Short-lived Zero-Knowledge Proofs and Signatures

Arasu Arun[1(✉)], Joseph Bonneau[1,2], and Jeremy Clark[3]

[1] New York University, New York, NY, USA
arasu@nyu.edu
[2] University of Melbourne, Melbourne, VIC, Australia
[3] Concordia University, Montreal, QC, Canada

Abstract. We introduce the short-lived proof, a non-interactive proof of knowledge with a novel feature: after a specified period of time, the proof is no longer convincing. This time-delayed loss of soundness happens "naturally" without further involvement from the prover or any third party. We propose definitions for short-lived proofs as well as the special case of short-lived signatures. We show several practical constructions built using verifiable delay functions (VDFs). The key idea in our approach is to allow any party to forge any proof by executing a large sequential computation. Some constructions achieve a stronger property called reusable forgeability in which one sequential computation allows forging an arbitrary number of proofs of different statements. We also introduces two novel types of VDFs, re-randomizable VDFs and zero-knowledge VDFs, which may be of independent interest. Our constructions for short-lived Σ-protocols and signatures are practically efficient for provers and verifiers, adding a few hundred bytes of overhead and tens to hundreds of milliseconds of proving/verification time.

Keywords: Zero-knowledge proofs · Signatures · VDFs · Time-based crypto

1 Introduction

A digital signature is forever. Or at least, until the underlying signature scheme is broken or the signing key is breached. This is often much more than what is required for real world applications: a signature might need to only provide authenticity for a few seconds to conduct an authenticated key exchange or verify the provenance of an email. At best, the long-lived authentication provided by standard signatures is often unnecessary. In certain cases, however, it may have significant undesirable consequences.

An illustrative example is the DKIM protocol [53] used by modern SMTP servers to sign outgoing email on behalf of the entire domain (e.g., `example.com`) with a single key. DKIM is primarily intended to prevent email spoofing [27]. As such, these signatures only need a lifetime of minutes for recipient SMTP servers to verify and potentially filter email. However DKIM signatures do not expire and instead provide long-lasting evidence of authenticity for old email messages, such as ones leaked through illicit data breaches [72]. As a result, cryptographers have suggested that DKIM servers should periodically rotate keys and reveal old private keys to provide deniability for old DKIM signatures [45].

Our Approach. A short-lived proof convinces the verifier of the following: either a claimed statement x is true or someone expended at least t steps of sequential work to forge the proof. The proof incorporates a random beacon value (e.g., the day's newspaper headline) to ensure it was not created before a specific time T_0. If a verifier observes the proof within Δ units of time after T_0, she will believe it is a valid proof if $\Delta < t$ because it would be impossible to have forged the proof within that time period. Once $\Delta \geq t$, the proof is no longer convincing as it may have been constructed through the forgery process.

Our constructions build on recent advances in time-based cryptography, specifically *verifiable delay functions* (VDFs) [16,64,77]. Under the hood, the sequential computation required for forging a proof or signature in all of our schemes is equivalent to evaluating a VDF on a random input.

Cryptographic Deniability. The idea that signatures should not be permanently verifiable is a special case of *cryptographic deniability.* This is often weaker than intuition suggests. Informally, deniability for signatures means there is no *additional* cryptographic proof that Alice sent a particular message. There may still be circumstantial proof such as logs or testimony, but these would exist whether or not cryptography was used at all.

A simple approach to deniability, as suggested for DKIM, is to publish secret information after running a protocol which enables any party to forge transcripts. Other approaches include deniable key exchange protocols (e.g., OTR messaging [21]) or designated verifier proofs/signatures [49] which limit verifiability to a specified set of parties. By contrast, short-lived proofs are non-interactive and publicly verifiable yet become deniable after a specified period of time without any further action by the prover. For signatures, the signer can maintain a long-lived key even as messages signed with it expire.

The fact that short-lived signatures provide deniability without the sender needing to interact with the recipient (or even know the receivers' public key) makes them uniquely qualified for achieving deniability in several practical scenarios, as we discuss in Sect. 10. An example is sending email with a single signature to a large, potentially unknown group of recipients. To our knowledge, ours is the first primitive to enable this.

To summarize our contributions, as outlined in Table 1:

- We define short-lived zero-knowledge proofs (§4) and short-lived signatures (§8). *Reusable forgeability* captures the useful property of a single slow computation enabling efficient proof forgery for any statement (§4.1).
- We propose a short-lived proof construction (with reusable forgeability) from any generic non-interactive zero-knowledge proof scheme and any VDF (§5).

Table 1. A comparison of our constructions. The symbol ◐ denotes schemes with a time-space tradeoff in the delay parameter t (see §9).

