the Dodis-Yampolskiy PRF and can be instantiated in the standard model but is limited to messages of size logarithmic in the security parameter. The second one has two versions: the long version is provably secure in the random oracle model, while the short version is provably secure in the generic bilinear group and random oracle model when instantiated with a specific scheme.

5.1 ACT from EQS and Dodis-Yampolskiy PRF for Small Messages

We start with an ACT construction from EQS and the Dodis-Yampolskiy PRF.

Construction 5.1 (ACT from EQS and Dodis-Yampolskiy). Let $\mathsf{EQS} = (\mathsf{EQS.Setup}, \mathsf{EQS.KGen}, \mathsf{EQS.Sign}, \mathsf{EQS.Adapt}, \mathsf{EQS.Verify})$ be an equivalence-class signature scheme over a bilinear group $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. We use the Dodis-Yampolskiy pseudorandom function $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}, x) = (\mathsf{u} + x)^{-1} \cdot \mathsf{G}_1$ over the cyclic group $\mathbb{G}_1$. We assume messages msg are in a subset of $\mathbb{Z}_p^*$ of size polynomial in the security parameter λ. An anonymous counting token construction ACT consists of the following algorithms:

ACT.GenParam(1^λ): Generate

1. a bilinear group $\mathcal{PG} \leftarrow \mathsf{GGen}(1^\lambda)$ and an extra random generator $\mathsf{G}_1' \in \mathbb{G}_1$
2. ZK argument CRS $\mathsf{ZK.crs} \leftarrow \mathsf{ZK.Setup}(\mathcal{R})$ where $\mathcal{R}$ is implicitly defined in $\mathsf{ACT.Request}$ ($\mathsf{ZK.crs}$ is used implicitly when generating and verifying ZK proofs),
3. EQS CRS $\mathsf{crs} \leftarrow \mathsf{EQS.Setup}(\mathcal{PG})$, and EQS keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{EQS.KGen}(\mathsf{crs})$.

Output: $\mathsf{pprm}_\mathsf{S} \leftarrow (\mathsf{ZK.crs}, \mathsf{crs}, \mathsf{pk})$
$\mathsf{privprm}_\mathsf{S} \leftarrow (\mathsf{pprm}_\mathsf{S}, \mathsf{sk})$.

ACT.ClientRegister$(\mathsf{pprm}_\mathsf{S})$:

1. Generate a Dodis-Yampolskiy PRF key $\mathsf{u}_\mathsf{C} \leftarrow\!\!\$\; \mathbb{Z}_p$,
2. Set $\mathsf{U}_\mathsf{C} = \mathsf{u}_\mathsf{C} \cdot \mathsf{G}_1$.

Output: $\mathsf{pprm}_\mathsf{C} \leftarrow \mathsf{U}_\mathsf{C}$
$\mathsf{privprm}_\mathsf{C} \leftarrow \mathsf{u}_\mathsf{C}$

ACT.TokenRequest$(\mathsf{pprm}_\mathsf{S}, \mathsf{privprm}_\mathsf{C}, \mathsf{msg} \in \mathbb{Z}_p^*)$:

1. Generate a random value $\mu \leftarrow\!\!\$\; \mathbb{Z}_p^*$.
2. Set $\vec{\mathsf{M}} \leftarrow (\mathsf{M}_1 = \mu^{-1} \cdot \mathsf{G}_1,\ \mathsf{M}_2 = \mu^{-1} \cdot \mathcal{F}_{\mathsf{DY}}(\mathsf{u}_\mathsf{C}, \mathsf{msg}),\ \mathsf{M}_3 = (\mu^{-1}\mathsf{msg}) \cdot \mathsf{G}_1')$.
3. Generate a proof $\pi_{\vec{\mathsf{M}}}$:

$$\pi_{\vec{\mathsf{M}}} : \mathsf{ZK}\big\{\exists\ \mu^{-1}, \mathsf{u}_C \in \mathbb{Z}_p : \mathsf{M}_1' = \mu^{-1} \cdot \mathsf{G}_1,\ \mathsf{M}_2' = \mu^{-1}(\mathsf{u}_\mathsf{C} + \mathsf{msg})^{-1} \cdot \mathsf{G}_1, \\ \mathsf{M}_3 = (\mu^{-1}\mathsf{msg}) \cdot \mathsf{G}_1,\ \mathsf{U}_\mathsf{C} = \mathsf{u}_\mathsf{C} \cdot \mathsf{G}_1\big\}$$

Output: $\mathsf{blindRequest} \leftarrow (\vec{\mathsf{M}}, \pi_{\vec{\mathsf{M}}})$
$\mathsf{rand}_\mathsf{msg} \leftarrow (\mathsf{msg}, \vec{\mathsf{M}}, \mu)$

ACT.Sign$(\mathsf{privprm}_\mathsf{S}, \mathsf{pprm}_\mathsf{C}, \mathsf{blindRequest})$

1. Parse $\mathsf{blindRequest} = (\vec{\mathsf{M}}, \pi_{\vec{\mathsf{M}}})$.
2. If verification of $\pi_{\vec{\mathsf{M}}}$ fails, abort.
3. Run $\rho \leftarrow \mathsf{EQS.Sign}(\mathsf{crs}, \mathsf{sk}, \vec{\mathsf{M}})$.

Output: $\mathsf{blindToken} \leftarrow \rho$

ACT.Unblind$(\mathsf{pprm}_\mathsf{S}, \mathsf{privprm}_\mathsf{C}, \mathsf{blindToken}, \mathsf{rand}_\mathsf{msg})$

1. Parse $\mathsf{rand}_\mathsf{msg} = (\mathsf{msg}, \vec{\mathsf{M}}, \mu)$, $\mathsf{blindToken} = \rho$.

2. Set $\vec{\mathsf{M}}' \leftarrow (\mathsf{G}_1,\ \mathcal{F}_{\mathsf{DY}}(\mathsf{u_C}, \mathsf{msg}),\ \mathsf{msg} \cdot \mathsf{G}_1')$.
3. Compute $\sigma \leftarrow \mathsf{EQS.Adapt}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}, \rho, \mu)$.
4. If $\sigma = \perp$, abort.

Output: $\Big(\mathsf{msg},\ \mathsf{tok} \leftarrow (\vec{\mathsf{M}}', \sigma)\Big)$

ACT.Verify$(\mathsf{pprm_S}, \mathsf{privprm}, \mathsf{msg}, \mathsf{tok})$

1. Set $\mathsf{bit} \leftarrow$ **true**.
2. Parse $\mathsf{tok} = (\vec{\mathsf{M}}' = (\mathsf{M}_1', \mathsf{M}_2', \mathsf{M}_3'), \sigma)$.
3. If $\mathsf{M}_1' \neq \mathsf{G}_1$ or $\mathsf{M}_3' \neq \mathsf{msg} \cdot \mathsf{G}_1'$, set $\mathsf{bit} \leftarrow$ **false**.
4. If **false** $= \mathsf{EQS.Verify}(\mathsf{pk}, \vec{\mathsf{M}}', \sigma)$, set $\mathsf{bit} \leftarrow$ **false**.

Output: $(\mathsf{bit},\ \mathsf{M}_2)$

Perfect correctness is straightforward.

6 Proofs for ACT from EQS Constructions

Unforgeability.

Theorem 6.1. *If* EQS *provides EUF-CMA security,* ZK *is a sound ZK argument, then the* ACT *scheme in Construction 5.1 satisfies unforgeability from Definition 3.2.*

Note that unforgeability does not rely on the pseudorandomness of the Dodis-Yampolskiy PRF.

Proof. Assume there is an adversary $\mathcal{A}$ that wins the unforgeability security game in Figure 3 with non-negligible probability. We construct two PPT adversaries: $\mathcal{B}_{\mathsf{EQS}}$ which breaks the EUF-CMA security of the EQS and $\mathcal{B}_{\mathsf{ZK}}$ which breaks the soundness of the ZK scheme. We show that:

$$\mathsf{Adv}^{\text{omuf}}_{\mathsf{ACT},\mathcal{A}}(\lambda) \leq (\mathsf{T} + \mathsf{R} + 1) \cdot \mathsf{Adv}^{\text{euf-cma}}_{\mathsf{EQS},\mathcal{B}_{\mathsf{EQS}}}(\lambda) + \mathsf{T} \cdot \mathsf{Adv}^{\text{snd}}_{\mathsf{ZK},\mathcal{B}_{\mathsf{ZK}}}(\lambda). \tag{5}$$

where $\mathsf{Adv}^{\text{snd}}_{\mathsf{ZK},\mathcal{B}_{\mathsf{ZK}}}(\lambda)$ is the probability defined in Eq. (2) on Page 9 (which itself implicitly defines a soundness game $\text{SND}_{\mathcal{B}_{\mathsf{ZK}}}(\lambda)$), T is the number of queries to the Sign oracle and R is the number of queries to the Register oracle.

Construction of the adversaries. Let us first construct $\mathcal{B}_{\mathsf{EQS}}$. $\mathcal{B}_{\mathsf{EQS}}$ obtains pk from the challenges for the EQS scheme and provides it as public parameters to $\mathcal{A}$. $\mathcal{B}_{\mathsf{EQS}}$ answers signing queries $\vec{\mathsf{M}}_j, \pi_{\vec{\mathsf{M}}_j}$ from $\mathcal{A}$ as follows. If $\pi_{\vec{\mathsf{M}}_j}$ fails to verify, abort. Else $\mathcal{A}$ queries the EQS challenger for signatures and returns the result. We consider the output $\mathcal{T} = (\mathsf{msg}_i, \mathsf{tok}_i)_{i\in[\mathsf{T}+1]}$ that $\mathcal{A}$ generates. We write $\mathsf{tok}_i = (\vec{\mathsf{M}}_i', \sigma_i)$. At the end, $\mathcal{B}_{\mathsf{EQS}}$ does the following:

– If $\mathcal{A}$ produced a Type-1 forgery, it uniformly at random chooses $i^* \in [\mathsf{T} + 1]$ and return the i^*-th token $\mathsf{tok}_{i^*} = (\vec{\mathsf{M}}_{i^*}', \sigma_{i^*})$ to the EQS challenger as its forgery.
– If $\mathcal{A}$ produced a Type-2 forgery with set $S \subset [\mathsf{T}]$. Assume without loss of generality that $|S| = \mathsf{R}+1$. $\mathcal{B}_{\mathsf{EQS}}$ select uniformly at random $i^* \in S$ and outputs the i^*-th token $\mathsf{tok}_{i^*} = (\vec{\mathsf{M}}_{i^*}', \sigma_{i^*})$ to the EQS challenger as its forgery.

Now, let us construct $\mathcal{B}_{\mathsf{ZK}}$. $\mathcal{B}_{\mathsf{ZK}}$ runs the ZK challenger from the soundness game and get the CRS (if any). Then it simulates perfectly the EQS game to $\mathcal{A}$. At the end, $\mathcal{B}_{\mathsf{ZK}}$ does the following:

– If $\mathcal{A}$ produced a Type-1 forgery, it outputs $\perp$ to the ZK soundess challenger.
– If $\mathcal{A}$ produced a Type-2 forgery with set $S \subset [\mathsf{T}]$. Assume without loss of generality that $|S| = \mathsf{R}+1$. $\mathcal{B}_{\mathsf{ZK}}$ select uniformly at random $j^* \in [\mathsf{T}]$ and outputs the j^*-th query $(\vec{\mathsf{M}}_{j^*}, \pi_{\vec{\mathsf{M}}_{j^*}})$ made by $\mathcal{A}$ to $\mathsf{ACT.Sign}$ to the ZK challenger (as a proof of an invalid statement).

Remark that $\mathcal{B}_{\mathsf{EQS}}$ and $\mathcal{B}_{\mathsf{ZK}}$ cannot know (in polynomial time) whether the output to their EQS/ZK challenger is valid, i.e., if they will win the $\mathrm{EQS}_{\mathcal{B}_{\mathsf{EQS}}}(\lambda)$ and $\mathrm{SND}_{\mathcal{B}_{\mathsf{ZK}}}(\lambda)$ games. However, this is not an issue to prove that the advantage of at least one of those two adversaries is non-negligible if the advantage of $\mathcal{A}$ in $\mathrm{OMUF}_{\mathcal{A}}(\lambda)$ is non-negligible.

