Hunting and Gathering – Verifiable Random Functions from Standard Assumptions with Short Proofs

Lisa Kohl *

Karlsruhe Institute of Technology, Karlsruhe, Germany
Lisa.Kohl@kit.edu

Abstract. A *verifiable random function (VRF)* is a pseudorandom function, where outputs can be *publicly verified*. That is, given an output value together with a proof, one can check that the function was indeed correctly evaluated on the corresponding input. At the same time, the output of the function is computationally indistinguishable from random for all non-queried inputs.

We present the first construction of a VRF which meets the following properties at once: It supports an exponential-sized input space, it achieves full adaptive security based on a non-interactive constant-size assumption and its proofs consist of only a logarithmic number of group elements for inputs of arbitrary polynomial length.

Our construction can be instantiated in symmetric bilinear groups with security based on the decision linear assumption. We build on the work of Hofheinz and Jager (TCC 2016), who were the first to construct a verifiable random function with security based on a non-interactive constant-size assumption. Basically, their VRF is a matrix product in the exponent, where each matrix is chosen according to one bit of the input. In order to allow verification given a symmetric bilinear map, a proof consists of all intermediary results. This entails a proof size of $\Omega(L)$ group elements, where L is the bit-length of the input.

Our key technique, which we call *hunting and gathering*, allows us to break this barrier by rearranging the function, which – combined with the partitioning techniques of Bitansky (TCC 2017) – results in a proof size of ℓ group elements for arbitrary $\ell \in \omega(1)$.

1 Introduction

A *pseudorandom function* is, roughly speaking, a function that can be efficiently evaluated if provided a key, but - given only black-box access - is computationally indistinguishable from a truly random function. Since introduced by Goldreich, Goldwasser and Micali [16] in 1986, pseudorandom functions have been proven

*Supported by ERC Project PREP-CRYPTO (724307), by DFG grant HO 4534/2-2 and by a DAAD scholarship. This work was done in part while visiting the FACT Center at IDC Herzliya, Israel.

useful in many applications. Nevertheless, their security guarantee is somewhat one-sided, as by definition a receiver not knowing the key cannot be sure that indeed he obtained the output value of the pseudorandom function.

To open doors to a wider range of applications, in 1999 Micali, Rabin and Vadhan [28] introduced the concept of *verifiable random functions*. A verifiable random function is a pseudorandom function, for which the key holder – in addition to the image – provides a proof of correct evaluation. The security requirement is *unique provability*, i.e. for each preimage there exists a valid proof for at most one function value. Verifiable random functions are applicable in many contexts, like [29, 30, 24, 26, 4, 6].

In order to achieve unique provability, the key holder has to publish a *verification key*, which can be viewed as a commitment to the function, such that *even a maliciously chosen verification key* commits to at most one function.

At first glance, employing a pseudorandom function together with a zero-knowledge proof seems promising. But, aiming for a non-interactive construction, the following problem arises. As proven in [17], non-interactive zero knowledge proofs require a common reference string. Letting a possibly malicious key holder choose this common reference string compromises soundness.

Recent generic constructions [18, 7, 5] choose a similar approach and build verifiable random functions based on non-interactive witness-indistinguishable proofs, which by [19] exist without trusted set-up based on standard assumptions. All of them, though, lack efficient instantiations.

At the birth of verifiable random functions, [28] took a different path. Namely, theirs and the following constructions build on functions which have an *implicit* and an *explicit* representation. While the implicit representation serves as the commitment to the function which can be published without compromising pseudorandomness, the explicit representation allows efficient evaluation. For instance, given a function with image in $\mathbb{Z}_p$, one can think of the function values in the exponent of some suitable group as an implicit representation, from which the function cannot be computed efficiently, but which commits the key holder to exactly one function.

A number of constructions [27, 11, 12, 1] followed this line of work, but up until the work of Hohenberger and Waters in 2010 [21] all of them come with some limitation: Either they are based on an interactive assumption, or they only allow polynomial-sized input space, or they do not achieve full adaptive security. In the following we only consider verifiable random functions which suffer from none of those limitations (so-called verifiable random functions *with all desired properties*). While there are many constructions of VRFs with all desired properties (see Figure 1), they all come at a cost: Either they have to rely on a non-constant-size assumption (i.e. an assumption depending on the security parameter and/or on some quantity of the adversary), or require large proofs. Thus, the following question was left open:

Open question. Do verifiable random functions with all desired properties exist such that further

Reference	$\lvert vk\rvert$	$\lvert\pi\rvert$	Assumption	Security loss
[HW10] [21]	$\mathcal{O}(L)$	$\mathcal{O}(L)$	$\mathcal{O}(L \cdot Q)$ - DDHE	$L \cdot Q/\varepsilon$
[BMR10] [9]	$\mathcal{O}(L)$	$\mathcal{O}(L)$	$\mathcal{O}(L)$ - DDH	L
[Jag15] [23]	$\mathcal{O}(L)$	$\mathcal{O}(L)$	$\mathcal{O}(\log(Q/\varepsilon))$ - DDH	$Q^\gamma/\varepsilon^{\gamma+1}$
[HJ16] [20]	$\mathcal{O}(L)$	$\mathcal{O}(L)$	DLIN	$L \log\lambda \cdot Q^{2/\mu}/\varepsilon^3$
[Ros17] [33]	$\mathcal{O}(L)$	$\mathcal{O}(L)$	DLIN	$L \log\lambda \cdot Q^{2/\mu}/\varepsilon^3$
[Yam17] [35]	$\omega(L \log\lambda)$	$\omega(\log\lambda)$	$\omega(L \log\lambda)$ - DDH	$Q^\gamma/\varepsilon^{\gamma+1}$
[Kat17] [25]	$\omega(\sqrt{L} \log\lambda)$	$\omega(\log\lambda)$	$\omega(\log^2\lambda)$ - DDH	$Q^\gamma/\varepsilon^{\gamma+1}$
This work	$\omega(L \log\lambda)$	$\omega(\log\lambda)$	DLIN	$\lvert\pi\rvert \log\lambda \cdot Q^{2/\mu}/\varepsilon^3$
	$\omega(L^{2+2\eta})$	$\omega(1)$	DLIN	$\lvert\pi\rvert \log\lambda \cdot Q^{2+2/\eta}/\varepsilon^3$

Fig. 1: Comparison with previous efficient constructions of VRFs with all desired properties. The **second and third column** give an overview of the sizes of the verification key and the proof *in number of group elements*, respectively. Throughout, by λ we denote the security parameter, by L the input length (a canonical choice is $L = \lambda$) and by $0 < \eta \leq 1$ an arbitrary constant (representing a trade-off between size of the verification key and security loss). In the **fourth column** we provide the underlying assumption. DDHE refers to the decisional Diffie-Hellman exponent assumption (see [21]), and DDH and DLIN to the decisional Diffie-Hellman and the decision linear assumption respectively. In the **last column** we give an overview of the security loss (in $\mathcal{O}$-notation). Here, $\varepsilon \geq 1/\mathsf{poly}(\lambda)$ and $Q \leq \mathsf{poly}(\lambda)$ refer to the advantage and the number of evaluation queries of the adversary, respectively. The constructions [23, 20, 33, 35, 25] and our first construction require an error correcting code $\{0,1\}^L \to \{0,1\}^n$ with minimal distance $n\mu$. For the security loss of [23, 35, 25] we refer to [35], Table 2 (in the eprint version) and [25], Table 1 (in the eprint version). There, γ is such that $\mu = 1 - 2^{-1/\gamma}$. Note that γ can be chosen as close to 1 as desired by choosing $\mu < 1/2$ and n large enough (see [15], Appendix E.1). For [20, 33] and our first construction, the admissible hash function is instantiated with [27], where μ is constant and $n \in \mathcal{O}(L)$. Note that the security loss stems from using the *artifical abort technique* by [34]. For our second construction we use the admissible hash function of [7]. We only give the most efficient instantiation of [35] and [25] regarding the proof size, as this is the focus of our work. As the generic constructions [18, 7, 5] do not come with efficient instantiations, they are omitted in the overview.

(A) the VRF security can be based on a standard *constant-size* assumption AND

(B) the proof consists of $o(L)$ group elements, where L is the bit-length of the input?

While previously only either part on its own was tackled ((A) by Hofheinz and Jager [20] and (B) by Yamada [35] and Katsumata [25]), our work answers this question affirmatively. Further, to our knowledge our construction constitutes the only verifiable random function with all desired properties that requires only ℓ group elements in the proof, where $\ell \in \omega(1)$ arbitrary. We achieve this at the price of larger verification keys and a larger security loss in the reduction to the underlying assumption.

Additionally, we give an instantiation which achieves the same efficiency (regarding the size of the verification key and proofs) as the recent work of Yamada [35], but based on a constant-size assumption.

We leave it as an open question to construct verifiable random functions with all desired properties that have short proofs and a short verification key. It is worth mentioning that, when allowing the input space to be polynomial-sized, even constructions with constant key and proof size (in number of group elements) exist [12].

Why constructing verifiable random functions is difficult. The main source of difficulty in constructing adaptively secure verifiable random functions is the requirement of unique provability. To see this, consider the reduction of pseudorandomness of the VRF to the underlying assumption. While the reduction has to answer all evaluation queries x with the *unique* function value y together with a corresponding proof, the reduction hopes to solve the underlying problem with the function value $y^\star$ corresponding to the challenge value $x^\star$. This imposes the following challenges in addressing the stated open question.

(A) In order to explain why many previous approaches rely on *non-constant-size* assumptions, we take [27] as a starting point. The core of the construction is simply the Naor-Reingold pseudorandom function [31], which, given a secret key $(a_{i,j})_{i\in\{1,\ldots,L\},j\in\{0,1\}}$ of randomly chosen exponents and an input $x \in \{0,1\}^L$, evaluates to

$$g^{\prod_{i=1}^{L} a_{i,x_i}}.$$

To reduce the security of the pseudorandom function to the decisional Diffie-Hellman assumption, Naor and Reingold [31] employ a hybrid argument, where they gradually randomize images given to the adversary. Using the same proof technique in order to prove VRF security, the reduction would have to provide proofs for random values. Recall that by unique provability even for a verification key which is set up maliciously there exists at most one function value for every preimage which has a validating proof. The reduction has thus no possibility to simulate proofs. As the result of employing a different proof strategy, the security of the verifiable random function has to be based on a non-constant-size computational assumption depending on the input length L of the VRF. Namely, given oracle access to $g^{\prod_{i\in S'} z_i}$ for every proper subset $S' \subsetneq S := \{z_1, \ldots, z_L\}$, it is assumed to be difficult to compute $g^{\prod_{i\in S} z_i}$.

As non-constant-size assumptions become stronger with increasing input length (or even worse depend on the number of adversarial queries [21]), basing the VRF security on constant-size assumptions is desirable.

The only work overcoming the described difficulty and achieving security based on a constant-size assumption so far is [20], who use a more complex underlying function, allowing them to again employ a hybrid argument in the proof of security. Their work will be the starting point of our construction.

(B) As the adversary is allowed to choose the evaluation and the challenge query *adaptively*, the reduction has to partition the input space ahead of time, such

that with noticeable probability it embeds the underlying problem in the challenge query $x^\star$, but is at the same time able to answer all evaluation queries $x^{(1)}, \ldots, x^{(Q)}$.

A common strategy to achieve this is via admissible hash functions [27, 8, 10, 3, 14, 23, 20] for partitioning. A function (or encoding) $\{0,1\}^L \to \Sigma^n$ (for some $n \in \mathbb{N}$ and alphabet Σ) is called *admissible hash function*, if there exists an efficient sampling algorithm returning a word $Y \in \Sigma^n$ and a subset of indices $I \subseteq \{1, \ldots, n\}$, such that for any choice of $x^{(1)}, \ldots, x^{(Q)}$ and $x^\star \notin \{x^{(1)}, \ldots, x^{(Q)}\}$ with noticeable probability we have

$$x^\star \in \mathcal{Y} := \{x \in \{0,1\}^L \mid \forall i \in I \colon \mathsf{AHF}(x)_i = Y_i\}$$

and

$$\{x^{(1)}, \ldots, x^{(Q)}\} \subseteq \mathcal{Z} := \{0,1\}^L \backslash \mathcal{Y}.$$

Therefore, the reduction can embed the underlying problem into all elements of $\mathcal{Y}$, while being able to answer all evaluation queries in $\mathcal{Z}$. The choice of the encoding is crucial, as an adversary may try to minimize the probability of successful partitioning by maliciously choosing the queries. The most efficient instantiation [27, 14] achieve $n = \Theta(L)$ for $\Sigma = \{0,1\}$ by employing a suitable error correcting code with sufficiently large minimal distance.

In most constructions employing admissible hash functions, the reduction embeds Y in some way into the public parameters, leading to parameter sizes of at least $n \cdot |\Sigma|$ ([8, 10, 9, 2, 23, 20]). In constructions of verifiable random functions a larger verification key often affects the proof size. This is due to the fact that the proof typically consists of intermediate results in order to make the output value verifiable (e.g. by employing a bilinear map). For this reason most previous constructions inherently require proofs to consist of $\Omega(L)$ group elements.

Strategies to achieve short proofs. Recall that an admissible hash function $\mathsf{AHF} : \{0,1\}^L \to \Sigma^n$ partitions a space into $\mathcal{Y} := \{x \in \{0,1\}^L \mid \forall i \in I \colon \mathsf{AHF}(x)_i = Y_i\}$ and $\mathcal{Z} := \{0,1\}^L \setminus \mathcal{Y}$. Note that the relevant information in Y (which has to be embedded into the public parameters in some way) only consists of $|I|$ bits (which is typically logarithmic in Q, where Q is the number of evaluation queries). Yamada [35] achieves shorter proofs by encoding the relevant information of Y into a bitstring consisting of only $\omega(\log Q)$ components and employing an admissible hash function based on the shorter bitstring. Katsumata [25] follows a similar approach and can be viewed as a combination of [35] and [23] to achieve security based on a weaker assumption.

While we build on the same observation, we follow a different strategy. Namely, we remove the dependency of the proof size on n and $|\Sigma|$ by rearranging the underlying pseudorandom function. As a result, our proof size only depends on the number of chosen indices $|I|$.

The instantiation of admissible hash functions by Lysyanskaya [27] (so-called *substring matching*), which is also employed in [20], yields $|I| \in \mathcal{O}(\log Q)$ (=

$\mathcal{O}(\log \lambda)$ for $Q \in \mathsf{poly}(\lambda)$). This results in a proof size of $\omega(\log \lambda)$ and a verification key size of $\omega(\lambda \log \lambda)$ (in number of group elements).

We observe that we can reduce the proof size even further to $\omega(1)$ group elements, by using the admissible hash fuction of Bitansky [7] (so-called *substring matching over polynomial alphabet*). This entails verification keys of $\omega(\lambda^{2+2\eta})$ group elements (where $0 < \eta \leq 1$ is an arbitrary constant influencing the security loss). The reason for the larger verification is that the underlying encoding function has to satisfy stronger properties in order to achieve $|I| \in \omega(1)$ and comes thus with larger parameters n and $|\Sigma|$.

