ZKBdf: A ZKBoo-based Quantum-Secure Verifiable Delay Function with Prover-secret

Teik Guan Tan[1], Vishal Sharma[2], Zeng Peng Li[3], Pawel Szalachowski[1], and Jianying Zhou[1]

[1] Singapore University of Technology and Design
tanteikg@gmail.com, pjszal@gmail.com, jianying_zhou@sutd.edu.sg
[2] Queen's University Belfast
v.sharma@qub.ac.uk
[3] Shandong University
zengpengliz@gmail.com

Abstract. Since the formalization of Verifiable Delay Functions (VDF) by Boneh et al. in 2018, VDFs have been adopted for use in blockchain consensus protocols and random beacon implementations. However, the impending threat to VDF-based applications comes in the form of Shor's algorithm running on quantum computers in the future which can break the discrete logarithm and integer factorization problems that existing VDFs are based on. Clearly, there is a need for quantum-secure VDFs. In this paper, we propose ZKBdf, which makes use of ZKBoo, a zero-knowledge proof system for verifiable computation, as the basis for realizing a quantum-secure VDF. We describe the algorithm, provide the security proofs, implement the scheme and measure the execution and size requirements. In addition, as ZKBdf extends the standard VDF with an extra "Prover-secret" feature, new VDF use-cases are also explored.

Keywords: Verifiable Delay Function · Zero-Knowledge Proof · Post-Quantum Cryptography

1 Introduction

The notion of delay may initially come across as a paradox. In the world of computing where systems are constantly tuned for higher processing throughput and more efficient communications while users demand shorter response time and immediate gratification, the need for delays seems counter-intuitive.

Yet, there are valid use-cases where delays are relevant. In 1996, Rivest et. al. [47] introduced the concept of time-lock puzzles where a published secret is locked, and can only be opened after a specified period of time. Time-lock puzzles require the use of a computer to execute a sequence of commands, thus consuming a certain amount of time, before the secret is revealed. They can be applied in the case of auction bids where a bidder submits a sealed auction bid which requires a duration longer than the auction window for it to be opened. The time-lock puzzle here prevents even the auctioneer from knowing

the bid until after the close of the auction. Delays can also be used to regulate the number of requests received. Later constructions of delay functions include proof-of-sequential work (PoSW) by Mahmoody et. al. [37] using sequential hash functions and Cohen et. al. [15] which uses a hash-graph structure to optimize the verification efficiency of a hash-chain. On a different use-case, Dwork and Noar [20] proposed imposing a computational overhead for every email received in order to reduce the amount of SPAM mail in 1993. This concept was formalized as proof-of-work (PoW) by Jacobsson and Juels [30] and made famous when Nakamoto's Bitcoin [42] used PoW as the blockchain consensus mechanism for miners to propose blocks approximately every 10 minutes. Delays are also used to ensure fairness for leader elections. Snow White's sleepy consensus by Pass and Shi [44] relies on delay functions to account for all possible network latency when receiving inputs from active nodes to compute an unbiased leader, while not waiting in vain for nodes that are inactive.

Boneh et. al. [7] provided the formal definition of VDFs in 2018. VDFs differ from time-lock puzzles in their unique properties of not requiring a trapdoor operation while computing a deterministic and unpredictable output. Informally, we view the time-lock puzzle as a fast-encrypt-slow-decrypt confidentiality analogy and VDF as a slow-generate-fast-verify integrity analogy for delay functions. This begs the question why isn't VDFs used more widely beyond consensus protocols and time-stamp / random beacons. In our research, we discover that the inclusion of a prover-secret as part of the VDF protocol can open up more use-cases. These are discussed in Section 5.

In this paper, we provide the realization of a quantum-secure VDF using ZKBoo [25] to implement a HMAC-SHA256 circuit. While using verifiable computing primitives to construct a VDF is proposed before [7, 10, 51], we include a security reduction of our implementation to ZKBoo with the Fiat-Shamir heuristic [23, 54] to make our implementation secure under a quantum random oracle model (QROM) [9], and without trusted setup. Other contributions include proposing a modified existential unforgeability under chosen-message attacks (EUF-CMA) [27] experiment for VDFs and extending the ZKBoo SHA256 implementation to HMAC-SHA256 [32]. The ZKBoo HMAC-SHA256 zero-knowledge proof of knowledge of the MACing key gives an additional "Prover-secret" feature to ZKBdf. New VDF use-cases which make use of ZKBdf's Prover-secret feature are also identified.

The organization of the paper is as follows. Section 2 introduces the background and related work. Section 3 provides the design of a quantum-secure VDF and security proofs. Section 4 describes the implementation of the proposed VDF with execution results. Section 5 discusses new VDF use-cases and section 6 concludes the paper.

2 Background

2.1 Hash-chain

The hash-chain consists of a sequence of One-Way Functions (OWF) H where the output of the previous OWF operation is treated as the input to the next OWF operation. For a delay parameter T, the hash-chain $Eval$ function is:

$$H^T(x) = \begin{cases} H(H^{T-1}(x)) & T > 1 \\ H(x) & T = 1 \end{cases} \tag{1}$$

Execution of the hash-chain requires the Prover to invoke H sequentially for T times. Assuming that the OWF function H is collision-resistant, a polynomial-bounded adversary will not be able to perform any meaningful pre-computation or shorten the process since the probability of guessing $H^{i+1}(x)$ when only $H^i(x)$ $\forall i < T$ is computed is negligible.

The verification of $H^T(x)$ is a matter of a time-space tradeoff. If only $[x, H^T(x)]$ is provided as the proof, then the Verifier will have to perform the same sequence of hashes as the Prover to verify $H^T(x)$. If the intermediate hash values $H^i(x)$ are also provided as part of the proof, then the Verifier can parallelize the verification process across multiple processing units, and shorten the verification time. Since execution complexity of the verification remains at $O(T)$ and can only be sped up through process parallelism, delay functions such as *Sloth* [36] which use hash-chains are not VDFs but termed as a *pseudo*−VDF [7].

2.2 Verifiable Delay Function

Definition 1. *We define* $VDF = (Setup, Eval, Verify)$ *as triple of algorithms with the following syntax*

- $Setup(\lambda) \rightarrow (e_k, v_k)$ takes in a security parameter λ, and outputs the evaluation key e_k, and verification key v_k.
- $Eval(e_k, Cha, T) \rightarrow (Res, \pi)$ takes in the evaluation key e_k, a one-time challenge Cha along with the time delay T and outputs the response to the challenge Res and the corresponding proof π. $Eval$ must require at least T units of time to execute even on a parallel computer.
- $Verify(v_k, Res, \pi, Cha, T) \rightarrow (result \in \{accept, reject\})$ takes in the verification key v_k, the response Res, proof pi, challenge Cha, and time delay T and outputs *accept* if and only if Res is the correct response to Cha and π is evaluated correctly. $Verify$ must be able to run at significantly less than T units of time.

