Scalable and Fine-Tuned Privacy Pass from Group Verifiable Random Functions

Dennis Faut*
University of Luxembourg & KASTEL SRL
Karlsruhe, Germany
dennis.faut@kit.edu

Julia Hesse†
IBM Research Europe
Zurich, Switzerland
jhs@zurich.ibm.com

Lisa Kohl‡
Centrum Wiskunde & Informatica (CWI)
Amsterdam, Netherlands
lisa.kohl@cwi.nl

Andy Rupp*
University of Luxembourg & KASTEL SRL
Belval, Luxembourg
andy.rupp@uni.lu

***Abstract*—Anonymous token schemes are cryptographic protocols for limiting the access to online resources to credible users. The resource provider issues a set of access tokens to the credible user that they can later redeem anonymously, i.e., without the provider being able to link their redemptions. When combined with credibility tests such as CAPTCHAs, anonymous token schemes can significantly increase user experience and provider security, without exposing user access patterns to providers.**

Current anonymous token schemes such as the Privacy Pass protocol by Davidson et al. rely on *oblivious pseudorandom functions* (OPRFs), which let server and user jointly compute randomly looking access tokens. For those protocols, token issuing costs are linear in the number of requested tokens.

In this work, we propose a new approach for building anonymous token schemes. Instead of relying on two-party computation to realize a privacy-preserving pseudorandom function evaluation, we propose to offload token generation to the user by using *group verifiable random functions* (GVRFs). GVRFs are a new cryptographic primitive that allow users to produce verifiable pseudorandomness. Opposed to standard VRFs, verification is *anonymous within the group* of credible users. We give a construction of group VRFs from the Dodis-Yampolskiy VRF and Equivalence-Class Signatures, based on pairings and a new Diffie-Hellman inversion assumption that we analyze in the Generic Group Model. Our construction enjoys compact public keys and proofs, while evaluation and verification costs are only slightly increased compared to the Dodis-Yampolskiy VRF.

By deploying a group VRF instead of a OPRF, we obtain an anonymous token scheme where communication as well as server-side computation during the issuing phase is constant and independent of the number of tokens a user requests. Moreover, by means of our new concept of updatable token policies, the number of unspent tokens in circulation can *retrospectively* (i.e., even after the credibility check) be decreased or increased in order to react to the current or expected network situation. Our tokens are further countable and publicly verifiable. This comes at the cost of higher computational efforts for token redemption and verification as well as somewhat weaker unlinkability guarantees compared to Privacy Pass.

***Index Terms*—Verifiable random functions, anonymous token schemes, Privacy Pass, pairing-based cryptography**

1. Introduction

CAPTCHAs are proofs of human work that protect internet resources from bot access and DoS attacks. Users are asked to solve a visual riddle, such as reading distorted letters or recognizing items in photographs, whenever a content provider deems their request suspicious. Solving the riddle reveals nothing about the identity of the user except that, most likely, there is a human behind the request. In particular, providers cannot *link* CAPTCHA actions, preventing them from tracking users on the internet. On the negative side, CAPTCHAs constitute annoying obstacles for users and can prevent them from accessing contents [22].

In 2018, Davidson et al. [14] proposed Privacy Pass, a way to significantly decrease the number of CAPTCHAs for users to solve. In a nutshell, the protocol works as follows. First, the server (i.e., content provider) is set up to hold a secret key K. Whenever a user proves that she is human via a CAPTCHA, the user's browser and server engage in an oblivious evaluation of $w := \mathsf{PRF}_K(t)$, where t is chosen at random by the browser. Such a protocol is called an oblivious pseudo-random function (OPRF), and it reveals nothing to the user beyond w, and nothing at all to the server. We call the pair (t, w) an *anonymous access token*, and the browser can use it next time the server doubts the user's credibility: instead of challenging

* *Supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs.*

† *Supported by the Swiss National Science Foundation (SNSF) under the AMBIZIONE grant "Cryptographic Protocols for Human Authentication and the IoT".*

‡ *Supported by the NWO Talent Programme Veni (VI.Veni.222.348) and the NWO Gravitation Project QSC.*

the user with a CAPTCHA, the browser sends (t, w)[1]. The server is convinced that the user has previously solved a CAPTCHA, as there is no other way to compute w such that $w = \mathsf{PRF}_K(t)$. Crucially, the spending of these tokens is *anonymous* and even *unlinkable*, meaning that the server has no means to determine which tokens have been generated by the same user. This property is ensured by the obliviousness of the PRF evaluation, where the server does not learn anything about the input t and the output w of the user. To save even more CAPTCHA work, [14] suggests that the server can agree to issue multiple tokens per completed CAPTCHA. Of course, this not only speeds up accesses but also reduces the number of CAPTCHAs that an attacker has to solve when gathering Privacy Pass tokens to be used in a DoS attack. [14] carefully considers the pro's and con's and ends up suggesting a number below 100. Privacy Pass enjoys fast token verification (one PRF evaluation and one hash), double spending protection, and seemless key rotation (server chooses a fresh OPRF key), while the communication and computation of its issuing phase is linear in the number n of requested tokens (n PRF evaluations). Privacy Pass is available through the Chrome and Firefox browsers.

A new approach to building anonymous access tokens. In this work, we build an alternative to Privacy Pass trading faster token issuing on the server-side for slower token verification, which is suitable for scenarios with few issuing servers and many verification servers (cf. 8. In our system, communication as well as server-side computation during the issuing phase is constant and independent of the number of tokens a user gets for each CAPTCHA solution. The constant-size data (called pre-token), received during issuance, enables a user to locally derive (multiple) tokens. Our tokens are further *fine-tunable* in the sense that the server can adaptively decide *how many* tokens any credible user is allowed to derive from the pre-token at any given point in time *after* the CAPTCHA was already solved. For example, a server can decide to allow every credible user to compute another 100 tokens whenever fast content delivery has priority, and it can decide to limit the number of newly created tokens to 1 per credible user whenever there is an increased risk of a bot attack, i.e., security has priority. This redemption policy update does not cause any overhead such as further interactions between issuing server and users. Besides this sort of time-wise fine-tuning, also a content-specific fine-tuning is possible (without having separate instances of our scheme), by allowing a different number of tokens per credible user for different kind of web contents. This could be especially beneficial in avoiding further DoS-Attacks by allowing less tokens for more computation-intensive tasks as API calls. See Section 8 and Appendix D for further details on updatable policies.

As in Privacy Pass, token spendings are anonymous and unlinkable, where we achieve a somewhat weaker form of unlinkability (cf. Section 8), key rotation is seemless, and double spending can be prevented by the server.

A key contribution of our paper is a new cryptographic primitive that we call *group verifiable random function* (GVRF), which is particularly useful for constructing anonymous tokens. Essentially, we can replace the OPRF in Privacy Pass with a GVRF, and directly gain the above-described issuing speedup and fine tuning. In a nutshell, a GVRF allows each user in a group to anonymously evaluate their *own* random function in a verifiable way. More precisely, if Alice and Bob are two group members (each holding a secret key sk_A, sk_B), their evaluations $\mathsf{VRF}_{sk_A}(x)$ and $\mathsf{VRF}_{sk_B}(x)$ of any input x do not reveal anything about sk_A and sk_B. This description is already enough to explain the usefulness of GVRFs in the context of anonymous tokens, and we will give a more comprehensive introduction of GVRFs later. The main idea is to replace the "global" PRF in Privacy Pass with the *user-specific* random functions of a GVRF. This approach makes the following simple but significant difference: we can re-use the user-generated parts t of the tokens among all users. Let us explain this in more detail. Assume tokens are of the form $(t, f(t))$, where $f()$ is a PRF in Privacy Pass, and a GVRF in our work. If $f()$ is a PRF and the server holds the PRF key K as in Privacy Pass, *every user* needs to receive tokens of the form $(t, \mathsf{PRF}_K(t))$ for the same K, as otherwise the server could recognize users by PRF keys. This actually decreases the efficiency of Privacy Pass, where more than half of the user's computation complexity is for verifying that the server indeed used the "global" PRF key K (cf. Table 1 in [14]), which ensures the user's anonymity. At the same time, it requires users to choose *random* values t, simply to ensure that nobody else already used t for their token request. On the other hand, our token scheme replaces the PRF by a user-specific VRF as $(t, \mathsf{VRF}_{sk}(t))$, where sk is a user's VRF evaluation key, which is certified by a group manager playing the role of the issuing server. However, if we would use a standard VRF with user-specific certified public keys for verification, the token system would not be anonymous. *Group* VRFs, as designed in this paper, allow for *anonymous* yet verifiable evaluations within the group of all users who ever got issued such evaluation secret key (i.e., whoever solved a CAPTCHA). Because every user still evaluates their own VRF, we can now re-use t among users! And with that, we can let the server install redemption policies even after a signing key sk was issued. For example, a server can say "I am now accepting tokens of the format $(t = 2024 - 03 - 19_a, \mathsf{VRF}_{sk}(t))$ with $a \in \{1, ..., 10\}$", thereby limiting the number of tokens that credible users can locally generate for their access requests to 10. On the downside, restricting the first component t of all tokens to come from a small set also causes some leakage: if two tokens with the same t are redeemed, it follows that they must have been generated by different users (cf. Section 8 for further discussions). Besides this new fine-tuning option, token systems from group VRFs achieve constant token issuance complexity on the server-side, because all that needs to be computed is a certified sk.

Our Group VRF Construction. We construct a group VRF by augmenting an asymmetric version of the Dodis-Yampolskiy (DY) VRF [15] with re-randomizable certificates of VRF public keys. Recall that the DY VRF in an

[1] This is a slight simplification: the Privacy Pass protocol does not send w in the clear, but $H(w, con)$ for some context data con that is known to both the server and the browser. This allows to bind the token spending to a specific context.

asymmetric bilinear group with a pairing $e : \mathbb{G}_1, \mathbb{G}_2 \to \mathbb{G}_T$, for a PRF key $sk \in \mathbb{Z}_p$, where $p = |\mathbb{G}_i|$, with corresponding public key $pk := g_1^{sk}$, computes the PRF value of $x \in \mathbb{Z}_p$ as $F_{sk}^{\mathsf{DY}}(x) := e(g_1, g_2)^{1/(x+sk)}$. A proof of correct computation of $F_{sk}^{\mathsf{DY}}(x)$ is given in form of a preimage $\pi := g_2^{1/(x+sk)}$ under the pairing. I.e., the computation was correctly performed with sk if $F_{sk}^{\mathsf{DY}}(x) = e(g_1, \pi)$ and $e(pk \cdot g_1^x, \pi) = e(g_1, g_2)$. We modify the DY PRF in essentially two ways to obtain a group VRF, following the ideas of Ganesh et al. [23] to achieve anonymity through non-uniqueness of public keys.

1) Public keys of the form (g_1, g_1^{sk}) are *signed* under a signing key held by the group manager, and such a signature ("certificate") needs to be attached to each computation and checked upon verification.
2) Public keys (g_1, g_1^{sk}) can be randomized to $(g_1^\tau, g_1^{\tau sk})$. In order to preserve correctness, we change the proof to $g_2^{1/(\tau(x+sk))}$, such that the randomization factor τ cancels out during verification.

While this already outlines our construction, we would like to point out two subtleties when adapting the DY VRF to a group VRF. First of all, adapting the standard version of the DY VRF in the symmetric bilinear group setting, i.e., where $F_{sk}^{\mathsf{DY}}(x) = e(g, g)^{1/(x+sk)}$, would lead to a candidate where anonymity could be trivially broken, as public keys $pk = (g, g^{sk})$, $\tilde{pk} = (g^\tau, g^{\tau sk})$ could be linked via checking $e(g, g^{\tau sk}) \overset{?}{=} e(g^\tau, g^{sk})$.

Second of all, step (1) requires care. Indeed, we cannot use a standard signature scheme, since a "static" signature on a user public key (g_1, g_1^{sk}) would void anonymity guarantees. Hence, we require a signature scheme where signatures can be randomized and adapted to verify under randomized messages (i.e., user public keys in our case). Fuchsbauer et al. [20] construct a signature scheme where signatures can be "mauled" to verify under another (random) message in the same equivalence class as the originally signed message. Plugging in their scheme in step (1) immediately yields our group VRF.

Further, it turns out challenging to prove our construction secure. Intuitively, to prove anonymity the reduction has to be able to generate proofs containing $\left(g_1^\tau, g_1^{\tau\alpha}, g_2^{\frac{1}{\tau(x+\alpha)}}\right)$ without knowing whether $\alpha = sk_0$ or $\alpha = sk_1$, which can be viewed as a combination of the decisional Diffie-Hellman (DDH) assumption over $\mathbb{G}_1$ and a bilinear version of the strong decisional Diffie-Hellman inversion assumption (which allows proving pseudorandomness of the DY PRF [15]). However, since the same challenge α is used for both, merely assuming hardness of the two assumptions is not sufficient.

Instead, we prove our construction secure under a new DDH-type assumption. We note that our VRF is pairing-based, has small proofs for correctness of evaluation and does not use "generic" NIZKs (e.g., Groth-Sahai proofs or Fiat-Shamir based Sigma protocols) for those proofs. For such VRFs, Brandt et al. [9] show that building on complex (non-standard) assumptions or relying on idealized models is unavoidable. We prove our new assumption to hold in the generic group model (GGM) [33], [30] instead of the more realistic algebraic group model (AGM) [21], [32], since the latter seem to only allow a reduction to a *non-standard* DL-type problem in our case. In fact, as a by-product we prove hardness in the GGM for a very broad class of decisional problems in bilinear settings, encompassing Uber-problems [8] as special instance, which might be of independent interest.

Naturally, the existence of a group manager comes with the question of which additional power the group manager has, compared to group members. For building anonymous token system, we require that the group manager cannot break any of the pseudorandomness or anonymity properties that group members enjoy. In particular, we require that group managers, just as group members, cannot lift the anonymity of honest members, link their evaluations, or falsely accuse them of having computed an evaluation. Moreover, our group VRF needs to be *dynamic* in the sense that it does not require to update the group public key whenever members join the group, and it guarantees security in the presence of maliciously generated public keys. We give formal definitions for (dynamic) group VRFs and all its desirable properties, and demonstrate that our group VRF satisfies them. Instantiating our GVRF with the structure-preserving signature scheme of [20] results in the following efficiency: A user public key including its certificate consists of 4 $\mathbb{G}_1$ and 1 $\mathbb{G}_2$ elements. Generating this certificate requires 3 exponentiations in $\mathbb{G}_1$ and 1 in $\mathbb{G}_2$. Verifying it takes 5 pairing evaluations. Computing a GVRF evaluation costs one $\mathbb{G}_T$ exponentiation, computing the VRF proof requires 4 $\mathbb{G}_1$ and 2 $\mathbb{G}_2$ exponentiations. The resulting proof consists of 4 $\mathbb{G}_1$ and 2 $\mathbb{G}_2$ elements. Verifying a GVRF evaluation takes 7 pairing evaluations and 1 $\mathbb{G}_1$ exponentiation.

Related Work. Anonymous VRFs [23] produce verifiable but anonymous pseudorandomness by allowing to transform the verification key in such a way that it still allows for verification, but cannot be linked to other public keys. Evaluation keys can however be generated by anybody, and hence anonymous VRFs do not direclty lend themselves to building anonymous token schemes. Our construction of group VRFs follows the idea of [23] with public key transformation, but additionally introduces a group manager that allows to control the overall set of users that can evaluate the VRF – without breaking the anonymity guarantees. Another related concept are *unique group signatures* [19], which mandate a group manager to control the group, but which do not preserve the anonymity of group members in front of this manager. We provide a more detailed comparison of GVRFs, anonymous VRFs and unique blind signatures in Appendix A.

Building on Privacy Pass [14], several recent works design *anonymous token* systems. Kreuter et al. [29] built anonymous tokens with so-called private metadata bit. These tokens embed a single private bit that is accessible only to the verifying authority. Their construction is DDH-based and uses non-interactive zero-knowledge proofs, which can be dispensed with at the cost of a weak correctness notion. Their construction can be viewed as modifying the oblivious PRF underlying Privacy Pass, the so-called 2Hash Diffie Hellman protocol, to achieve the desirable properties.

Chase et al. [12] revisits the definitions of hidden metadata bits of Kreuter et al. [29] and provides stronger versions, from algebraic MACs instead of PRFs.

Benhamouda et al. [4] build similar token system as Kreuter et al. but with *public* verification, essentially combining [29] with blind Schnorr signatures. Similarly, Silde and Strand [34] build anonymous token systems with *public* metadata that allows to add, e.g., an expiration date to tokens, and that are publicly verifiable. Such tokens can find adoption in contact tracing. Their work also suggests batch revokation methods for Privacy Pass and the token scheme of Kreuter et al., which can be used instead of key rotation at the token verifier (which would be arguably problematic in the case of public verifiability). The anonymous token schemes in our work can have both private and public verifiability. Chu et al. [13] formalise *rate-limited* Privacy Pass, as currently being standardised in IRTF [26], with two new entities called *mediator* and *issuer* to entforce rate-limiting. The anonymity trust assumption is that both parties do not collude. Instead of working with a vOPRF, a blinded signature scheme is used to sanitise tokens for anonymity. While such rate limiting is not the focus of our work, our scheme could be extended with an authentication scheme during token issuance to enforce rate limiting while still keeping verification anonymous without additonal communicational overhead. The disadvantage here is that rate limiting always applies to a set of users.

Potentially closest to our work is the token scheme by Benhamouda et al. [5], which builds so-called *counting* tokens. These tokens allow the verifier to count how many different users obtained a token for a particular context. Formally, this is achieved by (1) different users producing different tokens (as opposed to the one PRF in PrivacyPass), and (2) adding a rate limit that disallows the same user to successfully redeem two tokens on the same message. In our work, we follow a similar strategy and hence our token are also counting. However, while Benhamouda et al. still use an OPRF (the Boneh-Boyen PRF) and require communication per issued token, we replace the OPRF by a group VRF which naturally yields (1) and (2). On top, our GVRF-based token scheme allows to adaptively decide on messages, while their construction requires token contents to be decided prior to online issuance.

Finally, our approach of making token generation local by having a PRF key signed instead of the tokens itself bears resemblance with the construction of compact e-cash [11].

Outline of this paper. Section 3 introduces the concept of GVRF. GVRF-based anonymous tokens are explained in Section 4. Required building blocks and assumptions for GVRFs are described in Section 5. Section 6 presents our GVRF construction. Finally, Section 7 presents our benchmarks and Section 8 deployment considerations.

2. Notation

We will use the following notation. By $\lambda \in \mathbb{N}$ we denote the security parameter. By $x \overset{\$}{\leftarrow} S$ we denote the process of sampling an element x from set S uniformly at random. By $y \leftarrow x$ we denote the process of assigning y the value of x. We say a function is negligible in λ, if its inverse vanishes asymptotically faster than any polynomial in λ. We say that an algorithm A is probabilistic polynomial time (PPT), if A is a probabilistic algorithm with running time polynomial in its input length. We use $y \leftarrow A(x)$ to denote that y is assigned the output of A running on input x.

We consider the adversary in security experiments to be stateful, e.g., keeping all state from prior runs. When writing $\mathcal{A}\colon \perp$ in a non-interactive method, we denote an adversary arbitrarily deviating from the protocol description.

3. Group-Verifiable Random Functions

Our definition of group VRFs is inspired by Groth's dynamic group signatures [24]. We first present the plain definition, which only requires correctness, pseudorandomness and unique provability. Subsequently, we define additional notions such as (weak) unlinkability and unique opening.

In our definition of group verifiable random function, we will assume a group manager (also sometimes referred to as issuer), who can decide who can join the group.