Note that the efficiency gain in our approach (and similar in [35, 25]) crucially relies on the restriction to adversaries that ask a *polynomially bounded* number of evaluation queries Q. One could thus consider the construction of [20] with input size $L = \omega(\log \lambda)$ (where λ is the security parameter), thereby obtaining a verifiable random function with proofs consisting of $o(\lambda)$ group elements. We want to emphasize that in this approach, proofs still consist of $\Omega(L)$ group elements and are thus linear regarding the input size. We, on the other hand, achieve proof size $o(\lambda)$ in number of group elements *independent of the input size* (assuming the length of the input to be polynomially bounded).

Restricting to polynomial size adversaries, one could also achieve proofs of size $o(L)$ (in number of group elements) by evaluating the VRF on $H(x)$ for a collision resistant hash function $H\colon \{0,1\}^L \to \{0,1\}^{\omega(\log \lambda)}$. This approach, however, requires an exponential assumption, whereas we obtain short proofs solely relying on the decision linear assumption.

1.1 Technical Overview.

In the following we want to give an overview on how we achieve proofs consisting of $\omega(1)$ group elements. Roughly, our strategy is to rearrange the function from [20]. Recall that the raw-VRF of [20] is a matrix product in the exponent, where each factor depends on one bit of a suitable encoding of the input. Instead, we will have each factor depend on *all* bits of the encoding, and only take a product over $|I|$ factors, where I is the index set stemming from partitioning via an admissible hash function AHF. For $|I| \in \omega(1)$, we employ the instantiation of admissible hash functions by Bitansky [7].

It is worth noting that instantiating [20] with the admissible hash function of [7] would, on the contrary, yield larger proofs of size $\omega(L)$, as the technique of Bitansky requires $n \in \omega(L)$ for the output dimension of the encoding, which determines the proof size of [20].

We start by presenting the concept of verifiable vector hash functions (VVHF) introduced in [20], which can be seen as a pre-step of a VRF. Next, we give an overview of partitioning via admissible hash functions. This technique is employed by [20] and this work to construct an *adaptively programmable* VVHF, which in turn yields an *adaptively* secure VRF (via a generic transformation of [20]). As we build on the techniques of [20], we start with an overview of their approach, before presenting our construction.

Verifiable vector hash functions. Let $\mathbb{G}$ be a group of prime-order p with generator g. For a vector $\mathbf{v} = (v_1, v_2, v_3) \in \mathbb{Z}_p^3$ we employ the notation of [13] and write $g^{\mathbf{v}}$ to denote $(g^{v_1}, g^{v_2}, g^{v_3}) \in \mathbb{G}^3$ (and accordingly for matrices). A VVHF takes an evaluation key ek and an input x and outputs a vector $g^{\mathbf{v}} \in \mathbb{G}^3$ and a proof π (which can be verified given a verification key vk), such that the following holds:

- *Unique provability:* As for VRFs, there exists *at most one* image vector $g^{\mathbf{v}} \in \mathbb{G}^3$, for which a valid proof π exists (even for maliciously chosen verification keys).

Instead of pseudorandomness, the security property we require from a VVHF is *adaptive programmability (AP)*. That is, given a basis $\{g^{\mathbf{b}_1}, g^{\mathbf{b}_2}, g^{\mathbf{b}_3}\}$ of $\mathbb{G}^3$, there exists an alternative way of generating a verification key vk together with a trapdoor td (in the following referred to as *trapdoor key generation*), which allows evaluating the VVHF "in the given basis", i.e. given a trapdoor td and an input x, one can efficiently generate coefficients $c_1^x, c_2^x, c_3^x \in \mathbb{Z}_p^3$ together with a proof π, such that π is a valid proof for the output $g^{\mathbf{v}} := g^{\sum_{i=1}^3 c_i^x \mathbf{b}_i}$. Further, we require the following:

- *Indistinguishability:* The *trapdoor* verification keys are indistinguishable from *real* verification keys.
- *Well-distributed outputs:* With noticeable probability (that is with probability at least $1/\mathsf{poly}(\lambda)$), for any polynomially bounded number of input values $x^{(1)}, \ldots, x^{(Q)}$ and any designated input $x^\star$ with $x^\star \notin \{x^{(1)}, \ldots, x^{(Q)}\}$, we have $c_3^{x^{(\nu)}} = 0$ for all $\nu \in \{1, \ldots, Q\}$ and $c_3^{x^\star} \neq 0$ (where c_3^x is the third coefficient of the trapdoor evaluation on input x and trapdoor td). In other words, with noticeable probability the image vectors of all input values *except the designated one* lie in the 2-dimensional subspace of $\mathbb{G}^3$ generated by $g^{\mathbf{b}_1}$ and $g^{\mathbf{b}_2}$.

As shown in [20], this property together with the decision linear assumption in $\mathbb{G}$ suffices to construct verifiable random functions. The idea is to embed a part of the challenge of the decision linear assumption in $g^{\mathbf{b}_3}$.

Partitioning via admissible hash functions. In order to achieve well-distributed outputs one has to partition the preimage space into a set $\mathcal{Y}$ and a set $\mathcal{Z} := \{0,1\}^L \backslash \mathcal{Y}$, such that for any polynomially bounded number of input values $x^{(1)}, \ldots, x^{(Q)}$ and any designated input $x^\star$ with $x^\star \notin \{x^{(1)}, \ldots, x^{(Q)}\}$, we have $x^\star \in \mathcal{Y}$ and $x^{(\nu)} \in \mathcal{Z}$ for all $\nu \in \{1, \ldots, Q\}$ with noticeable probability. Then, one can set up the trapdoor key generation algorithm such that for all $x \in \mathcal{Z}$ it holds $c_3^x = 0$, and for all $x \in \mathcal{Y}$ it holds $c_3^x \neq 0$ (where c_3^x is the third coefficient of the trapdoor evaluation on input x).

Recall that admissible hash functions partition the space by employing a suitable encoding $\mathsf{AHF}\colon \{0,1\}^L \to \Sigma^n$ for some polynomial-sized alphabet Σ and $n \in \mathbb{N}$. To choose a partitioning, a subset of the indices $I \subseteq \{1, \ldots, n\}$ of suitable size and a word $Y \leftarrow_R \Sigma^n$ are drawn at random. The partitioning is chosen as

$$\mathcal{Y} := \{x \in \{0,1\}^L \mid \forall i \in I\colon \mathsf{AHF}(x)_i = Y_i\}.$$

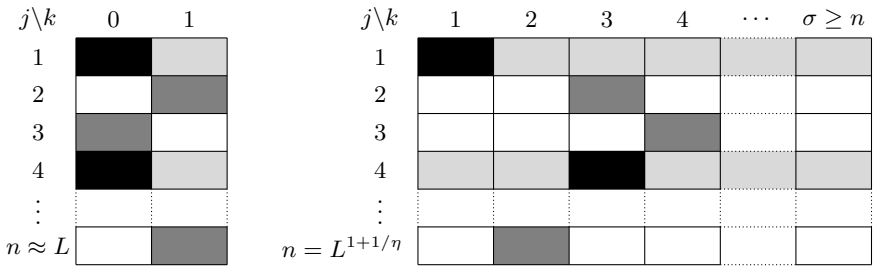

Fig. 2: A graphic representation of *partitioning via substring matching* and *partitioning via substring matching over a polynomial alphabet* $\Sigma = \{1, \ldots, \sigma\}$. In both cases we chose $I = \{1, 4\}$ (marked in light gray) and further $Y = (0, 0, 1, 0, \ldots, 1) \in \{0,1\}^n$ and $Y' = (1, 3, 4, 3, \ldots, 2) \in \Sigma^n$ (marked in gray and black). A word x lies in the set $\mathcal{Y}$ if and only if $\mathsf{AHF}(x)$ agrees with Y on the entries in black (and for Y' accordingly). Note that only the information in the light gray rows is necessary for partitioning. Recall that L is the bit-length of the input and $0 < \eta \leq 1$ an arbitrary constant.

For a graphic representation we refer to Figure 2. On the left-hand side the partioning via substring matching of [27] is depicted, whereas on the right-hand side we present the partioning via substring matching over a polynomial alphabet by [7]. Note that for the probability of successful partitioning, the underlying code and the index set size $|I|$ are crucial. The instantiation [27] achieves noticeable probability employing an error correcting code with minimal distance μn (for some constant $\mu < 1$) and $|I| \in \omega(\log \lambda)$. Error correcting codes satisfying this requirement exist with $n = \Omega(L)$ and $\Sigma = \{0, 1\}$.

To get by with $|I| \in \omega(1)$, [7] requires larger minimal distance of the underlying code for the following reason. An adversary could fix $x^\star$ and then choose $x^{(1)}, \ldots, x^{(Q)}$ such that $\mathsf{AHF}(x^\star)$ and $\mathsf{AHF}(x^{(\nu)})$ are "close" (for each $\nu \in \{1, \ldots, Q\}$). The smaller $|I|$ is, the more likely it gets that one of the $\mathsf{AHF}(x(\nu))$ agrees with $\mathsf{AHF}(x^\star)$ on all indices in I, and thus the more likely it gets that partitioning is not successful. This requirement on the encoding results in $n = L^{1+1/\eta}$ and increased alphabet size of $\sigma := |\Sigma| \geq n$. Here, $0 < \eta \leq 1$ is an arbitrary constant influencing the probability of successful partitioning.

The VVHF of Hofheinz and Jager [20]. The VVHF of [20] can be seen as a multi-dimensional version of the Naor-Reingold pseudorandom function. In order to achieve adaptive programmability, input values are first encoded using a suitable admissible hash function. To set up the evaluation and verification key, the key generation algorithm draws matrices $(\mathbf{M}_{j,k})_{j \in \{1,\ldots,n\}, k \in \Sigma}$ in $\mathbb{Z}_p^{3\times 3}$ and a vector $\mathbf{u} \in \mathbb{Z}_p^3$ at random. Note that this can be viewed as choosing a matrix for every cell of the partitioning table (as depicted in Figure 2). The evaluation key is

$$ek := \big((\mathbf{M}_{j,k})_{j \in \{1,\ldots,n\}, k \in \Sigma}, \mathbf{u}\big)$$

and the corresponding verification key is defined as

$$vk := \left(\left(g^{\mathbf{M}_{j,k}} \right)_{j \in \{1,\ldots,n\}, k \in \Sigma}, g^{\mathbf{u}} \right).$$

On input of the evaluation key ek and a value $x \in \{0,1\}^L$, the evaluation algorithm computes the output

$$g^{\left(\prod_{j=1}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j}\right)^\top \cdot \mathbf{u}},$$

i.e. the representation of $\mathsf{AHF}(x)$ decides which matrices are used. The corresponding proof $\pi := (\pi_1, \ldots, \pi_{n-1})$ consists of all intermediate values

$$\pi_\iota := g^{\left(\prod_{j=1}^{\iota} \mathbf{M}_{j,\mathsf{AHF}(x)_j}\right)^\top \cdot \mathbf{u}}$$

for $\iota \in \{1, \ldots, n-1\}$ and has thus linear size. Given the verification key, a proof can be verified employing a symmetric bilinear map.

For trapdoor key generation, the following property of a matrix product is employed in [20]: Let $\mathcal{U}_\prime, \ldots, \mathcal{U}_\backslash$ be 2-dimensional subspaces of $\mathbb{Z}_p^3$ and for all $j \in \{1, \ldots, n\}$, $k \in \Sigma$ let $\mathbf{M}_{j,k}$ be such that $\mathbf{M}_{j,k}^\top \cdot \mathcal{U}_{j-1} = \mathcal{U}_j$.[1] Then:

i.) If there exists a $j^\star \in \{1, \ldots, n\}$ such that $\mathbf{M}_{j^\star,\mathsf{AHF}(x)_{j^\star}}$ is of rank 2, then $(\prod_{j=1}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j})^\top$ maps $\mathbb{Z}_p^3$ (and thus in particular $\mathbb{Z}_p^3 \setminus \mathcal{U}_0$) to $\mathcal{U}_n$.
ii.) If for all $j \in \{1, \ldots, n\}$ the matrix $\mathbf{M}_{j,\mathsf{AHF}(x)_j}$ is of rank 3, then the product $(\prod_{j=1}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j})^\top$ maps $\mathbb{Z}_p^3 \setminus \mathcal{U}_0$ to $\mathbb{Z}_p^3 \setminus \mathcal{U}_n$.

Let $I \subseteq \{1, \ldots, n\}$ and $Y \in \Sigma^n$ constitute a partitioning of $\{0,1\}^L$ with $\mathcal{Y} := \{x \in \{0,1\}^L \mid \forall i \in I \colon \mathsf{AHF}(x)_i = Y_i\}$ and $\mathcal{Z} = \{0,1\}^L \backslash \mathcal{Y}$. Recall that to achieve *well-distributed outputs*, the goal is to set up trapdoor key generation such that $x \in \mathcal{Z} \Leftrightarrow c_3^x = 0$.

In [20] this is achieved as follows. Given a basis $\{g^{\mathbf{b}_1}, g^{\mathbf{b}_2}, g^{\mathbf{b}_3}\}$ of $\mathbb{G}^3$, the trapdoor key generation algorithm chooses vector spaces $\mathcal{U}_0, \ldots, \mathcal{U}_{n-1} \subseteq \mathbb{Z}_p^3$ and a vector $\mathbf{u} \leftarrow_R \mathbb{Z}_p^3 \backslash \mathcal{U}_0$ at random. Further, it defines $\mathcal{U}_n$ as the subspace generated by $\mathbf{b}_1$ and $\mathbf{b}_2$, and chooses $\mathbf{M}_{j,k}$ at random (subject to $\mathbf{M}_{j,k}^\top \cdot \mathcal{U}_{j-1} = \mathcal{U}_j$) of rank 2, whenever $j \in I$ and $k \neq Y_j$ (and of full-rank otherwise). In other words, it chooses all matrices corresponding to light gray cells in Figure 2 of rank 2, and all matrices corresponding to white, gray or black cells of rank 3. This implies that at least one matrix of rank 2 is part of the evaluation if and only if $x \in \mathcal{Z}$ (as in this case at least for one light gray line a matrix corresponding to the non-black cell is hit). And, by the choice of $\mathbf{u}$, $\mathcal{U}_n$ together with i.), we have $c_3^x = 0$ whenever at least one of the matrices in the product is of rank 2.

Note that, as $g^{\mathbf{b}_1}, g^{\mathbf{b}_2}$ and thus $\mathcal{U}_n$ are only known in the exponent, the trapdoor key generation algorithm can only compute $g^{\mathbf{M}_{n,k}}$ in the exponent (for

[1] For matrix $\mathbf{M} \in \mathbb{Z}_p^3$ and subspaces $\mathcal{U}, \mathcal{V} \subseteq \mathbb{Z}_p^3$, by $\mathbf{M}^\top \cdot \mathcal{U} = \mathcal{V}$ we denote the property that for all $\mathbf{u} \in \mathcal{U}$ we have $\mathbf{M}^\top \mathbf{u} \in \mathcal{V}$ and for each $\mathbf{v} \in \mathcal{V}$ there exists a $\mathbf{u}$ with $\mathbf{M}^\top \mathbf{u} = \mathbf{v}$.

all $k \in \Sigma$). This does not hinder evaluation, as all matrices $\mathbf{M}_{j,k}$ for $j < n, k \in \Sigma$ are known in $\mathbb{Z}_p^{3\times 3}$.