The $Correctness$ property of a VDF is defined such that the $Verify$ function will accept the output generated by the $Eval$ function while the $Soundness$ property of a VDF states that the probability of $accept \xleftarrow{R} Verify(v_k, Cha, Res', \pi', T)$ where $Res' \neq Res$ and $(Res, \pi) \xleftarrow{R} Eval(e_k, Cha, T)$ is negligible. In addition, Boneh et. al. [7] have listed three properties that embodies what a VDF has to exhibit:

- *Sequentiality.* Sequentiality is defined strictly with respect to the computation time of Res given (e_k, Cha) which cannot be less than T. The time required to generate proof π is excluded from the definition. However, we argue that the time needed to compute the proof π can also be taken into account. Hence, we re-define sequentiality as $Verify(v_k, Cha, Res, \pi, T) \rightarrow reject$ if time taken to compute (Res, π) from (e_k, Cha, T) is less than T when provided with polynomial-bounded computational power.
- *Efficient verifiability.* The $Verify$ function is expected to execute more efficiently as compared to $Eval$. It is required to complete in the order of $O(polylog(T))$ execution time.
- *Uniqueness.* For every input challenge Cha provided to $Eval$, there is a deterministic and unpredictable response Res. And it is computationally hard to find Cha when given only Res. This property makes VDFs applicable for use in random beacons and leader elections for distributed systems.

These three properties rule out time-lock puzzles, PoW, and PoSW as VDFs since time-lock puzzles do not follow the VDF protocol, and the proofs generated by PoW and PoSW are not unique [30, 37, 15].

Constructing a VDF typically involves iterating a non-"parallelizable" (or serialized) process as many times as necessary to account for the T delay, while having an efficient mechanism to verify the process. In the VDF survey [8], the two VDFs by Pietrzak [45] and Wesolowski [55] respectively use a common serialized process:

$$f(x) = H(x)^{2^T} \text{ over an abelian group } \mathbb{G}, \tag{2}$$

where $H(x)$ is repeatedly squared T times. This has been recognized as insecure [8] against a quantum-capable adversary who can compute the order of $\mathbb{G}$ using Shor's algorithm [49] in polynomial time.

To construct a quantum-secure VDF, a logical hypothesis would therefore be: *Can a quantum computer running a randomness service function as a VDF?* Mahmoody et. al [38] has shown this hypothesis to be flawed by proving the infeasibility of a black-box random oracle functioning as a VDF. De Foe et. al. [16] constructs a VDF using elliptic curve isogenies which is "quantum-annoying" but not quantum-secure. Chavez-Saab et. al. [13] proposes the use of isogenies embedded as an arithmetic structure within a succinct non-interactive argument (SNARG) [24] to be quantum-secure, but recent discoveries on the weakness of supersingular isogenies[11] casts doubts on the claims. The VDF implementation that comes closest to being quantum-secure is Buterin's VDF [10], but (to our best knowledge) lacks formal quantum-security proofs. It makes use of an iterated block cipher with Minimal Multiplicative Complexity (MiMC) [1] that is run sequentially, and then builds a SNARG to prove that the computation was performed correctly.

2.3 ZKBoo

ZKBoo [25] is a zero-knowledge proof system for verifiable computation. When ZKBoo is used by a Prover to prove the hash result of computing an OWF,

the Verifier has the assurance that the Prover knows the pre-image of the hash, without gaining any additional knowledge of the pre-image.

To generate the proof, ZKBoo uses Ishai et. al.'s [29] secure multiparty computation MPC-in-the-head to create an AND-NOT-XOR-ADD boolean circuit sequence of three branches. Unlike subsequent MPC-in-the-head proof constructions such as KKW [31], preprocessing and other process optimizations for ZKBoo is limited which lends itself well as a VDF. At the beginning of the proof generation during the preprocessing stage, the secret is randomly split into three shares and a deterministic random sequence $R_i[]$ is generated. During the circuit computation stage which makes up the bulk of the computation, the three shares then step through the circuit along the three branches respectively. At each of the AND and ADD gates, the three branches are pair-wise intertwined $(1 \leftrightarrow 2, 2 \leftrightarrow 3, 3 \leftrightarrow 1)$ in a sequential manner and XOR with the deterministic sequence $R_i[]$ to ensure no steps are skipped. The final result of the computation is committed in the proof but only two of the three views will be revealed to the Verifier for verification. The choice of which two views is determined at the proof extraction stage using the Fiat-Shamir heuristic [23] to make the proof non-interactive. To increase the cryptographic strength (or assurance) of the zero-knowledge proof, the same boolean circuit is re-run using different shares which are randomly generated. Since each stage prepares the necessary information for the next stage, each stage must wait for the previous stage to complete before commencing. During verification, the Verifier takes in the generated proof, and walks through the two of three views and has $\geq 50\%$ assurance, but no additional knowledge, that the Prover knows the secret value used to create the three shares. Both proof and verify processes execute in $O(N)$ complexity and all runs are parallelizable.

The strengths of ZKBoo are that it does not require any trusted setup, has efficient circuits for small computations and each round in the proof generation and verification are parallelizable. ZKBoo, however, has a large proof size. Chase et. al. [12] improves on ZKBoo by proposing ZKB++ which, in addition to reducing the proof size by almost half, introduces the use of the Unruh transformation [52] to arrive at security proofs in QROM.

2.4 Computationally Sound Probabilistic Checkable Proof (PCP)

PCP by Arora and Safra [2] is a system of proof that can efficiently reduce the verification time complexity of proofs by an order of $O(logN)$. It achieves this by first encoding the original proof into a local-testable format [4], then applying an oracle to select proof samples to be verified. The outcome is that the PCP verification process will yield errors for incorrect proofs with a high probability on low-degree testing, although proof generation time complexity and proof size remain asymmetrically larger. Separately, Kilian [34] introduces a zero-knowledge protocol where it is possible to use a subset of commitments to achieve concise arguments for a corresponding proof.

