Peer-to-Peer Networking and Applications
https://doi.org/10.1007/s12083-021-01243-x

Continuous improvement of script-driven verifiable random functions for reducing computing power in blockchain consensus protocols

Guanglai Guo[1] · Yan Zhu[1] · E Chen[1] · Guizhen Zhu[2] · Di Ma[3] · William ChengChung Chu[4]

Received: 2 May 2021 / Accepted: 1 September 2021

Abstract
In order to solve the problem of low efficiency and high energy consumption of the Proof-of-Work (PoW) consensus protocol in blockchain within a peer-to-peer network, some new protocols based on Verifiable Random Function (VRF) have emerged recently. However, these VRF-based consensus protocols do not actually give a concrete and efficient VRF construction. In view of this, we present three simple and practical VRF constructions from the RSA hardness assumption, the Decisional Diffie-Hellman (DDH) assumption and the Leftover Hash Lemma (LHL) respectively, the output size of which is continuously reduced for the design of efficient consensus protocol in blockchain. We also give a complete security analysis of our VRF constructions. Furthermore, we show a specific application of our VRF constructions in the famous Algorand consensus protocol. We illustrate a general approach to integrate our VRF constructions with block structure in blockchain. Comparing with PoW-based mining, we demonstrate the detailed process of VRF-based consensus protocol. Meanwhile, three new opcodes are designed for the scripting system in blockchain to develop a script pair, scriptProof and scriptHash, which provides secure and efficient block verification. Finally, we evaluate the performance of our VRF constructions in terms of storage and computational overheads, and the experimental evaluation results show our VRF constructions can significantly reduce the computing power of consensus protocol in blockchain.

Keywords Verifiable random function · Script instruction · Consensus protocol · Blockchain · Leftover hash lemma · Performance

1 Introduction

With the rapid development of e-commerce and digital finance, blockchain technology has been widely concerned in recent years [1]. Exactly, blockchain is a new application mode of computer technology such as distributed data storage, peer-to-peer transmission, consensus mechanism and signature algorithm. As the underlying technology of Bitcoin, blockchain can be regarded as a decentralized public ledger. All committed transactions are recorded in this ledger and jointly maintained by all nodes without the control of any third-party organization. Meanwhile, once a transaction in blockchain is agreed by all nodes and is packaged in blockchain, it is recorded by all nodes together and cannot be tampered with. In addition, as a decentralized ledger, blockchain records the input and output of each transaction, and can easily trace each transaction record through its chain structure. Therefore, blockchain has unique advantages of decentralization, tamper proof, traceability, etc, which is commonly considered as the cornerstone of building the trusted large-scale applications such as financial services [2], digital copyright [3], healthcare [4] and Internet of Things [5].

Consensus mechanism, that can make all nodes work together, is generally regarded as the core technology of blockchain. More precisely, nodes participate collaboratively in maintaining a trusted public ledger in chronological order and most of them keep a complete ledger backup that must be exactly consistent with each other. Therefore, consensus mechanism enables to solve the problem of data

✉ Yan Zhu
zhuyan@ustb.edu.cn

[1] School of Computer and Communication Engineering, University of Science and Technology, Beijing 10083, China

[2] Data Communication Science and Technology Research Institute, Beijing 100191, China

[3] University of Michigan-Dearborn, Michigan 48128, USA

[4] Tunghai University, Taichung 40704, Taiwan

Published online: 06 October 2021

consistency between untrusted nodes in blockchain. Our work is of great significance for improving the security and efficiency of peer-to-peer networks.

1.1 Motivation

Bitcoin, as the origin of blockchain, is the earliest and most famous platform of blockchain. Satoshi Nakamoto first introduced a Proof-of-Work (PoW) consensus protocol in Bitcoin [6], and it has been extensively employed in most applications of blockchain. Generally speaking, the PoW-based protocol should participate in solving a complex computational puzzle together, in which all nodes are required to compete for the privileges to propose new blocks by their computing power [7]. Any node with higher computing power might have greater chance to be a block proposer.

The SHA-256 function $y = Hash(x)$ serves as the computational puzzle in the PoW-based protocol, and it is essentially a Random Function (RF) with collision resistance. RF enables to generate a random output y on given input x. In the PoW, a target threshold T should be set to the binary form, i.e., $T = 0^n||R$, where 0^n denotes that the first n bits of T are all 0, and R indicates that the remaining bits are random values. On this basis, nodes should try different x repeatedly and then calculate the RF's output y until $y \leq T$ holds. The first node to reach the target threshold T has a greater advantage to obtain the privilege to propose a new block. So, the cost of malicious proposers destroying the system will be greatly increased since they must do a large number of computations (or control more than 51% of network's total computing power), so as to guarantee the security of the consensus process.

However, the PoW-based protocol has some limitations [8] in practical application. Taking Bitcoin as an example, we demonstrate main limitations as follows:

- *High energy consumption*. For proposing new blocks, nodes must do a large number of computations to find the solution of the computational puzzle, which results in huge energy consumption.
- *Transaction delay*. There exists a transaction delay in Bitcoin. Nodes should wait for the generation of the next six blocks in order to confirm the credibility of each block, the process of which will takes around one hour. So, the transaction throughput is not high.
- *Centralization of computational power*. Mining pools generally have advantages of winning the privileges to propose new blocks than individuals. Therefore, mining pools can dominate the processes of building new blocks in Bitcoin, which violates the decentralization of blockchain.

In 2016, Turing Award winner Gilad et al. [9] proposed a new VRF-based consensus protocol called Algorand, which aims to tackle the problem of low efficiency and high energy consumption of the PoW-based protocol. Besides, other new protocols based on Verifiable Random Function (VRF) [10] have been also introduced recently. VRF is essentially a (pseudo) RF with a non-interactive verifiable function. Informally, VRF is such an effective function that it takes a string x and a secret key sk as inputs and generates a random output $y = F(sk, x)$ with a corresponding proof $\pi = G(sk, x)$.

A standard VRF is required to satisfy the following properties. Firstly, given sk, the output y of $F(sk, x)$ should be unique and computable on x. Secondly, the output y needs to be random, i.e., an effective attacker, without knowing sk, can not distinguish the output y from a random value even if it is given oracle queries of $F(sk, x)$ and $G(sk, x)$ on any other point. Finally, anyone who holds the proof π along with pk can verify that the output y is indeed calculated correctly on x.

According to the work of Gilad et al. [9], it can be seen that VRF has some advantages to design efficient consensus protocols since the unique properties of VRF. We now illustrate the main advantages as follows:

- *Easy to calculate*. Nodes need only to perform VRF function $y = F(sk, x)$ once, and then compete for the privileges to propose new blocks by the distance between the VRF's output y and the preset target threshold T. This means that nodes with shorter distance should have more chances to win the privileges. So, VRF has ability to avoid the repeated calculation processes in the PoW-based protocol.
- *Easy to verify*. There exists a proof function $\pi = G(sk, x)$ in VRF. The proof π is used to verify the correctness of the output y on given x. Nodes who hold π and pk can easily verify that the new block is indeed built by the proposer who holds the corresponding sk. So, VRF is easy to confirm the privilege of the proposer and the validity of the block simultaneously.
- *Suitable for consensus protocols*. VRF has already been integrated into some new protocols such as Algorand [9], Ouroboros-Praos [11] and Dfinity [12]. These VRF-based protocols aim to improve consensus performance from the aspect of energy consumption, scalability of blockchain, transaction throughput, etc. Therefore, VRF is very suitable for designing efficient consensus protocols.

However, the concrete and efficient VRF constructions are not given in the existing VRF-based protocols, such as Algorand [9], Ouroboros-Praos [11] and Dfinity [12]. Meanwhile, almost known VRF functions are based on Bilinear maps, which are constructed on elliptic curves. Comparing

with the elliptic curve cryptosystem used in existing blockchains, the protocol design based on VRF with Bilinear maps is more complex, and the mathematics backgrand is more strict for professional developers. Therefore, in this paper, we aim to present some simple and practical constructions that are suitable for design of efficient consensus protocols.

1.2 Related work

In 1999, the concept of VRF was first introduced by Micali, Rabin and Vadhan [10]. They proposed a VRF construction based on the RSA assumption from a verifiable unpredictable function, which employed a Goldreich-Levin hardcore bit transformation [13]. Dodis et al. [14] gave a more efficient VRF construction on bilinear groups with small input spaces. This construction was simple without inefficent Goldreich-Levin transformation, and it was provably secure under a decisional bilinear Diffie-Hellman inversion assumption. Hohenberger et al. [15] presented a family of VRFs on bilinear groups, whose security relied on a decisional Diffie-Hellman exponent assumption. The input space of this kind of VRF reached exponentially large size by applying a collision-resistant hash function. Hofheinz et al. [16] constructed the first VRFs which efficiently achieved both exponential-sized input space and full adaptive security under a non-interactive, constant-size assumption. Based on the work in [16], Kohl [17] proposed a VRF construction with short proofs, which also satisfied exponential-sized input space and full adaptive security. They employed partitioning techniques of Bitansky [18] to effectively reduce the proof size to a logarithmic number of group elements.

Some researches focused on extending the notion of VRF to construct some variants of VRF. Brackerski et al. [19] defined weak VRFs whose pseudorandomness was required to hold only for randomly selected inputs. In addition, they gave two concrete constructions of weak VRFs from certified trapdoor permutations and a computational Diffie-Hellman assumption, respectively. Fuchsbauer [20] extended the notion of VRF to constrained VRFs and showed two instantiations, i.e., bit-fixing VRF and circuit-constrained VRF. Recently, Wang et al. [21] presented a family of Conditionally VRFs and showed a direct construction with Boolean access constraints. Liang et al. [22] defined static aggregate VRFs and presented a construction over bit-fixing sets with respect to product aggregation.

Other VRF constructions based on basic cryptography primitives were presented in recent years. Goyal et al. [23] demonstrated a generic way of building VRFs from more basic cryptographic primitives, such as non-interactive witness-indistinguishable proofs (NIWIs), admissible hash functions (AHFs), perfectly binding commitments and constrained pseudorandom functions. They also provided new constructions of non-interactive commitments from Learning with Errors (LWE) and learning parity with noise assumptions. Bitansky [18] presented new VRF constructions from verifiable function commitments and constrained pseudorandom functions, which relied on NIWIs proof system to achieve adaptive security. Brunetta et al. [24] provided their insights on constructing a lattice-based simulatable VRF using non interactive zero knowledge arguments and dual-mode commitment schemes, and they pointed out the main challenges that need to be addressed on it. Abraham [25] proposed a post-quantum VRF scheme from ring signatures using ring LWE and proved its security against known quantum attacks and quantum oracles. Jager et al. [26] constructed a new VRF scheme using computational AHFs, and it is currently the most efficient VRF with full adaptive security in standard model. The variants of AHFs-based VRF also include Jager's scheme under balanced AHFs [27] and Yamada's scheme under modified AHFs [28].

1.3 VRF-based consensus protocols

Recently, some new protocols based on VRF have emerged and been widely employed in practice. Several typical VRF-based protocols are elaborated as follows:

- *Algorand protocol* [9], is a novel cryptocurrency designed to confirm transactions in about one minute. It employs a new Byzantine agreement protocol to reach a consensus, which takes a committee as its basic operation unit. Meanwhile, the core of the protocol is VRF technique, which is responsible for randomly selecting proposer and verifier committees in a private and non-interactive way.
- *Ouroboros-Praos protocol* [11], is also a VRF-based consensus protocol for the adoption of a new cryptocurrency called Cardano. Time is divided into several epochs, each of which is composed of multiple slots. According to the stake distributions of stakeholders in each epoch, this protocol uses VRF to randomly select a slot leader from the stakeholders, so that the slot leader is qualified to publish a block.
- *Dfinity protocol* [12]. Similar to Algorand, this protocol is also operated under a committee. And it contains a decentralized random beacon, which actually employs VRF and BLS threshold signature scheme [29] to generate an unpredictable random seed. Furthermore, the seed is not only responsible for creating block-makers and notary committees, but also for determining the priority ranking of committee members.

From these work, it can be seen that VRF has made some progresses in the design of blockchain consensus protocols. Precisely speaking, VRF plays an important role in selecting

a block proposer based on the randomness and public verifiability of VRF. In addition, VRF has also been applied into a new blockchain system called DEXON [30] with Proof-of-Participation (PoP) protocol and a novel Verifiable Byzantine Fault Tolerance (VBFT) [31] hybrid algorithm in Ontology platform, respectively.