Note that the strategy of [20] requires the product to be taken over *all indices* of $\mathsf{AHF}(x)$. As the proof has to comprise all intermediate steps of the product in order to be verifiable with a symmetric pairing, the proof size is inherently linear in n and thus in L. We now explain how to overcome this barrier.

Towards our construction. Observe that hypothetically in order to achieve well-distributed outputs it would suffice to multiply all matrices with $j \in I$ (in other words to skip all white rows in Figure 2), thereby allowing much shorter proofs. The problem is, of course, that evaluation has to be independent of I.

We resolve this issue by setting up the underlying function in a different way. First of all, in order to be independent of I, the function evaluation has to be dependent on all indices of $\mathsf{AHF}(x)$ (and not only the ones in a fixed index set I). To this end, we first pretend $|I| = 1$. The idea is to replace the product by a sum, which can be evaluated directly given the verification key without requiring a pairing. More precisely, the prototype of our VVHF is of the form

$$g^{(\sum_{j=1}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j})^\top \mathbf{u}}.$$

The key in our proof of adaptive programmability are two observations concerning the sum of matrices. Namely, let $\mathcal{U}_0$, $\mathcal{U}_1$ be 2-dimensional subspaces of $\mathbb{Z}_p^3$ and for all j, k let $\mathbf{M}_{j,k}$ be such that $\mathbf{M}_{j,k}^\top \cdot \mathcal{U}_0 = \mathcal{U}_1$. Then:

iii.) If $\mathbf{M}_{j,\mathsf{AHF}(x)_j}$ is of rank 2 for all j, $(\sum_{j=1}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j})^\top$ maps $\mathbb{Z}_p^3$ (and thus in particular $\mathbb{Z}_p^3 \setminus \mathcal{U}_0$) to $\mathcal{U}_1$.

iv.) If there exists exactly one $j^\star \in \{1, \ldots, n\}$ such that $\mathbf{M}_{j^\star,\mathsf{AHF}(x)_{j^\star}}$ is of full rank (and the rest of the matrices are of rank at most 2), then the sum $(\sum_{j=1}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j})^\top$ maps $\mathbb{Z}_p^3 \setminus \mathcal{U}_0$ to $\mathbb{Z}_p^3 \setminus \mathcal{U}_1$. (This is due to the fact that for any $\mathbf{z} \in \mathbb{Z}_p^3 \setminus \mathcal{U}_0$ it holds $(\sum_{j=1, j\neq j^\star}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j})^\top \cdot \mathbf{z} \in \mathcal{U}_1$ and $\mathbf{M}_{j^\star,\mathsf{AHF}(x)_{j^\star}} \cdot \mathbf{z} \in \mathbb{Z}_p^3 \setminus \mathcal{U}_1$.)

Now, given a basis $\{g^{\mathbf{b}_1}, g^{\mathbf{b}_2}, g^{\mathbf{b}_3}\}$ of $\mathbb{G}^3$, the trapdoor key generation algorithm chooses a random 2 dimensional vector space $\mathcal{U}_0 \subseteq \mathbb{Z}_p^3$, defines $\mathcal{U}_1$ as the vector space generated by $\mathbf{b}_1$ and $\mathbf{b}_2$. Further, the algorithm chooses a vector $\mathbf{u} \in \mathbb{Z}_p^3 \setminus \mathcal{U}_0$ at random, and chooses $\mathbf{M}_{j,k}$ of rank 3 if $j \in I$ and $k = Y_j$, and of rank 2 otherwise. This corresponds to choosing the matrix corresponding to the single black cell in the partitioning table in Figure 2 of rank 3, and all other matrices (corresponding to gray, light gray and white cells) of rank 2. We have

$$x \in \mathcal{Z} \Rightarrow \forall j \in \{1, \ldots, n\} \colon \mathbf{M}_{j,\mathsf{AHF}(x)_j} \text{ is of rank 2} \overset{\text{iii.)}}{\Rightarrow} \sum_{j=1}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j}^\top \cdot \mathbf{u} \in \mathcal{U}_1,$$

$$x \in \mathcal{Y} \Rightarrow \mathbf{M}_{j^\star,\mathsf{AHF}(x)_{j^\star}} \text{ is of rank 3 for } j^\star \in I \overset{\text{iv.)}}{\Rightarrow} \sum_{j=1}^{n} \mathbf{M}_{j,\mathsf{AHF}(x)_j}^\top \cdot \mathbf{u} \in \mathbb{Z}_p^3 \setminus \mathcal{U}_1.$$

Our construction of a VVHF with short proofs. For $|I| > 1$, the idea is to repeat this strategy for every $i \in I$ and multiply the results. Note that in

order to have evaluation completely oblivious to I (which may vary in size), we employ an upper bound ℓ on the size of I. (Recall that for the instantiation of Bitansky [7], we can choose any $\ell \in \omega(1)$.)

We can now present our final construction. For every $i \in \{1, \ldots, \ell\}$ we choose a fresh set of matrices $(\mathbf{M}_{i,j,k})_{j \in \{1,\ldots,n\}, k \in \Sigma}$, each uniformly at random from $\mathbb{Z}_p^{3\times 3}$ and further a vector $\mathbf{u} \leftarrow_R \mathbb{Z}_p^3$. Our evaluation key is of the form

$$ek := \left((\mathbf{M}_{i,j,k})_{i \in \{1,\ldots,\ell\}, j \in \{1,\ldots,n\}, k \in \Sigma}, \mathbf{u}\right)$$

and the verification key is defined as

$$vk := \left(\left(g^{\mathbf{M}_{i,j,k}}\right)_{i \in \{1,\ldots,\ell\}, j \in \{1,\ldots,n\}, k \in \Sigma}, g^{\mathbf{u}}\right).$$

To evaluate our VVHF on input x, we compute

$$g^{\left(\prod_{i=1}^{\ell} \sum_{j=1}^{n} \mathbf{M}_{i,j,\mathsf{AHF}(x)_j}\right)^\top \mathbf{u}}$$

and publish the proof $\pi := (\pi_1, \ldots, \pi_{\ell-1})$ consisting of the intermediate steps in computing the product, that is

$$\pi_\iota := g^{\left(\prod_{i=1}^{\iota} \sum_{j=1}^{n} \mathbf{M}_{i,j,\mathsf{AHF}(x)_j}\right)^\top \mathbf{u}}$$

for all $\iota \in \{1, \ldots, \ell - 1\}$.

Trapdoor key generation proceeds as follows. For $i \in \{1, \ldots, |I|\}$, let j_i refer to the i-th index in I. On input of a basis $\{g^{\mathbf{b}_1}, g^{\mathbf{b}_2}, g^{\mathbf{b}_3}\}$ of $\mathbb{G}^3$, we choose vector spaces $\mathcal{U}_0, \ldots, \mathcal{U}_{\ell-1} \subseteq \mathbb{Z}_p^3$ of dimension 2 and define $\mathcal{U}_\ell$ to be the vector space generated by $\mathbf{b}_1, \mathbf{b}_2$. Further, we choose $\mathbf{u} \in \mathbb{Z}_p^3 \setminus \mathcal{U}_0$ and matrices $\mathbf{M}_{i,j,k}$ subject to $\mathbf{M}_{i,j,k} \cdot \mathcal{U}_{i-1} = \mathcal{U}_i$ as follows:

- For all $i \in \{1, \ldots, |I|\}$, we choose $\mathbf{M}_{i,j_i,Y_{j_i}}$ of *rank* 3.
- For all $i \in \{|I| + 1, \ldots, \ell\}$ and all $k \in \Sigma$, we choose $\mathbf{M}_{i,1,k}$ of *rank* 3. (These matrices constitute *dummy matrices* in order to make evaluation oblivious to $|I|$.)
- For all other indices i, j, k we choose $\mathbf{M}_{i,j,k}$ of *rank* 2.

To go back to Figure 2, this can be viewed as setting up ℓ copies of the partitioning table, where for $i \in \{1, \ldots, |I|\}$ we choose *only* the matrix corresponding to the black cell in row j_i (i.e. in the i-th light gray row) of rank 3 and all other matrices of rank 2. For $i \in \{|I| + 1, \ldots, \ell\}$, we choose all matrices corresponding to the first row of rank 3 (and all other matrices of rank 2). During evaluation, for each $i \in \{1, \ldots, \ell\}$, we sum up all matrices corresponding to the cells $(j, \mathsf{AHF}(x)_j)$ for $j \in \{1, \ldots, n\}$. Whenever $x \in \mathcal{Y}$, we hit exactly one matrix of rank 3 for all $i \in \{1, \ldots, \ell\}$, as for $i \leq |I|$ we hit the matrix corresponding to (j_i, Y_{j_i}) and for $i > |I|$ we always hit one matrix in the first row. Therefore, by iv.) and ii.) the output will be an element of $\mathbb{Z}_p^3 \setminus \mathcal{U}_\ell$. For all $x \in \mathcal{Z}$, on the other hand, there exists at least one $i \in \{1, \ldots, |I|\}$ for which the matrix of rank 3 is not part of the sum. Thus, by iii.) and i.) the output will be an element of $\mathcal{U}_\ell$.

Note that similar to [20] the trapdoor key generation algorithm can only compute $g^{\mathbf{M}_{\ell,j,k}}$ in the exponent for all $j \in \{1,\ldots,n\}, k \in \Sigma$, which is sufficient as all other matrices are known in $\mathbb{Z}_p^{3\times 3}$.

Similar to [20], *indistinguishability* of verification keys generated by the trapdoor key generation from real verification keys can be proven via a hybrid argument, employing the decision linear assumption (or more precisely, the 3-*rank assumption*), which states that it is computationally indistinguishable whether a matrix was drawn uniformly at random from $\{g^{\mathbf{M}} \in \mathbb{G}^{3\times 3} \mid \mathbf{M} \text{ has rank } 3\}$ or from $\{g^{\mathbf{M}} \in \mathbb{G}^{3\times 3} \mid \mathbf{M} \text{ has rank } 2\}$.

We call our approach *hunting and gathering*, as our strategy is to *hunt* out all values $\mathsf{AHF}(x)$ disagreeing with Y on at least one index in I. We do so by setting up ℓ sets of matrices and *gathering* the matrices corresponding to the characters of $\mathsf{AHF}(x)$ for each of these sets. If $\mathsf{AHF}(x)$ disagrees with Y on the i-th index of I, then this will show up in the sum of matrices corresponding to the i-th set.

2 Preliminaries

We will use the following notation throughout this paper. By $\lambda \in \mathbb{N}$ we denote the security parameter. By $L := L(\lambda) \in \mathbb{N}$ we denote the input length, a canonical choice is $L = \lambda$. Further, by the constant $d \geq 3$ we denote the parameter of our assumption. We implicitly assume all other parameters to depend on λ. For an arbitrary set $\mathcal{S}$, by $x \leftarrow_R \mathcal{S}$ we denote the process of sampling an element x from $\mathcal{S}$ uniformly at random. Throughout, $p \in \mathbb{N}$ will be prime. We interpret vectors $\mathbf{v} \in \mathbb{Z}_p^d$ as column-vectors, i.e. $\mathbf{v} \in \mathbb{Z}_p^{d\times 1}$. Further, by v_j we denote the j-th entry of $\mathbf{v}$ for $j \in \{1,\ldots,d\}$. We say that a function is *negligible in* λ if its inverse vanishes asymptotically faster than any polynomial in λ. We say that $\mathcal{A}$ is *probabilistic polynomial time* (PPT), if $\mathcal{A}$ is a probabilistic algorithm with running time polynomial in λ. We use $y \leftarrow \mathcal{A}(x)$ to denote that y is assigned the output of $\mathcal{A}$ running on input x.

In order to formally treat *uniqueness* of proofs, we take the notion of *certified bilinear group generators* from [20]. Note that in the following all numbered references refer to the eprint version [22] of [20].

Definition 1 (Certified bilinear group generator [22, Def. 2.1/2.2]). *A* bilinear group generator *is a PPT algorithm* $\mathsf{BG.Gen}$ *that on input* 1^λ *outputs* $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, \circ, \circ_T, e, \varphi, \varphi_T) \leftarrow \mathsf{BG.Gen}(1^\lambda)$ *such that the following are satisfied*

- p *is a* 2λ*-bit prime*
- $\mathbb{G}$ *and* $\mathbb{G}_T$ *are subsets of* $\{0,1\}^\lambda$*, defined by algorithmic descriptions of maps* $\varphi\colon \mathbb{Z}_p \to \mathbb{G}$ *and* $\varphi_T\colon \mathbb{Z}_p \to \mathbb{G}_T$
- $\circ$ *and* $\circ_T$ *are algorithmic descriptions of efficiently computable maps* $\rho\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ *and* $\rho_T\colon \mathbb{G}_T \times \mathbb{G}_T \to \mathbb{G}_T$*, such that the following hold*
  i. $(\mathbb{G}, \circ)$ *and* $(\mathbb{G}_T, \circ_T)$ *form algebraic groups*
  ii. φ *is a group isomorphism from* $(\mathbb{Z}_p, +)$ *to* $(\mathbb{G}, \circ)$
  iii. φ_T *is a group isomorphism from* $(\mathbb{Z}_p, +)$ *to* $(\mathbb{G}_T, \circ_T)$

– *e is the description of an efficiently computable non-degenerate bilinear map* $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$*, that is*
i. $e(\varphi(1), \varphi(1)) \neq \varphi_T(0)$
ii. for all $a \in \mathbb{Z}_p$*:* $e(\varphi(a), \varphi(1)) = e(\varphi(1), \varphi(a)) = \varphi_T(a)$

In the following we will only include $\varphi(1)$ *in the description of* $\mathcal{G}$*. Note that this suffices, as* $\varphi(1)$ *uniquely determines* φ *and* φ_T*.*

We say a group generator is certified*, if there exists a deterministic polynomial time algorithm* $\mathsf{BG.Vfy} = (\mathsf{BG.Vfy}_1, \mathsf{BG.Vfy}_2)$ *with the following properties*

Parameter validation. *Given a string* $\mathcal{G}$*, the algorithm* $\mathsf{BG.Vfy}_1(\mathcal{G})$ *outputs* 1 *if and only if* $\mathcal{G}$ *has the form* $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, \circ, \circ_T, e, \varphi(1)) \leftarrow \mathsf{BG.Gen}(1^\lambda)$ *and all requirements from above are satisfied.*

Recognition and unique representation of elements in $\mathbb{G}$**.** *Further, we require that each element in* $\mathbb{G}$ *has unique representation, which can be efficiently recognized. That is, on input of two strings* Π *and* s*,* $\mathsf{BG.Vfy}_2(\Pi, s)$ *outputs* 1 *if and only if* $\mathsf{BG.Vfy}_1(\Pi) = 1$ *and it holds that* $s = \varphi(x)$ *for some* $x \in \mathbb{Z}_p$*.*

Let $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, \circ, \circ_T, e, \varphi(1)) \leftarrow \mathsf{BG.Gen}(1^\lambda)$ be a bilinear group. We use the representation of group elements introduced in [13]. Namely, for $a \in \mathbb{Z}_p$, define $[a] := \varphi(a) \in \mathbb{G}$ as the *implicit representation* of a in $\mathbb{G}$. More generally, for any $n, m \in \mathbb{N}$ and any matrix $\mathbf{A} = (a_{ij}) \in \mathbb{Z}_p^{n \times m}$ we define $[\mathbf{A}]$ as the implicit representation of $\mathbf{A}$ in $\mathbb{G}$:

$$[\mathbf{A}] := \begin{pmatrix} \varphi(a_{11}) & \dots & \varphi(a_{1m}) \\ \vdots & & \vdots \\ \varphi(a_{n1}) & \dots & \varphi(a_{nm}) \end{pmatrix} \in \mathbb{G}^{n \times m}$$

Note that from $[a] \in \mathbb{G}$ it is hard to compute the value a if the Discrete Logarithm assumption holds in $\mathbb{G}$. Obviously, given $[a], [b] \in \mathbb{G}$ and a scalar $x \in \mathbb{Z}_p$, one can efficiently compute $[ax] \in \mathbb{G}$ and $[a + b] \in \mathbb{G}$.

