Statically Aggregate Verifiable Random Functions and Application to E-Lottery

cryptography

MDPI

Article

Statically Aggregate Verifiable Random Functions and Application to E-Lottery

Bei Liang [1,*], Gustavo Banegas [1] and Aikaterini Mitrokotsa [1,2]

[1] Department of Computer Science and Engineering, Chalmers University of Technology, SE-41296 Gothenburg, Sweden; gustavo@cryptme.in (G.B.); aikmitr@chalmers.se (A.M.)
[2] School of Computer Science, University of St. Gallen, CH-9000 St. Gallen, Switzerland
* Correspondence: lbei@chalmers.se

Received: 29 October 2020; Accepted: 9 December 2020; Published: 13 December 2020

Abstract: Cohen, Goldwasser, and Vaikuntanathan (TCC'15) introduced the concept of aggregate pseudo-random functions (PRFs), which allow efficiently computing the aggregate of PRF values over exponential-sized sets. In this paper, we explore the aggregation augmentation on verifiable random function (VRFs), introduced by Micali, Rabin and Vadhan (FOCS'99), as well as its application to e-lottery schemes. We introduce the notion of *static aggregate verifiable random functions* (Agg-VRFs), which perform aggregation for VRFs in a *static* setting. Our contributions can be summarized as follows: (1) we define static aggregate VRFs, which allow the efficient aggregation of VRF values and the corresponding proofs over super-polynomially large sets; (2) we present a static Agg-VRF construction over bit-fixing sets with respect to product aggregation based on the q-decisional Diffie–Hellman exponent assumption; (3) we test the performance of our static Agg-VRFs instantiation in comparison to a standard (non-aggregate) VRF in terms of costing time for the aggregation and verification processes, which shows that Agg-VRFs lower considerably the timing of verification of big sets; and (4) by employing Agg-VRFs, we propose an improved e-lottery scheme based on the framework of Chow et al.'s VRF-based e-lottery proposal (ICCSA'05). We evaluate the performance of Chow et al.'s e-lottery scheme and our improved scheme, and the latter shows a significant improvement in the efficiency of generating the winning number and the player verification.

Keywords: pseudorandom functions; verifiable random functions; aggregate pseudorandom functions; aggregate verifiable random functions

1. Introduction

Verifiable random functions (VRFs), initially introduced by Micali, Rabin, and Vadhan [1], can be seen as the public key equivalent of pseudorandom functions (PRFs) that, besides the *pseudorandomness* property (i.e., the function looks random at any input x), also provide the property of *verifiability*. More precisely, VRFs are defined by a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$ in such a way that they provide not only the efficient computation of the pseudorandom function $f_{\mathsf{sk}}(x) = y$ for any input x but also a non-interactive publicly verifiable proof $\pi_{\mathsf{sk}}(x)$ that, given access to pk, allows the efficient verification of the statement $f_{\mathsf{sk}}(x) = y$ for all inputs x. VRFs have been shown to be very useful in multiple application scenarios including key distribution centres [2], non-interactive lottery systems used in micropayments [3], domain name security extensions (DNSSEC) [4–6], e-lottery schemes [7], and proof-of-stake blockchain protocols such as Ouroboros Praos [8,9].

Cohen, Goldwasser, and Vaikuntanathan [10] were the first to investigate how to answer *aggregate queries* for PRFs over exponential-sized sets and introduced a type of augmented PRFs, called aggregate pseudo-random functions, which significantly enriched the existing family of (augmented) PRFs including constrained PRFs [11], key-homomorphic PRFs [12], and distributed PRFs [2]. Inspired by

the idea of aggregated PRFs [10], in this paper, we explore the aggregation of VRFs and introduce a new cryptographic primitive, *static aggregate verifiable random functions* (static Agg-VRFs), which allow not only the efficient aggregation operation both on function values and proofs but also the verification on the correctness of the aggregated results.

Aggregate VRFs allow the efficient aggregation of a large number of function values, as well as the efficient verification of the correctness of the aggregated function result by employing the corresponding aggregated proof. Let us give an example to illustrate this property. Consider a cloud-assisted computing setting where a VRF can be employed in the client–server model, i.e., Alice is given access to a random function where the function description (or the secret key) is stored by a server (seen as the random value provider). Whenever Alice requests an arbitrary bit-string x, the server simply computes the function value $y = f(x)$ together with the corresponding proof π and returns the tuple (x, y, π) to Alice. Alice may also request the aggregation (such as the product) of the function values over a *large* number of points (e.g., $x_1, x_2, \ldots, x_n$, which may match some pattern, such as having same bits on some bit locations). In this case, aggregate VRFs allow the server to compute the product of $f(x_1), \ldots, f(x_n)$ *efficiently*, instead of firstly evaluating $f(x_1), \ldots, f(x_n)$ and then calculating their product. On receiving either the function value y of an individual input or the aggregated function value y^{agg} over multiple inputs, Alice needs to verify the correctness of the returned value. VRFs allow the verification of the correctness of y using π, while, to verify the correctness of y^{agg}, there is a trivial way, namely firstly verifying (x_i, y_i, π_i) for $i = 1, \ldots, n$ using the verification algorithm of VRFs and then checking if $y^{\mathsf{agg}} = \Pi_{i=1}^{n} y_i$, but the running time of which depends on the number n. Via aggregate VRFs, the verification of y^{agg} can be achieved much more efficiently by using the aggregated proof π^{agg} that is generated by the server and returned to Alice along with y^{agg}.

A representative application of aggregate VRFs is in e-lottery schemes. More precisely, aggregate VRFs can be employed in VRF-based e-lottery schemes [7], where a random number generation mechanism is required to determine not only a winning number but also the public verifiability of the winning result, which guarantees that the dealer cannot cheat in the random number generation process. In this paper, we provide an e-lottery scheme, which has significant gain in the efficiency of generating the winning numbers and verifying the winning results. In a nutshell, VRF-based e-lottery schemes [7] proceed as follows: Initially, the dealer generates a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$ of VRFs and publishes the public key pk, together with a parameter $\mathcal{T}$ associated with the time (this is the input parameter controlling the time complexity of the delaying function $D(\cdot)$) during which the dealer must release the winning ticket value. To purchase the ticket, a player chooses his bet number s and obtains a ticket ticket_i (please refer to Section 4.2 for the generation of ticket ticket_i on a bet number x in detail) from the dealer. The dealer links the ticket to a blockchain, which could be created as $\mathsf{chain}_1 := H(\mathsf{ticket}_1)$, $\mathsf{chain}_i := H(\mathsf{chain}_{i-1} || \mathsf{ticket}_i)$ for $i > 1$, and publishes chain_j where j is the number of tickets sold so far. To generate the random winning number, the dealer first computes a VRF as $(w_0, \pi_0) = (f_{\mathsf{sk}}(d), \pi_{\mathsf{sk}}(d))$ on $d = D(h)$, where h is the final value of the blockchain (i.e., suppose there are n tickets sold, then $h := H(\mathsf{chain}_n)$). Assume that the numbers used in the lottery game are $\{1, 2, \ldots, N_{\mathsf{max}}\}$. If $w_0 > N_{\mathsf{max}}$, then the dealer iteratively applies the VRF on $w_{i-1} \| d$ to obtain $(w_i, \pi_i) = (f_{\mathsf{sk}}(w_{i-1} \| d), \pi_{\mathsf{sk}}(w_{i-1} \| d))$. Suppose that, within $\mathcal{T}$ units of time after the closing of the lottery session, until applying the VRF for t times, the dealer obtains (w_t, π_t) such that $w_t \leq N_{\mathsf{max}}$. Afterwards, the dealer publishes (w_t, π_t) as the winning number and the corresponding proof as well as all the intermediate tuples $(w_0, \pi_0), \ldots, (w_{t-1}, \pi_{t-1})$. If $\mathsf{s} = w_t$, a player wins. To verify the validity of a winning number w_t, each player verifies the validity of (w_i, π_i) for $i = 0, \ldots, t$.

Chow et al.'s e-lottery scheme [7] seems to be very promising when considering an ideal case that after a small number t of times that the VRF is applied, a function value w_t such that $w_t \leq N_{\mathsf{max}}$ can be obtained successfully. Otherwise, it means that the dealer needs to calculate the VRF more times, while the player needs to verify the correctness of more tuples in order to verify the winning result; the latter leads to large computational overhead and requires storage of all intermediate tuples of VRF function values and corresponding proofs, both from the dealer and the player.

Observe that both the evaluation and verification of multiple pairs of VRF function value/proof are time consuming. By using our aggregate VRF instantiation, we improve the e-lottery by devising the dealer to evaluate aggregate VRF twice at most so as to obtain a random winning number together with corresponding proof, thus rendering the verification for only such a single pair. This reduces the amount of data written to the dealer's storage space and also decreases the computational cost for the verification process of each player.

Our Contribution. We introduce the notion of *static aggregate verifiable random functions* (static Agg-VRFs). Briefly, a static Agg-VRF is a family of keyed functions each associated with a pair of keys, such that, *given* the secret key, one can compute the aggregation function for both the function values and the proofs of the VRFs over super-polynomially large sets in polynomial time, while, given the public key, the correctness of the aggregate function values could be checked by the corresponding aggregated proof. It is very important that the sizes of the aggregated function values and proofs should be independent of the size of the set over which the aggregation is performed. The security requirement of a static Agg-VRF states that access to an aggregate oracle provides no advantage to the ability of a polynomial time adversary to distinguish the function value from a random value, even when the adversary could query an aggregation of the function values over a specific set (of possibly super-polynomial size) of his choice.

In this paper, the aggregate operation we consider is the product of all the VRF values and proofs over inputs belonging to a super-polynomially large set. We show how to compute the product aggregation over a super-polynomial size set in polynomial time, since it is impossible to directly compute the product on a super-polynomial number of values. More specifically, we show how to achieve a static Agg-VRF under the Hohenberger and Waters' VRF scheme [13] for the product aggregation with respect to a bit-fixing set. We stress that after revisiting the JN-VRF scheme [14] proposed by Jager and Niehuesbased (currently the most efficient VRFs with full adaptive security in the standard model), we find that, even though JN-VRF almost enjoys the same framework of HW-VRF (since an admissible hash function H_{AHF} is applied on inputs x before evaluating the function value and the corresponding proof, which impacts negatively the nice pattern of all inputs in a bit-fixing set), it is impossible to perform productive aggregation of a super-polynomial number of values $f_{sk}(H_{\mathsf{AHF}}(x))$ efficiently over bit-fixing sets.

We implemented and evaluated the performance of our proposed static aggregate VRF in comparison to a standard (non-aggregate) VRF for inputs with different lengths i.e., 56, 128, 256, 512, and 1024 bits, in terms of the costing time for aggregating the function values, aggregating the proofs as well as the cost of verification for the aggregation. In all cases, our aggregate VRFs present significant computational advantage and are more efficient than standard VRFs. Furthermore, by employing aggregate VRFs for bit-fixing sets, we propose an improved *e-lottery scheme* based on the framework of Chow et al.'s VRF-based e-lottery proposal [7], by mainly modifying the winning result generation phase and the player verification phase. We implemented and tested the performance of both Chow et al.'s and our improved e-lottery schemes. Our improved scheme shows a significant improvement in efficiency in comparison to Chow et al.'s scheme.