Definition 1 (Group Verifiable Random Function). A *group verifiable random function* (GVRF) is a tuple of PPT algorithms defined as follows.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$ is an algorithm to set up the public parameters pp including the inputs space $\mathcal{X}$, the output space $\mathcal{Y}$. In the following we assume all algorithms to have access to pp without stating it explicitly.
- $(pk_G, sk_G) \leftarrow \mathsf{GroupKG}(pp)$ is a probabilistic algorithm (run by the issuer) that outputs the group public key pk_G and the group secret key sk_G, where the latter is kept private by the issuer.
- $b \leftarrow \mathsf{VerGroup}(pk_G)$ is an algorithm that takes as input a group public key pk_G and outputs a bit $b \in \{0,1\}$.
- $(pk, sk) \leftarrow \mathsf{KG}(pk_G)$ is a probabilistic algorithm run by a user that on input of the group public key pk_G, outputs a key pair (pk, sk).
- $\mathsf{crt} \leftarrow \mathsf{Join}(sk_G, pk)$ is a PPT algorithm executed by the issuer on input sk_G and user public key pk. The algorithm outputs a certificate crt.
- $b \leftarrow \mathsf{VerCert}(pk_G, pk, \mathsf{crt})$ is a deterministic algorithm that on input of the group public key pk_G, the user public key pk and the certificate crt, outputs a bit $b \in \{0,1\}$.
- $(y, \pi, \tau) \leftarrow \mathsf{Eval}(pk_G, pk, sk, \mathsf{crt}, x)$ is an algorithm that on input of the group public key pk_G, the user public key pk and secret key sk, the user value $x \in \mathcal{X}$, and the certificate crt, outputs a value $y \in \mathcal{Y}$, a proof π, and opening information τ.
- $b \leftarrow \mathsf{Ver}(pk_G, x, y, \pi)$ is an algorithm that on input of the group public key pk_G, an input-value pair $x \in \mathcal{X}, y \in \mathcal{Y}$ and a proof π and outputs a bit b.
- $b \leftarrow \mathsf{Judge}(pk_G, pk, x, y, \pi, \tau)$ is an algorithm that takes as input the group public key pk_G, a user public key pk, an input-value pair $x \in \mathcal{X}, y \in \mathcal{Y}$, a proof π, an opening information τ, and outputs a bit b.

A GVRF must satisfy correctness, pseudorandomness and unique provability defined below.

We discuss several aspects of the definition, in relation to the definitions of group signatures. As in [24], our

GVRF is “dynamic” in the sense that users can join the group at any time, without the need to update the group public key pk_G. Opposed to signature verification, algorithm Ver cannot take a user public key as input, since this would violate the desirable anonymity within the group. Hence, Ver needs to verify from only the group public key. Finally, users can decide to lift their anonymity using the Judge algorithm. In this work we focus on this user-centric flavor of deanonymization, but note that other approaches are possible (e.g., where the group manager is given the capability to trace users [24]).

3.1. Basic requirements on GVRFs

In the following we give basic requirements on our GVRF. Note that these correspond essentially to the standard VRF properties lifted to the group setting. In particular, note that our definition of unique verifiability essentially allows to recover the standard notion of unique verifiability by using Judge as verification algorithm. Note that in the following we give the definition relative to a single public key/ secret key pair (pk, sk), which implies that correctness holds for every honestly generated key pair in a group.

Definition 2 (Correctness of a GVRF). A GVRF $(\mathsf{Setup}, \mathsf{GroupKG}, \mathsf{VerGroup}, \mathsf{KG}, \mathsf{Join}, \mathsf{VerCert}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Judge})$ is *correct* if it satisfies the following conditions for all security parameters $\lambda \in \mathbb{N}$, for all $\{\mathcal{X}, \mathcal{Y}\} \subseteq pp$ in the image of $\mathsf{Setup}(1^\lambda)$, for all (pk_G, sk_G) in the image of $\mathsf{GroupKG}(pp)$ for all (pk, sk) in the image of $\mathsf{KG}(pp, 1^\lambda)$ and $\mathsf{crt} \leftarrow \mathsf{Join}(pp, sk_G, pk)$:

- *Correctness of certificates:* It holds

$$\mathsf{VerGroup}(pk_G) = 1 \text{ and } \mathsf{VerCert}(pk_G, pk, \mathsf{crt}) = 1.$$

- *VRF correctness:* For all $x \in \mathcal{X}$, for all (y, π, τ) in the image of $\mathsf{Eval}(pk_G, pk, sk, \mathsf{crt}, x)$, it holds

$$\mathsf{Ver}(pk_G, x, y, \pi) = 1.$$

- *Opening correctness:* For all $x \in \mathcal{X}$, for all (y, π, τ) in the image of $\mathsf{Eval}(pk_G, pk, sk, \mathsf{crt}, x)$, it holds

$$\mathsf{Judge}(pk_G, x, y, \tau, \pi) = 1.$$

We next formalize pseudorandomness of GVRF. The standard notion of pseudorandomness for VRFs is extended to the group setting in the following way: outputs of the GVRF need to appear pseudorandom to every member of the group, and to the group manager holding sk_G.

Definition 3 (Pseudorandomness). A GVRF $(\mathsf{Setup}, \mathsf{GroupKG}, \mathsf{VerGroup}, \mathsf{KG}, \mathsf{Join}, \mathsf{VerCert}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Judge})$ satisfies *pseudorandomness*, if for all admissible PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$ it holds

$$\Pr[\mathsf{Exp}^{\mathsf{pseudorandom}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2} \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pseudorandom}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda)$ is as defined in Figure 1, where we say an adversary is admissible if it provides pk_G and crt such that $\mathsf{VerGroup}(pk_G) = \mathsf{VerCert}(pk_G, pk, \mathsf{crt}) = 1$, and where the randomness is taken over the random coins of Setup, Join and $\mathcal{A}$, as well as the random choices of the bit b and image y_1.

```
$\mathsf{Exp}^{\mathsf{pseudorandom}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda)$:
$\{\mathcal{X}, \mathcal{Y}\} \subseteq pp \leftarrow \mathsf{Setup}(1^\lambda)$
$pk_G \leftarrow \mathcal{A}(pp)$
$\mathcal{Q} \leftarrow \emptyset$, $(sk, pk) \leftarrow \mathsf{KG}(pk_G)$,
$\mathsf{crt} \leftarrow \mathcal{A}(pk)$
$x^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{EvalO}}(\cdot,\cdot)}(1^\lambda)$
$(y, \pi, \tau) \leftarrow \mathsf{Eval}(pk_G, pk, sk, \mathsf{crt}, x^*)$
$y_0 \leftarrow y,\ y_1 \overset{\$}{\leftarrow} \mathcal{Y},\ b \overset{\$}{\leftarrow} \{0,1\}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{EvalO}}(\cdot)}(y_b)$
return $b = b^* \wedge x^* \notin \mathcal{Q}$

$\mathcal{O}_{\mathsf{Eval}}(x, \mathsf{crt})$:
$\mathcal{Q} = \mathcal{Q} \cup \{x\}$
$(y, \pi, \tau) \leftarrow \mathsf{Eval}(pk_G, pk, sk, \mathsf{crt}, x)$
return $(y, \pi)$
```

Figure 1: Pseudorandomness experiment for GVRF. The adversary wins if he can distinguish an evaluation of self-chosen x^* done with respect to some pk from a randomly chosen element from the image space, where only adversaries which provide a valid public key pk_G and valid certificates crt relative to pk are considered.

Next we define unique provability. Recall that unique provability is a property demanded from VRFs. It demands that it is hard to produce two distinct VRF values of the same input both with verifying proofs *relative to the same public key*. Note that we cannot require unique provability relative to the verification Ver, since Ver is defined relative to a group manger, and thus a potential group of public keys.[2] Once keys are opened via Judge to a unique public key though, we can require the usual unique provability. Namely:

Definition 4 (Unique Provability). We say a GVRF $(\mathsf{Setup}, \mathsf{GroupKG}, \mathsf{VerGroup}, \mathsf{KG}, \mathsf{Join}, \mathsf{VerCert}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Judge})$ satisfies *unique provability*, if for all $\lambda \in \mathbb{N}$, all public parameters pp in the image of $\mathsf{Setup}(1^\lambda)$, for all (pk_G, sk_G) in the image of $\mathsf{GroupKG}(pp)$, for all possible public keys pk (i.e., potentially maliciously generated), for all possible certificates crt, for all possible input values x, for all possible function values y_0, y_1 for all possible proofs π_0, π_1 and for all possible opening values τ_0, τ_1 it holds the following: if $\mathsf{Judge}(pk_G, pk, x, y_0, \pi_0, \tau_0) = \mathsf{Judge}(pk_G, pk, x, y_1, \pi_1, \tau_1) = 1$, then $y_0 = y_1$. (In other words, for each $x \in \mathcal{X}$ there exists at most one possible function value y to which x can be opened to under pk. By correctness it is exactly on y, whenever pk is generated honestly.)

3.2. Group-bounded provability

Recall that in the group setting an adversary could have potentially corrupted members of the group, and hence is in possession of an arbitrarily large fraction of the secret keys. For an adversary holding n secret keys (relative to some group defined by pk_G), it is thus easy to produce n images with verifying proofs for the same input value $x \in \mathcal{X}$. We capture the intuition that an adversary should not be able to produce *more* valid images of x (relative to pk_G) in the notion of group-bounded provability.

[2] To make the verification algorithm Ver meaningful for applications, we introduce the notion of “group-bounded provability” below.

```
$\mathsf{Exp}^{\mathsf{gb-prov}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda)$:
$pp \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{ctr} := 0$
$(pk_G, sk_G) \leftarrow \mathsf{GroupKG}(pp)$
$(x^*, y_1, \ldots, y_n, \pi_1, \ldots, \pi_n) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Join}}(\cdot)}(pp, pk_G)$
if $[\forall i \neq j \colon y_i \neq y_j] \wedge [\forall i \colon \mathsf{Ver}(pk_G, x^*, y_i, \pi_i) = 1]$
$\wedge\ \mathsf{ctr} < n$
    return $1$
else return $0$

$\mathcal{O}_{\mathsf{Join}}(pk)$:
$\mathsf{crt} \leftarrow \mathsf{Join}(sk_G, pk)$
$\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
return $\mathsf{crt}$
```

Figure 2: Group-bounded provability experiment for GVRF. The adversary wins if it can provide at least one more evaluation of self-chosen x^* than it posesses secret keys within the group.

```
$\mathsf{Exp}^{\mathsf{(weak\text{-})unlink}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda)$:
$\{\mathcal{X}, \mathcal{Y}\} \subseteq pp \leftarrow \mathsf{Setup}(1^\lambda)$
$pk_G \leftarrow \mathcal{A}(pp)$
$\mathcal{Q} := \emptyset,\ \mathcal{Q}_{\mathsf{open}} := \emptyset$
$(pk_0, sk_0) \leftarrow \mathsf{KG}(pk_G)$
$(pk_1, sk_1) \leftarrow \mathsf{KG}(pk_G)$
$(\mathsf{crt}_0, \mathsf{crt}_1) \leftarrow \mathcal{A}(pk_0, pk_1)$
$b \overset{\$}{\leftarrow} \{0, 1\}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^b_{\mathsf{Eval}}(\cdot), \mathcal{O}^0_{\mathsf{EvalO}}(\cdot), \mathcal{O}^1_{\mathsf{EvalO}}(\cdot)}(1^\lambda)$
return $(b == b^*)\ \wedge(\mathcal{Q} \cap \mathcal{Q}_{\mathsf{open}} == \emptyset)$

$\mathcal{O}^b_{\mathsf{Eval}}(x)$:
$\mathcal{Q} := \mathcal{Q} \cup \{x\}$
$(y, \pi, \tau) \leftarrow \mathsf{Eval}(pk_G, pk_b, sk_b, \mathsf{crt}_b, x)$
return $(y, \pi)$

$\mathcal{O}^0_{\mathsf{EvalO}}(x)$:
$\mathcal{Q}_{\mathsf{open}} := \mathcal{Q}_{\mathsf{open}} \cup \{x\}$
$(y, \pi, \tau) \leftarrow \mathsf{Eval}(pk_G, pk_0, sk_0, \mathsf{crt}_0, x)$
return $(y, \pi)$

$\mathcal{O}^1_{\mathsf{EvalO}}(x)$:
$\mathcal{Q}_{\mathsf{open}} := \mathcal{Q}_{\mathsf{open}} \cup \{x\}$
$(y, \pi, \tau) \leftarrow \mathsf{Eval}(pk_G, pk_1, sk_1, \mathsf{crt}_1, x)$
return $(y, \pi)$
```

Figure 3: (Weak-)Unlinkability experiment for GVRF. The adversary needs to recognize which of two public keys is used for a series of evaluations of self-chosen inputs. In the full unlinkability experiment, the adversary additionally has access to the oracles $\mathcal{O}^0_{\mathsf{EvalO}}, \mathcal{O}^1_{\mathsf{EvalO}}$ highlighted in red. Here, we only consider adversaries which provide a valid public key pk_G and valid certificates crt_β relative to pk_β for $\beta \in \{0, 1\}$.

Definition 5 (Group-bounded provability). We say a GVRF $(\mathsf{Setup}, \mathsf{GroupKG}, \mathsf{VerGroup}, \mathsf{KG}, \mathsf{Join}, \mathsf{VerCert}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Judge})$ has *group-bounded provability* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$ it holds

$$\Pr[\mathsf{Exp}^{\mathsf{gb-prov}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{gb-prov}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda)$ is as defined in Figure 2 and the randomness is taken over the random coins of Setup, Join, and $\mathcal{A}$.

3.3. (Weak) Unlinkability

Next, we define two flavors of unlinkability. Weak unlinkability essentially demands that evaluations of different preimages should not reveal by which group member they were computed. If this holds even if the adversary has already seen an arbitrary set of disjoint evaluations of the group members in question, we refer to it as (full) unlinkability.

Definition 6 ((Weak) Unlinkability). A GVRF $(\mathsf{Setup}, \mathsf{GroupKG}, \mathsf{VerGroup}, \mathsf{KG}, \mathsf{Join}, \mathsf{VerCert}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Judge})$ satisfies *(weak) unlinkability*, if for all admissible PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$ it holds

$$\Pr[\mathsf{Exp}^{\mathsf{(weak\text{-})unlink}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2} \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{(weak\text{-})unlink}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda)$ is as defined in Figure 3, where we say an adversary is admissible if it provides pk_G and $\mathsf{crt}_0, \mathsf{crt}_1$ such that $\mathsf{VerGroup}(pk_G) = 1$ and $\mathsf{VerCert}(pk_G, pk_0, \mathsf{crt}_0) = \mathsf{VerCert}(pk_G, pk_1, \mathsf{crt}_1) = 1$, and the randomness is taken over the random coins of Setup and $\mathcal{A}$, as well as the random choices of the bit b and image y_1.

3.4. Unique opening

Next we define unique opening, which prevents malicious users from claiming ownership of an input/ output pair x, y which was evaluated by an honest user under pk relative to a maliciously generated public key $pk^* \neq pk$.

Definition 7 (Unique Opening). We say a group VRF $\mathsf{GVRF} = (\mathsf{Setup}, \mathsf{GroupKG}, \mathsf{VerGroup}, \mathsf{KG}, \mathsf{Join}, \mathsf{VerCert}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Judge})$ satisfies *unique opening*, if for all $\lambda \in \mathbb{N}$, all public parameters pp in the image of $\mathsf{Setup}(1^\lambda)$, for all (pk_G, sk_G) in the image of $\mathsf{GroupKG}(pp)$, for all possible public keys pk_0, pk_1, for all possible certificates $\mathsf{crt}_0, \mathsf{crt}_1$, for all possible input values x, for all possible function values y for all possible proofs π_0, π_1 and for all possible opening values τ_0, τ_1 it holds that: if $\mathsf{Judge}(pk_G, pk_0, x, y, \pi_0, \tau_0) = \mathsf{Judge}(pk_G, pk_1, x, y, \pi_1, \tau_1) = 1$, then $pk_0 = pk_1$. (In other words, for each input/ output pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ there exists at most one possible public key pk such that (x, y) can be opened under pk.)

Note that for the use in anonymous token schemes the ability to de-anonymize oneself is not required, one could thus also define GVRFs without opening information τ, without a Judge algorithm, and without requiring *unique opening*. Since we believe that the possibility to "claim" an evaluation is useful in contexts like lottery schemes, where one might want to claim a lottery win, and since our instantiation naturally satisfies this additional requirement, we opted for the more general definition.

3.5. Power of the group manager

We want to stress that the only power our group manager has is to decide who can join the group by generating a certificate. He cannot, however, evaluate the VRF relative to a group member (pseudorandomness), link a VRF evaluation to a group member (unlinkability), or frame a group member for a (malicious) evaluation (pseudorandomness & unique provability).

4. Anonymous Tokens from Group VRFs

GVRFs are useful tools for building anonymous access token systems. We now give an algorithmic description of a *policy-based anonymous token scheme*. We aim at a general definition that can capture previous constructions such as Privacy Pass, but that is also suitable to demonstrate particular aspects of GVRF-based access tokens.

- $\mathsf{AT.Setup}(1^\lambda)$ is a global and trusted setup that is run once to generate the (re-usable) public parameters pp.

All parties and algorithms are assumed to have access to pp.

- $\mathsf{AT.ServerSetup}(S\colon pp)$ is run by S once and outputs a secret key sk_S to the server, and the public key pk_S to all parties.
- $\mathsf{AT.VerSSetup}(U\colon pk_S)$ is run by a user (or another entity advocating privacy) to verify the server setup. It outputs a bit $b \in \{0,1\}$.
- $\mathsf{AT.UpdatePolicy}(S\colon \{p_1,\ldots,p_n\})$ for any $n \in \mathbb{N}$ is run by S and sets $\mathcal{P} \leftarrow \{p_1,\ldots,p_n\}$ or outputs $\perp$.
- $\mathsf{AT.Generate}(U\colon pk_S; S\colon sk_S)$ is run between a user U and S. It outputs a pre-token pre or $\perp$ to user U.
- $\mathsf{AT.Expand}(U\colon p, \mathsf{aux}, \mathsf{pre})$ is an algorithm run by the user U that, on input a pre-token pre, auxiliary information aux and an element p from the policy $\mathcal{P}$ outputs a token t and a proof π.
- $\mathsf{AT.Verify}(V\colon t, \mathsf{aux}, \pi, \mathcal{P}, sk_S \,|\, pk_S)$ is a (stateful) algorithm run by V that outputs a bit b. If it is sufficient that the verifier inputs pk_S instead of sk_S we call the scheme *publicy verifiable*.

Policy-based anonymous token scheme. A policy-based anonymous token scheme (pbATS) is run with arbitrarily many users, one issuing server (S in the below) that grants access based on the policy and generated (pre-)tokens, and a verifier V. A pbATS adds a global policy to standard anonymous token schemes that allows the verifier to adaptively control the number of tokens that can be successfully redeemed by a credible user. To model the adaptivity, users first obtain a so-called *pre-token* pre from the issuing server ($\mathsf{AT.Generate}$). According to the current global policy, they can then locally expand that pre-token in several actual tokens ($\mathsf{AT.Expand}$), optionally including auxiliary information aux in the token, e.g., to bind it to a particular context. A change in policy can then allow users to compute more tokens even without interacting with the issuer again. The policy hence allows the verifier to set the security level: a strict policy $\mathcal{P} = \{p_1\}$ allows each user to extract exactly one token from each pre-token, requiring to again contact the issuing server if they want more tokens. A relaxed policy $\mathcal{P} = \{p_1,\ldots,p_n\}$ allows to expand each pre-token into n access tokens, taking off the load from the issuing server and hence enabling faster access to resources.

Intuitively, we want $\mathsf{AT.Verify}(V\colon t, \mathsf{aux}, \pi, \mathcal{P}, sk_S \mid pk_S)$ to output 1 if and only if there exists a policy $p \in \mathcal{P}$ and a pre-token pre generated by $\mathsf{AT.Generate}(U\colon pk_S, S\colon sk_S)$ such that (t,π) was an output of $\mathsf{AT.Expand}(U\colon p, \mathsf{aux}, \mathsf{pre})$ and t has not been used previously. In particular, we want *correctness* (i.e., a honestly generated token will be accepted by the verifier), *unforgeability* (i.e., a user holding k pre-tokens cannot generated more than $k \cdot |\mathcal{P}|$ accepting tokens relative to $\mathcal{P}$) and *unlinkability* (i.e., a server cannot link a pre-token pre with a token t).

