On the Real-World Instantiability of Admissible Hash Functions and Efficient Verifiable Random Functions

Tibor Jager*[1] and David Niehues**[2]

[1]Bergische Universität Wuppertal, Wuppertal, Germany,
tibor.jager@uni-wuppertal.de
[2]Paderborn University, Paderborn, Germany,
david.niehues@uni-paderborn.de

Abstract. *Verifiable random functions* (VRFs) are essentially digital signatures with additional properties, namely *verifiable uniqueness* and *pseudorandomness*, which make VRFs a useful tool, e.g., to prevent enumeration in DNSSEC Authenticated Denial of Existence and the CONIKS key management system, or in the random committee selection of the Algorand blockchain.
Most standard-model VRFs rely on *admissible hash functions* (AHFs) to achieve security against *adaptive* attacks in the standard model. Known AHF constructions are based on error-correcting codes, which yield *asymptotically* efficient constructions. However, previous works do not clarify how the code should be instantiated *concretely* in the real world. The *rate* and the *minimal distance* of the selected code have significant impact on the efficiency of the resulting cryptosystem, therefore it is unclear if and how the aforementioned constructions can be used in practice.
First, we explain inherent limitations of code-based AHFs. Concretely, we show that even if we were given codes that achieve the well-known Gilbert-Varshamov or McEliece-Rodemich-Rumsey-Welch bounds, existing AHF-based constructions of verifiable random functions (VRFs) can only be instantiated quite inefficiently. Then we introduce and construct *computational* AHFs (cAHFs). While classical AHFs are information-theoretic, and therefore work even in presence of computationally unbounded adversaries, cAHFs provide only security against computationally bounded adversaries. However, we show that cAHFs can be instantiated significantly more efficiently. Finally, we present a new VRF scheme using cAHFs and show that it is currently the most efficient verifiable random function with full adaptive security in the standard model.

Keywords: Admissible hash functions, verifiable random functions, error-correcting codes, provable security.

* Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823.

** Supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901/3).

1 Introduction

Verifiable random functions (VRFs), introduced by Micali, Rabin and Vadhan [37], are the public-key counterpart to Pseudorandom Functions (PRFs). The evaluation of a VRF $V_{\mathsf{sk}}(X)$ can only be privately computed using the secret key sk. The evaluation at X yields a pseudorandom value Y together with a non-interactive proof π. The verifier can use the public verification key vk and π to confirm that Y was correctly computed as $V_{\mathsf{sk}}(X)$. VRFs have recently found several interesting real-world applications. For instance, the Algorand blockchain uses a VRF to randomly select a committee for a Byzantine agreement [19]. Furthermore, VRFs can be used to prevent enumeration attacks against hash-based data structures, since the VRF can only be evaluated privately and the correctness of the computation can be publicly verified. This approach is used in the (currently inactive) IETF draft for DNSSEC Authenticated Denial of Existence [40,48] to prevent offline DNS-enumeration attacks. Yet another application domain of VRFs are key management systems. For instance, CONIKS, a modern transparent key management system, uses a VRF similarly to prevent leaking private data of users [36]. Due to these numerous practical applications, the IETF is currently standardizing VRFs [21]. These VRFs are efficient, but the accompanying security proofs rely on the random oracle heuristic [5], which can not be instantiated in general [13]. In this work, we consider efficient practical constructions of VRFs in the standard model, meaning without the random oracle heuristic.

Partitioning is a technique which is commonly used to prove security of cryptographic constructions, and the only known way to construct verifiable random functions, both in the standard model and in the random oracle model. Essentially, a partitioning proof divides a certain considered set, such as for instance the message space of a digital signature scheme, the domain of a verifiable random function, or the identity space of an identity-based encryption (IBE) scheme, into two subsets:

1. A *"controlled"* set, which contains all elements for which the simulator in the security proof is able to efficiently simulate, e.g., digital signatures, and
2. an *"uncontrolled"* set, which contains elements where the simulator is able to efficiently embed an instance of a computationally hard problem, such that an efficient adversary on the considered cryptographic construction can be turned into an efficient algorithm solving the hard problem.

Partitioning is often used in the *random oracle model* [5], where an idealized cryptographic hash function can be adaptively "programmed" in order to enable a successful simulation. For instance, the well-known security proofs of Full-Domain Hash signatures [6], BLS signatures [12], or Boneh-Franklin IBE [11] use this approach. Furthermore, partitioning is the only known way to prove security of unique signatures[1] or verifiable random functions (VRFs) [37] against *adaptive adversaries.*

Admissible hash functions (AHFs) are a generic and very useful tool to enable partitioning proofs in the *standard model*, that is, without random oracles. AHFs were formally introduced in [9,14,18], but had implicitly already been used by Lysyanskaya [33]. Essentially, the idea is that the AHF partitions the considered set "randomly" and invisibly to the adversary, such that with noticeable probability exactly the "right" elements fall into the "controlled" set. For instance, the "right" message may be exactly those messages for which the adversary queries a signature in the EUF-CMA security experiment. At the same time, exactly the "right" other elements fall into the "uncontrolled" set, e.g., the message for which the adversary produces a signature forgery. AHFs ensure that this holds with sufficiently high probability, even if the adversary chooses these values adaptively and possibly maliciously. AHFs are an ubiquitous tool in public-key cryptography, and have been used to realize numerous cryptographic primitives with strong adaptive security and without random oracles, such as unique signatures [33], verifiable random functions [7,23,26,33], different variants of identity-based encryption [1,9], Bonsai trees [14], programmable hash functions [18], and constrained PRFs [2].

For some primitives AHF-based partitioning proofs (and variants thereof, such as those defined in [7,29,50]) are still the only known way to achieve provable adaptive security with an efficient polynomial-time reduction in the standard model. For some other primitives, such as identity-based encryption [9], AHFs yielded the first constructions with adaptive security, and later more efficient constructions have been developed that apply other techniques that are specifically designed for a particular (class of) cryptosystem(s) and are not as generic as AHFs. We also view AHFs as an extremely useful generic tool that will most likely find applications to further advanced cryptographic constructions in the future.

Practical Instantiability of AHFs. Given the large number of cryptographic constructions based on AHFs, it is interesting and important to ask how AHF-based constructions can be instantiated in *practice*. Known constructions are based on different types of error-correcting codes with suitable minimal distance, which is required to be a *constant* fraction of the length of the code, in order to make the partitioning argument go through with noticeable success probability. There are many possible codes to choose from [20,44,47,51], which yield very different concrete instantiations with very different efficiency and security properties.

Most aforementioned works mention that one or another of these codes can be used to instantiate their AHFs in the *asymptotic* setting, but it is never clarified how their constructions can be instantiated *concretely*, by explaining how the underlying code and other cryptographic parameters must be chosen, taking

[1] That is, digital signatures where for any given (public key, message)-pair there exists only one unique string that is accepted as a signature by the verification algorithm.

into account the considered security parameter, deployment parameters such as the number of AHF evaluations by a realistic adversary, and the tightness of the security proof. The concrete choice of the code used to instantiate the AHF has a very significant impact on the efficiency of the resulting cryptosystem. Hence, while AHFs provide a powerful generic tool to achieve provable security *asymptotically*, it is completely unclear how efficiently they can be instantiated *concretely*.

Our contributions. Our main objective is to clarify how AHFs can be securely and efficiently instantiated concretely in practice, and to develop new techniques that enable a more efficient instantiation of AHF-based cryptosystems. To this end, we make the following contributions:

– We assess how AHFs can be instantiated with error-correcting codes (ECC). We show that while AHFs are theoretically sufficient to obtain polynomial-time constructions and security against polynomial-time adversaries in the asymptotic setting, they yield only extremely inefficient concrete instantiations. By applying bounds on ECCs from classical coding theory, we point out inherent limitations of concrete instantiations of the AHFs presented in prior work. Concretely, we show that even with codes that meet the Gilbert-Varshamov or McEliece-Rodemich-Rumsey-Welch (MRRW) bound, even optimized variants [29,50] of known verifiable random functions [26,33] have only very inefficient practical instantiations.
– Our first main novel technical contribution is the introduction of the notion of *computational* AHFs (cAHFs). Standard AHFs based on error-correcting codes are essentially an *information-theoretic* primitive, which works unconditionally and even for computationally unbounded adversaries, which of course is stronger than necessary for most applications. cAHFs therefore relax this requirement, in the sense that they are only required to partition the considered set successfully in the presence of a *computationally bounded* adversary. This will make it possible to overcome the aforementioned limitations of AHFs. We also give a concrete instantiation of cAHFs, based on the notion of *truncation collision resistant hash functions* from [27].
– Our second novel technical contribution is a new highly efficient verifiable random function (VRF), based on Jager's VRF [26]. We use it to showcase how cAHFs are applied in constructions and proofs. The new VRF is the currently most efficient verifiable random function with full adaptive security and exponential-sized input space, based on a non-interactive complexity assumption, in the standard model.

Classical balanced admissible hash functions, and therefore also the security proofs of VRFs based on those, which include [26,29,50], require reasonably close bounds on the number of VRF *evaluation queries* and advantage of an adversary. Our instantiation of cAHFs inherits from [27] that security proofs require knowledge of (sufficiently close bounds on) the *running time* and advantage of an adversary.

Related work. Boneh and Boyen [9] formally introduced AHFs to construct IBE without random oracles. *Balanced* AHFs were introduced in [26]. The balancedness makes it possible to apply AHF-based partitioning directly in security proofs considering "indistinguishability-based" security experiments, without requiring the *artificial abort* approach of Waters [49]. Balanced AHFs were used to construct verifiable random functions [7,23,26,50], IBE [50], constrained PRFs [3], and distributed PRFs [32]. We consider both standard and balanced AHFs in this work.

Several papers developed techniques to optimize schemes using AHFs. Yamada [50] and Katsumata [29] encode the information of the "controlled" set into shorter bit strings and employ the AHF on this shorter string. Recently, Kohl [31] applied a similar approach to the VRF construction of [23] to obtain a VRF with strong security properties and shorter proofs.

Most previous applications of AHFs consider a setting where a polynomially-bounded number of Q elements $X^{(1)}, \dots, X^{(Q)}$ must fall into the "controlled" set, while *one* "challenge" element X^* must fall into the "uncontrolled" set for the reduction in the security proof to be successful. This matches what is required for most common security experiments for primitives such as digital signatures, VRFs, IBE, and many others. Chen *et al.* [15] generalize this to AHFs that can handle more than one challenge element and ensure that $n > 1$ challenge elements $X^{(1)*}, \dots, X^{(n)*}$ fall into the "uncontrolled" set, and give a construction with $n = 2$.

AHFs are related to *programmable hash functions* (PHFs) [24,25], but are more general, in the sense that PHFs can generically be constructed from AHFs, but there exist cryptographic primitives, such as VRFs, for which only constructions based on AHFs are known to exist, but not on PHFs.

Changes since publication at SAC 2019. This paper is an extended and revised full version of a paper with the same title that appeared at SAC 2019 [28]. In comparison to the construction from the original publication [28], we give a new and improved VRF construction, which uses the same proof technique but reduces the size of the verification key significantly, since it requires only one group element per input bit instead of two. We detail the improvements in Section 5.2.

We have also updated and made minor corrections to the tables showing concrete key and proof sizes of different VRFs. In the previous version, we underestimated the proof sizes of the previous construction by Yamada [50], and the size of the public verification key in the construction by Jager [26].

Notation. Following usual conventions, we denote the natural numbers without zero by $\mathbb{N}$. We use $k \in \mathbb{N}$ as our security parameter. For $n \in \mathbb{N}$ we denote the set $\{1, \dots, n\}$ by $[n]$ and $[n] \cup \{0\}$ by $[n]_0$. Given a finite set S, we denote the power set as 2^S and write $x \overset{\$}{\leftarrow} S$ for drawing x uniformly at random from S. If $A(\cdot)$ is a probabilistic algorithm, we write $y \overset{\$}{\leftarrow} A(x)$ for executing A with input x and assigning the result of this execution to the variable y. For a vector $v \in S^n$ and $n \in \mathbb{N}$, we denote the i-th component of v as v_i. If not stated otherwise, all logarithms are to the base two and $\ln(x)$ denotes the natural logarithm of $x \in \mathbb{R}^{>0}$. We refer to a function $\epsilon : \mathbb{N} \to [0,1]$ as *negligible* if for all positive polynomials p and all $n \in \mathbb{N}$ large enough, it holds that $\epsilon(n) < 1/p(n)$.

2 Admissible Hash Functions

We first introduce some further notation specific to admissible hash functions. Let n, Q be polynomials over $\mathbb{N}$ and let $\mathcal{C} := \{C_k\}_{k \in \mathbb{N}}$ be a family of functions with $C_k : \Sigma^k \to \Sigma^{n(k)}$ for all $k \in \mathbb{N}$ and some finite alphabet Σ. Whenever it is clear from the context, we use n instead of $n(k)$ and Q instead of $Q(k)$. For a finite alphabet Σ with $\perp \notin \Sigma$, let $\Sigma_\perp := \Sigma \cup \{\perp\}$. For $d \in [n]$, we denote by $\Sigma_\perp^{(n,d)}$ the subset of $\Sigma_\perp^n$ containing all the elements of $\Sigma_\perp^n$ having exactly d components that are *not* $\perp$.

Binary biased PRF. An essential building block to define AHFs is the binary biased PRF F_K, that was introduced by Boneh and Boyen [9]. In some works, it is also referred to as *Bit-Fixing-Predicate.*

Definition 1 (The binary biased PRF [9]). *Let $C : \Sigma^k \to \Sigma^n$ and $K \in \Sigma_\perp^n$, then define*

$$F_K(X) := \begin{cases} 0, & \textit{if } \forall\, i \in [n] : C(X)_i = K_i \vee K_i = \perp \textit{ holds} \\ 1, & \textit{otherwise.} \end{cases} \tag{1}$$

Admissible hash functions. Now we are ready to define AHFs. Intuitively, an AHF realizes a partitioning of a set Σ^k into a "controlled" and an "uncontrolled" set, such that $X \in \Sigma^k$ lies in the "controlled" set *if and only if* $F_K(X) = 0$. For example, in a security proof for a digital signature scheme this would typically be used as follows:

- The reduction is able to simulate a signature for message $X \in \Sigma^k$ *if and only if* $F_K(X) = 1$.
- The reduction is able to extract the solution to a computationally hard problem from a signature for message $X^* \in \Sigma^k$ *if and only if* $F_K(X^*) = 0$.

This intuition yields the following generic definition of AHFs from [9].

Definition 2 (Admissible Hash Function [9]). *We call $\{C_\ell\}_{\ell \in \mathbb{N}}$ an $(n, Q, \gamma_{\min})$-*admissible hash function family *(AHF family) if there exists a PPT algorithm $K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, Q)$ generating $K \in \Sigma_\perp^n$ such that for all $(X^{(1)}, \ldots, X^{(Q)}, X^*) \in (\Sigma^k)^{Q+1}$ with $X^* \neq X^{(i)}$ it holds that*

$$\gamma_{\min}(k) \leq \Pr\left[F_K\left(X^{(1)}\right) = \cdots = F_K\left(X^{(Q(k))}\right) = 0 \wedge F_K\left(X^*\right) = 1\right], \tag{2}$$

where the probability is over the choice of $K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, Q)$.

Note that $\gamma_{\min}(k) \in [0,1]$ is a *lower* bound on the probability that the partitioning works as desired for any given sequence of values $(X^{(1)}, \dots, X^{(Q)}, X^*)$.