Core Technique. We present a construction of static aggregate VRFs, which performs the product aggregation over a bit-fixing set, following Hohenberger and Waters' [13] VRF scheme. A bit-fixing set consists of bit-strings which match a particular bit pattern. It can be defined by a pattern string $v \in \{0,1,\perp\}^{poly(\lambda)}$ as $S_v = \{x \in \{0,1\}^{poly(\lambda)} : \forall i, x_i = v_i \text{or} v_i = \perp\}$. The evaluation of the VRF on input $x = x_1 \| x_2 \| \ldots \| x_\ell$ is defined as $y = e(g,h)^{u_0 \Pi_{i=1}^{\ell} u_i^{x_i}}$, where $g, h, U_0 = g^{u_0}, \ldots, U_\ell = g^{u_\ell}$ are public keys and $u_0, \ldots, u_\ell$ are kept secret. The corresponding proofs of the VRF are given using a step ladder approach, namely, for $j = 1$ to ℓ, $\pi_j = g^{\Pi_{i=1}^{j} u_i^{x_i}}$ and $\pi_{\ell+1} = g^{u_0 \Pi_{i=1}^{\ell} u_i^{x_i}}$.

Let $\text{Fixed}(v) = \{i \in [\ell] : v_i \in \{0,1\}\}$ and $|\text{Fixed}(v)| = \tau$. To aggregate the VRF, let $\pi_0^{\mathsf{agg}} = g^{2^{\ell-\tau}}$, for $i = 1, \ldots, \ell$; we compute

$$\pi_i^{\mathsf{agg}} = \begin{cases} (\pi_{i-1}^{\mathsf{agg}})^{u_i^{v_i}} & \text{if } i \in \text{Fixed}(v) \\ (\pi_{i-1}^{\mathsf{agg}})^{(u_i+1)/2} & \text{if } i \notin \text{Fixed}(v) \end{cases} \tag{1}$$

and $\pi_{\ell+1}^{\mathsf{agg}} = (\pi_\ell^{\mathsf{agg}})^{u_0}$. The aggregated function value is computed as

$$y^{\mathsf{agg}} = e(g,h)^{u_0 \prod_{i \in \text{Fixed}(v)} u_i^{v_i} \prod_{i \in [\ell] \setminus \text{Fixed}(v)} (u_i+1)}. \tag{2}$$

The aggregation verification algorithm checks the following equations: for $i = 1, \ldots, \ell$

$$e(g, \pi_i^{\mathsf{agg}}) = \begin{cases} e(\pi_{i-1}^{\mathsf{agg}}, g) & \text{if } i \in \text{Fixed}(v) \text{and} v_i = 0 \\ e(\pi_{i-1}^{\mathsf{agg}}, U_i) & \text{if } i \in \text{Fixed}(v) \text{and} v_i = 1 \\ e(\pi_{i-1}^{\mathsf{agg}}, g \cdot U_i)^{1/2} & \text{if } i \notin \text{Fixed}(v) \end{cases} \tag{3}$$

and $e(\pi_{\ell+1}^{\mathsf{agg}}, g) = e(\pi_\ell^{\mathsf{agg}}, U_0)$ and $e(\pi_{\ell+1}^{\mathsf{agg}}, h) = y^{\mathsf{agg}}$.

Improved Efficiency. We provide some highlights on the achieved efficiency.

Efficiency of Aggregate VRF. The construction of our static aggregate VRF for a bit-fixing (BF) set achieves high performance in the verification process, since it takes only $\mathcal{O}(\ell)$ bilinear pairing operations, even when verifying an exponentially large set of function values, where ℓ denotes the input length. The experimental results show that, even for 1024 bits of inputs, the aggregation of 2^{1004} pairs of function values/proofs can be computed very efficiently in 6881 ms. Moreover, the time required to verify their aggregated function values/proofs of 2^{1004} pairs only increases 50%, comparing with the verification time for each single function value/proof pair of standard VRF. Sections 3.2 and 3.3 present a detailed efficiency discussion and our experimental tests and comparisons.

Efficiency of Improved E-Lottery Scheme. We test the performance of Chow et al.'s e-lottery scheme [7] and our improved (aggregate VRF based) counterpart and make a comparison. In our improved e-lottery scheme, the computation of the aggregate function value/proof pair and the verification are performed via a single step of Aggregation and AggVerify algorithms, respectively, while Chow et al.'s e-lottery scheme is processed by t steps. We perform some experiments on Chow et al.'s scheme to see how big/small the t is so as to reach the point where the dealer obtains (w_t, π_t) such that $w_t \leq N_{\mathsf{max}}$, thus figuring out the computation-time for the corresponding multiple function evaluation and verification. In the experiments, we ran 10 times Chow et al.'s scheme and we obtained the median of all the runs. We reached $t \approx 2$ and it took ≈100 s for each run of the winner generation and ≈5 s for player verification. In our improved version, the generation of the winner ticket costs less than 90 s, and the time for verification decreases to ≈2.5 s, which shows a significant improvement in efficiency.

Related work. We summarize relevant current state-of-the-art.

Verifiable Random Functions. Hohenberger and Waters' VRF scheme [13] is the first that shows all the desired properties for a VRF (we say that a VRF scheme has all the desired properties if it allows an exponential-sized input space, achieves full adaptive security, and is based on a non-interactive assumption). Formerly, there have been several VRF proposals [15–17], all of which have some limitations: they only allow a polynomial-sized input space, they do not achieve fully adaptive security, or they are based on an interactive assumption. Thus far, there are also many constructions of VRFs with all the desired properties based on the decisional Diffie–Hellman assumption (DDH) or the decision linear assumption (DLIN) presenting different security losses [18–22]. Kohl [22] provided a detailed summary and comparison of all existing efficient constructions of VRFs in terms of the underlying assumption, sizes of verification key and the corresponding proof, and the associated

security loss. Recently, Jager and Niehues [14] provided the most efficient VRF scheme with adaptive security in the standard model, relying on the computational admissible hash functions.

Aggregate Pseudorandom Functions. Cohen et al. [10] introduced the notion of aggregate PRFs, which is a family of functions indexed by a secret key with the functionality that, given the secret key, anyone is able to aggregate the values of the function over super-polynomially many PRF values with only a polynomial-time computation. They also proposed constructions of aggregate PRFs under various cryptographic hardness assumptions (one-way functions and sub-exponential hardness of the Decisional Diffie–Hellman assumption) for different types of aggregation operators such as sums and products and for several set systems including intervals, bit-fixing sets, and sets that can be recognized by polynomial-size decision trees and read-once Boolean formulas. In this paper, we explore how to aggregate VRFs, which involves efficient aggregations both on the function evaluations and on the corresponding proofs, while providing verifiability for the correctness of aggregated function value via corresponding proof.

E-lottery Schemes/Protocols. In 2005, Chow et al. [7] proposed an e-lottery scheme using a verifiable random function (VRF) and a delay function. To reduce the complexity in the (purchaser) verification phase, Liu et al. [23] improved Chow et al.'s scheme by proposing a multi-level hash chain to replace the original linear hash chain, as well as a hash-function-based delay function, which is more suitable for e-lottery networks with mobile portable terminals. Based on the secure one-way hash function and the factorization problem in RSA, Lee and Chang [24] presented an electronic t-out-of-n lottery on the Internet, which allows lottery players to simultaneously select t out of n numbers in a ticket without iterative selection. Given that the previous schemes [7,23,24] offer single participant lottery purchases on the Internet, Chen et al. [25] proposed an e-lottery purchase protocol that supports the joint purchase from multi-participants that enables them to safely and fairly participate in a mobile environment. Aiming to provide an online lottery protocol that does not rely on a trusted third party, Grumbach and Riemann [26] proposed a novel distributed e-lottery protocol based on the centralized e-lottery of Chow et al. [7] and incorporated the aforementioned multi-level hash chain verification phase of Liu et al. [23]. Considering that the existing works on e-lottery focus either on providing new functionalities (such as decentralization or threshold) or improving the hash chain or delay function, the building block of VRFs has received little attention. In this paper, we explore how to improve the efficiency of Chow et al.'s [7] e-lottery scheme by using aggregate VRFs.

2. Preliminaries

2.1. Verifiable Random Functions

Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y} \times \mathcal{P}$ be an efficient function, where the key space $\mathcal{K}$, domain $\mathcal{X}$, range $\mathcal{Y}$, and proof space $\mathcal{P}$ are dependent on the security parameter λ. Consider $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ as the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to (sk, pk)$ takes as input a security parameter λ and outputs a key pair (pk, sk). We say that sk is the secret key and pk is the verification key.
- $\mathsf{Eval}(sk, x) \to (y, \pi)$ takes as input the secret key sk and $x \in \mathcal{X}$ and outputs a function value $y \in \mathcal{Y}$ and a proof $\pi \in \mathcal{P}$. We write $\mathsf{Fun}_{sk}(x)$ to denote the function value y and $\mathsf{Prove}_{sk}(x)$ to denote the proof of correctness computed by Eval on input (sk, x).
- $\mathsf{Verify}(pk, x, y, \pi) \to \{0, 1\}$ takes as input the verification key pk, $x \in \mathcal{X}$, $y \in \mathcal{Y}$, and the proof $\pi \in \mathcal{P}$ and outputs a bit.

Definition 1. *We say that* $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is a* verifiable random function *(VRF) if all the following properties hold.*

1. Provability: *For all* $(pk, sk) \leftarrow \mathsf{Setup}(1^\lambda)$ *and inputs* $x \in \mathcal{X}$ *it holds: if* $(y, \pi) \leftarrow \mathsf{Eval}(sk, x)$, *then* $\mathsf{Verify}(pk, x, y, \pi) = 1$.

2. Uniqueness: *For all pk (not necessarily generated by* Setup*) and inputs* $x \in \mathcal{X}$*, there does not exist a tuple* (y_0, y_1, π_0, π_1) *such that:* $(1) y_0 \neq y_1$*,* $(2) \mathsf{Verify}(pk, x, y_0, \pi_0) = \mathsf{Verify}(pk, x, y_1, \pi_1) = 1$.
3. Pseudorandomness*: For all p.p.t. attackers* $D = (D_1, D_2)$*, there exists a negligible function* $\mu(\lambda)$ *such that:*

$$\Pr[(pk, sk) \leftarrow \mathsf{Setup}(1^\lambda); (x^*, \mathsf{st}) \leftarrow D_1^{\mathsf{Eval}(sk,\cdot)}(pk); y_0 = \mathsf{Fun}_{sk}(x^*); y_1 \leftarrow \mathcal{Y};$$
$$b \leftarrow \{0,1\}; b' \leftarrow D_2^{\mathsf{Eval}(sk,\cdot)}(y_b, \mathsf{st}) : b' = b \wedge x^* \notin L^{\mathsf{Eval}}] \leq \frac{1}{2} + \mu(\lambda),$$

where L^{Eval} *denotes the set of all inputs that D queries to its oracle* Eval.

Here, we note that Eval and Fun denote two different functions. The former denotes the function that outputs both function value y and proof π, while the latter denotes the function that only outputs function value y.

2.2. Bilinear Maps and the HW-VRF Scheme

Bilinear Groups. Let $\mathbb{G}$ and $\mathbb{G}_T$ be algebraic groups. A bilinear map is an efficient mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ which is both: (*bilinear*) for all $g \in \mathbb{G}$ and $a, b \leftarrow \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}$, then $e(g, g) \neq 1$.