1.4 Our contribution

We aim to construct practical and efficient VRF constructions which can be used to improve the performance of consensus protocol in blockchain. We summarize the main contributions of our work in this paper as follows:

- We propose three simple and practical VRF constructions in order to reduce the VRF's output size as much as possible. Our VRF construction over RSA (VRF-RSA) is inspired by one-way permutation built on RSA. Another two VRF constructions, VRF-DDH and VRF-LHL, are based on the Decisional Diffie-Hellman (DDH) assumption and Leftover Hash Lemma (LHL) respectively, where LHL enables an universal hash function to achieve an effective randomness extractor. As a result, the VRF's outputs y and π are continuously improved, e.g., the output size of VRF-LHL is only 64 bytes for 128-bit security strength. The computational overheads of **Prove** and **Verify** in VRF-DDH can be reduced to the millisecond level. Moreover, we give the full security proofs of our constructions.
- Taking VRF-RSA as an example, we provide a general approach to integrate our VRF constructions with block structure in blockchain. By replacing PoW-based mining with VRF-based consensus protocol, the VRF-RSA is applied into cryptographic sortation algorithm in the famous Algorand [9] to improve the performance of consensus protocol. Besides, three new opcodes are designed by extending the script interpreter of blockchain, and then we develop a script pair, scriptProof and scriptHash, under our VRF constructions to provide secure and efficient block verification. Moreover, our experimental evaluation results show the proposed VRF constructions can significantly reduce the computing power of consensus protocol in blockchain.

1.5 Organization

The rest of this paper is organized as follows. We introduce some basic concepts and preliminaries for VRF in Sect. 2. Next, we present three simple and practical VRF constructions and provide the full security analysis of our constructions in Sect. 3. In Sect. 4, we compare the performance of our VRF constructions with previous ones. In Sect. 5, we show a specific application of VRF in blockchain and VRF's script implementation and performance evaluation. Finally, we conclude this paper in Sect. 6.

2 Preliminaries

2.1 Verifiable random function

VRF was first introduced by Micali, Rabin and Vadhan. Informally, a VRF acts like a (pseudo) RF, but also contains a proof of correctness for its output. We now briefly recall the following standard definition of VRF.

Definition 1 (**Verifiable Random Function**) Given a function family $F(\cdot,\cdot) : \{0,1\}^{in(\kappa)} \to \{0,1\}^{out(\kappa)}$, where $in(\kappa)$ and $out(\kappa)$ are polynomials in security parameter κ. we say that it is a Verifiable Random Function if there exists algorithms (*Setup*, *Prove*, *Verify*) such that

1. **Setup**(1^κ) takes as input the security parameter κ, and outputs a pair of keys (sk, pk), where sk is the secret key and pk is the public key;
2. **Prove**(sk, x) takes sk and a string $x \in \{0,1\}^{in(\kappa)}$ as inputs, outputs a pair $(y, \pi) \leftarrow (F(sk,x), G(sk,x))$, where y is a function value and π is a proof of correctness;
3. **Verify**(pk, x, y, π) takes $pk, x \in \{0,1\}^{in(\kappa)}, y \in \{0,1\}^{out(\kappa)}$ and π as inputs, and outputs a bit. It verifies the correctness of y on given x by using the proof π and pk. It outputs 1 if the verification succeeds, and 0 otherwise.

Formally, we require that VRF holds the following security properties:

1. **Provability**. For all pairs $(pk, sk) \in Setup(1^\kappa)$ and all strings $x \in \{0,1\}^{in(\kappa)}$, if $(y, \pi) = Prove(sk, x)$, there exists a negligible polynomial μ such that

$$\Pr\left[\, Verify(pk, x, y, \pi) = 1 \,\right] 1 - \mu(\kappa). \tag{1}$$

2. **Uniqueness**. For all pairs $(pk, sk) \in Setup(1^\kappa)$ and all strings $x \in \{0,1\}^{in(\kappa)}$, there does not exist a tuple (y_1, y_2, π_1, π_2) such that

$$\Pr\left[y_1 \neq y_2 \,\middle|\, \begin{array}{l} Verify(pk, x, y_1, \pi_1) = 1, \\ Verify(pk, x, y_2, \pi_2) = 1 \end{array} \right] \leq \mu(\kappa) \tag{2}$$

for a negligible polynomial μ.

3. **Pseudorandomness**[1].For all PPT distinguishers D, there exists a negligible polynomial μ such that

[1] Randomness: We say that $F(sk, x)$ and $\{0,1\}^{out(\kappa)}$ are statistically indistinguishable if there exists a negligible statistical difference μ such that

$$\frac{1}{2}\sum_{\alpha} \left|\Pr[F(sk,x) = \alpha] - \Pr[\{0,1\}^{out(\kappa)} = \alpha]\right| \leq \mu(\kappa).$$

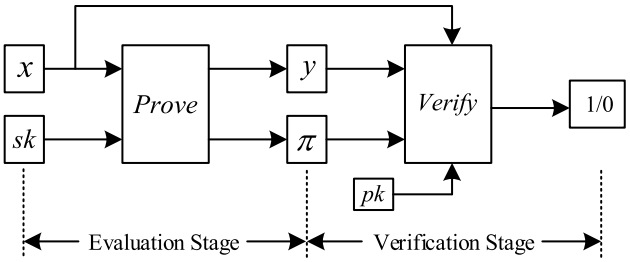

Fig. 1 The processing structure of VRF

$$\left|\Pr[D(1^{\kappa}, F(sk,x)) = 1] - \Pr[D(1^{\kappa}, \{0,1\}^{out(\kappa)}) = 1]\right| \leq \mu(\kappa). \tag{3}$$

More clearly, we illustrate the processing structure of VRF in Fig. 1, which consists of Prove and Verify modules. As shown in the figure, the processing can be divided into evaluation and verification stages. In the evaluation stage, the Prove module is used to evaluate the value y and the proof π. Subsequently, the Verify module is responsible for verifying the correctness of the value y with the proof π and pk in verification stage. It outputs 1 if the verification is successful, and 0 otherwise.

2.2 Hardness problems and complexity assumptions

We now give the hardness problems and complexity assumptions used in our VRF constructions. Hereafter, $\mathcal{A}$ is denoted as a probabilistic polynomial time (PPT) algorithm whose running time is bounded by the polynomial in the security parameter κ.

We start with the definitions of RSA problem and its hardness assumption in detail. Formally, the RSA problem in $\mathbb{Z}_N$ is defined as follows:

Definition 2 (**RSA Problem**) Given $\{e, N\}$ and a random $y \in \mathbb{Z}_N$, an algorithm $\mathcal{A}$ computes x such that $x^e \equiv y \bmod N$, where positive integer N is a product of two large distinct odd primes, p and q, and e is a randomly chosen positive integer less than and relatively prime to $\phi(N) = (p-1)(q-1)$.

We define by $Adv^{\mathrm{RSA}}_{e,N,y}(\mathcal{A})$ the advantage of an algorithm $\mathcal{A}$ in solving the RSA problem as

$$Adv^{\mathrm{RSA}}_{e,N,y}(\mathcal{A}) = \Pr[\mathcal{A}(y,N,e) = x | x^e \equiv y \bmod N]. \tag{4}$$

The RSA assumption in $\mathbb{Z}_N$ is the following.

Definition 3 ((t,ϵ)-**RSA Assumption**) We say that (t,ϵ)-RSA assumption holds in $\mathbb{Z}_N$ if there is no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in solving the RSA problem in $\mathbb{Z}_N$, i.e.,

$$Adv^{\mathrm{RSA}}_{e,N,y}(\mathcal{A}) \leq \epsilon. \tag{5}$$

Next, let $\mathbb{G}$ be a group of prime order p and g be its generator. We state the DDH problem and its hardness assumption in the group $\mathbb{G}$ in a formal way as follows:

Definition 4 (**Decisional Diffie-Hellman Problem**) Given a tuple $(g, g^{\alpha}, g^{\beta})$ as input, an algorithm $\mathcal{A}$ distinguishes $g^{\alpha\beta}$ from a random value T in $\mathbb{G}$, where α and β are randomly chosen in $\mathbb{Z}_q$ of large prime order q, and $q|p$.

We define by $Adv^{\mathrm{DDH}}_{g,\alpha,\beta}(\mathcal{A})$ the advantage of an algorithm $\mathcal{A}$ in solving the DDH problem as

$$Adv^{\mathrm{DDH}}_{g,\alpha,\beta}(\mathcal{A}) = \left|\Pr[\mathcal{A}(g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1] - \Pr[\mathcal{A}(g, g^{\alpha}, g^{\beta}, T) = 1]\right|. \tag{6}$$

Correspondingly, the DDH assumption in the group $\mathbb{G}$ of prime order p is the following.

Definition 5 ((t,ϵ)-**DDH Assumption**) We say that (t,ϵ)-DDH assumption holds in $\mathbb{G}$ if there is no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in solving the DDH problem in $\mathbb{G}$, i.e.,

$$Adv^{\mathrm{DDH}}_{g,\alpha,\beta}(\mathcal{A}) \leq \epsilon. \tag{7}$$

2.3 Universal hashing and leftover hash lemma

One of our VRF constructions relies on Leftover Hash Lemma (LHL) [32]. LHL demonstrates that an universal hash function is almost a good randomness extractor. Next, we briefly recall the definitions of the universal hashing and LHL, respectively.

Definition 6 (ρ-**Universal Hashing**) A family $\mathcal{H}$ of (deterministic) functions $h : \mathcal{X} \to \{0,1\}^v$ is a called ρ-universal hash family (on space $\mathcal{X}$), if for any $x_1 \neq x_2 \in \mathcal{X}$, we have $\Pr_{h\in\mathcal{H}}[h(x_1) = h(x_2)] \leq \rho$. We say that $\mathcal{H}$ is universal when $\rho = 1/2^v$.

Lemma 1 (**Leftover-Hash Lemma, LHL**) *Assume that the family $\mathcal{H}$ of functions $h : \mathcal{X} \to \{0,1\}^v$ is a $\frac{1+\gamma}{2^v}$-universal hash family. Then the extractor $Ext(x;h) = h(x)$ is an (m,ϵ)-extractor, where h is uniform over $\mathcal{H}$ and $\epsilon = \frac{1}{2}\sqrt{\gamma + \frac{2^v}{2^m}} = \frac{1}{2}\sqrt{\gamma + \frac{1}{2^L}}$ (recall, $L = m - v$ is the entropy loss). In particular, $\frac{1+3\epsilon^2}{2^v}$-universal hash functions yield $(v + 2\log(1/\epsilon), \epsilon)$-extractors.*

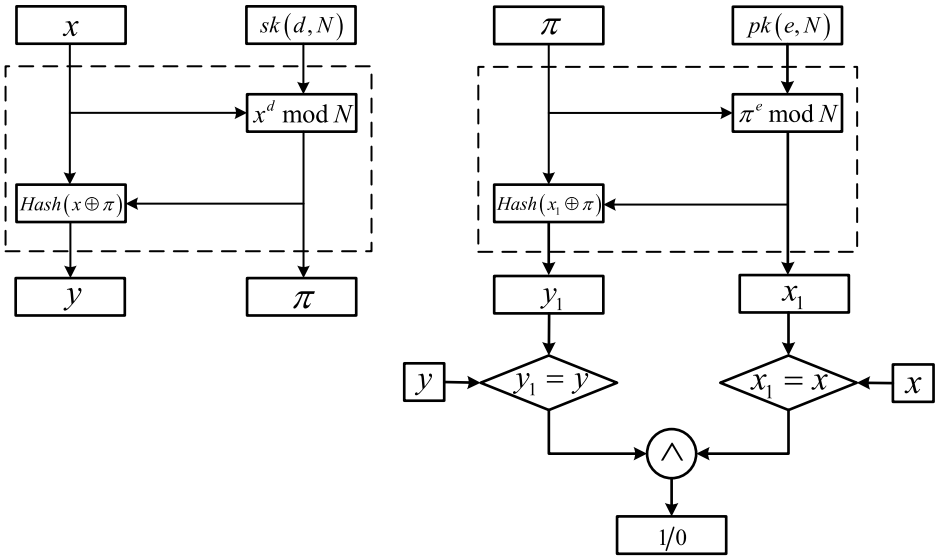

Fig. 2 The processing structures of Prove and Verify modules for VRF-RSA

(a) The processing of the Prove module **(b)** The processing of the Verify module

3 Constructions of verifiable random function

In this section, we give three concrete VRF constructions, whose security are based on the RSA assumption, the DDH assumption and LHL lemma, respectively. In order to improve the performance in VRF-based consensus protocol, the storage overheads of the proof π have fallen steadily among three VRF constructions. Moreover, we provide complete security analysis of our three VRF constructions.

3.1 VRF construction over RSA

We here illustrate the VRF construction over RSA (VRF-RSA). Let *Hash* be a hash function which maps an element in $\mathbb{Z}_N$ to an element in $\mathbb{Z}_N$, and the operator $\oplus$ is denoted as a XOR operation. Our VRF-RSA that contains three polynomial-time algorithms is as follows:

1. **Setup**(1^κ). It takes the security parameter κ as input, and then outputs the secret key $sk = \{d, N\}$ and the public key $pk = \{e, N\}$.
2. **Prove**(sk, x). It takes an integer x and the secret key sk as inputs, and then generates a value y with a corresponding proof π, which are defined as

$$\begin{cases} y &= F(sk, x) = Hash(\pi \oplus x), \\ \pi &= G(sk, x) \equiv x^d \bmod N. \end{cases} \tag{8}$$

3. **Verify**(pk, x, y, π). It verifies whether the value y is correctly computed on given x by using the proof π and the public key pk. Specifically, this algorithm will check

$$\begin{cases} \pi^e \bmod N &\equiv x, \\ Hash(\pi \oplus x) &= y. \end{cases} \tag{9}$$

If two above equations hold, the algorithm outputs 1, and 0 otherwise.