Micali's [41] computationally sound proof realizes a construction of Kilian's protocol and PCP using Merkle trees [40]. Computationally sound proofs build

on PCPs by reducing the size of proof needed to be sent to the Verifier, thus optimizing both space and time complexity in the verification process. Briefly, Micali's construction works as follows during proof generation to extract k proof elements out of the complete set of proofs for verification:

1. Computes the full set of PCP proofs π.
2. Build a Merkle-tree with each of the proof elements in π placed as a leaf node using an oracle.
3. With the root node of the Merkle tree as the seed, use a sampling oracle to select k leaf nodes.
4. Package these k proof elements along with the branches of the Merkle tree that traverse up to the root node as the PCP proof to be transmitted to the Verifier.

During verification, the following happens:

1. Verify that the branches of the Merkle tree lead up to the root node.
2. With the root node as the seed, use the same sampling oracle to select the expected k leaf nodes.
3. Verify the proof of each of the leaf nodes. An honest Prover is expected to have minimally transmitted these k proof elements.

3 A Quantum-Secure VDF

Definition 2. *We define ZKBdf = (Setup,Eval,Verify) is a triple of algorithms as follows:*

$ZKBdf.Setup() \rightarrow (e_k, H(e_k))$ is run by the Prover. It generates a Prover-secret $e_k \in \mathbb{Z}^+$ and public commitment $H(e_k)$ which is published.

$ZKBdf.Eval(e_k, Cha, T) \rightarrow (Res, \pi)_T$ is run by the Prover. It takes in the previously generated Prover-secret e_k, the random challenge Cha from the Verifier, and the required delay parameter T. It returns $Res = HMAC(Cha, e_k)$ and zero-knowledge computational proof $\pi = \{v_k = H(e_k)\}$.

$ZKBdf.Verify(H(e_k), Res, \pi, Cha, T) \rightarrow (result \in \{1, 0\})$ is run by the Verifier. It takes in the previously-generated challenge Cha, the response Res and proof π from the Prover, the required delay parameter T and the previously-published commitment $H(e_k)$. It returns 1 if and only if $Res = HMAC(Cha, e_k)$ and π is computationally verified correct.

The design intuition behind our proposed VDF, named ZKBdf (ZKBoo delay function) is to substitute the "hash" in the hash-chain with a serialized ZKBoo zero-knowledge proof of the pre-image of the hash, include an additional serialized ZKBoo circuit for HMAC to generate the unique return value, and then use PCP with Micali's Merkle Tree construction to reduce the size and time complexity of the verification. Such a design preserves the non-algebraic nature of the construction and adds a feature of "prover-secret" into the VDF.

3.1 Serializing ZKBoo

Since both proof generation and verification of ZKBoo are parallelizable, we need to modify ZKBoo such that proof generation is serialized while proof verification remains parallelizable. The trick we use is to change the way the deterministic sequence of random $R_i[]$ are generated. Instead of using the same seed to generate all the sequences, we use the Fiat-Shamir heuristic [23] to require the generation of $R_{i+1}[]$ be dependent on views w_{i-1} from the previous ZKBoo iteration:

$$R_i[] = \begin{cases} PRF(seed) & i = 1 \\ PRF(H(w_{i-1})) & otherwise \end{cases} \tag{3}$$

We therefore modify *ZKBoo Prove* to a serialized form in Algorithm 1 with changes marked in red. The main changes happen in lines 9 to 11 where instead of using a fixed seed to generate the sequences $R[]$ in all T iterations, we use the Fiat-Shamir heuristic [23] to require the generation of $R_{i+1}[]$ be dependent on the view w_{i-1} from the previous ZKBoo iteration.

To accommodate this change, the generation of $R_i[]$ is moved out of the preprocessing stage to between lines 9 and 10 of the circuit computation stage with the logic updated to reflect Equation (3). Also, line 19 is changed from `Compute` $h \leftarrow Hash(c)$ to `Compute` $h \leftarrow Hash(c_i)$ so that both the circuit computation and proof extraction stages become serialized. Verification remains parallelizable since the values needed to compute $R_i[]$ are already known.

3.2 ZKBdf Construction

ZKBdf is described in Figure 1.

There are two parties in the protocol, namely the Prover, who needs to prove that the time delay T has taken place, and the Verifier, who wants the Prover to delay for T duration. The protocol flows can be separated into three stages: *i) Setup:* which needs to be done once; *ii) Challenge-Response:* where the Verifier will issue an unpredictable challenge and the Prover has to respond with the proof that T time has passed; and *iii) Verify:* where the Verifier will verify the Prover's proof.

Structurally, the Prover computes what looks like a two-dimensional ZKBoo-chain with a Merkle tree on top, before extracting the PCP [2, 41] proofs to be sent to the Verifier. Functionally, ZKBdf is superior to hash-chains and its derivatives due to the added "Prover-secret" feature from the HMAC proof. A Prover using ZKBdf can proof knowledge of the Prover-secret e_k to the public Verifier without revealing e_k. On the other hand, ZKBdf's proof size and bit-strength increase with the delay parameter T since ZKBoo's iterative proof generation is the primary driver of the delay. This is in contrast with Buterin's VDF [10] whose MiMC delay circuit is separate from the argument proof.

The VDF properties achieved by ZKBdf include all the properties mentioned in Section 2.2 plus being quantum-secure. We briefly describe how each property is designed into ZKBdf before carrying out the formal proofs:

Algorithm 1: Modified *ZKBoo.Prove* algorithm to be serialized.

```
1  begin
2      r ← seed; e_k ← secret;
       // Preprocessing Stage
3      for i from 1 to T do
4          {s_i^1, s_i^2} ← random;
5          s_i^3 = e_k ⊕ s_i^1 ⊕ s_i^2;
6      end
       // can't be run in parallel
7      for i from 1 to T do
           // Circuit Computation Stage
8          if i == 1 then
9              R_i[] ← PRF(r)
10         else
11             R_i[] ← PRF(H(w_{i-1}))
12         end
13         while circuit is not complete do
14             if AND or ADD gate then
15                 {w_i^1, w_i^2, w_i^3} ← gate operation using {s_i^1, s_i^2, s_i^3}, R_i[]
16             else
                   // NOT or XOR gate
17                 {w_i^1, w_i^2, w_i^3} ← gate operation using {s_i^1, s_i^2, s_i^3}
18             end
19         end
20         c_i ←result from {w_i^1, w_i^2, w_i^3}; // c_i is the commitment for round i
           // Proof Extraction Stage
21         Compute h = Hash(c_i);
22         b ← next 2 bits from h;
23         switch b do
               // w_i is the 2-of-3 view for round i
24             case 00 do
25                 w_i ← w_i^1, w_i^2
26             end
27             case 01 do
28                 w_i ← w_i^2, w_i^3
29             end
30             case 10 do
31                 w_i ← w_i^3, w_i^1
32             end
33         end
34         otherwise do
35             retry b with next 2 bits from h
36         end
37     end
38     w ← {w_1, ..., w_T};
39     c ← {c_1, ..., c_T};
40     return z = {c, w}
41 end
```