Scheme	Section	Reusable forgeability	No pre-computation	VDF required	Proof/Sign type
			short-lived proofs		
Generic ZK	§5	●	●	any [16]	Generic SNARK
Σ-Precomp	§6.2	○	○	any [16]	Σ-protocol
Σ-rrVDF	§6.3	○	◐	re-randomizable (§A)	Σ-protocol
Σ-zkVDF	§7	◐	●	zero-knowledge (§7.1)	Σ-protocol
			short-lived signatures		
Generic ZK	§8	●	●	any [16]	Generic SNARK
$\Sigma_{\text{sign}} \vee$ zkVDF	§8	◐	●	zero-knowledge (§7.1, §C)	Σ signatures
Sign-Trapdoor	§8.1	○	●	trapdoor ([64,77])	RSA
Sign-Watermark	§8.2	◐	●	watermarkable ([77], §B)	RSA

- We propose a short-lived proof construction (§6.2) for any Σ-protocol (§2.2) and any VDF with a Σ-protocol for verification. Our basic scheme requires precomputation per-proof, which we can eliminate by introducing the concept of re-randomizable VDFs .
- We introduce the notion of zero-knowledge VDFs (§7) and use it to build a short-lived proof with reusable forgeability for Σ-protocols.
- We show that our general Σ-construction can be instantiated with Σ-signatures such as Schnorr (§8). We further introduce highly efficient short-lived signature schemes (§8.1) from trapdoor VDFs [77] and watermarkable VDFs which are as compact as a single VDF proof and offer reusable forgeability.

2 Preliminaries

2.1 Zero-knowledge Proofs and Arguments

We start with a basic background on zero-knowledge proofs, while referring the reader to [74] for more comprehensive introduction. Zero-knowledge proofs concern a relation $R \subset \mathcal{X} \times \mathcal{W}$ — a set of pairs (x, w) where x is called the statement or instance and w is called the witness. For example, for the relation of Diffie-Hellman tuples, we might write: $R_{\text{DH3}} = \{(x = (g, g_1, g_2, g_3), w = (a, b)) | g_1 = g^a \wedge g_2 = g^b \wedge g_3 = g^{ab}\}$. The set of all values x such that there exists a witness w for which $(x, w) \in R$ is the language L_R. It has been shown that all

NP-languages have a zero-knowledge proof system [43]. A non-interactive zero knowledge proof system Π_R for R is a trio of algorithms (we consider R to be hard-coded into all three):

- $\Pi.\mathsf{Setup}(\lambda) \rightarrow pp$
- $\Pi.\mathsf{Prove}(pp, x, w) \rightarrow \pi$
- $\Pi.\mathsf{Verify}(pp, x, \pi) \rightarrow \mathsf{Accept}/\mathsf{Reject}$

A proof system is *complete* if, for all $(x, w) \in R$, given a proof $\pi \leftarrow \mathsf{Prove}(pp, x, w)$, the verification algorithm $\mathsf{Verify}(pp, x, \pi)$ outputs Accept. A proof system is *sound* if an unbounded malicious prover (who does not know w) cannot produce an acceptable proof with probability greater than $2^{-\kappa}$ for knowledge error κ. The weaker notion of computational soundness holds for polynomial-time malicious provers; for simplicity we refer to such *argument systems* as proofs.

A *proof of knowledge* has an additional property roughly stating that an adversary must "know" a witness w to compute a proof for $(x, w) \in R$. Knowledge soundness for proofs-of-knowledge is formalized by defining an algorithm $\mathcal{A}$ which outputs an accepted proof π and demonstrating an efficient algorithm $\mathsf{Extract}$ which can interact with $\mathcal{A}$ and output a witness w such that $(x, w) \in R$. Depending on the proof construction, the extractor may need to rewind $\mathcal{A}$ (a rewinding extractor) or inspect $\mathcal{A}$'s internal state (a non-black box extractor).

A proof of knowledge is *zero knowledge* if the proof π reveals nothing about the witness w. Formally, this is established by demonstrating an efficient algorithm $\mathsf{Simulate}$ which, given any statement $x \in L_R$ can output simulated proofs $\tilde{\pi}$ indistinguishable from real proofs such that $\mathsf{Verify}(pp, x, \tilde{\pi}) \rightarrow \mathsf{Accept}$. $\mathsf{Simulate}$ may require additional power, such as the ability to *program* the random oracle to give specified responses.

If the system produces *succinct* (e.g., constant or poly-logarithmic sized) arguments, it is a SNARK (or zk-SNARK) for R, of which there are now many known constructions [13,42,46].