Remark 1. There have been several slightly different definitions of AHFs. The first definition of AHFs by Boneh and Boyen [9] includes the application of a collision resistant hash function before applying the function C above, allowing to process inputs of arbitrary length. However, most applications of AHFs, for example in [7,18,23,26,29,31,50], only consider fixed length inputs and therefore do not apply a collision resistant hash function. For this reason, we define standard AHFs with a bounded length input space and view the application of a collision resistant hash function in [9] only as a generic way of processing arbitrary length inputs with an information-theoretic AHF.

Balanced AHFs. Balanced AHFs are an extension of standard AHFs, introduced in [26]. Intuitively, a balanced AHF provides both a *lower* bound $\gamma_{\min}$ and an *upper* bound $\gamma_{\max}$ on the probability that partitioning works as desired for any given sequence of values $(X^{(1)}, \dots, X^{(Q)}, X^*)$. Furthermore, it is required that these bounds are *sufficiently close.* As shown in [26], this makes AHFs applicable in settings considering *indistinguishability-based* security experiments, such as indistinguishability of verifiable random functions. Balancedness makes it possible to avoid the "artificial abort" technique of Waters [49] and can be seen as an abstraction and adoption of a proof technique by Bellare and Ristenpart [4] to AHFs. Previous works [4,26] provide a detailed explanation of this.

Definition 3 (Balanced Admissible Hash Function [26]). *Let $\epsilon : \mathbb{N} \to [0,1]$ be a non-negligible function. We call $\{C_k\}_{k\in\mathbb{N}}$ an $(n, Q, \gamma_{\min}, \gamma_{\max})$-balanced AHF family if there exists a PPT $\mathsf{AdmSmp}(1^k, Q, \epsilon)$ and functions $\gamma_{\max}, \gamma_{\min} : \mathbb{N} \to [0,1]$, such that for all $(X^{(1)}, \dots, X^{(Q(k))}, X^*) \in (\Sigma^k)^{Q(k)+1}$ with $X^* \neq X^{(i)}$*

$$\gamma_{\min}(k) \le \Pr\left[F_K\left(X^{(1)}\right) = \cdots = F_K\left(X^{(Q(k))}\right) = 0 \wedge F_K\left(X^*\right) = 1\right] \le \gamma_{\max}(k)$$

holds, where the probability is over the choice of $K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, Q(k), \epsilon(k))$. We require that

$$\tau(k) := \epsilon(k)\gamma_{\min}(k) - \frac{\gamma_{\max}(k) - \gamma_{\min}(k)}{2}. \tag{3}$$

is a non-negligible function.

The term $\tau(k)$ may appear overly specific. However, it turns out that this is exactly what is required in common proofs that use balanced AHFs. We refer to [4,26] for details. As long as it does not lead to ambiguities, we will refer to both balanced admissible hash functions and admissible hash functions as AHFs in the sequel.

2.1 Instantiating AHFs from Error Correcting Codes

Error Correcting Codes. To describe how AHFs are instantiated from error correcting codes, let us first recap some fundamental notions.

Definition 4 (Hamming weight and distance). *Let $n \in \mathbb{N}$, q a prime power and $x, y \in \mathbb{F}_q^n$, then $\mathrm{wt}(x)$ is defined as the number of components of x that are not zero. We call $\mathrm{wt}(x)$ the* Hamming weight *of x and $\Delta(x, y) := \mathrm{wt}(x - y)$ the* Hamming distance *between x and y.*

Definition 5 (Linear ECCs). *Let q be a prime power, $k, n \in \mathbb{N}$ with $k < n$ and $G \in \mathbb{F}_q^{k \times n}$. Then $C = \{xG \mid x \in \mathbb{F}_q^k\}$ is the linear q-ary* error correcting code *(ECC) generated by the* generator matrix *G. We say that C has* minimal distance

$$d := \min_{\substack{x,x' \in \mathbb{F}_q^k \\ x \neq x'}} (\Delta(xG, x'G)).$$

Furthermore we refer to $\delta(C) := d/n$ as the relative minimal distance *and to $\mathcal{R}(C) := k/n$ as the* rate *of C. If C has a minimal distance of d, we say that C is a linear $[n, k, d]_q$ ECC.*

As usual, we will also refer to the mapping $C : \mathbb{F}_q^k \to F_q^n, x \mapsto xG$ as an ECC. If the alphabet of the code is clear from the context, we drop the index q.

AHFs from ECCs. To the best of our knowledge, all constructions of AHFs and bAHFs use an algorithm AdmSmp that samples K as follows. First $d \in [n]$ is calculated as a function of a bound on Q and the advantage ϵ of a given adversary $\mathcal{A}$ (the latter only for bAHFs). Then K is chosen uniformly at random from $K \overset{\$}{\leftarrow} \Sigma_{\perp}^{(n,d)}$.

Theorem 1 (Instantiation of AHFs using an ECC [18,26]). *Let q be a prime power and let $\{C_k\}_{k \in \mathbb{N}}$ be a family of $[n(k), k, n(k) \cdot \delta(k)]_q$ ECCs, where $\delta = \delta(k) \in [0, 1/2)$ denotes the relative distance of C_k. If $K \overset{\$}{\leftarrow} \mathbb{F}_q^{(n,d)}$ is chosen uniformly at random for some $d \in [n(k)]$, then*

$$\gamma_{\min}(k) \leq \Pr\left[F_K\left(X^{(1)}\right) = \cdots = F_K\left(X^{(Q)}\right) = 0 \wedge F_K\left(X^*\right) = 1\right] \leq \gamma_{\max}(k)$$

holds for all $X^{(1)}, \ldots, X^{(Q)}, X^* \in \left(\mathbb{F}_q^k\right)^{(Q+1)}$ and

$$\gamma_{\min}(k) := (1 - Q \cdot (1 - \delta(k))^d) \cdot q^{-d} \qquad \textit{and} \qquad \gamma_{\max}(k) := q^{-d}. \tag{4}$$

The proof is given in [26]. If

$$d := \log_{1-\delta}\left(\frac{-\ln(q)}{Q \cdot \ln\left(\frac{1-\delta}{q}\right)}\right) \tag{5}$$

is used, then $\gamma_{\min}$ is non-negligible as shown in [18]. Note that d in Equation 5 maximizes $\gamma_{\min}$ from Equation 4 compared to [18] and therefore yields slightly better parameters.

Balanced AHFs from ECCs. If the parameter d from Theorem 1 is set to

$$d := \log_{1-\delta}\left(\frac{-2\cdot\epsilon\cdot\ln(q)}{(2\cdot\epsilon+1)\cdot Q\cdot\ln\left(\frac{1-\delta}{q}\right)}\right)$$

then one can also prove that the above construction is a *balanced* AHF [26]. Again, this value of d improves on that given in [26], in the sense that $\tau = \epsilon\cdot\gamma_{\min} - (\gamma_{\max} - \gamma_{\min})/2$ is maximize.

Remark. Note that the choices of d above, just as the choices from [18,26], can yield a d larger than n for small security parameters. However, $d \in [n]$ holds for reasonable parameters as showcased in Section 6.

2.2 Efficiency Bounds on Admissible Hash Functions

In order to be able to efficiently instantiate the code-based AHFs, it is important to reduce the length of code words n. This is because applications usually embed the AHF in the public keys or public parameters of a cryptosystem, and the size of these depends on the length of code words. For example, the public key of the VRF of [26] contains two groups element for every bit in the output of the ECC. The VRFs in [29,50] reduce this, but still contain at least logarithmically many group elements. Even though asymptotically logarithmic, the number may still be impractically large when instantiated concretely - we discuss this below in Section 6. In the following, we provide the first analysis of inherent limitations of instantiating AHFs with ECCs, by applying results from coding theory.

Gilbert-Varshamov bound. In order to instantiate AHFs efficiently, we need a code that has both a high rate and a high relative minimal distance. Coding theorists worked on the construction of such codes and accompanying bounds for decades [34]. Asymptotically, the *Gilbert-Varshamov bound* guarantees the *existence* of well-suited families of binary ECCs, but we note that this result is *not constructive.*

Theorem 2 (Gilbert-Varshamov bound [20,47]). *For all $n \in \mathbb{N}$ and $c \in (0, 1/2)$, there exists an ECC $C \subset \mathbb{F}_2^n$ with*

$$\delta(C) \geq c \qquad \textit{and} \qquad R(C) \geq 1 - H_2(\delta).$$

H_2 denotes the binary entropy, defined as

$$H_2(p) := p\cdot\log(1/p) + (1-p)\cdot\log(1/(1-p)).$$

Even though the Gilbert-Varshamov (GV) bound guarantees the existence of families of binary ECCs with the parameters from above, no explicit construction of such a family attaining the GV bound is known so far. Random linear codes attain the GV bound [47]. However, as [46,17] show, there is no efficient algorithm that can compute or approximate the minimal distance of a random linear code, which would be necessary in order to instantiate the code concretely and efficiently. Also, algebraic geometric ECCs like [43] beat the GV bound, but only for larger, non-binary alphabets, which are not suitable for most constructions using AHFs.

Hence, when instantiating a family of (balanced) AHFs, we can treat the GV bound as an upper bound on what is possible with currently known families of binary ECCs. The GV bound yields that for $\delta = 0.2$, the best rate we can hope to achieve with known construction of families of ECCs is ≈ 0.28. For the VRF from [26], this would require a number of group elements in the public key that is at least about four times larger than the number of input bits. It is possible however to construct a family of binary ECCs that comes relatively close to this bound by concatenating algebraic geometry codes with binary error correcting codes [43, Section V].

In order to estimate the efficiency of code-based AHFs, we assumed a code family that achieves the GV bound, and computed the size of verification keys, secret keys, and proofs for different verifiable random functions that use AHFs to achieve adaptive security. See Section 6, Table 1 and Table 2.

Remark 2. Our analysis based on the GV bound is somewhat conservative, in the sense that no efficient construction of a family of codes that achieves the GV bound asymptotically is known (even though they are known to exist).[2] Therefore, our efficiency analysis of cryptosystems based on codes that meet the GV bound is currently overly optimistic, and any known code family would lead to even worse parameters.

McEliece-Rodemich-Rumsey-Welch (MRRW) bound. Things may improve when we consider instantiations for a specific input length k of the ECC.[3] For example for $k = 128$, there is a $[255, 128, 38]_2$ ECC C based on a BCH code [22]. Thus, this code beats the GV bound as $R(C) \approx 1/2$ and $\delta(C) \approx 0.15$, whereas the GV bound only allows for $R(C) \approx 0.39$ for $\delta(C) = 0.15$. However, even when considering concrete input sizes of ECCs, there are bounds on the relation between the rate and the relative minimal distance. The sharpest known bound for binary ECCs is the MRRW bound presented in Theorem 3.

Theorem 3 (MRRW bound [35]). *Let C be an $[n, k, d]_2$ ECC with relative distance $\delta(C) \in (0, 1/2)$ and let $g(x) := H_2((1 - \sqrt{1-x})/2)$. Then*

$$R(C) \leq \min_{0 \leq u \leq 1-2\delta} \{1 + g(u^2) - g(u^2 + 2\delta(C)u + 2\delta(C))\}$$

holds for the rate of C.

The MRRW-Bound once more yields limits on what can be achieved concretely. For example, every binary ECC C with $\delta(C) = 0.15$ inevitably has $R(C) < 0.58$. Analogously, any ECC C with $\delta(C) = 0.2$ has $R(C) < 0.47$.

[2] Codes based on expander graphs can get close to this bound, while not achieving it [45].

[3] For the VRFs in [23,26,29,31,50] this is identical to the input length.

Again, we estimate the efficiency of code-based AHFs by assuming a code that achieves the MRRW bound, and compute the size of verification keys, secret keys, and proofs for different AHF-based verifiable random functions. See Section Section 6, Table 1 and Table 2.

3 Computational Admissible Hash Functions

In order to overcome the inherent limitations of (balanced) AHFs and their instantiation with ECCs, we *relax the constraints* and consider a *computational* setting. To this end, we allow C to have inputs X, Y, $X \neq Y$, such that $\Delta(C(X), C(Y)) < \delta n$ or even $C(X) = C(Y)$. For an adversary, however, such inputs X and Y should be *computationally infeasible* to find. This relaxation allows us to reduce redundancy inherent to ECCs and thus the length of the output of C. We refer to this relaxed variant of AHFs as *computational admissible hash functions* (cAHFs).

Furthermore, we aim at defining cAHFs in a way that allows us to replace bAHFs in many constructions like [2,23,26,29,32,50], while making only minor modifications to the constructions or their accompanying security proofs. Since we instantiate cAHFs with families of hash functions in Section 3.1, we change the notation and use H for the computational setting instead of C in order to avoid confusion between the two settings.

Defining computational admissible hash functions. We keep using the binary biased PRF F_K from [9], since we strive to allow to *generically replace* bAHFs with cAHFs. Allowing H to have pairs of inputs X, Y with $X \neq Y$ and $H(X) = H(Y)$ comes with the problem that an adversary can have a collision for H hard coded. Therefore, we need to draw the function H from a *family of functions* $\mathcal{H}$. This requires us to incorporate H and do the following minor modification to the definition of F_K. Nevertheless, note that the definition of cAHFs is not syntactically bound to families of hash functions.

$$F_{K,H}(X) = \begin{cases} 0, & \text{if } \forall\, j \in [n] : H(X)_j = K_j \vee K_j = \bot \\ 1, & \text{otherwise.} \end{cases} \tag{6}$$

Definition 6 (Computational admissible hash function). *Let $\mathcal{H} = \{H : \{0,1\}^* \to \{0,1\}^n\}$ be a family of functions and $H \in \mathcal{H}$. For $K \in \{0,1,\bot\}^n$ and all $(X^{(1)}, \ldots, X^{(Q)}, X^*)$ with $X^{(i)}, X^* \in \{0,1\}^*$ and $X^{(i)} \neq X^*$ for all i, we let $X^{(Q+1)} := X^*$ to ease notation and define the events* coll *and* $\mathsf{badchal}$ *as follows.*

$$\begin{aligned} \mathsf{badchal} &\iff F_{K,H}(X^*) \neq 0 \\ \mathsf{coll} &\iff \exists\, i, j \textit{ with } X^{(i)} \neq X^{(j)} \textit{ s.t.} \\ &\qquad \forall\, \ell \in [n] : H(X^{(i)})_\ell = H(X^{(j)})_\ell \vee K_\ell = \bot \end{aligned}$$

Let $t_\mathcal{A} \in \mathbb{N}$ and $\epsilon_\mathcal{A} \in (0,1]$ such that $t_\mathcal{A}/\epsilon_\mathcal{A} < 2^k$, where k is the security parameter. We say that $\mathcal{H}$ is a family of computational admissible hash functions (cAHFs)*, if there is an efficient algorithm* $\mathsf{AdmSmp}(1^k, t_\mathcal{A}, \epsilon_\mathcal{A})$ *generating*

$K \in \{0,1,\bot\}^n$ such that for every adversary $\mathcal{A}$ running in time $t_\mathcal{A}$ outputting $(X^{(1)},\dots,X^{(Q)},X^*)$ it holds that

$$\tau(k) := \Pr[\neg\, \mathsf{badchal}](\epsilon_\mathcal{A} - \Pr[\mathsf{coll}]) \tag{7}$$

is non-negligible as a function in k. The probabilities are over the randomness used by $\mathcal{A}$, $H \xleftarrow{\$} \mathcal{H}$ and $K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, t_\mathcal{A}, \epsilon)$.