Our proof strategy is the following. We define events $\mathsf{E}_{1\mathsf{EQS},i}$, $\mathsf{E}_{2\mathsf{EQS},k}$, $\mathsf{E}_{2\mathsf{ZK},j}$ for $i \in [\mathsf{T}+1]$, $k \in [\mathsf{R}]$, $j \in [\mathsf{T}]$ so that:

– Exactly one of these events must happen if the adversary wins.
– Conditioned on any of these events, either $\mathcal{B}_{\mathsf{EQS}}$ or $\mathcal{B}_{\mathsf{ZK}}$ wins EUF-CMA or soundness respectively.

Note that we do not need that these events can efficiently be checked for the proof to go through.

Definition of Events. Let us first define events $\mathsf{E}_{1\mathsf{EQS},i}$ that are associated with a Type-1 forgery In that case, $\mathcal{A}$ generated $\mathsf{T}+1$ token by doing T signature queries. Vectors $\vec{\mathsf{M}}'_i$ in tokens tok_i are of the form $(\mathsf{G}_1, \mathsf{F}_i, \mathsf{E}_i)$ and are such that all pairs $(\mathsf{F}_i, \mathsf{E}_i)$ are distinct. Therefore, vectors $\vec{\mathsf{M}}'_i$ and $\vec{\mathsf{M}}'_j$ from two different tokens tok_i and tok_j cannot be in the span of the same element $\vec{\mathsf{M}} \in \mathsf{G}_1^3$ Since $\mathcal{A}$ queries Sign only T times and there are $\mathsf{T}+1$ valid tokens tok_i, there exists $i \in [\mathsf{T}+1]$ so that $\vec{\mathsf{M}}'_i$ is not in the span of any $\vec{\mathsf{M}}$ queried to Sign. And $\mathsf{tok}_i = (\vec{\mathsf{M}}'_i, \sigma_i)$ is a forgery for the EQS scheme.

We define $\mathsf{E}_{1\mathsf{EQS},i}$ the event that $\mathcal{A}$ produced a Type-1 forgery and $\vec{\mathsf{M}}'_i$ is not in the span of any $\vec{\mathsf{M}}$ queried to Sign, and no event $\mathsf{E}_{1\mathsf{EQS},i'}$ happened for $i' < i$. (This last constraint is to ensure that the events $\mathsf{E}_{1\mathsf{EQS},i}$ are disjoint.) Equivalently, $\mathsf{E}_{1\mathsf{EQS},i}$ is defined as the event that $\mathcal{A}$ produced a Type-1 forgery and $\vec{\mathsf{M}}'_i$ if the first of the vectors $\left\{\vec{\mathsf{M}}'_{i'}\right\}_{i'}$ that is not in the span of any $\vec{\mathsf{M}}$ queried to Sign.

Let us now consider Type-2 forgeries. Let us show that at least one of the following events must happen:

– Event $\mathsf{E}_{2\mathsf{EQS},k}$: $\mathcal{A}$ made a Type-2 forgery (and not a Type-1 forgery) for a set $S = \{i_1, \ldots, i_{\mathsf{R}}\}$ with $i_1 < \cdots < i_{\mathsf{R}}$, so that $\vec{\mathsf{M}}'_{i_k}$ is not in the span of any $\vec{\mathsf{M}}$ queried to Sign, and so that no event $\mathsf{E}_{2\mathsf{EQS},k'}$ happened for $k' \le k$.
– Event $\mathsf{E}_{2\mathsf{ZK},i}$:
  • $\mathcal{A}$ made a Type-2 forgery (and not a Type-1 forgery) and
  • its i-th signing query is $(\vec{\mathsf{M}}_i, \mathsf{G}_1, \pi_{\vec{\mathsf{M}}_i})$ where $\pi_{\vec{\mathsf{M}}_i}$ is valid but proves a wrong statement, that is $\vec{\mathsf{M}}_i \neq (\mu^{-1}\mathsf{G}_1,\, \mu^{-1}(\mathsf{u}+\mathsf{msg})^{-1}\mathsf{G}_1,\, \mu^{-1}\mathsf{msg})$ for u such that the client public parameter is $\mathsf{U} = \mathsf{u}\mathsf{G}_1$ and
  • no event $\mathsf{E}_{2\mathsf{EQS},k}$ nor $\mathsf{E}_{2\mathsf{ZK},j}$ happened for $j' < j$ and any k.

Let us assume by contradiction that none of the events happened (assuming $\mathcal{A}$ made a Type-2 forgery). Then since no $\mathsf{E}_{2\mathsf{EQS},k}$ happened, each $\vec{\mathsf{M}}'_{i_k}$ is in the span of some queried message $\vec{\mathsf{M}}_{j_k}$ (in the j_k-th query to Sign). As before, we can prove that j_k are necessarily all distinct. Since there are $\mathsf{R}+1$ of them and there are only R clients, at least two of them (for $k = \alpha$ and β) were made for the same client $\mathsf{U}^* = \mathsf{u}^*\mathsf{G}_1$. We can write these two queries as follows for some $\mu_1, \mu_2, \zeta_1, \zeta_2 \in \mathbb{Z}_p$

$$\vec{\mathsf{M}}_{j_\alpha} = \mu_1 \vec{\mathsf{M}}'_{k_\alpha} = \mu_1(\mathsf{G}_1, \zeta_1 \cdot \mathsf{G}_1, \mathsf{msg} \cdot \mathsf{G}_1) \tag{6}$$

$$\vec{\mathsf{M}}_{j_\beta} = \mu_2 \vec{\mathsf{M}}'_{k_\beta} = \mu_2(\mathsf{G}_1, \zeta_2 \cdot \mathsf{G}_1, \mathsf{msg} \cdot \mathsf{G}_1), \tag{7}$$

where msg is the message on which the Type-2 forgery was made. Since tags must be different, we have $\zeta_1 \neq \zeta_2$. But since the zero-knowledge proofs were all proving valid statements (as no event $\mathsf{E}_{2\mathsf{sk},i}$ occurred), we also have that:

$$\zeta_1 = (\mathsf{msg} + \mathsf{u}^*)^{-1} = \zeta_2.$$

This is a contradiction. So at least one of the events must happen.

Conclusion of the Proof. Since at least one of the events $\mathsf{E}_{1,i}$, $\mathsf{E}_{2\mathsf{EQS},k}$, $\mathsf{E}_{2\mathsf{ZK},i}$ must happen and these events are disjoint, we have that from the total probability rule:

$$\begin{aligned}
&\mathsf{Adv}^{\text{omuf}}_{\mathsf{ACT},\mathcal{A}}(\lambda) = \Pr[\text{OMUF}_{\mathcal{A}}(\lambda) = 1] \\
&= \sum_{i=1}^{\mathsf{T}+1} \Pr[\text{OMUF}_{\mathcal{A}}(\lambda) = 1 \text{ and } \mathsf{E}_{1\mathsf{EQS},i}] + \sum_{k=1}^{\mathsf{R}} \Pr[\text{OMUF}_{\mathcal{A}}(\lambda) = 1 \text{ and } \mathsf{E}_{2\mathsf{EQS},j}] \\
&\quad + \sum_{j=1}^{\mathsf{T}} \Pr[\text{OMUF}_{\mathcal{A}}(\lambda) = 1 \text{ and } \mathsf{E}_{2\mathsf{EQS},j}]
\end{aligned}$$

In addition, looking at the definition of $\mathcal{B}_{\mathsf{EQS}}$ and from the fact this reduction perfectly simulates the oracles for $\mathcal{A}$:

$$\begin{aligned}
&\sum_{i=1}^{\mathsf{T}+1} \Pr[\text{OMUF}_{\mathcal{A}}(\lambda) = 1 \text{ and } \mathsf{E}_{1\mathsf{EQS},i}] \\
&= \sum_{i=1}^{\mathsf{T}+1} \Pr\Big[\text{EUF-CMA}_{\mathcal{B}_{\mathsf{EQS}}}(\lambda) = 1 \text{ and } \mathsf{E}_{1\mathsf{EQS},i} \,\Big|\, i^* = i\Big] \\
&= \Pr[i^* = i]^{-1} \sum_{i=1}^{\mathsf{T}+1} \Pr\Big[\text{EUF-CMA}_{\mathcal{B}_{\mathsf{EQS}}}(\lambda) = 1 \text{ and } \mathsf{E}_{1\mathsf{EQS},i} \text{ and } i^* = i\Big] \\
&\leq \Pr[i^* = i]^{-1} \sum_{i=1}^{\mathsf{T}+1} \Pr\Big[\text{EUF-CMA}_{\mathcal{B}_{\mathsf{EQS}}}(\lambda) = 1 \text{ and } i^* = i\Big] \\
&= (\mathsf{T}+1) \cdot \Pr\Big[\text{EUF-CMA}_{\mathcal{B}_{\mathsf{EQS}}}(\lambda) = 1\Big].
\end{aligned}$$

Similarly, we have:

$$\begin{aligned}
&\sum_{k=1}^{\mathsf{R}} \Pr[\text{OMUF}_{\mathcal{A}}(\lambda) = 1 \text{ and } \mathsf{E}_{2\mathsf{EQS},k}] \leq \mathsf{R} \cdot \Pr\Big[\text{EUF-CMA}_{\mathcal{B}_{\mathsf{EQS}}}(\lambda) = 1\Big] \\
&\sum_{j=1}^{\mathsf{T}} \Pr[\text{OMUF}_{\mathcal{A}}(\lambda) = 1 \text{ and } \mathsf{E}_{2\mathsf{ZK},j}] \leq \mathsf{T} \cdot \Pr\big[\text{SND}_{\mathcal{B}_{\mathsf{ZK}}}(\lambda) = 1\big]
\end{aligned}$$

Together this yields Eq. (5).

□

Unlinkability.

Theorem 6.2. *If* EQS *provides signature adaption,* ZK *is a zero-knowledge ZK argument, the Dodis-Yampolskiy* $\mathcal{F}_{\mathsf{DY}}(\mathsf{u}, \mathsf{msg}) = (1/(\mathsf{u} + \mathsf{msg})) \cdot \mathsf{G}_1$ *is pseudorandom even when* uG_1 *is public, and the DDH assumption holds in* $\mathbb{G}_1$ *(which is implied by the SXDH assumption), then the* ACT *scheme in Construction 5.1 satisfies unlinkability from Definition 3.3.*

Recall from Section 2.2 that Dodis-Yampolskiy is pseudorandom under the 2^α-DDHI assumption in $\mathbb{G}_1$ when the input messages come from a subset of size 2^α of $\mathbb{Z}_p^*$.

Proof. We do the proof using hybrid games:

Hyb_0 This hybrid is the unlinkability security game.

Hyb_1 This hybrid is the same as the previous one with the following syntactic change: in any token request (whether to $\mathsf{Request}$ oracle or $\mathsf{Chl}_{\mathsf{issue}}$ oracle), $\vec{\mathsf{M}}$ is computed as $\vec{\mathsf{M}} = \mu^{-1}\vec{\mathsf{M}}'$ where:

$$\vec{\mathsf{M}}' = (\mathsf{M}'_1,\, \mathsf{M}'_2,\, \mathsf{M}'_3) = (\mathsf{G}_1,\, \mathcal{F}_{\mathsf{DY}}(\mathsf{u}_{\mathsf{C}}, \mathsf{msg}),\, \mathsf{msg} \cdot \mathsf{G}_1)$$

And in any associated $\mathsf{Unblind}$ computation (via a query to $\mathsf{Chl}_{\mathsf{redeem}}$), the same $\vec{\mathsf{M}}'$ as the one used in the token request is used instead of re-computing $\vec{\mathsf{M}}'$.

Hyb_2 This hybrid is similar to the previous one except now all the ZK CRS and all ZK proofs are simulated. This hybrid is computationally indistinguishable from the previous one thanks to the zero-knowledge property of the ZK argument.