We give the $(d-1)$-linear assumption in a similar form as provided in [20]. This is equivalent to the standard formulation in [13].

Definition 2 ($(d-1)$-linear assumption [22, Assumption 5.3]). *Let* $\mathcal{G} \leftarrow \mathsf{BG.Gen}(1^\lambda)$ *be the description of a bilinear group. The* $(d-1)$-linear assumption *over* $\mathcal{G}$ *states that for all PPT adversaries* $\mathcal{A}$ *the advantage*

$$\begin{aligned} \mathsf{Adv}^{(d-1)-\mathsf{lin}}_{\mathcal{G},\mathcal{A}} := \big| &\Pr[\mathcal{A}(1^\lambda, \mathcal{G}, [\mathbf{c}], [\mathbf{d}], [\textstyle\sum_{i=1}^{d-1} d_i/c_i]) \mid \mathbf{c}, \mathbf{d} \leftarrow_R \mathbb{Z}_p^{d-1}] \\ &- \Pr[\mathcal{A}(1^\lambda, \mathcal{G}, [\mathbf{c}], [\mathbf{d}], [r]) \mid \mathbf{c}, \mathbf{d} \leftarrow_R \mathbb{Z}_p^{d-1}, r \leftarrow_R \mathbb{Z}_p]\big| \end{aligned}$$

is negligible in λ*.*

For $d = 2$*, this corresponds to the* decisional Diffie-Hellman assumption (DDH)*.*

For $d = 3$*, this corresponds to the* decision linear assumption (DLIN)*.*

Note that given a bilinear group with symmetric pairing, the decisional Diffie-Hellman assumption does not hold. For the most efficient instantiation, we thus choose $d = 3$ for this work.

The following assumption can be viewed as a relaxation of the $(d-1)$-linear assumption.

Definition 3 (d-rank assumption [22, Assumption 4.1]). *Again, let $\mathcal{G} \leftarrow \mathsf{BG.Gen}(1^\lambda)$ be the description of a bilinear group. The* d-rank assumption *over $\mathcal{G}$ states that for all PPT adversaries $\mathcal{A}$ the advantage*

$$\mathsf{Adv}^{d-\mathsf{rank}}_{\mathcal{G},\mathcal{A}} := \big|\Pr[\mathcal{A}(1^\lambda, \mathcal{G}, [\mathbf{M}]) \mid \mathbf{M} \leftarrow_R \mathbb{Z}_p^{d\times d} \text{ of rank } d-1] - \Pr[\mathcal{A}(1^\lambda, \mathcal{G}, [\mathbf{M}]) \mid \mathbf{M} \leftarrow_R \mathbb{Z}_p^{d\times d} \text{ of rank } d]\big|$$

is negligible in λ.

By $e\colon \mathbb{G}^{d\times d} \times \mathbb{G}^d \to \mathbb{G}_T^d$ we denote the natural componentwise extension of e to $\mathbb{G}^{d\times d} \times \mathbb{G}^d$, that is let $\mathbf{M} = (m_{i,j})_{i,j} \in \mathbb{Z}_p^{d\times d}$ be a matrix and $\mathbf{x} = (x_i)_i \in \mathbb{Z}_p^d$ be a vector, then

$$e\colon \mathbb{G}^{d\times d} \times \mathbb{G}^d \to \mathbb{G}_T^d,\ ([\mathbf{M}], [\mathbf{x}]) \mapsto \begin{pmatrix} e([m_{1,1}], [x_1]) \circ_T \cdots \circ_T e([m_{1,d}], [x_d]) \\ \vdots \\ e([m_{d,1}], [x_1]) \circ_T \cdots \circ_T e([m_{d,d}], [x_d]) \end{pmatrix}.$$

2.1 Verifiable vector hash functions and verifiable random functions

Basically, verifiable vector hash functions (VVHF) are a pre-stage of verifiable random functions, where the image is a vector space. Further, instead of pseudo-randomness of the output, VVHFs are required to be *adaptively programmable*. An adaptively programmable VVHF has a trapdoor key generation algorithm which is indistinguishable from standard key generation and further meets *well-distributed outputs*, which allows transforming it to a verifiable random function via the generic transformation [20] whenever the decision linear assumption holds in the underlying group. In the following we will recall the definition of adaptively programmable VVHFs from [20], recall the definition of a verifiable random function (VRF) and present the generic transformation from an adaptively programmable VVHF to a VRF (without proof).

Definition 4 (Verifiable vector hash function (VVHF) [22, Def. 3.1]). *Let $\mathsf{BG.Gen}$ be a bilinear group generator and let $d \in \mathbb{N}$. A* verifiable vector hash function (VVHF) *for $\mathsf{BG.Gen}$ with domain $\{0,1\}^L$ and range $\mathbb{G}^d$ is a tuple of PPT algorithms $\mathsf{VVHF} := (\mathsf{VVHF.Gen}, \mathsf{VVHF.Eval}, \mathsf{VVHF.Vfy})$ with the following properties.*

- $\mathsf{VVHF.Gen}(\mathcal{G})$ *for $\mathcal{G} \leftarrow \mathsf{BG.Gen}$, outputs a verification key vk and an evaluation key ek.*
- $\mathsf{VVHF.Eval}(ek, x)$ *for an evaluation key ek and $x \in \{0,1\}^L$, outputs a function value $[\mathbf{v}] \in \mathbb{G}^d$ and a corresponding proof of correctness π.*

– $\mathsf{VVHF.Vfy}(vk, [\mathbf{v}], \pi, x)$ *is a deterministic algorithm that outputs a bit* $b \in \{0,1\}$.

Further, we require the following to hold.

Correctness. *We say that* VVHF *is* correct*, if for all* λ*, all* $\mathcal{G}$ *in the image of* $\mathsf{BG.Gen}(1^\lambda)$*, all* (vk, ek) *in the image of* $\mathsf{VVHF.Gen}(\mathcal{G})$*, all* $x \in \{0,1\}^L$ *and all* $([\mathbf{v}], \pi)$ *in the image of* $\mathsf{VVHF.Eval}(ek, x)$ *we have*

$$\mathsf{VVHF.Vfy}(vk, [\mathbf{v}], \pi, x) = 1.$$

Unique provability. *We say that a verifiable vector hash function* VVHF *satisfies* unique provability*, if for all possible* vk *(not necessarily created by* $\mathsf{VVHF.Gen}$*), all* $x \in \{0,1\}^L$*, all* $[\mathbf{v}_0], [\mathbf{v}_1] \in \mathbb{G}^d$ *and all possible proofs* π_0, π_1 *we have*

$$\mathsf{VVHF.Vfy}(vk, [\mathbf{v}_0], \pi_0, x) = \mathsf{VVHF.Vfy}(vk, [\mathbf{v}_1], \pi_1, x) = 1 \implies [\mathbf{v}_0] = [\mathbf{v}_1].$$

In other words, for any $x \in \{0,1\}^L$ *there exists a valid proof for at most one function value.*

Note that the following definition slightly differs from the notion of adaptive programmability in [20]. Namely, we additionally provide the algorithm $\mathsf{VVHF.TrapGen}$ with the parameter Q in the experiment for well-distributed outputs. Note that employing admissible hash functions to achieve full-adaptive security, in [20] $\mathsf{VVHF.TrapGen}$ already implicitly depends on Q to achieve well-distributed outputs. This does not affect the generic transformation from a verifiable vector hash function to a verifiable random function, as for this transformation the existence of a suitable tuple of trapdoor algorithms ($\mathsf{VVHF.TrapGen}$, $\mathsf{VVHF.TrapEval}$) suffices (without requiring explicit knowledge of Q).

Definition 5 (Adaptive programmability [22, Def. 3.3]). *We say that a verifiable vector hash function* $(\mathsf{VVHF.Gen}, \mathsf{VVHF.Eval}, \mathsf{VVHF.Vfy})$ *is* adaptively programmable (AP)*, if an additional tuple of algorithms* ($\mathsf{VVHF.TrapGen}$, $\mathsf{VVHF.TrapEval}$) *with the following properties exist.*

– $\mathsf{VVHF.TrapGen}(\mathcal{G}, Q, [\mathbf{B}])$ *for a bilinear group* $\mathcal{G} \leftarrow_R \mathsf{BG.Gen}(1^\lambda)$*, a parameter* Q *which is polynomially bounded in* λ *and a matrix* $[\mathbf{B}] \in \mathbb{G}^{d \times d}$*, outputs a verification key* vk *and a trapdoor* td*.*
– $\mathsf{VVHF.TrapEval}(td, x)$ *for a trapdoor* td *and* $x \in \{0,1\}^L$*, outputs a vector* $\mathbf{c} \in \mathbb{Z}_p^d$ *and a proof* π*.*

Further, we require the following.

Correctness. *We say that* $(\mathsf{VVHF.TrapGen}, \mathsf{VVHF.TrapEval})$ *satisfies* correctness *respective to* $\mathsf{VVHF.Vfy}$*, if for all* $\lambda \in \mathbb{N}$*, for all bilinear groups* $\mathcal{G}$ *in the image of* $\mathsf{BG.Gen}(1^\lambda)$*, for all* Q *that are polynomially bounded in* λ*, for all* $[\mathbf{B}] \in \mathbb{G}^{d \times d}$*, for all* $x \in \{0,1\}^L$ *for all* (vk, td) *in the image of* $\mathsf{VVHF.TrapGen}(\mathcal{G}, Q, [\mathbf{B}])$*, for all* $(\mathbf{c}, \pi)$ *in the image of* $\mathsf{VVHF.TrapEval}(td, x)$ *and for all* $[\mathbf{v}] := [\mathbf{B}] \cdot \mathbf{c}$ *it holds*

$$\mathsf{VVHF.Vfy}(vk, [\mathbf{v}], \pi, x) = 1.$$

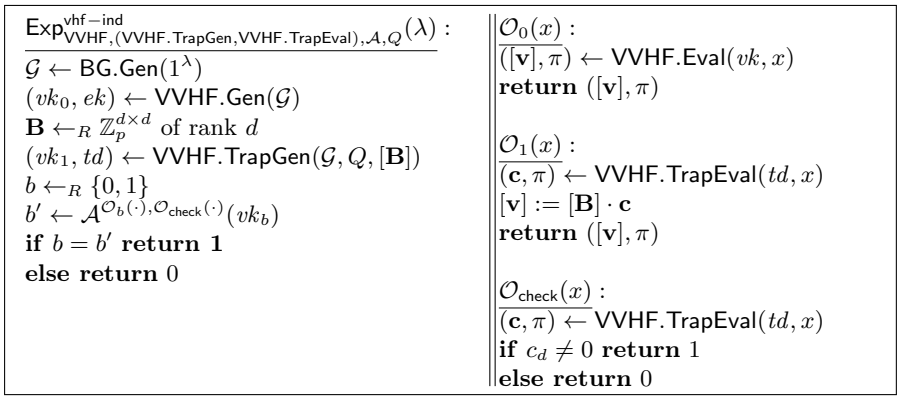

Fig. 3: VVHF Indistinguishability experiment. Note that $\mathcal{O}_{\mathsf{check}}$ always uses td and $\mathsf{VVHF.TrapEval}$, independently of bit b.

Indistiguishability. *We define an indistinguishability experiment in Figure 3. We say that* $(\mathsf{VVHF.TrapGen}, \mathsf{VVHF.TrapEval})$ *satisfies* indistinguishability*, if for all Q polynomial in λ and all PPT adversaries $\mathcal{A}$ we have that*

$$\mathsf{Adv}^{\mathsf{vhf-ind}}_{\mathsf{VVHF},(\mathsf{VVHF.TrapGen},\mathsf{VVHF.TrapEval}),\mathcal{A},Q}(\lambda) := \left|\Pr[\mathsf{Exp}^{\mathsf{vhf-ind}}_{\mathsf{VVHF},(\mathsf{VVHF.TrapGen},\mathsf{VVHF.TrapEval}),\mathcal{A},Q}(\lambda) = 1] - \frac{1}{2}\right|$$

is negligible in λ. In other words, we require that verification keys generated by $\mathsf{VVHF.TrapGen}$ *are indistinguishable from verification keys generated by* $\mathsf{VVHF.Gen}$*.*

Well-distributed outputs. *Let Q be polynomial in λ and let $x^{(1)}, \ldots, x^{(Q)}$, $x^\star \in \{0,1\}^L$ arbitrary with $x^\star \notin \{x^{(1)}, \ldots, x^{(Q)}\}$. Let $\mathcal{G} \leftarrow \mathsf{BG.Gen}(1^\lambda)$, $\mathbf{B} \leftarrow_R \mathbb{Z}_p^{d\times d}$ of rank d and $(vk, td) \leftarrow \mathsf{VVHF.TrapGen}(\mathcal{G}, Q, [\mathbf{B}])$. Further, for all $\nu \in \{1, \ldots, Q\}$ let $(\mathbf{c}^{(\nu)}, \pi) \leftarrow_R \mathsf{VVHF.TrapEval}(td, x^{(\nu)})$ and $(\mathbf{c}^\star, \pi) \leftarrow_R \mathsf{VVHF.TrapEval}(td, x^\star)$. Let $\Pr^{\mathsf{well-distr}}_{(\mathsf{VVHF.TrapGen},\mathsf{VVHF.TrapEval}),\lambda,Q}(\{x^{(\nu)}\}_\nu, x^\star)$ be the probability that $c_d^{(\nu)} = 0$ for all $\nu \in \{1, \ldots, Q\}$ and $c_d^\star \neq 0$ (where the probability is taken over the random coins of* $\mathsf{BG.Gen}, \mathsf{VVHF.TrapGen}, \mathsf{VVHF.TrapEval}$ *and the random choice of $\mathbf{B}$).*

We say that $(\mathsf{VVHF.TrapGen}, \mathsf{VVHF.TrapEval})$ *satisfies* well-distributed outputs*, if for all Q polynomial in λ and all $x^{(1)}, \ldots, x^{(Q)}, x^\star \in \{0,1\}^L$ with $x^\star \notin \{x^{(1)}, \ldots, x^{(Q)}\}$ we have*

$$\Pr^{\mathsf{well-distr}}_{(\mathsf{VVHF.TrapGen},\mathsf{VVHF.TrapEval}),\lambda,Q}(\{x^{(\nu)}\}_\nu, x^\star) \geq \frac{1}{\mathsf{poly}(\lambda)}.$$

.