- *Setup.*
  1. Both Prover and Verifier agree on a OWF $H()$ and delay T.
  2. Prover calls $ZKBdf.Setup$ to generate Prover-secret $e_k \in \mathbb{Z}^+$ and $commit = H(e_k)$ as the commitment.
- *Challenge-Response*
  1. Verifier generates random $Cha \in \mathbb{Z}^+$ and sends to Prover.
  2. Prover calls $ZKBdf.Eval$ to perform the following:
     (a) Compute $Res = HMAC(Cha, e_k) = H(e_k \oplus o_{pad}||H(e_k \oplus i_{pad}||Cha))$ [32, 6] and perform T iterations of the serialized ZKBoo proof (Algorithm 1) to obtain $z = \{z_1, z_2, ..., z_T\}$ where $z_i = \{c_i = \{v_k = H(e_k), Res = HMAC(Cha, e_k)\}, w_i\}$
     (b) Build a Merkle tree [40] using all the elements of z as the leaf nodes.
     (c) Use Micali's construction [41] to select the index of leaf nodes $\{j_1, j_2, ..., j_{polylog(T)}\}$ to form the set of PCPs [2] $\pi = \{\pi_1, \pi_2, ...\pi_{polylog(T)}\}$
     (d) Each π_i consists of a branch of the Merkle tree from the root node leading to and including the leaf nodes z_{j_i-1} and z_{j_i}.
  3. Prover sends Res and π to the Verifier.
- *Verify*
  1. Verifier calls $ZKBdf.Verify$ which forks parallel processes for each $\pi_i \in \pi$ and checks that:
     (a) The Cha used by the Prover is correct.
     (b) Root node is the same for all π_i.
     (c) π_i is correctly selected based on Micali's construction. [41].
     (d) Merkle tree branch leading to z_{j-1} and z_j is verified correctly.
     (e) Verify $v_k == commit$.
     (f) Extract w_{j-1} from z_{j-1} and compute $R_j[] = PRF(H(w_{j-1}))$.
     (g) Verify ZKBoo proofs z_{j-1} and z_j for $\{v_k = H(e_k)\}$ and $\{Res = HMAC(Cha, e_k)\}$.
  2. Verifier accepts Res is pseudo-random and computed from $HMAC(Cha, e_k)$ without knowing the value of e_k.

Fig. 1. ZKBdf - ZKBoo delay function

- **Execution asymmetry.** We introduce execution asymmetry between the the ZKBoo proof circuit and verify circuit in order to meet the VDF *Sequentiality* and *Efficient verifiability* properties.
  - *Sequentiality.* In Section 2.2, our re-definition of *Sequentiality* includes the time taken to compute the proof. Since ZKBoo [25] is not based on any algebraic primitives, we have construct a serialized version of the ZKBoo proof generation in Challenge-Response step 2a of Figure 1 by using the Fiat-Shamir heuristic [23] to make the random sequence $R_i[]$ for each subsequent round's proof circuit be dependent on the previous round (see Equation (3)). This dependency adds an additional verification step to the ZKBoo verify operation but does not affect the verification process which remains parallelizable for each round.
  - *Efficient verifiability.* We make use of computationally sound PCP [2, 41] to reduce the computational complexity of $ZKBdf.Verify$ to O(polylog(N)), thus achieving the *efficiently verifiable* requirement. In addition, the Verifier can utilize a polynomial number of (up to $\log^k T$) parallel-processing resources to obtain execution speed-up of $ZKBdf.Verify$ as compared to $ZKBdf.Eval$ which can only run sequentially.
- **Uniqueness.** To ensure the VDF response by the Prover is deterministic and unpredictable, the $ZKBdf.Eval$ function will incorporate a Pseudo-random function (PRF) [26] to compute Res.
- **Quantum-secure.** We have to prove our modified version of ZKBoo+PCP proofs is quantum-secure. Informally, ZKBoo is an MPC-based zero-knowledge proof system that is proven secure in the random oracle model [29]. Next, the OWF boolean circuit used in our proposed VDF is a collapsing hash function and hence is quantum-resistant [53]. Finally, we retain the use of the size-efficient Fiat-Shamir heuristic [23] and rely on [54, 18, 14] to arrive at quantum-security proofs.

The proposed ZKBdf also incorporates some of the properties related to continuous VDF [22]. Proof-generation is not an all-or-nothing process. A Prover can hand over a partial proof to another Prover to complete in a sequential manner, provided the Prover-secret e_k is shared. Similarly, a Verifier can verify part of the proof, or concurrently send parts of the proof verification to other Verifiers.

Assumption. There is an underlying assumption in our design intuition that there exists a common OWF that i) can be mapped into a ZKBoo boolean circuit; ii) can function as an OWF in the Fiat-Shamir heuristic; and iii) can be used as part of the PRF for generating the VDF response. In our construction, we have chosen this function to be SHA-family [21] of algorithms. In Section 4, our proposed VDF will be using SHA-256 as it has well-studied implementations with a realized ZKBoo boolean circuit [25], is already widely used as a OWF [42], is a collapsing hash function [53], and is standardized for use as a PRF [32].

The rest of this section covers the formal proofs for ZKBdf's completeness and soundness, execution asymmetry, uniqueness, and quantum security.

3.3 Completeness and Soundness

Claim 1 *ZKBdf (Definition 2) satisfies the VDF conditions of completeness and soundness.*

Proof. Based on Definition 2, we arrive at:

$$Pr\left[\begin{array}{l} ZKBdf.Verify(H(e_k), \\ \qquad res, \pi, Cha, T) == 1 \end{array} \middle| \begin{array}{l} (e_k, H(e_k)) \xleftarrow{R} ZKBdf.Setup() \\ (res, \pi) \xleftarrow{R} ZKBdf.Eval(e_k, Cha, T) \end{array}\right] = 1 \quad (4)$$

Equation (4) satisfies the VDF correctness definition 2 in Boneh et. al. [7].