Assumption 1 (q-Decisional Diffie–Hellman Exponent (q-DDHE))**.** *Let* $\mathbb{G}, \mathbb{G}_T$ *be groups of prime order* $p \in \Theta(2^\lambda)$*. For all p.p.t. adversaries* $\mathcal{A}$*, there exists a negligible function* μ *such that:*

$$\Pr[g, h \leftarrow \mathbb{G}; a \leftarrow \mathbb{Z}_p; y_0 = e(g, h)^{a^q}; y_1 \leftarrow \mathbb{G}_T; b \leftarrow \{0,1\};$$
$$b' \leftarrow \mathcal{A}(g, h, g^a, \ldots, g^{a^{q-1}}, g^{a^{q+1}}, \ldots, g^{a^{2q}}, y_b) : b = b'] \leq \frac{1}{2} + \mu(\lambda).$$

HW-VRF Scheme Here, we describe one of the elegant constructions of VRFs proposed by Hohenberger and Waters [13] (that is abbreviated as HW-VRF scheme). The latter is employed as a basis for our aggregate VRF scheme. HW-VRF is the first fully-secure VRF from the Naor-Reingold PRF [27] with exponential-size input space whose security is based on the so-called q-type complexity assumption, namely q-DDHE assumption, and is built as follows.

- $\mathsf{Setup}(1^\lambda, 1^\ell)$: The setup algorithm takes as input the security parameter λ as well as the input length ℓ. It firstly runs $\mathcal{G}(1^\lambda)$ to obtain the description of the groups $\mathbb{G}$, $\mathbb{G}_T$ and of a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. The description of $\mathbb{G}$ contains the generators $g, h \in \mathbb{G}$. Let $\{0,1\}^\ell$ be the input space. It next selects random values $u_0, u_1, \ldots, u_\ell \in \mathbb{Z}_p$ and sets $U_0 = g^{u_0}, U_1 = g^{u_1}, \ldots, U_\ell = g^{u_\ell}$. It then sets the keys as: $pk = (g, h, U_0, U_1, \ldots, U_\ell), sk = (u_0, u_1, \ldots, u_\ell)$.
- $\mathsf{Fun}(sk, x)$: For $x \in \{0,1\}^\ell$, the function Fun_{sk} evaluates $x = x_1 \| x_2 \| \ldots \| x_\ell$ as:

$$y = \mathsf{Fun}_{sk}(x) = e(g, h)^{u_0 \Pi_{i=1}^{\ell} u_i^{x_i}}.$$

- $\mathsf{Prove}(sk, x)$. This algorithm outputs a proof π, which is comprised as follows. Let $\pi_{\ell+1} = g^{u_0 \Pi_{i=1}^{\ell} u_i^{x_i}}$, for $j = 1$ to ℓ it computes: $\pi_j = g^{\Pi_{i=1}^{j} u_i^{x_i}}$. Set $\pi = (\pi_1, \ldots, \pi_\ell, \pi_{\ell+1})$.
- $\mathsf{Verify}(pk, x, y, \pi)$. Let $\pi = (\pi_1, \ldots, \pi_\ell, \pi_{\ell+1})$. To verify that y was computed correctly, proceed in a step-by-step manner by checking that

$$e(\pi_1, g) = \begin{cases} e(g, g), & \text{if } x_1 = 0; \\ e(U_1, g), & \text{otherwise.} \end{cases}$$

then for $i = 2, \ldots, \ell$ it checks, if the following equations are satisfied:

$$e(\pi_i, g) = \begin{cases} e(\pi_{i-1}, g), & \text{if } x_i = 0; \\ e(\pi_{i-1}, U_i), & \text{otherwise.} \end{cases}$$

Finally, it checks that $e(\pi_{\ell+1}, g) = e(\pi_\ell, U_0)$ and $e(\pi_{\ell+1}, h) = y$. It outputs 1, if and only if all checks verify. Otherwise, it outputs 0.

3. Static Aggregate VRFs

In a (static) aggregate PRF [10] (here, we call the aggregate PRF proposed by Cohen, Goldwasser, and Vaikuntanathan [10] as a static aggregate PRF since their aggregation algorithm needs the secret key of the PRF to be taken as input), there is an additional aggregation algorithm which given the secret key can (efficiently) compute the aggregated result of all the function values over a set of all the inputs in polynomial time, even if the input set is of super-polynomial size. Note that in an aggregate VRF, similarly to an aggregate PRF, an additional aggregation algorithm is brought into the ordinary VRF [1]. Thus, aggregate VRFs can be regarded as an extension of ordinary VRFs. The static aggregate VRF differs from a static aggregate PRF [10] in that given the secret key the aggregation operation is performed not only on the function values but also on the corresponding proofs. Moreover, the resulted aggregate function value can be publicly verified by using aggregate proof (together with the public key and the input subset), which proves that the aggregate function value is a correct result on the aggregation of all function values over the input subset.

Cohen, Goldwasser, and Vaikuntanathan [10] were the first to consider the notion of aggregate PRFs over the super-polynomial large but *efficiently recognizable* set classes. In their model, they treat the efficiently recognizable set ensemble as a family of predicates, i.e., for any set S there exists a polynomial-size boolean circuit $C : \{0,1\}^* \to \{0,1\}$ such that $x \in S$ if and only if $C(x) = 1$. Boneh and Waters [11] also employed such a predicate to define the concept of constrained PRFs with respect to a constrained set. In this paper, we employ the concept and formalization of the efficiently recognizable set in the definition of static aggregate VRFs.

Recall that a verifiable random function (VRF) [1] is a function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y} \times \mathcal{P}$ defined over a secret key space $\mathcal{K}$, a domain $\mathcal{X}$, a range $\mathcal{Y}$, and a proof space $\mathcal{P}$ (and these sets may be parameterized by the security parameter λ). Let $\mathsf{Fun} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ denote the mapping of random function evaluations on arbitrary inputs and $\mathsf{Prove} : \mathcal{K} \times \mathcal{X} \to \mathcal{P}$ denote the mapping of proof evaluations on inputs, each of which can be computed by a deterministic polynomial time algorithm.

Let $\Psi_\lambda : (\mathcal{Y}_\lambda, \mathcal{P}_\lambda)^* \to (\mathcal{Y}_\lambda, \mathcal{P}_\lambda)$ be the aggregation function that takes as inputs multiple pairs of values from the range $\mathcal{Y}_\lambda$ and the proof space $\mathcal{P}_\lambda$ of the function family, and aggregates them to output an aggregated function value in the range $\mathcal{Y}_\lambda$ and the corresponding aggregated proof in the proof space $\mathcal{P}_\lambda$.

Definition 2 (Static Aggregate VRF). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a VRF function family where each function $F \in \mathcal{F}_\lambda : \mathcal{K} \times \mathcal{X} \to \mathcal{Y} \times \mathcal{P}$ computable in polynomial time is defined over a key space $\mathcal{K}$, a domain $\mathcal{X}$, a range $\mathcal{Y}$ and a proof space $\mathcal{P}$. Let $\mathcal{S}$ be an efficiently recognizable ensemble of sets $\{\mathcal{S}_\lambda\}_\lambda$ where for any $S \in \mathcal{S}$, $S \subset \mathcal{X}$, and $\Psi_\lambda : (\mathcal{Y}_\lambda, \mathcal{P}_\lambda)^* \to (\mathcal{Y}_\lambda, \mathcal{P}_\lambda)$ be an aggregation function. We say that $\mathcal{F}$ is an $(\mathcal{S}, \Psi)$-static aggregate verifiable random function family (abbreviated $(\mathcal{S}, \Psi)$-sAgg-VRFs) if it satisfies:*

- **Efficient aggregation:** *There exists an efficient (computable in polynomial time) algorithm $\mathsf{Aggregate}_{F,\mathcal{S},\Psi}(sk, S) \to (y_{\mathsf{agg}}, \pi_{\mathsf{agg}})$ which on input the secret key sk of a VRF and a set $S \in \mathcal{S}$, outputs aggregated results $(y_{\mathsf{agg}}, \pi_{\mathsf{agg}}) \in \mathcal{Y} \times \mathcal{P}$ such that for any $S \in \mathcal{S}$, $\mathsf{Aggregate}_{F_{sk},\mathcal{S},\Psi}(sk, S) = \Psi(F_{sk}(x_1), \ldots, F_{sk}(x_{|S|}))$ where $F_{sk}(x_i) = (y_i = \mathsf{Fun}_{sk}(x_i), \pi_i = \mathsf{Prove}_{sk}(x_i))$ for $i = 1, \ldots, |S|$;*
- **Verification for aggregation:** *There exists an efficient (computable in polynomial time) algorithm $\mathsf{AggVerify}(pk, S, y_{\mathsf{agg}}, \pi_{\mathsf{agg}}) \to \{0,1\}$ which on input the aggregated function value y_{agg} and the proof*

π_{agg} *for an ensemble* $S \in \mathcal{S}$ *of the domain, verifies if it holds that* $y_{\mathsf{agg}} = \Psi(\mathsf{Fun}_{sk}(x_1), \ldots, \mathsf{Fun}_{sk}(x_{|S|}))$ *using the aggregated proof* π_{agg}.

- **Correctness of aggregated values:** *For all* $(pk, sk) \leftarrow \mathsf{Setup}(1^\lambda)$, *set* $S \in \mathcal{S}$ *and the aggregate function* $\Psi \in \Psi_\lambda$, *let* $(y, \pi) \leftarrow \mathsf{Eval}(sk, x)$ *and* $(y_{\mathsf{agg}}, \pi_{\mathsf{agg}}) \leftarrow \mathsf{Aggregate}_{F,\mathcal{S},\Psi}(sk, S)$, *then* $\mathsf{AggVerify}(pk, S, y_{\mathsf{agg}}, \pi_{\mathsf{agg}}) = 1$.
- **Pseudorandomness:** *For all p.p.t. attackers* $D = (D_1, D_2)$, *there exists a negligible function* $\mu(\lambda)$ *s.t.:*

$$\begin{aligned}\Pr[(pk, sk) \leftarrow \mathsf{Setup}(1^\lambda); (x^*, \mathsf{st}) \leftarrow D_1^{\mathsf{Eval}(sk,\cdot),\mathsf{Aggregate}_{F,\mathcal{S},\Psi}(sk,\cdot)}(pk); b \leftarrow \{0,1\};\\ y_0 = \mathsf{Fun}_{sk}(x^*);\ y_1 \leftarrow \mathcal{Y}; b' \leftarrow D_2^{\mathsf{Eval}(sk,\cdot),\mathsf{Aggregate}_{F,\mathcal{S},\Psi}(sk,\cdot)}(y_b, \mathsf{st}) :\\ b' = b \wedge C_{S_i}(x^*) = 0 \textit{for all} S_i \in L^{\mathsf{Agg}} \wedge x^* \notin L^{\mathsf{Eval}}] \leq \frac{1}{2} + \mu(\lambda),\end{aligned}$$

where L^{Eval} *is the set of all inputs that* D *queries to its oracle* Eval, L^{Agg} *consists of all the sets* S_i *that* D *queries to its oracle* Aggregate, *and* C_{S_i} *is the polynomial-size boolean circuit that is able to recognize the ensemble* S_i.

- **Compactness:** *There exists a polynomial* $poly(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$, $x \in \mathcal{X}$, *set* $S \in \mathcal{S}$ *and the aggregate function* $\Psi \in \Psi_\lambda$, *it holds with overwhelming probability over* $(pk, sk) \leftarrow \mathsf{Setup}(1^\lambda)$, $(y, \pi) \leftarrow \mathsf{Eval}(sk, x)$ *and* $\mathsf{Aggregate}_{F,\mathcal{S},\Psi}(sk, S) \rightarrow (y_{\mathsf{agg}}, \pi_{\mathsf{agg}})$ *that the resulting aggregated value* y_{agg} *and aggregated proof* π_{agg} *has size* $|y_{\mathsf{agg}}|, |\pi_{\mathsf{agg}}| \leq poly(\lambda, |x|)$. *In particular, the size of* y_{agg} *and* π_{agg} *are independent of the size of the set* S.