Hyb_3 This hybrid is similar to the previous one except that for every token request and associated unblind query to $\mathsf{Chl}_{\mathsf{redeem}}$ (if any), M'_2 is chosen uniformly at random. This hybrid is indistinguishable from the previous one assuming $\mathcal{F}_{\mathsf{DY}}$ is pseudorandom.

Hyb_4 This hybrid is similar to the previous one except that in any $\mathsf{Unblind}$ computation, instead of computing σ as:

$$\sigma \leftarrow \mathsf{EQS.Adapt}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}, \rho, \mu)$$

it is computed as:

$$\rho' \leftarrow \mathsf{EQS.Sign}(\mathsf{crs}, \mathsf{sk}, \vec{\mathsf{M}}'), \quad \sigma \leftarrow \mathsf{EQS.Adapt}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}', \rho', 1)$$

Remember that in the unlinkability game, $\mathsf{privprm}_{\mathsf{S}}$ is chosen by the challenger so the challenger knows $\mathsf{privprm}_{\mathsf{S}} = \mathsf{sk}$.

This hybrid is computationally indistinguishable from the previous one under signature adaptation of the EQS scheme.

Given another generator G''_1 of $\mathbb{G}_1$, we can write:

$$\vec{\mathsf{M}}' = (\mathsf{G}_1,\, \beta\mathsf{G}''_1,\, \mathsf{msg}\mathsf{G}'_1)$$

where we set $\mathsf{M}'_2 = \beta\mathsf{G}''_1$ as a uniformly random element from $\mathbb{G}_1$. We can then write

$$\vec{\mathsf{M}} = (\nu\mathsf{G}_1,\, \beta \cdot (\nu \cdot \mathsf{G}''_1),\, \mathsf{msg} \cdot (\nu \cdot \mathsf{G}'_1)) \tag{8}$$

where $\nu = \mu^{-1}$ is uniformly random in $\mathbb{Z}_p^*$ and different for each token request. Remark indeed that in this hybrid μ is only used once per token request when setting $\vec{\mathsf{M}} = \mu^{-1}\vec{\mathsf{M}}'$.

Hyb_5 This hybrid is similar to the previous one except that in any token request computation, $\vec{\mathsf{M}}$ is now chosen uniformly at random from $\mathbb{G}_1^3$ instead of as specified in Eq. (8).

This hybrid is computationally indistinguishable from the previous one under the DDH assumption in $\mathbb{G}_1$. Indeed, the DDH assumption can be used to show that $(\nu\mathsf{G}_1,\, \nu \cdot \mathsf{G}''_1,\, \nu \cdot \mathsf{G}'_1)$ are indistinguishable from random group elements in $\mathbb{G}_1^3$.

In the last hybrid, the output of any token request is independent of the inputs msg and u_{C}. The output of an unblind request only includes the underlying message and is independent of the output of the associated token request: no element from the computation of the token request is used when unblinding in the last hybrid except the message. Thus from the adversary point of view, in the last hybrid $\mathsf{b}_{\mathsf{chl}}$ is uniformly random and the adversary's advantage is 0. This concludes the proof. □

Parameter Generation for Unlinkability. Unlinkability assumes that the ACT parameters are generated honestly and the issuer only sees the secret key. To alleviate this requirement, we can instead work in the CRS model, where the ZK and EQS CRS are put in the CRS and generated by a trusted party. The EQS signature adaptation holds even for maliciously generated pk, but our unlinkability proof uses sk. Therefore, we would also need that the issuer includes in $\mathsf{pprm}_{\mathsf{S}}$ a zero-knowledge proof of knowledge of the EQS secret key sk for the unlinkability proof above to go through. This can be done efficiently using either Groth-Sahai proofs or Fiat-Shamir proofs.

The same discussion applies to Construction 6.3.

Instantiation. The construction in Construction 5.1 can be instantiated in the standard model using the EQS from Section 6.1. in the full version of the paper [BRS23] and using Groth-Sahai proofs for the ZK argument. Following the notations from [EG14], the Groth-Sahai proof will need to make (with the cost in the number of group elements):

– $3 \times \mathsf{sca}_{\mathbb{G}_2}$ for the commitments to μ^{-1}, $\mathsf{u_C}$, and msg
  $\looparrowright$ cost $= 6 \times \mathbb{G}_2$
– $4 \times \mathsf{MConst}_{G_2}$ to prove the following 4 equations:

$$\mathsf{M}'_1 \bullet \mathsf{G}_2 = \mathsf{G}_1 \bullet (\mu^{-1} \cdot \mathsf{G}_2) \qquad \mathsf{M}'_2 \bullet (\mathsf{u_C} \cdot \mathsf{G}_2) + \mathsf{M}'_2 \bullet (\mathsf{msg} \cdot \mathsf{G}_2) = \mathsf{G}_1 \bullet (\mu^{-1} \cdot \mathsf{G}_2)$$
$$\mathsf{M}'_3 \bullet \mathsf{G}_2 = \mathsf{M}'_1 \bullet (\mathsf{msg} \cdot \mathsf{G}_2) \qquad \mathsf{U_C} \bullet \mathsf{G}_2 = \mathsf{G}_1 \bullet (\mathsf{u_C} \cdot \mathsf{G}_2)$$

  $\looparrowright$ cost $= 4 \times \mathbb{G}_1$

6.1 ACT from EQS and a Random-Oracle-Based PRF

The second ACT construction from EQS leverages the PRF in the RO model $\mathsf{PRF}(\mathsf{K}, x) = \mathsf{K} \cdot \mathsf{H}(x)$ where $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$ is a hash function that can be modeled as a random oracle. This PRF was folklore and has been formally defined in [NPR99].

The ACT construction is similar to Construction 5.1 but the message $\vec{\mathsf{M}}$ generated by $\mathsf{ACT.Request}$ is generated as:

$$\vec{\mathsf{M}} \leftarrow \big(\mathsf{M}_1 = \mu^{-1} \cdot \mathsf{G}_1,\ \mathsf{M}_2 = \mu^{-1} \cdot \mathsf{u_C} \cdot \mathsf{H}(\mathsf{msg}),\ \mathsf{M}_3 = \mu^{-1} \cdot \mathsf{H}(\mathsf{msg})\big)$$

instead of

$$\vec{\mathsf{M}} \leftarrow \big(\mathsf{M}_1 = \mu^{-1} \cdot \mathsf{G}_1,\ \mathsf{M}_2 = \mu^{-1} \cdot (\mathsf{u_C} + \mathsf{msg})^{-1} \cdot \mathsf{G}_1,\ \mathsf{M}_3 = (\mu^{-1}\mathsf{msg}) \cdot \mathsf{G}_1\big).$$

Note that in both cases $\mathsf{M}_2 = \mathsf{PRF}(\mathsf{u_C}, \mathsf{msg})$, but with a different PRF. Importantly the statements proven by the ZK proof in this new construction do not need to evaluate the hash function H, so they can be still efficiently instantiated with Fiat-Shamir or Groth-Sahai.

We present two versions: the long version uses dimension-3 vectors $\vec{\mathsf{M}}$ and includes M_1 (like the Dodis-Yampolkiy-based construction), while the short version uses dimension-2 vectors without M_1. The short version is not proven unforgeable but we show that a specific instantiation of it is unforgeable in Section 8, in the generic (bilinear) group model.

Construction 6.3 (ACT from EQS and Random Oracle (long and short versions)). Let ($\mathsf{EQS.Setup}$, $\mathsf{EQS.KGen}$, $\mathsf{EQS.Sign}$, $\mathsf{EQS.Adapt}$, $\mathsf{EQS.Verify}$) be an equivalence-class signature scheme. An anonymous counting token construction ACT consists of the following algorithms:

ACT.GenParam(1^λ): Generate

1. a bilinear group $\mathcal{PG} \leftarrow \mathsf{GGen}(1^\lambda)$,
2. ZK argument CRS $\mathsf{ZK.crs} \leftarrow \mathsf{ZK.Setup}(\mathcal{R})$ where $\mathcal{R}$ is implicitly define in $\mathsf{ACT.Request}$ ($\mathsf{ZK.crs}$ is used implicitly when generating and verifying ZK proofs),
3. EQS CRS $\mathsf{crs} \leftarrow \mathsf{EQS.Setup}(\mathcal{PG})$, and EQS keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{crs})$.

Output: $\mathsf{pprm_S} \leftarrow (\mathsf{ZK.crs}, \mathsf{crs}, \mathsf{pk})$
$\mathsf{privprm_S} \leftarrow (\mathsf{pprm_S}, \mathsf{sk})$.

ACT.ClientRegister$(\mathsf{pprm_S})$:

1. Generate a PRF key $\mathsf{u_C} \leftarrow\!\!\$\ \mathbb{Z}_p$.

2. Set $\mathsf{U_C} = \mathsf{u_C} \cdot \mathsf{G}_1$.

Output: $\mathsf{pprm_C} \leftarrow \mathsf{U_C}$
$\mathsf{privprm_C} \leftarrow \mathsf{u_C}$

ACT.TokenRequest$(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{msg})$:

1. Generate a random value $\mu \leftarrow\!\!\$\; \mathbb{Z}_p^*$.
2. Compute $\mathsf{M}'_3 \leftarrow \mathsf{H}(\mathsf{msg})$ and $\mathsf{M}'_2 \leftarrow \mathsf{u_C} \cdot \mathsf{H}(\mathsf{msg})$.
3. Set $\vec{\mathsf{M}} \leftarrow (\mathsf{M}_1 = \mu^{-1} \cdot \mathsf{G}_1,\ \mathsf{M}_2 = \mu^{-1} \cdot \mathsf{M}'_2,\ \mathsf{M}_3 = \mu^{-1} \cdot \mathsf{M}'_3)$.
4. Generate a proof $\pi_{\vec{\mathsf{M}}}$:

$$\pi_{\vec{\mathsf{M}}} : \mathsf{ZK}\big\{\exists \mathsf{u}_C \in \mathbb{Z}_p : \mathsf{M}_2 = \mathsf{u_C} \cdot \mathsf{M}_3 \textbf{ and } \mathsf{U_C} = \mathsf{u_C} \cdot \mathsf{G}_1)\big\}$$

Output: $\mathsf{blindRequest} \leftarrow (\vec{\mathsf{M}}, \pi_{\vec{\mathsf{M}}})$
$\mathsf{rand_{msg}} \leftarrow (\mathsf{msg}, \vec{\mathsf{M}}, \mu)$

ACT.Sign$(\mathsf{privprm_S}, \mathsf{pprm_C}, \mathsf{blindRequest})$

1. Parse $\mathsf{blindRequest} = (\vec{\mathsf{M}}, \pi_{\vec{\mathsf{M}}})$.
2. If verification of $\pi_{\vec{\mathsf{M}}}$ fails, abort.
3. Run $\rho \leftarrow \mathsf{EQS.Sign}(\mathsf{crs}, \mathsf{sk}, \vec{\mathsf{M}})$.

Output: $\mathsf{blindToken} \leftarrow \rho$

ACT.Unblind$(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{blindToken}, \mathsf{rand_{msg}})$

1. Parse $\mathsf{rand_{msg}} = (\mathsf{msg}, \vec{\mathsf{M}}, \mu)$, $\mathsf{blindToken} = \rho$.
2. Set $\vec{\mathsf{M}}' \leftarrow (\mathsf{G}_1, \mathsf{u_C} \cdot \mathsf{H}(\mathsf{msg}),\ \mathsf{H}(\mathsf{msg}))$.
3. Compute $\sigma \leftarrow \mathsf{EQS.Adapt}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}, \rho, \mu)$.
4. If $\sigma = \perp$, abort.

Output: $\Big(\mathsf{msg},\ \mathsf{tok} \leftarrow (\vec{\mathsf{M}}', \sigma)\Big)$

ACT.Verify$(\mathsf{pprm_S}, \mathsf{privprm}, \mathsf{msg}, \mathsf{tok})$

1. Set $\mathsf{bit} \leftarrow$ **true**.
2. Parse $\mathsf{tok} = (\vec{\mathsf{M}}' = (\mathsf{M}'_1, \mathsf{M}'_2, \mathsf{M}'_3), \sigma)$.
3. If $\mathsf{M}'_1 \neq \mathsf{G}_1$ or $\mathsf{M}'_3 \neq \mathsf{H}(\mathsf{msg})$, set $\mathsf{bit} \leftarrow$ **false**.
4. If **false** $= \mathsf{EQS.Verify}(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}', \sigma)$, set $\mathsf{bit} \leftarrow$ **false**.