In ZKBdf, the time delay parameter T defines the number of rounds ZKBoo is run. For every increment of T, ZKBdf linearly increases the time delay and exponentially increases (doubling) the security bit-strength of the proof. The existence of an algorithm O(poly(T)) $\mathcal{A}$ is

$$Pr\left[\begin{array}{r} ZKBdf.Verify(H(e_k), Res', \pi', Cha, T) \\ == 1 \\ (res', \bullet) \neq ZKBdf.Eval(e_k, Cha, T) \end{array} \middle| \begin{array}{l} (e_k, H(e_k)) \xleftarrow{R} ZKBdf.Setup() \\ (res', \pi') \xleftarrow{R} \mathcal{A}(e_k, Cha, T) \end{array}\right] \quad (5)$$
$$= \frac{1}{2^T} \leq negli(T)$$

Equation (5) satisfies the VDF soundness definition 3 in Boneh et. al. [7]. □

3.4 Execution Asymmetry

Lemma 2 (Sequentiality) *If there exists an algorithm $\mathcal{A}$ where an adversary with polynomial-bounded computing resources can take $< T$ time-units to compute $(Res, \pi) \xleftarrow{R} \mathcal{A}(e_k, Cha, T)$, then*

$$Pr\left[ZKBdf.Verify(H(e_k), Res, \pi, Cha, T) == 1\right] < negli(T). \quad (6)$$

Proof. We have established that the modified $ZKBoo.Prove$ algorithm (see Algorithm 1) has limited preprocessing steps and is non-parallelizable. Next, we take the assumption that the fastest time needed to complete 1 cycle of the ZKBoo HMAC-SHA-256 boolean circuit, z_i, is 1 time-unit.

• If $T == 1$, then by Claim 1 ($Soundness$) of ZKBdf, the fastest time possible to compute $(Res, \pi) \xleftarrow{R} ZKBdf.Eval(e_k, Cha, T) \geq$ time taken to compute $z_i = 1$ time-unit.

• If $T > 1$, then the computation of z_T can only start after $R_{T-1}[]$ is available. Since it is computationally infeasible to find $R_{T-1}[]$ (Equation (3)) without z_{T-1} and the probability of guessing $R_{T-1}[]$ is negligible, time taken to compute $(Res, \pi) \xleftarrow{R} ZKBdf.Eval(e_k, Cha, T) \geq$ time taken for z_{T-1} to be generated + time taken to compute $z_T = (T-1) + 1 = T$.

Since time taken for $(Res, \pi) \xleftarrow{R} \mathcal{A}(e_k, Cha, T) < T$, then
$(Res, \bullet) \nleftarrow ZKBdf.Eval(e_k, Cha, T)$ and by Equation (5), Lemma 2 is true. □

Lemma 3 (Efficient verifiability) *A Verifier requires O(polylog(T)) to complete the execution of $ZKBdf.Verify$ such that ZKBdf remains computationally sound [41].*

Proof. Arora and Safra [2] has shown that to create sound PCPs, the probability that the verifier accepts each randomly-selected proof, given that the proof is incorrect, must be $< \frac{1}{2}$. Following from Claim 1 (*Soundness*) of ZKBdf, Micali's construction [41] is used in a Merkle tree (see Figure 2) to reduce the size and execution complexity of the original set of ZKBoo proofs $z = \{z_1, z_2, ..., z_T\}$ into $\pi = \{\pi_1, \pi_2, ..., \pi_{polylog(T)}\}$.

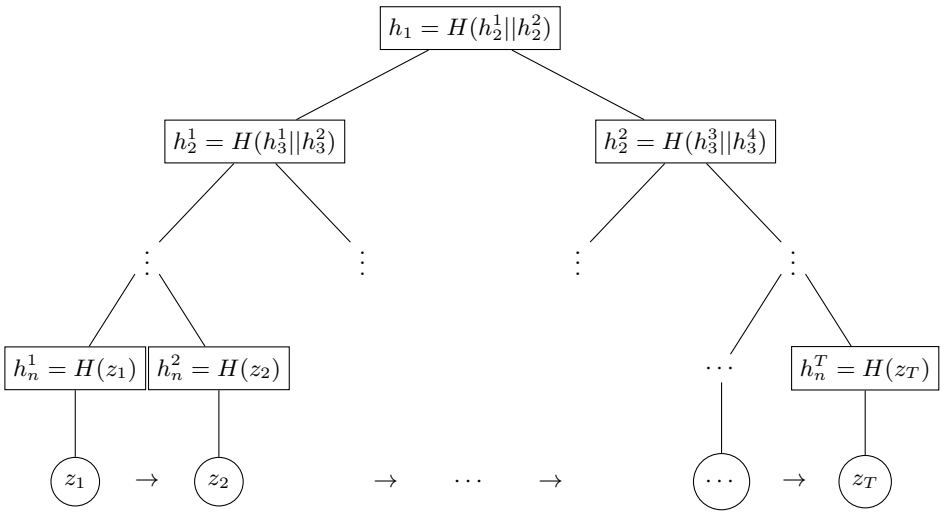

Fig. 2. Merkle tree of ZKBoo proofs to extract computational sound proofs

Each proof, $\pi_i = \{h_1, h_2, ..., h_n^{j_i-1}, h_n^{j_i}, z_{j_i-1}, z_{j_i}\}$ where j_i is the index randomly selected by the Fiat-Shamir heuristic, contains the branch of the Merkle tree from the root node leading to 2 leaf nodes z_{j_i-1} and z_{j_i}. Since both z_{j_i-1} and z_{j_i} are to be verified,

$$Pr\left[\pi_i \text{ is verified correct} | \pi_i \text{ is incorrect}\right] \leq \frac{2}{3} * \frac{2}{3} = \frac{4}{9} < \frac{1}{2} \tag{7}$$

Equation (7) shows that the probability of a false negative verification of π_i is $\leq \frac{4}{9}$ which is $< \frac{1}{2}$ required for creating PCPs [2]. Since $|\pi| = polylog(T)$, $ZKBdf.Verify$ runs at O(polylog(T)). □

3.5 Uniqueness

Lemma 4 (Uniqueness) *The Res returned from ZKBdf.Eval is deterministic and in-distinguishable from random, unless the proof π is provided.*

Proof. In Challenge-Response step 2a of Figure 1, we use the HMAC primitive [32] where $Res = HMAC(Cha, e_k)$. Since HMAC is a quantum-secure PRF [6, 5, 50] when using SHA-256 as the underlying hash function, Lemma 4 is proven. □

3.6 Quantum-secure

Definition 3. *We design a modified security experiment on the basis of EUF-CMA [27] for VDFs as a series of exchanges between a challenger and an adversary.*

Step i) The challenger performs *Setup* and sends the verification key v_k to the adversary.
Step ii) The adversary can choose any challenge Cha_i and time T_i ask the challenger to compute the response.
Step iii) The challenger performs $Eval(e_k, Cha_i, T_i)$ and returns the response (Res_i, π_i) to the adversary. Step ii) and iii) are repeated as many times as necessary.
Step iv) At the end of the experiment, the adversary is provided with e_k and T', and has less than T' time to output a challenge Cha' that is not within the set of challenges Cha_i requested in Step ii)., and the response (Res', π') that will return accept when $Verify(v_k, Res', \pi', Cha', T')$ is called.