We stress that the set S over which the aggregation is performed can be *super-polynomially* large. Clearly, given exponential numbers of values $F_{sk}(\cdot)$, it is impossible to perform aggregation on them but yet, we show how to efficiently compute the aggregation function on an exponentially large set with respect to a concrete VRF given the secret key.

Some explanations on the notion of static aggregate VRFs. Firstly, the algorithm $\mathsf{Aggregate}_{F,\mathcal{S},\Psi}$ achieves an efficient aggregation on function values/proofs over super-polynomially large sets S in polynomial time. We stress that our aim is to work on super-polynomially large sets, since, for any constant size of sets, the (productive) aggregation can be computed trivially, given the function value/proof pairs on all inputs in such a set. Secondly, the verification algorithm AggVerify is employed to efficiently verify the correctness of the aggregated function values y_{agg}. Given $\{(x_i, y_i, \pi_i)\}_{i=1}^{|S|}$ and the aggregated function value y_{agg}, there is a trivial way to verify the correctness of y_{agg}, by verifying the correctness of each tuple (x_i, y_i, π_i) for $i = 1, \ldots, \mid S \mid$ and then checking if $y_{\mathsf{agg}} = \Pi_{i=1}^{|S|} y_i$, which is not computable in polynomial time if S is a super-polynomially large set. Therefore, our main concern is to achieve efficient verification on y_{agg} via the corresponding proof π_{agg}, the size of which is independent of the size of S. Thirdly, the condition $\mathsf{AggVerify}(pk, S, y_{\mathsf{agg}}, \pi_{\mathsf{agg}}) = 1$ is interpreted as that value y_{agg} is a correct result on the aggregation of $\{\mathsf{Fun}_{sk}(x_i)\}_{i=1}^{|S|}$, i.e., $y_{\mathsf{agg}} = \Psi(\mathsf{Fun}_{sk}(x_1), \ldots, \mathsf{Fun}_{sk}(x_{|S|}))$, by using the corresponding proof π_{agg}. We note that the verification for the aggregation does not violate the uniqueness of the underlying basic VRF. Indeed, there probably exist different sets S_1 and S_2 that result in a same y_{agg}, but the uniqueness for any input point $x \in S_1$ ($x \in S_2$) always holds. Looking ahead, in our instantiation of aggregate VRFs, to find two sets $S_1 \neq S_2$ such that $\prod_{x_i \in S_1} \mathsf{Fun}_{sk}(x_i) = \prod_{x_i \in S_2} \mathsf{Fun}_{sk}(x_i)$ is computationally hard, without knowledge of sk. Lastly, the condition $\mathsf{AggVerify}(pk, S, y_{\mathsf{agg}}, \pi_{\mathsf{agg}}) = 1$ does not imply $\mathsf{Verify}(pk, x_i, y_i, \pi_i) = 1$ for all $i = 1, \ldots, \mid S \mid$, since by maintaining a correct pair $(y_{\mathsf{agg}}, \pi_{\mathsf{agg}})$, we always can alter any two tuples as $(x_i, y_i \cdot r, \pi_i)$ and $(x_j, y_j \cdot r^{-1}, \pi_j)$ for any random $r \in \mathbb{G}$, which means $\mathsf{Verify}(x_i, y_i \cdot r, \pi_i) = \mathsf{Verify}(x_j, y_j \cdot r^{-1}, \pi_j) = 0$.

3.1. A Static Aggregate VRF for Bit-Fixing Sets

We now propose a static aggregate VRF, whose aggregation function is to compute products over bit-fixing sets. In a nutshell, a bit-fixing set consists of bit-strings, which match a particular bit pattern.

We naturally represent such sets by a string in $\{0,1,\perp\}^{poly(\lambda)}$ with 0 and 1 indicating a fixed bit location and $\perp$ indicating a free bit location. To do so, we define for a pattern string $v \in \{0,1,\perp\}^{poly(\lambda)}$ the bit-fixing set as $S_v = \{x \in \{0,1\}^{poly(\lambda)} : \forall i, x_i = v_i \text{or} v_i = \perp\}$.

We show based on an elegant construction of VRFs proposed by Hohenberger and Waters [13] (abbreviated as HW-VRF scheme) how to compute the productive aggregation function over a bit-fixing set in polynomial time; thus, yielding a static aggregate VRF. Please refer to Section 2.2 for detailed description of HW-VRF scheme. The aggregation algorithm for bit-fixing sets takes as input the VRF secret key sk and a string $v \in \{0,1,\perp\}^{\ell}$. Let $\text{Fixed}(v) = \{i \in [\ell] : v_i \in \{0,1\}\}$ and $|\text{Fixed}(v)| = \tau$. The aggregation algorithm and the verification algorithm for an aggregated function value and the corresponding proof works as follows:

- $\text{Aggregate}(sk, v)$:

  Let $\pi_0^{\mathsf{agg}} := g^{2^{\ell-\tau}}$. We define the aggregated proof as $\pi^{\mathsf{agg}} = (\pi_1^{\mathsf{agg}}, \ldots, \pi_\ell^{\mathsf{agg}}, \pi_{\ell+1}^{\mathsf{agg}})$, where for $i = 1, \ldots, \ell$,

$$\pi_i^{\mathsf{agg}} = \begin{cases} (\pi_{i-1}^{\mathsf{agg}})^{u_i^{v_i}} & \text{if} i \in \text{Fixed}(v) \\ (\pi_{i-1}^{\mathsf{agg}})^{(u_i+1)/2} & \text{if} i \notin \text{Fixed}(v). \end{cases}$$

  and $\pi_{\ell+1}^{\mathsf{agg}} = (\pi_\ell^{\mathsf{agg}})^{u_0}$. The aggregated function value is defined as:

$$y^{\mathsf{agg}} = e(g,h)^{u_0 \cdot (\prod_{i \in \text{Fixed}(v)} u_i^{v_i}) \cdot (\prod_{i \in [\ell] \setminus \text{Fixed}(v)} (u_i+1))}.$$

- $\text{AggVerify}(pk, v, y^{\mathsf{agg}}, \pi^{\mathsf{agg}})$:

  Parse $\pi^{\mathsf{agg}} = (\pi_1^{\mathsf{agg}}, \ldots, \pi_\ell^{\mathsf{agg}}, \pi_{\ell+1}^{\mathsf{agg}})$. Let $\pi_0^{\mathsf{agg}} = g^{2^{\ell-\tau}}$. The aggregation verification algorithm checks if the following equations are satisfied: for $i = 1, \ldots, \ell$

$$e(g, \pi_i^{\mathsf{agg}}) = \begin{cases} e(\pi_{i-1}^{\mathsf{agg}}, g) & \text{if } i \in \text{Fixed}(v) \text{and} v_i = 0 \\ e(\pi_{i-1}^{\mathsf{agg}}, U_i) & \text{if } i \in \text{Fixed}(v) \text{and} v_i = 1 \\ e(\pi_{i-1}^{\mathsf{agg}}, g \cdot U_i)^{1/2} & \text{if } i \notin \text{Fixed}(v). \end{cases}$$

  and $e(\pi_{\ell+1}^{\mathsf{agg}}, g) = e(\pi_\ell^{\mathsf{agg}}, U_0)$ and $e(\pi_{\ell+1}^{\mathsf{agg}}, h) = y^{\mathsf{agg}}$. Output 1 if and only if all checks verify. Otherwise, output 0.

Letting $\mathcal{S}^{\text{BF}} = \{\mathcal{S}^{\text{BF}}_{\ell(\lambda)}\}_{\lambda \in \mathbb{N}}$ where $\mathcal{S}^{\text{BF}}_{\ell(\lambda)} = \{0,1,\perp\}^{\ell}$ is the bit-fixing sets on $\{0,1\}^{\ell}$, we now prove the following theorem:

Theorem 1. *Let $\epsilon > 0$ be a constant. Choose the security parameter $\lambda = \Omega(\ell^{1/\epsilon})$, and assume the $(2^{\lambda^\epsilon}, 2^{-\lambda^\epsilon})$-hardness of q-DDHE over the group $\mathbb{G}$ and $\mathbb{G}_T$. Then, the collection of verifiable random functions F defined above is a secure aggregate VRF with respect to the subsets $\mathcal{S}^{BF}$ and the product aggregation function over $\mathbb{G}$ and $\mathbb{G}_T$.*

The compactness follows straightforward, since the aggregated function value $y^{\mathsf{agg}} \in \mathbb{G}_T$ and the aggregated proof $\pi^{\mathsf{agg}} = (\pi_1^{\mathsf{agg}}, \ldots, \pi_{\ell+1}^{\mathsf{agg}}) \in \mathbb{G}^{\ell+1}$, the sizes of which are independent of the size of the bit-fixing set S_v, i.e., $2^{\ell-\tau}$.

The proof for pseudorandomness is similar to that of HW-VRF scheme in [13] since our static aggregate VRF is built on the ground of HW-VRF and the only phase we need to deal with in the proof is to simulate the responses of the aggregation queries. Here, we provide the simulation routine that the q-DDHE solver executes to act as a challenger in the pseudorandomness game of the aggregated VRFs. The detailed analysis of the game sequence is similar to the related descriptions in [13].

Proof of Theorem 1. Let $Q(\lambda)$ be a polynomial upper bound on the number of queries made by a p.p.t. distinguisher D to the oracles Eval and Aggregate. We use D to create an adversary $\mathcal{B}$ such that,

if D wins in the pseudorandomness game for aggregate VRFs with probability $\frac{1}{2} + \frac{3\epsilon}{64Q(\ell+1)}$, then $\mathcal{B}$ breaks the q-DDHE assumption with probability $\frac{1}{2} + \frac{3\epsilon}{64Q(\ell+1)}$, where $q = 4Q(\ell+1)$, and ℓ is the input length of the static Agg-VRFs.

Given $(\mathbb{G}, p, g, h, g^a, \ldots, g^{a^{q-1}}, g^{a^{q+1}}, \ldots, g^{a^{2q}}, y)$, to distinguish $y = e(g,h)^{a^q}$ from $y \leftarrow \mathbb{G}_T$, $\mathcal{B}$, proceed as follows:

Setup. Set $m = 4Q$ and choose an integer $k \xleftarrow{\$} [0, \ell]$. It then picks random integers $r_1, \ldots, r_\ell, r'$ from the interval $[0, m-1]$ and random elements $s_1 \ldots, s_\ell, s' \in \mathbb{Z}_p$, which are all kept internal by $\mathcal{B}$.

For $x \in \{0,1\}^\ell$, let x_i denote the ith bit of x. Define the following functions:

$$C(x) = m(1+k) + r' + \sum_{i=1}^{\ell} x_i r_i, \hat{C}(x,i) = \sum_{j=1}^{i} x_j r_j, J(x) = s' \prod_{i=1}^{\ell} s_i^{x_i}, \hat{J}(x,i) = \prod_{j=1}^{i} s_j^{x_j}$$

$\mathcal{B}$ sets $U_0 = (g^{a^{m(1+k)+r'}})^{s'}$ and $U_i = (g^{a^{r_i}})^{s_i}$ for $i = 1, \ldots, \ell$. It sets the public key as $(\mathbb{G}, p, g, h, U_0, \ldots, U_\ell)$, and the secret key implicitly includes the values $u_0 = a^{m(1+k)+r'} s'$ and $\{u_i = a^{r_i} s_i\}_{i \in [1,\ell]}$.