2.2 Sigma Protocols

Our constructions in Sects. 6 and 7 target a special class of interactive zero-knowledge proof systems called Σ-protocols [70]. A Σ-protocol [70] is a three-move interactive protocol between a prover P and verifier V:

1. P runs $\Sigma.\mathsf{Commit}(x) \rightarrow a$ and sends a to V.
2. V runs $\Sigma.\mathsf{Challenge}() \rightarrow c$ and sends c to P.
3. P runs $\Sigma.\mathsf{Respond}(x, w, a, c) \rightarrow z$ and sends z to V.
4. V accepts if $\Sigma.\mathsf{Verify}(x, a, c, z) \rightarrow \mathsf{Accept}$.

We write Σ_R to denote a Σ-protocol for relation R. A Σ-protocol has *special soundness* if there exists an algorithm $\Sigma.\mathsf{Extract}(x, a, c, c', z, z')$ which outputs a witness w for x given any two accepting transcripts of the form (x, a, c, z) and (x, a, c', z') with $c \neq c'$. A Σ-protocol is honest verifier zero-knowledge if it has an efficient algorithm $\Sigma.\mathsf{Simulate}(x) \rightarrow (\tilde{a}, \tilde{c}, \tilde{z})$ such that $\Sigma.\mathsf{Verify}(x, \tilde{a}, \tilde{c}, \tilde{z}) \rightarrow \mathsf{Accept}$ and the distribution of $(x, \tilde{a}, \tilde{c}, \tilde{z})$ is indistinguishable from transcripts of a genuine interaction between a verifier and prover knowing a witness w.

Every Σ-protocol can be transformed into a non-interactive, fully secure (i.e., no honesty assumption on the verifier) zero-knowledge proof in the random oracle model using the Fiat-Shamir heuristic [39], in which the challenge is generated as $c = \mathcal{O}(x, a)$ where $\mathcal{O}$ is the random oracle. $\Sigma.\mathsf{Extract}$ and $\Sigma.\mathsf{Simulate}$ make use of rewinding the other party and programming the random oracle.

2.3 Disjunction of Σ-protocols

The set of relations with Σ-protocols is closed under conjunction and disjunction [32]. The classic protocol for disjunction of Σ-protocols, which we denote Σ-OR, is due to Cramer et al. [32].[1] Let Σ_{R_1} and Σ_{R_2} be Σ-protocols for relations R_1 and R_2 respectively. Assume the prover wants to prove the disjunction of the statement $x = (x_1, x_2)$ and knows a witness w_1 showing that $(x_1, w_1) \in R_1$ (knowing w_2 showing that $(x_2, w_2) \in R_2$ is a symmetric case). The proof is constructed as follows:

$\underline{\mathsf{Protocol}\ \Sigma_{R_1 \vee R_2}(x_1, w_1, x_2, -)}$:

1. P runs $\Sigma_{R_2}.\mathsf{Simulate}(x_2) \rightarrow (\tilde{a_2}, \tilde{c_2}, \tilde{z_2})$
2. P runs $\Sigma_{R_1}.\mathsf{Commit}(x_1) \rightarrow a_1$
3. P send $(a_1, \tilde{a_2})$ to V
4. V sends $c = \Sigma.\mathsf{Challenge}()$ to P
5. P sets $c_1 = c \oplus \tilde{c_2}$
6. P runs $\Sigma_{R_1}.\mathsf{Respond}(x_1, w_1, a_1, c_1) \rightarrow z_1$
7. P sends $(c_1, \tilde{c_2}, z_1, z_2)$ to V
8. V accepts if $c = c_1 \oplus \tilde{c_2}$ and both:
   - $\Sigma_{R_1}.\mathsf{Verify}(x_1, a_1, c_1, z_1) \rightarrow \mathsf{Accept}$
   - $\Sigma_{R_2}.\mathsf{Verify}(x_2, \tilde{a_2}, \tilde{c_2}, \tilde{z_2}) \rightarrow \mathsf{Accept}$

2.4 Beacons

A beacon [65] is a continual source of unpredictable public randomness. A beacon's output at time T_i should be uniformly random and unpredictable as of time T_{i-1}. We assume that beacon values are drawn uniformly randomly from a space $|\mathcal{B}| \geq 2^{\lambda}$ for the security parameter λ. All our protocols assume an input beacon value denoted b. In practice, NIST operates a centralized beacon which publishes 512 random bits every minute [4]. The drand project operates a public beacon publishing 256 bits every 30 s [2] using a multi-party randomness protocol [73]. Other potential beacons include web server challenges [48], stock market prices [26], and blockchain data [20].

[1] Ciampi et al. [25] later introduced a different Σ-OR protocol with certain advantages over the Cramer et al. construction.