Claim 5 (Quantum-Secure) *ZKBdf is EUF-CMA quantum-secure if the probability, that any polynomial-time quantum-capable adversary can win the modified EUF-CMA experiment, is negligible.*

In order for any adversary to win the experiment, the adversary will need at least one of the following 5 cases to happen with non-negligible probability:

Case a) Obtain the evaluation key e_k before Step iv); or
Case b) Obtain the challenge-response pair Cha', Res' before Step iv).; or
Case c) Start the generation of any ZKBoo-proofs z_i prior to having e_k; or
Case d) Generate the set of ZKBoo-proofs $z = \{z_1, ...z_T\}$ non-sequentially; or
Case e) Creating the proof $\pi = \{\pi_1, .., \pi_{polylog(T)}\}$ without needing to completely generate the set of ZKBoo-proofs z.

In ZKBdf, $v_k = H(e_k)$ where SHA-256 is used as the OWF $H()$. The collision-resistance of SHA-256 has been extensively studied [33, 39, 28] in both classical and quantum settings. Unruh [53] then showed that SHA-256 is a collapsing function which is "analogous to collision-resistance in the post-quantum setting". Hence, it is unlikely that Case a) happens.

Without knowledge of e_k, the adversary will have to guess the value of response $Res' = HMAC(Cha', \cdot)$ where $Cha' \notin \{Cha_1, ..., Cha_i\}$ previously done in Step ii). Since the response is proven unique in Lemma 4, it is unlikely that Case b) happens.

Each ZKBoo-proof z_i contains the zero-knowledge proof of $v_k = H(e_k)$ and $Res' = HMAC(Cha', e_k)$. Since the adversary does not have knowledge of e_k or Res' prior to Step iv), it is unlikely that Case c) happens.

Lemma 2 has shown ZKBdf to satisfy the sequentiality property in the classical setting. To prove sequentiality in the quantum setting, we have to show that the Fiat-Shamir heuristic [23] used to serialize (see Equation (3)) remains secure. This has been proven by Don et. al. [18] on the basis that the hash is collapsing (which SHA-256 is) and thus implies that Case d) is unlikely to happen.

Finally, Chiesa et. al. [14] has proven that Kilian's protocol [34] (which Micali's construction [41] is based on) is a collapsing protocol and can be used to securely construct a post-quantum argument of knowledge. This implies that Case e) is unlikely to happen and completes the proof of Claim 5.

4 Implementation

To realize the design, we implemented[4] $zkbdf.Eval$ and $zkbdf.Verify$ to observe the proof performance and proof size. As a benchmark, we also included a pseudo-VDF version of $zkbdf.Verify$, called $zkbdf.VerifyPseudo$[5], which performs the verification on the entire zkboo-proof, without the optimized PCP-proofs. This would allow us to understand the extent of execution and size optimization that computationally sound PCP provides. Table 1 recaps the properties of the modules.

Table 1. Characteristics of ZKBdf modules implemented

Properties	$zkbdf.Eval$	$zkbdf.VerifyPseudo$	$zkbdf.Verify$
Proof Execution	$O(T)$	$O(T)$	$O(polylog(T))$
CPU utilization	sequential	parallel	parallel
Proof Size	$O(T)$ or $O(polylog(T))$	$O(T)$ -	- $O(polylog(T))$

All executions were performed on an Intel I5-8250U 8th Gen machine with 8 CPU cores and 8GB RAM, running 64-bit Windows 10. No operating system level CPU scheduling or adjustments were done.

4.1 Execution

We want to observe how the value of delay T is translated into actual execution. Since the soundness property of ZKBdf is dependent on T, it is not meaningful

[4] Source codes at https://github.com/tanteikg/zkbdf

[5] We define $ZKBdf.VerifyPseudo(H(e_k), Res, z, Cha, T) \rightarrow (result \in \{1, 0\})$ as a function run by the Verifier. The difference with $zkbdf.Verify$ is that the input proof is the entire set of zkboo proofs z instead of the PCP proofs π. As a reference, $zkbdf.VerifyPseudo$ achieves the same completeness, soundness, sequentiality, uniqueness, and quantum-secure properties. It only does not achieve the efficient-verifiability property to make it a VDF.

if T is too small (i.e. < 50). We varied the time delay parameter T from 50 up to 350 in steps of 50 and captured the average execution times and proof sizes generated. Figure 3 plots the execution times of the 3 modules against the delay parameter.

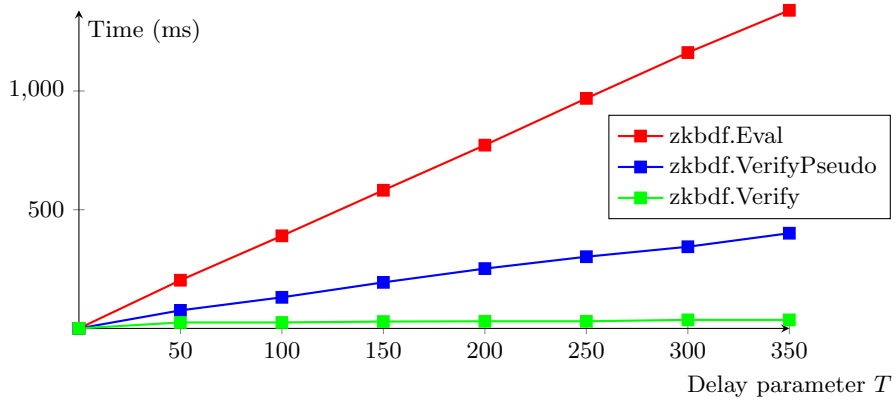

Fig. 3. Execution time against delay parameter T

The execution times of $zkbdf.Eval$ can be observed to linearly increase with T while the execution times of $zkbdf.Verify$ increases less significantly since it executes in $O(polylog(T))$. Due to the parallel-CPU utilization during execution, $zkbdf.VerifyPseudo$ executes faster than $zkbdf.Eval$, but also increases linearly.

When comparing the ratio of verification time versus evaluation time (see Figure 4), $zkbdf.Verify$ execution time drops below 10% of $zkbdf.Eval$ execution time for $T > 100$ and continues to decrease to almost 2.5% when $T = 350$. $zkbdf.VerifyPseudo$ execution time continues to hover at around 30% of $zkbdf.Eval$ execution time even for increasing T.