Oracle Queries to $\mathsf{Eval}(sk, \cdot)$**.** The distinguisher D will make queries of VRF evaluations and proofs. On receiving an input x, $\mathcal{B}$ first checks if $C(x) = q$ and aborts if this is true. Otherwise, it defines the function value as $F(x) = e((g^{a^{C(x)}})^{J(x)}, h)$, and the corresponding proof as $\pi = (\pi_0, \pi_1, \ldots, \pi_\ell)$ where $\pi_0 = (g^{a^{C(x)}})^{J(x)}$, $\pi_i = (g^{a^{\hat{C}(x,i)}})^{\hat{J}(x,i)}$ for $i = 1, \ldots, \ell$. Note that for any $x \in \{0,1\}^\ell$ it holds:

1. The maximum value of $C(x)$ is $m(1+\ell) + (1+\ell)(m-1) = (2m-1)(1+\ell) < 2m(1+\ell) = 2q$.
2. The maximum value of $\hat{C}(x,i)$ is $\ell(m-1) < m(1+\ell) = q$ for $i \in [\ell]$.

As a result, if $C(x) \neq q$, $\mathcal{B}$ could answer all the Eval queries.

Oracle Queries to $\mathsf{Aggregate}_{F_{sk},\mathcal{S},\Psi}(\cdot)$**.** The distinguisher D will also make queries for aggregate values. On receiving a pattern string $v \in \{0,1,\perp\}^\ell$, $\mathcal{B}$ uses the above secret key to compute the aggregated proof and the aggregate function value. More precisely, $\mathcal{B}$ answers the query $\mathsf{Aggregate}_{F_{sk},\mathcal{S},\Psi}(S_v)$ as follows: Let $\pi_0^{\mathsf{agg}} := g^{2^{\ell-\tau}}$. Since the aggregated proof is defined as $\pi^{\mathsf{agg}} = (\pi_1^{\mathsf{agg}}, \ldots, \pi_\ell^{\mathsf{agg}}, \pi_{\ell+1}^{\mathsf{agg}})$, where, for $i = 1, \ldots, \ell$,

$$\pi_i^{\mathsf{agg}} = \begin{cases} (\pi_{i-1}^{\mathsf{agg}})^{u_i^{v_i}} & \text{if } i \in \text{Fixed}(v) \\ (\pi_{i-1}^{\mathsf{agg}})^{(u_i+1)/2} & \text{if } i \notin \text{Fixed}(v). \end{cases}$$

and $\pi_{\ell+1}^{\mathsf{agg}} = (\pi_\ell^{\mathsf{agg}})^{u_0}$, $\mathcal{B}$ will compute concretely:

$$\pi_1^{\mathsf{agg}} = \begin{cases} (g^{a^{v_1 r_1} s_1^{v_1}})^{2^{\ell-\tau}} & \text{if } 1 \in \text{Fixed}(v) \\ g^{2^{\ell-\tau-1}(1+a^{r_1}s_1)} & \text{if } 1 \notin \text{Fixed}(v) \end{cases}$$

and, for $j = 2, \ldots, \ell$, $\pi_j^{\mathsf{agg}} = g^{2^{\ell-\tau-\bar{\tau}_j}\left(\Pi_{i \in [j] \cap \text{Fixed}(v)} (a^{r_i} s_i)^{v_i}\right)\left(\Pi_{i \in \text{Flex}(v_j)} (1+a^{r_i} s_i)\right)}$ where $\text{Flex}(v_j) := \{i \in [\ell] : i \leq j \wedge v_i = \perp\}$ and $\bar{\tau}_j := |\text{Flex}(v_j)|$. The above value could be computed by $\mathcal{B}$ through its knowledge of r_i, s_i. The value of

$$\pi_{\ell+1}^{\mathsf{agg}} = g^{\left(\prod_{i \in [\ell] \cap \text{Fixed}(v)} (a^{r_i} s_i)^{v_i}\right) \cdot \left(\prod_{i \in [\ell] \setminus \text{Fixed}(v)} (1+a^{r_i} s_i)\right) \cdot a^{m(1+k)+r'} \cdot s'}$$

can be handled similarly using m, k, r', s'. While the aggregated function value is defined as $y^{\mathsf{agg}} = e(\pi_{\ell+1}^{\mathsf{agg}}, h)$.

Challenge. D will send a challenge input x^* with the condition that x^* is never queried to its Eval oracle. If $C(x^*) = q$, $\mathcal{B}$ returns the value y. When D responds with a bit b', $\mathcal{B}$ outputs b' as its

guess to its own q-DDHE challenger. If $C(x^*) \neq q$, $\mathcal{B}$ outputs a random bit as its guess. This ends our description of q-DDHE adversary $\mathcal{B}$. □

Remark 1. Discussion on the impossibility of productive aggregation on JN-VRF for bit-fixing sets. *Recently, based on q-DDH-assumption, Jager and Niehues [14] proposed the currently most efficient VRFs (that is abbreviated as JN-VRF scheme) with full adaptive security in the standard model. JN-VRF almost enjoys the same framework of HW-VRF, and the only difference is that in the former an admissible hash function* H_{AHF} *is applied on inputs x before evaluating the function value and corresponding proof, while the latter is not. We stress that hash function* $H_{\mathsf{AHF}} : \{0,1\}^{\ell} \to \{0,1\}^{n}$ *on inputs x destroys the nice pattern of all inputs in a bit-fixing set, which implies that, for any* $x \in S_v$*, i.e., for all* $i \in [\ell]$, $x_i = v_i \vee v_i = \perp$*, there does not exist a bit-string* $v'\{0,1,\perp\}^n$ *such that* $H_{\mathsf{AHF}}(x) = h_1 \| \ldots \| h_n \notin S_{v'}$*, where* $S_{v'} = \{h_j \in \{0,1\}^n : \forall j, h_j = v'_j or v'_j = \perp\}$*. Otherwise, it is possible to find the collisions of* H_{AHF}*. Therefore, given* exponential *numbers of values* $F_{sk}(H_{\mathsf{AHF}}(x))$*, it is impossible to perform productive aggregation over them efficiently by using the same technique as in the last subsection.*

3.2. Efficiency Analysis

Analysis of Costs. The instantiation in Section 3.1 is very compact since the aggregated function value consists of a single element in $\mathbb{G}_T$, while the aggregated proof is composed of $\ell + 1$ elements in $\mathbb{G}$, which are independent of the size of a set S. The Aggregate algorithm simply requires at most ℓ multiplications plus one exponentiation to compute y^{agg} and $\ell + 2$ exponentiations to evaluate π^{agg}, which needs much less computation compared to computing $2^{\ell-\tau}$ multiplications to obtain y^{agg} and $2^{\ell-\tau} \cdot (\ell + 1)$ multiplications to obtain π^{agg} on all $2^{\ell-\tau}$ number of inputs in S. The AggVerify algorithm simply requires at most $(2\ell + 3)$ pairing operations, while $2^{\ell-\tau} \cdot (2\ell + 3)$ pairings are needed for verifying $2^{\ell-\tau}$ number of function values/proofs on all inputs in S.

We summarize the cost for the Aggregate and AggVerify algorithms in Table 1, where MUL is the shortened form of the multiplication operation, EXP is the abbreviation for the exponentiation operation, and ADD denotes the addition operation.

Table 1. The computation operations for the static aggregate VRF scheme with respect to bit-fixing sets.

Scheme	Assump.	Input Length	Cost for Aggregating Function Value	Cost for Aggregating Proof	Cost on Verification for Aggregation
HW-VRF [13]	q-DDHE	ℓ	$2^{\ell-\tau}$ MUL on $\mathbb{G}_T$	$2^{\ell-\tau} \cdot (\ell + 1)$ MUL on $\mathbb{G}$	$2^{\ell-\tau} \cdot (\ell + 1)$ bilinear pairings
Our static Agg-VRFs for bit-fixing sets	q-DDHE	ℓ	ℓ MUL on $\mathbb{Z}_p$ & one EXP	$(2\ell + 3)$ EXP	$(2\ell + 3)$ bilinear pairings

3.3. Implementation and Experimental Results

Choice of elliptic curves and pairings. In our implementation, we use Type A curves as described in [28], which can be defined as follows. Let q be a prime satisfying $q = 3 \mod 4$ and let p be some odd dividing $q + 1$. Let E be the elliptic curve defined by the equation $y^2 = x^3 + x$ over $\mathbb{F}_q$; then, $E(\mathbb{F}_q)$ is supersingular, $\#E(\mathbb{F}_q) = q + 1$, $\#E(\mathbb{F}_{q^2}) = (q + 1)^2$, and $\mathbb{G} = E(\mathbb{F}_q)[p]$ is a cyclic group of order p with embedding degree $k = 2$. Given map $\Psi(x, y) = (-x, iy)$, where i is the square root of -1, Ψ maps points of $E(\mathbb{F}_q)$ to points of $E(\mathbb{F}_{q^2}) \backslash E(\mathbb{F}_q)$, and if f denotes the Tate pairing on the curve $E(\mathbb{F}_{q^2})$, then defining $e : \mathbb{G} \times \mathbb{G} \to \mathbb{F}_{q^2}$ by $e(P, Q) = f(P, \Psi(Q))$ gives a bilinear nondegenerate map. For more details about the choice of parameters, please refer to [28]. In our case, we use the standard parameters proposed by Lynn [28] (https://crypto.stanford.edu/pbc/), where q has 126 bits and $p = 7$ 307508186654516213611192455715049014059765596 17. To generate random elements, we use libsodium (https://libsodium.gitbook.io/). Our implementation uses the programming language "C" and the GNU Multiple Precision Arithmetic for arithmetic with big numbers. We use the GCC version 10.0.1 with the following compilation flags: "-O3 -m64 -fPIC -pthread -MMD -MP -MF".

Implementing HW-VRF. In our implementation, we use the bilinear map as pairing implemented by Lynn [28] for the BLS signature scheme. We notice that, when computing the function value $\mathsf{Fun}_{sk}(x) = e(g,h)^{u_0\Pi_{i=1}^{\ell} u_i^{x_i}}$, we usually compute first the bilinear $e(g,h)$, and then do the exponentiation. However, it is expensive to do the exponentiation of an element in $\mathbb{G}_T$. To improve the efficiency of computing $\mathsf{Fun}_{sk}(x)$, we use the following mathematical trick: $e(g,h)^{ab} = e(g^a, h^b)$, which implies that we calculate $\mathsf{Fun}_{sk}(x)$ as $e(g^{u_0}, h^{\Pi_{i=1}^{\ell} u_i^{x_i}})$. Since the computation of g^a (or h^b) corresponds to the scalar multiplication of a point P (or Q) by a scalar a (or b), using this trick, we avoid the exponentiation on an element in $\mathbb{G}_T$ by requiring cost of two scalar multiplications of a point of the curve.

Implementing our static Agg-VRFs. Since p is fixed, when calculating the aggregated proof as $\pi_i^{\mathsf{agg}} := (\pi_{i-1}^{\mathsf{agg}})^{(u_i+1)/2}$, we can precompute the inversion of 2 and thus only need to compute $(\pi_{i-1}^{\mathsf{agg}})^{(u_i+1)\mathsf{inv}(2)}$ by the scalar multiplication of a point on curve with scalar $(u_i+1) \star \mathsf{inv}(2)$. We use a similar approach when computing $e(\pi_{i-1}^{\mathsf{agg}}, g \cdot U_i)^{1/2}$; in this case, we always perform $e((\pi_{i-1}^{\mathsf{agg}})^{\mathsf{inv}(2)}, g \cdot U_i)$. Again, $(\pi_{i-1}^{\mathsf{agg}})^{\mathsf{inv}(2)}$ corresponds to the scalar multiplication of a point with scalar $\mathsf{inv}(2)$, while $g \cdot U_i$ corresponds to the additive operation on two points on the elliptic curve.