Output: $(\mathsf{bit},\ \mathsf{M}_2)$

Perfect correctness is straightforward.

Unforgeability of the Long version.

Theorem 6.4. *If* EQS *provides EUF-CMA security,* ZK *is a sound ZK argument, and* H *is a collision-resistant hash function, then the long version of the* ACT *scheme in Construction 6.3 satisfies unforgeability from Definition 3.2.*

Note that unforgeability does not rely on the pseudorandomness and does not model H as a random oracle but just requires H to be collision resistant. On the other hand, the theorem only holds for the long version and not the short version.

Proof. We do a proof by an hybrid argument:

Hyb_0 This hybrid corresponds to the unforgeability game.

Hyb_1 This hybrid is similar to the first hybrid except the challenger aborts if the adversary produces two tokens for two messages $\mathsf{msg} \neq \mathsf{msg}'$ so that $\mathsf{H}(\mathsf{msg}) = \mathsf{H}(\mathsf{msg}')$.

The two games are indistinguishable by collision resistance of H.

Now, a similar proof to the one for Theorem 6.1 can be done to reduce EUF-CMA security of EQS and soundness of the ZK argument to Hyb_1. Concretely, the only two changes are in the argument that, at least one of the $\mathsf{E}_{1\mathsf{EQS},i}$, $\mathsf{E}_{2\mathsf{EQS},k}$ or $\mathsf{E}_{2\mathsf{ZK},j}$ must happen (recall those events from the proof of Theorem 6.1). Assuming a Type-1 forgery, one of the $\mathsf{E}_{1,i}$ must happen since we ensured that all hashes $\mathsf{H}(\mathsf{msg})$ are distinct and then the argument is the same as in Theorem 6.1. For the Type-2 forgery events, if none happen, we have a similar arguments as in the previous proof, except that Eqs. (6) and (7) are replaced by: there $\mu_1, \mu_2, \mathsf{Z}_1, \mathsf{Z}_2, \mathsf{M}'_3 \in \mathbb{Z}_p$

$$\vec{\mathsf{M}}_{j_\alpha} = \mu_1 \vec{\mathsf{M}}'_{k_\alpha} = \mu_1(\mathsf{G}_1,\, \mathsf{Z}_1,\, \mathsf{M}'_3)$$
$$\vec{\mathsf{M}}_{j_\beta} = \mu_2 \vec{\mathsf{M}}'_{k_\beta} = \mu_2(\mathsf{G}_1,\, \mathsf{Z}_2,\, \mathsf{M}'_3),$$

(where $\mathsf{M}'_3 = \mathsf{H}(\mathsf{msg})$) and since tags are different $\mathsf{Z}_1 \neq \mathsf{Z}_2$. On the other hand, the zero-knowledge proof are assumed at this point to be proving valid statement which ensures that:

$$\mathsf{Z}_1 = \mathsf{u}^* \cdot \mathsf{M}'_3 = \mathsf{Z}_2$$

which allows to conclude the proof as before. □

Unlinkability.

Theorem 6.5. *If* EQS *provides signature adaption,* ZK *is a zero-knowledge ZK argument, and the DDH assumption holds in* $\mathbb{G}_1$ *(which is implied by the SXDH assumption), then the* ACT *scheme in Construction 5.1 satisfies unlinkability from Definition 3.3, when* H *is modeled as a random oracle.*

Contrary to the construction Construction 5.1, we do not require messages to be short for unlinkability to hold as $\mathsf{PRF} : (\mathsf{u}, \mathsf{msg}) \mapsto \mathsf{u} \cdot \mathsf{H}(\mathsf{msg})$ is pseudorandom for any the message input set $\{0,1\}^*$, assuming the DDH assumption in $\mathbb{G}_1$ and when H is modelled as a random oracle (even when $\mathsf{u} \cdot \mathsf{G}_1$ is public).

The proof of pseudorandomness of PRF is done by a direct reduction to DDH: given a tuple $(\mathsf{G}_1, X, Y, Z) \in \mathsf{G}_1^4$ that is either a Diffie-Hellman tuple or a random tuple, set $\mathsf{u} \cdot \mathbb{G}_1 = X$. For each query msg to H, generate a fresh tuple $(\mathsf{G}_1, X, Y_{\mathsf{msg}}, Z_{\mathsf{msg}})$ using random self-reducibility of DDH (so that $(Y_{\mathsf{msg}}, Z_{\mathsf{msg}})$ is a fresh random Diffie-Hellman pair with regards to (G_1, X) if (G_1, X, Y, Z) was a Diffie-Hellman tuple, and is uniformly random otherwise). Then set $\mathsf{H}(\mathsf{msg}) = Y_{\mathsf{msg}}$. Now for any query $\mathsf{PRF}(\mathsf{u}, \mathsf{msg})$, simulate a query to $\mathsf{H}(\mathsf{msg})$ if msg was not queried to H, and return Z_{msg}. When (G_1, X, Y, Z) is a Diffie-Hellman tuple, the PRF is evaluated properly while when it is a random tuple, the queries $\mathsf{PRF}(\mathsf{u}, \mathsf{msg})$ are all answered uniformly at random.

Proof. The proof is similar to the one of Theorem 6.2. The main difference is that the Dodis-Yampolkiy PRF is replaced by the PRF $\mathsf{PRF} : (\mathsf{u}, \mathsf{msg}) \mapsto \mathsf{u} \cdot \mathsf{H}(\mathsf{msg})$ that is proven above pseudorandom even when $\mathsf{u} \cdot \mathsf{G}_1$ is public. □

Instantiation. The long version of Construction 6.3 can be instantiated in the random oracle model using the EQS from Section 6.1. in the full version of the paper [BRS23] and using a Fiat-Shamir proof for the ZK argument.

Concretely, the Fiat-Shamir proof needs to prove that $\log_{\mathsf{G}_1}(\mathsf{U}_\mathsf{C}) = \log_{\mathsf{M}_3}(\mathsf{M}_2)$ and consists of the following:

- Generate random scalar $v \leftarrow\$ \mathbb{Z}_p^*$.
- Compute the Fiat-Shamir commitments: $V \leftarrow v \cdot \mathsf{G}_1$, $W \leftarrow v \cdot \mathsf{M}_3$.
- Derive the Fiat-Shamir challenge: $c \leftarrow \mathsf{H}'(\mathsf{U}_\mathsf{C}, \vec{\mathsf{M}}, V, W)$, where $\mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_p$ is a hash function that will be modeled as a random oracle.
- Compute the Fiat-Shamir response: $\xi \leftarrow v + c \cdot \mathsf{u}_\mathsf{C}$.
- Set the proof to be: $\pi_{\vec{\mathsf{M}}} \leftarrow (\xi, c) \in \mathbb{Z}_p^2$.[7]

The proof is verified by computing $V' \leftarrow \xi \cdot \mathsf{G}_1 - c \cdot \mathsf{U}_\mathsf{C}$ and $W' \leftarrow \xi \cdot \mathsf{M}_3 - c \cdot \mathsf{M}_2$, and then checking whether $c = \mathsf{H}'(\mathsf{U}_\mathsf{C}, \vec{\mathsf{M}}, V', W')$.

7 Equivalent-Class Signature Constructions

In this section, we present two EQS constructions from bilinear maps: one with security from any signature scheme and ZK argument of knowledge, and one in the generic bilinear maps model, which achieves better concrete efficiency. The former construction can be efficiently instantiated in the standard model under the SXDH assumption.

7.1 EQS in the Standard Model

We formally provide a generic construction of an EQS scheme from any signature scheme and ZK argument. The idea of the construction was already present for example in the introduction of [FG18].

Construction 7.1. Let $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ be a (classical) existentially unforgeable (EUF-CMA) signature scheme with message space $\mathbb{G}_1^\ell$ and ZK be a ZK argument *of knowledge* system.

EQS.Setup(1^λ): Generate

1. ZK parameters $\mathsf{crs} \leftarrow \mathsf{ZK.Setup}(\mathcal{R})$ for the relation $\mathcal{R}$ defined implicitly below in $\mathsf{EQS.Sign}$ (crs is used implicitly when generating and verifying ZK proofs)

Output: crs

EQS.KGen(crs):

Output: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$

EQS.Sign$(\mathsf{crs}, \mathsf{sk}, \vec{\mathsf{M}} \in \mathbb{G}_1^\ell)$:

Output: $\rho \leftarrow \mathsf{Sign}(\mathsf{sk}, \vec{\mathsf{M}})$

EQS.Adapt$(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}, \rho, \mu)$:

[7] Actually the challenge c can be reduced to λ bits while keeping the security of the Fiat-Shamir transform.

1. Immediately return $\perp$ if $\mathbf{false} = \mathsf{Verify}(\mathsf{pk}, \vec{\mathsf{M}}, \rho)$.
2. Set $\Upsilon \leftarrow \mu \mathbb{G}_2$ and $\vec{\mathsf{M}}' \leftarrow \mu \cdot \vec{\mathsf{M}}$.
3. Generate the ZK proof:

$$\pi : \mathsf{ZK}\big\{\exists\!\!\!\exists\, \vec{\mathsf{M}} \in \mathbb{G}_1^\ell,\ \exists\!\!\!\exists\, \Upsilon \in \mathbb{G}_2,\ \exists\!\!\!\exists\, \rho : \\ \mathbf{true} = \mathsf{Verify}(\mathsf{pk}, \vec{\mathsf{M}}, \rho) \textbf{ and } \vec{\mathsf{M}}' \bullet \mathsf{G}_2 = \vec{\mathsf{M}} \bullet \Upsilon\big\}$$

Output: $\sigma \leftarrow \pi$

EQS.Verify$(\mathsf{crs}, \mathsf{pk}, \vec{\mathsf{M}}', \sigma)$: runs the ZK verification algorithm on σ.

The reason why we prove knowledge of $\Upsilon = \mu \mathsf{G}_2$ instead of directly μ is that Groth-Sahai proofs allow to prove efficiently knowledge of group elements but not knowledge of scalar (which would require a bit-by-bit approach or something similar, which is much less efficient). Note that an argument of knowledge of μ could be used instead if the ZK proof system is more efficient this way, since knowledge of $\mu \in \mathbb{Z}_p$ implies knowledge of $\Upsilon \in \mathbb{G}_2$.

EUF-CMA.

Theorem 7.2. *If* $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ *is an EUF-CMA signature scheme and* ZK *is a knowledge-sound ZK argument, then the* EQS *scheme in Construction 7.1 satisfies EUF-CMA.*

Proof. Let us call Condition $(\star)$ this part of the condition to win the EUF-CMA game:

$$\forall \vec{\mathsf{M}} \in \mathcal{Q}_{\mathsf{Sign}},\ \vec{\mathsf{M}}'^* \notin \mathsf{Span}(\vec{\mathsf{M}}),$$

where $\vec{\mathsf{M}}^*$ is the message from the forgery.

We do a proof by hybrid games:

Hyb_0 This hybrid corresponds to the EQS EUF-CMA game EUF-CMA$_{\mathcal{A}}(\lambda)$, defined in Fig. 1. Note that the game is not polynomial time: Condition $(\star)$ may not be checked in polynomial time without knowing the discrete logarithm of the forged message (or of the signed messages).

Hyb_1 This hybrid is similar to the previous hybrid except the ZK setup is done by the extractor from the knowledge soundness. This game is statistically indistinguishable from the previous one under setup indistinguishability from the knowledge soundness. Note that it is important here that setup indistinguishability is a statistical property as this hybrid (and the previous one) are not polynomial-time.