Remark 3. The term τ in Equation 7 is the equivalent of τ for bAHFs in Definition 3, in the sense that it conveniently describes a term that typically occurs in a reduction-based security proof that uses a computational AHF. Intuitively, it captures a security proof that in a first step aborts when coll occurs and in a second step aborts when $\mathsf{badchal}$ occurs. See Section 5.3 for a concrete application.

3.1 cAHFs from Truncation Collision Resistant Hash Functions

We show how to construct very efficient cAHFs based on *truncation collision-resistant hash functions* (TCRHFs), as introduced in [27].

Truncation collision resistant hash functions. Let $H : \{0,1\}^* \to \{0,1\}^n$ be a cryptographic hash function. We write $H_{:j} : \{0,1\}^* \to \{0,1\}^j$ to denote the hash function H, with outputs truncated to the first j bits. Essentially, a hash function is *truncation collision resistant*, if for every prefix of length $j \in [n]$ there is no significantly more efficient algorithm to find a collision for $H_{:j}$ than the birthday attack. Note that this property is likely satisfied by standard cryptographic hash functions, like SHA-3. Furthermore, as explained in [27], one can easily obtain a suitable *family* of hash functions from a standard hash function, e.g. by choosing a random key that is prefixed to all hash function inputs. We refer to such families of hash functions as families of *keyed hash functions*[4].

Definition 7 (Truncation collision resistance [27]). *Let $\mathcal{H} = \{H : \{0,1\}^* \to \{0,1\}^n\}$ be a family of keyed hash functions. For $j \in [n]$, we say that an adversary $\mathcal{A}$ j-breaks the truncation collision resistance of $\mathcal{H}$, if it runs in time $t_\mathcal{A}$ and*

$$\Pr_{H \xleftarrow{\$} \mathcal{H}}\left[\begin{array}{l}(x_0,\dots x_q) \xleftarrow{\$} \mathcal{A}(H) : \\ \exists\, u,v \text{ s.t. } H_{:j}(x_u) = H_{:j}(x_v) \wedge x_u \neq x_v\end{array}\right] > \frac{t_\mathcal{A}(t_\mathcal{A}-1)}{2^{j+1}}.$$

We say $\mathcal{H}$ is truncation collision resistant*, if there exists no adversary $\mathcal{A}$ j-breaking the truncation collision resistance of H for any $j \in [n]$.*

We deem truncation collision resistance a reasonable assumption since truncated versions of SHA-256 (to 224 bits) and SHA-512 (to 384 bits) have already been standardized by NIST in [38]. Furthermore, [39] defines *extendable-output functions* (XOF) based on SHA-3. These allow to extend the output of the hash function to an arbitrary length while maintaining collision resistance.

[4] A detailed discussion of keyed hash functions can be found in [30].

Useful technical lemma. The following lemma is a variant of [27, Lemma 1], tailored to our application and cAHFs, which yields better parameters than the corresponding result in [27]. The condition $t/\epsilon < 2^k$ captures that we consider an efficient adversary, i. e. one with a small time complexity, a high success probability or both. Choosing j as small as possible such that $(2t(2t-1))/2^j \leq \epsilon/2$ holds then yields the bounds on j and $1/2^j$ below. The lower bound on $1/2^j$ is important because this is the probability that a prefix of length j matches a random binary string of length j. A more thorough explanation can be found in [27].

Lemma 1. *Let $t \in \mathbb{N}$, $\epsilon \in (0,1]$ such that $t/\epsilon < 2^k$, and $j := \lceil \log(4t(2t-1)/\epsilon) \rceil$. Then it holds that*

$$j \in \{1, \ldots, 2k+3\}, \qquad \frac{2t(2t-1)}{2^j} \leq \frac{\epsilon}{2}, \text{ and } \qquad \frac{1}{2^j} \geq \frac{\epsilon}{16t^2 - 8t}.$$

Proof. We start by proving $j \in \{1, \ldots, 2k+3\}$.

$$\begin{aligned} j = \lceil \log(4t(2t-1)/\epsilon) \rceil &\leq \left\lceil \log\left(4 \cdot 2^k (2t-1)\right) \right\rceil \\ &\leq \left\lceil \log\left(8 \cdot 2^k t\right) \right\rceil \leq \left\lceil \log\left(2^k 2^{k+3}\right) \right\rceil = 2k+3 \end{aligned}$$

Since $4t(2t-1) = 8t^2 - 4t > 1$ for all $t \in \mathbb{N}$ and $\epsilon \in (0,1]$, we have $\log(4t(2t-1)/\epsilon) > 0$ and therefore $j \geq 1$.

We proceed to prove $2t(2t-1)/2^j \leq \epsilon/2$.

$$\frac{2t(2t-1)}{2^j} = \frac{2t(2t-1)}{2^{\lceil \log(4t(2t-1)/\epsilon) \rceil}} \leq \frac{\epsilon 2t(2t-1)}{4t(2t-1)} = \frac{\epsilon}{2}$$

Finally, we have

$$\frac{1}{2^j} = \frac{1}{2^{\lceil \log(4t(2t-1)/\epsilon) \rceil}} \geq \frac{1}{2} \cdot \frac{\epsilon}{4t(2t-1)} = \frac{\epsilon}{16t^2 - 8t}.$$

Constructing cAHFs from TCRHFs. Lemma 1 enables us to prove that a family of TCRHFs is also a family of cAHFs. Note that even though the first definition of AHFs in [9] already incorporates collision resistant hash functions, they are only used to enable the processing of arbitrary length inputs, while the core of the AHF in [9] is the error correcting code that yields an information-theoretic AHF, which we replace with TCRHFs.

Theorem 4. *Let $\mathcal{H} = \{H : \{0,1\}^* \to \{0,1\}^{2k+3}\}$ be a family of truncation collision resistant keyed hash functions in the sense of Definition 7. Then $\mathcal{H}$ is a family of computational AHFs. In particular, let $t_{\mathcal{A}} \in \mathbb{N}$ and $\epsilon_{\mathcal{A}} \in (0,1]$ such that $t_{\mathcal{A}}/\epsilon_{\mathcal{A}} < 2^k$. Then for every adversary $\mathcal{A}$ running in time $t_{\mathcal{A}}$ that, given $H \xleftarrow{\$} \mathcal{H}$, outputs $X^{(1)}, \ldots, X^{(Q)}, X^* \in \{0,1\}^*$ with $X^{(i)} \neq X^*$, there is an algorithm $\mathsf{AdmSmp}(1^k, t, \epsilon)$ such that*

$$\Pr[\neg\, \mathsf{badchal}](\epsilon - \Pr[\mathsf{coll}]) \geq \epsilon^2/(32t^2 - 16t).$$

In particular, if $t_{\mathcal{A}}$ is polynomial in k and ϵ is non-negligible in k, then $\epsilon_{\mathcal{A}}^2/(32t_{\mathcal{A}}^2)$ is also non-negligible.

Proof. Let $n := 2k+3$. The algorithm $\mathsf{AdmSmp}(1^k, t, \epsilon)$ sets $j := \lceil \log(4t(2t-1)/\epsilon) \rceil$, samples $K' \xleftarrow{\$} \{0,1\}^j$, and defines $K := K' \| \perp^{n-j}$, where $\|$ denotes string concatenation and $\perp^{n-j}$ the string consisting of $(n-j)$-times the $\perp$-symbol. In total the key K consists of j uniformly random bits, padded to a string of length n in $\{0,1,\perp\}^n$ by appending $\perp$-symbols. Note that $n \geq j$ by Lemma 1.

Recall that $\neg\,\mathsf{badchal}$ occurs iff $F_{K,H}(X^*) = 0$. For our construction, this means that the first j bits of $H(X^*)$ are *identical* to K', the first j bits of K. Since K' is chosen uniformly random, and independent of $H \xleftarrow{\$} \mathcal{H}$, this happens with probability 2^{-j}. We therefore have

$$\Pr[\neg\,\mathsf{badchal}] = \frac{1}{2^j} \geq \frac{\epsilon_{\mathcal{A}}}{16t_{\mathcal{A}}^2 - 8t},$$

where the inequality uses Lemma 1. Furthermore, recall that coll occurs, if the adversary outputs, as queries or as challenge, two values $X \neq Y$ such that $H(X)$ and $H(Y)$ are identical in all positions where K is not $\perp$. In particular, we then have $H_{:j}(X) = H_{:j}(Y)$. We therefore claim that we have

$$\Pr[\mathsf{coll}] \leq \frac{\epsilon_{\mathcal{A}}}{2}.$$

We prove this upper bound on $\Pr[\mathsf{coll}]$ by contradiction. Assume $\mathcal{A}$ outputs $X = (X^{(1)}, \ldots, X^{(Q)}, X^*)$ such that $\Pr[\mathsf{coll}] > \epsilon/2$. Then we can construct an adversary $\mathcal{B}$ that j-breaks the truncation collision resistance of $\mathcal{H}$. $\mathcal{B}$ runs $\mathcal{A}$, waits for $\mathcal{A}$ to output X and then outputs X itself. $\mathcal{B}$'s running time consists of the time to execute $\mathcal{A}$ plus the time to output X, yielding[5] $t_{\mathcal{B}} \leq 2 \cdot t_{\mathcal{A}}$. Thus, $\mathcal{B}$ is an algorithm with success probability at least $\epsilon_{\mathcal{A}}/2$ in j-breaking the truncation collision resistance of $\mathcal{H}$. We furthermore have

$$\Pr[\mathsf{coll}] > \epsilon_{\mathcal{A}}/2 \geq \frac{2t_{\mathcal{A}}(2t_{\mathcal{A}}-1)}{2^j} \geq \frac{2t_{\mathcal{A}}(2t_{\mathcal{A}}-1)}{2^j} = \frac{t_{\mathcal{B}}(t_{\mathcal{B}}-1)}{2^j} > \frac{t_{\mathcal{B}}(t_{\mathcal{B}}-1)}{2^{j+1}},$$

where the second inequality follows from Lemma 1. This contradicts the truncation collision resistance of $\mathcal{H}$ and therefore proves the upper bound on $\Pr[\mathsf{coll}]$. In conclusion, the following equation yields the theorem.

$$\Pr[\neg\,\mathsf{badchal}](\epsilon_{\mathcal{A}} - \Pr[\mathsf{coll}]) \geq \frac{\epsilon_{\mathcal{A}}}{16t_{\mathcal{A}}^2 - 8t}\left(\epsilon_{\mathcal{A}} - \frac{\epsilon_{\mathcal{A}}}{2}\right) = \frac{\epsilon_{\mathcal{A}}^2}{32t_{\mathcal{A}}^2 - 16t}$$

Note that if $t_{\mathcal{A}}$ is polynomial in k and $\epsilon_{\mathcal{A}}$ is non-negligible in k, then $\epsilon_{\mathcal{A}}^2/(32t_{\mathcal{A}}^2 - 16t)$ is also non-negligible.

[5] One could tighten the upper bound $t_{\mathcal{B}}$ to $t_{\mathcal{A}} + Q$. However, it would at most save a factor of two in the run time of $\mathcal{B}$ and would complicate the analysis. We therefore use the slightly less tight bound.

4 Verifiable Random Functions and Their Security

Verifiable random functions are essentially pseudorandom functions, where each function V_{sk} is associated with a secret key sk and a corresponding public verification key vk. Given sk and an element X from the domain of V_{sk}, one can efficiently compute a *non-interactive, publicly verifiable* proof π that $Y = V_{\mathsf{sk}}(X)$ was computed correctly. For security, it is required that for each X only one unique value Y such that the statement "$Y = V_{\mathsf{sk}}(X)$" can be proven may exist (*unique provability*), and that $V_{\mathsf{sk}}(X)$ is indistinguishable from random, if no corresponding proof is given (*pseudorandomness*).

Syntax of VRFs. Formally, a VRF consists of algorithms $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Vfy})$ with the following syntax.

- $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{Gen}(1^k)$ takes as input a security parameter k and outputs a key pair $(\mathsf{vk}, \mathsf{sk})$. We say that sk is the *secret key* and vk is the *verification key.*
- $(Y, \pi) \overset{\$}{\leftarrow} \mathsf{Eval}(\mathsf{sk}, X)$ takes as input a secret key sk and $X \in \{0,1\}^*$, and outputs a function value $Y \in \mathcal{Y}$, where $\mathcal{Y}$ is a finite set, and a proof π. We write $V_{\mathsf{sk}}(X)$ to denote the function value Y computed by Eval on input (sk, X).
- $\mathsf{Vfy}(\mathsf{vk}, X, Y, \pi) \in \{0,1\}$ takes as input a verification key vk, $X \in \{0,1\}^*$, $Y \in \mathcal{Y}$, and proof π, and outputs a bit.

$\mathsf{Initialize}$:	$\mathsf{Evaluate}(X)$:	$\mathsf{Challenge}(X^*)$:	$\mathsf{Finalize}(b')$:
$(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{Gen}(1^k)$	$(Y, \pi) \overset{\$}{\leftarrow} \mathsf{Eval}(\mathsf{sk}, X)$	$(Y_0, \pi) \overset{\$}{\leftarrow} \mathsf{Eval}(\mathsf{sk}, X^*)$	If $b = b'$ then
Return vk	Return (Y, π)	$Y_1 \overset{\$}{\leftarrow} \mathcal{Y}$	Return 1
		Return Y_b	Return 0

Fig. 1. Procedures defining the VRF security experiment.

Definition 8. $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Vfy})$ *is a* verifiable random function *(VRF) if all of the following hold.*

Correctness. *For all* $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{Gen}(1^k)$ *and* $X \in \{0,1\}^*$ *holds: if* $(Y, \pi) \overset{\$}{\leftarrow} \mathsf{Eval}(\mathsf{sk}, X)$*, then* $\mathsf{Vfy}(\mathsf{vk}, X, Y, \pi) = 1$. *Algorithms* Gen, Eval, Vfy *are polynomial-time.*

Unique provability. *For all* $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{Gen}(1^k)$ *and all* $X \in \{0,1\}^*$*, there does not* exist *any tuple* (Y_0, π_0, Y_1, π_1) *such that* $Y_0 \neq Y_1$ *and* $\mathsf{Vfy}(\mathsf{vk}, X, Y_0, \pi_0) = \mathsf{Vfy}(\mathsf{vk}, X, Y_1, \pi_1) = 1$.

Pseudorandomness. *Consider an attacker* $\mathcal{A}$ *with access (via oracle queries) to the procedures defined in Figure 1. Let* $G^{\mathcal{A}}_{\mathsf{VRF}}$ *denote the game where* $\mathcal{A}$ *first queries* $\mathsf{Initialize}$*, then* $\mathsf{Challenge}$*, then* $\mathsf{Finalize}$*. The output of* $\mathsf{Finalize}$ *is the* *output of the game. Moreover, $\mathcal{A}$ may arbitrarily issue* $\mathsf{Evaluate}$*-queries, but only after querying* $\mathsf{Initialize}$ *and before querying* $\mathsf{Finalize}$*. We say that $\mathcal{A}$ is* legitimate*, if $\mathcal{A}$ never queries* $\mathsf{Evaluate}(X)$ *and* $\mathsf{Challenge}(X^*)$ *with $X = X^*$ throughout the game. We define the advantage of $\mathcal{A}$ in breaking the pseudo-randomness as*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{VRF}}(k) := \Pr\left[G_{\mathsf{VRF}}^{\mathcal{A}} = 1\right] - 1/2$$

5 Verifiable Random Functions from cAHFs

In this section, we show a new construction of VRFs. Our construction is closely related to the verifiable random function of Jager [26], which uses a balanced AHF based on error-correcting codes. We show how the error correcting codes can be replaced by a standard hash function by using *computational* AHFs (cAHFs). We then show how cAHFs are used in a proof in Section 5.3. Furthermore, we improve Jager's VRF with a new proof technique that allows us to halve the size of public verification keys and secret keys.