For clarity, the processing structures of Prove and Verify modules for VRF-RSA are illustrated in Fig. 2, respectively. In Fig. 2a, we display the processing of the Prove module in evaluation stage. Obviously, the forward processing is entirely consistent with its inverse. Similarly, the processing of the Verify module in verification stage is shown in Fig. 2b. It is easy to see that the top half of the processing structure for the Verify module is the same with the Prove module in Fig. 2a. So the two modules should have common algorithm. Subsequently, the Verify module will check whether $x_1 = x$ and $y_1 = y$ hold simultaneously in oder to verify the correctness of y on given x.

Formally, we provide the Algorithm 1 to illustrate the common processing of Prove and Verify modules, where *Mod*() is denoted as a modular function and *BitXor*() is denoted as a XOR function. The inputs of this algorithm are α and β, and its outputs are λ and γ. However, both inputs and outputs are actually different in these two modules: the inputs of the Prove module are x and d, and its outputs are y and π. But for the Verify module, the inputs are π and e, and its outputs are y and x. Therefore, four variants, α, β, λ and γ, represent different meanings, respectively. Specifically, two variants, α and β, represent x and d in the Prove module, as well as two outputs, λ and γ, indicate y and π, respectively. Similarly, in the Verify module, four variants, α, β, λ and γ, represent π, e, y and x, respectively.

Algorithm 1 Prove-verify(α, β, N)

Input: an integer α and a pair (β, N).
Output: two values λ and γ.
1: $\gamma = Mod(\alpha^{\beta}, N)$;
2: $temp = BitXor(\gamma, \alpha)$;
3: $\lambda = Hash(temp)$;
4: **Return** λ, γ.

We now prove that our VRF-RSA satisfies the required security properties, as follows:

1. **Provability**. For all pairs of keys $(pk = \{e, N\}, sk = \{d, N\}) \in Setup(1^{\kappa})$ and any integer x, from the definitions of $F(sk, x)$ and $G(sk, x)$, it follows immediately that $\pi^e \bmod N \equiv (x^d \bmod N)^e \bmod N \equiv x$ and $Hash(\pi \oplus x) = y$. In other words, the value y enables to be deterministically obtained from the proof π. We have thus $Verify(pk, x, y, \pi) = 1$.
2. **Uniqueness**. Consider the public key $pk = \{e, N\}$, an integer x, and pairs of values $(y_\rho, \pi_\rho) \in (\mathbb{Z}_N, \mathbb{Z}_N)$ that satisfy $Verify(pk, x, y_\rho, \pi_\rho) = 1$, for both $\rho \in \{1, 2\}$. It suffices to show that $y_1 = y_2$. Specifically, the first verification equation yields $\pi_\rho^e \bmod N \equiv x$, which by the properties of RSA signature scheme implies that $\pi_\rho \equiv x^d \bmod N$, as an one-way permutation, for both $\rho \in \{1, 2\}$. Subsequently, the second verification equation yields $y_\rho = Hash(\pi_\rho \oplus x) = Hash(x^d \bmod N \oplus x)$ for both $\rho \in \{1, 2\}$, which implies $y_1 = Hash(x^d \bmod N \oplus x) = y_2$. It means that there is only one unique y on given x that can be proved to be valid with the proof π.
3. **Pseudorandomness**. Regarding the proof of pseudorandomness property, we present the following Theorem 1.

Theorem 1 *Suppose the (t, ϵ)-RSA assumption holds in $\mathbb{Z}_N$. Our VRF-RSA is a (t', ϵ)-secure VRF, where $t' \approx t$, i.e., there not exists a PPT adversary can distinguish VRF's output from a random value in t'-time with advantage ϵ, thereby breaking the RSA assumption.*

Proof. For the sake of contradiction, suppose there exists an adversary $\mathcal{A}$, running in time t', can distinguish the function $F(sk, x)$ from a random value in $\mathbb{Z}_N$ with non-negligible probability ϵ, then we will build a simulator $\mathcal{B}$ to break the RSA assumption with non-negligible probability by using the advantage of $\mathcal{A}$.

Setup. $\mathcal{B}$ generates the public key $pk = \{e, N\}$ and the secret key $sk = \{d, N\}$, then sends the public key pk to $\mathcal{A}$. Regarding a challenge z^* which $\mathcal{A}$ wants to invert, $\mathcal{B}$ processes it as follows:

- If the bit length of z^* is strictly less than that of N, then define $z' = z^*$;
- Otherwise, try repeatedly a random $\alpha \in \mathbb{Z}_N$ until the bit length of $z^*\alpha^e \bmod N$ is less than that of N, then define $z' \equiv z^*\alpha^e \bmod N$.

Oracle Queries. Without loss of generality, we assume that $\mathcal{A}$ makes at most Q queries and never repeats queries. Once $\mathcal{A}$ issues a query on $x_i (1 \le i < Q)$ to $\mathcal{B}$, then $\mathcal{B}$ responses to the query x_i as follows:

- If the query $x_i = x^*$ (x^* is a challenge query), then $\mathcal{B}$ outputs fail;
- Otherwise, $\mathcal{B}$ processes as follows:

  1. define $x_i = z_i$, then response with z_i;
  2. define $\pi_i = c_i$, then response with c_i;
  3. draw a uniformly random value u_i, program $Hash(c_i \oplus z_i) = u_i$ and $y_i = u_i$, then response with u_i.

Thus, $\mathcal{B}$ will eventually send $\{z_i, c_i, u_i\}$ as response values of the query x_i to $\mathcal{A}$ in this stage.

Challenge Query. $\mathcal{A}$ issues a challenge query on x^* which corresponds to the inverse challenge z^*. If the challenge query is different from x^*, then $\mathcal{B}$ outputs fail. Otherwise, $\mathcal{B}$ certainly outputs $z' \equiv (\pi^*)^e \bmod N$, where π^* is the RSA inverse of z'. Besides, $\mathcal{B}$ defines $z' \equiv z^*\alpha^e \bmod N$. So the RSA inverse of z^* enables to be obtained by dividing the answer with α, i.e., the RSA inverse of z^* is π^*/α.

Guess. Eventually $\mathcal{A}$ outputs its guess b', and then $\mathcal{B}$ outputs the same bit b' as its guess.

Success probability. We now analyze the probability that $\mathcal{B}$ wins the game. Since queries to *Hash* are responded consistently if *Hash* is programmed on the query and are answered randomly otherwise, so the probability that the *Hash* query table contains π^*, which is the RSA inverse of z', is ϵ. We have thus completed the proof of Theorem 1.

In literature [33], Goldberg et al. also proposed an RSA-based VRF scheme (called GVPR-VRF). In the aspect of construction, the output expression of GVPR-VRF is defined as $y = Hash(\pi)$, but our VRF-RSA scheme improves it into $y = Hash(\pi \oplus x)$. This is the main difference between these two schemes. Aiming to the difference, we analyze these two schemes from the aspects of strict expression and forgery resistance.

Firstly, considering x is the input of $F(sk, x)$, the output y of $F(sk, x)$ should be correlated with x according to the VRF definition. From the expression $y = Hash(\pi)$ in GVPR-VRF, the output y does not establish a direct correlation with x, but employs a proof π to establish the indirect correlation with x. Instead, the expression $y = Hash(\pi \oplus x)$ of VRF-RSA binds

π and x by using XOR operation, so as to establish the direct correlation between x and y. Considering such a direct correlation implies the uniqueness of VRF, it indicates that our VRF-RSA complies strictly with the VRF definition.

Secondly, VRF forgery is a serious attack that an adversary finds pairs (x, y, π) to pass VRF verification after learning some valid instances. According to the expressions $y = Hash(\pi)$ and $y = Hash(\pi \oplus x)$, it is easy to see that the forgery attack may originate from hash collision for these two schemes. Since the hash function is a mapping from large input space to small output space, the hash collision is inevitable, i.e., there must exist two distinct inputs to produce identical hash value. For $y = Hash(\pi)$ in GVPR-VRF, we assume that an adversary finds two distinct proofs, π and π', which produce a hash collision (e.g., SHA-256 has been attacked by [34]). According to $x = \pi^e \bmod N$ and $x' = \pi'^e \bmod N$, the adversary can easily find two distinct VRF inputs x and x' corresponding to the identical output y. Thus, any adversary, without knowing the private key sk, can forge two valid instances, (x, y, π) and (x', y, π'), that pass VRF verification as long as the adversary finds a hash collision.

On the other hand, the output expression of VRF-RSA is defined as $y = Hash(\pi \oplus x) = Hash(x^d \oplus x)$. Assuming two distinct inputs, z and z', are found to produce a hash collision, an adversary tries to find x, x' to meet $z = x^d \oplus x$ and $z' = x'^d \oplus x'$. Thus, we easily get $x = (z \oplus x)^e \bmod N$ and $x' = (z' \oplus x')^e \bmod N$. Further, the adversary needs to solve these two equations if he wants to find two distinct VRF inputs x and x' corresponding to the identical output y. Let $r = z \oplus x$ and $r' = z' \oplus x'$. The above equations are transformed into $z \oplus r = r^e \bmod N$ and $z' \oplus r' = r'^e \bmod N$. However, it is difficult to solve r and r' to get x and x', even if sieve method needs $\mathcal{O}(\sqrt{N})$ computational complexity to solve it[2]. In other words, for an effective VRF forgery attack, the adversary needs to not only find a hash collision, but also solve the above problem about $\mathcal{O}(2^{1536})$ operations for N is 3072 bits. So, VRF-RSA has stronger forgery resistance than GVPR-VRF.

3.2 VRF construction over DDH

The size of the proof π in VRF-RSA is too large since its size is affected by the security strength of RSA. So, we propose the VRF construction over DDH (VRF-DDH) with shorter proof. Let H_1 be a hash function that maps an element in $\mathbb{G}$ of prime order p and an element in $\mathbb{Z}_q$ of prime order q to an element in $\mathbb{Z}_q$, and let H_2 be a hash function which maps an element in $\mathbb{Z}_q$ to an element in $\mathbb{G}$. Our VRF-DDH is as follows:

1. **Setup**(1^κ). It takes the security parameter κ as input, and outputs the secret key $sk = \{g, p, q, h = g^\alpha, \alpha, \beta\}$ and the public key $pk = \{g, p, q, h = g^\alpha, \gamma = g^\beta\}$, where α and β are randomly chosen in $\mathbb{Z}_q$.
2. **Prove**(sk, x). It takes an integer x and the secret key sk as intputs, and then generates a value y with a corresponding proof π. Here we firstly define the proof π as follows:
   - Compute a hash function $c = H_1(h^\beta, x)$;
   - Compute $z \equiv \beta + c/\alpha \bmod q$;
   - Define the proof π as a binary tuple

   $$\pi = G(sk, x) = (z, c), \tag{10}$$

   where the size of π is $2\log(q)$ bits.
   Next, we define the value y for the integer x as

   $$y = F(sk, x) = H_2(z \oplus x). \tag{11}$$

3. **Verify**(pk, x, y, π). It verifies whether the value y is correctly computed on given x with the proof π and the public key pk. Specifically, two verification equations are given as follows:

   $$\begin{cases} H_1(h^z g^{-c}, x) = c, \\ H_2(z \oplus x) = y. \end{cases} \tag{12}$$

   If two above verifications succeed, this algorithm outputs 1, and 0 otherwise.

Subsequently, the Algorithms 2 and 3 are presented to clearly illustrate the processing of the Prove and Verify modules, respectively. Obviously, the two algorithm processes are relatively simple and need only perform two hash operations. Thus, our VRF-DDH should have low computational overhead.

Algorithm 2 Prove(sk, x)

Input: an integer x and the secret key $sk = \{g, p, q, h = g^\alpha, \alpha, \beta\}$.
Output: a value y and a proof $\pi = (z, c)$.
1: $c = H_1(h^\beta, x)$;
2: $temp = \beta + c/\alpha$;
3: $z = Mod(temp, q)$;
4: $temp = BitXor(z, x)$;
5: $y = H_2(temp)$;
6: **Return** y, z, c.

Algorithm 3 Verify(pk, x, y, π)

Input: a pair (x, y), a proof $\pi = (z, c)$, and the public key $pk = \{g, p, q, h = g^\alpha, \gamma = g^\beta\}$.
Output: 1 or 0.
1: $temp = h^z * g^{-c}$;
2: $c_1 = H_1(temp, x)$;
3: $temp = BitXor(z, x)$;
4: $y_1 = H_2(temp)$;
5: **if** $c_1 - c == 0$ and $y_1 - y == 0$ **then**
6: $temp = 1$;
7: **else**
8: $temp = 0$;
9: **end if**
10: **Return** $temp$.

[2] For example, Shanks algorithm, one of the famous sieve methods, can realize the computational complexity of $\mathcal{O}(\sqrt{N})$ to find out r and r'.