Comparison. We tested the performance of our static Agg-VRFs in comparison to a standard (non-aggregate) VRF, for five different input lengths, i.e., 56, 128, 256, 512, and 1024 bits. In all cases, we set the size of the fixed-bit equal to 20. Thus, naturally, we wanted to compare the efficiency of our aggregated VRF versus the evaluation and corresponding verification of 2^{36}, 2^{108}, 2^{236}, 2^{492}, and 2^{1004} VRF values. To perform our comparisons, we recorded the verification time for 100 pairs of function values and their corresponding proofs, if the verification is performed one-by-one (i.e., without using the aggregation) versus the corresponding performance of employing our proposed static aggregate VRF. Obviously, it holds $100 \ll 2^{36}$, $100 \ll 2^{108}$, $100 \ll 2^{236}$, $100 \ll 2^{492}$, and $100 \ll 2^{1004}$. In fact, it is fine to choose any number that is smaller than 2^{36}. We choose 100 to have sensible running time for the performance of the standard (non-aggregate) VRF. By taking the 56 bits input length with 20 fixed bits as an example, the bit-fixing set should contain 2^{36} elements; then, we should consider the verification time for 2^{36} pairs of function values-proofs, which is drastically larger than the running time when we evaluate the verification for only 100 pairs. Thus, showing that our aggregate VRF is much more efficient than the evaluation and corresponding verification of 100 VRF values obviously implies that it is more efficient than the evaluation and corresponding verification of 2^{36}, 2^{108}, 2^{236}, 2^{492}, and 2^{1004} VRF values, correspondingly.

Table 2 shows the result of our experiments. The column "Verify" corresponds to the required time for verifying a single pair of function value/proof. We tested how much time it costs to aggregate all the function values and their proofs for inputs belonging to the bit-fixing set. Furthermore, we evaluated the verification time to check the aggregated function value/proof. The column "Total Verification" corresponds to the total required time for verifying 100 pairs of function values/proofs via the standard VRFs (i.e., verification one-by-one), while the column "AggVerify" represents the costing time for verifying the aggregated value/proof via aggregate VRF (i.e., aggregated verification algorithm). The experimental results show that, even for 1024 bits of inputs, the aggregation of 2^{1004} pairs of function values/proofs can be computed very efficiently in 6881 ms. Moreover, the time required to verify their aggregated function values/proofs of 2^{1004} pairs only increases 50% compared to the verification time for each single function value/proof pair of HW-VRFs.

Table 2. Running time (milliseconds) of the aggregate VRFs and standard (non-aggregate) VRF.

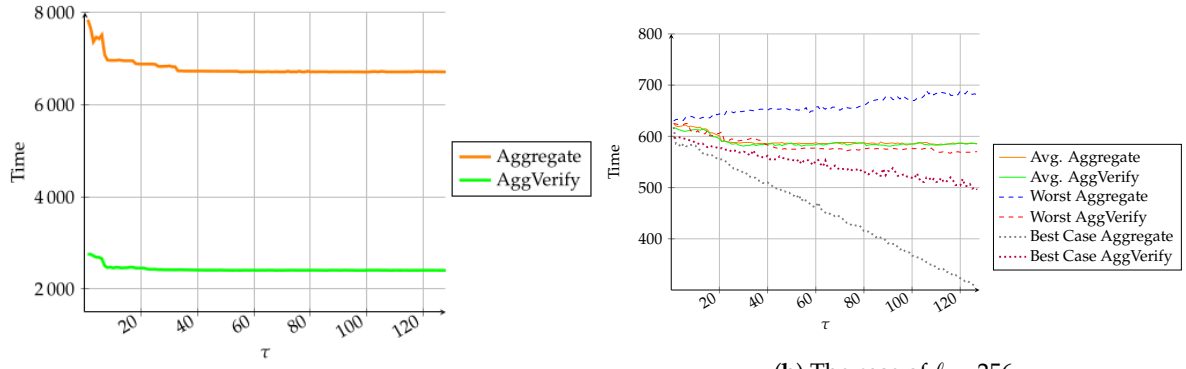
	HW-VRFs [13]			**Aggregate VRFs**		
Input Length (bits)	**Verify (ms)**	**Blocks Num.**	**Total Verification (ms)**	**Fixed-Bit Size**	**Aggregate (ms)**	**Our AggVerify (ms)**
56	89	100	949	20	41	122
128	197	100	22371	20	196	298
256	472	100	52199	20	602	579
512	842	100	95233	20	1924	1212
1024	1556	100	164129	20	6881	2459

We stress that our implementation is hardware independent. The only requirement is to have a compiler that is able to translate C code to the specific architecture. To give an estimation of what would happen if a different frequency in a computer architecture is used to run our code for HW-VRFs as well as our aggregate VRFs, we considered the original run using 56, 128, 256, 512, and 1024 bits, respectively. Then, we computed the difference between the frequencies and multiply for this result, as shown in Table 3. For different frequencies (GHz), the verification time for the aggregated function values/proofs increases 30–50%, compared to that for each single function value/proof pair of the HW-VRFs, as shown in Table 3.

Table 3. Table of estimation of time for different frequencies.

Size	Frequency (GHz)	Time (ms) for AggVerify	Time (ms) for Verify of HW-VRFs [13]
56	1.6	122	89
	2.1	85	62
	3.0	70	51
128	1.6	298	197
	2.1	208	138
	3.0	172	114
256	1.6	579	472
	2.1	405	330
	3.0	335	274
512	1.6	1212	842
	2.1	848	589
	3.0	702	488
1024	1.6	2459	1556
	2.1	1696	1089
	3.0	1426	902

Moreover, we performed experiments for the cases where the input lengths ℓ are equal to 256 (depicted in Figure 1b) and 1024 (in Figure 1a), respectively, by choosing different numbers τ of the fixed bits to see the variation of the costing time on the aggregation and verification processes. When $\ell = 256$, we ran experiments for three cases, i.e., worst-case where all τ fixed bits are 1, best-case where all τ fixed bits are 0, and average-case where τ fixed bits are chosen at random from $\{0, 1\}$. In the worst-case, the Aggregate algorithm requires 256 multiplications plus 1 exponentiation to compute y^{agg} and 258 exponentiation to evaluate π^{agg}, while the AggVerify algorithm requires 515 pairing operations, as shown in Figure 1b with square dot dashed line, which cost almost the same amount of time with different τ. In the best-case, the Aggregate algorithm requires $(256 - \tau)$ multiplications plus 1 exponentiation to compute y^{agg} and $(258 - \tau)$ exponentiation to evaluate π^{agg}, while the AggVerify algorithm requires $(516 - \tau)$ pairing operations, as shown in Figure 1b with round dot dashed line, where the running time decreases with the increase of τ. The average-case, as shown with solid lines in Figure 1b, lies between the range of the best-case and the worst-case. When $\ell = 1024$, we show the time cost on the aggregation and verification algorithms in average-case, i.e., for randomly chosen τ fixed bits.



(**a**) The case of $\ell = 1024$

(**b**) The case of $\ell = 256$

Figure 1. Time in milliseconds with respect to different numbers of fixed bits τ in the aggregate VRFs considering the cases of $\ell = 256$ and $\ell = 1024$.

4. Application to the E-Lottery Scheme

4.1. Discussion on the Practical Instantiation of Chow et al.'s E-Lottery

Chow et al. [7] proposed an e-lottery scheme based on VRFs, where a random number generation mechanism is required to determine not only a winning number (chosen from a predefined domain) but also the public verifiability of the winning result, which guarantees that the dealer cannot cheat in the random number generation process. Let the numbers used in the lottery game be $\{1, 2, \ldots, N_{\mathsf{max}}\}$. To generate the winning number, Chow et al. [7] employed a VRF that maps a bit-string of k length into a bit-string of l length. More precisely, the dealer firstly computes $(w_0, \pi_0) := (\mathsf{VRF.Fun}(\mathsf{sk}_{\mathsf{VRF}}, d), \mathsf{VRF.Prove}(\mathsf{sk}_{\mathsf{VRF}}, d))$, where d is the 'hash value' of all tickets sold so far. If $w_0 > N_{\mathsf{max}}$, then the dealer iteratively calculates $(w_i, \pi_i) := (\mathsf{VRF.Fun}(\mathsf{sk}_{\mathsf{VRF}}, w_{i-1} \| d), \mathsf{VRF.Prove}(\mathsf{sk}_{\mathsf{VRF}}, w_{i-1} \| d))$ for $i = 1, 2, \ldots$, until obtaining (w_t, π_t) such that $w_t \leq N_{\mathsf{max}}$. Afterwards, the dealer publishes the final (w_t, π_t) and the intermediate tuples (w_i, π_i) for $i = 0, 1, \ldots, t-1$ of VRF function values and their corresponding proofs.

To instantiate such an e-lottery scheme, a practical and concrete VRF scheme is needed. To the best of our knowledge, so far, most of the efficient instantiations of VRFs are based on bilinear maps (e.g., [13,14,18,19,22]), where the input spaces of VRFs are defined over binary strings and the function values on inputs are elements in a group $\mathbb{G}_T$, i.e., $w_i \in \mathbb{G}_T$, while the verification needs to compute some pairings.

Chow et al.'s construction [7] seems to be very promising when considering an ideal case that after a small number t of times that the VRF is applied, a function value w_t such that $w_t \leq N_{\mathsf{max}}$ can be obtained successfully. Nevertheless, what if t is a large number? Then, it implies that the dealer needs to calculate the VRF more times, while the player needs to verify the correctness of more tuples in order to verify the winning result, which also requires more consumption on pairing-computations from both the dealer and the player.

In the following section, we show how we modify Chow et al.'s e-lottery scheme [7] by employing our aggregate VRF, in order to reduce the computational overhead for the verification process of each player.

4.2. An E-Lottery Scheme Based on Aggregate VRFs

Observing that the intermediate tuple (w_i, π_i) is a VRF function value/proof on an input $w_{i-1} \| d$, which has the same last $\mid d \mid$ bits and consists of a bit-fixing set. Thus, we compress all the intermediate tuples into a single function value $w^{\mathsf{agg}} = e(g,h)^{u_0 \cdot (\prod_{i=\ell-|d|+1}^{\ell} u_i^{d_i}) \cdot (\prod_{i=1}^{\ell-|d|}(u_i+1))}$ and corresponding proof π^{agg}. If the exponent of w^{agg} is less than N_{max}, w^{agg} can be set as the winning number. Otherwise, the dealer can output a value $w'^{agg} = e(g,h)^{u_0 \cdot (\prod_{i=\ell-|d|+1}^{\ell} u_i^{d_i}) \cdot (\prod_{i=1, i \notin \{j_1, \ldots, j_k\}}^{\ell-|d|}(u_i+1))}$ for indices $j_1, \ldots, j_k \in$

$[1, \ell - |d|]$ as a winning number whose exponent is less than N_{max}. Thus, the player only needs to verify the correctness of a single aggregated function value/proof via the efficient aggregation verification algorithm. This approach reduces not only the amount of data written to the dealer's storage space but also the computational cost for the verification process of each player. Our improved e-lottery scheme shares the same framework as Chow et al.'s scheme [7], with some modifications in the winning result generation phase and the player verification phase. The concrete description of our improved e-lottery scheme is as follows.