4.2 Proof Size

To understand the optimization in proof size due to Micali's construction [41], we list sizes of proofs against delay parameter T in Table 2.

Table 2. Size of proof (in bytes) against delay parameter T

Size	Complexity	T=50	T=100	T=150	T=200	T=250	T=300	T=350
z Proof	$O(T)$	1485200	2970400	4455600	5940800	7426000	8911200	10396400
π Proof	$O(polylog(T))$	360384	422224	484192	485792	487392	549488	551088

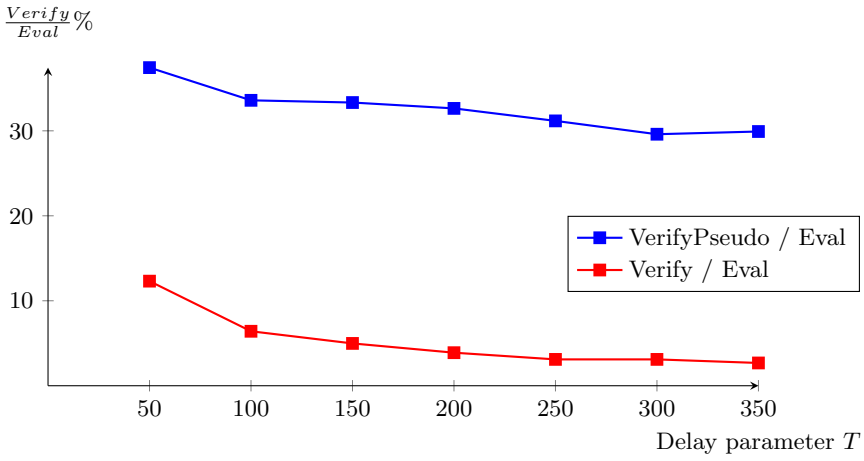

Fig. 4. Percentage of $\frac{Verify}{Eval}$ against delay parameter T

In comparison[6] with Pietrzak's [45] and Wesolowski's [55] VDFs, both their proof sizes grow at O(logT) complexity and remain below 200KB [56] even for large T values. While proof sizes of ZKBdf can be halved through ZKB++ optimizations [12], they are unlikely to approach the sizes below 200KB for large T values.

5 ZKBdf Application Areas

At present, we see the use of VDFs in consensus protocols for blockchains such as Ethereum (ethereum.org), Tezos (tezos.foundation) and Chia (chia.net) as well as in constructing time-stamping services and random beacons [22, 51, 35]. However, we believe that there is a wider use-case for VDFs if the functionality of a Prover-secret is included. In this section, we take an exploratory approach to identify other possible use-cases where applications can use ZKBdf to improve outcomes. These are described below.

5.1 Limiting Authentication Retries

We find a use-case where a delay function is needed during the authentication process to limit a brute-force attack against a backend authentication service. Broken Authentication is amongst the top 10 security risks highlighted by OWASP (Open Web Application Security Project) [43] where one of the ways to address such risks is to introduce an increasing delay for repeated failed authentication attempts. Such a setup, however, requires the backend authentication

[6] The quantum-secure VDF by Chavez-Saab et. al. [13] lacks published implementation details for comparison.

service to maintain failed authentication states for every user which inadvertently adds resource overheads and complexity especially in distributed systems. There are also many protocols such as Bitcoin [42], Transport Layer Security [46] and Wi-FI Protected Access (IEEE 802.11-2020) which do not require tracking of failed authentication attempts. A stateless delay mechanism using client-side puzzles is presented by Aura et. al [3] where every authentication is preceded with a PoW challenge which the authentication client needs to solve, before the server verifies the solution. Similar mechanisms are also used to prevent brute-force denial-of-service network attacks [17] and limiting peer-to-peer sybil attacks [19].

The advantage of using ZKBdf instead of a client-side PoW puzzle is that the number of authentication retries that a hacker can make is deterministic and no longer dependent on the amount of resources available to the hacker. Increasing the amount of CPU/memory resources at the hacker's end does not increase the number of authentication retries, and this will serve to deter hackers while not increasing the carbon footprint caused by ever-more complex puzzles.

5.2 Improving Auction Liveliness

In a classical English auction, an item is put on offer for participants to bid in an open outcry manner. The auctioneer asks for participants to place bids higher than the previous bid, and when an elapsed period has occurred without any participants placing any higher bids, the auction is closed with the winner being the participant who submitted the latest (and highest) bid. Online auctions that happen on the Internet, on the other hand, mostly do not have a concept of an elapsed time since the last bid. Instead, there is auction end-time whereby the highest bid received before the end-time is the winning bid. This has given rise to situations where participants are passive throughout most of the auction period and are only active at the closing moments of the auction where bid sniping [48] occurs.

The ZKBdf protocol could be used to allow honest participants to determine the end of the auction prior to the end-time. When a participant submits a valid highest bid, the participant is issued with a VDF challenge which effectively starts the elapsed time computation. If no higher bid is received prior to the participant completing the VDF challenge and submitting the response, then the participant would have won the auction, thus ending the auction. The advantage of using the VDF here is that we expect participants will no longer wait till the closing moments before bidding. Another possible positive outcome would be the detection of shill bidding (an agent working for a corrupt seller to bid up the price without any intention to buy) since the seller's agent is unlikely to submit the VDF response to close the auction.

6 Conclusion

In this paper, we introduced ZKBdf, a provably quantum-secure VDF built on a ZKBoo [25] zero-knowledge proof of knowledge of a HMAC-SHA256 [32] se-

cret key. In order to achieve the VDF property of sequentiality, the generation of each subsequent iterative ZKBoo proof is modified to include a Fiat-Shamir transform [23] of the previous view. The verification process of the set of ZKBoo proofs is then optimized using Micali's construction [41] to perform computationally sound PCP proof-verification [2] thereby achieving O(polylog(T)) in both execution and proof size. We have provided proofs for completeness, soundness, sequentiality, efficient verifiability, and uniqueness. We have also used a modified EUF-CMA experiment to prove ZKBdf to be quantum-secure. The implementation of ZKBdf is tested and both performances, as well as proof sizes, are measured. ZKBdf includes an added feature of "Prover-secret" on top of standard VDFs and new use-cases are identified. Future work includes optimizing the ZKBdf proof sizes and exploring the use of ZKBdf in post-quantum authentication.