We now prove that our VRF-DDH satisfies the required security properties, as follows:

1. **Provability**. For all pairs of keys $(pk = \{g, p, q, h = g^{\alpha}, \gamma = g^{\beta}\}, sk = \{g, p, q, h = g^{\alpha}, \alpha, \beta\}) \in Setup(1^{\kappa})$ and any integer x, from the definitions of $F(sk, x)$ and $G(sk, x)$, we can easily get

$$h^{z} g^{-c} = h^{\beta + c/\alpha} g^{-c} = h^{\beta} (g^{\alpha})^{c/\alpha} g^{-c} = h^{\beta}. \tag{13}$$

So it follows immediately that

$$H_1(h^{z} g^{-c}, x) = H_1(h^{\beta}, x) = c \tag{14}$$

and $H_2(z \oplus x) = y$. That is, the value y enables to be deterministically obtained from the proof $\pi = (z, c)$, we have thus $Verify(pk, x, y, \pi) = 1$.
2. **Uniqueness**. Assume that there exists the public key $pk = \{g, p, q, h = g^{\alpha}, \gamma = g^{\beta}\}$, an integer x, and pairs of values $(y_{\rho}, \pi_{\rho} = (z_{\rho}, c_{\rho})) \in (\mathbb{G}, \mathbb{Z}_q)$ that satisfy $Verify(pk, x, y_{\rho}, \pi_{\rho}) = 1$, for $\rho \in \{1, 2\}$. It is obvious to see that $y_1 = y_2$. Specifically, from the definition of the proof $G(sk, x)$, we can get

$$z_{\rho} = \beta + c_{\rho}/\alpha = \beta + H_1(g^{\beta}, x)/\alpha, \tag{15}$$

where $\rho \in \{1, 2\}$. Furthermore, the second verification equation yields $y_{\rho} = H_2(z_{\rho} \oplus x) = H_2((\beta + H_1(g^{\beta}, x)/\alpha) \oplus x)$ for both $\rho \in \{1, 2\}$, which implies $y_1 = H_2((\beta + H_1(g^{\beta}, x)/\alpha) \oplus x) = y_2$. Thus there is only one unique y on given x that can be proved to be valid with the proof π.
3. **Pseudorandomness**. We present the following Theorem 2 with respect to the proof of pseudorandomness property.

Theorem 2 *Suppose the (t, ϵ)-DDH assumption holds in a group $\mathbb{G}$ of prime order p. Our VRF scheme is a (t', ϵ')-secure VRF with n-bit input, where $t' \approx t$ and $\epsilon \geq \frac{1}{2^n}\epsilon'$.*

Proof. For the sake of contradiction, suppose there exists an adversary $\mathcal{A}$, running in time t', can distinguish the function $F(sk, x)$ from a random element r in group $\mathbb{G}$ with non-negligible advantage ϵ', i.e., $Adv^{\mathrm{DDH}}_{g,\alpha,\beta}(\mathcal{A}) \geq \epsilon'$, then we will build an simulator $\mathcal{B}$ that uses the advantage of $\mathcal{A}$ to break the DDH assumption with non-negligible probability.

- **Input to the reduction.** The simulator $\mathcal{B}$ is given a tuple $(g, g^{\alpha}, g^{\beta}, T) \in \mathbb{G}$, where T is either $g^{\alpha\beta} \in \mathbb{G}$ or a random element r in $\mathbb{G}$. Finally, $\mathcal{B}$ outputs 1 if $T = g^{\alpha\beta}$ and 0 otherwise.
- **Setup.** We assume that $\mathcal{A}$ will issue a challenge query on x^* to distinguish $F(sk, x^*)$ from a random element r in $\mathbb{G}$. $\mathcal{B}$ generates the public key $pk = \{g, p, q, h = g^{\alpha}, \gamma = g^{\beta}\}$ and the secret key $sk = \{g, p, q, h = g^{\alpha}, \alpha, \beta\}$, then sends the public key pk to $\mathcal{A}$.
- **Oracle Queries.** Without loss of generality, we assume that $\mathcal{A}$ makes at most Q queries and never repeats queries. Once $\mathcal{A}$ issues a query on x_i $(1 \leq i < Q)$ to $\mathcal{B}$, then $\mathcal{B}$ responses to the query x_i as follows:
  - If the query $x_i = x^*$(x^* is challenge query), then $\mathcal{B}$ aborts and outputs a uniformly random guess $b' \in \{0, 1\}$;
  - Otherwise, $\mathcal{B}$ processes as follows:
    1. choose a uniformly random value c_i from $\mathbb{Z}_q$, program $c_i = H_1(h^{\beta}, x_i)$, then response with c_i;
    2. define $z_i \equiv \beta + c_i/\alpha \bmod q$, then response with z_i;
    3. draw a uniformly random value u_i from $\mathbb{Z}_q$, program $H_2(z_i \oplus x_i) = u_i$ and define $y_i = u_i$, then response with u_i.

  Thus, $\mathcal{B}$ will send a tuple (z_i, c_i, u_i) as response values of the query x_i to $\mathcal{A}$ on this stage.
- **Challenge Query.** $\mathcal{A}$ issues a challenge query on x^*. If the challenge query is different from x^*, then $\mathcal{B}$ aborts and outputs a uniformly random guess $b' \in \{0, 1\}$. Otherwise, $\mathcal{B}$ outputs a value T^* which is either $F(sk, x^*)$ or a random element r in $\mathbb{G}$, then sends it to $\mathcal{A}$.
- **Guess.** Finally, $\mathcal{A}$ outputs a guess b' according to

$$b' = \begin{cases} 0, & T^* = r \\ 1, & T^* = F(sk, x^*) \end{cases} \tag{16}$$

and then $\mathcal{B}$ outputs the same b' as its guess.
- **Success probability.** We now analyze the probability that $\mathcal{B}$ wins the game. Let $\mathcal{B}_{win}$ denote the event that $\mathcal{B}$ wins the game. Similarly, $\mathcal{B}_{abort}$ and $\overline{\mathcal{B}_{abort}}$ are denoted as the events that $\mathcal{B}$ aborts and performs the simulation, respectively. On stage of challenge query, considering that $\mathcal{B}$ aborts only when the challenge query is different from x^*, we have thus $\Pr[\mathcal{B}_{abort}] = 1 - 1/2^n$. Moreover, if $\mathcal{B}$ aborts, then it outputs a uniformly random bit $b' \in \{0, 1\}$. It means that the success probability of $\mathcal{B}$ in this case relies on a flip of a coin $b' \in \{0, 1\}$, so we have $\Pr[\mathcal{B}_{win} | \mathcal{B}_{abort}] = 1/2$. On the other hand, if $\mathcal{B}$ does not abort, then $\mathcal{B}$ can calculate $c^* = H_1(T, x^*)$, $z^* \equiv \beta + c^*/\alpha \bmod q$ and $F(sk, x^*) = H_2(z^* \oplus x^*)$ in sequence since $T = g^{\alpha\beta}$. It is obvious that $F(sk, x^*)$ can be obtained from T. So, $\mathcal{B}$ wins the game in this case with the same advantage as $\mathcal{A}$, whose non-negligible advantage is ϵ'. Thus we have $\Pr[\mathcal{B}_{win} | \overline{\mathcal{B}_{abort}}] = 1/2 + Adv^{\mathrm{DDH}}_{g,\alpha,\beta}(\mathcal{A}) \geq 1/2 + \epsilon'$. Thus, the probability that $\mathcal{B}$ wins the game is

$$\begin{aligned}\Pr[B_{win}] =& \Pr[B_{win}|B_{abort}] \cdot \Pr[B_{abort}] + \\ & \Pr[B_{win}|\overline{B_{abort}}] \cdot \Pr[\overline{B_{abort}}] \\ \geq& \frac{1}{2} \cdot (1 - \frac{1}{2^n}) + (\frac{1}{2} + \epsilon') \cdot \frac{1}{2^n} \\ =& \frac{1}{2} + \frac{1}{2^n}\epsilon'.\end{aligned} \tag{17}$$

It means that $\mathcal{B}$ wins the game with non-negligible advantage $\epsilon \geq \frac{1}{2^n}\epsilon'$. However, since the (t, ϵ)-DDH assumption holds in group $\mathbb{G}$, there not exists such a simulator $\mathcal{B}$ that wins the game with non-negligible advantage ϵ. In other words, there not exists an adversary $\mathcal{A}$, running in time t', can distinguish the function $F(sk, x)$ from a random element r in group $\mathbb{G}$ with non-negligible advantage ϵ', so the theorem holds. We complete the proof of Theorem 2.

3.3 VRF construction over LHL

We now present the VRF construction over LHL (VRF-LHL) in order to further reduce the storage overhead of VRF. Let H_3 be a universal hash function that maps an element in $\mathbb{Z}_q$ to an v-bit bitstring. Our VRF-LHL that contains three polynomial-time algorithms is as follows:

1. **Setup**(1^κ). It takes the security parameter κ as input, and outputs the secret key $sk = \{g, p, q, \alpha, \beta\}$ and public key $pk = \{g, p, q, h = g^\alpha, \gamma = g^\beta\}$,where α and β are randomly chosen in $\mathbb{Z}_q$.
2. **Prove**(sk, x). It takes an integer x and the secret key sk as inputs, and then generates a value y with a corresponding proof π, which are defined as

$$\begin{cases} y &= F(sk, x) = H_3(\pi \oplus x), \\ \pi &= G(sk, x) \equiv (x - \alpha\gamma)/\beta \bmod q, \end{cases} \tag{18}$$

where the size of π is $\log(q)$ bits.

3. **Verify**(pk, x, y, π). It verifies whether the value y is correctly computed on given x by using the proof π along with the public key pk. Specifically, this algorithm will check

$$\begin{cases} \gamma^\pi h^\gamma &\equiv g^x \bmod p, \\ H_3(\pi \oplus x) &= y. \end{cases} \tag{19}$$

If two above equations holds, this algorithm outputs 1, and 0 otherwise.

We also provide the Algorithms 4 and 5 to illustrate the processing of the Prove and Verify modules, respectively. Obviously, the two algorithms are rather simple in structure and easy to be implemented in practice. So, our VRF-LHL have high computational efficiency.

Algorithm 4 Prove(sk, x)

Input: an integer x and the secret key $sk = \{g, p, q, \alpha, \beta\}$.
Output: a value y and a proof π.
1: $temp = x - \alpha * \gamma$;
2: $temp = temp/\beta$;
3: $\pi = Mod(temp, q)$;
4: $temp = BitXor(\pi, x)$;
5: $y = H_3(temp)$;
6: **Return** y, π.

Algorithm 5 Verify(pk, x, y, π)

Input: a pair (x, y), a proof π, and the public key $pk = \{g, p, q, h = g^\alpha, \gamma = g^\beta\}$.
Output: 1 or 0.
1: $temp = \gamma^\pi * h^\gamma$;
2: $x_1 = Mod(temp, p)$;
3: $x_2 = Mod(g^x, p)$;
4: $temp = BitXor(\pi, x)$;
5: $y_1 = H_3(temp)$;
6: **if** $x_1 - x_2 == 0$ and $y_1 - y == 0$ **then**
7: $temp = 1$;
8: **else**
9: $temp = 0$;
10: **end if**
11: **Return** $temp$.

We now prove that our VRF-LHL satisfies the required security properties, as follows:

1. **Provability**. For all pairs of keys $(pk = \{g, p, q, h = g^\alpha, \gamma = g^\beta\}, sk = \{g, p, q, \alpha, \beta\}) \in Setup(1^\kappa)$, and any integer x, from the definitions of $F(sk, x)$ and $G(sk, x)$, it follows immediately that

$$\gamma^\pi h^\gamma = (g^\beta)^{(x-\alpha\gamma)/\beta}(g^\alpha)^\gamma \equiv g^x \bmod p, \tag{20}$$

and $H_3(\pi \oplus x) = y$. That is, the value y can be deterministically obtained from the proof π, we have thus $Verify(pk, x, y, \pi) = 1$.
2. **Uniqueness**. Assume that there exists the public key $pk = \{g, p, q, h = g^\alpha, \gamma = g^\beta\}$, an integer x, and pairs of values $(y_\rho, \pi_\rho) \in (\mathbb{G}, \mathbb{Z}_q)$ that satisfy $Verify(pk, x, y_\rho, \pi_\rho) = 1$, for both $\rho \in \{1, 2\}$. It is obvious to see that $y_1 = y_2$. Specifically, from the first verification equation, we enable to yield

$$\gamma^{\pi_\rho} h^\gamma = g^{\beta\pi_\rho} g^{\alpha\gamma} = g^{\beta\pi_\rho + \alpha\gamma} \equiv g^x \bmod p, \tag{21}$$

and further get $\pi_\rho \equiv (x - \alpha\gamma)/\beta \bmod q$ for both $\rho \in \{1, 2\}$ according to the multiplication operation rules of the same base power. Meanwhile, the second verification equation yields $y_\rho = H_3(\pi_\rho \oplus x) = H_3((x - \alpha\gamma)/\beta \bmod q \oplus x)$ for

Table 1 Performance comparisons of our three VRF schemes with previous schemes

Item		[26]	[14]	VRF-RSA	VRF-DDH	VRF-LHL
Size	sk	$(2n+5)\|\mathbb{Z}_q\|$	$\|\mathbb{Z}_q\|$	$2\|\mathbb{Z}_N\|$	$2\|\mathbb{Z}_q\|$	$2\|\mathbb{Z}_q\|$
	pk	$(2n+7)\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$2\|\mathbb{Z}_N\|$	$2\|\mathbb{G}\|$	$2\|\mathbb{G}\|$
	π	$(2n+4)\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$\|\mathbb{Z}_N\|$	$2\|\mathbb{Z}_q\|$	$\|\mathbb{Z}_q\|$
	y	$\|\mathbb{G}_T\|$	$\|\mathbb{G}_T\|$	$\|\mathbb{Z}_N\|$	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$
Overhead	sk	8352B	32B	768B	64B	64B
	pk	100992B	384B	768B	768B	768B
	π	99840B	384B	384B	64B	32B
	y	768B	768B	384B	384B	384B
Computing time	Prove	$[H]+(2n+5)[E_p]+[B]$	$[E_p]+[B]$	$[H]+[E_N]$	$2[H]+[E_p]$	$[H]$
	Verify	$[H]+(4n+6)[E_p]+3[B]$	$[E_p]+3[B]$	$[H]+[E_N]$	$2[H]+2[E_p]$	$[H]+3[E_p]$
Cryptography primitive		Bilinear map on elliptic curve	Bilinear map on elliptic curve	RSA signature	Schnorr signature	ELGamal signature

both $\rho \in \{1,2\}$, which implies $y_1 = H_3((x-\alpha\gamma)/\beta \bmod q \oplus x) = y_2$. So, there is only one unique y on given x that can be proved to be valid with the proof π.