Before formally defining these properties, we will explain how Privacy Pass can be viewed as a pbATS. Recall that Privacy Pass is based on a verifiable oblivious pseudorandom function (vOPRF). A vOPRF is a two-party primitive, where a server holds a secret key k, such that the client (user) can evaluate the pseudorandom function $f_k(\cdot)$ on inputs x of her choice, such that the server does not learn anything about the choice x, and the client does not learn anything about the secret key k (except the output $f_k(x)$). Verifiability further guarantess that the client can verify that $f_k(x)$ was correctly evaluated. Now, to obtain a token in Privacy Pass, the user receives $y = f_k(x)$ for an x of her choice. The user then computes $\pi := \mathsf{MAC}_{k'}(\mathsf{aux})$ for $k' := \mathsf{KDF}(y)$, where KDF is some key derivation function that allows to derive a key k' for a message authentication code MAC.[3] To redeem, the user can send (x,π) to the server, which accepts if and only if x is fresh (i.e., has not been queried before) and it can recompute π from x, k and aux. Intuitively, we have unforgeability because of the pseudorandomness of $f_k(\cdot)$ (i.e., generating an accepting token would correspond to guessing the output value on a fresh x') and unlinkability because the server did not learn anything about x at the time of the token generation. With this, it is straightforward to obtain a pbATS. as follows.

- $\mathsf{PPass.Setup}(1^\lambda)$ generates public parameters for the vOPRF used by Privacy Pass.
- $\mathsf{PPass.ServerSetup}(S\colon pp)$ runs the vOPRF setup and returns an vOPRF key $sk_S = k$ to the server and a public commitment pk_S to the key k to all parties.
- $\mathsf{PPass.VerSSetup}(U\colon pk_S)$ is not explicitly supported by Privacy Pass, instead a correct server setup (knowledge of k) is implicitly checked as part of the proof for a valid pre-token.
- Privacy Pass does not support $\mathsf{AT.UpdatePolicy}(S\colon \mathcal{P})$.
- $\mathsf{PPass.Generate}(U\colon pk_S; S\colon sk_S)$ invokes a vOPRF evaluation between the user U and the server S, where the user inputs a randomly chosen message x and the server inputs the vOPRF key $sk_S = k$, and the user obtains the pre-token $(x, f_k(x))$. If the output does not verify, the user outputs $\perp$. Proofs of multiple vOPRF evaluations are batchable.
- $\mathsf{PPass.Expand}(U\colon \perp, \mathsf{aux}, \mathsf{pre})$ parses $\mathsf{pre} =: (x, f_k(x))$, returns the token $t = x$ and proof $\pi = \mathsf{MAC}_{\mathsf{KDF}(y)}(\mathsf{aux})$, i.e., each pre-token gives exactly one token.[4]
- $\mathsf{PPass.Verify}(V\colon t, \mathsf{aux}, \pi, \mathcal{P}, sk_S)$ is run by S, who checks if $\pi = \mathsf{MAC}_{\mathsf{KDF}(f_k(t))}(\mathsf{aux})$ for $k = sk_S$ and t has not been used (note that the policy $\mathcal{P}$ is ignored).

Because each verifying token t requires one interaction with the server, Privacy Pass does not allow the verifier to install policies that would allow to locally generate more tokens through Expand. Furthermore, since access to $sk_S = k$ is required in $\mathsf{AT.Verify}$, Privacy Pass requires $S = V$ to be set, i.e. it is not publicly verifiable. There is an extension that uses a blinded signature scheme instead of a vOPRF to make Privacy Pass publicly verifiable [35]. We note that batching techniques can be applied to render the issuance of n tokens to significantly be more bandwith-efficient and verifier-efficient than n times the cost of $\mathsf{Generate}$, particularly through batching the proof of correct PRF evaluation in the vOPRF. We will later

[3] Note that for the purpose of obtaining a pbATS, one could also consider a simplified version of Privacy Pass without auxiliary input aux, where we set $\pi = f_k(x)$.

[4] Note that one can derive many distinct tuples (x, π_i) from a pre-token pre by using different auxiliary inputs aux_i, but since they all share the same x these would be linkeable by the server. To avoid double-spending the server checks freshness of $t = x$.

take these techniques into account when comparing our $pbATS$ to Privacy Pass.

We now explain how a pbATS $\mathsf{AT}_{\mathsf{GVRF}}$ with a flexible policy can be obtained based on group VRFs. The construction is pretty straightforward: pre-tokens are GVRF user key pairs certified by S who takes the role of the GVRF group manager. Tokens are GVRF evaluations of elements x from the policy under the user secret key. This allows users to locally generate as many tokens as the policy has elements. Token verification equals the verification of a GVRF evaluation, and expiration of all issued pre-tokens and tokens works by generating a fresh GVRF group key pair. More formally, let GVRF denote a group VRF. Then we define a pbATS as follows.

- $\mathsf{AT}_{\mathsf{GVRF}}.\mathsf{Setup}(1^\lambda)$ outputs $pp \leftarrow \mathsf{GVRF}.\mathsf{Setup}(1^\lambda)$, where pp includes a description of the input set $\mathcal{X}$ of the GVRF.
- $\mathsf{AT}_{\mathsf{GVRF}}.\mathsf{ServerSetup}(S\colon pp)$ generates a key pair $(pk_S, sk_S) \leftarrow \mathsf{GVRF}.\mathsf{GroupKG}(pp)$, and outputs sk_S to S and pk_S to all users.
- $\mathsf{AT}_{\mathsf{GVRF}}.\mathsf{VerSSetup}(U\colon pk_S)$ returns the output of $\mathsf{GVRF}.\mathsf{VerGroup}(pk_S)$.
- $\mathsf{AT}_{\mathsf{GVRF}}.\mathsf{UpdatePolicy}(S\colon X)$ for any $X \subset \mathcal{X}$ sets $\mathcal{P} \leftarrow X$.
- To run $\mathsf{AT}_{\mathsf{GVRF}}.\mathsf{Generate}(U\colon pk_S, S\colon sk_S)$, the user generates a key pair $(pk, sk) \leftarrow \mathsf{GVRF}.\mathsf{KG}(pk_S)$ and sends pk to the server. The server generates $\mathsf{crt} \leftarrow \mathsf{Join}(sk_S, pk)$ and sends crt to the user. The user checks if the obtained certificate verifies via $\mathsf{VerCert}(pk_S, pk, \mathsf{crt}) = 1$, if yes it outputs $\mathsf{pre} = (pk_S, sk, \mathsf{crt})$, otherwise it outputs $\perp$.
- To evaluate $\mathsf{AT}_{\mathsf{GVRF}}.\mathsf{Expand}(U\colon x, \perp, \mathsf{pre})$, the user parses the pre-token $\mathsf{pre} =: (pk_S, sk, \mathsf{crt})$, computes the GVRF evaluation $(y, \pi, \tau) \leftarrow \mathsf{Eval}(pk_S, pk, sk, \mathsf{crt}, x)$ and outputs token $t := (x, y)$ and proof π.
- $\mathsf{AT}_{\mathsf{GVRF}}.\mathsf{Verify}(V\colon t, \perp, \pi, \mathcal{P}, pk_S)$ parses $t =: (x, y)$ and outputs 1 if $x \in \mathcal{P}$ and $\mathsf{Ver}(pk_S, x, y, \pi) = 1$ and t has not been used, and 0 otherwise.

Before proving that this indeed yields a pbATS with flexible policy, we formally define desirable properties of a pbATS.

Correctness. For correctness, we require that Expand always produces verifying tokens when run with a fresh element from the policy, a pre-token generated by Generate, an arbitrary auxiliary information. More formally:

Definition 8 (Correctness and freshness of pbATS). We say that a pbATS $\mathsf{AT} = (\mathsf{Setup}, \mathsf{ServerSetup}, \mathsf{VerSSetup}, \mathsf{UpdatePolicy}, \mathsf{Generate}, \mathsf{Expand}, \mathsf{Verify})$ is *correct*, if for all $\lambda \in \mathbb{N}$, $pp \leftarrow \mathsf{Setup}(1^\lambda)$, for all $(pk_S, sk_S) \leftarrow \mathsf{ServerSetup}(S\colon pp)$, we have that $\mathsf{VerSSetup}(U\colon pk_S) = 1$, and for all possible policies $\mathcal{P}$ in the image of UpdatePolicy, for all policy elements $p \in \mathcal{P}$, for all pre-tokens $\mathsf{pre} \leftarrow \mathsf{AT}.\mathsf{Generate}(U\colon pk_S, S\colon sk_S)$, we have that each (t, π) in the image of $\mathsf{AT}.\mathsf{Expand}(U\colon p, \mathsf{pre})$ will pass the verification, i.e., it holds that $\mathsf{AT}.\mathsf{Verify}(V\colon t, \pi, \mathcal{P}, sk_S \,|\, pk_S) = 1$.

We further require *freshness* of the generated token: If $(t, \pi) \leftarrow \mathsf{AT}.\mathsf{Expand}(U\colon p, \mathsf{pre})$ and $(t', \pi') \leftarrow \mathsf{AT}.\mathsf{Expand}(U\colon p', \mathsf{pre}')$ with $p \neq p'$ or $\mathsf{pre} \neq \mathsf{pre}'$ (for $p, p' \in \mathcal{P}$ and honestly generated $\mathsf{pre}, \mathsf{pre}'$), we require $t \neq t'$, except with negligible probability.

```
$\mathsf{Exp}^{\mathsf{unforgeable}}_{\mathsf{AT},\mathcal{A}}(1^\lambda)$:
$ctr := 0$ , $pp \leftarrow \mathsf{Setup}(1^\lambda)$
$(pk_S, sk_S) \leftarrow \mathsf{ServerSetup}(S\colon pp)$
$(\mathcal{P}, (t_i, \pi_i, \mathsf{aux}_i)_{i\in[\ell]}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Gen}}()}(pp, pk_S)$
if $[\forall i \neq j\colon t_i \neq t_j] \wedge \mathsf{ctr} \cdot |\mathcal{P}| < \ell \wedge$
$[\forall i\colon \mathsf{Ver}(V : t_i, \mathsf{aux}_i, \pi_i, \mathcal{P}, sk_S) = 1]$
    return $1$
else return $0$
```

```
$\mathcal{O}_{\mathsf{Gen}}()$:
$\mathsf{pre} \leftarrow \mathsf{AT}.\mathsf{Generate}(\mathcal{A} : \perp, S : sk_S)$
$\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
return $\mathsf{pre}$
```

Figure 4: Unforgeability experiment for anonymous token schemes.

Correctness of our GVRF-based token scheme $\mathsf{AT}_{\mathsf{GVRF}}$ immediately follows from the correctness of the GVRF. If the output space $\mathcal{Y}$ of GVRF is sufficiently large (i.e., of superpolynomial size), then freshness follows from the pseudorandomness of the GVRF.

A note on freshness. Note that when redeeming a token the server only checks if a token has already been spent (which by freshness does not happen for a newly generated token, except with negligible probability). If a user's token has not been spent yet, an adversary getting hold of it may spend it. This is a trade-off for more efficiency, since proving freshness would require an interactive protocol (involving a server challenge). Instead, our tokens are static objects (as in Privacy Pass) and enable faster redemption.

Unforgeability. We require that it is hard to produce a verifying token beyond what the policy allows to expand from pre-tokens obtained from the issuer. More formally:

Definition 9 (Unforgeability pbATS). We say a pbATS $\mathsf{AT} = (\mathsf{Setup}, \mathsf{ServerSetup}, \mathsf{VerSSetup}, \mathsf{UpdatePolicy}, \mathsf{Generate}, \mathsf{Expand}, \mathsf{Verify})$ is *unforgeable*, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$ it holds

$$\Pr[\mathsf{Exp}^{\mathsf{unforgeable}}_{\mathsf{AT},\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{unforgeable}}_{\mathsf{AT},\mathcal{A}}(1^\lambda)$ is as defined in Figure 4.

Unforgeability ensures that cooperating users cannot create fresh tokens, beyond the ones issued by the issuing server. This property corresponds to "One-More-Token security" in Privacy Pass.

Theorem 1 (Unforgeability of $\mathsf{AT}_{\mathsf{GVRF}}$). Let $\mathsf{AT}_{\mathsf{GVRF}}$ as defined above and with GVRF GVRF. If GVRF has group-bounded provability, then $\mathsf{AT}_{\mathsf{GVRF}}$ is unforgeable.

Proof: We explain how a token forger $\mathcal{A}$ against $\mathsf{AT}_{\mathsf{GVRF}}$ yields a group-bounded provability attacker $\mathcal{B}$. The reduction is straightforward and essentially only relays values.

$\mathcal{B}$ obtains pp and pk_G from the group-bounded provability challenger and forwards these to the unforgeability attacker $\mathcal{A}$. When $\mathcal{A}$ makes use of his Generate oracle on a public key pk, $\mathcal{B}$ needs to reply back with the corresponding certificate crt, which he obtains from his Join oracle $\mathcal{O}_{\mathsf{Join}}$ on input pk. Let ctr denote the number of Generate queries by $\mathcal{A}$. Finally, $\mathcal{A}$ outputs $\mathcal{P} := \{p_1, \dots, p_n\}, t_1 := (x_1, y_1), \dots, t_l := (x_l, y_l)$ and proofs $\pi_1, \dots, \pi_\ell$.

```
$\mathsf{Exp}^{\mathsf{unlinkable}}_{\mathsf{AT},\mathcal{A}}(1^\lambda)$:
    $pp \leftarrow \mathsf{Setup}(1^\lambda)$
    $pk_S \leftarrow \mathcal{A}(pp)$
    $\mathsf{pre}_0 \leftarrow \mathsf{AT.Generate}(U_0 : pk_S, \mathcal{A} \colon \bot)$
    $\mathsf{pre}_1 \leftarrow \mathsf{AT.Generate}(U_1 : pk_S, \mathcal{A} \colon \bot)$
    $\mathcal{Q} := \emptyset,\ \mathcal{Q}_{\mathsf{open}} := \emptyset$
    $b \xleftarrow{\$} \{0,1\}$
    $b^* \leftarrow \mathcal{A}^{\mathcal{O}^b_{\mathsf{TokChall}}(\cdot), \mathcal{O}^0_{\mathsf{Tok}}(\cdot), \mathcal{O}^1_{\mathsf{Tok}}(\cdot)}()$
    return $(b == b^*) \wedge (\mathcal{Q} \cap \mathcal{Q}_{\mathsf{open}} == \emptyset)$

$\mathcal{O}^b_{\mathsf{TokChall}}(p, \mathsf{aux})$:
    $\mathcal{Q} := \mathcal{Q} \cup \{p\}$
    $(t, \pi) \leftarrow \mathsf{AT.Expand}(U_b : p, \mathsf{aux}, \mathsf{pre}_b)$
    return $(t, \pi)$

$\mathcal{O}^0_{\mathsf{Tok}}(p, \mathsf{aux})$:
    $\mathcal{Q}_{\mathsf{open}} := \mathcal{Q}_{\mathsf{open}} \cup \{p\}$
    $(t, \pi) \leftarrow \mathsf{AT.Expand}(U_0 : p, \mathsf{aux}, \mathsf{pre}_0)$
    return $(t, \pi)$

$\mathcal{O}^1_{\mathsf{Tok}}(p, \mathsf{aux})$:
    $\mathcal{Q}_{\mathsf{open}} := \mathcal{Q}_{\mathsf{open}} \cup \{p\}$
    $(t, \pi) \leftarrow \mathsf{AT.Expand}(U_1 : p, \mathsf{aux}, \mathsf{pre}_1)$
    return $(t, \pi)$
```

Figure 5: Unlinkable experiment for anonymous token schemes.

Assuming $\mathcal{A}$'s output constitutes a valid forgery, we have $\mathsf{ctr} \cdot n < \ell$ and $t_i \neq t_j$ for all $i \neq j$ and $\mathsf{Ver}(V \colon t_i, \bot, \pi_i, \mathcal{P}, sk_s) = 1$ for all $i \in [\ell]$. Thus, by the pigeonhole principle there must exist a $p^* \in \mathcal{P}$ and subset $J \subseteq [l]$ with $|J| > \mathsf{ctr}$ and $x_j = p^*$ for all $j \in J$. $\mathcal{B}$ then submits $(p^*, (y_j)_{j \in J}, (\pi_j)_{j \in J})$ as forgery.

Since $(x_i, y_i) \neq (x_j, y_j)$ for all $i \neq j$ and $x_i = x_j = p^* \, \forall\, i, j \in J$, it must hold $y_i \neq y_j$ for all $i, j \in J$. Further, $\mathsf{Ver}(V \colon t_i, \mathsf{aux}_i, \pi_i, \mathcal{P}, sk_s) = 1$ for all $i \in [\ell]$ implies $\mathsf{Ver}(pk_S, p^*, y_i, \pi_i) = 1$ for all $i \in J$. Hence, if $\mathcal{A}$ outputs a valid forgery, then $\mathcal{B}$ wins the group-bounded provability experiment. □

Unlinkability. We require that it is hard to link a token to a particular user. In our abstraction of a pbATS, a user is an owner of a pre-token, and hence we demand that it is hard to decide whether two tokens, for policy elements $p \neq p'$, were expanded from the same pre-token, or different ones. We require unlinkability to hold even against a *malicious* issuer of pre-tokens. More formally:

Definition 10 (Unlinkability pbATS). We call a pbATS $\mathsf{AT} = (\mathsf{Setup}, \mathsf{ServerSetup}, \mathsf{VerSSetup}, \mathsf{UpdatePolicy}, \mathsf{Generate}, \mathsf{Expand}, \mathsf{Verify})$ *unlinkable*, if for all admissible PPT adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$ it holds

$$|\Pr[\mathsf{Exp}^{\mathsf{unlinkable}}_{\mathsf{AT},\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{unlinkable}}_{\mathsf{AT},\mathcal{A}}(1^\lambda)$ is as defined in Figure 5 and an adversary is admissible if it outputs pk_S with $\mathsf{VerSSetup}(pk_S) = 1$ and $\mathsf{pre}_i \neq \bot$ for $i \in \{0, 1\}$.

Theorem 2 (Unlinkability of $\mathsf{AT}_{\mathsf{GVRF}}$***).*** Let $\mathsf{AT}_{\mathsf{GVRF}}$ as defined above with GVRF. If GVRF is fully unlinkable (cf. Def. 6), then $\mathsf{AT}_{\mathsf{GVRF}}$ is unlinkable.

Proof: We explain how an unlinkability adversary $\mathcal{A}$ against $\mathsf{AT}_{\mathsf{GVRF}}$ yields an attacker $\mathcal{B}$ against the full unlinkability of GVRF. The reduction is straightforward since $\mathcal{B}$ essentially only forwards values between the unlinkability challenger and $\mathcal{A}$. More detailed, $\mathcal{B}$ obtains public parameters pp and runs $\mathcal{A}$ to obtain pk_S. Upon obtaining pk_0, pk_1 from the challenger, $\mathcal{B}$ sends pk_0, pk_1 to $\mathcal{A}$ during the two Generate executions, and receives back certificates $\mathsf{crt}^*_0, \mathsf{crt}^*_1$, which it checks and passes to the challenger. The token generation oracle $\mathcal{O}^i_{\mathsf{Tok}}()$ is implemented using oracle $\mathcal{O}^1_{\mathsf{EvalO}}(\cdot)$. Finally, $\mathcal{A}$'s challenge oracle $\mathcal{O}^b_{\mathsf{TokChall}}(p)$ is implemented using $\mathcal{B}$'s oracle $\mathcal{O}^b_{\mathsf{Eval}}(\cdot)$, and $\mathcal{A}$'s decision bit is adopted by $\mathcal{B}$. Clearly, $\mathcal{B}$ is successful if and only if $\mathcal{A}$ is. □

5. Building Blocks

In this section we recall the definition of signatures on equivalence classes, and introduce the one-more bilinear DDH/DDHI assumption, on which our construction relies.