5.1 Bilinear Groups.

Our VRF construction uses *Bilinear Groups*. We assume that they are generated based on the security parameter. Following [23], we refer to such a generation algorithm as *Bilinear Group Generator* as defined below.

Definition 9. *A* Bilinear Group Generator *is a probabilistic polynomial-time algorithm* GrpGen *that takes as input a security parameter k (in unary) and outputs $\Pi = (p, \mathbb{G}, \mathbb{G}_T, \circ, \circ_T, e, \phi(1)) \xleftarrow{\$} \mathsf{GrpGen}(1^k)$ such that the following requirements are satisfied.*

1. *p is a prime and $\log(p) \in \Omega(k)$*
2. *$\mathbb{G}$ and $\mathbb{G}_T$ are subsets of $\{0,1\}^*$, defined by algorithmic descriptions of maps $\phi : \mathbb{Z}_p \to \mathbb{G}$ and $\phi_T : \mathbb{Z}_p \to \mathbb{G}_T$.*
3. *$\circ$ and $\circ_T$ are algorithmic descriptions of efficiently computable (in the security parameter) maps $\circ : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ and $\circ_T : \mathbb{G}_T \times \mathbb{G}_T \to \mathbb{G}_T$, such that*
   a) *$(\mathbb{G}, \circ)$ and $(\mathbb{G}_T, \circ_T)$ form algebraic groups,*
   b) *ϕ is a group isomorphism from $(\mathbb{Z}_p, +)$ to $(\mathbb{G}, \circ)$ and*
   c) *ϕ_T is a group isomorphism from $(\mathbb{Z}_p, +)$ to $(\mathbb{G}_T, \circ_T)$.*
4. *e is an algorithmic description of an efficiently computable (in the security parameter) bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We require that e is non-degenerate, that is,*

$$x \neq 0 \Rightarrow e(\phi(x), \phi(x)) \neq \phi_T(0).$$

The unique provability property of VRFs requires each group element from the bilinear group to have a unique representation. Therefore, we use the notion of *Certified Bilinear Groups*, introduced by Hofheinz and Jager in [23]. They allow us to formally prove *Unique Provability* of our VRF by ensuring that each group element has a unique representation that can efficiently be verified. Formally, we define certified bilinear groups as follows.

Definition 10. *We say that group generator* GrpGen *is certified, if there exist deterministic polynomial-time (in the security parameter) algorithms* GrpVfy *and* $\mathsf{GrpElemVfy}$ *with the following properties.*

Parameter Validation. *Given the security parameter (in unary) and a string* Π*, which is not necessarily generated by* GrpGen*, algorithm* $\mathsf{GrpVfy}(1^k, \Pi)$ *outputs 1* if and only if Π *has the form*

$$\Pi = (p, \mathbb{G}, \mathbb{G}_T, \circ, \circ_T, e, \phi(1))$$

and all requirements from Definition 9 are satisfied.

Recognition and Unique Representation of Elements of $\mathbb{G}$. *Furthermore, we require that each element in* $\mathbb{G}$ *has a* unique *representation, which can be efficiently recognized. That is, on input the security parameter (in unary) and two strings* Π *and* s*,* $\mathsf{GrpElemVfy}(1^k, \Pi, s)$ *outputs 1 if and only if* $\mathsf{GrpVfy}(1^k, \Pi) = 1$ *and it holds that* $s = \phi(x)$ *for some* $x \in \mathbb{Z}_p$*. Here* $\phi : \mathbb{Z}_p \to \mathbb{G}$ *denotes the fixed group isomorphism contained in* Π *to specify the representation of elements of* $\mathbb{G}$*.*

Our definition of certifiedness deviates from the original definition [23] in two regards. First, in addition to GrpVfy, we also require the existence of an algorithm $\mathsf{GrpElemVfy}$ that is used to verify the encoding of a group element, whereas in [23], GrpVfy serves both purposes. Second, the security parameter (in unary) is an input to both verification algorithms in our definition.

5.2 VRF Construction

Let $\mathcal{H} = \{H : \{0,1\}^* \to \{0,1\}^n\}$ be a family of keyed hash functions for some $n \in \mathbb{N}$, let GrpGen be a certified bilinear group generator and let $\mathcal{VF} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Vfy})$ be the following algorithms.

Key generation. $\mathsf{Gen}(1^k)$ runs $\Pi \xleftarrow{\$} \mathsf{GrpGen}(1^k)$, chooses a random hash function $H \xleftarrow{\$} \mathcal{H}$ and random generators $g, h \xleftarrow{\$} \mathbb{G}$, where $\mathbb{G}$ is from Π, and $\alpha_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [n+1]_0$. It then defines $g_i := g^{\alpha_i}$ and the keys as

$$\mathsf{vk} := \big(H, \Pi, g, h, \{g_i\}_{i\in[n+1]_0}\big) \qquad \text{and} \qquad \mathsf{sk} := \{\alpha_i : i \in [n+1]_0\}.$$

Evaluation. On input vk, sk and $X \in \{0,1\}^*$, Eval computes $(H_1, \ldots, H_n) := H(X)$,

$$\alpha_X := \alpha_0 \cdot \left(\prod_{i=1}^{n} \alpha_i^{H_i}\right) \cdot \alpha_{n+1} \qquad \text{and} \qquad Y = V_{\mathsf{sk}}(X) := e(g,h)^{\alpha_X}.$$

To compute the proof, it sets $\pi_0 := g^{\alpha_0}$ and then computes $\pi_1, \ldots, \pi_n$ as

$$\pi_i := \pi_{i-1}^{\left(\alpha_i^{H_i}\right)}$$

for $i \in [n]$. Finally, it sets $\pi_{n+1} := \pi_n^{\alpha_{n+1}}$ and outputs $(Y, \pi = (\pi_1, \ldots, \pi_{n+1}))$.

Verification. Given 1^k, vk, $X \in \{0,1\}^*$ and $(Y, \pi = (\pi_1, \ldots, \pi_{n+1}))$, Vfy tests if Y and π contain only valid group elements using GrpVfy and $\mathsf{GrpElemVfy}$, and outputs 0 if not. Then it computes $(H_1, \ldots, H_n) := H(X) \in \{0,1\}^n$, defines $\pi_0 := g_0$, and outputs 1 if and only if for all $n \in [n]$ it holds that

$$e(\pi_i, g) = \begin{cases} e(\pi_{i-1}, g) & \text{if } H_i = 0 \text{ and} \\ e(\pi_{i-1}, g_i) & \text{if } H_i = 1, \end{cases} \tag{8}$$

and both

$$e(\pi_{n+1}, g) = e(\pi_n, g_{n+1}) \quad \text{and} \quad Y = e(\pi_{n+1}, h) \tag{9}$$

hold.

Comparison to Jager's VRF. Our construction improves Jager's VRF in two aspects. First, our verification and secret key contain *only one element* for each $i \in [n]$, compared to two in Jager's VRF. This improvement is possible by encoding the keys of the cAHF into the public and secret keys in a more efficient way. We explain this technique in detail in the proof of Theorem 5. The second improvement is that we instantiate the VRF with a *cAHF instead of an bAHF.* Therefore, the Eval and Vfy apply *a standard hash function instead of an error correcting code* to each input. This changes the input space of the VRF from $\{0,1\}^k$ to $\{0,1\}^*$. In the same way, cAHFs are applicable to the VRFs of Katsumata [29] and Yamada [50]. Notice, that the first improvement is also applicable when the VRF is instantiated with a bAHF instead of a cAHF.

Remark 4. The terms α_{n+1} and g_{n+1} in our construction are equivalent to appending a bit that is always set to one to each output of the hash function. This ensures that there is no input to the hash function resulting in an all-zero output, which is needed in the security proof. Alternatively, we could assume preimage resistance of the hash function and use that there is no efficient adversary that is able to find a preimage for the all-zeros output. We could then guess a position that is one for all inputs provided by the adversary. We chose to introduce α_{n+1} and g_{n+1} instead for simplicity and clarity.

Correctness. For any $k \in \mathbb{N}$ and any $X \in \{0,1\}^*$, let $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^k)$ and $(Y, \pi) \leftarrow \mathsf{Eval}(\mathsf{sk}, X)$. We now consider the behaviour of $\mathsf{Vfy}(1^k, \mathsf{vk}, X, (Y, \pi))$. Since Y and π are results of Eval, both GrpVfy and $\mathsf{GrpElemVfy}$ output 1 and thus Vfy does not reject the input. Now let $(H_1, \ldots, H_n) := H(X)$ and $\pi_0 := g_0$. Then for all $n \in [n]$, we have

$$e(\pi_i, g) = e\left(\pi_{i-1}^{\alpha_i^{H_i}}\right) = \begin{cases} e(\pi_{i-1}^{\alpha_i^0}, g) = e(\pi_{i-1}, g) & \text{if } H_i = 0 \text{ and} \\ e(\pi_{i-1}^{\alpha_i^1}, g) = e(\pi_{i-1}, g^{\alpha_i}) = e(\pi_{i-1}, g_i) & \text{if } H_i = 1. \end{cases}$$

Therefore, Equation 8 is fulfilled. Furthermore, we have

$$e(\pi_{n+1}, g) = e(\pi_n^{\alpha_{n+1}}, g) = e(\pi_n, g_{n+1}),$$

which fulfills the first part of Equation 9. Moreover, notice that for all $i \in [n]$, we have

$$\pi_i = g^{\alpha_0 \prod_{j=1}^{i} \alpha_j^{H_j}} \qquad \text{and} \qquad \pi_{n+1} = \pi_n^{\alpha_{n+1}} = g^{\alpha_0 \left(\prod_{j=1}^{i-1} \alpha_j^{H_j}\right)\alpha_{n+1}} = g^{\alpha_X}.$$

Thus, $e(\pi_{n+1}, h) = e(g^{\alpha_X}, h) = e(g, h)^{\alpha_X}$ holds, fulfilling Equation 9's second part. Hence, Vfy outputs 1 on input $(1^k, \mathsf{vk}, X, (Y, \pi))$. Finally, as can easily be verified, $\mathsf{Gen}, \mathsf{Eval}$ and Vfy are all polynomial-time algorithms.

Unique Provability. We show that for any $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{Gen}(1^k)$ and $X \in \{0,1\}^*$, there are unique Y, π such that $\mathsf{Vfy}(1^k, \mathsf{vk}, X, Y, \pi) = 1$. Let $(H_1, \ldots, H_n) := H(X)$, then, since elements from $\mathbb{G}$ and $\mathbb{G}_T$ have an unique encoding, $\pi_i = \pi_{i-1}^{(\alpha_i)^{H_i}}$ is the unique group element fulfilling Equation 8. By induction, π_i is therefore uniquely defined for all $n \in [n]$. Analogously, π_{n+1} is uniquely defined by the first part of Equation 9. Finally, since π_{n+1} is uniquely defined, so is Y by the second part of Equation 9.

Pseudorandomness. We will prove the *pseudorandomness* of our construction based on the q-decisional Diffie-Hellman assumption, also used when proving Pseudorandomness of Jager's VRF [26], with small $q = \lceil \log(4t_{\mathcal{A}}(2t_{\mathcal{A}} - 1)/\epsilon_{\mathcal{A}})) \rceil = \mathcal{O}(\log k)$.

Definition 11. *For a bilinear group generator* GrpGen*, let* $G_{\mathcal{B}}^{q\text{-}\mathsf{DDH}}(k)$ *be the following game. The experiment runs* $\Pi \overset{\$}{\leftarrow} \mathsf{GrpGen}(1^k)$*, samples* $g, h \overset{\$}{\leftarrow} \mathbb{G}$*,* $x \overset{\$}{\leftarrow} \mathbb{Z}_{|\mathbb{G}|}$ *and* $b \overset{\$}{\leftarrow} \{0,1\}$*. Then it defines* $T_0 := e(g,h)^{x^{q+1}}$ *and* $T_1 \overset{\$}{\leftarrow} \mathbb{G}_T$*. Finally, it runs* $b' \overset{\$}{\leftarrow} \mathcal{B}(g, g^x, \ldots, g^{x^q}, h, T_b)$*, and outputs* 1 *if* $b = b'$*, and* 0 *otherwise. We denote with*

$$\mathsf{Adv}_{\mathcal{B}}^{q\text{-}\mathsf{DDH}}(k) := \Pr\left[G_{\mathcal{B}}^{q\text{-}\mathsf{DDH}}(k) = 1\right] - 1/2$$

the advantage *of* $\mathcal{B}$ *in breaking the* q-DDH*-assumption for groups generated by* GrpGen*, where the probability is taken over the randomness of the challenger and* $\mathcal{B}$*. We say that the* q-DDH*-assumption holds relative to* GrpGen*, if* $\mathsf{Adv}_{\mathcal{B}}^{q\text{-}\mathsf{DDH}}(k)$ *is negligible in* k *for all PPT algorithms* $\mathcal{B}$*.*

Note that the q-DDH assumption is trivially implied by the decisional $q+1$ Bilinear Diffie-Hellman Exponent assumption introduced in [10].

Theorem 5. *If* $\mathcal{VF}$ *is instantiated with the family of computational admissible hash functions from Theorem 4, then for any legitimate attacker* $\mathcal{A}$ *that breaks the* pseudorandomness *of* $\mathcal{VF}$ *in time* $t_{\mathcal{A}}$ *with advantage* $\epsilon_{\mathcal{A}} := \mathsf{Adv}_{\mathcal{A}}^{\mathsf{VRF}}(k)$*, there exists an algorithm* $\mathcal{B}$ *that, given (sufficiently close approximations of)* $t_{\mathcal{A}}$ *and* $\epsilon_{\mathcal{A}}$*, breaks the* q-DDH *assumption with* $q = \lceil \log(4t_{\mathcal{A}}(2t_{\mathcal{A}} - 1)/\epsilon_{\mathcal{A}}) \rceil$ *in time* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ *and with advantage*

$$\mathsf{Adv}_{\mathcal{B}}^{q\text{-}\mathsf{DDH}}(k) \geq \epsilon_{\mathcal{A}}^2/(32t_{\mathcal{A}}^2 - 16t_{\mathcal{A}}).$$

Remarkably, the proof of Theorem 5 is significantly simpler than the corresponding AHF-based proof from [26].

5.3 Proof of Theorem 5

We prove the theorem with a sequence of games. In the sequel let us write E_i to denote the event that Game i outputs "1".

```
$\underline{\mathsf{Finalize}_1(b'):}$
$K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$
If $b = b'$ then return 1
Return 0

$\underline{\mathsf{Finalize}_2(b'):}$
$K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$
$\boxed{\text{If } \mathsf{coll} \text{ then}}$
    $\boxed{b^* \xleftarrow{\$} \{0,1\}}$
    $\boxed{\text{Return } b^*}$
If $b = b'$ then return 1
Return 0

$\underline{\mathsf{Finalize}_3(b'):}$
$K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$
$\boxed{\text{If } \mathsf{coll} \text{ or } \mathsf{badchal} \text{ then}}$
    $b^* \xleftarrow{\$} \{0,1\}$
    Return $b^*$
If $b = b'$ then return 1
Return 0

$\underline{\mathsf{Finalize}_4(b'):}$
$K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$
$\boxed{\text{If } \mathsf{bad} \text{ then}}$
    $b^* \xleftarrow{\$} \{0,1\}$
    Return $b^*$
If $b = b'$ then return 1
Return 0
```

Fig. 2. Procedures used in the proof of Theorem 5. New or modified statements are highlighted in boxes.