Hyb_2 This hybrid is similar to the previous hybrid except that after $\mathcal{A}$ outputs $(\vec{\mathsf{M}}^*, \sigma^*)$ and after verifying σ^* (but before checking Condition $(\star)$), the hybrid aborts (and return 0) if the extractor $\mathsf{Extract}$ cannot extract a valid witness out of $\sigma^* = \pi^*$.
This hybrid is computationally indistinguishable from the previous one under extractability of the ZK argument of knowledge, because the added abort event happens with negligible probability (and up to the abort even, the hybrid runs in polynomial time).
Let $\vec{\mathsf{M}}^* \in \mathbb{G}_1^\ell$, $\Upsilon^* \in \mathbb{G}_2$, ρ^* be the extracted witness.

Hyb_3 This hybrid is similar to the previous hybrid except before checking Condition $(\star)$, we abort if $\vec{\mathsf{M}}^*$ is one of the queries $\vec{\mathsf{M}} \in \mathcal{Q}_{\mathsf{Sign}}$.
This hybrid is perfectly indistinguishable from the previous one since the new abort is only triggered when Condition $(\star)$ does not hold. Indeed, validity of the extracted witness implies that $\vec{\mathsf{M}}'^* \bullet \mathsf{G}_2 = \vec{\mathsf{M}}^* \bullet \Upsilon^*$ which in turns implies that $\vec{\mathsf{M}}'^* \in \mathsf{Span}(\vec{\mathsf{M}}^*)$ and Condition $(\star)$ does not hold (for $\vec{\mathsf{M}} = \vec{\mathsf{M}}^* \in \mathcal{Q}_{\mathsf{Sign}}$).

Hyb_4 This hybrid is similar to the previous hybrid except we do not test Condition ($\star$) anymore. The winning probability of the adversary in this hybrid is at most the one in the previous hybrid.

We now reduce EUF-CMA of the underlying signature scheme to this last hybrid. The extracted ρ^* is a valid signature on $\vec{\mathsf{M}}^*$ that was never queried before.

This concludes the proof. $\square$

Signature Adaptation.

Theorem 7.3. *If* ZK *is a zero-knowledge argument, then the* EQS *scheme in Construction 7.1 satisfies signature adaptation.*

Proof. We do a proof by hybrid games:

Hyb_0 This hybrid corresponds to the EQS signature adaptation game $\text{SIG-ADP}_{\mathcal{A}}(\lambda)$, defined in Fig. 1.

Hyb_1 This hybrid is similar to the previous one except the two signatures σ_0 and σ_1 are now simulated using the ZK simulator (and crs is also simulated). In particular σ_0 and σ_1 are computed just from $\vec{\mathsf{M}}' = \mu\vec{\mathsf{M}}$ (and pk), without using ρ and ρ' provided by the adversary.
This hybrid is computationally indistinguishable from the previous one because ZK is zero-knowledge.

We conclude by remarking that σ_0 and σ_1 are computed exactly the same way in the last hybrid and the probability of the adversary guessing $\mathsf{b}_{\mathsf{chl}}$ is therefore exactly $1/2$. $\square$

Efficient Instantiation in the Standard Model. To efficiently instantiate the above generic construction, we need an efficient signature scheme and an efficient ZK proof system. For standard model instantiation, one solution is to use Groth-Sahai ZK arguments and structure-preserving signature (SPS) schemes. A structure-preserving signature scheme is such that verification consists in pairing-product equations so that it can be efficiently verified by Groth-Sahai arguments.

Concretely, we can use the Jutla-Roy SPS scheme from [JR17]. Signatures are vectors $\rho \in \mathbb{G}_1^5 \times \mathbb{G}_2$ (independent of ℓ) while verification is done using only two pairing-product equation. Following the notations from [EG14], the Groth-Sahai proof will need to make (with the cost in number of group elements):

- $5 \times \mathsf{com}_{\mathbb{G}_1} + 1 \times \mathsf{enc}_{\mathbb{G}_2}$ for the commitment to ρ
  $\looparrowright$ cost $= 10 \times \mathbb{G}_1 + 2 \times \mathbb{G}_2$
- $\ell \times \mathsf{com}_{\mathbb{G}_1}$ for the commitment of $\vec{\mathsf{M}}$
  $\looparrowright$ cost $= 2\ell \times \mathbb{G}_1 + 0 \times \mathbb{G}_2$
- $1 \times \mathsf{sca}_{\mathbb{G}_2}$ for the commitment of μ (which can be extracted as $\Upsilon = \mu\mathbb{G}_2$)
  $\looparrowright$ cost $= 0 \times \mathbb{G}_1 + 2 \times \mathbb{G}_2$
- $1 \times \mathsf{PConst}_{\mathbb{G}_2} + 1 \times \mathsf{PPE}$ proof of equations for the verification of ρ (the former is the left part of the verification equation in [JR17, Fig. 2] and the latter for the right part of it)
  $\looparrowright$ cost $= 4 \times \mathbb{G}_1 + 6 \times \mathbb{G}_2$
- $1 \times \mathsf{ME}_{\mathbb{G}_1}$ proof for the equation $\vec{\mathsf{M}}' \bullet \mathbb{G}_2 = \vec{\mathsf{M}} \bullet \Upsilon$
  $\looparrowright$ cost $= 2 \times \mathbb{G}_1 + 4 \times \mathbb{G}_2$

In total σ contains $(16 + 2\ell) \times \mathbb{G}_1 + 12 \times \mathbb{G}_2$ group elements.

7.2 EQS in the Generic Bilinear Group Model

Fuchsbauer, Hanser, and Slamanig constructed an efficient EQS in the generic group model in [FHS19]. This EQS satisfies our (weaker) security notions when applying the following minor change $\mathsf{EQS.Adapt}$ aborts if the signature does not verify.

8 ACTs in the Generic Bilinear Group Model

In this section, we show that the short variant of Construction 6.3 is unforgeable in the generic bilinear group model and random oracle model, when implemented with the EQS scheme from [FHS19] and Fiat-Shamir for equality of discrete logarithms for the ZK proof in $\mathsf{ACT.Sign}$ (as in the instantiation in Section 6.1). It achieves better concrete efficiency than all our other constructions at the cost of security holding in the generic bilinear group model (GBGM).

For the sake of completeness, we describe the full protocol below.

Construction 8.1 (ACT in GBGM). Let $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be a bilinear group, $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$ and $\mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_p$ be two hash functions that will be modeled as two random oracles. In this construction, we denote elements from $\mathbb{G}_2$ with a hat, e.g., $\hat{X}_2$, matching notation from [FHS19] (except the generators G_1 and G_2). Vectors $\vec{\mathsf{M}}$ have two elements indexed 2 and to match Construction 6.3. Recall that $\mathbb{G}_T$ is also written additively and that we use $\bullet$ to denote the pairing operation $e(\mathsf{X}, \hat{\mathsf{Y}}) = \mathsf{X} \bullet \hat{\mathsf{Y}}$. An anonymous counting token construction ACT consists of the following algorithms:

ACT.GenParam(1^λ): Generate

1. a bilinear group $\mathcal{PG} \leftarrow \mathsf{GGen}(1^\lambda)$
2. EQS secret key $\mathsf{sk} = (x_2, x_3) \leftarrow\$ (\mathbb{Z}_p^*)^2$
3. EQS public key $\mathsf{pk} \leftarrow (\hat{\mathsf{X}}_2 \leftarrow x_2 \cdot \mathsf{G}_2,\ \hat{\mathsf{X}}_3 \leftarrow x_3 \cdot \mathsf{G}_2)$

Output: $\mathsf{pprm_S} \leftarrow \mathsf{pk}$,
$\mathsf{privprm_S} \leftarrow \mathsf{sk}$.

ACT.ClientRegister$(\mathsf{pprm_S})$:

1. Generate a PRF key $\mathsf{u_C} \leftarrow\$ \mathbb{Z}_p$,
2. Set $\mathsf{U_C} \leftarrow \mathsf{u_C} \cdot \mathsf{G}_1$

Output: $\mathsf{pprm_C} \leftarrow \mathsf{U_C}$
$\mathsf{privprm_C} \leftarrow \mathsf{u_C}$

ACT.TokenRequest$(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{msg})$:

1. Generate random scalars $v, \mu \leftarrow\$ \mathbb{Z}_p^*$.
2. Compute $\mathsf{M}'_3 \leftarrow \mathsf{H}(\mathsf{msg})$ and $\mathsf{M}'_2 \leftarrow \mathsf{u_C} \cdot \mathsf{H}(\mathsf{msg})$.
3. Set $\vec{\mathsf{M}} \leftarrow (\mathsf{M}_2 = \mu^{-1}\mathsf{M}'_2,\ \mathsf{M}_3 = \mu^{-1}\mathsf{M}'_3)$
4. Compute the Fiat-Shamir commitments: $V = v \cdot \mathsf{G}_1$, $W = v \cdot \mathsf{M}_3$.
5. Derive the Fiat-Shamir challenge: $c \leftarrow \mathsf{H}'(\mathsf{U_C}, \vec{\mathsf{M}}, V, W)$.
6. Compute the Fiat-Shamir response: $\xi \leftarrow v + c \cdot \mathsf{u_C}$.
7. Set the proof to be: $\pi_{\vec{\mathsf{M}}} \leftarrow (\xi, c)$.

Output: $\mathsf{blindRequest} \leftarrow (\vec{\mathsf{M}}, \pi_{\vec{\mathsf{M}}})$
$\mathsf{rand_{msg}} \leftarrow (\mathsf{msg}, \vec{\mathsf{M}}, \mu)$

ACT.Sign$(\mathsf{privprm_S}, \mathsf{pprm_C}, \mathsf{blindRequest})$

1. Parse $\mathsf{blindRequest} = (\vec{\mathsf{M}},\ \pi_{\vec{\mathsf{M}}} = (\xi, c))$.
2. Verify the ZK proof $\pi_{\vec{\mathsf{M}}}$: namely compute $V' \leftarrow \xi \mathsf{G}_1 - c\mathsf{U}_\mathsf{C}$, $W' \leftarrow \xi \mathsf{M}_3 - c\mathsf{M}_2$, and abort if $c \neq \mathsf{H}'(\mathsf{U}_\mathsf{C}, \vec{\mathsf{M}}, \mathsf{V}', \mathsf{W}')$.
3. Generate a random $y \leftarrow_R \mathbb{Z}_p^*$.
4. Compute the EQS signature: $\mathsf{Z} \leftarrow y \cdot (x_2 \cdot \mathsf{M}_2 + x_3 \cdot \mathsf{M}_3)$, $\mathsf{Y} \leftarrow y^{-1} \cdot \mathsf{G}_1$, and $\hat{\mathsf{Y}} \leftarrow y^{-1} \cdot \mathsf{G}_2$.

Output: $\mathsf{blindToken} \leftarrow \rho = (\mathsf{Z}, \mathsf{Y}, \hat{\mathsf{Y}})$

ACT.Unblind$(\mathsf{pprm_S}, \mathsf{privprm_C}, \mathsf{blindToken}, \mathsf{rand_{msg}})$

1. Parse $\mathsf{rand_{msg}} = (\mathsf{msg}, \vec{\mathsf{M}}, \mu)$, $\mathsf{blindToken} = \rho = (\mathsf{Z}, \mathsf{Y}, \hat{\mathsf{Y}})$.
2. Generate a random scalar $\psi \leftarrow\!\!\$\ \mathbb{Z}_p^*$.
3. Compute $\sigma \leftarrow (\mathsf{Z}' \leftarrow \mu\psi\mathsf{Z},\ \mathsf{Y}' \leftarrow \psi^{-1}\mathsf{Y},\ \hat{\mathsf{Y}}' \leftarrow \psi^{-1}\hat{\mathsf{Y}})$
4. Set $\vec{\mathsf{M}}' \leftarrow (\mathsf{u_C} \cdot \mathsf{H}(\mathsf{msg}),\ \mathsf{H}(\mathsf{msg}))$.
5. Abort if $\mathsf{M}_2' \bullet \hat{\mathsf{X}}_2 + \mathsf{M}_3' \bullet \hat{\mathsf{X}}_3 \neq \mathsf{Z} \bullet \hat{\mathsf{Y}}$ or if $\mathsf{Y} \bullet \mathsf{G}_2 \neq \mathsf{G}_1 \bullet \hat{\mathsf{Y}}$ or $\mathsf{Y} = 0$ or $\hat{\mathsf{Y}} = 0$.