5.1. Signatures on Equivalence Classes

For our construction we require a randomizable signature scheme. To this end, we recall the definition of [20] of signatures on equivalence classes in the following. Let $\mathbb{G}_1$ be a group of prime order p, and $\mathbb{G}_1^* := \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$. For vectors $\vec{u} \in \mathbb{G}_1^\ell$, where $\ell \in \mathbb{N}$, and $\rho \in \mathbb{Z}_p$ we denote by $\vec{u}^\rho$ the pointwise exponentiation $(u_1^\rho, \ldots, u_\ell^\rho)$. Then, the equivalence relation we consider in this paper is of the form

$$\mathcal{R}_{\mathsf{DDH}} := \{(\vec{u}, \vec{v}) \in (\mathbb{G}_1^*)^\ell \times (\mathbb{G}_1^*)^\ell \mid \exists \rho \in \mathbb{Z}_p^* \colon \vec{v} = \vec{u}^\rho\},$$

where in this paper we always have $\ell = 2$. For $\vec{u} \in (\mathbb{G}_1^*)^\ell$, this relation defines the equivalence class

$$[\vec{u}]_{\mathcal{R}_{\mathsf{DDH}}} := \{\vec{v} \in (\mathbb{G}_1^*)^\ell \mid \exists \rho \in \mathbb{Z}_p^* \colon \vec{v} = \vec{u}^\rho\}.$$

In the following we will typically consider $\ell = 2$.

We recall the definition of signatures on equivalence classes of [20]. Note that in [20] such signatures are referred to as *structure-preserving signatures*, but as the definition does not explicitly required the signature to be structure-preserving, we will simply refer to such signatures as *signatures on equivalence classes* in the following. Further, we will fix ℓ in advance (rather than giving it as parameter to BGGen), since in the paper we will only consider $\ell = 2$.

Definition 11 (Signatures on Equivalence Classes ([20], Def. 15)). An ℓ-dimensional EQ-$\mathcal{R}$ signature scheme consists of a tuple of algorithms $\mathsf{SIG} = (\mathsf{BGGen}_{\mathcal{R}}, \mathsf{KG}_{\mathcal{R}}, \mathsf{Sign}_{\mathcal{R}}, \mathsf{ChRep}_{\mathcal{R}}, \mathsf{Vfy}_{\mathcal{R}})$ such that the following holds:

$\mathsf{BGGen}_{\mathcal{R}}(1^\lambda)$**:** Is a probabilistic polynomial-time algorithm that on input of the security parameter 1^λ outputs $\mathsf{BG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p, g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$, and $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map.

$\mathsf{KG}_{\mathcal{R}}(\mathsf{BG})$**:** Is a probabilistic alogrithm, which on input of a bilinear group BG and dimension ℓ outputs a key pair (pk, sk).

$\mathsf{Sign}_{\mathcal{R}}(sk, \vec{u})$**:** Is a probabilistic polynomial time algorithm, which on input of a secret key sk and a representative $\vec{u} \in (\mathbb{G}_1^*)^\ell$, outputs a signature σ.

$\mathsf{ChRep}_{\mathcal{R}}(pk, \vec{u}, \sigma, \rho)$**:** Is a probabilistic algorithm, which on input of a public key pk, representative $\vec{u} \in (\mathbb{G}_1^*)^\ell$, signature σ and scalar $\rho \in \mathbb{Z}_p^*$, computes a new representative $\vec{v} = \vec{u}^\rho$ and an updated signature $\hat{\sigma}$, and outputs $(\vec{v}, \hat{\sigma})$.

$\mathsf{Vfy}_{\mathcal{R}}(pk, \vec{u}, \sigma)$**:** Is a deterministic algorithm, which on input of a public key pk, representative $\vec{u}$ and signature σ outputs a bit $b \in \{0, 1\}$.

```
$\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathsf{SIG},\mathcal{A}}(1^\lambda)$:
$\mathsf{BG} \leftarrow \mathsf{BGGen}_\mathcal{R}(1^\lambda)$
$(pk, sk) \leftarrow \mathsf{KG}_\mathcal{R}(\mathsf{BG})$
$\mathcal{Q} := \emptyset$
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sign}}(\cdot)}(\mathsf{BG}, pk)$
if $\forall m \in \mathcal{Q}$: $[m^\star]_\mathcal{R} \neq [m]_\mathcal{R} \wedge \mathsf{Vfy}(pk, m^\star, \sigma^\star) = 1$
    return $1$
else return $0$

$\mathcal{O}_{\mathsf{sign}}(m)$:
$\mathcal{Q} := \mathcal{Q} \cup \{m\}$
$\sigma \leftarrow \mathsf{Sign}(sk, m)$
return $\sigma$
```

Figure 6: Unforgeability experiment for signatures on equivalence classes.

$\mathsf{VKey}_\mathcal{R}(pk, sk)$: Is a deterministic algorithm, which on input of a public key pk and secret key sk outputs a bit $b \in \{0, 1\}$.

We further require the scheme to satisfy *correctness*, *EUF-CMA security* and *perfect signature adaptation* as defined in the following.

Definition 12 (Correctness). We say an ℓ-dimensional EQ-$\mathcal{R}$ signature scheme $\mathsf{SIG} = (\mathsf{BGGen}_\mathcal{R}, \mathsf{KG}_\mathcal{R}, \mathsf{Sign}_\mathcal{R}, \mathsf{ChRep}_\mathcal{R}, \mathsf{Vfy}_\mathcal{R})$ is *correct*, if for all security parameters $\lambda \in \mathbb{N}$, for all bilinear groups BG in the image of $\mathsf{BGGen}_\mathcal{R}(1^\kappa)$, for all key pairs (pk, sk) in the image of $\mathsf{KG}_\mathcal{R}(\mathsf{BG})$, for all $\vec{u} \in (\mathbb{G}^*)^\ell$, for all σ in the image of $\mathsf{Sign}_\mathcal{R}(sk, \vec{u})$, it holds

$$\mathsf{VKey}_\mathcal{R}(pk, sk) = 1 \text{ and } \mathsf{Vfy}_\mathcal{R}(pk, \vec{u}, \sigma) = 1.$$

Further, for all $\rho \in \mathbb{Z}_p^*$, we require

$$\mathsf{Vfy}_\mathcal{R}(pk, \vec{v}, \hat{\sigma}) = 1,$$

where $(\vec{v}, \hat{\sigma}) \leftarrow \mathsf{ChRep}_\mathcal{R}(pk, \vec{u}, \sigma, \rho)$ and $\vec{v} = \vec{u}^\rho$.

Unforgeability for EQ-$\mathcal{R}$ signature schemes demands that it be hard to forge a verifying signature on an adversarially-chosen message m^* from a *fresh* equivalence class. Note that it is easy to produce fresh signatures using the $\mathsf{ChRep}_\mathcal{R}$ algorithm on a signature obtained from the signing oracle. However, $\mathsf{ChRep}_\mathcal{R}$ is "class preserving", meaning that it can only produce signatures for messages in the same equivalence class as the original message. Hence, in the EUF-CMA experiment, the adversary is challenged to produce a forgery for a message from an equivalence class for which it has never received a signature from its oracle. We will use EUF-CMA security in our construction to allow the group manager to control the size of participant set (i.e., key holders) in a group.

Definition 13 (EUF-CMA security). We say an EQ-$\mathcal{R}$ signature scheme $\mathsf{SIG} = (\mathsf{BGGen}_\mathcal{R}, \mathsf{KG}_\mathcal{R}, \mathsf{Sign}_\mathcal{R}, \mathsf{ChRep}_\mathcal{R}, \mathsf{Vfy}_\mathcal{R})$ satisfies *unforgeability*, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$

$$\Pr[\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathsf{SIG},\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathsf{SIG},\mathcal{A}}$ is as defined in Figure 6.

To prove anonymity we further need that freshly generated signatures are indistinguishable from re-randomized signatures, which is captured by the notion of *perfect signature adaptation.*

Definition 14 (Perfect signature adaptation). Let $\ell > 1$. We say an ℓ-dimensional EQ-$\mathcal{R}$ signature scheme $\mathsf{SIG} = (\mathsf{BGGen}_\mathcal{R}, \mathsf{KG}_\mathcal{R}, \mathsf{Sign}_\mathcal{R}, \mathsf{ChRep}_\mathcal{R}, \mathsf{Vfy}_\mathcal{R}, \mathsf{VKey}_\mathcal{R})$ *perfectly adapts signatures*, if for all $\lambda \in \mathbb{N}$, all bilinear groups BG in the image of $\mathsf{BGGen}_\mathcal{R}(1^\lambda)$ and all tuples $(sk, pk, \vec{u}, \sigma, \mu)$ with

$$\mathsf{VKey}(pk, sk) = 1,\ \vec{u} \in (\mathbb{G}_1^*)^\ell,\ \mathsf{Vfy}_\mathcal{R}(pk, \vec{u}, \sigma) = 1,$$
$$\mu \in \mathbb{Z}_p^*$$

the outputs of $\mathsf{ChRep}_\mathcal{R}(pk, \vec{u}, \sigma, \mu)$ and $\mathsf{Sign}_\mathcal{R}(sk, \mu \cdot \vec{u})$ are identically distributed.

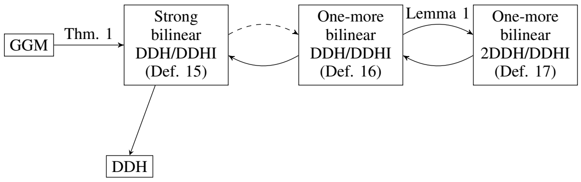
GGM
Thm. 1
Strong bilinear DDH/DDHI (Def. 15)
One-more bilinear DDH/DDHI (Def. 16)
Lemma 1
One-more bilinear 2DDH/DDHI (Def. 17)
DDH

Figure 7: Implications of the various assumptions defined in this work. Dashed arrows hold only if the challenge space of the implied assumption is restricted to polynomial size.

Instantiating Signatures on Equivalence Classes. Fuchsbauer et al. [20] give an instantiation of signature schemes on equivalence classes (Scheme 1, [20]) in the generic group model, where public keys consist of two group elements over $\mathbb{G}_2$, and signatures consist of two group element over $\mathbb{G}_1$ plus one group element over $\mathbb{G}_2$. Computing the re-randomization of a signature requires three group exponentiation (+ two group exponentiations to compute the re-randomized message), and verifications costs five pairing evaluations.

5.2. The One-More Bilinear DDH/DDHI Assumption

For proving anonymity of our construction, we rely on a "one-more" type assumption, which can be viewed as a strenghtening of bilinear variants of both the DDH and SDDHI assumption [10], [19] (although it seems incomparable to the latter). We start with a non-interactive version of this assumption, where we require the challenge space $\mathcal{X} \subset \mathbb{Z}_p$ to be of polynomial-size. We refer to this as the strong bilinear DDH/DDHI assumption. In our full version [17], we show that this assumption holds in the generic group model.

We refer the reader to Figure 7 for an overview of the cryptographic assumptions presented in this subsection, and the relations between them.

Definition 15 (Strong Bilinear DDH/DDHI Assumption). Let $\mathcal{G}$ be a asymmetric bilinear group generator returning groups $\mathsf{BG} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, p, g_1, g_2, e)$. Let $\mathcal{X} \subset \mathbb{Z}_p$ be a polynomial-sized set. Then, we say that the *strong bilinear DDH/DDHI assumption with respect to challenge space* $\mathcal{X}$ is hard relative to $\mathcal{G}$, if for every PPT adversary $\mathcal{A}$ and every polynomial q there exists a negligible function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$ it holds

$$\Pr\left[\mathcal{A}\left(\begin{matrix}\mathsf{BG}, \{(x, g_1^{\tau_{x,i}}, g_1^{\alpha\tau_{x,i}}, g_2^{\frac{1}{\tau_{x,i}(\alpha+x)}})\}_{x\neq x^*, i\in[q]}, \\ g_1^\alpha, z_b\end{matrix}\right) = b\right]$$
$$\leq \frac{1}{2} + \mathsf{negl}(\lambda), \text{where}$$

$$\mathsf{BG} \leftarrow \mathcal{G}(1^\lambda), b \xleftarrow{\$} \{0,1\}, \alpha, \tau, y \xleftarrow{\$} \mathbb{Z}_p, x^* \xleftarrow{\$} \mathcal{X},$$
$$\forall x \in \mathcal{X} \setminus \{x^*\}, i \in [q] \colon \tau_{x,i} \xleftarrow{\$} \mathbb{Z}_p,$$
$$z_0 := (g_1^\tau, g_1^{\alpha\tau}, g_2^{\frac{1}{\tau(\alpha+x^*)}}), z_1 := (g_1^\tau, g_1^y, g_2^{\frac{1}{y+\tau\cdot x^*}})$$

Theorem 1 (Hardness of Strong Bilinear DDH/DDHI in GGM). Let $\mathcal{P}$ be the strong bilinear DDH/DDHI problem from Definition 15. Then any generic group algorithm sending at most t queries to the generic group oracle can solve $\mathcal{P}$ with advantage at most $\epsilon \leq (2560(|\mathcal{X}|q+1)^5(t\log(p)+t+1)^2)/p$.

For a proof of this theorem we refer to our full version [17].

Relation to other assumption. Our assumption is a combination of the bilinear decisional Diffie-Hellman (BDDH) assumption over $\mathbb{G}_1$, stating that given g_1^α, g_1^τ, it is hard to distinguish $g_1^{\alpha\tau}$ from random, and a bilinear version of the SDDHI assumption [10], [19] (in the following referred to as BDDHI), where the adversary has to distinguish $e(g_1,g_2)^{\frac{1}{\alpha+x^*}}$ from random and gets oracle access to an oracle $\mathcal{O}_{\mathsf{proof}}$, which outputs $g_2^{\frac{1}{\alpha+x}}$ for any $x^* \neq x$. Intuitively, the only thing an adversary can do to break the new assumption is to pair elements and check for equality. To rule out such attacks, we slightly adapt the assumptions and show that the resulting combined assumption is hard in the GGM. In particular, we "weaken" the oracle of the BDDHI security assumption, as the oracle would allow an adversary to trivally break BDDH. In this sense, the assumption is incomparable to the BDDHI security assumption, but in spirit our assumption can be viewed as a strengthening of both.

One-more bilinear DDH/DDHI assumption. To remove the requirement of $\mathcal{X}$ being polynomial-sized, we introduce an interactive version of the assumption, where the adversary gets to choose the challenge $x^\star$ after seeing g_1^α, which we refer to as one-more bilinear DDH/DDHI. It is easy to see that this is implied by the non-interactive variant when the challenge space is restricted to polynomial-sized $\mathcal{X} \subset \mathbb{Z}_p$, since the reduction can simply guess the challenge x^* ahead of time, and implement $\mathcal{O}_{\mathsf{proof}}$ using $\{(x, g_1^{\tau_x}, g_1^{\alpha\tau_x}, g_2^{\frac{1}{\tau_x(\alpha+x)}})\}_{x\neq x^*}$.

Definition 16 (One-More Bilinear DDH/DDHI Assumption). Let $\mathcal{G}$ be a asymmetric bilinear group generator returning groups $\mathsf{BG} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$. Then, we say that the *one-more DDH/DDHI assumption* is hard relative to $\mathcal{G}$, if for every admissible PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a negligible function $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$ it holds

$$\Pr\left[\mathcal{A}_1^{\mathcal{O}_{\mathsf{proof}}(\cdot)}(\mathsf{state}, z_b) = b\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda), \text{ where}$$

$$\mathsf{BG} \leftarrow \mathcal{G}(1^\lambda), b \xleftarrow{\$} \{0,1\}, \alpha, \tau, y \xleftarrow{\$} \mathbb{Z}_p, Q := \emptyset,$$
$$(\mathsf{state}, x^*) \leftarrow \mathcal{A}_0^{\mathcal{O}_{\mathsf{proof}}(\cdot)}(\mathsf{BG}, g_1^\alpha)$$
$$z_0 := (g_1^\tau, g_1^{\alpha\tau}, g_2^{\frac{1}{\tau(\alpha+x^*)}}), z_1 := (g_1^\tau, g_1^y, g_2^{\frac{1}{y+\tau\cdot x^*}}) \text{ and}$$

where $\mathcal{O}_{\mathsf{proof}}(x)$ on the i-th query samples $\tau_i \leftarrow \mathbb{Z}_p$, outputs $(g_1^{\tau_i}, g_1^{\alpha\tau_i}, g_2^{\frac{1}{\tau_i(\alpha+x)}})$ and sets $Q := Q \cup \{x\}$. We say a PPT adversary $(\mathcal{A}_0, \mathcal{A}_1)$ is admissible if $x^* \notin Q$.

To show anonymity of our construction we rely on a variant of the above assumption, which allows to simulate proofs relative to two secret keys α and $\tilde{\alpha}$. We state the assumption and show that it is implied by the one-more bilinear DDH/DDHI assumption in the Appendix F.

Definition 17 (One-More Bilinear 2-DDH/DDHI Assumption). Let $\mathcal{G}$ be a asymmetric bilinear group generator returning groups $\mathsf{BG} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$. Then, we say that the *one-more 2-DDH/DDHI assumption* is hard relative to $\mathcal{G}$, if for every admissible PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a negligible function $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}_{>0}$, such that for all $\lambda \in \mathbb{N}$ it holds

$$\Pr\left[\mathcal{A}_1^{\mathcal{O}^0_{\mathsf{proof}}(\cdot), \mathcal{O}^1_{\mathsf{proof}}(\cdot)}(\mathsf{state}, z_b) = b\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda), \text{ where}$$

$$\mathsf{BG} \leftarrow \mathcal{G}(1^\lambda), b \xleftarrow{\$} \{0,1\}, \alpha_0, \alpha_1, \tau, y \xleftarrow{\$} \mathbb{Z}_p, Q := \emptyset,$$
$$(\mathsf{state}, x^*) \leftarrow \mathcal{A}_0^{\mathcal{O}^0_{\mathsf{proof}}(\cdot), \mathcal{O}^1_{\mathsf{proof}}(\cdot)}(\mathsf{BG}, g_1^{\alpha_0}, g_1^{\alpha_1})$$
$$z_b := (g_1^\tau, g_1^{\alpha_b\tau}, g_2^{\frac{1}{\tau(\alpha_b+x^*)}}),$$

and where for $\beta \in \{0,1\}$ $\mathcal{O}^\beta_{\mathsf{proof}}(x)$ on the i-th query samples $\tau_{i,\beta} \leftarrow \mathbb{Z}_p$ and outputs $\{(g_1^{\tau_{i,\beta}}, g_1^{\alpha_\beta\tau_{i,\beta}}, g_2^{\frac{1}{\tau_{i,\beta}(\alpha_\beta+x)}})\}$ and sets $Q := Q \cup \{x\}$. We say a PPT adversary is admissible, if it outputs $x^* \notin Q$.

Lemma 1. If the one-more bilinear DDH/DDHI assumption (Def. 16) is hard relative to $\mathcal{G}$, then so is the one-more bilinear 2-DDH/DDHI assumption (Def. 17).