References

1. Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 191–219. Springer (2016)
2. Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of NP. Journal of the ACM (JACM) **45**(1), 70–122 (1998)
3. Aura, T., Nikander, P., Leiwo, J.: DOS-resistant authentication with client puzzles. In: International workshop on security protocols. pp. 170–177. Springer (2000)
4. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the twenty-third annual ACM symposium on Theory of computing. pp. 21–32 (1991)
5. Bellare, M.: New proofs for NMAC and HMAC: Security without collision-resistance. In: Annual International Cryptology Conference. pp. 602–619. Springer (2006)
6. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Annual international cryptology conference. pp. 1–15. Springer (1996)
7. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Annual international cryptology conference. pp. 757–788. Springer (2018)
8. Boneh, D., Bünz, B., Fisch, B.: A Survey of Two Verifiable Delay Functions. IACR Cryptol. ePrint Arch. **2018**, 712 (2018)
9. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 41–69. Springer (2011)
10. Buterin, V.: STARKs, Part 3: Into the Weeds (2018), available Online: https://vitalik.ca/general/2018/07/21/starks_part_3.html, last accessed March 2022
11. Castryck, W., Decru, T.: An efficient key recovery attack on sidh (preliminary version). Cryptology ePrint Archive (2022)
12. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from

symmetric-key primitives. In: Proceedings of the 2017 acm sigsac conference on computer and communications security. pp. 1825–1842 (2017)
13. Chavez-Saab, J., Henríquez, F.R., Tibouchi, M.: Verifiable Isogeny Walks: Towards an Isogeny-based Postquantum VDF. Cryptology ePrint Archive (2021)
14. Chiesa, A., Ma, F., Spooner, N., Zhandry, M.: Post quantum succinct arguments. Electronic Colloquium on Computational Complexity, (38) (2021), https://eccc.weizmann.ac.il//eccc-reports/2021/TR21-038/index.html
15. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 451–467. Springer (2018)
16. De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 248–277. Springer (2019)
17. Dean, D., Stubblefield, A.: Using Client Puzzles to Protect TLS. In: USENIX Security Symposium. vol. 42 (2001)
18. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Security of the fiat-shamir transformation in the quantum random-oracle model. In: Annual International Cryptology Conference. pp. 356–383. Springer (2019)
19. Douceur, J.R.: The sybil attack. In: International workshop on peer-to-peer systems. pp. 251–260. Springer (2002)
20. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Annual International Cryptology Conference. pp. 139–147. Springer (1992)
21. Eastlake 3rd, D., Hansen, T.: US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF) (2011), available Online: https://tools.ietf.org/html/rfc6234, last accessed March 2022
22. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 125–154. Springer (2020)
23. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 186–194. Springer (1986)
24. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Proceedings of the forty-third annual ACM symposium on Theory of computing. pp. 99–108 (2011)
25. Giacomelli, I., Madsen, J., Orlandi, C.: Zkboo: Faster zero-knowledge for boolean circuits. In: 25th {usenix} security symposium ({usenix} security 16). pp. 1069–1083 (2016)
26. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM (JACM) **33**(4), 792–807 (1986)
27. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17**(2), 281–308 (1988)
28. Hosoyamada, A., Sasaki, Y.: Quantum Collision Attacks on Reduced SHA-256 and SHA-512. Cryptology ePrint Archive, Report 2021/292 (2021), https://eprint.iacr.org/2021/292
29. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. pp. 21–30 (2007)
30. Jakobsson, M., Juels, A.: Proofs of work and bread pudding protocols. In: Secure information networks, pp. 258–272. Springer (1999)

31. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 525–537 (2018)
32. Kelly, S., Frankel, S.: Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec (2007), available Online: https://www.ietf.org/rfc/rfc4868.txt, last accessed March 2022
33. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: attacks on Skein-512 and the SHA-2 family. In: International Workshop on Fast Software Encryption. pp. 244–263. Springer (2012)
34. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 723–732 (1992)
35. Landerreche, E., Stevens, M., Schaffner, C.: Non-interactive cryptographic timestamping based on verifiable delay functions. In: International Conference on Financial Cryptography and Data Security. pp. 541–558. Springer (2020)
36. Lenstra, A.K., Wesolowski, B.: A random zoo: sloth, unicorn, and trx. IACR Cryptol. ePrint Arch. **2015**, 366 (2015)
37. Mahmoody, M., Moran, T., Vadhan, S.: Publicly verifiable proofs of sequential work. In: Proceedings of the 4th conference on Innovations in Theoretical Computer Science. pp. 373–388 (2013)
38. Mahmoody, M., Smith, C., Wu, D.J.: Can verifiable delay functions be based on random oracles? ICALP (2020)
39. Mendel, F., Nad, T., Schläffer, M.: Improving local collisions: new attacks on reduced SHA-256. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 262–278. Springer (2013)
40. Merkle, R.C.: One way hash functions and DES. In: Conference on the Theory and Application of Cryptology. pp. 428–446. Springer (1989)
41. Micali, S.: Computationally sound proofs. SIAM Journal on Computing **30**(4), 1253–1298 (2000)
42. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), available Online: https://bitcoin.org/bitcoin.pdf, last accessed March 2022
43. OWASP: OWASP Top Ten 2017: A2:2017-Broken Authentication (2017), available Online: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication, last accessed March 2022
44. Pass, R., Shi, E.: The sleepy model of consensus. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 380–409. Springer (2017)
45. Pietrzak, K.: Simple verifiable delay functions. In: 10th innovations in theoretical computer science conference (itcs 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
46. Rescorla, E.: The transport layer security (TLS) protocol version 1.3 (2018), available Online: https://tools.ietf.org/html/rfc8446, last accessed March 2022
47. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
48. Roth, A.E., Ockenfels, A.: Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the Internet. American economic review **92**(4), 1093–1103 (2002)
49. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)

50. Song, F., Yun, A.: Quantum security of NMAC and related constructions. In: Annual International Cryptology Conference. pp. 283–309. Springer (2017)
51. Starkware: Presenting: VeeDo a STARK-based VDF Service (2020), available Online: https://medium.com/starkware/presenting-veedo-e4bbff77c7ae, last accessed March 2022
52. Unruh, D.: Non-interactive zero-knowledge proofs in the quantum random oracle model. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 755–784. Springer (2015)
53. Unruh, D.: Collapse-binding quantum commitments without random oracles. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 166–195. Springer (2016)
54. Unruh, D.: Post-quantum security of Fiat-Shamir. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 65–95. Springer (2017)
55. Wesolowski, B.: Efficient verifiable delay functions. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 379–407. Springer (2019)
56. Yang, Z., Qin, B., Wu, Q., Shi, W., Liang, B.: Experimental comparisons of verifiable delay functions. In: International Conference on Information and Communications Security. pp. 510–527. Springer (2020)