Definition 6 (Verifiable random functions [27], Notation [22, Def. 5.1]). *Let* $\mathsf{VRF} := (\mathsf{VRF.Gen}, \mathsf{VRF.Eval}, \mathsf{VRF.Vfy})$ *be a tuple of polynomial-time algorithms of the following form.*

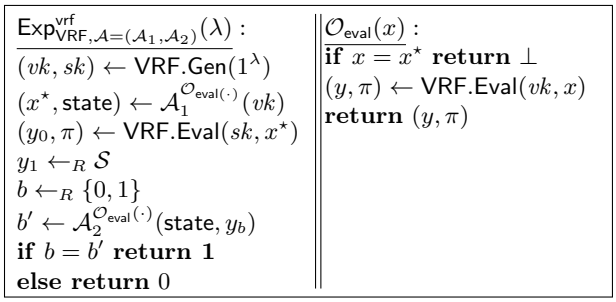

Fig. 4: The VRF security experiment.

- $\mathsf{VRF.Gen}(1^\lambda)$ *outputs a secret key* sk *and a verification key* vk*.*
- $\mathsf{VRF.Eval}(sk, x)$ *for a secret key* sk *and an input* $x \in \{0,1\}^L$*, outputs a function value* $y \in \mathcal{S}$ *(where* $\mathcal{S}$ *is a finite set) and a proof* π*.*
- $\mathsf{VRF.Vfy}(vk, x, y, \pi)$ *is a deterministic algorithm that for a verification key* vk*, an input* $x \in \{0,1\}^L$*, an output* $y \in \mathcal{S}$ *and a proof* π*, outputs a bit* $b \in \{0,1\}$*.*

We say VRF *is a* verifiable random function*, if the following properties hold.*

Correctness. *For all* $\lambda \in \mathbb{N}$*, for all* (vk, sk) *in the image of* $\mathsf{VVHF.Gen}(1^\lambda)$*, for all* $x \in \{0,1\}^L$ *and for all* (y, π) *in the image of* $\mathsf{VVHF.Eval}(sk, x)$ *it holds*

$$\mathsf{VRF.Vfy}(vk, x, y, \pi) = 1.$$

Unique provability. *We say that a verifiable random function* VRF *satisfies* unique provability*, if for all possible* vk *(not necessarily created by* $\mathsf{VRF.Gen}$*), all* $x \in \{0,1\}^L$*, all* $y_0, y_1 \in \mathcal{S}$ *and all possible proofs* π_0, π_1 *we have*

$$\mathsf{VRF.Vfy}(vk, x, y_0, \pi_0) = \mathsf{VRF.Vfy}(vk, x, y_1, \pi_1) = 1 \implies y_0 = y_1.$$

In other words, for any $x \in \{0,1\}^L$ *there exists a valid proof for at most one function value.*

Pseudorandomness. *We define a VRF security experiment in Figure 4. We say that a verifiable random function* VRF *is* pseudorandom*, if for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *we have that*

$$\mathsf{Adv}^{\mathsf{vrf}}_{\mathsf{VRF},\mathcal{A}}(\lambda) := \left| \Pr[\mathsf{Exp}^{\mathsf{vrf}}_{\mathsf{VRF},\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right|$$

is negligible in λ*. In other words, we require that the output of* $\mathsf{VRF.Eval}$ *is indistinguishable from random values in* $\mathcal{S}$*.*

Given a verifiable vector hash function with adaptive programmability, one can obtain a verifiable random function via the generic construction of [20]. As we will employ this generic transformation, we want to recall it in the following.

Let $\mathsf{BG.Gen}$ be a certified bilinear group generator according to Definition 1. Let $\mathsf{VVHF} := (\mathsf{VVHF.Gen}, \mathsf{VVHF.Eval}, \mathsf{VVHF.Vfy})$ be a vector hash function according to Definition 4. Let $\mathsf{VRF} := (\mathsf{VRF.Gen}, \mathsf{VRF.Eval}, \mathsf{VRF.Vfy})$ be defined as follows.

- $\mathsf{VRF.Gen}(1^\lambda)$ runs $\mathcal{G} \leftarrow \mathsf{BG.Gen}(1^\lambda)$ and $(ek, vk') \leftarrow \mathsf{VVHF.Gen}(\mathcal{G})$. Further, it chooses a random vector $\mathbf{w} \leftarrow_R (\mathbb{Z}_p^\star)^d$, defines $sk := (\mathcal{G}, ek, \mathbf{w})$ and $vk := (\mathcal{G}, vk', [\mathbf{w}])$ and outputs (vk, sk).
- $\mathsf{VRF.Eval}(sk, x)$ for $sk = (\mathcal{G}, ek, \mathbf{w})$ and $x \in \{0,1\}^L$ first runs

  $$([\mathbf{v}], \pi') \leftarrow \mathsf{VVHF.Eval}(ek, x).$$

  Then, $\mathsf{VRF.Eval}$ computes the function value y and an additional proof $[\mathbf{z}] \in \mathbb{G}^d$ as

  $$y := \prod_{i=1}^{d} \left[\frac{v_i}{w_i}\right] \text{ and } [\mathbf{z}] := \left[\left(\frac{v_1}{w_1}, \frac{v_2}{w_2}, \ldots, \frac{v_d}{w_d}\right)^\top\right],$$

  where, $\prod$ corresponds to the $\circ$ operation over $\mathbb{G}$. Finally, $\mathsf{VRF.Eval}$ sets $\pi := ([\mathbf{v}], \pi', [\mathbf{z}])$ and outputs (y, π).
- $\mathsf{VRF.Vfy}(vk, x, y, \pi)$ outputs 1 if and only if all of the following properties are satisfied:
  - The verification key vk has the form $vk = (\mathcal{G}, vk', [\mathbf{w}])$ such that $[\mathbf{w}] \in \mathbb{Z}_p^d$ and the bilinear group parameters and the group elements contained in vk are valid, which can be checked by running $\mathsf{BG.Vfy}_1$ and $\mathsf{BG.Vfy}_2$.
  - The input x is an element of $\{0,1\}^L$.
  - The proof π has the form $\pi = ([\mathbf{v}], \pi', [\mathbf{z}])$ with $\mathsf{VVHF.Vfy}(vk', [\mathbf{v}], \pi', x) = 1$ and both vectors $[\mathbf{v}]$ and $[\mathbf{z}]$ contain only validly-encoded group elements, which can be checked by running $\mathsf{BG.Vfy}_2$.
  - It holds that $[z_i] = [v_i/w_i]$ for all $i \in \{1, \ldots, d\}$ and $y = [\sum_{i=1}^d v_i/w_i]$. This can be checked by testing

    $$e([z_i], [w_i]) \stackrel{?}{=} e([v_i], [1]) \; \forall i \in \{1, \ldots, d\} \text{ and } y \stackrel{?}{=} \prod_{i=1}^{k} [z_i].$$

By the following theorem this construction yields a verifiable random function. For a proof we refer to [20].

Theorem 1 ([22, Theorem 5.2/5.4]). *If the $(d-1)$-linear assumption holds relative to* $\mathsf{BG.Gen}$ *and if the tuple* $\mathsf{VVHF} := (\mathsf{VVHF.Gen}, \mathsf{VVHF.Eval}, \mathsf{VVHF.Vfy})$ *is an adaptively programmable hash function, then* $\mathsf{VRF} := (\mathsf{VRF.Gen}, \mathsf{VRF.Eval}, \mathsf{VRF.Vfy})$ *is a verifiable vector hash function satisfying all requirements of Definition 6.*

2.2 Partitioning Based on Admissible Hash Functions

In order to achieve verifiable random functions with adaptive security, we have to partition the input space into $\mathcal{Y}$ and $\mathcal{Z}$ such that with noticeable probability all evaluation queries are in $\mathcal{Z}$ while the challenge query is in $\mathcal{Y}$.

One commonly used method for partitioning are *admissible hash functions*, a concept first formalized in [8]. As [14] we allow the output space to consist of vectors over a alphabet Σ of polynomial size.

Definition 7 (Admissible hash functions, [8, 14]). *Let n be polynomial in λ and Σ an alphabet of size $\sigma := |\Sigma|$ polynomial in λ. Let*

$$\mathsf{AHF}\colon \{0,1\}^L \to \Sigma^n,$$

$Y \in \Sigma^n$ and $I \subseteq \{1,\dots,n\}$ define the partitioning

$$\mathcal{Y} := \left\{x \in \{0,1\}^L \mid \mathsf{AHF}(x)_j = Y_j \text{ for all } j \in I\right\} \text{ and } \mathcal{Z} := \{0,1\}^L \setminus \mathcal{Y}.$$

We say that AHF *is* Q-admissible*, if there exists a PPT algorithm* AHF.Part *that on input $(1^\lambda, Q)$ returns a value $Y \in \Sigma^n$ and a set of indices $I \subseteq \{1,\dots,n\}$, such that for any $x^{(1)},\dots,x^{(Q)},x^\star \in \{0,1\}^L$ with $x^\star \notin \{x^{(1)},\dots,x^{(Q)}\}$ we have*

$$\begin{aligned}\Pr^{\mathsf{part}}_{\mathsf{AHF},\mathsf{AHF.Part},\lambda,Q}(\{x^{(\nu)}\}_\nu, x^\star) :=& \Pr[x^\star \in \mathcal{Y} \wedge x^{(\nu)} \in \mathcal{Z}\ \forall \nu \in \{1,\dots,Q\}]\\ \geq& 1/\mathsf{poly}(\lambda),\end{aligned}$$

where the probability is taken over the random coins of AHF.Part*. We say that* AHF *is an* admissible hash function (AHF) *if* AHF *is Q-admissible for all Q that are polynomially bounded in λ.*

For our construction we will employ two instantiations of admissible hash functions. The first is the so-called *substring matching* by [27]. In [7] this partitioning method is generalized to polynomial output alphabets. This allows us to shrink the proof size even to $\omega(1)$ group elements.

Note that a common problem arising using partitioning techniques is that the abort probability of the security experiment might depend on the sequence of queries of the adversary. While in [7] this issue is resolved by requiring so-called *balanced* partitioning as proposed in [23], in [20] the *artificial abort* technique from [34] is employed going from verifiable vector hash functions to verifiable random functions. As we apply the transformation of [20] in a black-box way, for our purposes the concept of admissible hash functions is sufficient.

Substring matching. In the following we give the instantiation of admissible hash function from Lysyanskaya [27]. To this aim let $\mathsf{Enc}\colon \{0,1\}^L \to \{0,1\}^n$ be an error correcting code with minimal distance μn for a constant μ (i.e. any two codewords differ in at least μn positions). Note that there exist efficient instantiations of Enc with $n \in \mathcal{O}(L)$. For a proof of the following lemma we refer to [14], Theorem 2.

Lemma 1 ([27, 14, 7]). *Let $\mathsf{AHF}\colon \{0,1\}^L \to \{0,1\}^n$ be an error correcting code with minimal distance μn for a constant μ. Then there exists an algorithm* AHF.Part *such that* AHF *is an admissible hash function. In particular, for any $x^{(1)},\dots,x^{(Q)},x^\star \in \{0,1\}^L$ we have*

$$\Pr^{\mathsf{part}}_{\mathsf{AHF},\mathsf{AHF.Part},\lambda,Q}(\{x^{(\nu)}\}_\nu, x^\star) \geq (2Q)^{-1/(\mu \log e)}/2$$

Further, the size of the set I returned by $\mathsf{AHF.Part}(1^\lambda, Q)$ is logarithmic in λ for any Q which is polynomially bounded in λ.

Substring matching over polynomial alphabets The original instantiation by Bitansky [7] requires $n \in \Omega(L^2)$ and an error correcting code with minimum distance $n - L + 1$. To achieve a smaller verification key, we observe that it is actually sufficient to choose a code with $n \in \Omega(L^{1+\eta})$ for some constant $\eta > 0$ with minimum distance at least $n-L+1$. A suitable instantiation for both is the following code from [32]. We only give the encoding function, as it is sufficient for our purposes.

Remark 1 (Reed-Solomon-Code). Let $\sigma \geq n$ such that σ is a prime-power and let $\Sigma := \mathbb{F}_\sigma$ be the finite field consisting of σ elements. Let $u_1, \ldots, u_n \in \mathbb{F}_\sigma$ be n pairwise different elements and for $x \in \{0,1\}^L$ let $p_x \in \mathbb{F}_\sigma[\mathsf{X}]$ be defined via $p_x[\mathsf{X}] := \sum_{i=0}^{L-1} x_i \mathsf{X}^i$. Then

$$\mathsf{Enc}\colon \{0,1\}^L \to \Sigma^n, x \mapsto (p_x(u_1), \ldots, p_x(u_n))$$

defines a code. Further, for $x \neq y \in \{0,1\}^L$ the polynomial $p_x - p_y \neq 0$ has degree at most $L-1$. The code has thus minimal distance $n-L+1$ as required.

As the following lemma slightly deviates from the original lemma in [7], we provide a proof.

Lemma 2 ([7] Section 4.1.1 (in the eprint version)). *Let $0 < \eta \leq 1$ be a constant and let $n, \sigma \in \mathcal{O}\left(L^{1+\eta}\right)$ such that $\mathsf{AHF}\colon \{0,1\}^L \to \Sigma^n$ is an error correcting code with minimal distance $n-L+1$ and alphabet size $|\Sigma| = \sigma$. Then, there exists an algorithm $\mathsf{AHF.Part}$ such that AHF is an admissible hash function such that for any $x^{(1)}, \ldots, x^{(Q)}, x^\star \in \{0,1\}^L$ we have*

$$\Pr^{\mathsf{part}}_{\mathsf{AHF},\mathsf{AHF.Part},\lambda,Q}(\{x^{(\nu)}\}_\nu, x^\star) \geq (2Q)^{-1-1/\eta-\mathcal{O}(1/\log\lambda)}/2$$

Further, the size of the set I returned by $\mathsf{AHF.Part}(1^\lambda, Q)$ is constant for any Q which is polynomially bounded in λ.

Proof. We define $\mathsf{AHF.Part}$ as the algorithm that on input $(1^\lambda, Q)$ chooses a random value $Y \leftarrow_R \Sigma^n$, sets $c := \log(2Q)/(\eta \log\lambda)$, and returns Y together with random subset $I \subseteq \{1, \ldots, n\}$ of size c.