6. Construction

We propose a GVRF based on the VRF by Dodis and Yampolskiy [15], adapted to asymmetric bilinear group with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$, combined with a suitable signature scheme over equivalence classes. The construction is given in Figure 8. The main idea is as follows. A GVRF evaluation on input x is essentially a VRF evaluation of the Dodis-Yampolskiy VRF, i.e., $y := e(g_1,g_2)^{1/(x+sk)}$ is an evaluation of x under VRF public key $pk := g_1^{sk}$, and $\pi' := g_2^{1/(x+sk)}$ is the proof of correct evaluation. Using the latter one can verify correctness of y with respect to pk by means of the equations (1) $e(g_1^x \cdot pk, \pi') = e(g_1, g_2)$ and (2) $e(g_1, \pi') = y$. To enforce membership in a group, a group manager could sign pk, and evaluations are only deemed valid if they (a) verify w.r.t π' and (b) a valid signature on pk is presented. There is one remaining problem with this approach: if a user would now produce evaluations of multiple inputs, they would become linkable through pk. We prevent this by a conceptually simple measure: we let users *randomize* their public keys and the group manager's signature on their public key, such that they become unlinkable to computationally bounded attackers (the randomization term is τ in Figure 8). This step requires us to let the group manager sign using a signature scheme on equivalence classes, where each equivalence class then corresponds to the public keys of one user. However, due the randomization of pk, also the DY PRF verification equations need to adapted. We extend the public key by an additional component g_1 (keeping track of the randomization) and also adapt the VRF proof π' in order to get rid of the randomization factor τ in the

```
$\mathsf{Setup}(1^\lambda)$:
$\mathsf{BG} \leftarrow \mathsf{SIG.BGGen}_{\mathcal{R}}(1^\lambda)$
$\mathsf{BG} =: (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, p, g, e)$
$\mathsf{crs} \leftarrow \mathsf{PS.PGen}(\mathsf{BG})$
$\mathcal{X} := \mathbb{Z}_p$
$\mathcal{Y} := \mathbb{G}_3$
Output $pp := (\mathsf{BG}, \mathsf{crs}, \mathcal{X}, \mathcal{Y})$

$\mathsf{GroupKG}(pp)$:
$(pk^{\mathsf{SIG}}, sk^{\mathsf{SIG}}) \leftarrow \mathsf{SIG.KG}(\mathsf{BG})$
$\pi_G \leftarrow \mathsf{PS.PPrv}(\mathsf{crs}, pk^{\mathsf{SIG}}, sk^{\mathsf{SIG}})$
$pk_G := (pk^{\mathsf{SIG}}, \pi_G)$
$sk_G := sk^{\mathsf{SIG}}$
Output $(pk_G, sk_G)$

$\mathsf{VerGroup}(pk_G)$
Parse $pk_G =: (pk^{\mathsf{SIG}}, \pi_G)$
$b \leftarrow \mathsf{PS.Ver}(\mathsf{crs}, pk^{\mathsf{SIG}}, \pi_G)$
Output $b$

$\mathsf{KG}(pk_G)$:
$sk \xleftarrow{\$} \mathbb{Z}_p$
$pk_1 := g_1,\ pk_2 := g_1^{sk}$
Output $((pk_1, pk_2), sk)$

$\mathsf{Join}(sk_G, pk)$:
$\mathsf{crt} \leftarrow \mathsf{SIG.Sign}_{\mathcal{R}}(sk_G, pk)$
Output $\mathsf{crt}$

$\mathsf{VerCert}(pk_G, pk, \mathsf{crt})$
Parse $pk_G =: (pk^{\mathsf{SIG}}, \pi_G)$
$b \leftarrow \mathsf{SIG.Vfy}_{\mathcal{R}}(pk_G, pk, \mathsf{crt})$
Output $b$

$\mathsf{Eval}(pk_G, pk, sk, \mathsf{crt}, x)$:
Parse $pk_G =: (pk^{\mathsf{SIG}}, \pi_G)$
$\tau \xleftarrow{\$} \mathbb{Z}_p^*$
$y := e(g_1, g_2)^{1/(x+sk)}$
$\pi' := g_2^{1/(\tau(x+sk))}$
$(\tilde{pk}, \tilde{\mathsf{crt}}) \leftarrow \mathsf{SIG.ChRep}_{\mathcal{R}}(pk^{\mathsf{SIG}}, pk, \mathsf{crt}, \tau)$
Output $(y, (\pi', \tilde{pk}, \tilde{\mathsf{crt}}), \tau)$

$\mathsf{Ver}(pk_G, x, y, \pi)$:
Parse $pk_G =: (pk^{\mathsf{SIG}}, \pi_G)$
Parse $\pi =: (\pi', \tilde{pk}, \tilde{\mathsf{crt}})$
Parse $\tilde{pk} =: (\tilde{pk}_1, \tilde{pk}_2)$
If $\mathsf{SIG.Vfy}_{\mathcal{R}}(pk^{\mathsf{SIG}}, \tilde{pk}, \tilde{\mathsf{crt}}) = 1 \wedge$
  $e((\tilde{pk}_1)^x \cdot \tilde{pk}_2, \pi') = e(g_1, g_2) \wedge e(\tilde{pk}_1, \pi') = y$
  output $1$
Else output $0$

$\mathsf{Judge}(pk_G, pk, x, y, \pi, \tau)$:
Parse $\pi =: (\pi', \tilde{pk}, \tilde{\mathsf{crt}})$
If $\tilde{pk} = pk^\tau \wedge \mathsf{Ver}(pk_G, pk, x, y, \pi) = 1$
  output $1$
Else output $0$
```

Figure 8: Construction $\mathsf{GVRF} = (\mathsf{Setup}, \mathsf{GroupKG}, \mathsf{VerGroup}, \mathsf{KG}, \mathsf{Join}, \mathsf{VerCert}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Judge})$ based on the Dodis-Yampolskiy VRF [15]. Recall that all algorithms are assumed to have access to $pp = (\mathsf{BG}, \mathsf{crs}, \mathcal{X}, \mathcal{Y})$.

VRF verification equations. This adaption is not generic but specific to the instance of the VRF [15] and the EQ-$\mathcal{R}$ signature [20] we consider.

Note that to ensure an honest setup of the group manager's signature key, we make use of a non-interactive zero-knowledge proof of knowledge PS proving that $(pk^{\mathsf{SIG}}, sk^{\mathsf{SIG}})$ are of the form as specified by $\mathsf{SIG.VKey}_{\mathcal{R}}$. This proof π_G is generated in GroupKG and verified in VerGroup. That means, it only needs to be computed once by the group manager when it generates a new key pair, and it only needs to be verified once by a user for this new key pair (or this job can even be outsourced to an entity users trust). Thus, the efficiency of PS is not that crucial (unless pk_G is very short-lived). The verification of the group key setup is essential in untrusted environments involving a potentially malicious group manager as in our pseudorandomness and unlinkability experiments. More precisely, from a security proof perspective, PS is needed as the reduction will need sk^{SIG} for generating fresh signatures, which is obtained by using the extractability of PS.

We achieve the following security result for our GVRF, proven formally in Appendix E.

Theorem 3. Let $\mathcal{G}$ be a bilinear group generator, such that the one-more bilinear DDH/DDHI assumption (Def. 16) is hard relative to $\mathcal{G}$. Let $\mathcal{R} := \mathcal{R}_{\mathsf{DDH}}$ be the DDH relation over $\mathbb{G}_1^2$ and let $\mathsf{SIG} = (\mathsf{BGGen}_{\mathcal{R}}, \mathsf{KG}_{\mathcal{R}}, \mathsf{Sign}_{\mathcal{R}}, \mathsf{ChRep}_{\mathcal{R}}, \mathsf{Vfy}_{\mathcal{R}}, \mathsf{VKey}_{\mathcal{R}})$ be a 2-dimensional EQ-$\mathcal{R}_{\mathsf{DDH}}$ signature scheme that satisfies perfect signature adaptation. Further, let $\mathsf{PS} = (\mathsf{PGen}, \mathsf{PTGen}, \mathsf{PPrv}, \mathsf{PVer}, \mathsf{PSim}, \mathsf{PExt})$ be a non-interactive zero knowledge proof of knowledge for the relation defined by $\mathsf{SIG.VKey}_{\mathcal{R}}$. Then, the GVRF given in Figure 8 is a group VRF with unlinkability. Further, if SIG additionally satisfies EUF-CMA security (and the proof system PS satisfies zero knowledge as before), the GVRF also satisfies group-bounded provability. Finally, the GVRF satisfies unique opening unconditionally.

7. Implementation

Set-up. To evaluate the practical feasibility of our pbATS, we compare the execution times of Privacy Pass and GVRF. Both schemes were implemented as multi-threaded C++23 applications, focusing solely on cryptographic operations and payload sizes. The client runs on a notebook with an AMD Ryzen 7 PRO 5875U @ 4.50 GHz, 32 GB RAM and Ubuntu 22.04.1 LTS (kernel 6.5), while the server uses an AMD EPYC 9274F 24-Core @ 4.3 GHz, 128 GB RAM, and Ubuntu 22.04.2 LTS (kernel 6.8.0), utilizing all CPU cores. The client, by contrast, limits thread usage to simulate resource-constrained devices.

We use RELIC Toolkit v0.7.0 [2] to instantiate GVRFs over BLS12-381 and Privacy Pass over NIST P-256, both offering 128-bit security. GVRF also employs the equivalence-class signature scheme from [20]. As to the best of our knowledge RELIC does not offer fast P-256 support for x86, we benchmark both schemes without faster optimizations. Optimized GVRF results are included separately under "GVRF*" in the Tables. The code is available under Appendix G.

Token Generation. Privacy Pass clearly has the overall performance advantage when examining the entire token generation process on both the client and server side in Table 1, but places a heavier load on the issuing servers in terms of computation and bandwidth. For example, the entire token generation of 25 tokens takes 18.33 ms for Privacy Pass, compared to 97.24 ms for GVRF and 56.76 ms for GVRF*. Doubling the number of tokens roughly doubles the execution time in the case of Privacy Pass.

Further, the results show that server-side AT.Generate is independent of the number of tokens for GVRF and takes a constant 435 μs and for GVRF* 163 μs with a total of 307 bytes to transmit. The request consists of 2 $\mathbb{G}_1$ elements, namely the public key, which is 102 bytes, and the response is a certificate on this public key, which is just an SPS-EQ signature consisting of 2 $\mathbb{G}_1$ and 1 $\mathbb{G}_2$ element, and a global policy of 2 $\mathbb{Z}_p$ elements. In total the response is 205 bytes. GVRF outperform Privacy Pass with a 10-token batch, where GVRF* is comparable with the server-side issuance time of 2 Privacy Pass tokens, being more bandwith and computational efficient here. For token sizes greater than 1, GVRF performs better in both, through the constant communicational and computational costs, since one issuing request is required for a client to generate tokens as many as the policy allows. Looking at the server-side throughput of 10 token batches in Table 1: GVRF outperforms Privacy Pass by a factor of 2 and GVRF* by a factor of almost 6 with a constant number of 2300 issuing req/s for GVRF and 6104 issuing req/s for GVRF*. However, on the client side, since expanding

TABLE 1: "**Token generation** costs (AT.Generate() + AT.Expand(), where AT.Expand() can be performed offline) on client and server-side for Privacy Pass (PP), GVRF (G), and GVRF* (G*), averaged over 100k runs. Timings are in ms; throughput (issuing requests per second (issue req/s)). Network latency is excluded, as both protocols require only one communication round. Communication costs are given in bytes and inspects the payload size only. **Best performance** in each row is highlighted in **bold**. (Note that for token generation time per client, we mark both best overall time (PP) and best online time (G* for $>$ 1 token) in **bold**.)"

No. of Tokens	Token Generation Per Entity (ms)						Communication Costs (bytes)				Throughput (issue requests/s)		
	Server			Client			Server		Client		Server		
	PP	G	G*	PP	G	G*	PP	G/G*	PP	G/G*	PP	G	G*
1	**0.081**	0.435	0.163	**1.1** + **0.006**	2.8 + 3.8	1.6 + 2.2	**107**	205	**37**	102	**12300**	2300	6104
10	0.838	0.435	**0.163**	**4.1** + **0.06**	2.8 + 38	**1.6** + 22	404	**205**	334	**102**	1193	2300	**6104**
25	1.98	0.435	**0.163**	**16.2** + **0.15**	2.8 + 95	**1.6** + 55	899	**205**	829	**102**	505	2300	**6104**
50	4.07	0.435	**0.163**	**31** + **0.3**	2.8 + 190	**1.6** + 110	1724	**205**	1654	**102**	246	2300	**6104**
100	8.47	0.435	**0.163**	**61** + **0.6**	2.8 + 380	**1.6** + 220	3374	**205**	3304	**102**	118	2300	**6104**

TABLE 2: "Average web-server-side **verification** AT.Verify() times (ms) for Privacy Pass (PP), GVRF (G) and GVRF* (G*), with communication costs and server throughput (verification requests per second). Throughput reflects cryptographic operations only. Communication costs (in bytes) inspect the payload size only. Excluding network latency. **Best performance** in each row is highlighted in **bold**."

Verify (ms)			Communication (bytes)		Throughput (verify requests/s)		
PP	G	G*	PP	G/G*	PP	G	G*
0.094	7	4	**71**	781	**10 600**	142	244

a token by using pairings is cryptographically expensive, it takes 3.8ms per token generation for GVRF and 2.2ms for GVRF*.

Expanding tokens from pre-tokens is done on client-side and can be done offline and on-demand. Here, a delay of 4ms is still an imperceptible for a human. For both protocols, multiple executions of AT.Expand are independent and can happen in parallel to speed up computation. Furthermore, a signing request for 50 pre-tokens in Privacy Pass transmits 1.654 kB of data and the response 1.724 kB, increasing with the number of tokens, because elliptic curve points equal to the number of tokens must be sent in the request and response. In AT.Expand, on the other hand, Privacy Pass is able to use a keyed hash function based on SHA-256, which outperforms expensive public key operations in GVRF. The token size is 71 bytes for Privacy Pass and 781 bytes for GVRF, since it consists of multiple group elements. Thus, GVRF require 11 times more memory than Privacy Pass.

Token Verification. Overall, Privacy Pass is performing significantly better in the verification process than GVRF. For Privacy Pass it takes 0.01 ms to verify a token, for GVRF 7 ms and for GVRF* 4 ms. As the latter is way slower than Privacy Pass and puts more load on the server-side, 7ms and 4ms are still acceptable by distributing the load across multiple verification servers. When looking at the server throughput, where Privacy Pass achieves 10 600 verification req/s, GVRF 142 verification req/s and GVRF* 244 verification req/s, GVRF-based AT needs multiple servers to keep up with Privacy Pass. Token verification requires transmitting 4 $\mathbb{G}_1$, 2 $\mathbb{G}_2$, 1 $\mathbb{G}_T$, 1 $\mathbb{Z}_p$ elements for GVRF, which are 781 bytes in our implementation. On the other hand, Privacy Pass requires transmitting 1 $\mathbb{Z}_p$ element, 1 MAC (32bytes), which were 71 bytes in our benchmarks. Here, Privacy Pass clearly outperforms GVRF communication and computation-wise.

8. Deployment Considerations

In this section, we outline key considerations for deploying GVRF-based pbATS as an alternative to Privacy Pass. GVRF-based pbATS offer a different tradeoff: reduced computation for issuing servers at the cost of more effort for clients and verifiers. Also unlinkability is weaker, but support updatable policies, which Privacy Pass does not.

Performance tradeoff. As backed by our benchmarks in Section 7, GVRF-based schemes enable lightweight token issuance for servers, avoiding bottlenecks when few servers handle issuance. However, clients face higher computational costs, and verification is more complex than in Privacy Pass. This makes GVRF-based tokens less suitable when verification is more centralized.

The tradeoff is favorable when issuance is centralized but verification is distributed—e.g., motivated by the deployment models in Sections 4.2, 4.4, and 5.2 in RFC 9576 [27]. Centralizing issuers reduces key exposure risks, since sharing the same secret key is required, while distributed verifiers can share public keys only and only need a distributed database to prevent double-spending.

GVRF-based schemes seem to be well suited for such an environment as communication and server-side costs in the token issuance process are constant and comparable to the issuance process of a single token in Privacy Pass (cf. Table 1), even though the client is able to extend its pre-token to not only one but as many tokens as the policy allows. Moreover, the number of issuing requests can further be reduced by using the policy update feature which enables clients already in possession of a valid pre-token to generate new tokens without any interaction. On the other hand, the high verification times of GVRF tokens need to be compensated by distributing the load to a high number of verifying servers if client-behavior does not already induce an appropriate distribution.

Limits of our unlinkability notion. In the unlinkability experiment we capture that an adversary cannot distinguish whether two accepted tokens (p, t, π) and (p', t', π') for policy elements $p \neq p'$ were generated by the same user or two different users. This does not prevent leakage based on the used policy elements though.

First of all, it is important that users choose the policy element at random from the set of available policy elements. Otherwise, if users use tokens in a deterministic sequence (i.e., $p_1, p_2, \ldots$, according to some ordering), an adversary corrupting both the issuing server and a set of websites can potentially link queries based on the used policy, as they give an ordering on the redeemed tokens. In an extreme case, where a single user makes more requests than any other user, the latest token redemptions by this user can be fully linked based on the advanced policy elements.

Even with this mitigation, the policy elements give some leakage. Namely, if $p = p'$, our scheme reveals that two accepted tokens must stem from two different anonymous users (in the following we refer to this as "unlinking" leakage). This kind of leakage is inherent to our definition, where verifying a token requires to know the exact p, as unforgeability requires that a user cannot generate more than one accepting token relative to the same p to allow for rate limiting.

Whether the unlinking leakage is acceptable requires a careful case study depending on the setting. We believe that this kind of leakage is not significant in large-scale settings, such as Cloudflare's with 32M requests per second, where the probability that two requests stem from different users is high in any case. To see this, consider the following simplified setting, where, say, all servers accept tokens relative to some fixed policy elements $\{p_1, \ldots, p_{100}\}$, and there are $100 \cdot N$ token redemptions in total. What an adversary corrupting both the issuing server and all redemption servers can now do, is to split up all redemption queries (consisting of a token and the redemption sever) into 100 buckets based on the used policy element, which is part of the token. As we assume that policies are chosen at random, we expect all buckets to roughly contain N redemption queries (which, for simplification, we assume to be exactly N in the following). Now, if we look at a single user, we know that he can have made at most one redemption query in each bucket (assuming he only made one pretoken query), as he cannot have redeemed two tokens corresponding to the same policy element p. As he could have made 0 or 1 out of N possible redemption queries in each bucket, this gives $N+1$ possible choices per bucket (corresponding to no redemption, redemption corresponding to redemption query 1, …, redemption corresponding to redemption query N), resulting in $(N+1)^{100}$ possibilities in total. While this leakage can be combined with additional leakage (such as timing leakage, which can also be made in a fully unlinkable setting), we believe that the achieved unlinkability gives a high level of individual privacy whenever the bucket sizes are expected to be large. Additionally, redemption servers that receive many queries are expected to occur in all buckets evenly, therefore not allowing for any leakage.

In small-scale settings, however, this leakage can be an issue. For instance, assume the extreme case of a size-2 policy and 2-user setting with $N = 4$ websites: If both users generate tokens relative to the same p_1, p_2 to visit websites W_1, W_2 and W_1', W_2', respectively. Then, an adversary corrupting both the issuing server and websites can deduce that both users visited exactly two websites and that either W_1 and W_2 or W_1 and W_2' were visited by the same user (and similarly for W_1'), corresponding to an entropy loss of 50%.

We note that the leakage in our construction could be avoided by p not being part of the token, but a range proof, that proves that p is within the policies interval. Since this slows down verification even further, we accepted this leakage in our construction. Privacy Pass, on the other hand, achieves a stronger notion of unlinkability, where two requests can neither be linked to the same, nor to different users, and is thus the preferable choice in small-scale or other settings where the unlinking leakage is not acceptable.

Policies and their usage (cf. Appendix D for details). Policies in our GVRF-based pbATS instantiation are public subsets $\mathcal{P} \subset \mathbb{Z}_p = \{0, \ldots, p-1\}$, where p is a large prime. As for each $x \in \mathcal{P}$, a user can generate exactly one token using its pre-token (by evaluation with x) which can only be spent once, $|\mathcal{P}|$ determines the number of tokens currently available per pre-token. $\mathcal{P}$ can be represented by an interval $[a, b] \subset \mathbb{Z}_p$, where it suffices to make a and the length of the interval known to the users.