Output: $\Big(\mathsf{msg},\ \mathsf{tok} \leftarrow (\mathsf{M}_2',\ \sigma \leftarrow (\mathsf{Z}', \mathsf{Y}', \hat{\mathsf{Y}}'))\Big)$

ACT.Verify$(\mathsf{pprm_S}, \mathsf{privprm}, \mathsf{msg}, \mathsf{tok})$

1. Parse $\mathsf{tok} = (\mathsf{M}_2',\ \sigma = (\mathsf{Z}', \mathsf{Y}', \hat{\mathsf{Y}}'))$.
2. Compute $\mathsf{M}_3' \leftarrow \mathsf{H}(\mathsf{msg})$.
3. Set $\mathsf{bit} \leftarrow$ **true**.
4. If $\mathsf{M}_2' \bullet \hat{\mathsf{X}}_2 + \mathsf{M}_3' \bullet \hat{\mathsf{X}}_3 \neq \mathsf{Z} \bullet \hat{\mathsf{Y}}$ or $\mathsf{Y} \bullet \mathsf{G}_2 \neq \mathsf{G}_1 \bullet \hat{\mathsf{Y}}$ or $\mathsf{Y} = 0$ or $\hat{\mathsf{Y}} = 0$, set $\mathsf{bit} \leftarrow$ **false**.

Output: $(\mathsf{bit},\ \mathsf{tag} \leftarrow \mathsf{M}_2')$.

Correctness and unlinkability follow from correctness and unlinkability of Construction 6.3.

Unforgeability.

Theorem 8.2. *If $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is modelled as a generic bilinear group, if H and H' are modelled as a random oracle, then the ACT scheme in Construction 6.3 satisfies unforgeability from Definition 3.2.*

The generic bilinear group model was introduced in [Nec94, Sho97]. In the generic group model, group elements are represented by random strings or handles and there is an oracle that returns the handles for the sum of two group elements $X + Y$ in the same group, and the pairing of a group element in $\mathbb{G}_1$ and a group element in $\mathbb{G}_2$.

Proof. The challenger answers every query $\mathsf{H}(\mathsf{msg})$ by picking a fresh random scalar $h_\mathsf{msg} \leftarrow\!\!\$\ \mathbb{Z}_p$ and setting $\mathsf{H}(\mathsf{msg}) = h_\mathsf{msg} \cdot \mathsf{G}_1$.

We denote by $\mathsf{blindRequest}_j = (\vec{\mathsf{M}}_j, \pi_j)$ the j-th query to Sign, by y_j the scalar $y \in \mathbb{Z}_p$ used by the oracle for this query, and by $\rho_j = (\mathsf{Z}_j, \mathsf{Y}_j, \hat{\mathsf{Y}}_j)$ the output of the oracle. We denote by $\mathsf{tok}_i = (\mathsf{M}_{i,2}', \sigma_i)$ the i-th token output on the adversary, by msg_i the associated message, and by $\vec{\mathsf{M}}_i'$ the associated vector $\vec{\mathsf{M}}_i' = (\mathsf{M}_{i,2}', \mathsf{M}_{i,3}')$ where $\mathsf{M}_{i,3}' = \mathsf{H}(\mathsf{msg}_i)$.

We represent all group elements that the adversary sees (by query the group oracles or any other oracle) as (Laurent) polynomial with indeterminate x_1, x_2, y_j (for $j \in [\mathsf{T}]$), and h_msg (for queries msg to H). We recall that the Schwartz-Zippel lemma ensures that working with such Laurent polynomials, instead of their actual evaluation is indistinguishable from the adversary point of view, as with overwhelming probability, two different Laurent polynomials will yield two different elements when evaluated on random points.

Now, we follow the proof of [FHS19][8]. The only difference is that the constant terms $\pi_\star$ of the polynomials generated by the adversary (p.17) are no more scalars but instead multilinear polynomials in the $\{h_{\mathsf{msg}}\}$. The other coefficients $(\rho_\star, \chi_\star, \psi_\star)$ are all still scalars.

Following the proof, we get that for every message $\vec{\mathsf{M}'}_i = (\mathsf{M}'_{i,2}, \mathsf{M}'_{i,3})$, for which the adversary provide a signature $\sigma_i = (\mathsf{Z}'_i, \mathsf{Y}'_i, \hat{\mathsf{Y}}'_i)$, there exists $j_i \in [\mathsf{T}]$, and $\alpha_i \in \mathbb{Z}_p$ such that:

$$\vec{\mathsf{M}'}_i = \alpha_i \cdot \vec{\mathsf{M}}_{j_i}$$

(In the proof from [FHS19], α_i is the product $\rho_{z,n}\psi_{y,n}$, and as argued above those coefficients are scalars, and do not contain any h_{msg}.)

Type-1 Forgeries. Let us first rule out Type-1 forgeries. If such a forgery happens, since there are $\mathsf{T}+1$ valid tokens output by the adversary but only T queries to Sign, there must exist $i \neq i'$ and $j = j_i = j_{i'}$, so that:

$$\vec{\mathsf{M}'}_i = \alpha_i \cdot \vec{\mathsf{M}}_j \qquad \text{and} \qquad \vec{\mathsf{M}'}_{i'} = \alpha_{i'} \cdot \vec{\mathsf{M}}_j$$

Since $\mathsf{M}'_{i,3} = \mathsf{H}(\mathsf{msg}_i) = h_{\mathsf{msg}_i} \cdot \mathsf{G}_1$ and $\mathsf{M}'_{i',3} = h_{\mathsf{msg}_{i'}} \cdot \mathsf{G}_1$, we have that:

$$h_{\mathsf{msg}_i} = \alpha_i \mathsf{M}_{j,3} \qquad \text{and} \qquad h_{\mathsf{msg}_{i'}} = \alpha_{i'} \mathsf{M}_{j,3}$$

which is impossible since h_{msg_i} and $h_{\mathsf{msg}_{i'}}$ are two different indeterminates.

Type-2 Forgeries. If such a forgery happens, since there are only R registered users but $\mathsf{R}+1$ forgeries for the same message msg, there must exist $i \neq i'$, $j = j_i$, and $j' = j_{i'}$ so that:

$$\vec{\mathsf{M}'}_i = \alpha_i \cdot \vec{\mathsf{M}}_j \qquad \text{and} \qquad \vec{\mathsf{M}'}_{i'} = \alpha_{i'} \cdot \vec{\mathsf{M}}_{j'}$$

and the j-th and j'-th query to Sign were made for the same client C, with public parameters $\mathsf{pprm}_C = \mathsf{U}$. With a similar reasoning as for Type-1 forgeries, we get that:

$$\mathsf{M}_{j,3} = \mathsf{M}_{j',3} = \mathsf{H}(\mathsf{msg}) = h_{\mathsf{msg}} \tag{9}$$

Let us now show that $\log \mathsf{M}_{j,2} \cdot \log \mathsf{U} = \log \mathsf{M}_{j,3}$ where log represents the discrete logarithm in base G_1 or equivalently, the evaluation of the Laurent polynomials. Since the signing query was successful, the Fiat-Shamir proof was successful, and the adversary produced V', W' so that:

$$\mathsf{V}' = \xi \mathsf{G}_1 - c\mathsf{U} \qquad \text{and} \qquad \mathsf{W}' = \xi \mathsf{M}_{j,3} - c\mathsf{M}_{j,2}$$

where $c = \mathsf{H}'(\mathsf{U}_\mathsf{C}, \vec{\mathsf{M}}_j, V', W')$. This can be rewritten in matrix form:

$$\begin{pmatrix} \log \mathsf{V}' \\ \log \mathsf{W}' \end{pmatrix} = \mathit{\Gamma} \cdot \begin{pmatrix} \xi \\ -c \end{pmatrix} \qquad \text{where } \mathit{\Gamma} = \begin{pmatrix} 1 & \log \mathsf{M}_{j,3} \\ \log \mathsf{U} & \log \mathsf{M}_{j,2} \end{pmatrix}$$

If $\mathit{\Gamma}$ is invertible, the above system has a unique solution, so the above proof can hold only for a single $c = \mathsf{H}'(\mathsf{U}_\mathsf{C}, \vec{\mathsf{M}}_j, V', W')$. Since H' is modeled as a random oracle, it would output the only possible c with probability $1/p$. Thus, except with negligible probability, $\mathit{\Gamma}$ is not invertible and thus $\log \mathsf{M}_{j,2} \cdot \log \mathsf{U} = \log \mathsf{M}_{j,3}$.

The same holds true for $\log \mathsf{M}_{j',2} \cdot \log \mathsf{U} = \log \mathsf{M}_{j',3}$.

We conclude using Eq. (9) that $\mathsf{M}'_{j,2} = \mathsf{M}'_{j',2}$. So the tags ($\mathsf{tag}$) of both forgeries are identical, which means such a Type-2 forgery cannot happen. □

[8] Eprint version `https://eprint.iacr.org/archive/2014/944/20210121:101436`

Parameter Generation for Unlinkability. Unlinkability assumes that the ACT parameters are generated honestly and the issuer only sees the secret key. Following the discussion in Section 6, it is sufficient to add a Fiat-Shamir proof of knowledge of the discrete logarithms x_2 and x_3 in $\mathsf{pprm}_{\mathsf{S}}$ to get unlinkability even for maliciously-generated parameters. Note that we do not need a CRS for this construction and that those proofs are extremely efficient (namely consisting of two scalars from $\mathbb{Z}_p$).

9 General Rate Limiting

Our ACT constructions guarantee that each registered client can obtain a single token per message. A generalization of this property is to enforce a different threshold of how many tokens per message a client can obtain. The challenge here is to enforce this without revealing to the issuer or verifier how many tokens a client has obtained for a specific message, as long as it is below the threshold.

One approach to achieve that from an ACT construction that supports one token per message rate limit is to change the input on which we evaluate the PRF under the client's key, which is used to enforce rate limiting. In particular, if we want to allow 2^k different tokens per message, we will evaluate that PRF on inputs of the form $\mathsf{msg}||s$ where msg is the actual message that will be used for the token verification and s is a value of length k bits which can be chosen by the client. Thus, a client can generate 2^k different inputs for the rate-limiting PRF for each message and hence can obtain 2^k tokens per message. The issuer will not be able to distinguish requests for the same message vs requests for different messages as long as no more than 2^k requests per message have been submitted by a client.

The only remaining part is to have the client give a proof that it has added a suffix of k bits to its message before evaluating the rate-limiting PRF. Unfortunately providing s in the clear does not preserve unlinkability across clients. Even though it could be chosen at random in $[0, 2^k - 1]$ by the client, the same value s for the same message can be used only for tokens that belong to different clients. This leaks information about clients: if the verifier sees two redemptions with the same value s, it learns they correspond to two different clients.

Thus we need to provide a ZK proof for the correct length of s without revealing it. This can be done by the client committing to the k bits of its value $s = s_1\| \ldots \|s_k$ and proving that given $\mathsf{Commit}(\mathsf{msg})$ the value that is input for the rate-limiting PRF evaluation is of the form $2^k \cdot \mathsf{msg} + \sum_{i=0}^{k-1} 2^i \cdot s_{i+1}$.

References

AFK22. Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 113–142. Springer, Heidelberg, November 2022.

AGR$^+$16. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, December 2016.

AOS02. Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 415–432. Springer, Heidelberg, December 2002.

Bar16. Elaine Barker. Recommendation for key management, part 1: General, 2016-01-28 2016.

BB04. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, Heidelberg, May 2004.

BBG$^+$20. James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1253–1269. ACM Press, November 2020.

BDL$^+$12. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *J. Cryptogr. Eng.*, 2(2):77–89, 2012.

BIK$^+$17. Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1175–1191. ACM Press, October / November 2017.

Bow17. Sean Bowe. Bls12-381: New zk-snark elliptic curve construction. `https://electriccoin.co/blog/new-snark-curve/`, March 2017.

BP97. Niko Bari and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494. Springer, Heidelberg, May 1997.

BPW12. David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, Heidelberg, December 2012.