3. **Randomness**. The famous LHL lemma simply states that a randomness extractor can be constructed from an universal hash family, which provides an appropriate method to prove the randomness property. Specifically, we propose Theorem 3 with respect to the randomness proof.

Theorem 3 *Suppose the H_3 is a $1/2^v$-universal hash function, then our VRF scheme is a (m,ϵ)-secure VRF with n-bit input under LHL, where $\epsilon \geq \frac{1}{2}\sqrt{2^{v-m}}$.*

Proof. According to LHL, one universal hash function leads to an efficient randomness extractor. Specifically, if X is a distribution of min-entropy m over space $\mathcal{X}$, $\mathcal{H}$ is a family of functions from $\mathcal{X}$ to $\{0,1\}^v$, and H is a $1/2^v$ -universal hash function randomly chosen from $\mathcal{H}$, then we will construct an (m,ϵ)-extractor that satisfies the statistical distance between $H(X)$ and the uniform distribution U_v on $\{0,1\}^v$ is bounded by $\frac{1}{2}\sqrt{2^{v-m}}$. That is, there not exists an algorithm can distinguish the extracted randomness $H(X)$ from uniform distribution U_v with advantage greater than $\epsilon = \frac{1}{2}\sqrt{2^{v-m}}$. Therefore, according to LHL, if the H_3 in our VRF scheme is a $1/2^v$-universal hash function, then we will achieve a (m,ϵ)-secure VRF with n-bit input. That is, there not exists an adversary can distinguish the function $F(sk,x)$ from a random value with advantage greater than ϵ. We complete the proof of Theorem 3.

4 Comparisons

In this section, we compare our three VRF schemes, VRF-RSA, VRF-DDH and VRF-LHL, with two representative schemes proposed in the literature [14, 26] in terms of space and computation complexity. The comparison results are summarized in Table 1.

Now, let us explain some notations used in Table 1. For analysis of storage overheads, $|\mathbb{Z}_N|$, $|\mathbb{Z}_q|$, $|\mathbb{G}|$ and $|\mathbb{G}_T|$ represent the size of elements in groups $\mathbb{Z}_N$, $\mathbb{Z}_q$, $\mathbb{G}$ and $\mathbb{G}_T$, respectively. To measure the computational cost, we denote by $[H]$ the cost of a hash operation. Similarly, the cost of an exponential operation in $\mathbb{Z}_N$ and $\mathbb{G}$ are denoted as $[E_N]$ and $[E_p]$, respectively. Besides, the cost of calculating bilinear map on elliptic curve is denoted as $[B]$ in the schemes [14, 26]. We omit the algebraic calculation in $\mathbb{Z}_q$ since it is very efficient.

In the schemes [14, 26], assuming that the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is constructed on the curve $y^2 \equiv x^3 + x \bmod p$ over the field $\mathbb{F}_p$ for some prime $p \equiv 3 \bmod 4$, where $\mathbb{G}$ and $\mathbb{G}_T$ are two multiplicative groups of order q and the integer q is a prime factor of $p+1$. Under 128-bit security strength, it is required that the RSA modulus N is 3072 bits, i.e., $|\mathbb{Z}_N|$=3072 bits. Correspondingly, the size of elements in $\mathbb{G}$ is also 3072 bits, i.e., $|\mathbb{G}|$= 3072 bits. When the embedding degree of the curve is 2, the size of elements in $\mathbb{G}_T$ should be 6144 bits, i.e., $|\mathbb{G}_T| = 6144$ bits. The prime q of the cyclic subgroup $\mathbb{Z}_q$ needs only to be 256-bit integer, i.e., $|\mathbb{Z}_q|$= 256 bits.

From Table 1, it can be seen that the schemes [14, 26] construct their VRFs from bilinear map primitives, while our three schemes are directly based on RSA, Schnorr and ELGamal signatures, respectively. The storage overheads of four major entities, the secret key sk, the public key pk, the proof π and the VRF's output y, are also listed in Table 1. In the scheme [26], it is easy to see that the sizes of three entities, sk, pk and π, are linear correlation with security strength n, but y is of a fixed size. In addition, all of the four entities are still fixed size in the scheme [14] and our schemes. As shown in Table 1, the storage overheads of sk and pk in our schemes are slightly larger than these of [14] but smaller

than [26] when n is 128 bits. However, the storage overheads of π and y in our schemes are much smaller than these of the schemes [14, 26]. For example, the storage overheads of π and y in VRF-LHL are only 32 and 384 bytes, respectively. Our VRF-LHL significantly improves both of the sizes of π and y. Therefore, the comparison results indicates that our schemes have lower storage overheads in practice.

Next, we compare the computation complexity of **Prove** and **Verify** algorithms. Obviously, the computing time of **Prove** and **Verify** in [14] are significantly smaller than these of [26], which are linear correlation with n. Moreover, both **Prove** and **Verify** of their schemes require high computational cost since they all use bilinear map on elliptic curve. However, our VRF schemes only involve exponential and hash operations in single group without bilinear map. For example, the **Prove** in VRF-DDH only needs to perform one exponential operation in $\mathbb{G}$ and two hash operations, so that its computational overhead is $[E_p] + 2[H]$. So, our schemes should have lower computational overheads.

5 Application of VRFs in blockchain

In this section, we show a specific application of our VRF constructions in efficient consensus protocol of blockchain. Specifically, we demonstrate the detailed process of our VRF-based consensus protocol design and its application in Algorand [9]. A general approach is also illustrated to integrate our VRF constructions with block structure in blockchain. Besides, we show a script implementation of our VRF constructions in blockchain. Further, the experimental evaluation of our VRF constructions is conducted in terms of storage and computational overheads. Finally, we compare the VRF scheme used in Algorand with our VRF-LHL scheme from the aspects of performance and security.

5.1 VRF-based consensus protocols

In order to address the problem of low efficiency and high energy consumption in the PoW-based protocol of Bitcoin, Turing Award winner Gilad et al. [9] proposed a new VRF-based consensus protocol called Algorand in 2016. Roughly speaking, Algorand uses a cryptographic sortation algorithm to select committee members, which are responsible for completing consensus process on some candidate block. Figure 3 illustrates the consensus process based on the cryptographic sortation algorithm in Algorand.

Algorand consensus [9] is mainly divided into two stages, as shown in Fig. 3. In the first stage, the cryptographic sortation algorithm will be executed to randomly choose a small fraction of users from the entire network in a private and non-interactive manner, thereby forming a proposer committee whose members are in charge of proposing new candidate blocks. In the second stage, another subset of users will be randomly selected to form a verifier committee based on the algorithm, then the members reach BA* consensus on one of the proposed candidate blocks.

Obviously, the cryptographic sortation algorithm, that can ensure the randomness of the selected committee members, is required to be executed at the beginning of each stage in Algorand [9]. The algorithm implementation mainly relies on VRF, which is essentially an efficient RF with a non-interactive verifiable mechanism. Instead of PoW-based mining, using VRF can efficiently avoid the repeated calculations of nodes, randomly select block proposers, and easily verify the proposed block. Therefore, VRF is very suitable for designing efficient consensus protocols in blockchain.

We now give a general approach to integrate our VRF constructions with block structure in blockchain. Generally speaking, in the design of VRF-based protocols, nodes are required to implement VRF which takes a secret key sk and data x as inputs and generates a unique y and a proof π. Figure 4 illustrates the integration of our VRF constructions with the block header in the blockchain.

Specifically, as shown in Fig. 4, the nNonce field is used to store π, which acts as the proof of the block. The fields except the nNonce in the block header is used to generate x together, i.e., x=$nVersion||hashPrevBlock||hashMerkleRoot||nTime||nBits$, where || denotes the concatenation operator. The VRF's output y is stored in the hashPrevBlock field. In addition, sk and pk are stored in the wallet of the node and the scriptPubkey field (see Fig. 5) of the coinbase transaction in the block, respectively.

Next, taking VRF-RSA as an example, we demonstrate the detailed process of the VRF-based consensus protocol design as follows:

Step 1. The node collects pending transactions of the whole network within a period of time, and then adds a coinbase transaction which is used for issuing new rewards. The secret key $sk = \{d, N\}$ is stored in its wallet, and the corresponding $pk = \{e, N\}$ is filled in the scriptPubkey field. Next, the node verifies and packages these transactions.

Step 2. The node calculates the hashMerkleRoot of these transactions, and then it fills the informations of the nVersion, hashPrevBlock, nTime and nBits fields. Next, the node calculates the concatenation of these fields to obtain x.

Step 3. The node calculates $\pi \equiv x^d \bmod N$ by using $sk = \{d, N\}$, then writes π into the nNonce field.

Step 4. The node uses the double SHA-256 to calculate the output of the block header, i.e., $y = Hash(\pi \oplus x)$. Eventually, the node forms a candidate block.

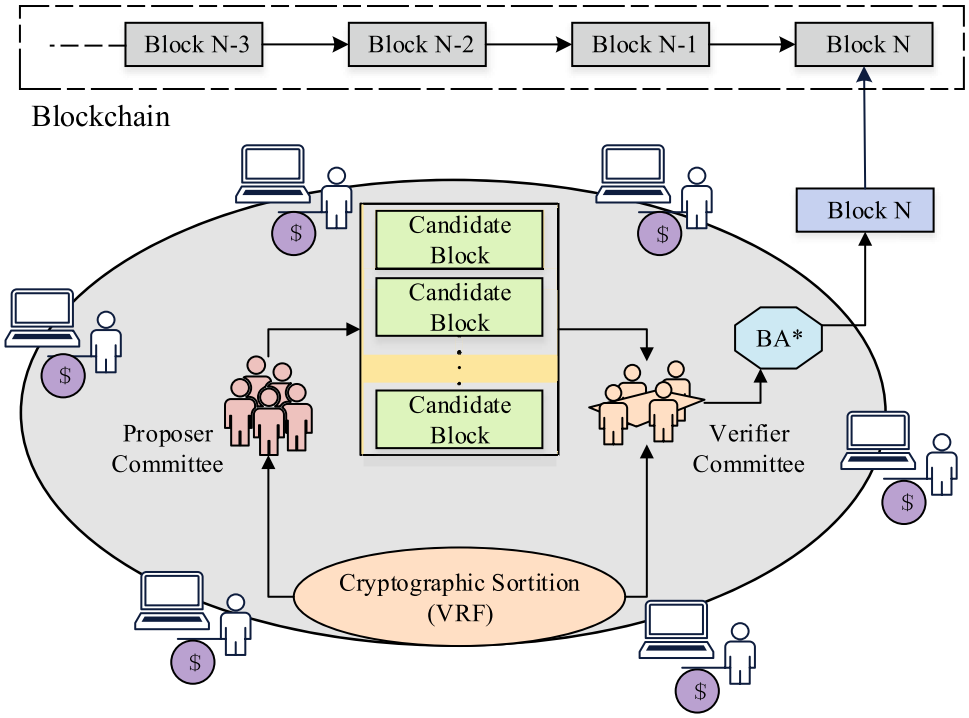

Fig. 3 The consensus process based on the cryptographic sortation algorithm in Algorand

Subsequently, our VRF construction can be applied into the cryptographic sortation algorithm, which helps to randomly select the proposer and verifier committees in Algorand. Once a candidate block is proposed by the proposer committee and is broadcast on the entire network, the verifier committee can verify the validity of this block based on the verifiable mechanism in VRF. In other words, the verifiers who hold π and pk can confirm that the proposer indeed has the privilege to propose the block.