In this way, one can easily enforce global policies, i.e., fixing a maximum number of tokens which can be used by each pre-token holder and redeemed at all websites. It also allows to realize more advanced policies like time-, website-, content-, or service-dependent policies by encoding time stamps or IDs as prefix of $\mathbb{Z}_p$ values. For instance, the policies $\mathcal{P}_{9999} = [999901, 999930]$ and $\mathcal{P}_{8888} = [888801, 888810]$ allow a pre-token holder to redeem 30 tokens at the website with ID 9999 and only 10 at the website with ID 8888, respectively. Combinations of different policy types are also possible. Note that we do not support user-specific policies as Rate-limited Privacy Pass [13] does at the cost of introducing a trusted mediator entity.

Another important feature are policy updates as a means to react to the current or expected network situation by *retrospectively* decreasing or increasing the number of unspent tokens in circulation. For instance, if for the current policy there is an overload of token-based website requests, we can decrease the total number of remaining tokens by decreasing the size of the policy interval.

In Privacy Pass, not inherently supporting the concept of updatable policies, the effects of advanced policies can only be approximated in an inefficient way, e.g., by frequently rotating pk_S (time-dependent policies) or running several instances of Privacy Pass in parallel (website-/content-/service-dependent policies). In general, with Privacy Pass we cannot really implement the main goal of policy updates, i.e., to retrospectively adjust the number of unspent tokens. All we can do is to invalidate all unspent tokens at once by renewing the issuing key. On the other hand, a potential downside of policy updates affecting *all* holders of valid pre-tokens is the following: As long as pk_S stays the same, we cannot selectively adapt only the size of *newly* issued token batches without also adjusting the size of all previous ones, which however, is easily possible in Privacy Pass.

References

[1] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 649–666. Springer, Berlin, Heidelberg, August 2011.

[2] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient Library for Cryptography. https://github.com/relic-toolkit/relic, 2020.

[3] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Compact e-cash and simulatable VRFs revisited. In Hovav Shacham and Brent Waters, editors, *PAIRING 2009*, volume 5671 of *LNCS*, pages 114–131. Springer, Berlin, Heidelberg, August 2009.

[4] Fabrice Benhamouda, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Publicly verifiable anonymous tokens with private metadata bit. *IACR Cryptol. ePrint Arch.*, page 4, 2022.

[5] Fabrice Benhamouda, Mariana Raykova, and Karn Seth. Anonymous counting tokens. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 245–278. Springer, Singapore, December 2023.

[6] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch Groth-Sahai. In Jianying Zhou and Moti Yung, editors, *ACNS 10International Conference on Applied Cryptography and Network Security*, volume 6123 of *LNCS*, pages 218–235. Springer, Berlin, Heidelberg, June 2010.

[7] Jan Bobolz, Fabian Eidens, Stephan Krenn, Daniel Slamanig, and Christoph Striecks. Privacy-preserving incentive systems with highly efficient point-collection. In Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese, editors, *ASIACCS 20*, pages 319–333. ACM Press, October 2020.

[8] Xavier Boyen. The uber-assumption family (invited talk). In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 39–56. Springer, Berlin, Heidelberg, September 2008.

[9] Nicholas Brandt, Dennis Hofheinz, Julia Kastner, and Akin Ünal. The price of verifiability: Lower bounds for verifiable random functions. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 747–776. Springer, Cham, November 2022.

[10] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 201–210. ACM Press, October / November 2006.

[11] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 302–321. Springer, Berlin, Heidelberg, May 2005.

[12] Melissa Chase, F. Betül Durak, and Serge Vaudenay. Anonymous tokens with stronger metadata bit hiding from algebraic MACs. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 418–449. Springer, Cham, August 2023.

[13] Hien Chu, Khue Do, and Lucjan Hanzlik. On the security of rate-limited privacy pass. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 2871–2885. ACM, 2023.

[14] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018.

[15] 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, Berlin, Heidelberg, January 2005.

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

[17] Dennis Faut, Julia Hesse, Lisa Kohl, and Andy Rupp. Scalable and fine-tuned privacy pass from group verifiable random functions. Cryptology ePrint Archive, Paper 2025/659, 2025.

[18] Valerie Fetzer, Max Hoffmann, Matthias Nagel, Andy Rupp, and Rebecca Schwerdt. P4TC - provably-secure yet practical privacy-preserving toll collection. *Proc. Priv. Enhancing Technol.*, 2020(3):62–152, 2020.

[19] Matthew K. Franklin and Haibin Zhang. Unique group signatures. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 643–660. Springer, Berlin, Heidelberg, September 2012.

[20] 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.

[21] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018.

[22] Ruti Gafni and Idan Nagar. Captcha – security affecting user experience. *Issues in Informing Science and Information Technology*, 13:63–77, 01 2016.

[23] Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. Proof-of-stake protocols for privacy-aware blockchains. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 690–719. Springer, Cham, May 2019.

[24] Jens Groth. Fully anonymous group signatures without random oracles. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 164–180. Springer, Berlin, Heidelberg, December 2007.

[25] 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, Berlin, Heidelberg, April 2008.

[26] Scott Hendrickson, Jana Iyengar, Tommy Pauly, Steven Valdez, and Christopher A. Wood. Rate-Limited Token Issuance Protocol. Internet-Draft draft-ietf-privacypass-rate-limit-tokens-06, Internet Engineering Task Force, April 2024. Work in Progress.

[27] Alex Davidson Jana Iyengar and Christopher A. Wood. The Privacy Pass Architecture. Internet-Draft draft-ietf-privacypass-architecture, Internet Engineering Task Force, June 2024. Completed.

[28] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 581–600. Springer, Berlin, Heidelberg, March 2006.

[29] 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, Cham, August 2020.

[30] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Berlin, Heidelberg, December 2005.

[31] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Berlin, Heidelberg, August 1992.

[32] Lior Rotem and Gil Segev. Algebraic distinguishers: From discrete logarithms to decisional uber assumptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 366–389. Springer, Cham, November 2020.

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

[34] 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, Cham, May 2022.

[35] Sofia Celi Alex Davidson Steven Valdez and Christopher A. Wood. Privacy Pass Issuance Protocols. Internet-Draft draft-ietf-privacypass-protocol, Internet Engineering Task Force, June 2024. Completed.

Appendix A. Related Cryptographic Primitives

In this section we contextualize group VRFs among related cryptographic primitives. Ganesh et al. [23] were the first to introduce the concept of *anonymous VRFs*. Their central idea is that anonymity requires non-uniqueness of public keys, such that each evaluation w.r.t a specific PRF key can be verified under a *different* public key. Their construction for the PRF $H(x)^{sk}$ is secure in the random oracle model under the DDH assumption. However, as for standard VRFs, anybody can generate a VRF key pair and hence there is no built-in way to control the anonymity set. Ganesh et al. [23] suggest to circumvent this drawback by proving in zero-knowledge that a randomized public key was generated from a public list of keys. In our work, we suggest a new notion of anonymous VRFs which captures the anonymity set. Namely, we generalize their model to the group setting, allowing us to control the amount of PRF keys in the system through a group manager. While we follow the idea of [23] to introduce non-unique public keys, we select the DY PRF as underlying PRF and demonstrate that it can be augmented with anonymity guarantees in the standard model, without relying on generic NIZKs.

Unique group signatures are a concept that is closely related to group VRFs. A notable difference is that signatures do not need to be pseudorandom, and thus certain applications such as lotteries work better with VRFs than signatures. Franklin and Zhang [19] introduce unique group signatures, and build them from VRFs, committed PRF keys, and NIZK proofs of correct computation [3]. While their construction relies on similar assumptions to ours, it is significantly less efficient in terms of signature/ image and proof size. Further, Franklin et al. [19] require (and allow) the group manager to deanonymize signers while we aim at avoiding such a central and powerful entity.

Finally, numerous works explicitly equip the Dodis-Yampolskiy PRF with anonymity, e.g., [18], [11], [7], through usage of generic NIZK proofs of correct computation, without revealing the public key. Our group VRF might be plugged into these constructions to obtain more efficient variants of these systems.

We provide a detailed comparison of related schemes' properties and costs in Table 3. The table shows all cryptographic primitives in the literature that allow to produce verifiable pseudorandomness in an anonymous fashion. The crucial difference is the *control over the anonymity set*, which does not exist in anonymous VRFs as presented in [23]. Our work shows that we can add such control. While this was previously already demonstrated in [19] through the concept of unique group signature, our work provides two significant improvements: first, we demonstrate that control of the anonymity set can be added (in terms of a group manager) but *without* sacrificing the anonymity property to the group manager, as is the case in [19]. Second, we greatly improve upon the efficiency of [19], which relies on computationally heavy tools such as Groth-Sahai proofs [24] and tag-based CCA-secure encryption [28]. We further note that the dynamic construction in [19] does not come with a formal security proof.

	Anonymous	Group VRFs	Unique group
	VRFs [23]	**this work**	signatures [19]
Pseudorandomness	✓	✓	(✓)
Anonymity	✓	✓	✓
Dynamic (adaptive joining)	✓	✓	✓
Controlled anonymity set	✗	✓	✓
Anon. against group manager	-	✓	✗
Security against malicious pk	✓	✓	✓

TABLE 3: Cryptographic primitives for producing verifiable but anonymous pseudorandomness. Group VRFs are the only primitive that allow to control the anonymity set by an authority without revealing the evaluator's identity to that authority. The "-" for [23] is because they do not require a group manager. Unique group signatures are not necessarily pseudorandom, but the dynamic construction of [19] is.

Appendix B. Non-Interactive Zero-Knowledge Proofs

In the following we give a definition of non-interactive zero knowledge proofs of knowledge based on [25], where we additionally require a knowledge extractor. Note that the latter can be instantiated by adding an encryption of the witness and extending the proof system to the corresponding language. The trapdoor will then contain a key to decrypt.

Definition 18. Let $\mathcal{R}$ be a efficiently computable relation. A non-interactive zero-knowledge proof system consists of a tuple of PPT algorithms $\mathsf{PS} := (\mathsf{PGen}, \mathsf{PTGen}, \mathsf{PPrv}, \mathsf{PVer}, \mathsf{PSim}, \mathsf{PExt})$ of the following syntax.

- $\mathsf{PGen}(pp)$ on input of the public parameters pp generates a common reference string crs.
- $\mathsf{PTGen}(pp)$ on input of the public parameters pp generates a common reference string crs and additionally a trapdoor td.
- $\mathsf{PPrv}(\mathsf{crs}, x, w)$ on input of a common reference string crs, statement x and witness w with $(x, w) \in \mathcal{R}$ outputs a proof π.
- $\mathsf{PVer}(\mathsf{crs}, x, \pi)$ on input of a common reference string crs, statement x and proof π outputs a bit $b \in \{0, 1\}$.
- $\mathsf{PSim}(\mathsf{crs}, \mathsf{td}, x)$ on input a common reference string crs with trapdoor td and a statement x outputs a proof π.
- $\mathsf{PExt}(\mathsf{crs}, \mathsf{td}, x, \pi)$ on input a common reference string crs with trapdoor td and a statement x with proof π outputs a witness w.

We further require the following to hold.

- **Completeness:** For all public parameters pp, for all $(x, w) \in \mathcal{R}$, we have $\mathsf{PVer}(\mathsf{crs}, x, \pi) = 1$.

- **Zero knowledge:** For all public parameters pp, we have that the distribution of $\mathsf{crs} \leftarrow \mathsf{PGen}(pp)$ and crs obtained via $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{PTGen}(pp)$ are computationally indistinguishable.

  For all statements $x \in \mathcal{L}$ with w such that $(x, w) \in \mathcal{R}_{\mathcal{L}}$ we further have that $\mathsf{PPrv}(\mathsf{crs}, x, w)$ and $\mathsf{PSim}(\mathsf{crs}, \mathsf{td}, x)$ are identically distributed for $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{PTGen}(1^\lambda)$.
- **Knowledge soundness:** For all public parameters pp, for all PPT adversaries $\mathcal{A}$ for $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{PTGen}(pp)$, for all statements x, for $\pi \leftarrow \mathcal{A}(pp, \mathsf{crs})$ the following holds: If $\mathsf{PVer}(\mathsf{crs}, x, \pi) = 1$, then the extractor yields a witness $w \leftarrow \mathsf{PExt}(\mathsf{crs}, \mathsf{td}, x, \pi)$ such that $(x, w) \in \mathcal{R}$, except with negligible probability. Here, the probability is taken over the random coins of $\mathsf{PTGen}, \mathcal{A}$ and PExt.

Appendix C. GVRF from NIZK

Combining a PRF with a NIZK proof is often the approach that is taken in privacy-preserving protocols to obtain an unlinkable and verifiable PRF. Our GVRF instantiation is significantly more efficient than such a construction. Of course, the exact performance gap highly depends on the concrete approach and chosen primitives. For concreteness, let us consider an approach where the proof of correct PRF evaluation $F_{sk}(x) = y$ is for a language of the following form $\{(x, y) \mid \exists\ sk, C, r, \sigma \text{ s.t } C = \mathsf{Com}(sk; r), \mathsf{Ver}(pk^{\mathsf{SIG}}, \sigma, C) = 1, F_{sk}(x) = y\}$, where pk^{SIG} is the signature verification key of an authority (e.g., the group manager for GVRFs). When instantiating the above using the DY-PRF [15], Pedersen commitments [31], the optimal structure-preserving signature scheme by ABE [1], and (SXDH-based) Groth-Sahai NIZKs [25], we obtain the following estimates based on Figure 13 in [16] and Table 1 in [6]: generating a proof requires about 90 $\mathbb{G}_1$ and 60 $\mathbb{G}_2$ exponentiations, verifying it about 120 pairing evaluations. This compares to 4 $\mathbb{G}_1$ and 2 $\mathbb{G}_2$ exponentiations for proof generation and 7 pairing evaluations plus one $\mathbb{G}_1$ exponentiation for proof verification in our GVRF construction.

Appendix D. Policies and Their Usage

Using the instantiation of a GVRF proposed in Section 6, the policies $\mathcal{P}$ of our pbATS scheme are subsets of $\mathbb{Z}_p = \{0, \ldots, p-1\}$, where p is the prime order of a bilinear group setting. The size of $\mathcal{P}$ determines the number of tokens available to each user holding a pre-token valid under the current pk_S. Policies are public sets that do not need to contain random numbers, but may consist of consecutive numbers. Hence, $\mathcal{P}$ might be represented by an interval $[a, b] \subset \mathbb{Z}_p$. In this case, to fix a policy it suffices to make a and the length of the interval public (e.g., by transmitting it to the user after solving a CAPTCHA, just before token verification, etc.).

Basic policy types. Using this approach, several basic policy types can be implemented, including:

- *Global policies* fix a maximum number of tokens which can be used by each pre-token holder and redeemed at all websites for all possible contents and services until the policy gets updated. A global policy can be defined by fixing an interval $[a, b] \subset \mathbb{Z}_p$.
- *Time-dependent policies* fix a maximum number of tokens which can be used by each pre-token holder within a certain period of time. To define such a policy, the idea is to encode rough fixed-length time stamps ts (e.g., in the format YYYY-MM-DD-HH) as "prefix" of a $\mathbb{Z}_p$ value and append a fixed-length counter, i.e., $\mathcal{P}_{ts,n} := \{ts||\mathsf{pad}(c) \mid 1 \leq c \leq n\} = [ts||\mathsf{pad}(1), ts||n] \subset \mathbb{Z}_p$, where $\mathsf{pad}(c)$ is a representation of c using the same number of digits as n (i.e., zero-digits are prepended if needed). For instance, $\mathcal{P} = [202503101301, 202503101330]$ allows each pre-token holder to spend 30 tokens on March 10th, 2025, between 13:00 and 13:59. Clearly, the token verifying entities will need to automatically update the policy they accept according to the current time. If we assume an agreement on a common time zone and that the clocks of users are roughly synchronized with those of the verifying entities, it suffices to only transfer the current n to the user to fix a policy. Note that for a new policy of size n, each pre-token holder will be able to redeem n tokens independent of how many tokens it spent wrt. the previous policy.
- *Website/content/service-dependent policies* fix website-, content-, or service-specific maximum numbers of tokens which can be used by each pre-token holder. For instance, it could be useful to allow pre-token holders to redeem more tokens (i.e., make a larger number of requests without CAPTCHA) for static, small-sized content than for costly dynamically-generated (e.g., by API requests) or large-size content (e.g., software downloads). To define such policies, we can follow a similar approach as for time-dependent policies: we assign fixed-length unique IDs to websites, content or service types, which are then encoded as "prefix" of a $\mathbb{Z}_p$ value and append by a fixed-length counter, i.e., $\mathcal{P}_{\mathsf{ID},n} := [\mathsf{ID}||\mathsf{pad}(1), \mathsf{ID}||n]$. Note that when a pre-token holder has redeemed all its tokens associated with $\mathcal{P}_{\mathsf{ID}_i,n_i}$, it is still able to redeem tokens associated with a different policy $\mathcal{P}_{\mathsf{ID}_j,n_j}$ without being forced to request a new pre-token (i.e., solve a CAPTCHA).

Combination of basic policies. Obviously, it could also be useful to combine certain basic policies. For instance, we can easily obtain time-dependent policies for different content types by just prepending IDs to the definitions of the time-dependent intervals. Another interesting use-case could be combining a global policy with content-dependent policies in the sense that a pre-token owner has an overall limit of tokens it can spent as well as content-specific limits. For this purpose, we enforce that every time a content-specific token gets spent, also a global token needs to be spent and the request is only accepted if both tokens have not been spent before. Note that both types of tokens can be generated using the same pre-token.

Policy updates. Policy updates are a means to react to the current or expected network situation. For instance, if for the current policy there is an overload of token-based requests, we can decrease the total number of remaining tokens by adjusting the size of the policy interval. Similarly, if we start with a very small-sized policy, but the webservers could easily handle more requests in the current situation, then we can replace the policy with a larger one. As a change of policy does not render pre-tokens invalid, all current pre-token holders can generate new tokens according to the new policy without interacting with the issuing server. There are different possibilities to adjust the current policy, including:

- Update by *disjunct set*: Independently of the number of tokens a pre-token holder already redeemed for the old policy, it can redeem a number of tokens equal to the size of the new policy.
- Update by *subset*: This is, e.g., obtained when decreasing the upper bound of the interval of a policy. In this case, pre-token holders may redeem a number of tokens with the updated policy which is at most equal to the size of the new policy (as it may have already spent tokens before associated with the new smaller interval). Note that the operator can count the total number of remaining tokens over all pre-token holders which would be available for a subset policy.
- Update by *superset*: This is, e.g., obtained when increasing the upper bound of the policy interval. Again, pre-token holders may redeem at most a number of tokens equal to the size of the new policy and at least equal to the difference of the sizes of new and old policy. Also, one can count how many unspent tokens would be available when doing such an update.

Invalidation of pre-tokens. Updating policies results in new tokens for *all* pre-token holders. If at some point, the number of pre-token holders is considered too large, all pre-tokens can be invalidated at once by changing the key pair (sk_S, pk_S) of the issuing server. Our current construction does not support expiration dates for pre-tokens to invalidate them automatically. In order to avoid a peak of pre-token issuing requests by users whose pre-tokens have just been invalidated, one can still use the old server public key (in parallel to the new one) for the verification of tokens for a certain transitioning period while the new server secret key is used for issuing new pre-tokens. The policy used for the old tokens should then stay the same or could be cut in size to speed-up the transitioning process. As soon as the holder of an old pre-token redeemed all tokens according to the old policy it will contact the issuing server to the a new pre-token.

Simulating policies with Privacy Pass. In Privacy Pass, we do not have the concept of updatable policies as described above. However, let us consider if and how the effects of policies can be simulated by Privacy Pass.

First, observe that our construction with a fixed non-updatable global policy of size n is essentially functionally equivalent to Privacy Pass issuing a fixed batch of n tokens. In both cases, each user can redeem n tokens for any resource before the issuing server need to be contacted again, in our system to get a fresh pre-token and in Privacy Pass to get a new batch.