BRS23. Fabrice Benhamouda, Mariana Raykova, and Karn Seth. Anonymous counting tokens. Cryptology ePrint Archive, Paper 2023/320, 2023. `https://eprint.iacr.org/2023/320`.

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

CDV23. Melissa Chase, F. Betül Durak, and Serge Vaudenay. Anonymous tokens with stronger metadata bit hiding from algebraic macs. In *Advances in Cryptology – CRYPTO 2023*, 2023.

Cha82. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

CM99. Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 107–122. Springer, Heidelberg, May 1999.

Cra97. Ronald Cramer. *Modular design of secure yet practical cryptographic protocols*. PhD thesis, University of Amsterdam, 1997.

CS03. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, August 2003.

DGS$^+$18. Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018.

DY05. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.

EG14. Alex Escala and Jens Groth. Fine-tuning Groth-Sahai proofs. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 630–649. Springer, Heidelberg, March 2014.

FG18. Georg Fuchsbauer and Romain Gay. Weakly secure equivalence-class signatures from standard assumptions. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 153–183. Springer, Heidelberg, March 2018.

FHS19. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.

FO97. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 16–30. Springer, Heidelberg, August 1997.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GKR$^+$21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

GMR98. Rosario Gennaro, Daniele Micciancio, and Tal Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In Li Gong and Michael K. Reiter, editors, *ACM CCS 98*, pages 67–72. ACM Press, November 1998.

GMY03. Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 177–194. Springer, Heidelberg, May 2003.

Gra22. Kevin Graney. Privacy Sandbox k-anonymity Server. https://github.com/WICG/turtledove/blob/main/FLEDGE_k_anonymity_server.md#privacy-enhancements-we-are-exploring, 2022.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

HIP+22. Scott Hendrickson, Jana Iyengar, Tommy Pauly, Steven Valdez, and Christopher A. Wood. Rate-Limited Token Issuance Protocol. https://datatracker.ietf.org/doc/draft-privacypass-rate-limit-tokens/, 2022.

HS21. Lucjan Hanzlik and Daniel Slamanig. With a little help from my friends: Constructing practical anonymous credentials. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2004–2023. ACM Press, November 2021.

JR17. Charanjit S. Jutla and Arnab Roy. Improved structure preserving signatures under standard bilinear assumptions. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 183–209. Springer, Heidelberg, March 2017.

KLOR20. Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous tokens with private metadata bit. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 308–336. Springer, Heidelberg, August 2020.

MPR+20. Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2020.

Nec94. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

NPR99. Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346. Springer, Heidelberg, May 1999.

Ode09. Goldreich Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 1st edition, 2009.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

SS22. Tjerand Silde and Martin Strand. Anonymous tokens with public metadata and applications to private contact tracing. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 179–199. Springer, Heidelberg, May 2022.

TCR+22. Nirvan Tyagi, Sofía Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A Wood. A fast and simple partially oblivious prf, with applications. In *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part II*, pages 674–705. Springer, 2022.

Appendix

A Sigma Protocols

Definition A.1 (Sigma Protocol). *A sigma protocol proving a relation $\mathcal{R}$ consists of three PPT algorithms $(\mathcal{C}, \mathcal{R}, \mathcal{V})$:*

- $\alpha \leftarrow \mathcal{C}(x, w; r)$*: given a statement x and a witness w output initial message α computed with randomness r.*
- $\beta \leftarrow \mathcal{R}(x, w, r, e)$*: using the statement x and the witness w together with the randomness used to generate the first message, compute a response β to a challenge e.*
- $\mathsf{bit} \leftarrow \mathcal{V}(x, \alpha, \beta, e)$*: compute verification bit bit for a statement x using the initial message α, the random challenge e and the response β.*

Figure 8 show the interactions between a prover P and a verifier V in a sigma protocol. Using the Fiat-Shamir heuristic sigma protocol can be made interactive by sampling e using a random oracle on the input and output from the first message.

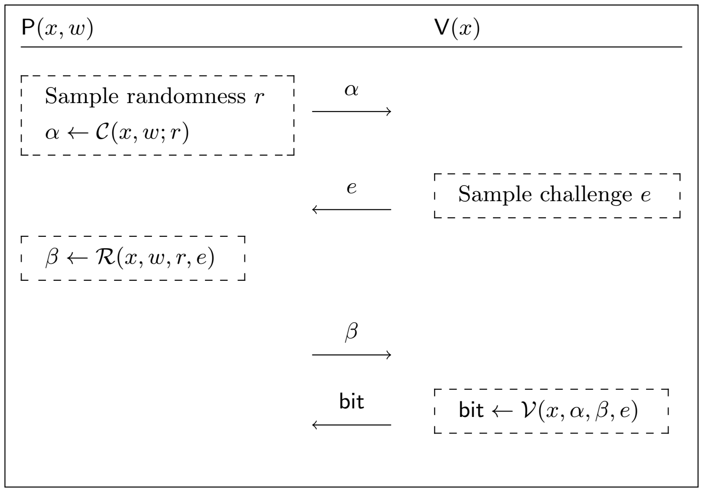


Fig. 8: Interactive Sigma Protocol.

Previous work have proven composition for sigma protocols for AND and OR relations as well as general monotone formulas [CDS94, AOS02, GMY03]. Here we overview some of the building block sigma protocols that we use in our constructions. To optimize communication efficiency, we note that the verifier can derive message α from β, e and the statement, thus α does not need to be sent.

Construction A.2. Sigma protocols for proving knowledge and equality over committed values with Pedersen commitment $\mathsf{com}_x = x\mathsf{G} + r\mathsf{H}$ and Comenisch-Shoup encrypted value $\mathsf{Enc}(x) = (a\mathsf{G}_{\mathsf{CS}}, x\mathsf{H}_{\mathsf{CS}} + a\mathsf{Y}_{\mathsf{CS}})$.

- $\mathcal{C}(x, w; r)$: Generate random values y, r', b and output

$$\mathsf{com}_r = y\mathsf{G} + r'\mathsf{H}, \quad \mathsf{Enc}(y) = (b\mathsf{G}_{\mathsf{CS}}, y\mathsf{H}_{\mathsf{CS}} + b\mathsf{Y}_{\mathsf{CS}}).$$

- $\mathcal{R}(x, w, r, e)$: Compute and output $\alpha = y + e \cdot x$, $\mathsf{H}' = (r' + e \cdot r) \cdot \mathsf{H}$, $\mathsf{H}'' = (b + e \cdot a)$.
- $\mathcal{V}(\mathsf{com}_x, \mathsf{Enc}(x), \mathsf{com}_r, \mathsf{Enc}(y), \alpha, \mathsf{H}', \mathsf{H}'', e)$: Output the bit of checking that both comparisons hold:

$$\begin{aligned} e \cdot (\mathsf{com}_x) + \mathsf{com}_r &= \alpha \cdot \mathsf{G} + \mathsf{H}' \\ e \cdot \mathsf{Enc}(x) + \mathsf{Enc}(y) &= \alpha \cdot \mathsf{G}_{\mathsf{CS}} + \mathsf{H}''. \end{aligned}$$

Construction A.3. Let $\mathsf{PRF}(\mathsf{u}, \mathsf{msg}) = (\mathsf{u} + \mathsf{msg})^{-1} \cdot \mathsf{G}$, then we instantiate the proof

$$\begin{aligned} \pi_{\mathsf{PRF}} : \mathsf{ZK}\{\ \exists\ \mathsf{msg}, \mathsf{u} : \ & \mathsf{F} = (\mathsf{u} + \mathsf{msg})^{-1} \cdot \mathsf{G}, \\ & \mathsf{com}_{\mathsf{msg}} = \mathsf{msg} \cdot \mathsf{H}_{\mathsf{CS}} + r_{\mathsf{msg}} \cdot \mathsf{Y}_{\mathsf{CS}}, \\ & \tilde{\mathsf{com}}_{\mathsf{msg}} = \mathsf{msg} \cdot \mathsf{G} + r_{\mathsf{msg}} \cdot \mathsf{H}, \\ & \mathsf{com}_{\mathsf{u}} = \mathsf{u} \cdot \mathsf{G} + r_{\mathsf{u}} \cdot \mathsf{H}\} \end{aligned}$$

as a sigma protocol as follows:

- $\mathcal{C}(\mathsf{F}, \mathsf{com}_{\mathsf{msg}}, \tilde{\mathsf{com}}_{\mathsf{msg}}, \mathsf{com}_{\mathsf{u}}, \mathsf{u}, \mathsf{msg}; r)$:

$$\begin{aligned} \alpha \leftarrow (& \mathsf{com}'_{\mathsf{msg}} = \mathsf{msg}' \cdot \mathsf{H}_{\mathsf{CS}} + r'_{\mathsf{msg}} \cdot \mathsf{Y}_{\mathsf{CS}} \\ & \tilde{\mathsf{com}}'_{\mathsf{msg}} = \mathsf{msg}' \cdot \mathsf{G} + r'_{\mathsf{msg}} \cdot \mathsf{H} \\ & \mathsf{com}'_{\mathsf{u}} = \mathsf{u}' \cdot \mathsf{G} + r'_{\mathsf{u}} \cdot \mathsf{H}, \\ & \mathsf{F}' = (\mathsf{msg}' + \mathsf{u}')\mathsf{F}) \end{aligned}$$

– $\mathcal{R}(\mathsf{F}, \mathsf{com}_{\mathsf{msg}}, \mathsf{com}_{\mathsf{u}}, \mathsf{u}, \mathsf{msg}; r, e)$: Output

$$\beta \leftarrow \Big(\mathsf{msg}'' = \mathsf{msg}' + e \cdot \mathsf{msg},\ r''_{\mathsf{msg}} = r'_{\mathsf{msg}} + e \cdot r_{\mathsf{msg}}$$
$$\mathsf{u}'' = \mathsf{u}' + e \cdot \mathsf{u},\ r''_{\mathsf{u}} = r'_{\mathsf{u}} + e \cdot \mathsf{u} \Big)$$

– $\mathcal{V}(\mathsf{F}, \mathsf{com}_{\mathsf{msg}}, \mathsf{com}_{\mathsf{u}}, \alpha, \beta, e)$ Check that all of the following hold:

$$\begin{aligned}
&\mathsf{com}'_m + e \cdot \mathsf{com}_{\mathsf{msg}} = \mathsf{msg}'' \cdot \mathsf{H}_{\mathsf{CS}} + r''_{\mathsf{msg}} \cdot \mathsf{Y}'_{\mathsf{CS}} \\
&\tilde{\mathsf{com}}'_m + e \cdot \tilde{\mathsf{com}}_{\mathsf{msg}} = \mathsf{msg}'' \cdot \mathsf{G} + r''_{\mathsf{msg}} \cdot \mathsf{H} \\
&\tilde{\mathsf{com}}'_m + e \cdot \mathsf{com}_{\mathsf{msg}} = \mathsf{msg}'' \cdot \mathsf{H}_{\mathsf{CS}} + r''_{\mathsf{msg}} \cdot \mathsf{H}'_{\mathsf{CS}} \\
&\mathsf{com}'_{\mathsf{u}} + e \cdot \mathsf{com}_{\mathsf{u}} = \mathsf{u}'' \cdot \mathsf{G} + r''_{\mathsf{u}} \cdot \mathsf{H}_{\mathsf{CS}} \\
&(\mathsf{msg}'' + \mathsf{u}'') \cdot \mathsf{F} = \mathsf{F}' + e \cdot \mathsf{G} \\
&\mathsf{msg}'' <= q \cdot 2^{2\lambda+1} \\
&\mathsf{u}'' <= q \cdot 2^{2\lambda+1}
\end{aligned}$$

The communication of the above protocol is 1 Pedersen commitment (in addition to 1 Extractable commitment to m and a Pedersen commitment to k, which we assume have already been communicated to the verifier), and 4 scalars. Of the scalars, 2 are of size $\log(p) + 2\lambda$ bits, and 2 are $\log(N) + 2\lambda$ bits, where N is the RSA modulus for the Pedersen Commitments and Camenisch Shoup encryption, and p is the size of the DY-PRF group. The total size of the scalars is $2\log(p) + 2\log(N) + 8\lambda$ bits.