More specifically, the workflow of the block verification is described as follows: Firstly, the verifier calculates x based on the block header, and then obtains π and y that are stored in the nNonce and hashPreBlock fileds, respectively. Secondly, the verifier gets the $pk = \{e, N\}$ of the proposer from the scriptPubkey field in the coinbase transaction. Finally, the verifier will check whether the equations $\pi^e \bmod N \equiv x$ and $y = Hash(\pi \oplus x)$ holds. If these two equations holds, then the verifier confirms that the block proposer has the privilege and the block is valid.

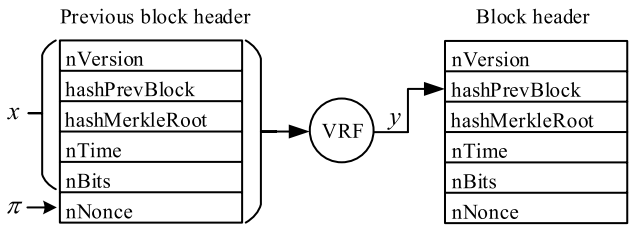

Fig. 4 The integration of our VRF constructions with the block header in blockchain

5.2 Improvement of scripting system

Blockchain script is a simple, Forth-like, stack-based program, which is specifically designed for transaction verification. The scripting language is intentionally not Turing Complete, i.e., it has no loops or complex flow control capabilities. Such a restricted script can easily resist attacks with malicious code so as to ensure the security and efficiency of blockchain transactions. Besides, the script is automatically executed by the interpreter in blockchain node, which significantly reduces user-side computational overheads. For

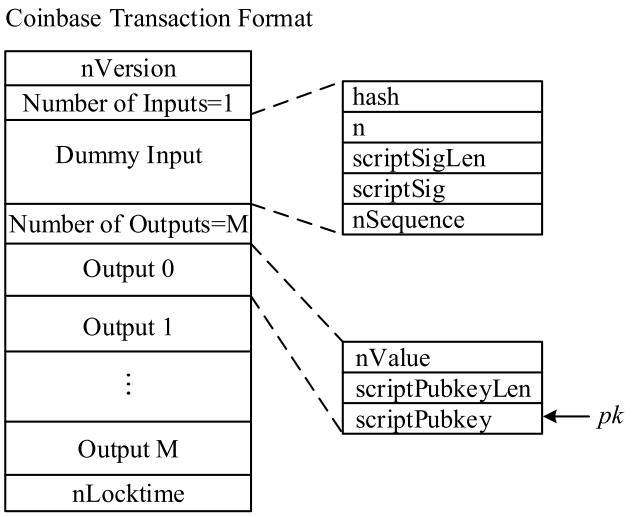

Fig. 5 Storage location of pk in coinbase transaction

Table 2 The definitions of existing and new added opcodes in the block verification script

Type	Word	Opcode	Input	Output	Description
Existing opcode	OP_DUP	0x76	a	a, a	Duplicates the top stack item a.
	OP_XOR	0x86	a, b	out	Calculates Boolean XOR of a and b.
	OP_EQUAL	0x87	a, b	True/false	Returns 1 if a and b are equal, 0 otherwise.
	OP_EQUALVERIFY	0x88	a, b	None/fail	The script execution proceeds if a and b are equal, it terminates with an error otherwise. Note that a and b are finally removed from the stack.
	OP_HASH256	0xaa	in	hash	The input is hashed two times with SHA-256.
New opcodes	OP_RSA_PK	0xb1	None	e, N	Gets RSA's public key $\{e, N\}$ from the scriptPubkey field in coinbase transaction, then e and N are pushed onto the stack.
	OP_VRF_INPUT	0xb2	None	VRF's input x	Calculates VRF's input x from the block header, then x is pushed onto the stack.
	OP_EXP_MOD	0xb3	a, b, c	out	Pops the top three items a, b and c, then b to the power of a under modulus c is pushed onto the stack.

example, Bitcoin uses such a scripting system to realize the cryptographic process of transactions. Meanwhile, five different types of signature scripts are designed to increase the flexibility of cryptographic constructions. In view of this, we can complete the execution of VRF algorithm based on the scripting system so as to provide secure and efficient block verification in the consensus process.

The structure of blockchain script is relatively simple and flexible. Generally, a script consists of multiple data values and operators, where the data values are enclosed in angle brackets and the operator names start with the string "OP_". The operator is encoded in a single byte, and the byte value is called the opcode of the operator. For example, the most common signature script, Pay to Public Key Hash (P2PKH), is defined in Bitcoin transaction as

$$\underbrace{\langle \text{sig} \rangle \langle \text{PubK} \rangle}_{\text{scriptSig}} \quad \underbrace{\begin{array}{l}\text{OP\_DUP OP\_HASH160 } \langle \text{PubKHash} \rangle \\ \text{OP\_EQUALVERIFY OP\_CHECHSIG}\end{array}}_{\text{scriptPubKey}}.$$

The P2PKH script includes two parts, scriptSig and scriptPubkey, which are used to verify the validity of a transaction. The values, corresponding to $\langle \text{sig} \rangle$, $\langle \text{PubK} \rangle$ and $\langle \text{PubKHash} \rangle$, will be pushed onto the stack, and the opcodes, OP_DUP, OP_HASH160, OP_EQUALVERIFY and OP_CHECKSIG, are used to invoke specific functions. For example, the OP_DUP is a duplication operator that duplicates the top stack item.

In order to automatically implement block verification based on VRF-RSA, three new opcodes are required to be defined and some existing opcodes are utilized, as shown in Table 2. The new opcode, OP_EXP _MOD, is a modular exponential operator. In addition, another two new opcodes, OP_RSA_PK and OP_VRF _INPUT, are used to obtain RSA's public key $pk = (e, N)$ from the scriptPubkey field in coinbase transaction (see Fig. 5) and the VRF's input x from the block header (see Fig. 4), respectively. Then, the block verification script based on VRF-RSA is defined as follows:

$$\underbrace{\begin{array}{l}\langle \pi \rangle \text{ OP\_DUP OP\_RSA\_PK OP\_EXP\_MOD} \\ \text{OP\_DUP OP\_VRF\_INPUT OP\_EQUALVERIFY}\end{array}}_{\text{scriptProof}} \quad \underbrace{\begin{array}{l}\text{OP\_XOR OP\_HASH256} \\ \langle y \rangle \text{ OP\_EQUALVERIFY}\end{array}}_{\text{scriptHash}}.$$

Obviously, the script consists of two parts, scriptProof and scriptHash. Specifically, the validity of the VRF's input x can be automatically checked by executing scriptProof, which is stored in the nNonce field of the block header. Combined with scriptProof, scriptHash can be used to verify the correctness of the VRF's output y and stored in the hashPreBlock field of the block header. Obviously, the script is responsible for automatic execution of **Verify** algorithm in VRF, which helps to realize more secure and efficient block verification.

Next, we illustrate the execution process of the block verification script. The stack state during the script execution is shown in Table 3. The script is processed from left to right and its execution proceeds as follows:

Step 1-5. The stack is initialized and emptied, then the proof $\langle \pi \rangle$ is pushed onto the stack and duplicated by the OP_DUP operator. Next, the OP_RSA_PK operator obtains RSA's public key $\langle e \rangle$ and $\langle N \rangle$ from the scriptPubkey field in coinbase transaction (see Fig. 5) and pushes them onto the stack. Then, the OP_EXP_MOD operator calculates $\langle \pi^e \bmod N \rangle$ and then pushes it onto the stack.

Step 6-8. $\langle \pi^e \bmod N \rangle$ is duplicated by the OP_DUP operator, then the OP_VRF_INPUT operator calculates the VRF's input x from the block header (see Fig.4) and pushes it onto the stack. Next, the OP_EQUALVERIFY operator compares $\langle \pi^e \bmod N \rangle$ with $\langle x \rangle$. If they are equal, the script execution proceeds. Otherwise, it terminates with an error.

Step 9-10. The OP_XOR operator performs a XOR operation on $\langle \pi \rangle$ and $\langle \pi^e \bmod N \rangle$. Next, the output of the

Table 3 The stack state during the execution of the block verification script

Step	Stack	Script	Description
1	Empty	$\langle\pi\rangle$ OP_DUP OP_RSA_PK OP_EXP_MOD OP_DUP OP_VRF_INPUT OP_EQUALVERIFY OP_XOR OP_HASH256 $\langle y\rangle$ OP_EQUAL	scriptProof and scriptHash are combined.
2	$\langle\pi\rangle$	OP_DUP OP_RSA_PK OP_EXP_MOD OP_DUP OP_VRF_INPUT OP_EQUALVERIFY OP_XOR OP_HASH256 $\langle y\rangle$ OP_EQUAL	$\langle\pi\rangle$ is added to the stack.
3	$\langle\pi\rangle\langle\pi\rangle$	OP_RSA_PK OP_EXP_MOD OP_DUP OP_VRF_INPUT OP_EQUALVERIFY OP_XOR OP_HASH256 $\langle y\rangle$ OP_EQUAL	Top stack item is duplicated.
4	$\langle\pi\rangle\langle\pi\rangle\langle e\rangle\langle N\rangle$	OP_EXP_MOD OP_DUP OP_VRF_INPUT OP_EQUALVERIFY OP_XOR OP_HASH256 $\langle y\rangle$ OP_EQUAL	$\langle e\rangle$ and $\langle N\rangle$ are added to the stack.
5	$\langle\pi\rangle\langle\pi^e \bmod N\rangle$	OP_DUP OP_VRF_INPUT OP_EQUALVERIFY OP_XOR OP_HASH256 $\langle y\rangle$ OP_EQUAL	$\langle\pi^e \bmod N\rangle$ is added to the stack.
6	$\langle\pi\rangle\langle\pi^e \bmod N\rangle\langle\pi^e \bmod N\rangle$	OP_VRF_INPUT OP_EQUALVERIFY OP_XOR OP_HASH256 $\langle y\rangle$ OP_EQUAL	Top stack item is duplicated.
7	$\langle\pi\rangle\langle\pi^e \bmod N\rangle\langle\pi^e \bmod N\rangle\langle x\rangle$	OP_EQUALVERIFY OP_XOR OP_HASH256 $\langle y\rangle$ OP_EQUAL	The VRF's input $\langle x\rangle$ is added to the stack.
8	$\langle\pi\rangle\langle\pi^e \bmod N\rangle$	OP_XOR OP_HASH256 $\langle y\rangle$ OP_EQUAL	The script execution proceeds if the top two stack items are equal.
9	$\langle\pi \oplus \pi^e \bmod N\rangle$	OP_HASH256 $\langle y\rangle$ OP_EQUAL	Boolean XOR of the top two items is calculated.
10	$\langle Hash(\pi \oplus \pi^e \bmod N)\rangle$	$\langle y\rangle$ OP_EQUAL	Top stack item is hashed.
11	$\langle Hash(\pi \oplus \pi^e \bmod N)\rangle\langle y\rangle$	OP_EQUAL	$\langle y\rangle$ is added to the stack.
12	True/false	Empty	Equality is checked between the top two stack items.

SHA-256 of $\langle\pi \oplus \pi^e \bmod N\rangle$ is generated and pushed onto the stack by the OP_HASH256 operator.

Step 11-12. The VRF's output y is pushed onto the stack. Next, the OP_EQUAL operator pops up and compares $\langle Hash(\pi \oplus \pi^e \bmod N)\rangle$ with $\langle y\rangle$. If they are equal, a value True is pushed onto the stack. It means that the block verification is successful. Otherwise, False is pushed onto the stack, the verification fails.

5.3 Performance analysis

We now analyze the performance of our VRF constructions in the design of blockchain consensus protocols in terms of storage and computational overheads. In VRF-RSA, let the size of the modulus N be 1024 bits (Considering that the current consensus time of Bitcoin is about 10 minutes, such a short period of time is sufficient for VRF to resist adversarial attacks). Correspondingly, the proof π needs to be 128-byte integer. Figure 6 illustrates the block header fields and their sizes in the PoW-based and VRF-based protocols, respectively.

Specifically, as shown in Fig. 6, the nNonce field is used to store the scriptProof that contains 128-byte π and 6-byte opcodes in the VRF-based protocol. The scriptHash consists of 32-byte y and 3-byte opcodes, which is stored in the hashPreBlock filed. In addition, the other fields in the block header remain unchanged. So, the block header size in the VRF-based protocol is larger than that in the PoW. However, comparing with the block size (870K bytes), the 133-byte increase of the block header is very small, which will lightly affect the number of transactions in the block.

Another two VRF constructions, VRF-DDH and VRF-LHL, will bring more efficient storage overheads. For example, under 128-bit security strength, the proof π in $\mathbb{Z}_q$ only needs to be 32-byte integer. So, the scriptProof in VRF-DDH contains 64-byte π and 6-byte opcodes, i.e., it requires 70-byte storage overhead. Moreover, the scriptProof in VRF-LHL contains 32-byte π and 6-byte opcodes, so, its storage overhead can be further reduced to 38 bytes.