Let $i_1, \ldots, i_\iota \in \{1, \ldots, n\}$. Then by $i_{[l]}$ we denote the set $\{i_1, \ldots, i_\iota\}$. First, assume $S := \{x^{(1)}, \ldots, x^{(Q)}\}$ and $x^\star$ to be fixed. Then, for any $x^{(\nu)} \in S$, we have

$$\begin{aligned}
&\Pr_{I\subset\{1,\ldots,n\},|I|=c}\left[\mathsf{AHF}(x^{(\nu)})\big|_I = \mathsf{AHF}(x^\star)\big|_I\right] \\
&= \prod_{j=1}^{c} \Pr_{i_j \notin i_{[j-1]}}\left[\mathsf{AHF}(x^{(\nu)})_{i_j} = \mathsf{AHF}(x^\star)_{i_j} \,\middle|\, \mathsf{AHF}(x^{(\nu)})\big|_{i_{[j-1]}} = \mathsf{AHF}(x^\star)\big|_{i_{[j-1]}}\right] \\
&\leq \prod_{j=1}^{c}\left(\frac{L-j}{n}\right) \leq \prod_{j=1}^{c}\frac{1}{L^\eta} = L^{-c\eta},
\end{aligned}$$

where the first inequality follows from the fact that two codewords of AHF have at most $L-1$ bits in common and the second inequality from $n = L^{1+\eta}$. Further, for any fixed $x^\star, I$, we have

$$\Pr_{Y \leftarrow \Sigma^n}[\mathsf{AHF}(x^\star)\big|_I = Y\big|_I] = \sigma^{-c}.$$

Via a union bound we obtain

$$\begin{aligned}\Pr^{\mathsf{part}}_{\mathsf{AHF},\mathsf{AHF.Part},\lambda,Q}(\{x^{(\nu)}\}_\nu, x^\star) &\geq \sigma^{-c} \cdot (1 - Q \cdot L^{-c\eta}) \\ &\geq (2Q)^{-1-1/\eta-\log C/(\eta \log \lambda)}/2,\end{aligned}$$

where C is a constant with $\sigma \leq C \cdot n^{1+\eta}$.

Further, we have $|I| = c \in \mathcal{O}(1)$ as η constant and $\log(2Q) \in \mathcal{O}(\log \lambda)$ for any Q which is polynomially bounded in λ.

3 Verifiable random function with short proofs

In this section we present our construction of an adaptively programmable verifiable vector hash function, which can be seen as a rearrangement of the VVHF of [20]. Via our technique of *hunting and gathering* we achieve significantly shorter proofs. Applying the generic transformation of [20] (see Theorem 1) finally yields an adaptively secure verifiable random function. For the resulting sizes of verification key and proofs for different instantiations of the admissible hash function, we refer to Remark 2 subsequent to our construction.

Definition 8. *Let* $\mathsf{AHF}\colon \{0,1\}^L \to \Sigma^n$ *together with* $\mathsf{AHF.Part}$ *be an admissible hash function and* ℓ *be an upper bound on the set* $I \subseteq \{1,\dots,n\}$ *output by* $\mathsf{AHF.Part}(1^\lambda, Q)$ *(for* Q *polynomial in* λ*). Let* $\mathsf{BG.Gen}$ *be a certified bilinear group generator and let* $\mathcal{G} \leftarrow_R \mathsf{BG.Gen}(1^\lambda)$*. We define a verifiable vector hash function* $\mathsf{VVHF} := (\mathsf{VVHF.Gen}, \mathsf{VVHF.Eval}, \mathsf{VVHF.Vfy})$ *as follows.*

- $\mathsf{VVHF.Gen}$ *is a probabilistic algorithm that on input of group parameters* $\mathcal{G}$ *samples matrices* $\mathbf{M}_{i,j,k} \leftarrow_R \mathbb{Z}_p^{d\times d}$ *for all* $i \in \{1,\dots,\ell\}$*,* $j \in \{1,\dots,n\}$ *and* $k \in \Sigma$ *uniformly at random. Further the algorithm samples a vector* $\mathbf{u} \leftarrow_R \mathbb{Z}_p^d \backslash \{0\}$ *and outputs the evaluation key*

$$ek := \left((\mathbf{M}_{i,j,k})_{i\in\{1,\dots,\ell\},j\in\{1,\dots,n\},k\in\Sigma}, \mathbf{u}\right)$$

and the verification key

$$vk := \left([\mathbf{M}_{i,j,k}]_{i\in\{1,\dots,\ell\},j\in\{1,\dots,n\},k\in\Sigma}, [\mathbf{u}]\right).$$

- $\mathsf{VVHF.Eval}$ *is an algorithm that on input of an evaluation key* ek *and a preimage* $x \in \{0,1\}^L$ *first computes the admissible hash value* $X := \mathsf{AHF}(x) \in \Sigma^n$ *of* x *and further, for each* $\iota \in \{1,\dots,\ell\}$*, the vector*

$$\mathbf{v}_\iota := \left(\prod_{i=1}^{\iota}\sum_{j=1}^{n} \mathbf{M}_{i,j,X_j}\right)^\top \mathbf{u}. \tag{1}$$

Finally, VVHF.Eval *outputs the image*

$$[\mathbf{v}] \coloneqq [\mathbf{v}_\ell]$$

and the proof

$$\pi \coloneqq [\mathbf{v}_1, \ldots, \mathbf{v}_{\ell-1}]\,.$$

– VVHF.Vfy *is an algorithm that on input of a verification key vk, a preimage x with image $\mathsf{AHF}(x) = X = (X_1, \ldots, X_n) \in \Sigma^n$, an image $[\mathbf{v}] = [\mathbf{v}_\ell]$ and a proof $\pi = [\mathbf{v}_1, \ldots, \mathbf{v}_{\ell-1}]$ checks whether for all $\iota \in \{1, \ldots, \ell\}$ and $[\mathbf{v}_0] \coloneqq [\mathbf{u}]$ it holds*

$$e([\mathbf{1}_d], [\mathbf{v}_\iota]) = e(\sum_{j=1}^{n} \left[\mathbf{M}_{\iota,j,X_j}\right]^\top, [\mathbf{v}_{\iota-1}]) \tag{2}$$

and returns 1 if and only if this is the case.

Remark 2. Recall that we consider inputs of arbitrary polynomial length L. The following numbers correspond to $d = 3$ (i.e. security based on the decision linear assumption).

The admissible hash function by Lysyanskaya [27] (see Lemma 1) has parameters $\ell \in \omega(\log L)$, $n \in \mathcal{O}(L)$ and $|\Sigma| = 2$. Therefore, instantiating our construction with this AHF and applying the generic transformation of [20] yields a verifiable random function with proofs of size $3(\ell+1) \in \omega(\log \lambda)$ and a verification key of size $18\ell n + 6 \in \omega(L \log \lambda)$ (in number of group elements).

Alternatively, the admissible hash function by Bitansky [7] (see Lemma 2) comes with parameters $\ell \in \omega(1)$ and $n, |\Sigma| \in \mathcal{O}\left(L^{1+\eta}\right)$ for an arbitrary constant $\eta > 0$. Employing this instantiation (together with the generic transformation of [20]) thus yields a verifiable random function with proofs consisting of $3(\ell+1) \in \omega(1)$ group elements and a verification key comprising $18\ell n+6 \in \omega(L^{2+2\eta})$ group elements.

Note that the additional $3 \cdot 2$ group elements in the proofs and the additional 3 group elements in the verification key stem from the generic transformation (see Theorem 1).

Lemma 3 (Correctness and unique provability). *The tuple* VVHF *given in Definition 8 is a verifiable vector hash function.*

Proof. **Correctness.** Let ek and vk as in Definition 8. Let $x \in \{0,1\}^L$ and $X := \mathsf{AHF}(x)$. Let $[\mathbf{v}] = [\mathbf{v}_\ell]$ be an image and $\pi = [\mathbf{v}_1, \ldots, \mathbf{v}_{\ell-1}]$ be a corresponding proof computed by the algorithm VVHF.Eval on input ek, x. Let $[\mathbf{v}_0] \coloneqq [\mathbf{u}]$. For all $\iota \in \{1, \ldots, \ell\}$ we have

$$\mathbf{v}_\iota = \left(\prod_{i=1}^{\iota}\sum_{j=1}^{n} \mathbf{M}_{i,j,X_j}\right)^\top \mathbf{u} = \sum_{j=1}^{n} \mathbf{M}_{\iota,j,X_j}^\top \cdot \underbrace{\left(\prod_{i=1}^{\iota-1}\sum_{j=1}^{n} \mathbf{M}_{i,j,X_j}\right)^\top \mathbf{u}}_{=\mathbf{v}_{\iota-1}}$$

by (1), and thus (2) follows.

Unique provability. For each $\iota \in \{1, \ldots, \ell\}$ there exists exactly one $\mathbf{v}_\iota$ satisfying (2) respective to the verification key and $\mathbf{v}_0, \ldots, \mathbf{v}_{\iota-1}$. As the group described by $\mathcal{G}$ satisfies recognition and unique representation of group elements, unique provability follows.

Lemma 4 (Adaptive Programmability). *If the d-rank assumption holds relative to* $\mathsf{BG.Gen}$*, then the verifiable vector hash function* $\mathsf{VVHF} = (\mathsf{VVHF.Gen}, \mathsf{VVHF.Eval}, \mathsf{VVHF.Vfy})$ *given in Definition 8 satisfies adaptive programmability. More precisely, there exist a tuple of trapdoor algorithms* $(\mathsf{VVHF.TrapGen}, \mathsf{VVHF.TrapEval})$*, such that the following hold.*

Correctness. $(\mathsf{VVHF.TrapGen}, \mathsf{VVHF.TrapEval})$ *satisfies correctness respective to* $\mathsf{VVHF.Vfy}$.

Indistinguishability. *For any Q polynomially bounded in λ and any PPT adversaries $\mathcal{A}$ with running time $t_\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ with running time $t_\mathcal{B} \approx t_\mathcal{A}$ such that*

$$\mathsf{Adv}^{\mathsf{vhf-ind}}_{\mathsf{VVHF},(\mathsf{VVHF.TrapGen},\mathsf{VVHF.TrapEval}),\mathcal{A},Q}(\lambda) \leq \ell \cdot \mathsf{Adv}^{d-\mathsf{rank}}_{\mathcal{G},\mathcal{B}}(\lambda) + \mathcal{O}\left(\frac{\ell n \sigma}{p}\right).$$

Well-distributed outputs. *For any polynomial Q in λ and for any $x^{(1)}, \ldots, x^{(Q)}, x^\star \in \{0,1\}^L$ with $x^\star \notin \{x^{(1)}, \ldots, x^{(Q)}\}$ it holds*

$$\begin{aligned}\Pr^{\mathsf{well-distr}}_{(\mathsf{VVHF.TrapGen},\mathsf{VVHF.TrapEval}),\lambda,Q}(\{x^{(\nu)}\}_\nu, x^\star) &= \Pr^{\mathsf{part}}_{\mathsf{AHF},\mathsf{AHF.Part},\lambda,Q}(\{x^{(\nu)}\}_\nu, x^\star) \\ &\geq \frac{1}{\mathrm{poly}(\lambda)}.\end{aligned}$$

Proof. We define the following tuple of algorithms:

– $\mathsf{VVHF.TrapGen}$ is a probabilistic algorithm that on input of the group parameters $\mathcal{G}$, a parameter Q (which is required to be polynomially bounded in λ) and a matrix $[\mathbf{B}] \in \mathbb{Z}_p^{d\times d}$ proceeds as follows.
First, $\mathsf{VVHF.TrapGen}$ samples for each $i \in \{0, \ldots, \ell-1\}$ a subspace $\mathcal{U}_i$ of dimension $d-1$ independently and uniformly at random. $\mathcal{U}_\ell$ is defined to be the subspace spanned by the first $d-1$ unit vectors. Further, the algorithm chooses $\mathbf{u} \leftarrow_R \mathbb{Z}_p^d \backslash \mathcal{U}_0$.
Next, $\mathsf{VVHF.TrapGen}$ runs $\mathsf{AHF.Part}$ to obtain $(Y, I) \leftarrow \mathsf{AHF.Part}(1^\lambda, Q)$ and samples $\mathbf{T}_{i,j,k} \in \mathbb{Z}_p^{d\times d}$ for each $i \in \{1, \ldots, \ell\}, j \in \{1, \ldots, n\}, k \in \Sigma$ as follows.
- Let j_i be the i-th value in I. Then, for each $i \in \{1, \ldots, |I|\}$, the algorithm samples a matrix $\mathbf{T}_{i,j_i,Y_{j_i}} \in \mathbb{Z}_p^{d\times d}$ uniformly *of rank d* subject to

$$\mathbf{T}_{i,j_i,Y_{j_i}}^\top \cdot \mathcal{U}_{i-1} = \mathcal{U}_i.$$

- Further, for each $i \in \{|I|+1, \ldots, \ell\}$ and all $k \in \Sigma$, again the algorithm samples a matrix $\mathbf{T}_{i,1,k} \in \mathbb{Z}_p^{d\times d}$ uniformly *of rank d* subject to

$$\mathbf{T}_{i,1,k}^\top \cdot \mathcal{U}_{i-1} = \mathcal{U}_i.$$

- For the remaining $i \in \{1, \ldots, \ell\}$, $j \in \{1, \ldots, n\}$ and $k \in \Sigma$ the algorithm samples $\mathbf{T}_{i,j,k} \in \mathbb{Z}_p^{d \times d}$ uniformly *of rank* $d-1$ subject to

$$\mathbf{T}_{i,j,k}^\top \cdot \mathbb{Z}_p^d = \mathcal{U}_i.$$

Finally, the algorithm sets

$$[\mathbf{M}_{i,j,k}] \coloneqq \begin{cases} [\mathbf{T}_{i,j,k}] & \text{if } i \in \{1, \ldots, \ell-1\} \\ \mathbf{T}_{i,j,k} \cdot [\mathbf{B}]^\top & \text{if } i = \ell \end{cases}$$

for all $i \in \{1, \ldots, \ell\}, j \in \{1, \ldots, n\}$ and $k \in \Sigma$ and outputs the trapdoor

$$td \coloneqq \left((\mathbf{T}_{i,j,k})_{i \in \{1,\ldots,\ell\}, j \in \{1,\ldots,n\}, k \in \Sigma}, \mathbf{u}, [\mathbf{B}] \right)$$

and the verification key

$$vk \coloneqq \left([\mathbf{M}_{i,j,k}]_{i \in \{1,\ldots,\ell\}, j \in \{1,\ldots,n\}, k \in \Sigma}, [\mathbf{u}] \right).$$

– $\mathsf{VVHF.TrapEval}$ is a probabilistic algorithm that on input of a trapdoor td and a preimage $x \in \{0,1\}^L$ first computes the admissible hash value $X \coloneqq \mathsf{AHF}(x) \in \Sigma^n$ of x and further, for each $\iota \in \{1, \ldots, \ell\}$, the vector

$$\mathbf{v}_\iota \coloneqq \left(\prod_{i=1}^{\iota} \sum_{j=1}^{n} \mathbf{T}_{i,j,X_j} \right)^\top \mathbf{u}. \tag{3}$$

Finally, $\mathsf{VVHF.TrapEval}$ outputs the vector

$$\mathbf{c} \coloneqq \mathbf{v}_\ell$$

and the proof

$$\pi \coloneqq [\mathbf{v}_1, \ldots, \mathbf{v}_{\ell-1}].$$

In the following we prove that the tuple $(\mathsf{VVHF.TrapGen}, \mathsf{VVHF.TrapEval})$ meets the required properties.