To realize separate token spending limits per solved CAPTCHA for individual websites, contents, or services, separate issuing (OPRF) keys would be required in parallel. Also, a user would need to explicitly request tokens for a particular resource which is not the case in our system.

To approximate the effects of time-dependent policies, i.e., during the current time period only token batches (of size dependent on the period) issued during the same period are valid, one would need to change the issuing key for each time period and adjust the size of issued token batches accordingly. As the change of the issuing key invalidates tokens from the previous period, all users need to first interact with the issuing server again. In our system, old tokens get invalidated by switching to the new policy interval which can then be used by all pre-token holders, where the pre-token could have been issued in the current or any previous period, to generate tokens for the new period.

In general, with Privacy Pass we cannot really implement the main goal of policy updates, i.e., to retrospectively decrease or increase the number of unspent tokens in circulation. All we can do to decrease the number of unspent tokens is to renew the issuing key, which invalidates every unspent token. On the other hand, changing the number of issued tokens by means of a policy update in our system, does not only affect new issuing requests but always also all previous ones. Hence, if we increase the current policy by a larger (disjunct) one, not only new requests result in more tokens but also all old requests. In Privacy Pass, it is possible to only increase the size of new batches without affecting the size of previous ones. In our system, we would need to renew the issuing key to achieve this.

Appendix E. Proof of Theorem 3

Proof: We start with showing *Correctness.* Let $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $(pk_G, sk_G) \leftarrow \mathsf{GroupKG}(pp)$, $x \in \mathcal{X}$, $((pk_1, pk_2), sk) \leftarrow \mathsf{KG}(pk_G)$ and $(y, (\pi', \tilde{pk}, \tilde{\mathsf{crt}}), \tau) \leftarrow \mathsf{Eval}(pk_G, pk, sk, \mathsf{crt}, x)$. Then, the following holds:

- The public parameters are of the form $pp = (\mathsf{BG}, \mathcal{X}, \mathcal{Y})$, where $\mathsf{BG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, p, g_1, g_2, e)$ defines a bilinear group, the input and output space are defined as $\mathcal{X} = \mathbb{Z}_p$ and $\mathcal{Y} = \mathbb{G}_3$;
- the key pair (pk_G, sk_G) is of the form $pk_G = (pk^{\mathsf{SIG}}, \pi_G)$ and $sk_G = sk^{\mathsf{SIG}}$, such that π_G is a NIZK proof of knowledge for $\mathsf{SIG.VKey}_{\mathcal{R}}(pk^{\mathsf{SIG}}, sk^{\mathsf{SIG}}) = 1$ and $(pk^{\mathsf{SIG}}, sk^{\mathsf{SIG}})$ is a key pair for the EQ-$\mathcal{R}$ signature scheme;
- the secret-key/ public-key pair is of the form $sk \in \mathbb{Z}_p$ and $pk = (g_1, g_1^{sk})$, i.e., (pk, sk) is a key pair for (an asymmetric version of) the Dodis-Yampolskiy VRF (DY PRF);
- the certificate crt is in the image of $\mathsf{SIG.Sign}_{\mathcal{R}}(sk_G, pk)$, i.e., crt is an EQ-$\mathcal{R}_{\mathsf{DDH}}$ signature for message pk;
- the output value $y = e(g_1, g_2)^{1/(x+sk)}$ corresponds to the output of the DY PRF;
- the first part of the proof $\pi' = g_2^{\frac{1}{\tau(x+sk)}}$ corresponds to the proof of the DY PRF re-randomized with $1/\tau$ (where τ is the opening information);

- the second part of the proof $(\tilde{pk}, \tilde{\mathsf{crt}})$ corresponds to the public key pk of the DY PRF re-randomized with τ, together with a re-randomized EQ-$\mathcal{R}$ signature $\tilde{\mathsf{crt}}$ of $\tilde{pk} = pk^{\tau}$.

First of all, by the correctness of the EQ-$\mathcal{R}$ signature scheme and completeness of the proof system PS it holds

$$\mathsf{VerGroup}(pk_G) = \mathsf{PS.Ver}(\mathsf{crs}, pk_G, \pi) = 1$$

$$\mathsf{VerCert}(pk_G, pk, \mathsf{crt}) = \mathsf{SIG.Vfy}_{\mathcal{R}}(pk_G, pk, \mathsf{crt}) = 1.$$

By the correctness of the EQ-$\mathcal{R}$ signature scheme for re-randomized signatures it further holds

$$\mathsf{SIG.Vfy}_{\mathcal{R}}(pk_G, \tilde{pk}, \tilde{\mathsf{crt}}) = 1.$$

Further, it holds

$$\begin{aligned} e(\tilde{pk}_1^{x} \cdot \tilde{pk}_2, \pi') &= e(g_1^{x\cdot\tau} \cdot g_1^{sk\cdot\tau}, g_2^{1/\tau(x+sk)}) \\ &= e(g_1^{\tau(x+sk)}, g_2^{1/\tau(x+sk)}) = e(g_1, g_2) \end{aligned}$$

and

$$e(\tilde{pk}_1, \pi') = e(g_1^{\tau}, g_2^{1/\tau(x+sk)}) = e(g_1, g_2)^{1/(x+sk)} = y$$

due to the bilinearity of e. Hence we have $\mathsf{Ver}(pk_G, x, y, \pi) = 1$. Next, we have

$$\tilde{pk} = pk^{\tau}$$

and hence $\mathsf{Judge}(pk_G, pk, x, y, \tau, \pi) = 1$. Overall, correctness of GVRF follows.

Pseudorandomness. We prove pseudorandomness of GVRF based on the perfect signature adaption of SIG, the zero knowledge and knowledge soundness of PS and the one-more bilinear 2-DDH/DDHI assumption (Definition 17), which is implied by the DDH/DDHI assumption (Lemma 1).

The reduction is shown in Figure 9. First, we can switch the generation of the crs of PS to the trapdoor generation $(\mathsf{crs}, \mathsf{td})$. By the zero knowledge property of PS the adversary is able to detect this switch with at most negligible probability.

Next, note that by the knowledge soundness of PS, if the adversary provides a valid proof π_G we can extract a valid secret signing key sk^{SIG} from the adversary except with negligible probability. Therefore, switching the signature generation in the evaluation oracle to fresh signatures does not change the view of the adversary by the perfect signature adaption of SIG.

Further, by setting $sk := \alpha$ we have that the public key and all proofs and output values satisfy the correct distribution, and thus we have that the view of $\mathcal{A}$ is distributed as in the pseudorandomness experiment depicted in Figure 1

Finally, we have that y_b is either the real GVRF value or random, since $y_0 = e(g_1^{\tau}, g_2^{\frac{1}{\tau\cdot(x^*+\alpha)}}) = e(g_1, g_2)^{1/(x^*+\alpha)})$ or $y_1 = e(g_1^{\tau}, g_2^{\frac{1}{\tau\cdot x^*+y}})$ (which is distributed uniformly at random as y is random), and hence the success probability of $\mathcal{B}$ is equal to the success probability of $\mathcal{A}$, which concludes the proof.

Unique provability. Let $pp \leftarrow \mathsf{Setup}(1^{\lambda})$, $(pk_G, sk_G) \leftarrow \mathsf{GroupKG}(1^{\lambda})$. We have to show that for all public keys pk, for all input values x, output values y_0, y_1, proofs π_0, π_1 and opening information τ_0, τ_1, for which it holds that $\mathsf{Judge}(pk_G, pk, x, y_0, \pi_0, \tau_0) = \mathsf{Judge}(pk_G, pk, x, y_1, \pi_1, \tau_1) = 1$, it holds $y_0 = y_1$. Let $\pi_b =: (\pi_b', \tilde{pk}_b, \tilde{\mathsf{crt}}_b)$ for $b \in \{0,1\}$. Then it must hold $\tilde{pk}_0 = pk^{\tau_0}$ and $\tilde{pk}_1 = pk^{\tau_1}$, as well as $\mathsf{Ver}(pk_G, pk, x, y_0, \pi_0) = \mathsf{Ver}(pk_G, pk, x, y_1, \pi_1) = 1$. This implies

(i) $\tilde{pk}_0 = \tilde{pk}_1^{\rho}$ for $\rho := \tau_0/\tau_1 \in \mathbb{Z}_p$,

(ii) $e(\tilde{pk}_{b,1}^{x^*} \cdot \tilde{pk}_{b,2}, \pi_b') = e(g_1, g_2)$ for $b \in \{0,1\}$

(iii) $e(\tilde{pk}_{b,1}, \pi_b') = y_b$ for $b \in \{0,1\}$.

Together, this yields

$$\begin{aligned} e(\tilde{pk}_{1,1}^{x^*} \cdot \tilde{pk}_{1,2}, \pi_1') &\overset{\text{(ii)}}{=} e(g_1, g_2) \overset{\text{(ii)}}{=} e(\tilde{pk}_{0,1}^{x^*} \cdot \tilde{pk}_{0,2}, \pi_0') \\ &\overset{\text{(i)}}{=} e(\tilde{pk}_{1,1}^{x^*\cdot\rho} \cdot \tilde{pk}_{1,2}^{\rho}, \pi_0') = e(\tilde{pk}_{1,1}^{x^*} \cdot \tilde{pk}_{1,2}, {\pi_0'}^{\rho}), \end{aligned}$$

and hence $\pi_0' = {\pi_1'}^{1/\rho}$ (iv). With this we obtain

$$\begin{aligned} y_0 &\overset{\text{(iii)}}{=} e(\tilde{pk}_{0,1}, \pi_0') \overset{\text{(i,iv)}}{=} e(\tilde{pk}_{1,1}^{\rho}, {\pi_1'}^{1/\rho}) = e(\tilde{pk}_{1,1}, \pi_1') \\ &\overset{\text{(iii)}}{=} y_1 \end{aligned}$$

as required.

Group-bounded provability. We give a reduction of the group-bounded provability of GVRF to the EUF-CMA security of the signature scheme SIG, using the zero knowledge property of PS which allows the reduction to switch real proofs of correct group key generation to simulated proofs. Recall that an adversary $\mathcal{A}$ breaks

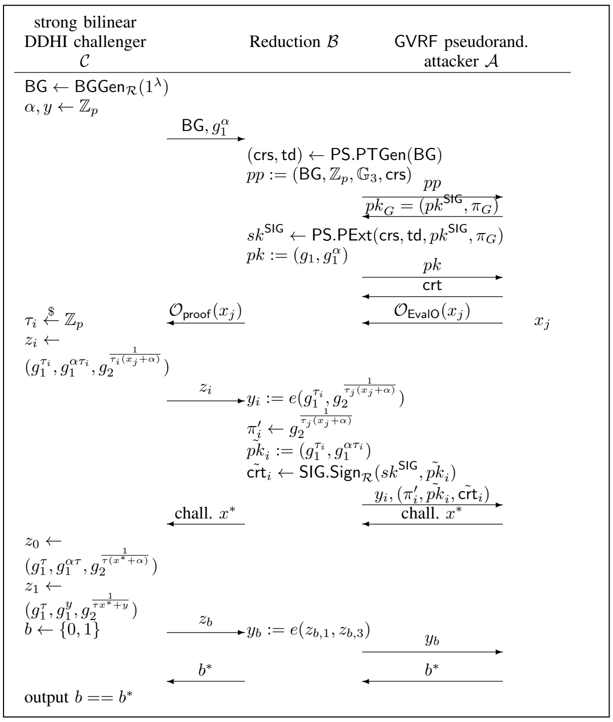
strong bilinear DDHI challenger $\mathcal{C}$
Reduction $\mathcal{B}$
GVRF pseudorand. attacker $\mathcal{A}$
$\mathsf{BG} \leftarrow \mathsf{BGGen}_{\mathcal{R}}(1^{\lambda})$
$\alpha, y \leftarrow \mathbb{Z}_p$
$\mathsf{BG}, g_1^{\alpha}$
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{PS.PTGen}(\mathsf{BG})$
$pp := (\mathsf{BG}, \mathbb{Z}_p, \mathbb{G}_3, \mathsf{crs})$
pp
$pk_G = (pk^{\mathsf{SIG}}, \pi_G)$
$sk^{\mathsf{SIG}} \leftarrow \mathsf{PS.PExt}(\mathsf{crs}, \mathsf{td}, pk^{\mathsf{SIG}}, \pi_G)$
$pk := (g_1, g_1^{\alpha})$
pk
crt
$\tau_i \overset{\$}{\leftarrow} \mathbb{Z}_p$
$\mathcal{O}_{\mathsf{proof}}(x_j)$
$\mathcal{O}_{\mathsf{EvalO}}(x_j)$
x_j
$z_i \leftarrow$
$(g_1^{\tau_i}, g_1^{\alpha\tau_i}, g_2^{\frac{1}{\tau_i(x_j+\alpha)}})$
z_i
$y_i := e(g_1^{\tau_i}, g_2^{\frac{1}{\tau_j(x_j+\alpha)}})$
$\pi_i' \leftarrow g_2^{\frac{1}{\tau_j(x_j+\alpha)}}$
$\tilde{pk}_i := (g_1^{\tau_i}, g_1^{\alpha\tau_i})$
$\tilde{\mathsf{crt}}_i \leftarrow \mathsf{SIG.Sign}_{\mathcal{R}}(sk^{\mathsf{SIG}}, \tilde{pk}_i)$
$y_i, (\pi_i', \tilde{pk}_i, \tilde{\mathsf{crt}}_i)$
chall. x^*
chall. x^*
$z_0 \leftarrow$
$(g_1^{\tau}, g_1^{\alpha\tau}, g_2^{\frac{1}{\tau(x^*+\alpha)}})$
$z_1 \leftarrow$
$(g_1^{\tau}, g_1^{y}, g_2^{\frac{1}{\tau x^*+y}})$
$b \leftarrow \{0,1\}$
z_b
$y_b := e(z_{b,1}, z_{b,3})$
y_b
b^*
b^*
output $b == b^*$

Figure 9: Reduction from the pseudorandomness of GVRF to the one-more bilinear 2-DDH/DDHI assumption (Def. 17). Note that the adversary $\mathcal{A}$ receives access to $\mathcal{O}_{\mathsf{Eval}}(\cdot)$ both before and after sending the challenge x^*, which for simplicity is not depicted in the figure. Further, the reduction $\mathcal{B}$ aborts, if x^* appears at any point as evaluation query, or if $\mathcal{A}$ fails to provide a valid pk_G and/ or valid certificate crt relative to pk.

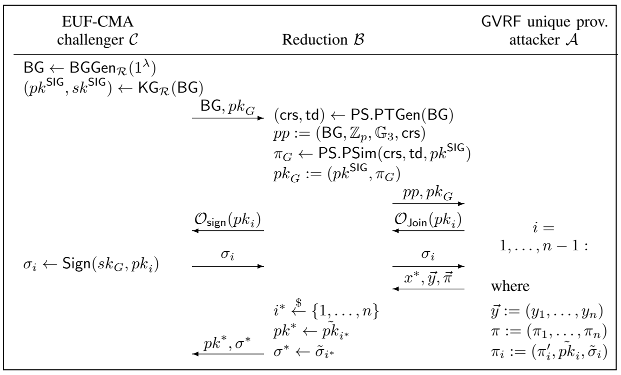
EUF-CMA challenger $\mathcal{C}$
Reduction $\mathcal{B}$
GVRF unique prov. attacker $\mathcal{A}$
$\mathsf{BG} \leftarrow \mathsf{BGGen}_{\mathcal{R}}(1^\lambda)$
$(pk^{\mathsf{SIG}}, sk^{\mathsf{SIG}}) \leftarrow \mathsf{KG}_{\mathcal{R}}(\mathsf{BG})$
BG, pk_G
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{PS.PTGen}(\mathsf{BG})$
$pp := (\mathsf{BG}, \mathbb{Z}_p, \mathbb{G}_3, \mathsf{crs})$
$\pi_G \leftarrow \mathsf{PS.PSim}(\mathsf{crs}, \mathsf{td}, pk^{\mathsf{SIG}})$
$pk_G := (pk^{\mathsf{SIG}}, \pi_G)$
pp, pk_G
$\mathcal{O}_{\mathsf{sign}}(pk_i)$
$\mathcal{O}_{\mathsf{Join}}(pk_i)$
$i = 1, \ldots, n-1:$
$\sigma_i \leftarrow \mathsf{Sign}(sk_G, pk_i)$
σ_i
σ_i
$x^*, \vec{y}, \vec{\pi}$
where
$i^* \overset{\$}{\leftarrow} \{1, \ldots, n\}$
$pk^* \leftarrow \tilde{pk}_{i^*}$
pk^*, σ^*
$\sigma^* \leftarrow \tilde{\sigma}_{i^*}$
$\vec{y} := (y_1, \ldots, y_n)$
$\pi := (\pi_1, \ldots, \pi_n)$
$\pi_i := (\pi'_i, \tilde{pk}_i, \tilde{\sigma}_i)$

Figure 10: Reduction from the group-bounded provability of GVRF to the EUF-CMA security of SIG.

unique provability if it produces one more evaluation y of x^* than group members it impersonates.

If the adversary is able generate proof for n distinct image values $y_1, \ldots, y_n$, it is able to generate $(\tilde{pk}_1, \tilde{\mathsf{crt}}_1), \ldots, (\tilde{pk}_n, \tilde{\mathsf{crt}}_n)$ such that $\mathsf{SIG.Vfy}_{\mathcal{R}}(pk^{\mathsf{SIG}}, \tilde{pk}_i, \mathsf{crt}_i) = 1$, where $(\tilde{pk}_1, \tilde{\mathsf{crt}}_1)$ is part of the (valid) proof π_i to preimage y_i. By the proof of unique verifiability, if $y_i \neq y_j$ for all $i \neq j$, we must also have $[\tilde{pk}_i]_{\mathcal{R}_{\mathsf{DDH}}} \neq [\tilde{pk}_j]_{\mathcal{R}_{\mathsf{DDH}}}$ for each $i \neq j$. Since the adversary only received the certificates for $\mathsf{ctr} < n$ public keys from the adversary, we can thus transform an adversary on the group-bounded provability of GVRF to an adversary on the EUF-CMA security on the signature scheme. However, since the reduction cannot recognize equivalence classes either, it needs to guess which of the overall n VRF values supplied by the GVRF attacker constitutes the forgery under pk_G. We present the formal reduction $\mathcal{B}$ in Figure 10.

Since $[\tilde{pk}_i]_{\mathcal{R}_{\mathsf{DDH}}} \neq [\tilde{pk}_j]_{\mathcal{R}_{\mathsf{DDH}}}$ for all $i, j \in \{1, \ldots, n\}$, $i \neq j$, the probability that $\mathcal{B}$ picks i^* such that no message of the equivalence class $[\tilde{pk}_{i^*}]$ was ever signed by the oracle is at least $1/n$. Because reduction $\mathcal{B}$ perfectly implements the unique provability experiment for $\mathcal{A}$, with $\mathsf{Ver}(pk_G, \tilde{pk}_\ell, \tilde{\sigma}_\ell) = 1$ for all $\ell = 1, \ldots, n$ it follows that the advantage of reduction $\mathcal{B}$ in winning the EUF-CMA experiment is at least $\Pr[\mathsf{Exp}^{\mathsf{gb-prov}}_{\mathsf{GVRF},\mathcal{A}}(1^\lambda) = 1]/n$. This concludes the proof.

Unlinkability. We show unlinkability of GVRF under the one-more bilinear 2-DDH/DDHI assumption (omb-2-DDH/DDHI, Def. 17), the perfect signature adaptation of the signature scheme SIG and the zero knowledge and knowledge soundness properties of the proof system PS. Recall that by Lemma 1, the omb-2-DDH/DDHI is implied by the one-more bilinear DDH/DDHI assumption (Lemma 1).

For the proof we first observe that replacing the honest generation of crs by a trapdoor generation is computationally indistinguishable by the zero knowledge property of the proof system. Further, by the knowledge soundness of the proof system, either the reduction aborts (if the adversary fails to provide a public key pk^{SIG} with valid proof π) or the reduction successfully extracts a secret key sk_G with $\mathsf{SIG.VKey}_{\mathcal{R}}(pk_G, sk_G) = 1$, except with negligible probability.