Game 0. This is the original VRF security game, as described in Definition 8. By definition, we have

$$\Pr[E_0] = 1/2 + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{VRF}}(k)$$

Game 1. Recall that Theorem 5 assumes knowledge of (sufficiently close approximations of) the running time $t_{\mathcal{A}}$ and the advantage $\epsilon_{\mathcal{A}}$. In this game we replace the $\mathsf{Finalize}$ procedure with $\mathsf{Finalize}_1$, which additionally uses $t_{\mathcal{A}}$ and $\epsilon_{\mathcal{A}}$ by running $K \xleftarrow{\$} \mathsf{AdmSmp}(t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$, as depicted in Figure 1. Note that K and H now define the function $F_{K,H}(X)$ from Equation 6 and events coll and $\mathsf{badchal}$

as

$$\begin{aligned}\mathsf{badchal} &\iff F_{K,H}(X^*) \neq 0\\ \mathsf{coll} &\iff \exists\, i,j \text{ with } X^{(i)} \neq X^{(j)} \text{ s.t.}\\ &\qquad \forall\, \ell \in [n] : H(X^{(i)})_\ell = H(X^{(j)})_\ell \vee K_\ell = \bot\end{aligned}$$

like in Definition 6. Note that we denote with $X^{(1)}, \ldots, X^{(Q)}$ the values queried by $\mathcal{A}$ to $\mathsf{Evaluate}$, and with X^* the value queried to $\mathsf{Challenge}$. These modifications are purely conceptual and perfectly hidden from $\mathcal{A}$, such that we have

$$\Pr[E_1] = \Pr[E_0].$$

Game 2. This game proceeds identically to Game 1, except that the challenger aborts if event coll occurs. We formally express this change by replacing $\mathsf{Finalize}_1$ with $\mathsf{Finalize}_2$ from Figure 1. By applying Shoup's Difference Lemma [42], we get

$$\Pr[E_2] \geq \Pr[E_1] - \Pr[\mathsf{coll}].$$

Game 3. This game proceeds identically to Game 2, except that we replace $\mathsf{Finalize}_2$ with $\mathsf{Finalize}_3$, which outputs a random bit if $\mathsf{badchal}$ occurs. We have

$$\begin{aligned}\Pr[E_3] &= \Pr[E_3 \wedge \mathsf{badchal}] + \Pr[E_3 \wedge \neg\,\mathsf{badchal}]\\ &= \Pr[E_3 \mid \mathsf{badchal}]\,(1 - \Pr[\neg\,\mathsf{badchal})]) + \Pr[E_3 \mid \neg\,\mathsf{badchal}]\Pr[\neg\,\mathsf{badchal}]\\ &= 1/2 + \Pr[\neg\,\mathsf{badchal}]\,(\Pr[E_3 \mid \neg\,\mathsf{badchal}] - 1/2)\\ &= 1/2 + \Pr[\neg\,\mathsf{badchal}]\,(\Pr[E_2 \mid \neg\,\mathsf{badchal}] - 1/2)\\ &= 1/2 + \Pr[\neg\,\mathsf{badchal}]\,(\Pr[E_2] - 1/2)\end{aligned}$$

The third equality uses that $\Pr[E_3 \mid \mathsf{badchal}] = 1/2$, since a random bit is returned if $\mathsf{badchal}$ occurs, the fourth uses that by definition of the games it holds that $\Pr[E_3 \mid \neg\,\mathsf{badchal}] = \Pr[E_2 \mid \neg\,\mathsf{badchal}]$, and the last uses $\Pr[E_2 \mid \neg\,\mathsf{badchal}] = \Pr[E_2]$, since Game 2 is independent of $\mathsf{badchal}$. This is because K is only sampled after $\mathcal{A}$ made all its queries and stated its challenge and K is therefore perfectly hidden from $\mathcal{A}$.

Game 4. We replace events $\mathsf{badchal}$ and coll with an equivalent event, in order to simplify the construction of adversary $\mathcal{B}$. We let bad denote the event that $\mathsf{coll} \vee \mathsf{badchal}$. In Game 3 the experiment outputs a random bit $b^* \overset{\$}{\leftarrow} \{0,1\}$ if ($\mathsf{badchal} \vee \mathsf{coll}$) occurs. Now we output a random bit if bad occurs, which is equivalent. Formally, we achieve this by replacing $\mathsf{Finalize}_3$ with $\mathsf{Finalize}_4$ as defined in Figure 1, and get

$$\Pr[E_4] = \Pr[E_3]$$

Summing up probabilities from Game 0 to Game 4, we get

$$\begin{aligned}\Pr[E_4] &\geq 1/2 + \Pr[\neg\,\mathsf{badchal}]\left(\Pr[E_0] - 1/2 - \Pr[\mathsf{coll}]\right) \\ &= 1/2 + \Pr[\neg\,\mathsf{badchal}]\left(\epsilon_{\mathcal{A}} - \Pr[\mathsf{coll}]\right) \\ &\geq 1/2 + \tau(k) \end{aligned} \tag{10}$$

for some non-negligible function $\tau(k)$, where the last inequality is due to the definition of cAHFs (see Equation 7).

*Reduction from the q-*DDH *assumption.* Now we are ready to describe our algorithm $\mathcal{B}$ that solves the q-DDH problem by *perfectly* simulating Game 4 for adversary $\mathcal{A}$. When instantiated with the computational AHF from Theorem 4, a q-DDH instance with $q = \lceil \log(4t_{\mathcal{A}}(2t_{\mathcal{A}} - 1)/\epsilon_{\mathcal{A}} \rceil$ is sufficient.

The only minor difference between Game 4 and the simulation by $\mathcal{B}$ is that $\mathcal{B}$ aborts "as early as possible". That is, it samples the AHF key $K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$ and random bit b^* already in the Initialize procedure, and checks whether bad occurs after each Evaluate or Challenge query of $\mathcal{A}$. If bad occurs, then it immediately outputs b^*, rather than waiting for the adversary to query Finalize. Obviously, this does not modify the probability of E_4. We proceed by describing $\mathcal{B}$.

Description of algorithm $\mathcal{B}$. The input of $\mathcal{B}$ is the q-DDH-challenge $(\tilde{g}, \tilde{g}^x, \ldots, \tilde{g}^{x^q}, h, T)$. $\mathcal{B}$ first tests $\tilde{g}^x \overset{?}{=} 1$. Since $\tilde{g}$ is a generator of $\mathbb{G}$, this holds if and only if $x = 0$. In this case, $\mathcal{B}$ tests $e(1, h) \overset{?}{=} T$ and outputs 0 if the statement is true and 1 otherwise. Obviously, this is the correct solution to the q-DDH-challenge if $x = 0$. We therefore assume $x \neq 0$ for the remainder of the proof.

Recall that $\mathcal{A}$ is playing the pseudorandomness game for the VRF from Definition 8, meaning it tries to distinguish a VRF output from a random value with the help of Initialize, Evaluate, Challenge and Finalize. $\mathcal{B}$ simulates these operations by executing the corresponding procedures from Figure 3. Finally, $\mathcal{B}$ outputs either the random bit b^* if event bad occurs, or otherwise whatever Finalize returns. In Figure 3, we denote the number of positions in K that are not 1 by $\sigma(K)$.

Initialization. The values $(\tilde{g}, h, \tilde{g}^x)$ in Initialize are from the q-DDH-challenge. $\mathcal{B}$ computes the g_i-values exactly as in the original Gen-algorithm for most $i \in [n]$ by choosing $\alpha_i \xleftarrow{\$} \mathbb{Z}_{|\mathbb{G}|}$ and setting $g_i := g^{\alpha_i}$, but with the exception that

$$g_i := \begin{cases} g^{\alpha_i + 1/x} & \text{if } K_i = 0 \\ g^{\alpha_i + x} & \text{if } K_i = 1 \end{cases}$$

for all $i \in [n]$ with $K_i \neq \perp$. Note that due to our choice of a cAHF in Theorem 5, we have that $\sigma(K) \leq q$. Therefore, $\mathcal{B}$ can compute $g^{\alpha_0 + x^{q-\sigma(K)-1}}$ since $q - \sigma(K) - 1$ is at least -1 and at most $q - 1$, and $g^{1/x} = \tilde{g}, \ldots, g^{x^{q-1}} = \tilde{g}^{x^q}$ are part of the q-DDH-challenge. $\mathcal{B}$ can compute $g^{\alpha_i + 1/x}$ for the same reason. All g_i-values are distributed correctly.

```
Initialize:
$K \xleftarrow{\$} \mathsf{AdmSmp}(1^k, t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$
$H \xleftarrow{\$} \mathcal{H}$
$b^* \xleftarrow{\$} \{0,1\}$
$g := \tilde{g}^x$
For $i \in [n+1]_0$ do
    $\alpha_i =\xleftarrow{\$} \mathbb{Z}_{|\mathbb{G}|}$
$g_0 := g^{\alpha_0 + x^{q-\sigma(K)-1}}$
$g_{n+1} := g^{\alpha_{n+1}+x}$
For $i \in [n]$ do
    If $K_i = 0$ then $g_i := g^{\alpha_i + 1/x}$
    If $K_i = 1$ then $g_i := g^{\alpha_i + x}$
    If $K_i = \perp$ then $g_i := g^{\alpha_i}$
$\mathsf{vk} := \big(H, g, h, \{g_i\}_{i \in [n+1]_0}\big)$
Return $\mathsf{vk}$
```

```
Evaluate($X$):
If $F_{K,H}(X) = 0$ or $\mathsf{coll}$ then output $b^*$
$Y := e(g^{P_{w,X}(x)}, h)$
For $j \in [n]$ do
    $\pi_j := g^{P_{w,X}(x)}$
$\pi := (\pi_1, \ldots, \pi_n)$
Return $(Y, \pi)$
```

```
Challenge($X^*$):
If $F_{K,H}(X) = 1$ then output $b^*$
Compute $\gamma_0, \ldots, \gamma_{q+1}$ s.t.
    $P_{w,X}(x) = \sum_{i=0}^{q+1} \gamma_i x^i$
$Y^* := T^{\gamma_{q+1}} \cdot \prod_{i=1}^{q} e((g^{x^i})^{\gamma_i}, h)$
Return $Y^*$
```

```
Finalize$^{\mathsf{VRF}}(b')$:
Return $b'$
```

Fig. 3. Procedures for the simulation of the VRF pseudorandomness experiment by $\mathcal{B}$.

Helping definitions. To explain how $\mathcal{B}$ responds to Evaluate and Challenge queries made by $\mathcal{A}$, we define four sets $I_{w,X}, I^0_{w,X}, I^1_{w,X}$ and $I^\perp_{w,X}$, which depend on a VRF input $X \in \{0,1\}^*$ and an integer $w \in [n]$. $I_{w,X} \subseteq [w] \subseteq [n]$ is the set of all indices such that $H(X)_i = 1$. The other sets partition $I_{w,X}$ in the respective subsets of indices where K_i is $0, 1$ or $\perp$. Formally, the sets are defined as follows.

$$\begin{aligned}
I_{w,X} &:= \{i \in [w] : H(X)_i = 1\},\\
I^0_{w,X} &:= \{i \in [w] : H(X)_i = 1 \wedge K_i = 0\},\\
I^1_{w,X} &:= \{i \in [w] : H(X)_i = 1 \wedge K_i = 1\} \text{ and}\\
I^\perp_{w,X} &:= \{i \in [w] : H(X)_i = 1 \wedge K_i = \perp\},
\end{aligned}$$

where K and H are from $\mathcal{B}$'s choice in Initialize in Figure 3. Note that $I_{w,X} = I^0_{w,X} \cup I^1_{w,X} \cup I^\perp_{w,X}$. Based on these sets, we define polynomials $P_{w,X}(x)$. We let $P_{0,X}(x) := \alpha_0 + x^{q-\sigma(K)-1}$ and define

$$P_{w,X}(x) = \begin{cases} P_{w-1,X}(x) \cdot (\alpha_i + 1/x) & \text{if } w \in I^0_{w,X},\\ P_{w-1,X}(x) \cdot (\alpha_i + x) & \text{if } w \in I^1_{w,X},\\ P_{w-1,X}(x) \cdot \alpha_i & \text{if } w \in I^\perp_{w,X} \text{ and}\\ P_{w-1,X}(x) & \text{otherwise,} \end{cases}$$

for $w \in [n]$ and $X \in \{0,1\}^*$. Finally, we define $P_{n+1,X}(x) := P_{n,X}(x) \cdot (\alpha_{n+1} + x)$. We note some important properties of these helping definitions in the following lemma.

Lemma 2. *Let $I^0_{w,X}$ and $I^1_{w,X}$ be as above, then*

$$\left|I^0_{w,X}\right| \leq q - \sigma(K) \qquad \text{and} \qquad \left|I^1_{w,X}\right| \leq \sigma(K)$$

holds for all $w \in [n]$. Furthermore, define $P_{w,X}$ as above, then

$$\deg(P_{w,X}) = q - \sigma(K) - 1 - \left|I^0_{w,X}\right| + \left|I^1_{w,X}\right|$$

holds for all $w \in [n]$, where deg *denotes the degree of a polynomial. In particular, we have*

$$-1 \leq \deg(P_{w,X}) \leq q - 1$$

for all $w \in [n]$. Note that we simplify the notation by writing $\deg(f) = -k$ *for a rational function f such that $1/f$ is a polynomial of degree $k \in \mathbb{N}$.*

We postpone the proof of Lemma 2 to Section 5.4 as to not interrupt the proof of Theorem 5. Similarly to [26], we observe the following.

1. For all X with $F_{K,H}(X) = 1$ and $w \in [n+1]_0$, we have $-1 \leq \deg(P_{w,X}(x)) \leq q-1$. Note that in contrast to Lemma 2, the bound also holds for $w = n+1$. For this, observe that for all X with $F_{K,H}(X) = 1$, there is an $i \in [n]$ such that $K_i \neq \perp$ and $K_i \neq H(X)_i$. Therefore, at least one of the following conditions hold.

   $$\left|I^0_{w,X}\right| > 0 \text{ for all } w \geq i \qquad \left|I^1_{w,X}\right| < q - \sigma(K) \text{ for all } w \in [n]$$

   By Lemma 2, we therefore have $\deg(P_{n,X}) \leq q-2$ implying $\deg(P_{n+1,X}) \leq q-1$. Hence, $\mathcal{B}$ can efficiently compute $g^{P_{w,X}(x)} = \tilde{g}^{x \cdot P_{w,X}(x)}$ for all $w \in [n+1]_0$. To this end, $\mathcal{B}$ first computes the coefficients $\gamma_0, \ldots, \gamma_q$ of the polynomial $x \cdot P_{w,X}(x) = \sum_{i=0}^{q} \gamma_i x^i$ with degree at most q, and then

   $$\tilde{g}^{x \cdot P_{w,X}(x)} := \tilde{g}^{\sum_{i=0}^{q} \gamma_i x^i} = \prod_{i=0}^{q} (\tilde{g}^{x^i})^{\gamma_i}$$

   using the terms $(\tilde{g}, \tilde{g}^x, \ldots, \tilde{g}^{x^q})$ from the q-DDH challenge.
2. If $F_{K,H}(X) = 0$, then $H(X)_i = K_i$ for all $i \in [n]$ where $K_i \neq \perp$ and thus $\left|I^0_{n,X}\right| = 0$ and $\left|I^1_{n,X}\right| = \sigma(K)$ hold. By Lemma 2, $x \cdot P_{n+1,X}(x)$ therefore has degree $q+1$. We do not know how $\mathcal{B}$ can efficiently compute $g^{P_{n+1,X}(x)} = \tilde{g}^{x \cdot P_{n+1,X}(x)}$ in this case.