The PoW-based protocol requires high computing power and energy consumption. Taking Bitcoin as an example, the Bitcoin's computing power has touched a new high of around 140 exahashes[3] per second (EH/s). Meanwhile, the Cambridge Bitcoin Electricity Consumption Index (CBECI) shows that the estimated power to run Bitcoin has reached 10.97 gigawatts (GW) a day. However, the VRF-based protocol only needs to perform VRF operation once, which avoids the repeated calculations of the double SHA-256 for the block header in the PoW. Therefore, the VRF-based protocol can greatly reduce the computational overheads and save the energy consumption.

[3] one exahash is one quintillion hashes, i.e., 1 EH = 10^{18} hashes.

nVersion	4 bytes
hashPrevBlock	32 bytes
hashMerkleRoot	32 bytes
nTime	4 bytes
nBits	4 bytes
nNonce	4 bytes

(a) The block header used by PoW

nVersion	4 bytes
scriptHash	**35** bytes
hashMerkleRoot	32 bytes
nTime	4 bytes
nBits	4 bytes
scriptProof	**134** bytes

(b) The block header used by VRF

Fig. 6 The block header fields and their sizes in the PoW-based and VRF-based protocols, respectively

In addition, the VRF's output y that is stored in the hashPreBlock field can maintain the chain structure of blockchain. Considering the $sk = \{d, N\}$ of each node is unique, π that is obtained from $\pi \equiv x^d \bmod N$ should also be different. So, the probability that the outputs y of two distinct blocks will be same is extremely negligible since the collision resistance property of cryptographic hash function. In other words, y can be securely considered as a unique identifier of the block.

The unchanged hashMerkleRoot field stores the root hash of Merkle tree that is formed by transactions in blockchain. Once a transaction in the block is maliciously forged or tampered with, the root hash will be changed dramatically. Therefore, the hashMerkleRoot field is associated with transactions in the block to ensure their integrity.

5.4 Experimental evaluation

We conduct experiments to evaluate the efficiency of the proposed VRF schemes by using Wolfram Mathematica 9.0.1.0 software. All the programs are executed on a Windows 10 (64-bit) PC with Intel(R) Core(TM) i5-4590S CPU

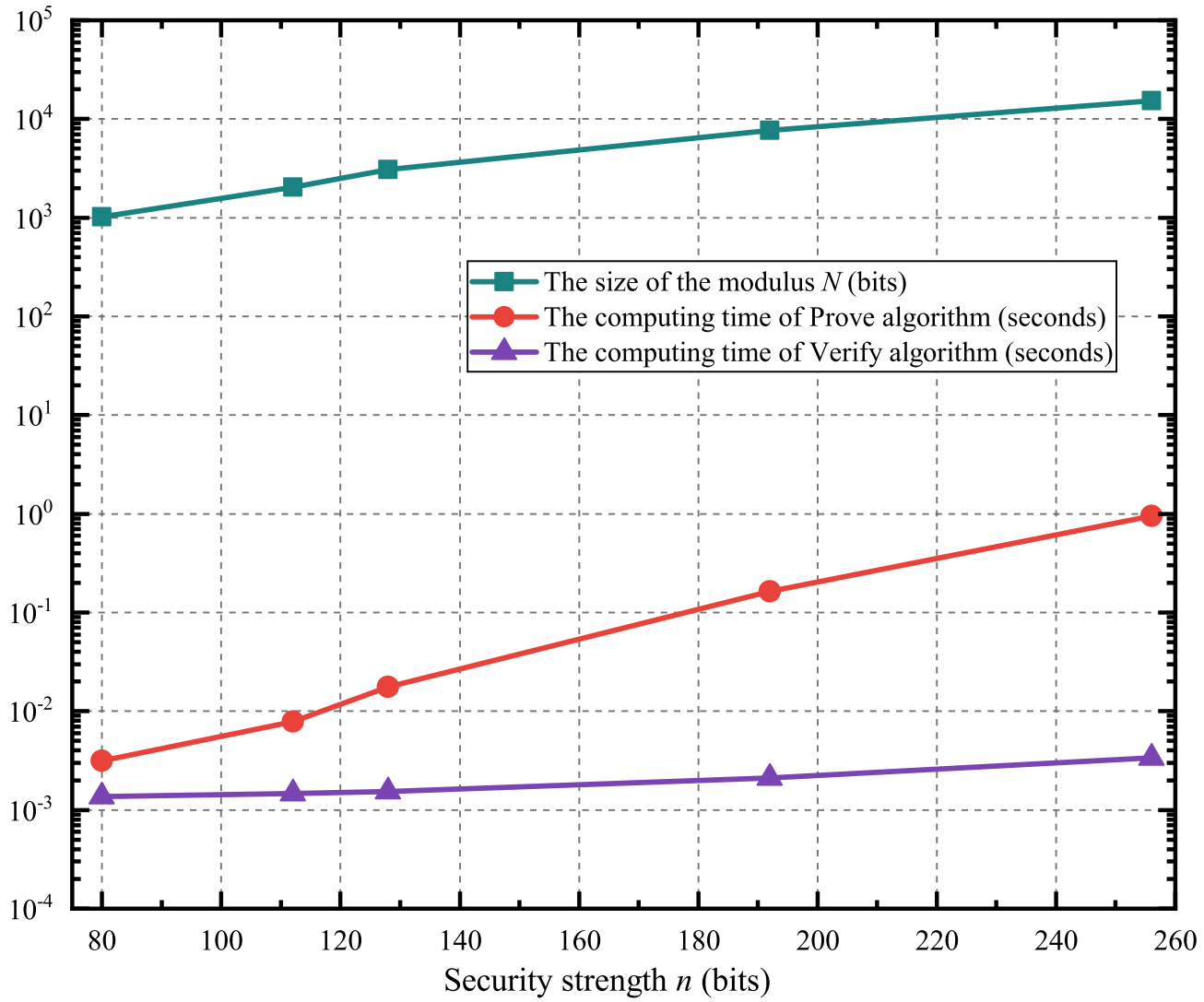

Fig. 7 Computing time of **Prove** and **Verify**, and the size of modulus N under different security strengths

@ 3.00GHz processor and 4G DDR3-RAM. Let hash function be SHA-256 cryptographic algorithm and the value of public exponent e be 65537. Firstly, we test the computational overhead of VRF-RSA under the different security strengths n, which is from 80 to 260. Figure 7 shows computing time of **Prove** and **Verify** algorithms in VRF, and the size of modulus N under five common security strengths n in logarithmic coordinates. Obviously, the larger the size of N, the higher the computational overheads of the two algorithms. In addition, the computing time of **Verify** is much shorter than that of **Prove**, which can greatly reduce the block verification cost of the verifiers.

In order to show the uptrend of curves more clearly, the general numerical ordinate is used to represent the computing time. As shown in Fig. 8, we demonstrate the relationship between the computing time of **Prove / Verify** and the security strength n. Obviously, the computing time of the two algorithms is an approximate quadratic function of n. When the security strength reaches 256 bits, the computing time of **Prove** and **Verify** is less than 1 second and 0.0035 seconds, respectively. Therefore, the computational overheads of VRF-RSA can be reduced to the millisecond level.

Next, we compare the computational overheads of our three VRF schemes under 128-bit security strength. The experimental computation overheads of **Prove** and **Verify** in our schemes are listed in Table 4. Obviously, **Prove** and **Verify** need only to perform a few exponential and hash operations, and their computational overheads can reach the millisecond level. For example, the **Prove** in VRF-DDH only needs to perform one 256-bit exponential operation in $\mathbb{G}$ and two SHA-256 operations, its time complexity and computational overhead are $[E_p] + 2[H]$ and 0.00230161 seconds, respectively.

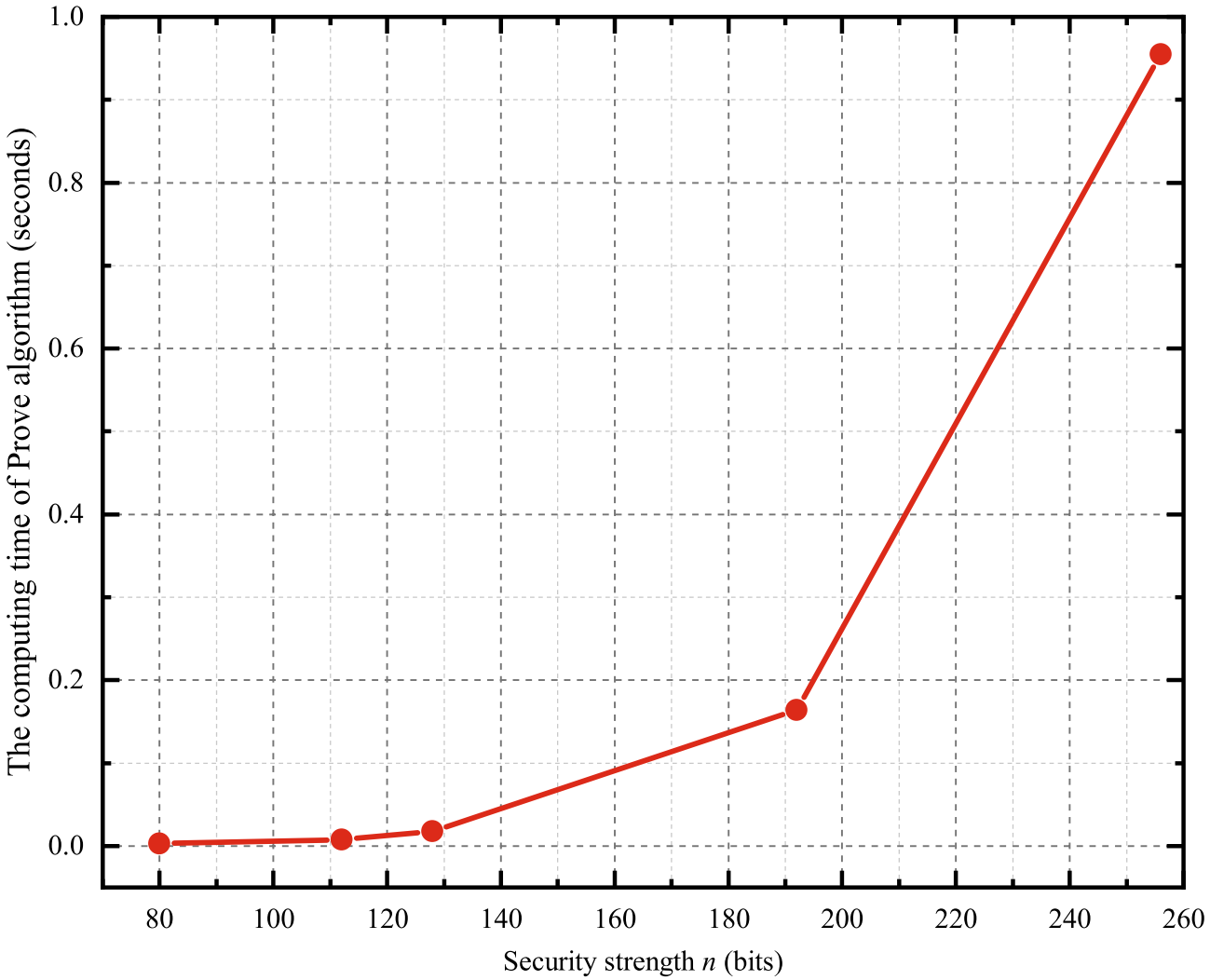

Fig. 8 The relationship between the computing time of **Prove / Verify** and the security strength n

Table 4 The experimental computation overheads of **Prove** and **Verify** in three VRF schemes

VRF Scheme	Algorithm	Time Complexity	Computational Overhead (second)
VRF-RSA	Prove	$[E_N] + [H]$	0.01771249
	Verify	$[E_N] + [H]$	0.00153443
VRF-DDH	Prove	$[E_p] + 2[H]$	0.00230161
	Verify	$2[E_p] + 2[H]$	0.00280243
VRF-LHL	Prove	$[H]$	0.00006672
	Verify	$3[E_p] + [H]$	0.02947671

In VRF-RSA, it is easy to see that **Prove** and **Verify** have the same time complexity. However, the computational overhead of **Verify** is much smaller than that of **Prove** since the bit length of the selected e (17 bits) is far less than that of the private exponent d. Moreover, considering that only one SHA-256 operation needs to be performed in VRF-LHL, the time complexity of **Prove** is $[H]$ and the computational overhead can be reduced to the microsecond level. However, **Verify** needs to perform one 3072-bit exponential operation in $\mathbb{G}$, so its computational overhead reaches 0.02947671 seconds. Therefore, three proposed VRF schemes have low computational overheads in practical applications and can meet the efficiency requirements of consensus protocols in blockchain.

5.5 Performance comparisons

Since Algorand was proposed in [9], VRF attracts wide attention in bockchain. Actually, Algorand did not give a specific VRF scheme, but used the existing scheme (called GNPR-VRF) proposed by Goldberg et al. [35]. We also compare their scheme with our VRF-LHL sche-me from two aspects of performance and security. Firstly, GNPR-VRF is implemented over Curve25519 [36] and the curve equation can be expressed as $y^2 = x^3 + 486662x^2 + x$. In this setting, GNPR-VRF was proven to reach approximately 128-bit security on random oracle model under the DDH assumption. Referring to Sect. 4, VRF-LHL also achieves 128-bit or higher security based on LHL lemma.