Construction A.4. Let CS be the Camenisch Shoup encryption, which is also used for the extractable commitment $\mathsf{Commit}_{\mathsf{Ext}}$ with a different set of parameters, and $\mathsf{Commit}_{\mathsf{Ped}}$ be the Pedersen commitment, then we instantiate

$$\begin{aligned}
\pi = \mathsf{ZK}\{\exists\ & a, b, \mathsf{msg}, r, t_a, t_b, t_{\mathsf{msg}}, t_r, t_{ar} \text{ s.t. :} \\
& \mathsf{ct}_\beta = \mathsf{Enc}(\mathsf{PK}_{\mathsf{CS}}, a \cdot \mathsf{msg} + b \cdot p) + a \cdot \mathsf{ct}_{\mathsf{u}} + a \cdot r \cdot \mathsf{ct}_{\mathsf{y}}), \\
& \mathsf{com}_a = \mathsf{Commit}_{\mathsf{Ext}}(a, t_a), \mathsf{com}_r = \mathsf{Commit}_{\mathsf{Ped}}(r, t_r), \mathsf{com}_{ar} = \mathsf{Commit}_{\mathsf{Ped}}(a \cdot r, t_{ar}), \\
& \mathsf{com}_b = \mathsf{Commit}_{\mathsf{Ped}}(b, t_b), \mathsf{com}_{\mathsf{msg}} = \mathsf{Commit}_{\mathsf{Ext}}(\mathsf{msg}, t_{\mathsf{msg}}), \\
& a < p \cdot 2^{2\lambda+1}, r < p \cdot 2^{2\lambda+1}, b < p^2 \cdot 2^{3\lambda+1}\}.
\end{aligned}$$

– $\mathcal{C}(x, w; r)$ where:
  - statement: $\mathsf{com}_{\mathsf{msg}}$, com_r, com_a, com_b, com_{ar}, com_{am}, ct_β, $\mathsf{ct}_{\mathsf{u}} = (\mathsf{U}_{\mathsf{ct}_{\mathsf{u}}}, \mathsf{W}_{\mathsf{ct}_{\mathsf{u}}})$, $\mathsf{ct}_{\mathsf{y}} = (\mathsf{U}_{\mathsf{ct}_{\mathsf{y}}}, \mathsf{W}_{\mathsf{ct}_{\mathsf{y}}})$
  - witness:

$$\begin{aligned}
&\mathsf{com}_{\mathsf{msg}} = (\mathsf{U}_{\mathsf{msg}} = t_{\mathsf{msg}} \cdot \mathsf{G}_{\mathsf{Ext}}, \mathsf{W}_{\mathsf{msg}} = \mathsf{msg} \cdot \mathsf{H}_{\mathsf{Ext}} + t_{\mathsf{msg}} \cdot \mathsf{Y}_{\mathsf{Ext}}) \\
&\mathsf{com}_r = r\mathsf{G} + t_r\mathsf{H} \\
&\mathsf{com}_a = (\mathsf{U}_a = t_a \cdot \mathsf{G}_{\mathsf{Ext}}, \mathsf{W}_a = a \cdot \mathsf{H}_{\mathsf{Ext}} + t_a \cdot \mathsf{Y}_{\mathsf{Ext}}) \\
&\mathsf{com}_b = b\mathsf{G} + t_b\mathsf{H} \\
&\mathsf{com}_{ar} = (a \cdot r) \cdot \mathsf{G} + (a \cdot t_r) \cdot \mathsf{H} \\
&\mathsf{com}_{am} = (a \cdot \mathsf{msg}) \cdot \mathsf{H}_{\mathsf{Ext}} + (a \cdot t_{\mathsf{msg}}) \cdot \mathsf{Y}_{\mathsf{Ext}} \\
&\mathsf{Enc}(\mathsf{PK}_{\mathsf{CS}}, a \cdot \mathsf{msg} + b \cdot p) = (t_{\mathsf{Enc}} \cdot \mathsf{G}_{\mathsf{CS}}, (a \cdot \mathsf{msg} + b \cdot p) \cdot \mathsf{H}_{\mathsf{CS}} + (t_{\mathsf{Enc}} \cdot \mathsf{Y}_{\mathsf{CS}})) \\
&\mathsf{ct}_\beta = \mathsf{Enc}(\mathsf{PK}_{\mathsf{CS}}, a \cdot \mathsf{msg} + b \cdot p) + a \cdot \mathsf{ct}_{\mathsf{u}} + a \cdot r \cdot \mathsf{ct}_{\mathsf{y}} = (\mathsf{U}_\beta, \mathsf{W}_\beta) \\
&\mathsf{U}_\beta = t_{\mathsf{Enc}} \cdot \mathsf{G}_{\mathsf{CS}} + a \cdot \mathsf{U}_{\mathsf{ct}_{\mathsf{u}}} + a \cdot r \cdot \mathsf{U}_{\mathsf{ct}_{\mathsf{y}}} \\
&\mathsf{W}_\beta = (a \cdot \mathsf{msg} + b \cdot p) \cdot \mathsf{H} + a \cdot \mathsf{W}_{\mathsf{ct}_{\mathsf{u}}} + (a \cdot r) \cdot \mathsf{W}_{\mathsf{ct}_{\mathsf{y}}} + t_{\mathsf{Enc}} \cdot \mathsf{Y}_{\mathsf{CS}}
\end{aligned}$$

- Generate δ as:

$$
\begin{aligned}
&\tilde{\mathsf{com}}_{\mathsf{msg}} = (t'_{\mathsf{msg}} \cdot \mathsf{G}_{\mathsf{Ext}}, \mathsf{msg}'\mathsf{H}_{\mathsf{Ext}} + t'_{\mathsf{msg}}\mathsf{Y}_{\mathsf{Ext}})\\
&\tilde{\mathsf{com}}_r = r'\mathsf{G} + t'_r\mathsf{H},\\
&\tilde{\mathsf{com}}_a = (t'_a\mathsf{G}_{\mathsf{Ext}}, a'\mathsf{G}_{\mathsf{Ext}} + t'_a\mathsf{H}_{\mathsf{Ext}}),\\
&\tilde{\mathsf{com}}_b = b'\mathsf{G} + t'_b\mathsf{H},\\
&\tilde{\mathsf{com}}_{ar} = \alpha'_{ar}\mathsf{G} + t'_{ar}\mathsf{H},\\
&\hat{\mathsf{com}}_{ar} = a'\mathsf{com}_r\\
&\tilde{\mathsf{com}}_{am} = \alpha'_{am}\mathsf{H}_{\mathsf{Ext}} + t'_{am}\mathsf{Y}_{\mathsf{Ext}}\\
&\hat{\mathsf{com}}_{am} = a'\mathsf{W}_{\mathsf{msg}}\\
&\tilde{\mathsf{U}}_\beta = t'_{\mathsf{Enc}} \cdot \mathsf{G}_{\mathsf{CS}} + a' \cdot \mathsf{U}_{\mathsf{ct}_u} + \alpha'_{ar} \cdot \mathsf{U}_{\mathsf{ct}_y}\\
&\tilde{\mathsf{W}}_\beta = (\alpha'_{am} + b' \cdot p) \cdot H + a' \cdot \mathsf{W}_{\mathsf{ct}_u} + \alpha'_{ar} \cdot \mathsf{W}_{\mathsf{ct}_y} + t'_{\mathsf{Enc}} \cdot \mathsf{Y}_{\mathsf{CS}}
\end{aligned}
$$

– $\mathcal{R}(x, w, r, e)$: Compute and return τ as

$$
\begin{aligned}
&\mathsf{msg}'' = \mathsf{msg}' + e \cdot \mathsf{msg}\\
&t''_{\mathsf{msg}} = t'_{\mathsf{msg}} + e \cdot t_{\mathsf{msg}}\\
&r'' = r' + e \cdot r\\
&t''_r = t'_r + e \cdot t_r\\
&a'' = a' + e \cdot a\\
&t''_a = t'_a + e \cdot t_a\\
&b'' = b' + e \cdot b\\
&t''_b = t'_b + e \cdot t_b\\
&\alpha''_{ar} = \alpha'_{ar} + e \cdot a \cdot r\\
&t''_{ar} = t'_{ar} + e \cdot a \cdot t_r\\
&\alpha''_{am} = \alpha'_{am} + e \cdot a \cdot \mathsf{msg}\\
&t''_{am} = t'_{am} + e \cdot a \cdot t_{\mathsf{msg}}\\
&t''_{\mathsf{Enc}} = t'_e nc + e \cdot t_{\mathsf{Enc}}
\end{aligned}
$$

– $\mathcal{V}(x, \delta, \tau, e)$: Check:

$$(t''_{\mathsf{msg}} \cdot \mathsf{G}_{\mathsf{Ext}}, \mathsf{msg}'' \cdot \mathsf{H}_{\mathsf{Ext}} + t''_{\mathsf{msg}} \cdot \mathsf{Y}_{\mathsf{Ext}}) = \tilde{\mathsf{com}}_{\mathsf{msg}} + e \cdot \mathsf{com}_{\mathsf{msg}}$$
$$r''\mathsf{G} + t''_r\mathsf{H} = \tilde{\mathsf{com}}_r + e \cdot \mathsf{com}_r$$
$$(t''_a \cdot \mathsf{G}_{\mathsf{Ext}}, a'' \cdot \mathsf{H}_{\mathsf{Ext}} + t''_a \cdot \mathsf{Y}_{\mathsf{Ext}}) = \tilde{\mathsf{com}}_a + e \cdot \mathsf{com}_a$$
$$b''\mathsf{G} + t''_b\mathsf{H} = \tilde{\mathsf{com}}_b + e \cdot \mathsf{com}_b$$
$$a'' \cdot \mathsf{com}_r = \hat{\mathsf{com}}_{ar} + e \cdot \mathsf{com}_{ar}$$
$$\alpha''_{ar} \cdot \mathsf{G} + t''_{ar} \cdot \mathsf{H} = \tilde{\mathsf{com}}_{ar} + e \cdot \mathsf{com}_{ar}$$
$$a'' \cdot \mathsf{W}_{\mathsf{msg}} = \hat{\mathsf{com}}_{am} + e \cdot \mathsf{com}_{am}$$
$$\alpha''_{am} \cdot \mathsf{H}_{\mathsf{Ext}} + t''_{am} \cdot \mathsf{Y}_{\mathsf{Ext}} = \tilde{\mathsf{com}}_{am} + e \cdot \mathsf{com}_{am}$$
$$\tilde{\mathsf{U}}_\beta + e \cdot \mathsf{U}_\beta = t''_{\mathsf{Enc}} \cdot \mathsf{G}_{\mathsf{CS}} + a'' \cdot \mathsf{U}_{\mathsf{ct_u}} + \alpha''_{ar} \cdot \mathsf{U}_{\mathsf{ct_y}}$$
$$\tilde{\mathsf{W}}_\beta + e \cdot \mathsf{W}_\beta = (\alpha''_{am} + p \cdot b'') \cdot \mathsf{H}_{\mathsf{CS}} + a'' \cdot \mathsf{W}_{\mathsf{ct_u}} + \alpha''_{ar} \cdot \mathsf{W}_{\mathsf{ct_y}} + t''_{\mathsf{Enc}} \cdot \mathsf{Y}_{\mathsf{CS}}$$
$$\mathsf{msg}'' <= q \cdot 2^{2\lambda+1}$$
$$r'' <= q \cdot 2^{2\lambda+1}$$
$$a'' <= q \cdot 2^{2\lambda+1}$$
$$b'' <= 2q \cdot 2^{3\lambda+1}$$

Using the optimization we discussed above, the communication cost of the above protocol is the size of the statement, which is 4 Pedersen commitments, 2 extractable commitments and 1 CS ciphertext (assuming that $\mathsf{ct_u}$, $\mathsf{ct_y}$ are known to the verifier), and 13 scalars corresponding to the openings. Of the scalars, we have 3 of size $\log(p) + 2\lambda$, 2 of size $2\log(p) + 2\lambda$, 1 of size $2\log(p) + 3\lambda$ and 7 of size $\log(N) + 2\lambda$. In total, the scalars have size $9\log(p) + 7\log(N) + 27\lambda$ bits.