Let $H : \{0,1\}^* \to \{0,1\}^{\ell_H}$ be a collision-resistant hash function and $D : \{0,1\}^{\ell_H} \to \{0,1\}^{\ell_D}$ a verifiable delay function (VDF). Please refer to Appendix A for a detailed description of the VDF that we use. Let VRF be a function with input space $\{0,1\}^{\ell}$, where $\ell > \ell_D$, the function value space $\mathbb{G}_T$ and the proof space $\mathbb{G}^{\ell+1}$. Suppose any element in group $\mathbb{G}_T$ is represented as a bit-string with length ℓ_{bt}.

Setup:

Dealer side:

1. Generate a public/secret key pair of VRF as $(\mathsf{pk}_{\mathsf{VRF}}, \mathsf{sk}_{\mathsf{VRF}}) \leftarrow \mathsf{VRF.Setup}(1^\lambda)$ and key pair of a signature scheme as $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) \leftarrow \mathsf{SIG.Setup}(1^\lambda)$.
2. Choose an arbitrary integer $N_{max} \in \mathbb{Z}_p^*$. The numbers used in the lottery game are $\{1, 2, \ldots, N_{max}\}$.
3. Publish a collision-resistant hash function, public key of VRF $\mathsf{pk}_{\mathsf{VRF}}$, public key of signature scheme $\mathsf{pk}_{\mathsf{SIG}}$, the delaying function $D(\cdot)$, and the amount of time $\mathcal{T}$ in which the dealer must release the generated winning ticket value.

Ticket Purchase:

Player side:

1. The player chooses $x \in \mathbb{Z}_p^*$ as bet number and randomly samples $r \leftarrow \mathbb{Z}_p^*$. r is kept secret.
2. The player obtains a sequence number s of the ticket from the dealer.
3. Compute $H(x\|s\|r)$, and send $\mathsf{ticket}_i = s\|(x \oplus r)\|H(x\|s\|r)$ to the dealer.

Dealer side:

1. The dealer generates a signature for ticket ticket_i as $\sigma_i \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}_{\mathsf{SIG}}, \mathsf{ticket}_i)$ and returns σ_i to the player to acknowledge the recipient of player's purchase request.
2. The dealer creates the state of ticket_1 as $\mathsf{st}_1 := H(\mathsf{ticket}_1)$, and $\mathsf{st}_i := H(\mathsf{st}_{i-1}\|\mathsf{ticket}_i)$ for $i = 2, 3, \ldots$.
3. The dealer generates blocks which contain: (1) the current state $\mathsf{st}_i \in \{0,1\}^{\ell_H}$; (2) ticket ticket_i; and (3) signature σ_i for ticket ticket_i under $\mathsf{sk}_{\mathsf{SIG}}$, e.g., with the following block structure:
$$B_i = (\mathsf{st}_i, \mathsf{ticket}_i, \sigma_i) := (\mathsf{st}_i := H(\mathsf{st}_{i-1}\|\mathsf{ticket}_i), \mathsf{ticket}_i, \sigma_i).$$
4. The dealer links all blocks to a blockchain, which is a sequence of blocks $B_1, B_2, \ldots$.
5. The dealer publishes a blockchain $\mathcal{C} = B_1, B_2, \ldots, B_n$ where n is the number of tickets sold so far. The length of a chain $\mathsf{len}(\mathcal{C}) = n$ is its number of blocks. The structure of a blockchain $\mathcal{C}$ is depicted in Figure 2:

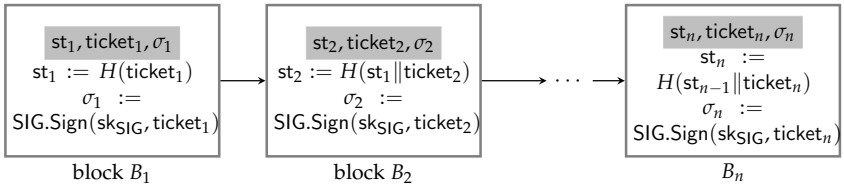


Figure 2. A blockchain $\mathcal{C}$ with n tickets sold so far.

Winning Result Generation:

Dealer side:

1. Let the final state of the blockchain $\mathcal{C}$ be st_n; the dealer computes $d := D(\mathsf{st}_n)$ by the delaying function and publishes d.
2. Pad $d \in \{0,1\}^{\ell_D}$ with $\perp$ as $\tilde{d} := \perp^{\ell-\ell_D} \| d$. Let $\tilde{d} = \perp^{\ell-\ell_D} \| d_1 \| \cdots \| d_{\ell_D}$. Define a set $S_{\tilde{d}} := \{\xi \in \{0,1\}^{\ell} : \forall i \in [\ell], \xi_i = \tilde{d}_i \text{or} \tilde{d}_i = \perp\}$.
3. The dealer calculates the productive aggregation of all $|S_{\tilde{d}}| = 2^{\ell-\ell_D}$ numbers of function values and their corresponding proofs by using the efficient aggregation algorithm as $(y^{\mathsf{agg}}, \pi^{\mathsf{agg}}) := \text{Aggregate}(\mathsf{sk}_{\mathsf{VRF}}, \tilde{d})$. More precisely, since $\mathsf{sk}_{\mathsf{VRF}} = (u_0, u_1, \ldots, u_\ell)$, which is defined by HW-VRFs scheme in Section 2.2, we have $y^{\mathsf{agg}} = e(g,h)^{u_0 \cdot (\prod_{i=\ell-\ell_D+1}^{\ell} u_i^{d_i}) \cdot (\prod_{i=1}^{\ell-\ell_D}(u_i+1))}$.
4. Let $\mathsf{EXP}(y^{\mathsf{agg}}) := u_0 \cdot (\prod_{i=\ell-\ell_D+1}^{\ell} u_i^{d_i}) \cdot (\prod_{i=1}^{\ell-\ell_D}(u_i+1))$. The dealer checks if $\mathsf{EXP}(y^{\mathsf{agg}}) (\mathrm{mod} p-1) \leq N_{max}$. If it is true, then set $y_{\mathsf{win}} := y^{\mathsf{agg}}$ as the winning result.
5. Otherwise, the dealer chooses a random index $\zeta \in [1, \ell-\ell_D]$ and then uses ζ to define a new wildcard $\tilde{d}' \leftarrow \perp^{\zeta-1} \| 0 \| \perp^{\ell-\ell_D-\zeta} \| d_1 \| \cdots \| d_{\ell_D}$. The dealer computes $\mathsf{EXP}(\tilde{d}') := u_0 \cdot (\prod_{i=\ell-\ell_D+1}^{\ell} u_i^{d_i}) \cdot (\prod_{i=1, i\neq\zeta}^{\ell-\ell_D}(u_i+1))$, and checks if $\mathsf{EXP}(\tilde{d}') (\mathrm{mod} p-1) \leq N_{max}$. Once finding a $\zeta \in [1, \ell-\ell_D]$ s.t. $\mathsf{EXP}(\tilde{d}') (\mathrm{mod} p-1) \leq N_{max}$, the dealer sets $y_{\mathsf{win}} := e(g,h)^{\mathsf{EXP}(\tilde{d}')}$ as the winning result and computes the corresponding proof by using the efficient aggregation algorithm $\text{Aggregate}(\mathsf{sk}_{\mathsf{VRF}}, \tilde{d}')$.
6. The dealer publishes the winning result and its proof $(y_{\mathsf{win}}, \pi_{\mathsf{win}})$ together with corresponding $\tilde{d}'$ within Δ units of time after the closing of the lottery session.

Prize Claiming:

1. The player checks if $e(g,h)^x = y_{\mathsf{win}}$. If it is true, the player wins.
2. The player submits (s,r) to the dealer.
3. The dealer checks whether there exists a ticket ticket_i in the blockchain $\mathcal{C}$ such that $\mathsf{ticket}_i = s \| (x \oplus r) \| H(x \| s \| r)$.
4. If it is true, the dealer checks whether the tuple (s,r) has already been published (i.e., the prize has been claimed by someone already).
5. If the prize is not yet claimed, the dealer pays the player and publishes (s,r).

Player Verification:
Player side:

1. The player checks whether his/her ticket(s) is/are included in the blockchain $\mathcal{C}$ and checks whether the final state st_n of the blockchain $\mathcal{C}$ is correct.
2. The player verifies the correctness of d by using the verification algorithm of VDF.
3. The player parses $\tilde{d}' = \perp^{\zeta-1} \| 0 \| \perp^{\ell-\ell_D-\zeta} \| d_1 \| \cdots \| d_{\ell_D}$ and checks if $d = d_1 \| \cdots \| d_{\ell_D}$.
4. The player verifies the correctness of y_{win} by using the verification algorithm $b \leftarrow \text{AggVerify}(\mathsf{pk}_{\mathsf{VRF}}, \tilde{d}', y_{\mathsf{win}}, \pi_{\mathsf{win}})$.
5. For each winning ticket published, the players verify the validity of $s \| (x \oplus r) \| H(x \| s \| r)$.

4.3. Implementation and Comparison on Chow et al.'s/Improved E-Lottery

To implement Chow et al.'s e-lottery scheme, we use the instantiation of HW-VRF presented in Section 2.2, while using our aggregate VRF scheme in Section 3.1 to implement the improved counterpart. When implementing the winning result generation phase and the player verification phase, we reuse part of the code for our implementation in Section 3.3. For the required delay functions, we use an instantiation of VDFs proposed by Pietrzak [29] (see Appendix A for details), which is defined as $D(x) := x^{2^{\mathcal{T}}} (\mathrm{mod} N)$, where $N = p_1 q_1$ is a product of two large, distinct primes p_1 and q_1, x is a random element in $\mathbb{Z}_N^*$, and $\mathcal{T}$ is the timing parameter. For the signature scheme, we used EdDSA [30] provided in the libsodium (https://libsodium.gitbook.io/doc/) library.

Parameters. In our concept implementation, we consider Pietrzak's VDF scheme [29] with the following parameters, where the statistical soundness security parameter of a proof is $\lambda_{\mathsf{Sound}} = 100$,

the time parameter is $\mathcal{T} = 2^{40}$, the bit-lengths of two safe primes p_1 and q_1 are $\lambda_{\mathsf{QR}}/2 = 256$ bit safe primes p_1, q_1, and the bit-length of an RSA modulus $N = p_1 q_1$ is $\lambda_{\mathsf{RSA}} = 512$. (Here, we set $\lambda_{\mathsf{RSA}} = 512$ as a toy example of implementation, which can be easily scaled to 1024 or 2048, if we adjust the corresponding parameters of the hash function as well as the inputs/outputs spaces of the VRFs.) Then, the VDF defines a function $D : \{0,1\}^{256} \to \{0,1\}^{512}$. We use SHA-256 as a hash function $H : \{0,1\}^* \to \{0,1\}^{256}$. The inputs/outputs of the VRFs are set equal to 1024 binary strings. We randomly choose an integer $N_{max} \in \mathbb{Z}_p$ s.t. $1 \leq N_{max} < p-1$, and the numbers used in the lottery game are $\{1, 2, \ldots, N_{max}\}$. For every outcome $w \in \mathbb{G}_T$ of VRFs evaluation, to check if w can be set as a winning number, the dealer actually checks if the discrete log of w under the base generator $e(g,h)$ lies in the set $\{1, 2, \ldots, N_{max}\}$. In fact, as explained in the implementation of HW-VRF and the aggregate VRF in Section 3.3, to obtain the function value $\mathsf{Fun}_{sk}(x) = e(g,h)^{u_0 \Pi_{i=1}^{\ell} u_i^{x_i}}$, the exponent $u_0 \Pi_{i=1}^{\ell} u_i^{x_i}$ is computed beforehand, which makes it handy to decide if w can be set as a winning number or not.