Responding to $\mathsf{Evaluate}$*-queries.* As depicted in Figure 3, if $F_{K,H}(X^{(i)}) = 1$, then procedure $\mathsf{Evaluate}$ can compute the group elements $g^{P_{w,X}(x)}$ as explained above. Note that in this case the response to the $\mathsf{Evaluate}(X^{(i)})$-query of $\mathcal{A}$ is correct. Moreover, $\mathcal{B}$ outputs the random bit b^* and aborts if there was an earlier query $X^{(j)}$ to $\mathsf{Evaluate}$ such that the first q positions of $H(X^{(j)})$ and $H(X^{(i)})$ match, meaning event coll occurred. $\mathcal{B}$ also outputs the random bit b^* if $F_{K,H}(X) = 0$ since it implies the abort condition bad. This is because if $F_{K,H}(X) = 0$, then throughout the experiment either coll or $\mathsf{badchal}$ *must occur*: For $\mathsf{badchal}$ not to occur, we require $F_{K,H}(X^*) = 0$, but this implies that $H(X^{(i)})$ and $H(X^*)$ match on the first q positions, causing event coll.

Responding to the Challenge-*query.* If $F_{K,H}(X^*) = 0$, then procedure Challenge computes

$$Y^* := T^{\gamma_{q+1}} \cdot \prod_{i=0}^{q} e((\tilde{g}^{x^i})^{\gamma_i}, h) = T^{\gamma_{q+1}} \cdot e(\tilde{g}^{\sum_{i=0}^{q} \gamma_i x^i}, h)$$

where $\gamma_0, \ldots, \gamma_{q+1}$ are the coefficients of the degree-$(q+1)$-polynomial $x \cdot P_{n+1,X^*}(x) = \sum_{i=0}^{q+1} \gamma_i x^i$. Note that if $T = e(\tilde{g}, h)^{x^{q+1}}$, then it holds that $Y^* = V_{\mathsf{sk}}(X^*)$. Moreover, if T is uniformly random, then so is Y^*. If $F_{K,H}(X^*) = 1$, then $\mathcal{B}$ outputs the random bit b^* and aborts.

$\mathcal{B}$'s running time. The running time $t_{\mathcal{B}}$ of $\mathcal{B}$ consists of the running time $t_{\mathcal{A}}$ of $\mathcal{A}$ plus the time required to simulate the oracles as depicted in Figure 3. The latter step essentially consists of the operations defined in the construction of the VRF in Section 5.2 plus minor operations like sampling K, evaluating $F_{K,H}$, checking for collisions, calculate the γ_i to compute $g^{P_{w,X}}(x)$ and some group operations to compute the g_i. Thus, we have $t_{\mathcal{B}} \approx t_{\mathcal{A}}$.

$\mathcal{B}$'s success probability. Let $c \in \{0, 1\}$ denote the random bit chosen by the q-DDH challenger. $\mathcal{B}$ perfectly simulates $G_{\mathsf{VRF}}^{\mathcal{A}}(k)$ with $b = c$. Hence, by Equation 10, we get

$$\mathsf{Adv}_{\mathcal{B}}^{q\text{-}\mathsf{DDH}}(k) \geq \Pr[E_4] \geq 1/2 + \tau(k)$$

for a non-negligible function τ. In particular, when instantiated concretely with the computational AHF from Theorem 4, then we have

$$\mathsf{Adv}_{\mathcal{B}}^{q\text{-}\mathsf{DDH}}(k) \geq 1/2 + \epsilon_{\mathcal{A}}^2/(32t_{\mathcal{A}}^2 - 16t_{\mathcal{A}}).$$

5.4 Proof of Lemma 2

Observe that by the Definition of AdmSmp in Theorem 4 and the choice of q in Theorem 5 we have $\sigma(K) \leq q$. Furthermore, $I_{n,X}^1$ contains up to one element for each $i \in [n]$ such that $K_i = 1$, which are at most $\sigma(K)$ many. Analogously, $I_{w,X}^0$ can contain at most $q - \sigma(K)$ elements since K has only $q - \sigma(K)$ positions where it is 0. We proceed by proving that

$$\deg(P_{w,X}) = q - \sigma(K) - 1 - \left|I_{K,w,X}^0\right| + \left|I_{K,w,X}^1\right| \tag{11}$$

holds for all $w \in [n]$. Observe that in the definition of $P_{w,X}$ each $i \in I_{w,X}^0$ adds a factor $(\alpha_i + 1/x)$ to $P_{w-1,X}$ and by that decreases the degree of the polynomial by one. Analogously, each element in $I_{w,X}^1$ increases the degree of the polynomial by one. Finally, $P_{0,X}$ is defined as $(\alpha_0 + x^{q-\sigma(K)-1})$ yielding, together with the upper bounds on $|I_{w,X}|^0$ and $|I_{w,X}|^1$ above, Equation 11.
We finish the proof of Lemma 2, by showing that

$$-1 \leq \deg(P_{w,X}) \leq q - 1$$

holds for all $w \in [n]$. By Equation 11 and $|I_{w,X}| \leq \sigma(K)$, the following holds.

$$\begin{aligned}\deg(P_{w,X}) &= q - \sigma(K) - 1 - \left|I^0_{w,X}\right| + \left|I^1_{w,X}\right| \\ &\leq q - \sigma(K) - 1 + \left|I^1_{w,X}\right| \\ &\leq q - \sigma(K) - 1 + \sigma(K) = q - 1.\end{aligned}$$

We conclude the lower bound analogously by applying Equation 11 and $|I_{w,X}| \leq q - \sigma(K)$.

$$\begin{aligned}\deg(P_{w,X}) &= q - \sigma(K) - 1 - \left|I^0_{w,X}\right| + \left|I^1_{w,X}\right| \\ &\geq q - \sigma(K) - 1 - \left|I^0_{w,X}\right| \\ &\geq q - \sigma(K) - 1 - q - \sigma(K) = -1\end{aligned}$$

6 Comparison of VRF Instantiations

We compare key and proof sizes of the VRFs from Katsumata [29], Yamada [50], Jager [26] and our VRF from Section 5.3 in two different types of instantiation: using bAHFs with ECCs and using cAHFs with TCRHFs from Section 3.1. We do not compare Yamada's third VRF, because it relies on a much stronger polynomial q-type assumption [50, Appendix C (in the eprint version)]. Comparing the concrete number of group elements in keys and proofs shows that instantiating the VRFs with cAHFs *significantly reduces the key and proof sizes.* Furthermore, comparing our new VRF in Section 5.2 to any of the other VRFs shows that our new VRF has *either significantly smaller secret keys, verification keys or proofs than each other VRF.*

Formulas for key and proof sizes. Table 1 shows the sizes of the verification keys $|\mathsf{vk}|$, secret keys $|\mathsf{sk}|$ and proofs $|\pi|$ as the number of group elements they contain in dependence of k, Q, ϵ, δ and t. The caption precisely explains how the key and proof sizes relate to these variables. Furthermore, the table shows the advantage of the solver $\mathrm{Adv}_{\mathcal{B}}$ against the respective hard problem. For the VRFs of of [29] and [50], this is the q-DBDHI assumption introduced by Boneh and Boyen [8]. For our VRF and Jager's VRF [26] this is the q-DDH assumption. Even though the assumptions differ, the key and proof sizes are comparable for the following reasons.

1. Cheons algorithm [16] is the most efficient known generic algorithm to solve both, the q-DDH assumption and the q-DBDHI assumption.
2. Except for Yamada's VRFs [50], all schemes considered share almost the same q in the assumption. Also, Yamada's VRF relies on a $\mathcal{O}(k\log(k))$-DBDHI assumption [50], and is therefore reasonably close the q of the other schemes. Hence, Yamada's VRFs would only need to use slightly larger groups to compensate the stronger assumption.

Construction	Instantiation	$\vert\mathsf{vk}\vert$ $\#\mathbb{G}$	$\vert\mathsf{sk}\vert$ $\#\mathbb{G}$	$\vert\pi\vert$ $\#\mathbb{G}$	$\mathrm{Adv}_{\mathcal{B}}$
[29] Sec. 5.1	ECCs	$3 + \zeta d$	$d\zeta + 1$	$d + dn + \zeta + 1$	$\tau + \mathsf{stat}$
	cAHF	$3 + \zeta^{\mathsf{tcrh}} j$	$\zeta^{\mathsf{tcrh}} + 1$	$j + j(n^{\mathsf{tcrh}}) + \zeta^{\mathsf{tcrh}} + 1$	$\tau^{\mathsf{tcrh}} + \mathsf{stat}$
[29] Sec. 5.3	ECCs	$3 + d(2^{\zeta/2+2} - 2)$	$d\zeta + 1$	$2d - 1$	$\tau + \mathsf{stat}$
	cAHF	$3 + j(2^{\zeta^{\mathsf{tcrh}}/2+2} - 2)$	$j\zeta^{\mathsf{tcrh}} + 1$	$2j - 1$	$\tau^{\mathsf{tcrh}} + \mathsf{stat}$
[50] Sec. 6.1	ECCs	$dn_1 + 2$	d	dn_2	$\tau + \mathsf{stat}$
	cAHF	$jn_1^{\mathsf{tcrh}} + 2$	j	jn_2^{tcrh}	$\tau^{\mathsf{tcrh}} + \mathsf{stat}$
[50] Sec. 6.3	ECCs	$d + 2$	d	$d(n_1 + n_2 - 1)$	$\tau + \mathsf{stat}$
	cAHF	$j + 2$	j	$j(n_1^{\mathsf{tcrh}} + n_2^{\mathsf{tcrh}} - 1)$	$\tau^{\mathsf{tcrh}} + \mathsf{stat}$
[26]	ECCs	$2n + 2$	$2n$	n	τ
	cAHFs	$2n^{\mathsf{tcrh}} + 2$	$2n^{\mathsf{tcrh}}$	n^{tcrh}	τ^{tcrh}
Our construction	ECCs	$n + 4$	$n + 2$	$n + 1$	τ
	cAHFs (as shown)	$n^{\mathsf{tcrh}} + 4$	$n^{\mathsf{tcrh}} + 2$	$n^{\mathsf{tcrh}} + 1$	τ^{tcrh}

Table 1. The sizes of vk, sk, π as the number of group elements and the advantage of the solver in the security proof for the instantiation of our VRF and the VRFS of Jager [26], Yamada [50] and Katsumata [29]. $d = \lfloor (2Q + Q/\epsilon)/\log(1-\delta) \rfloor$ is the number of positions in the key of the bAHF that are not $\perp$. Note that [29,26,50] all chose d in this way. For the VRFs from Katsumata [29], $\zeta = \lfloor \log(2n) \rfloor + 1$ is the number of bits required to encode an element from $[2n]$. τ is as in Definition 3 and describes the advantage of a solver against the underlying q-type assumption with $q := d$ for the instantiation using ECCs and $q := j$ using TCRHFs. stat represents statistically negligible values introduced in the security proofs of Yamada's and Katsumata's VRFs [50,29]. Note that for Yamada's VRFs [50], n_1, $n_2 \in \mathbb{N}$ can be chosen freely such that $n = n_1 n_2$. Analogously, we have $n^{\mathsf{tcrh}} := 2k + 3$ as the output length of the truncation collision resistant hash function, ζ^{tcrh} is the number of bits required to encode an element from $[n^{\mathsf{tcrh}}]$. As in Theorem 4 the length of the prefix used for the TCRHFs is $j := \lceil 4t(2t-1)/\epsilon \rceil$. Again, $n_1^{\mathsf{tcrh}}, n_2^{\mathsf{tcrh}} \in \mathbb{N}$ can be chosen freely such that $n^{\mathsf{tcrh}} = n_1^{\mathsf{tcrh}} n_2^{\mathsf{tcrh}}$. Finally, we have $\tau^{\mathsf{tcrh}} = \epsilon^2/(32t^2 - 16t)$ from Theorem 4.

Instantiation choices. The concrete number of group elements in keys and proofs depends on some instantiation choices, which we describe here. Since the VRF instantiation with cAHFs takes inputs from $\{0,1\}^*$, we level the playing field by assuming that the instantiations using ECCs first hash the inputs with a collision resistant hash function $H : \{0,1\}^* \to \{0,1\}^\alpha$ and thus the VRFs also take inputs from $\{0,1\}^*$. We let $\alpha = 2k$, where k is the security parameter, to ensure the collision resistance of H against birthday attacks. Hence, when an ECC C is used in an instantiation, then C takes inputs from $\{0,1\}^{2k}$. We list the key sizes of the different VRFs, instantiated with cAHFs and with ECCs in Table 2.

Unfortunately, assessing the potential efficiency of the instantiation using bAHFs with ECCs is only possible to a limited degree because finding the best known ECC for a given input length and relative minimal distance is non-trivial for larger numbers. Code tables [22], the to the best of our knowledge most extensive collection of best known codes for different parameters, only lists binary codes of length up to 256, which is to small for reasonable security parameters. Therefore, we compare the instantiation with cAHFs and TCRHFs to the instantiation with bAHFs the following ECCs.

- We consider primitive BCH codes that we puncture to achieve the desired relative minimal distance. Again, tables in for example [41, Table 9.1] only list codes for lengths up to 1023. We therefore wrote a small program that finds the most suited primitive BCH code for this purpose. It can be found at `https://github.com/DavidNiehues/bch-code-search`. The program considers the Bose distance of the BCH codes instead of the design distance. The caption of Table 2 states the used primitive BCH code explicitly.
- Furthermore, we present key and proof sizes under the assumptions that ECCs on the GV and MRRW bound can be efficiently instantiated.

Assuming instantiations with ECCs on the GV and MRRW bound gives the instantiation with bAHFs and ECCs an advantage over instantiations using cAHFs with TCRHFs, since ECCs on the MRRW-bound are the best theoretically possible ECCs and it is not known whether ECCs on the GV-bound can be constructed efficiently.

In the calculation of the key sizes of Yamada's VRFs [50], we pick $n_1 = n_2$ as $\lceil \sqrt{n} \rceil$ in order to make the parameter sizes comparable. We make this choice, because Yamada suggest picking n_1 and n_2 close to $\sqrt{n}$ [50] and because choosing actual divisors of n would make key and proof sizes heavily depend on the factorization of n. This would lead to misleading results. Furthermore, we chose δ such that all instantiations achieve the same advantage of the solver. This makes the different instantiations comparable. Note that we did not incorporate the statistically negligible terms stat in the calculation of the advantage.

Smaller keys and proofs using cAHFs. Table 2 shows the concrete number of group elements of the different instantiations in the setting with $k = 128$, $Q = 2^{25}$, $t = 2^{50}$ and $\epsilon = 2^{-25}$. The instantiation of the VRFs with cAHFs improves the size of keys and proofs significantly, even compared to bAHFs *instantiated with the best theoretically possible ECCs* on the MRRW bound. Concretely, using cAHFs with TCRHFs instead of bAHFs with ECCs on the MRRW bound *reduces the size of the proofs of the VRF in Section 5.1 in [29] by* $\approx 61\%$ in the setting of Table 2. Compared to an instantiation with ECCs on the GV bound, we reduce the proof size by even $\approx 78\%$. Compared to the instanitation with punctured primitive BCH codes, the improvement is even $\approx 87\%$. Particularly, keys and proofs whose size depends linearly on n shrink when the VRFs are instantiated with cAHFs. Over all key and proof sizes affected by the improvement, the reduction amounts for at least 9% of the size of an instantiation with ECCs on the MRRW bound. Note that the size of all keys and proofs stays at least the same. Hence, by making an additional (but from a practical point of view plausible and natural) hardness assumption, we can reduce the key and proof sizes significantly, which may be useful for many practical applications of VRF. Furthermore, it hints at the usefulness of cAHFs for other primitives.