Next, observe that since by assumption our signature scheme satisfies perfect signature adaptation, and since the reduction aborts if the adversary fails to provide valid certificates $\mathsf{crt}_0, \mathsf{crt}_1$ with $\mathsf{SIG.Vfy}_{\mathcal{R}}(pk_G, pk_0, \mathsf{crt}_0) = \mathsf{SIG.Vfy}_{\mathcal{R}}(pk_G, pk_0, \mathsf{crt}_1) = 1$, it does not change the view of the adversary if we replace re-randomized signatures by freshly generated signatures under the secret key sk_G.

Next, we define a set of hybrids $\mathsf{H}_0, \ldots, \mathsf{H}_q$, where q denotes the number of queries to $\mathcal{O}^b_{\mathsf{Eval}}(\cdot)$. Hybrid H_i, for $i = 0, \ldots, q$, denotes the unlinkability game (cf. Figure 3) with the modification that the evaluation oracle $\mathcal{O}^b_{\mathsf{Eval}}(\cdot)$ no longer depends on the bit b (and is thus in the following refered to as $\mathcal{O}_{\mathsf{Eval}}(\cdot)$), but instead the first i queries are answered with sk_1, and queries $i+1, \ldots, q$ are answered with sk_0.

Consequently, H_0 denotes the unlinkability game where the adversary has access to oracle $\mathcal{O}^0_{\mathsf{Eval}}(\cdot)$, and H_q is equal to the unlinkability game where the adversary has access to oracle $\mathcal{O}^1_{\mathsf{Eval}}(\cdot)$ (with the only difference that re-randomized signatures are replaced by freshly generated ones, which does not change the view of the adversary as elaborated above).

Now, for each $i = 0, ..., q-1$, we build an adversary $\mathcal{B}_i$ solving the one-more bilinear 2-DDH/DDHI problem given a PPT adversary $\mathcal{A}$ distinguishing any two of the consecutive hybrids. The reduction is depicted in Figure 11.

For $b^* = 0$, we now have that $\mathcal{B}$ perfectly simulates H_i, since in this case we have:

- $\tilde{pk}_{i+1} = (z_{j,1}, z_{j,2}) = (g_1^\tau, g_1^{\alpha_0 \tau}) = (g_1, A_0)^\tau = pk_0^\tau$,
- $\pi_{i+1} = z_{j,2} = g_2^{\frac{1}{\tau(\alpha_0 + x_{i+1})}}$ and
- $y_{i+1} = e(z_{j,1}, z_{j,3}) = e(g_1, g_2)^{\frac{1}{\alpha_0 + x_{i+1}}}$,

and thus the $(i+1)$-st evaluation query is answered using sk_0 (as well as all previous queries, independently of the challenge bit b^*). Similarly, if $b^* = 1$, we obtain that $\mathcal{B}$ perfectly simulates H_{i+1}, since in this case the $(i+1)$-st evaluation query is answered using sk_1 (as well as all following queries, independently of the challenge bit b^*). Now, let $\mathcal{B}$ denote an adversary which first samples $i \overset{\$}{\leftarrow} \{0, \ldots, q-1\}$ and then runs $\mathcal{B}_i$ on $\mathcal{A}$. Then, we obtain the following:

$$
\begin{aligned}
&\Pr[b^* \leftarrow \mathcal{B}] - \frac{1}{2} \\
&= \frac{1}{2} \Pr[0 \leftarrow \mathcal{B} \mid b^* = 0] + \frac{1}{2} \Pr[1 \leftarrow \mathcal{B} \mid b^* = 1] - \frac{1}{2} \\
&= \frac{1}{2} \Pr[1 \leftarrow \mathcal{B} \mid b^* = 0] - \frac{1}{2} \Pr[1 \leftarrow \mathcal{B} \mid b^* = 1] \\
&= \sum_{j=0}^{q-1} \frac{1}{2q} \cdot (\Pr[1 \leftarrow \mathcal{B}_i \mid b^* = 0 \wedge i = j] \\
&\quad - \Pr[0 \leftarrow \mathcal{B}_i \mid b^* = 1 \wedge i = j]) \\
&= \frac{1}{2q} \cdot \sum_{j=0}^{q-1} (\Pr[1 \leftarrow \mathcal{A} \text{ in } H_i \mid i = j] - \Pr[0 \leftarrow \mathcal{A} \text{ in } H_{i+1} \mid i = j]) \\
&= \frac{1}{2q} \cdot (\Pr[1 \leftarrow \mathcal{A} \text{ in } H_0] - \Pr[1 \leftarrow \mathcal{A} \text{ in } H_q]) \,.
\end{aligned}
$$

Since $\Pr[\mathcal{B} \text{ outputs } b^*] - \frac{1}{2}$ is negligible by the one-more blinear 2-DDH/DDHI assumption, so is the probability that the adversary $\mathcal{A}$ can distinguish between the evaluation oracle $\mathcal{O}^0_{\mathsf{Eval}}(\cdot)$ and the evaluation oracle $\mathcal{O}^1_{\mathsf{Eval}}(\cdot)$.

Unique opening. It is left to prove unique opening.

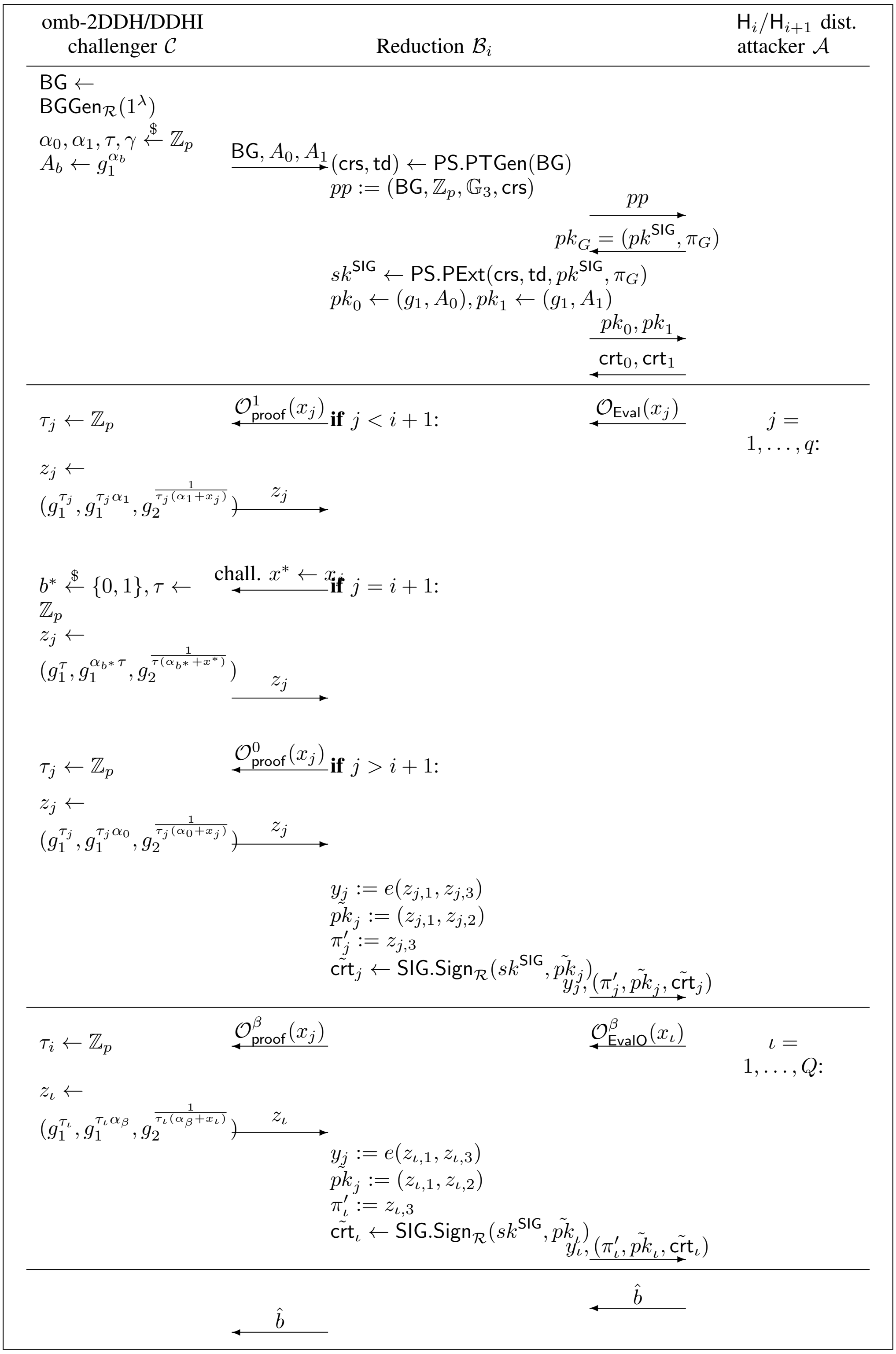
omb-2DDH/DDHI challenger $\mathcal{C}$
Reduction $\mathcal{B}_i$
$\mathsf{H}_i/\mathsf{H}_{i+1}$ dist. attacker $\mathcal{A}$
$\mathsf{BG} \leftarrow \mathsf{BGGen}_{\mathcal{R}}(1^\lambda)$
$\alpha_0, \alpha_1, \tau, \gamma \xleftarrow{\$} \mathbb{Z}_p$
$A_b \leftarrow g_1^{\alpha_b}$
BG, A_0, A_1
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{PS.PTGen}(\mathsf{BG})$
$pp := (\mathsf{BG}, \mathbb{Z}_p, \mathbb{G}_3, \mathsf{crs})$
pp
$pk_G = (pk^{\mathsf{SIG}}, \pi_G)$
$sk^{\mathsf{SIG}} \leftarrow \mathsf{PS.PExt}(\mathsf{crs}, \mathsf{td}, pk^{\mathsf{SIG}}, \pi_G)$
$pk_0 \leftarrow (g_1, A_0), pk_1 \leftarrow (g_1, A_1)$
pk_0, pk_1
$\mathsf{crt}_0, \mathsf{crt}_1$
$\tau_j \leftarrow \mathbb{Z}_p$
$\mathcal{O}^1_{\mathsf{proof}}(x_j)$
if $j < i+1$:
$\mathcal{O}_{\mathsf{Eval}}(x_j)$
$j = 1, \dots, q$:
$z_j \leftarrow (g_1^{\tau_j}, g_1^{\tau_j \alpha_1}, g_2^{\frac{1}{\tau_j(\alpha_1 + x_j)}})$
z_j
$b^* \xleftarrow{\$} \{0,1\}, \tau \leftarrow \mathbb{Z}_p$
chall. $x^* \leftarrow x_j$
if $j = i+1$:
$z_j \leftarrow (g_1^{\tau}, g_1^{\alpha_{b^*}\tau}, g_2^{\frac{1}{\tau(\alpha_{b^*} + x^*)}})$
z_j
$\tau_j \leftarrow \mathbb{Z}_p$
$\mathcal{O}^0_{\mathsf{proof}}(x_j)$
if $j > i+1$:
$z_j \leftarrow (g_1^{\tau_j}, g_1^{\tau_j \alpha_0}, g_2^{\frac{1}{\tau_j(\alpha_0 + x_j)}})$
z_j
$y_j := e(z_{j,1}, z_{j,3})$
$\tilde{pk}_j := (z_{j,1}, z_{j,2})$
$\pi'_j := z_{j,3}$
$\tilde{\mathsf{crt}}_j \leftarrow \mathsf{SIG.Sign}_{\mathcal{R}}(sk^{\mathsf{SIG}}, \tilde{pk}_j)$
$y_j, (\pi'_j, \tilde{pk}_j, \tilde{\mathsf{crt}}_j)$
$\tau_i \leftarrow \mathbb{Z}_p$
$\mathcal{O}^\beta_{\mathsf{proof}}(x_j)$
$\mathcal{O}^\beta_{\mathsf{EvalO}}(x_\iota)$
$\iota = 1, \dots, Q$:
$z_\iota \leftarrow (g_1^{\tau_\iota}, g_1^{\tau_\iota \alpha_\beta}, g_2^{\frac{1}{\tau_\iota(\alpha_\beta + x_\iota)}})$
z_ι
$y_j := e(z_{\iota,1}, z_{\iota,3})$
$\tilde{pk}_j := (z_{\iota,1}, z_{\iota,2})$
$\pi'_\iota := z_{\iota,3}$
$\tilde{\mathsf{crt}}_\iota \leftarrow \mathsf{SIG.Sign}_{\mathcal{R}}(sk^{\mathsf{SIG}}, \tilde{pk}_\iota)$
$y_\iota, (\pi'_\iota, \tilde{pk}_\iota, \tilde{\mathsf{crt}}_\iota)$
$\hat{b}$
$\hat{b}$

Figure 11: Reduction from the unlinkability of GVRF to the one-more bilinear 2-DDH/DDHI problem. The adversary can query the oracles in arbitrary order. The reduction aborts if the adversary fails to provide a valid public key pk_G or valid certificates $\mathsf{crt}_0, \mathsf{crt}_1$ for pk_0 or pk_1, respectively, or if the adversary asks any x to $\mathcal{O}_{\mathsf{Eval}}(\cdot)$ *and* $\mathcal{O}^\beta_{\mathsf{EvalO}}(\cdot)$ for any β.

Recall that our public keys are of the form $pk = (g_1, A)$. It thus suffices to show that for any fixed input/ output pair (x, y) there cannot exist two public keys pk_0, pk_1, proofs π_0, π_1 and opening information τ_0, τ_1 such that $\mathsf{Judge}(pk_G, pk_0, x, y, \pi_0, \tau_0) = \mathsf{Judge}(pk_G, pk_1, x, y, \pi_1, \tau_1) = 1$ but $[pk_0]_{\mathcal{R}_{\mathsf{DDH}}} \neq [pk_1]_{\mathcal{R}_{\mathsf{DDH}}}$, since $[pk_0]_{\mathcal{R}_{\mathsf{DDH}}} \neq [pk_1]_{\mathcal{R}_{\mathsf{DDH}}}$ and $pk_{0,1} = pk_{1,1} = g_1$ implies $pk_{0,2} = pk_{2,1}$ as required.

Let $\pi_b =: (\pi'_b, \tilde{pk}_b, \tilde{\mathsf{crt}}_b)$ the corresponding proof $b \in \{0, 1\}$. Then, the above implies

(i) $e(\tilde{pk}_{b,1}^{x} \cdot \tilde{pk}_{b,2}, \pi'_b) = e(g_1, g_2)$ for $b \in \{0, 1\}$
(ii) $e(\tilde{pk}_{b,1}, \pi'_b) = y$ for $b \in \{0, 1\}$
(iii) $\tilde{pk}_b = pk_b^{\tau_b}$ for $b \in \{0, 1\}$.

Further, let $\rho \in \mathbb{Z}_p$ be such that $\tilde{pk}_{0,1} = \tilde{pk}_{1,1}^{\rho}$ (iv) (note that such a ρ always exists, since $\tilde{pk}_{0,1}, \tilde{pk}_{1,1} \in \mathbb{G}_1$ are group elements). With this we obtain

$$e(\tilde{pk}_{1,1}, \pi'_1) \overset{\text{(i)}}{=} y \overset{\text{(i)}}{=} e(\tilde{pk}_{0,1}, \pi'_0) \overset{\text{(iv)}}{=} e(\tilde{pk}_{1,1}^{\rho}, \pi'_0)$$
$$= e(\tilde{pk}_{1,1}, {\pi'_0}^{\rho})$$

and hence $\pi'_0 = {\pi'_1}^{1/\rho}$ (v). With this, we obtain

$$e(\tilde{pk}_{1,1}^{x^*} \cdot \tilde{pk}_{1,2}, \pi'_1) \overset{\text{(i)}}{=} e(g_1, g_2) \overset{\text{(i)}}{=} e(\tilde{pk}_{0,1}^{x^*} \cdot \tilde{pk}_{0,2}, \pi'_0)$$
$$\overset{\text{(iv,v)}}{=} e(\tilde{pk}_{1,1}^{x^* \cdot \rho} \cdot \tilde{pk}_{0,2}, {\pi'_1}^{1/\rho})$$

which implies

$$e(\tilde{pk}_{1,2}, \pi'_1) {=} e(\tilde{pk}_{0,2}, {\pi'_1}^{1/\rho})$$

and thus $\tilde{pk}_{0,2} = \tilde{pk}_{1,2}^{\rho}$ and hence $\tilde{pk}_0 = \tilde{pk}_1^{\rho}$, which by (iii) implies $[pk_0]_{\mathcal{R}_{\mathsf{DDH}}} = [pk_1]_{\mathcal{R}_{\mathsf{DDH}}}$ as required.

□

Appendix F. Proof of Lemma 1

Let $\mathcal{A}$ be an adversary on the one-more bilinear 2-DDH/DDHI assumption. Then, we construct an adversary $\mathcal{B}$ on the one-more bilinear DDH/DDHI assumption as follows. On input $\mathsf{BG}, g_1^{\alpha}$, the adversary chooses $b \overset{\$}{\leftarrow} \{0, 1\}$ and $\alpha_b \leftarrow \mathbb{Z}_p$ at random, implicitly sets $\alpha_{1-b} := \alpha$ (without knowing the exponent α), and sends $(\mathsf{BG}, g_1^{\alpha_0}, g_1^{\alpha_1})$ to $\mathcal{A}$. On input of a proof-query $\mathcal{O}^{\beta}_{\mathsf{proof}}(x)$ by $\mathcal{A}$, it proceeds as follows: If $\beta = b$ it simulates $\mathcal{O}^{\beta}_{\mathsf{proof}}$ using α_b. If $\beta = 1 - b$, it forwards x to its own challenge query and forwards the response $(g_1^{\tau_i}, g_1^{\alpha\tau_i}, g_2^{\frac{1}{\tau_i(\alpha+x)}})$ to $\mathcal{A}$. Next, it forwards x^* by $\mathcal{A}$ to its own experiment, and hands the challenge query z^* to $\mathcal{A}$. If x^* appears as query to either of the proof oracles at any point, $\mathcal{B}$ aborts. Otherwise, on output $b' = b$ by $\mathcal{A}$, the adversary $\mathcal{B}$ outputs 0 ("real") to its own experiment. Else, it outputs 1 ("random") to its own experiment. Towards the analysis of the success probability of $\mathcal{B}$, note that if $b^* = 0$ (where b^* is the bit chosen by its own experiment), then $\mathcal{B}$ perfectly simulates the one-more bilinear 2-DDH/DDHI game for $\mathcal{A}$ relative to the chosen bit b. Further, if $\mathcal{B}$ was successful (i.e., guesses $b' = b$), then so is $\mathcal{A}$. If $b^* = 1$ on the other hand, the view of $\mathcal{A}$ is independent of the challenge bit b. The probability it outputs 1 to its own experiment is thus exactly $\frac{1}{2}$. Overall, we obtain that $\mathcal{B}$ has advantage

$$\epsilon_{\mathcal{B}} := \Pr[\mathcal{B} \text{ is successful}] - \frac{1}{2}$$
$$= \frac{1}{2} \underbrace{\Pr[\mathcal{B} \text{ is successful} | b^* = 0]}_{= \epsilon_{\mathcal{A}} + \frac{1}{2}} + \frac{1}{2} \underbrace{\Pr[\mathcal{B} \text{ is successful} | b^* = 1]}_{= \frac{1}{2}}$$
$$- \frac{1}{2} = \frac{1}{2} \cdot \epsilon_{\mathcal{A}},$$

where $\epsilon_{\mathcal{A}} := \Pr[\mathcal{A} \text{ is successful}] - \frac{1}{2}$. This concludes the proof.

Appendix G. Data Availability

All source code to reproduce our benchmarks is publicly released on GitHub under an open-source license: https://github.com/dennisfInf/GVRF#