Results. Table 4 shows the implementation results for both Chow et al.'s and our improved e-lottery schemes, particularly regarding the costing time for the winning number generation and the player verification. We employ a small blockchain scenario, where we have 1000 blocks, that is, there are 1000 random tickets in the system. The running time in Table 4 is for the experimental scenario that in Chow et al.'s e-lottery scheme the VRF is applied only twice and a function value is obtained, whose exponent $\mathsf{EXP}(w_1) (\mathrm{mod} p-1) \leq N_{\mathsf{max}}$, while in our scheme the exponent of y^{agg} is beyond N_{max}, i.e., $\mathsf{EXP}(y^{\mathsf{agg}}) (\mathrm{mod} p-1) > N_{max}$, so that extra time is taken to search for another y'^{agg} whose exponent is below N_{max}. We consider such a case, because it optimally shows the efficiency gain of our e-lottery scheme over Chow et al.'s, since in this case the fewest iterative applications of VRFs are involved in Chow et al.'s scheme, while the extra step of recomputing y'^{agg} occurs in ours. The experimental results show that our scheme takes less time to generate the winning ticket than Chow et al.'s, and the player verification phase takes almost half less time as that of Chow et al.'s.

Table 4. Running time (seconds) of the e-lottery schemes.

Lottery Scheme	Winning Number Generation	Player Verification
Scheme in [7]	96.12992 s	4.384418 s
Our Scheme	90.13322 s	2.782610 s

5. Conclusions

Inspired by the idea of aggregated PRFs [10], in this paper, we investigate how we may efficiently aggregate VRF values and their corresponding proofs. We introduce the notion of static aggregate VRFs. We show how to achieve static aggregate VRFs under the Hohenberger and Waters' VRF scheme [13] for the product aggregation with respect to a bit-fixing set, based on the q-decisional Diffie–Hellman exponent assumption. Furthermore, we apply our aggregate VRFs to improve the prior VRF-based e-lottery scheme in the efficiency of generating the winner number and player verification phases. We test the performances of our static aggregate VRFs and improved e-lottery scheme, which show significant computation advantage and efficiency gains compared to the original counterparts.

As future work, it would be interesting to explore if it is possible to realize static aggregate VRFs for much more expressive sets, such as sets that can be recognized by polynomial-size decision trees, read-once Boolean formulas, or polynomial-size boolean circuits. Furthermore, it would be interesting to consider a public dynamic aggregate VRF, which allows taking any two fresh (aggregate) function values and proofs and combine them into a new aggregate function value and proof, without requiring the exponential number of inputs (from an exponentially large set) to be known in advance, as required in static aggregation.

Author Contributions: Conceptualization, B.L. and A.M.; methodology, B.L.; implementation, G.B.; formal analysis, B.L.; writing—original draft preparation, B.L. and G.B.; writing—review and editing, A.M.; supervision, A.M.; project administration, B.L.; and funding acquisition, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the GENIE Chalmers CryptoQuaC project, the VR PRECIS project, the WASP expedition project "Massive, Secure, and Low-Latency Connectivity for IoT Applications", and the National Nature Science Foundation of China (No. 61972124).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PRFs	Pseudorandom functions
VRFs	Verifiable random functions
Agg-VRFs	Aggregate verifiable random functions
DDHE	Decisional Diffie–Hellman Exponent

Appendix A. Verifiable Delay Functions (VDFs)

Here, we give an instantiation of VDFs proposed by Pietrzak [29], which is used in our improved e-lottery scheme.

Given as input an RSA group $\mathbb{Z}_N^*$, where $N = pq$ is a product of two large, distinct primes p and q, a random element $x \in \mathbb{Z}_N^*$, and a timing parameter T, compute $D(x) = x^{2^T} (\mathrm{mod} N)$. As we know, anyone with the knowledge of $\phi(N) = (p-1)(q-1)$ can efficiently compute $D(x)$ with two exponentiations, by first computing $e = 2^T (\mathrm{mod} \phi(N))$, followed by x^e. Anyone who does not know $\phi(N)$ (or, equivalently, the factorization of N), in order to compute $D(x)$, is required to perform T sequential squarings in the group $\mathbb{Z}_N^*$ even on a parallel computer with polynomial numbers of processors. Without the group order of $\mathbb{Z}_N^*$ (or, equivalently, the factorization of N), the computation of $D(x)$ requires T sequential squarings in the group.

To convince any verifier that the published value y is correct, namely that $y = x^{2^T} (\mathrm{mod} N)$, Pietrzak [29] provided a succinct public-coin interactive argument for the language

$$L := \{(\mathbb{Z}_N^*, x, y, T) : y = x^{2^T} \in \mathbb{Z}_N^*\}. \tag{A1}$$

The verifier and prover do so as follows.

1. $\mathcal{V}$ sends to $\mathcal{P}$ a random r in $\mathbb{Z}_{2^\lambda}$.
2. Both $\mathcal{P}$ and $\mathcal{V}$ compute $x_1 \leftarrow x^r \cdot \mu$ and $y_1 \leftarrow \mu^r \cdot y$.
3. $\mathcal{P}$ and $\mathcal{V}$ recursively engage in an interactive proof for statement $(\mathbb{Z}_N^*, x_1, y_1, T/2) \in L$, namely that $y_1 = x_1^{2^{\frac{T}{2}}} \in \mathbb{Z}_N^*$.

This subprotocol is repeated $\log_2 T$ times, each time halving the time parameter T until $T = 1$, at which point $\mathcal{V}$ can efficiently verify correctness of the claim itself.

References

1. Micali, S.; Rabin, M.; Vadhan, S. Verifiable random functions. In Proceedings of the 40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039), New York, NY, USA, 17–19 October 1999; pp. 120–130.
2. Naor, M.; Pinkas, B.; Reingold, O. Distributed Pseudo-Random Functions and KDCs. In *EUROCRYPT 1999*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 327–346.
3. Micali, S.; Rivest, R.L. *Micropayments Revisited*; CT-RSA 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 149–163.
4. Papadopoulos, D.; Wessels, D.; Huque, S.; Naor, M.; Včelák, J.; Reyzin, L.; Goldberg, S. *Making NSEC5 Practical for DNSSEC*; Cryptology ePrint Archive, Report 2017/099; IACR 2017. Available online: https://eprint.iacr.org/2017/099 (accessed on 8 February 2017).
5. Goldberg, S.; Naor, M.; Papadopoulos, D.; Reyzin, L.; Vasant, S.; Ziv, A. *NSEC5: Provably Preventing DNSSEC Zone Enumeration*; NDSS: New York, NY, USA, 2015.

6. Papadopoulos, D.; Wessels, D.; Huque, S.; Naor, M.; Vcelák, J.; Reyzin, L.; Goldberg, S. Can NSEC5 be practical for DNSSEC deployments? *IACR Cryptol. Eprint Arch.* **2017**, *2017*, 99.
7. Chow, S.S.M.; Hui, L.C.K.; Yiu, S.M.; Chow, K.P. An e-Lottery Scheme Using Verifiable Random Function. In *ICCSA 2005*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 651–660.
8. David, B.; Gaži, P.; Kiayias, A.; Russell, A. Ouroboros Praos: An Adaptively-Secure, Semi-Synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 66–98.
9. Badertscher, C.; Gazi, P.; Kiayias, A.; Russell, A.; Zikas, V. *Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability*; Technical Report; Cryptology ePrint Archive, Report 2018/378; IACR 2018. Available online: https://eprint.iacr.org/2018/378.pdf (accessed on 25 April 2018).
10. Cohen, A.; Goldwasser, S.; Vaikuntanathan, V. Aggregate Pseudorandom Functions and Connections to Learning. In *TCC 2015*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 61–89.
11. Boneh, D.; Waters, B. Constrained Pseudorandom Functions and Their Applications. In *ASIACRYPT 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 280–300.
12. Boneh, D.; Lewi, K.; Montgomery, H.; Raghunathan, A. Key homomorphic PRFs and their applications. In *CRYPTO 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 410–428.
13. Hohenberger, S.; Waters, B. Constructing verifiable random functions with large input spaces. In *EUROCRYPT 2010*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 656–672.
14. Jager, T.; Niehues, D. On the Real-World Instantiability of Admissible Hash Functions and Efficient Verifiable Random Functions. In *SAC 2019*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 303–332.
15. Lysyanskaya, A. Unique Signatures and Verifiable Random Functions from the DH-DDH Separation. In *CRYPTO 2002*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 597–612.
16. Dodis, Y. Efficient Construction of (Distributed) Verifiable Random Functions. In *PKC 2003*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 1–17.
17. Dodis, Y.; Yampolskiy, A. A Verifiable Random Function with Short Proofs and Keys. In *PKC 2005*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 416–431.
18. Jager, T. Verifiable Random Functions from Weaker Assumptions. In *TCC 2015*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 121–143.
19. Hofheinz, D.; Jager, T. Verifiable Random Functions from Standard Assumptions. In *TCC 2016*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 336–362.
20. Yamada, S. Asymptotically Compact Adaptively Secure Lattice IBEs and Verifiable Random Functions via Generalized Partitioning Techniques. In *CRYPTO 2017*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 161–193.
21. Katsumata, S. On the Untapped Potential of Encoding Predicates by Arithmetic Circuits and Their Applications. In *ASIACRYPT 2017*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 95–125.
22. Kohl, L. Hunting and Gathering—Verifiable Random Functions from Standard Assumptions with Short Proofs. In *PKC 2019*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 408–437.
23. Liu, Y.; Hu, L.; Liu, H. Using an efficient hash chain and delaying function to improve an e-lottery scheme. *Int. J. Comput. Math.* **2007**, *84*, 967–970. [CrossRef]
24. Lee, J.S.; Chang, C.C.; Fellow, IEEE. Design of electronic *t*-out-of-*n* lotteries on the Internet. *Comput. Stand. Interfaces* **2009**, *31*, 395–400. [CrossRef]
25. Chen, C.L.; Chiang, M.L.; Lin, W.C.; Li, D.K. A novel lottery protocol for mobile environments. *Comput. Electr. Eng.* **2016**, *49*, 146–160. [CrossRef]
26. Grumbach, S.; Riemann, R. Distributed Random Process for a Large-Scale Peer-to-Peer Lottery. In *Distributed Applications and Interoperable Systems*; Chen, L.Y., Reiser, H.P., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; pp. 34–48.
27. Naor, M.; Reingold, O. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM (JACM)* **2004**, *51*, 231–262. [CrossRef]
28. Lynn, B. On the Implementation of Pairing-Based Cryptosystems. Ph.D. Thesis, Stanford University Stanford, Stanford, CA, USA, 2007.
29. Pietrzak, K. Simple Verifiable Delay Functions. In *Leibniz International Proceedings in Informatics (LIPIcs)*; ITCS 2019; Blum, A., Ed.; Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2018; Volume 124, pp. 60:1–60:15. [CrossRef]

30. Bernstein, D.J.; Duif, N.; Lange, T.; Schwabe, P.; Yang, B.Y. High-speed high-security signatures. *J. Cryptogr. Eng.* **2012**, *2*, 77–89. [CrossRef]