Efficiency of our new VRF. Table 2 shows that our new VRF nearly halves the size of the verification and secret key of Jager's VRF while maintaining the proof size. Comparing our VRF with the VRFs of Katsumata and Yamada shows that

Construction	Instantiation	$\lvert \mathsf{vk} \rvert$ $\#\mathbb{G}$	$\lvert \mathsf{sk} \rvert$ $\#\mathbb{G}$	$\lvert \pi \rvert \#\mathbb{G}$	$\mathrm{Adv}_{\mathcal{B}}$
[29] Sec. 5.1	BCH	1551	1549	261883	$\approx 2^{-155}$
	GV	1551	1549	154942	$\approx 2^{-155}$
	MRRW	1422	1420	85797	$\approx 2^{-155}$
	cAHF	1283	1281	33291	$\approx 2^{-155}$
[29] Sec. 5.3	BCH	32769	1549	257	$\approx 2^{-155}$
	GV	32769	1549	257	$\approx 2^{-155}$
	MRRW	23094	1420	257	$\approx 2^{-155}$
	cAHF	16131	1281	255	$\approx 2^{-155}$
[50] Sec. 6.1	BCH	5936	129	5934	$\approx 2^{-155}$
	GV	4517	129	4515	$\approx 2^{-155}$
	MRRW	3356	129	3354	$\approx 2^{-155}$
	cAHF	2178	128	2176	$\approx 2^{-155}$
[50] Sec. 6.2	BCH	131	129	11739	$\approx 2^{-155}$
	GV	131	129	8901	$\approx 2^{-155}$
	MRRW	131	129	6579	$\approx 2^{-155}$
	cAHF	130	128	4224	$\approx 2^{-155}$
[26]	BCH	4060	4058	2029	$\approx 2^{-155}$
	GV	2402	2400	1200	$\approx 2^{-155}$
	MRRW	1330	1328	664	$\approx 2^{-155}$
	cAHF	520	518	259	$\approx 2^{-155}$
Our construction	BCH	2033	2031	2030	$\approx 2^{-155}$
	GV	1204	1202	1201	$\approx 2^{-155}$
	MRRW	668	666	665	$\approx 2^{-155}$
	cAHF	263	261	260	$\approx 2^{-155}$

Table 2. Key and proof sizes for $k = 128, Q = 2^{25}, t = 2^{50}, \epsilon = 2^{-25}$ and $\delta = 0.235$. In consequence, we have $d = 129$. Puncturing a primitive $[2047, 264, 495]$ BCH-code 18 times to a $[2029, 264, 477]$ code yields $n^{\mathsf{BCH}} = 2029, n_1^{\mathsf{BCH}} = 46, n_2^{\mathsf{BCH}} = 46$ and $\zeta^{\mathsf{BCH}} = 12$. If an ECC on the GV bound is used, this implies $n^{\mathsf{GV}} = 1200, n_1^{\mathsf{GV}} = 35, n_2^{\mathsf{GV}} = 35$ and $\zeta^{\mathsf{GV}} = 12$. Analogously, if an ECC on the MRRW bound is used, this implies $n^{\mathsf{MRRW}} = 664, n_1^{\mathsf{MRRW}} = 26, n_2^{\mathsf{MRRW}} = 26$ and $\zeta^{\mathsf{MRRW}} = 11$. Finally, if the VRFs are instantiated with a cAHF using TCRHFs, we have $n^{\mathsf{tcrh}} = 259, n_1^{\mathsf{tcrh}} = 17, n_2^{\mathsf{tcrh}} = 17, j = 128$ and $\zeta^{\mathsf{tcrh}} = 10$.

our VRF, in the worst case, has secret or verification keys about twice the size of the verification or secret keys of VRFs by Katsumata and Yamada. At the same time, when instantiated with cAHFs, our VRF always has either verification or secret keys with size only $\approx 20\%$ of the size of the respective keys of the VRFs of Yamada and Katsumata with the same instantiation. We provide more comparisons with different k, ϵ in Appendix A supporting both, the efficiency of cAHFs and our new VRF.

References

1. Michel Abdalla, Dario Fiore, and Vadim Lyubashevsky. From selective to full security: Semi-generic transformations in the standard model. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 316–333. Springer, Heidelberg, May 2012.
2. Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively single-key secure constrained prfs for NC1. *IACR Cryptology ePrint Archive*, 2018:1000, 2018.
3. Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Heidelberg, August 2018.

4. Mihir Bellare and Thomas Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 407–424. Springer, Heidelberg, April 2009.
5. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.
6. Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, Heidelberg, May 1996.
7. Nir Bitansky. Verifiable random functions from non-interactive witness-indistinguishable proofs. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 567–594. Springer, Heidelberg, November 2017.
8. Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223–238. Springer, Heidelberg, May 2004.
9. Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 443–459. Springer, Heidelberg, August 2004.
10. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, Heidelberg, May 2005.
11. Dan Boneh and Matthew K. Franklin. Identity based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.
12. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.
13. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
14. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology*, 25(4):601–639, October 2012.
15. Yu Chen, Qiong Huang, and Zongyang Zhang. Sakai-ohgishi-kasahara identity-based non-interactive key exchange revisited and more. *Int. J. Inf. Sec.*, 15(1):15–33, 2016.
16. Jung Hee Cheon. Security analysis of the strong Diffie-Hellman problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 1–11. Springer, Heidelberg, May / June 2006.
17. Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 475–485. IEEE Computer Society, 1999.
18. Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 513–530. Springer, Heidelberg, August 2013.
19. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017.

20. Edgar Nelson Gilbert. A comparison of signalling alphabets. *Bell System Technical Journal*, 31:504–522, 5 1952.
21. Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Vcelak. Verifiable random functions (vrfs). Internet-Draft draft-irtf-cfrg-vrf-05, IETF Secretariat, August 2019. `http://www.ietf.org/internet-drafts/draft-irtf-cfrg-vrf-05.txt`.
22. Markus Grassl. Bounds on the minimum distance of linear codes and quantum codes. Online available at `http://www.codetables.de`, 2007. Accessed on 2019-01-20.
23. Dennis Hofheinz and Tibor Jager. Verifiable random functions from standard assumptions. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 336–362. Springer, Heidelberg, January 2016.
24. Dennis Hofheinz, Tibor Jager, and Eike Kiltz. Short signatures from weaker assumptions. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 647–666. Springer, Heidelberg, December 2011.
25. Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. *Journal of Cryptology*, 25(3):484–527, July 2012.
26. Tibor Jager. Verifiable random functions from weaker assumptions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 121–143. Springer, Heidelberg, March 2015.
27. Tibor Jager and Rafael Kurek. Short digital signatures and ID-KEMs via truncation collision resistance. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 221–250. Springer, Heidelberg, December 2018.
28. Tibor Jager and David Niehues. On the real-world instantiability of admissible hash functions and efficient verifiable random functions. In *Selected Areas in Cryptography - SAC 2019 - 26th International Conference.*
29. Shuichi Katsumata. On the untapped potential of encoding predicates by arithmetic circuits and their applications. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 95–125. Springer, Heidelberg, December 2017.
30. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* Chapman and Hall/CRC Press, 2007.
31. Lisa Kohl. Hunting and gathering - verifiable random functions from standard assumptions with short proofs. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 408–437. Springer, Heidelberg, April 2019.
32. Benoît Libert, Damien Stehlé, and Radu Titiu. Adaptively secure distributed PRFs from LWE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 391–421. Springer, Heidelberg, November 2018.
33. Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 597–612. Springer, Heidelberg, August 2002.
34. Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes.* North Holland mathematical library. Amsterdam [u.a.] : North Holland, 10. impr. edition, 1998.
35. Robert J. McEliece, Eugene R. Rodemich, Howard Rumsey Jr., and Lloyd R. Welch. New upper bounds on the rate of a code via the delsarte-macwilliams inequalities. *IEEE Trans. Information Theory*, 23(2):157–166, 1977.
36. Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing key transparency to end users. In Jaeyeon

Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 383–398. USENIX Association, August 2015.
37. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.
38. National Institute of Standards and Technology. Fips pub 180–4: Secure hash standard, August 2015. `http://dx.doi.org/10.6028/NIST.FIPS.180-4`.
39. National Institute of Standards and Technology. Fips pub 202: Sha-3 standard: Permutation-based hash and extendable-output functions, August 2015. `http://dx.doi.org/10.6028/NIST.FIPS.202`.
40. Dimitrios Papadopoulos, Duane Wessels, Shumon Huque, Moni Naor, Jan Včelák, Leonid Reyzin, and Sharon Goldberg. Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Report 2017/099, 2017. `http://eprint.iacr.org/2017/099`.
41. William Wesley Peterson and Edward J. Weldon. *Error-correcting codes.* Cambridge, Massachusetts: MIT Press, 2. ed., 9. print. edition, 1988.
42. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/2004/332`.
43. Kenneth W. Shum, Ilia Aleshnikov, P. Vijay Kumar, Henning Stichtenoth, and Vinay Deolalikar. A low-complexity algorithm for the construction of algebraic-geometric codes better than the gilbert-varshamov bound. *IEEE Trans. Information Theory*, 47(6):2225–2241, 2001.
44. Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Trans. Information Theory*, 42(6):1710–1722, 1996.
45. Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 238–251. ACM, 2017.
46. Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Trans. Information Theory*, 43(6):1757–1766, 1997.
47. Rom Rubenovich Varshamov. Estimate of the number of signals in error correcting codes. *Dokl. Acad. Nauk SSSR*, 117(5):739–741, 1957.
48. Jan Vcelak, Sharon Goldberg, Dimitrios Papadopoulos, Shumon Huque, and David Lawrence. Nsec5, dnssec authenticated denial of existence. Internet-Draft draft-vcelak-nsec5-08, IETF Secretariat, December 2018. `https://www.ietf.org/archive/id/draft-vcelak-nsec5-08.txt`.
49. Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Heidelberg, May 2005.
50. Shota Yamada. Asymptotically compact adaptively secure lattice IBEs and verifiable random functions via generalized partitioning techniques. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 161–193. Springer, Heidelberg, August 2017.
51. Gilles Zémor. On expander codes. *IEEE Trans. Information Theory*, 47(2):835–837, 2001.

A Further Comparisons

We provide further comparisons like Table 2. The results we present are calculated using the formulas stated in Table 1. Compared to Table 2, we provide key and proof sizes for $k \in \{100, 128, 256\}$ and $\epsilon \in \{2^{-25}, 2^{-50}\}$. For every combination of k and ϵ, δ is chosen such that the advantages for the instantiation using ECCs and the instantiation using TCRHFs are approximately the same. All variables have the same semantics as in Section 6. The results show that using the cAHF instantiated with a TCRHF reduces key and proof sizes significantly.

Construction	Instantiation	$\lvert\mathsf{vk}\rvert\ \#\mathbb{G}$	$\lvert\mathsf{sk}\rvert\ \#\mathbb{G}$	$\lvert\pi\rvert\#\mathbb{G}$	$\mathrm{Adv}_{\mathcal{B}}$
[29] Sec. 5.1	BCH	2005	2003	567966	$\approx 2^{-205}$
	GV	1851	1849	225931	$\approx 2^{-205}$
	MRRW	1697	1695	110892	$\approx 2^{-205}$
	cAHF	1380	1378	31222	$\approx 2^{-205}$
[29] Sec. 5.3	BCH	55443	2003	307	$\approx 2^{-205}$
	GV	39119	1849	307	$\approx 2^{-205}$
	MRRW	27569	1695	307	$\approx 2^{-205}$
	cAHF	13467	1378	305	$\approx 2^{-205}$
[50] Sec. 6.1	BCH	9396	154	9394	$\approx 2^{-205}$
	GV	6008	154	6006	$\approx 2^{-205}$
	MRRW	4160	154	4158	$\approx 2^{-205}$
	cAHF	2297	153	2295	$\approx 2^{-205}$
[50] Sec. 6.2	BCH	156	154	18634	$\approx 2^{-205}$
	GV	156	154	11858	$\approx 2^{-205}$
	MRRW	156	154	8162	$\approx 2^{-205}$
	cAHF	155	153	4437	$\approx 2^{-205}$
[26]	BCH	7376	7374	3687	$\approx 2^{-205}$
	GV	2934	2932	1466	$\approx 2^{-205}$
	MRRW	1440	1438	719	$\approx 2^{-205}$
	cAHF	408	406	203	$\approx 2^{-205}$
Our construction	BCH	3691	3689	3688	$\approx 2^{-205}$
	GV	1470	1468	1467	$\approx 2^{-205}$
	MRRW	723	721	720	$\approx 2^{-205}$
	cAHF	207	205	204	$\approx 2^{-205}$

Table 3. Key and proof sizes for $k = 100, Q = 2^{25}, t = 2^{50}, \epsilon = 2^{-50}$ and $\delta = 0.286$. In consequence, we have $d = 154$. Puncturing a primitive $[4095, 211, 1463]$ BCH-code 408 times to a $[3687, 211, 1055]$ code yields $n^{\mathsf{BCH}} = 3687, n_1^{\mathsf{BCH}} = 61, n_2^{\mathsf{BCH}} = 61$ and $\zeta^{\mathsf{BCH}} = 13$. If an ECC on the GV bound is used, this implies $n^{\mathsf{GV}} = 1466, n_1^{\mathsf{GV}} = 39, n_2^{\mathsf{GV}} = 39$ and $\zeta^{\mathsf{GV}} = 12$. Analogously, if an ECC on the MRRW bound is used, this implies $n^{\mathsf{MRRW}} = 719, n_1^{\mathsf{MRRW}} = 27, n_2^{\mathsf{MRRW}} = 27$ and $\zeta^{\mathsf{MRRW}} = 11$. Finally, if the VRFs are instantiated with a cAHF using TCRHFs, we have $n^{\mathsf{tcrh}} = 203, n_1^{\mathsf{tcrh}} = 15, n_2^{\mathsf{tcrh}} = 15, j = 153$ and $\zeta^{\mathsf{tcrh}} = 9$.