We firstly compare the storage overheads of these two schemes. GNPR-VRF uses the above elliptic curve over $\mathbb{F}_q$, where q is a 256-bit prime. Any point (x, y) on the curve is represented by 512 bits, where $x, y \in \mathbb{F}_q$. However, VRF-LHL operates in finite field $\mathbb{F}_p$ of prime order p, which is required to be a 3072-bit integer. Thus, the size of the GNPR-VRF's output y is 32 bytes, which is lower than that of VRF-LHL (384 bytes). However, the proof π in VRF-LHL needs only 32-byte storage overhead, which is lower than that in GNPR-VRF (80 bytes). Therefore, our VRF-LHL scheme has lower storage overhead for π.

We next evaluate the computation time of **Prove** and **Verify** algorithms. Firstly, compared with VRF-LHL, the **Verify** in GNPR-VRF needs more (three hash and four exponent) operations. On the other hand, the **Verify** in GNPR-VRF is more efficient than that of VRF-LHL as a result that GNPR-VRF operates in $\mathbb{F}_q$ of 256-bit q. Moreover, the **Prove** in GNPR-VRF involves three hash operations and three exponential operations in $\mathbb{F}_q$, but VRF-LHL only requires one hash operation. So the **Prove** in VRF-LHL has lower computational overheads than that in GNPR-VRF.

Finally, GNPR-VRF used in Algorand is based on elliptic-curve cryptography (ECC). It employs so special curve that is completely different from the curves used in cryptography textbooks. Although the advantage of the scheme on ECC is small storage space, developers need to have a good mathematical background on elliptic curve and abstract algebra. In contrast to it, our VRF-LHL scheme is entirely based on finite-field cryptography, which has relatively low development difficulty in practical applications.

6 Conclusion

In this paper, we propose three simple and efficient VRF constructions which are practically applicable to the consensus protocol design of blockchain. We also provide full security analysis of our constructions. Furthermore, we show a specific application of our constructions in the famous Algorand consensus protocol, and analyze the performance of our constructions in VRF-based consensus protocol design in terms of storage and computation overheads. The performance analysis and experimental evaluation illustrate that the designed protocol based on our VRF constructions can effectively reduce the computational resources of PoW-based protocols and improve consensus efficiency.

Acknowledgements This work was supported by the National Key Technologies R&D Programs of China (2018YFB1402702) and the National Natural Science Foundation of China (61972032).

References

1. Zheng Z, Xie S, Dai H, Chen X, Wang H (2017) An overview of blockchain technology: Architecture, consensus, and future trends. In 2017 IEEE International Congress on Big Data, BigData Congress 2017, Honolulu, HI, USA. IEEE Computer Society pp 557–564
2. Li Y, Shi W, Kumar M, Chen J (2018) Dycrem: Dynamic credit risk management using edge-based blockchain. In 2018 IEEE/ACM Symposium on Edge Computing, SEC 2018, Seattle, WA, USA. IEEE pp 344–346
3. Wu Y, Lu Z, Yu F, Luo X (2019) Rapid consortium blockchain for digital right management. In Genetic and Evolutionary Computing - Proceedings of the Thirteenth International Conference on Genetic and Evolutionary Computing, ICGEC. Qingdao, China, vol. 1107 of Advances in Intelligent Systems and Computing, Springer pp 447–454

4. Jabbar R, Fetais N, Krichen M, Barkaoui K (2020) Blockchain technology for healthcare: Enhancing shared electronic health record interoperability and integrity. In IEEE International Conference on Informatics, IoT, and Enabling Technologies, ICIoT 2020, Doha, Qatar. IEEE pp 310–317
5. Cho EM, Perera MNS (2020) Efficient certificate management in blockchain based internet of vehicles. In 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID 2020, Melbourne, Australia. IEEE pp 794–797
6. Nakamoto S (2019) Bitcoin: A peer-to-peer electronic cash system. Tech. rep, Manubot
7. Pass R, Seeman L, Shelat A (2017) Analysis of the blockchain protocol in asynchronous networks. In Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France. Proceedings, Part II (2017), vol. 10211 of Lecture Notes in Computer Science pp 643–673
8. Hazari SS, Mahmoud QH (2020) Improving transaction speed and scalability of blockchain systems via parallel proof of work. Future Internet 12(8):125
9. Gilad Y, Hemo R, Micali S, Vlachos G, Zeldovich N (2017) Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China. ACM pp 51–68
10. Micali S, Rabin MO, Vadhan SP (1999) Verifiable random functions. In 40th Annual Symposium on Foundations of Computer Science, FOCS '99. New York, NY, USA, IEEE Computer Society pp 120–130
11. David B, Gazi P, Kiayias A, Russell A (2018) Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel. Proceedings, Part II vol. 10821. Springer pp 66–98
12. Hanke T, Movahedi M, Williams D (2018) DFINITY technology overview series, consensus system. CoRR abs/1805.04548
13. Goldreich O, Levin LA (1989) A hard-core predicate for all one-way functions. In Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA, D. S. Johnson, Ed., ACM pp 25–32
14. Dodis Y, Yampolskiy A (2005) A verifiable random function with short proofs and keys. In Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland. Proceedings. Springer 3386:416–431
15. Hohenberger S, Waters B (2010) Constructing verifiable random functions with large input spaces. In Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera Proceedings. Springer 6110:656–672
16. Hofheinz D, Jager T (2016) Verifiable random functions from standard assumptions. In Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel. Proceedings, Part I. Springer 9562:336–362
17. Kohl L (2019) Hunting and gathering - verifiable random functions from standard assumptions with short proofs. In Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China. Proceedings, Part II. Springer 11443:408–437
18. Bitansky N (2017) Verifiable random functions from non-interactive witness-indistinguishable proofs. In Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA. Proceedings, Part II. Springer 10678:567–594
19. Brakerski Z, Goldwasser S, Rothblum GN, Vaikuntanathan V (2009) Weak verifiable random functions. In Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA. Proceedings. Springer 5444:558–576
20. Fuchsbauer G (2014) Constrained verifiable random functions. In Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy. Proceedings. Springer 8642:95–114
21. Wang Q, Feng R, Zhu Y (2018) Verifiable random functions with boolean function constraints. Sci China Inf Sci 61(3):039105:1–039105:3
22. Liang B, Banegas G, Mitrokotsa A (2020) Statically aggregate verifiable random functions and application to e-lottery. Cryptogr 4(4):37
23. Goyal R, Hohenberger S, Koppula V, Waters B (2017) A generic approach to constructing and proving verifiable random functions. In Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA. Proceedings, Part II. Springer 10678:537–566
24. Brunetta C, Liang B, Mitrokotsa A (2018) Lattice-based simulatable vrfs: Challenges and future directions. J Internet Serv Inf Secur 8(4):57–69
25. Abraham E (2018) Post-quantum verifiable random functions from ring signatures. IACR Cryptol ePrint Arch 2018:1231
26. Jager T, Niehues D (2019) On the real-world instantiability of admissible hash functions and efficient verifiable random functions. In Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Waterloo, ON, Canada. Revised Selected Papers. Springer 11959:303–332
27. Jager T (2015) Verifiable random functions from weaker assumptions. In Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland. Proceedings Part II. Springer 9015:121–143
28. Yamada S (2017) Asymptotically compact adaptively secure lattice ibes and verifiable random functions via generalized partitioning techniques. In Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA. Proceedings, Part III. Springer 10403:161–193
29. Boneh D, Lynn B, Shacham H (2001) Short signatures from the weil pairing. In Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia. Proceedings. Springer 2248:514–532
30. Chen T, Huang W, Kuo P, Chung H, Chao T (2018) DEXON: A highly scalable, decentralized dag-based consensus algorithm. IACR Cryptol ePrint Arc 2018:1112
31. Brotsis S, Kolokotronis N, Limniotis K, Shiaeles S (2020) On the security of permissioned blockchain solutions for iot applications. In 2020 6th IEEE Conference on Network Softwarization (NetSoft). IEEE pp 465–472
32. Barak B, Dodis Y, Krawczyk H, Pereira O, Pietrzak K, Standaert F, Yu Y (2011) Leftover hash lemma, revisited. In Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA. Proceedings. Springer 6841:1–20
33. Goldberg S, Vcelak J, Papadopoulos D, Reyzin L (2018) Verifiable random functions (vrfs)
34. Dobraunig C, Eichlseder M, Mendel F (2015) Analysis of SHA-512/224 and SHA-512/256. In Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand. Proceedings, Part II of Lecture Notes in Computer Science. Springer 9453:612–630
35. Goldberg S, Naor M, Papadopoulos D, Reyzin L (2016) Nsec5 from elliptic curves: Provably preventing dnssec zone enumeration with shorter responses. IACR Cryptol. ePrint Arch. 2016:83
36. Bernstein DJ (2006) Curve25519: new diffie-hellman speed records. In International Workshop on Public Key Cryptography. Springer pp 207–228

Guanglai Guo is currently a Ph.D. candidate in the school of Computer and Communication Engineering, University of Science and Technology (USTB), Beijing, China. He received the M.S. degree in control theory from USTB, China, in 2017. His research interests include cryptography and secure multi-party computation.

Yan Zhu is currently a professor in the school of computer and communication engineering at the University of Science and Technology Beijing (USTB), China. He was an associate professor at Peking University (PKU) in China from 2007 to 2012. He was a visiting associate professor in the Arizona State University (ASU) from 2008 to 2009, and a visiting research investigator of the University of Michigan-Dearborn in 2012. His research interests include cryptography, secure group computation, secure multi-party computation, and network security. He is a member of the IEEE.

E Chen received the B.S. degree from the department of School of Mathematics and Physics, University of Science and Technology Beijing, China. She is currently a Ph.D. candidate with the department of School of Computer and Communication Engineering, University of Science and Technology Beijing, China. Her research interests include attribute based system and lattice cryptography.

Guizhen Zhu graduated from Institute for Advanced Study, Tsinghua University with a Ph.D. degree in 2013. She was a visit scholar at department of Mathematics, University for California, Irvine from 2010 to 2011. She is currently a senior engineer at Data and Communication Science Technology Research Institute. Her main research area includes post-quantum cryptography, especially lattice-based cryptography, homomorphic encryption and its applications.

Di Ma received the Ph.D. degree in computer science from the University of California, Irvine, CA, USA, in 2009. She is currently an Associate Professor of Computer Science with the Computer and Information Science Department, College of Engineering and Computer Science (CECS), University of Michigan-Dearborn, Dearborn, MI, USA. She is also the Interim Associate Dean for Graduate Education and Research and the Director of the Cybersecurity Center for Education, Research, and Outreach, CECS. Her research is supported by the National Science Foundation, the National Highway Traffic Safety Administration, the Air Force Office of Scientific Research, Intel, Ford, and Research in Motion. She was with IBM Almaden Research Center in 2008 and the Institute for Infocomm Research, Singapore, from 2000 to 2005. She is broadly interested in the general area of security, privacy, and applied cryptography. Her research interests span a wide range of topics, including connected and autonomous vehicle security, smartphone and mobile device security, radio frequency identification and sensor security, and data privacy. Dr. Ma was the recipient of the Tan Kah Kee Young Inventor Award in 2004, the Distinguished Research Award from the College of Engineering and Computer Science of the University of Michigan-Dearborn in 2017, and the 2018 Trevor O. Jones Outstanding Paper Award from SAE International.

William Cheng-Chung Chu received the M.S. and Ph.D. degrees in computer science from Northwestern University, Evanston, IL, USA, in 1987 and 1989, respectively. He is currently a Distinguished Professor with the Department of Computer Science, Tunghai University, Taichung, Taiwan, where he had served as the Director of Software Engineering and Technologies Center from 2004 to 2016 and as the Dean of Research and Development office from 2004 to 2007. From 1994 to 1998, he was the Dean of Engineering College and an Associate Professor with the Department of Information Engineering and Computer Science, Feng Chia University. He was a Research Scientist with the Software Technology Center, Lockheed Missiles and Space Company, Inc. In 1992, he was also a Visiting Scholar with Stanford University. He has edited several books and authored or coauthored more than 100 referred papers and book chapters, as well as participated in many international activities, including organizing international conferences, serving as the steering committee for the IEEE Computer Society Signature Conference on Computers, Software and Applications, the Asia-Pacific Software Engineering Conference, the IEEE International Conference on Software Quality, Reliability and Security, the International Symposium on System and Software Reliability, and the program committee of more than 70 international conferences. His current research interests include software engineering artificial intelligence and big data analytics. Prof. Chu was the recipient of special contribution awards in both 1992 and 1993 and a PIP Award in 1993 at Lockheed Missiles and Space Company, Inc. He is an Associate Editor for the IEEE TRANSACTIONS ON RELIABILITY, the Journal of Software Maintenance and Evolution, the International Journal of Advancements in Computing Technology, and the Journal of Systems and Software.