Correctness. Let (td, vk) be the output of $\mathsf{VVHF.TrapGen}$ on input $(\mathcal{G}, Q, [\mathbf{B}])$. Let $x \in \{0,1\}^L$ be an input value and let $X \coloneqq \mathsf{AHF}(x)$ its encoding. Let $\mathbf{c} = \mathbf{v}_\ell$ and $\pi = [\mathbf{v}_1, \ldots, \mathbf{v}_{\ell-1}]$ be provided by $\mathsf{VVHF.TrapEval}$ on input $[\mathbf{B}]$. Then, for $\iota \in \{1, \ldots, \ell-1\}$ equation (3) yields

$$\mathbf{v}_\iota = \left(\prod_{i=1}^{\iota} \sum_{j=1}^{n} \mathbf{T}_{i,j,X_j} \right)^\top \mathbf{u} = \left(\prod_{i=1}^{\iota} \sum_{j=1}^{n} \mathbf{M}_{i,j,X_j} \right)^\top \mathbf{u}$$

and for $\iota = \ell$ we have

$$\mathbf{B} \cdot \mathbf{c} = \mathbf{B} \cdot \left(\prod_{i=1}^{\ell} \sum_{j=1}^{n} \mathbf{T}_{i,j,X_j} \right)^\top \mathbf{u} = \sum_{j=1}^{n} \underbrace{\mathbf{B} \cdot \mathbf{T}_{\ell,j,X_j}^\top}_{=\mathbf{M}_{\ell,j,X_j}^\top} \cdot \left(\prod_{i=1}^{\ell-1} \sum_{j=1}^{n} \underbrace{\mathbf{T}_{i,j,X_j}}_{=\mathbf{M}_{i,j,X_j}} \right)^\top \mathbf{u}.$$

Thus, correctness follows from the correctness of $\mathsf{VVHF.Eval}$.

Indistinguishability. The proof strategy follows the one of Lemma 4.6 in [22] closely. For $\kappa \in \{1, \ldots, \ell\}$ we define the following algorithm $\mathsf{VVHF.TrapGen}^{(\kappa)}$.

- First, the algorithm $\mathsf{VVHF.TrapGen}^{(\kappa)}$ samples $(d-1)$-dimensional subspaces $\mathcal{U}_0, \ldots, \mathcal{U}_\kappa \subseteq \mathbb{Z}_p^d$ and a vector $\mathbf{u} \leftarrow_R \mathbb{Z}_p^d \setminus \mathcal{U}_0$ uniformly at random.
- Second, the algorithm calls $\mathsf{AHF.Part}$ on input $(1^\lambda, Q)$ to obtain Y and I.
- For all $i \leq \kappa$, the algorithm chooses $\mathbf{T}_{i,j,k}$ according to $\mathsf{VVHF.TrapGen}$ and sets $\mathbf{M}_{i,j,k} := \mathbf{T}_{i,j,k}$ for all $j \in \{1, \ldots, n\}, k \in \Sigma$.
- For all $i > \kappa$, the algorithm chooses $\mathbf{M}_{i,j,k} \leftarrow_R \mathbb{Z}_p^{d \times d}$ for all $j \in \{1, \ldots, n\}$, $k \in \Sigma$.

We define the following series of games. We consider the indistinguishability experiment of Definition 4. In game $\mathbf{G}_0$ the verification key is generated by the $\mathsf{VVHF.Gen}$ algorithm and in game $\mathbf{G}_2$ the verification key is generated by the $\mathsf{VVHF.TrapGen}$ algorithm. We prove the indistinguishability of $\mathbf{G}_0$ and $\mathbf{G}_2$ by a series of games. We define game $\mathbf{G}_{1.\kappa}$ to be the game where $\mathsf{VVHF.Gen}$ is replaced by $\mathsf{VVHF.TrapGen}^{(\kappa)}$, respectively. By ε_κ we denote the probability that an adversary $\mathcal{A}$ outputs 1 in $\mathbf{G}_i$, that is $\varepsilon_\kappa := \Pr[\mathcal{A} \text{ outputs } 1]$. It remains to show that for every PPT adversary $\mathcal{A}$, $|\varepsilon_0 - \varepsilon_2|$ is negligible.

Transition $\mathbf{G}_0 \rightsquigarrow \mathbf{G}_{1.0}$: In game $\mathbf{G}_{1.0}$ the vector $\mathbf{u}$ is chosen uniformly at random from $\mathbb{Z}_p^d \setminus \mathcal{U}_0$ for a random subspace $\mathcal{U}_0$ instead of uniformly random from $\mathbb{Z}_p^d \setminus \{0\}$. As the view of $\mathcal{A}$ is independent of $\mathcal{U}_0$ we obtain $\varepsilon_1 = \varepsilon_0$.

Transition $\mathbf{G}_{1.\kappa-1} \rightsquigarrow \mathbf{G}_{1.\kappa}$: Given an adversary $\mathcal{A}$, we construct an adversary $\mathcal{B}$ on the d-rank problem as follows. Let $[\mathbf{A}]$ be the input to $\mathcal{B}$. Then $\mathcal{B}$ sets up the verification key as follows

- First, $\mathcal{B}$ samples $(d-1)$-dimensional subspaces $\mathcal{U}_0, \ldots, \mathcal{U}_{\kappa-1} \subseteq \mathbb{Z}_p^d$ and a vector $\mathbf{u} \leftarrow_R \mathbb{Z}_p^d \setminus \mathcal{U}_0$ uniformly at random.
- Next, the algorithm calls $\mathsf{AHF.Part}$ on input $(1^\lambda, Q)$ to obtain Y and I.
- For all $i \leq \kappa - 1, j \in \{1, \ldots, n\}, k \in \Sigma$, $\mathcal{B}$ chooses $\mathbf{T}_{i,j,k}$ according to $\mathsf{VVHF.TrapGen}$ and sets $\mathbf{M}_{i,j,k} := \mathbf{T}_{i,j,k}$.
- For $i = \kappa, j \in \{1, \ldots, n\}, k \in \Sigma$, the adversary proceeds as follows. Whenever $\mathsf{VVHF.TrapGen}$ would choose a matrix $\mathbf{T}_{\kappa,j,k}$ of rank $d-1$, $\mathcal{B}$ chooses a fresh $\mathbf{R}_{j,k} \leftarrow_R \mathbb{Z}_p^{d \times d}$ and sets $[\mathbf{M}_{\kappa,j,k}] := [\mathbf{T}_{\kappa,j,k}] := \mathbf{R}_{j,k} \cdot [\mathbf{A}]^\top$. Whenever $\mathsf{VVHF.TrapGen}$ would choose a matrix $\mathbf{T}_{\kappa,j,k}$ of rank d, $\mathcal{B}$ chooses a fresh basis $\{\mathbf{c}_1^{j,k}, \ldots, \mathbf{c}_{d-1}^{j,k}\}$ of $\mathcal{U}_{\kappa-1}$ (this is efficiently computable as $\mathcal{B}$ chooses $\mathcal{U}_{\kappa-1}$ itself) and further $d-1$ vectors $[\mathbf{d}_1^{j,k}], \ldots, [\mathbf{d}_{d-1}^{j,k}]$ in the image of $[\mathbf{A}]$. Further, $\mathcal{B}$ chooses $\mathbf{c}_d^{j,k}, \mathbf{d}_d^{j,k} \leftarrow_R \mathbb{Z}_p^d$, such that $\mathbf{C}_{j,k} := \left(\mathbf{c}_1^{j,k} | \cdots | \mathbf{c}_d^{j,k}\right)$ is invertible. Finally, $\mathcal{B}$ sets $[\mathbf{D}_{j,k}] := \left[\mathbf{d}_1^{j,k} | \cdots | \mathbf{d}_d^{j,k}\right]$ and $[\mathbf{M}_{\kappa,j,k}] := [\mathbf{T}_{\kappa,j,k}] := ([\mathbf{D}_{j,k}] \cdot \mathbf{C}_{j,k}^{-1})^\top$.
- For all $i > \kappa$, $\mathcal{B}$ chooses $\mathbf{M}_{i,j,k} \leftarrow_R \mathbb{Z}_p^{d \times d}$ for all $j \in \{1, \ldots, n\}, k \in \Sigma$.

Now, $\mathcal{B}$ forwards the verification key to $\mathcal{A}$. To answer evaluation queries, for $\iota \geq \kappa$ the adversary $\mathcal{B}$ can compute $[\mathbf{v}_\iota]$ as

$$[\mathbf{v}_\iota] = \left(\prod_{i=\iota}^{\kappa+1} \sum_{j=1}^{n} \mathbf{M}_{i,j,\mathsf{AHF}(x)_j}^\top [\mathbf{M}_{\kappa,j,\mathsf{AHF}(x)_j}]^\top \prod_{i=\kappa-1}^{1} \sum_{j=1}^{n} \mathbf{M}_{i,j,\mathsf{AHF}(x)_j}^\top \right) \mathbf{u}.$$

(Note that the factors are multiplied in reverse order, as we moved the transpose into the product in the above equation.)
It remains to prove that if $\mathbf{A}$ has rank $d-1$, then $\mathcal{B}$ simulates game $\mathbf{G}_{\kappa-1}$ and $\mathbf{G}_\kappa$ otherwise. We first consider the case that $\mathbf{A}$ has rank $d-1$. Let $\mathcal{U}_\kappa := [\mathbf{A}] \cdot \mathbb{Z}_p^d$ be the $d-1$-dimensional image of $[\mathbf{A}]$. Then the following holds

- For all $j \in \{1,\dots,n\}, k \in \Sigma$ the matrix $\mathbf{R}_{j,k} \cdot [\mathbf{A}]^\top$ is of rank $d-1$ with $([\mathbf{A}] \cdot \mathbf{R}_{j,k}^\top) \cdot \mathbb{Z}_p^d = [\mathbf{A}] \cdot \mathbb{Z}_p^d = \mathcal{U}_\kappa$. Further, note that for every j,k, $\mathbf{T}_{\kappa,j,k} := \mathbf{R}_{j,k} \cdot [\mathbf{A}]^\top$ is distributed independently and uniformly at random conditioned on $\mathbf{T}_{\kappa,j,k} \cdot \mathbb{Z}_p^d = \mathcal{U}_\kappa$.
- For all $j \in \{1,\dots,n\}, k \in \Sigma$ the vectors $[\mathbf{d}_1^{j,k}], \dots, [\mathbf{d}_{d-1}^{j,k}]$ form a basis of $\mathcal{U}_\kappa$ and further $\mathbf{D}_{j,k}$ is full rank with overwhelming probability[2]. In this case $([\mathbf{D}_{j,k}] \cdot \mathbf{C}_{j,k}^{-1})^\top$ is full rank with

$$([\mathbf{D}_{j,k}] \cdot \mathbf{C}_{j,k}^{-1}) \cdot \mathcal{U}_{\kappa-1} = [\mathbf{D}_{j,k}] \cdot \underbrace{(\mathbf{C}_{j,k}^{-1} \cdot \mathcal{U}_{\kappa-1})}_{=\{\mathbf{z} \in \mathbb{Z}_p^d \mid z_d = 0\}} = \mathcal{U}_\kappa.$$

  Again, note that for every j,k, $\mathbf{T}_{\kappa,j,k} := ([\mathbf{D}_{j,k}] \cdot \mathbf{C}_{j,k}^{-1})^\top$ is distributed independently and uniformly at random conditioned on $\mathbf{T}_{\kappa,j,k} \cdot \mathcal{U}_{\kappa-1} = \mathcal{U}_\kappa$.

We now assume that $\mathbf{A}$ is full rank. Due to the invertibility of $\mathbf{A}$, for all $j \in \{1,\dots,n\}, k \in \Sigma$ the matrix $\mathbf{R}_{j,k} \cdot [\mathbf{A}]^\top$ is distributed uniformly at random over $\mathbb{G}^{d \times d}$. As further $[\mathbf{D}_{j,k}]$ is distributed uniformly at random over $\mathbb{G}^{d \times d}$, the same holds for $([\mathbf{D}_{j,k}] \cdot \mathbf{C}_{j,k}^{-1})^\top$.
Finally, on input b from $\mathcal{A}$, $\mathcal{B}$ outputs **"rank $d-1$"** if $b = 0$ and **"rank d"** otherwise. Altogether, we obtain

$$|\varepsilon_{i.\kappa} - \varepsilon_{i.\kappa-1}| \leq \mathsf{Adv}^{d-\mathsf{rank}}_{\mathcal{G},\mathcal{B}}(\lambda) + \mathcal{O}(n\sigma/p).$$

Transition $\mathbf{G}_{1.\ell} \rightsquigarrow \mathbf{G}_2$: In game $\mathbf{G}_2$ the subspace $\mathcal{U}_\ell$ is the subspace spanned by the first $d-1$ unit vectors (instead of chosen uniformly at random). Further, $\mathbf{M}_{\ell,j,k}$ is defined to equal $\mathbf{T}_{\ell,j,k} \cdot [\mathbf{B}]^\top$. Recall that $\mathbf{B} \leftarrow_R \mathbb{Z}_p^{d \times d}$ is chosen uniformly at random from all invertible matrices. Now, in both games $\mathbf{G}_{1.\ell}$ and game $\mathbf{G}_2$, $\mathbf{M}_{\ell,j,k}^\top$ maps $\mathcal{U}_{\ell-1}$ to a uniform $d-1$ dimensional subspace (namely, in game $\mathbf{G}_2$ to the space spanned by the first $d-1$ column vectors of $\mathbf{B}$). We have thus

$$\varepsilon_2 = \varepsilon_{1.\ell}.$$

Finally, we obtain

$$\begin{aligned}\mathsf{Adv}^{\mathsf{vhf-ind}}_{\mathsf{VVHF},(\mathsf{VVHF.TrapGen},\mathsf{VVHF.TrapEval}),\mathcal{A},Q}(\lambda) &= |\varepsilon_2 - \varepsilon_0| \\ &= |\varepsilon_{1.\ell} - \varepsilon_{1.0}|\end{aligned}$$

[2] More precisely, with probability at least $1 - (d-1)/p - 1/p = 1 - d/p$.

$$\leq \sum_{\kappa=1}^{\ell} |\varepsilon_{1.\kappa} - \varepsilon_{1.\kappa-1}|$$
$$\leq \ell \cdot \mathsf{Adv}^{d-\mathsf{rank}}_{\mathcal{G},\mathcal{B}}(\lambda) + \mathcal{O}(\ell n \sigma / p).$$

Well-distributed outputs. Let $x^{(1)}, \ldots, x^{(Q)}, x^\star \in \{0,1\}^L$ be arbitrary with $x \notin \{x^{(1)}, \ldots, x^{(Q)}\}$. Recall that by Definition 7 choosing Y and I partitions the preimage space into sets

$$\mathcal{Y} := \left\{x \in \{0,1\}^L \mid \mathsf{AHF}(x)_j = Y_j \text{ for all } j \in I\right\} \text{ and } \mathcal{Z} := \{0,1\}^L \setminus \mathcal{Y}.$$