Construction	Instantiation	$\lvert\mathsf{vk}\rvert\ \#\mathbb{G}$	$\lvert\mathsf{sk}\rvert\ \#\mathbb{G}$	$\lvert\pi\rvert\#\mathbb{G}$	$\mathrm{Adv}_{\mathcal{B}}$
[29] Sec. 5.1	BCH	1551	1549	259174	$\approx 2^{-155}$
	GV	1422	1420	121143	$\approx 2^{-155}$
	MRRW	1422	1420	67092	$\approx 2^{-155}$
	cAHF	1155	1153	26122	$\approx 2^{-155}$
[29] Sec. 5.3	BCH	32769	1549	257	$\approx 2^{-155}$
	GV	23094	1420	257	$\approx 2^{-155}$
	MRRW	23094	1420	257	$\approx 2^{-155}$
	cAHF	11267	1153	255	$\approx 2^{-155}$
[50] Sec. 6.1	BCH	5807	129	5805	$\approx 2^{-155}$
	GV	4001	129	3999	$\approx 2^{-155}$
	MRRW	2969	129	2967	$\approx 2^{-155}$
	cAHF	1922	128	1920	$\approx 2^{-155}$
[50] Sec. 6.2	BCH	131	129	11481	$\approx 2^{-155}$
	GV	131	129	7869	$\approx 2^{-155}$
	MRRW	131	129	5805	$\approx 2^{-155}$
	cAHF	130	128	3712	$\approx 2^{-155}$
[26]	BCH	4018	4016	2008	$\approx 2^{-155}$
	GV	1878	1876	938	$\approx 2^{-155}$
	MRRW	1040	1038	519	$\approx 2^{-155}$
	cAHF	408	406	203	$\approx 2^{-155}$
Our construction	BCH	2012	2010	2009	$\approx 2^{-155}$
	GV	942	940	939	$\approx 2^{-155}$
	MRRW	523	521	520	$\approx 2^{-155}$
	cAHF	207	205	204	$\approx 2^{-155}$

Table 4. Key and proof sizes for $k = 100, Q = 2^{25}, t = 2^{50}, \epsilon = 2^{-25}$ and $\delta = 0.235$. In consequence, we have $d = 129$. Puncturing a primitive $[2047, 209, 511]$ BCH-code 39 times to a $[2008, 209, 472]$ code yields $n^{\mathsf{BCH}} = 2008, n_1^{\mathsf{BCH}} = 45, n_2^{\mathsf{BCH}} = 45$ and $\zeta^{\mathsf{BCH}} = 12$. If an ECC on the GV bound is used, this implies $n^{\mathsf{GV}} = 938, n_1^{\mathsf{GV}} = 31, n_2^{\mathsf{GV}} = 31$ and $\zeta^{\mathsf{GV}} = 11$. Analogously, if an ECC on the MRRW bound is used, this implies $n^{\mathsf{MRRW}} = 519, n_1^{\mathsf{MRRW}} = 23, n_2^{\mathsf{MRRW}} = 23$ and $\zeta^{\mathsf{MRRW}} = 11$. Finally, if the VRFs are instantiated with a cAHF using TCRHFs, we have $n^{\mathsf{tcrh}} = 203, n_1^{\mathsf{tcrh}} = 15, n_2^{\mathsf{tcrh}} = 15, j = 128$ and $\zeta^{\mathsf{tcrh}} = 9$.

Construction	Instantiation	$\lvert\mathsf{vk}\rvert\ \#\mathbb{G}$	$\lvert\mathsf{sk}\rvert\ \#\mathbb{G}$	$\lvert\pi\rvert\#\mathbb{G}$	$\mathrm{Adv}_{\mathcal{B}}$
[29] Sec. 5.1	BCH	2005	2003	581672	$\approx 2^{-205}$
	GV	1851	1849	289071	$\approx 2^{-205}$
	MRRW	1697	1695	141846	$\approx 2^{-205}$
	cAHF	1533	1531	39791	$\approx 2^{-205}$
[29] Sec. 5.3	BCH	55443	2003	307	$\approx 2^{-205}$
	GV	39119	1849	307	$\approx 2^{-205}$
	MRRW	27569	1695	307	$\approx 2^{-205}$
	cAHF	19281	1531	305	$\approx 2^{-205}$
[50] Sec. 6.1	BCH	9550	154	9548	$\approx 2^{-205}$
	GV	6778	154	6776	$\approx 2^{-205}$
	MRRW	4776	154	4774	$\approx 2^{-205}$
	cAHF	2603	153	2601	$\approx 2^{-205}$
[50] Sec. 6.2	BCH	156	154	18942	$\approx 2^{-205}$
	GV	156	154	13398	$\approx 2^{-205}$
	MRRW	156	154	9394	$\approx 2^{-205}$
	cAHF	155	153	5049	$\approx 2^{-205}$
[26]	BCH	7554	7552	3776	$\approx 2^{-205}$
	GV	3754	3752	1876	$\approx 2^{-205}$
	MRRW	1842	1840	920	$\approx 2^{-205}$
	cAHF	520	518	259	$\approx 2^{-205}$
Our construction	BCH	3780	3778	3777	$\approx 2^{-205}$
	GV	1880	1878	1877	$\approx 2^{-205}$
	MRRW	924	922	921	$\approx 2^{-205}$
	cAHF	263	261	260	$\approx 2^{-205}$

Table 5. Key and proof sizes for $k = 128, Q = 2^{25}, t = 2^{50}, \epsilon = 2^{-50}$ and $\delta = 0.286$. In consequence, we have $d = 154$. Puncturing a primitive $[4095, 259, 1399]$ BCH-code 319 times to a $[3776, 259, 1080]$ code yields $n^{\mathsf{BCH}} = 3776, n_1^{\mathsf{BCH}} = 62, n_2^{\mathsf{BCH}} = 62$ and $\zeta^{\mathsf{BCH}} = 13$. If an ECC on the GV bound is used, this implies $n^{\mathsf{GV}} = 1876, n_1^{\mathsf{GV}} = 44, n_2^{\mathsf{GV}} = 44$ and $\zeta^{\mathsf{GV}} = 12$. Analogously, if an ECC on the MRRW bound is used, this implies $n^{\mathsf{MRRW}} = 920, n_1^{\mathsf{MRRW}} = 31, n_2^{\mathsf{MRRW}} = 31$ and $\zeta^{\mathsf{MRRW}} = 11$. Finally, if the VRFs are instantiated with a cAHF using TCRHFs, we have $n^{\mathsf{tcrh}} = 259, n_1^{\mathsf{tcrh}} = 17, n_2^{\mathsf{tcrh}} = 17, j = 153$ and $\zeta^{\mathsf{tcrh}} = 10$.

Construction	Instantiation	$\lvert\mathsf{vk}\rvert$ $\#\mathbb{G}$	$\lvert\mathsf{sk}\rvert$ $\#\mathbb{G}$	$\lvert\pi\rvert\#\mathbb{G}$	$\mathrm{Adv}_{\mathcal{B}}$
[29] Sec. 5.1	BCH	1551	1549	261883	$\approx 2^{-155}$
	GV	1551	1549	154942	$\approx 2^{-155}$
	MRRW	1422	1420	85797	$\approx 2^{-155}$
	cAHF	1283	1281	33291	$\approx 2^{-155}$
[29] Sec. 5.3	BCH	32769	1549	257	$\approx 2^{-155}$
	GV	32769	1549	257	$\approx 2^{-155}$
	MRRW	23094	1420	257	$\approx 2^{-155}$
	cAHF	16131	1281	255	$\approx 2^{-155}$
[50] Sec. 6.1	BCH	5936	129	5934	$\approx 2^{-155}$
	GV	4517	129	4515	$\approx 2^{-155}$
	MRRW	3356	129	3354	$\approx 2^{-155}$
	cAHF	2178	128	2176	$\approx 2^{-155}$
[50] Sec. 6.2	BCH	131	129	11739	$\approx 2^{-155}$
	GV	131	129	8901	$\approx 2^{-155}$
	MRRW	131	129	6579	$\approx 2^{-155}$
	cAHF	130	128	4224	$\approx 2^{-155}$
[26]	BCH	4060	4058	2029	$\approx 2^{-155}$
	GV	2402	2400	1200	$\approx 2^{-155}$
	MRRW	1330	1328	664	$\approx 2^{-155}$
	cAHF	520	518	259	$\approx 2^{-155}$
Our construction	BCH	2033	2031	2030	$\approx 2^{-155}$
	GV	1204	1202	1201	$\approx 2^{-155}$
	MRRW	668	666	665	$\approx 2^{-155}$
	cAHF	263	261	260	$\approx 2^{-155}$

Table 6. Key and proof sizes for $k = 128, Q = 2^{25}, t = 2^{50}, \epsilon = 2^{-25}$ and $\delta = 0.235$. In consequence, we have $d = 129$. Puncturing a primitive $[2047, 264, 495]$ BCH-code 18 times to a $[2029, 264, 477]$ code yields $n^{\mathsf{BCH}} = 2029, n_1^{\mathsf{BCH}} = 46, n_2^{\mathsf{BCH}} = 46$ and $\zeta^{\mathsf{BCH}} = 12$. If an ECC on the GV bound is used, this implies $n^{\mathsf{GV}} = 1200, n_1^{\mathsf{GV}} = 35, n_2^{\mathsf{GV}} = 35$ and $\zeta^{\mathsf{GV}} = 12$. Analogously, if an ECC on the MRRW bound is used, this implies $n^{\mathsf{MRRW}} = 664, n_1^{\mathsf{MRRW}} = 26, n_2^{\mathsf{MRRW}} = 26$ and $\zeta^{\mathsf{MRRW}} = 11$. Finally, if the VRFs are instantiated with a cAHF using TCRHFs, we have $n^{\mathsf{tcrh}} = 259, n_1^{\mathsf{tcrh}} = 17, n_2^{\mathsf{tcrh}} = 17, j = 128$ and $\zeta^{\mathsf{tcrh}} = 10$.

Construction	Instantiation	$\lvert\mathsf{vk}\rvert$ $\#\mathbb{G}$	$\lvert\mathsf{sk}\rvert$ $\#\mathbb{G}$	$\lvert\pi\rvert\#\mathbb{G}$	$\mathrm{Adv}_{\mathcal{B}}$
[29] Sec. 5.1	BCH	2159	2157	1177961	$\approx 2^{-205}$
	GV	2005	2003	577822	$\approx 2^{-205}$
	MRRW	1851	1849	283527	$\approx 2^{-205}$
	cAHF	1686	1684	78960	$\approx 2^{-205}$
[29] Sec. 5.3	BCH	78543	2157	307	$\approx 2^{-205}$
	GV	55443	2003	307	$\approx 2^{-205}$
	MRRW	39119	1849	307	$\approx 2^{-205}$
	cAHF	27390	1684	305	$\approx 2^{-205}$
[50] Sec. 6.1	BCH	13554	154	13552	$\approx 2^{-205}$
	GV	9550	154	9548	$\approx 2^{-205}$
	MRRW	6624	154	6622	$\approx 2^{-205}$
	cAHF	3521	153	3519	$\approx 2^{-205}$
[50] Sec. 6.2	BCH	156	154	26950	$\approx 2^{-205}$
	GV	156	154	18942	$\approx 2^{-205}$
	MRRW	156	154	13090	$\approx 2^{-205}$
	cAHF	155	153	6885	$\approx 2^{-205}$
[26]	BCH	15298	15296	7648	$\approx 2^{-205}$
	GV	7504	7502	3751	$\approx 2^{-205}$
	MRRW	3682	3680	1840	$\approx 2^{-205}$
	cAHF	1032	1030	515	$\approx 2^{-205}$
Our construction	BCH	7652	7650	7649	$\approx 2^{-205}$
	GV	3755	3753	3752	$\approx 2^{-205}$
	MRRW	1844	1842	1841	$\approx 2^{-205}$
	cAHF	519	517	516	$\approx 2^{-205}$

Table 7. Key and proof sizes for $k = 256, Q = 2^{25}, t = 2^{50}, \epsilon = 2^{-50}$ and $\delta = 0.286$. In consequence, we have $d = 154$. Puncturing a primitive $[8191, 521, 2731]$ BCH-code 543 times to a $[7648, 521, 2188]$ code yields $n^{\mathsf{BCH}} = 7648, n_1^{\mathsf{BCH}} = 88, n_2^{\mathsf{BCH}} = 88$ and $\zeta^{\mathsf{BCH}} = 14$. If an ECC on the GV bound is used, this implies $n^{\mathsf{GV}} = 3751, n_1^{\mathsf{GV}} = 62, n_2^{\mathsf{GV}} = 62$ and $\zeta^{\mathsf{GV}} = 13$. Analogously, if an ECC on the MRRW bound is used, this implies $n^{\mathsf{MRRW}} = 1840, n_1^{\mathsf{MRRW}} = 43, n_2^{\mathsf{MRRW}} = 43$ and $\zeta^{\mathsf{MRRW}} = 12$. Finally, if the VRFs are instantiated with a cAHF using TCRHFs, we have $n^{\mathsf{tcrh}} = 515, n_1^{\mathsf{tcrh}} = 23, n_2^{\mathsf{tcrh}} = 23, j = 153$ and $\zeta^{\mathsf{tcrh}} = 11$.

Construction	Instantiation	$\lvert\mathsf{vk}\rvert$ $\#\mathbb{G}$	$\lvert\mathsf{sk}\rvert$ $\#\mathbb{G}$	$\lvert\pi\rvert\#\mathbb{G}$	$\mathrm{Adv}_{\mathcal{B}}$
[29] Sec. 5.1	BCH	1809	1807	920946	$\approx 2^{-155}$
	GV	1680	1678	309743	$\approx 2^{-155}$
	MRRW	1551	1549	171454	$\approx 2^{-155}$
	cAHF	1411	1409	66060	$\approx 2^{-155}$
[29] Sec. 5.3	BCH	65793	1807	257	$\approx 2^{-155}$
	GV	46443	1678	257	$\approx 2^{-155}$
	MRRW	32769	1549	257	$\approx 2^{-155}$
	cAHF	22915	1409	255	$\approx 2^{-155}$
[50] Sec. 6.1	BCH	10967	129	10965	$\approx 2^{-155}$
	GV	6323	129	6321	$\approx 2^{-155}$
	MRRW	4775	129	4773	$\approx 2^{-155}$
	cAHF	2946	128	2944	$\approx 2^{-155}$
[50] Sec. 6.2	BCH	131	129	21801	$\approx 2^{-155}$
	GV	131	129	12513	$\approx 2^{-155}$
	MRRW	131	129	9417	$\approx 2^{-155}$
	cAHF	130	128	5760	$\approx 2^{-155}$
[26]	BCH	14278	14276	7138	$\approx 2^{-155}$
	GV	4802	4800	2400	$\approx 2^{-155}$
	MRRW	2658	2656	1328	$\approx 2^{-155}$
	cAHF	1032	1030	515	$\approx 2^{-155}$
Our construction	BCH	7142	7140	7139	$\approx 2^{-155}$
	GV	2404	2402	2401	$\approx 2^{-155}$
	MRRW	1332	1330	1329	$\approx 2^{-155}$
	cAHF	519	517	516	$\approx 2^{-155}$

Table 8. Key and proof sizes for $k = 256, Q = 2^{25}, t = 2^{50}, \epsilon = 2^{-25}$ and $\delta = 0.235$. In consequence, we have $d = 129$. Puncturing a primitive $[8191, 520, 2731]$ BCH-code 1053 times to a $[7138, 520, 1678]$ code yields $n^{\mathsf{BCH}} = 7138, n_1^{\mathsf{BCH}} = 85, n_2^{\mathsf{BCH}} = 85$ and $\zeta^{\mathsf{BCH}} = 14$. If an ECC on the GV bound is used, this implies $n^{\mathsf{GV}} = 2400, n_1^{\mathsf{GV}} = 49, n_2^{\mathsf{GV}} = 49$ and $\zeta^{\mathsf{GV}} = 13$. Analogously, if an ECC on the MRRW bound is used, this implies $n^{\mathsf{MRRW}} = 1328, n_1^{\mathsf{MRRW}} = 37, n_2^{\mathsf{MRRW}} = 37$ and $\zeta^{\mathsf{MRRW}} = 12$. Finally, if the VRFs are instantiated with a cAHF using TCRHFs, we have $n^{\mathsf{tcrh}} = 515, n_1^{\mathsf{tcrh}} = 23, n_2^{\mathsf{tcrh}} = 23, j = 128$ and $\zeta^{\mathsf{tcrh}} = 11$.