We hope that we have $x^\star \in \mathcal{Y}$ and $x^{(\nu)} \in \mathcal{Z}$ for all $\nu \in \{1, \ldots, Q\}$. As AHF is an admissible hash function, this is the case at least with probability

$$\Pr^{\mathsf{part}}_{\mathsf{AHF},\mathsf{AHF.Part},\lambda,Q}(\{x^{(i)}\}_i, x^\star) \geq \frac{1}{\mathrm{poly}(\lambda)}.$$

Let $\mathbf{c}^x$ the output vector of $\mathsf{VVHF.TrapEval}$ on input $x \in \{0,1\}^L$. To prove well-distributed outputs by previous considerations it suffices to show

$$c^x_d = 0 \Leftrightarrow x \in \mathcal{Z}.$$

Note that by construction of the subspace $\mathcal{U}_\ell$ we have $c^x_d = 0 \Leftrightarrow \mathbf{c}^x \in \mathcal{U}_\ell$. For all $x \in \{0,1\}^L$, $\iota \in \{1, \ldots, \ell\}$ let

$$\mathbf{v}^x_\iota := \left(\prod_{i=1}^{\iota} \sum_{j=1}^{n} \mathbf{T}_{i,j,\mathsf{AHF}(x)_j}\right)^\top \mathbf{u}. \tag{4}$$

Recall that $\mathbf{u} \in \mathbb{Z}_p^d \backslash \mathcal{U}_0$ and that for all $\mathbf{T}_{i,j,k}$ we have $\mathbf{T}^\top_{i,j,k} \cdot \mathcal{U}_{\iota-1} = \mathcal{U}_\iota$.Therefore, it holds $\mathbf{v}^x_\iota \in \mathcal{U}_\iota$ if

$$\mathbf{v}^x_{\iota-1} \in \mathcal{U}_{\iota-1} \text{ OR } \sum_{j=1}^{n} \mathbf{T}^\top_{\iota,j,\mathsf{AHF}(x)_j} \cdot (\mathbb{Z}_p^d \backslash \mathcal{U}_{\iota-1}) \subseteq \mathcal{U}_\iota.$$

In order to prove the claim it thus suffices to show
i.) $x \in \mathcal{Z} \Rightarrow \exists \iota \in \{1, \ldots, \ell\}\colon \sum_{j=1}^{n} \mathbf{T}^\top_{\iota,j,\mathsf{AHF}(x)_j} \cdot (\mathbb{Z}_p^d \backslash \mathcal{U}_{\iota-1}) \subseteq \mathcal{U}_\iota$ and
ii.) $x \in \mathcal{Y} \Rightarrow \forall \iota \in \{1, \ldots, \ell\}\colon \sum_{j=1}^{n} \mathbf{T}^\top_{\iota,j,\mathsf{AHF}(x)_j} \cdot (\mathbb{Z}_p^d \backslash \mathcal{U}_{\iota-1}) \subseteq \mathbb{Z}_p^d \backslash \mathcal{U}_\iota$.
Let j_ι be the ι-th index in I. For all $x \in \mathcal{Z}$ there exist a $\iota \in \{1, \ldots, |I|\}$ with $\mathsf{AHF}(x)_{j_\iota} \neq Y_{j_\iota}$. By construction for this ι we have $\mathbf{T}^\top_{\iota,j,\mathsf{AHF}(x)_j} \cdot \mathbb{Z}_p^d = \mathcal{U}_\iota$ for all $j \in \{1, \ldots, n\}$. This implies in particular $\sum_{j=1}^{n} \mathbf{T}^\top_{\iota,j,\mathsf{AHF}(x)_j} \cdot (\mathbb{Z}_p^d \backslash \mathcal{U}_{\iota-1}) \subseteq \mathcal{U}_\iota$ as required. We thus have $x \in \mathcal{Z} \implies c^x_d = 0$.
For all $x \in \mathcal{Y}$ and for all $\iota \in \{1, \ldots, |I|\}$ it holds $\mathsf{AHF}(x)_{j_\iota} = Y_{j_\iota}$. By construction we have $\mathbf{T}^\top_{\iota,j_\iota,Y_{j_\iota}} \cdot (\mathbb{Z}_p^d \backslash \mathcal{U}_{\iota-1}) = \mathbb{Z}_p^d \backslash \mathcal{U}_\iota$. For all $\iota \in \{1, \ldots, |I|\}$, $\mathbf{u}_{\iota-1} \in \mathbb{Z}_p^d \backslash \mathcal{U}_{\iota-1}$ we thus have

$$\sum_{j=1}^{n} \mathbf{T}^\top_{\iota,j,\mathsf{AHF}(x)_j} \cdot \mathbf{u}_{\iota-1} = \underbrace{\mathbf{T}^\top_{\iota,j_\iota,\mathsf{AHF}(x)_{j_\iota}} \cdot \mathbf{u}_{\iota-1}}_{\in \mathbb{Z}_p^d \backslash \mathcal{U}_\iota} + \underbrace{\sum_{j \neq j_\iota} \mathbf{T}^\top_{\iota,j,\mathsf{AHF}(x)_j} \cdot \mathbf{u}_{\iota-1}}_{\in \mathcal{U}_\iota} \in \mathbb{Z}_p^d \backslash \mathcal{U}_\iota.$$

Further, for all $\iota > |I|$ we have $\mathbf{T}_{\iota,j,k}$ uniform of rank d (subject to $\mathbf{T}_{\iota,j,k}^\top \cdot \mathcal{U}_{\iota-1} = \mathcal{U}_\iota$) if and only if $j = 1$. For all $\iota > |I|$, $\mathbf{u}_{\iota-1} \in \mathbb{Z}_p^d \backslash \mathcal{U}_{\iota-1}$ it thus holds

$$\sum_{j=1}^{n} \mathbf{T}_{\iota,j,\mathsf{AHF}(x)_j}^\top \cdot \mathbf{u}_{\iota-1} = \underbrace{\mathbf{T}_{\iota,1,\mathsf{AHF}(x)_1}^\top \cdot \mathbf{u}_{\iota-1}}_{\in \mathbb{Z}_p^d \backslash \mathcal{U}_\iota} + \underbrace{\sum_{j=2}^{n} \mathbf{T}_{\iota,j,\mathsf{AHF}(x)_j}^\top \cdot \mathbf{u}_{\iota-1}}_{\in \mathcal{U}_\iota} \in \mathbb{Z}_p^d \backslash \mathcal{U}_\iota.$$

Altogether, we obtain $x \in \mathcal{Y} \implies c_d^x \neq 0$.

Acknowledgments. I would like to thank the anonymous reviewers of TCC 2018 and PKC 2019 for their helpful comments. Further, I would like to thank my advisor Dennis Hofheinz for his support and helpful feedback.

References

[1] Michel Abdalla, Dario Catalano, and Dario Fiore. "Verifiable Random Functions from Identity-Based Key Encapsulation". In: *EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. LNCS. Springer, Heidelberg, Apr. 2009, pp. 554–571. DOI: 10.1007/978-3-642-01001-9_32.

[2] Michel Abdalla, Dario Catalano, and Dario Fiore. "Verifiable Random Functions: Relations to Identity-Based Key Encapsulation and New Constructions". In: *Journal of Cryptology* 27.3 (July 2014), pp. 544–593. DOI: 10.1007/s00145-013-9153-x.

[3] Michel Abdalla, Dario Fiore, and Vadim Lyubashevsky. "From Selective to Full Security: Semi-generic Transformations in the Standard Model". In: *PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Vol. 7293. LNCS. Springer, Heidelberg, May 2012, pp. 316–333. DOI: 10.1007/978-3-642-30057-8_19.

[4] Man Ho Au, Willy Susilo, and Yi Mu. "Practical Compact E-Cash". In: *ACISP 07*. Ed. by Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson. Vol. 4586. LNCS. Springer, Heidelberg, July 2007, pp. 431–445.

[5] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. *A note on VRFs from Verifiable Functional Encryption*. Cryptology ePrint Archive, Report 2017/051. http://eprint.iacr.org/2017/051. 2017.

[6] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. "Compact E-Cash and Simulatable VRFs Revisited". In: *PAIRING 2009*. Ed. by Hovav Shacham and Brent Waters. Vol. 5671. LNCS. Springer, Heidelberg, Aug. 2009, pp. 114–131. DOI: 10.1007/978-3-642-03298-1_9.

[7] Nir Bitansky. "Verifiable Random Functions from Non-interactive Witness-Indistinguishable Proofs". In: *TCC 2017, Part II*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10678. LNCS. Springer, Heidelberg, Nov. 2017, pp. 567–594. DOI: 10.1007/978-3-319-70503-3_19.

[8] Dan Boneh and Xavier Boyen. "Secure Identity Based Encryption Without Random Oracles". In: *CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. LNCS. Springer, Heidelberg, Aug. 2004, pp. 443–459. DOI: 10.1007/978-3-540-28628-8_27.

[9] Dan Boneh, Hart William Montgomery, and Ananth Raghunathan. "Algebraic pseudorandom functions with improved efficiency from the augmented cascade". In: *ACM CCS 10*. Ed. by Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov. ACM Press, Oct. 2010, pp. 131–140. DOI: 10.1145/1866307.1866323.

[10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. "Bonsai Trees, or How to Delegate a Lattice Basis". In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, 2010, pp. 523–552. DOI: 10.1007/978-3-642-13190-5_27.

[11] Yevgeniy Dodis. "Efficient Construction of (Distributed) Verifiable Random Functions". In: *PKC 2003*. Ed. by Yvo Desmedt. Vol. 2567. LNCS. Springer, Heidelberg, Jan. 2003, pp. 1–17. DOI: 10.1007/3-540-36288-6_1.

[12] Yevgeniy Dodis and Aleksandr Yampolskiy. "A Verifiable Random Function with Short Proofs and Keys". In: *PKC 2005*. Ed. by Serge Vaudenay. Vol. 3386. LNCS. Springer, Heidelberg, Jan. 2005, pp. 416–431. DOI: 10.1007/978-3-540-30580-4_28.

[13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. "An Algebraic Framework for Diffie-Hellman Assumptions". In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 129–147. DOI: 10.1007/978-3-642-40084-1_8.

[14] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. "Programmable Hash Functions in the Multilinear Setting". In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 513–530. DOI: 10.1007/978-3-642-40041-4_28.

[15] Oded Goldreich. "Computational Complexity: A Conceptual Perspective". In: *ACM Sigact News* 39.3 (2008), pp. 35–39.

[16] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to Construct Random Functions". In: *Journal of the ACM* 33.4 (Oct. 1986), pp. 792–807.

[17] Oded Goldreich and Yair Oren. "Definitions and Properties of Zero-Knowledge Proof Systems". In: *Journal of Cryptology* 7.1 (1994), pp. 1–32.

[18] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. *A Generic Approach to Constructing and Proving Verifiable Random Functions*. Cryptology ePrint Archive, Report 2017/021. http://eprint.iacr.org/2017/021. 2017.

[19] Jens Groth, Rafail Ostrovsky, and Amit Sahai. "New Techniques for Noninteractive Zero-Knowledge". In: *J. ACM* 59.3 (June 2012), 11:1–11:35. ISSN: 0004-5411. DOI: 10.1145/2220357.2220358. URL: http://doi.acm.org/10.1145/2220357.2220358.

[20] Dennis Hofheinz and Tibor Jager. "Verifiable Random Functions from Standard Assumptions". In: *TCC 2016-A, Part I*. Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9562. LNCS. Springer, Heidelberg, Jan. 2016, pp. 336–362. DOI: 10.1007/978-3-662-49096-9_14.

[21] Susan Hohenberger and Brent Waters. "Constructing Verifiable Random Functions with Large Input Spaces". In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, 2010, pp. 656–672. DOI: 10.1007/978-3-642-13190-5_33.

[22] Dennis Hofheinz; Tibor Jager. *Verifiable Random Functions from Standard Assumptions*. Cryptology ePrint Archive, Report 2015/1048. http://eprint.iacr.org/2015/1048. 2015.

[23] Tibor Jager. "Verifiable Random Functions from Weaker Assumptions". In: *TCC 2015, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 121–143. DOI: 10.1007/978-3-662-46497-7_5.

[24] Stanislaw Jarecki and Vitaly Shmatikov. "Handcuffing Big Brother: an Abuse-Resilient Transaction Escrow Scheme". In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 590–608. DOI: 10.1007/978-3-540-24676-3_35.

[25] Shuichi Katsumata. "On the Untapped Potential of Encoding Predicates by Arithmetic Circuits and Their Applications". In: *ASIACRYPT 2017, Part III*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10626. LNCS. Springer, Heidelberg, Dec. 2017, pp. 95–125. DOI: 10.1007/978-3-319-70700-6_4.

[26] Moses Liskov. "Updatable Zero-Knowledge Databases". In: *ASIACRYPT 2005*. Ed. by Bimal K. Roy. Vol. 3788. LNCS. Springer, Heidelberg, Dec. 2005, pp. 174–198. DOI: 10.1007/11593447_10.

[27] Anna Lysyanskaya. "Unique Signatures and Verifiable Random Functions from the DH-DDH Separation". In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 597–612. DOI: 10.1007/3-540-45708-9_38.

[28] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. "Verifiable Random Functions". In: *40th FOCS*. IEEE Computer Society Press, Oct. 1999, pp. 120–130. DOI: 10.1109/SFFCS.1999.814584.

[29] Silvio Micali and Leonid Reyzin. "Soundness in the Public-Key Model". In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001, pp. 542–565. DOI: 10.1007/3-540-44647-8_32.

[30] Silvio Micali and Ronald L. Rivest. "Micropayments Revisited". In: *CT-RSA 2002*. Ed. by Bart Preneel. Vol. 2271. LNCS. Springer, Heidelberg, Feb. 2002, pp. 149–163. DOI: 10.1007/3-540-45760-7_11.

[31] Moni Naor and Omer Reingold. "Number-theoretic constructions of efficient pseudo-random functions". In: *Journal of the ACM* 51.2 (2004), pp. 231–262.

[32] Irving S Reed and Gustave Solomon. "Polynomial codes over certain finite fields". In: *Journal of the society for industrial and applied mathematics* 8.2 (1960), pp. 300–304.

[33] Răzvan Roşie. *Adaptive-Secure VRFs with Shorter Keys from Static Assumptions*. Cryptology ePrint Archive, Report 2017/750. http://eprint.iacr.org/2017/750. 2017.

[34] Brent R. Waters. "Efficient Identity-Based Encryption Without Random Oracles". In: *EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 114–127. DOI: 10.1007/11426639_7.

[35] Shota Yamada. "Asymptotically Compact Adaptively Secure Lattice IBEs and Verifiable Random Functions via Generalized Partitioning Techniques". In: *CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. LNCS. Springer, Heidelberg, Aug. 2017, pp. 161–193. DOI: 10.1007/978-3-319-63697-9_6.