2.5 Verifiable Delay Functions

Verifiable delay functions (VDFs) are defined by a trio of algorithms:[2]

- $\mathsf{VDF.Setup}(\lambda, t) \rightarrow pp$
- $\mathsf{VDF.Eval}(pp, b) \rightarrow (y, \pi)$
- $\mathsf{VDF.Verify}(pp, b, y, \pi) \rightarrow \mathsf{Accept/Reject}$

Boneh et al. formalized VDFs and offer formal security definitions [16]. Informally, VDFs satisfy three important properties: (1) Verifiability, meaning that the verification algorithm is efficient (at most polylogarithmic in t and λ) and always accepts when given genuine output from Eval. (2) VDF evaluation must be a function, meaning that Eval is a deterministic algorithm and it is computationally infeasible to find two pairs $(b, y), (b, y')$ with $y \neq y'$ that Eval will accept. And (3) VDFs must impose a computational delay. Roughly speaking, computing a VDF successfully with non-negligible probability over a uniformly distributed challenge b should be impossible without executing t sequential steps. Throughout this paper, we will refer to this property as "t-Sequentiality" to emphasize the time delay parameter. All of our constructions reduce forging a proof to evaluating a VDF on a random input. Formally:

Property 1 (Sequentiality of VDFs (from [16]*)).* For functions $\sigma(t), p(t)$, the VDF is (p, σ)-Sequential if for all randomized algorithms $\mathcal{A}_0$ which run in total time $O(poly(t, \lambda))$, and $\mathcal{A}_1$ which run in parallel time $\sigma(t)$ on at most $p(t)$ processors, the probability of winning the following game is negligible:

1. $pp \leftarrow \mathsf{Setup}(\lambda, t)$
2. $\alpha \leftarrow \mathcal{A}_0(\lambda, pp, t)$ // advice string
3. Challenger samples VDF input b
4. $y_A \leftarrow \mathcal{A}_1(\alpha, pp, b)$
5. Adversary wins if $y_A = y$ where $y, \pi \leftarrow \mathsf{Eval}(pp, b)$

VDFs constructions have been proposed from generic succinct proofs [16], repeated squaring in groups of unknown order [64,77], permutation polynomials [16], isogenies [37], and homomorphic encryption [50]. Earlier work proposed "weak" VDFs based on computing square roots mod p [35,55]. Proof-of-sequential work (PoSW) [28,56] is a similar primitive that does not require the evaluation to have a unique mapping. Modern VDF constructions are in fact the most efficient known PoSW constructions; for simplicity we present all our constructions using the notation and terminology of VDFs.

An important limitation of all VDF constructions is that they can only guarantee a certain number of steps of sequential computation are required. The real-world or "wall-clock" time needed to execute this computation varies based on the speed of available computing platforms. To manage this limitation, conventional wisdom suggests using a VDF with a relatively simple evaluation function for which highly optimized hardware is available to honest parties, limiting the speedup available to attackers. For this reason, repeated-squaring based VDFs are considered the most practical candidates today.

[2] The VDF challenge is traditionally denoted x. We use b to avoid confusion with x as the statement of a zero-knowledge proof.

2.6 VDFs from Repeated Squaring

We focus in particular on VDF constructions which utilize repeated squaring in a group of unknown order, as these have useful algebraic properties for building short-lived proofs and signatures. VDF evaluation is simply $y = \mathsf{VDF.Eval}(b) = b^{2^t}$. Wesolowski [77] and Pietrzak [64] introduced two distinct approaches for efficiently proving that $y = b^{2^t}$ in a group of unknown order:

Wesolowski Proofs: First, the Prover provides $\tilde{y}$, claiming $\tilde{y} = b^{2^t}$. The verifier provides a random prime ℓ as a challenge. Both parties compute, via long division, the unique values q, r such that $2^t = q\ell + r$ and $0 \leq r < \ell$. Finally, the prover outputs a proof $\pi = b^q$. The verifier accepts if and only if $\tilde{y} = q^\ell b^r$.

Pietrzak Proofs: As before, the Prover provides $\tilde{y}$, claiming $\tilde{y} = b^{2^t}$. The prover then provides a value v and asserts that $v = b^{2^{t/2}}$. The verifier chooses a random challenge r and they both compute $\tilde{y}' = \tilde{y} \cdot v^r, b' = v \cdot b^r$. The verifier could manually verify that $\tilde{y}' = (b')^{(2^{t/2})}$ by computing $\frac{t}{2}$ squarings, half as many as the original problem of verifying that $\tilde{y} = b^{(2^t)}$. Alternately, the prover can recursively prove that $\tilde{y}' = (b')^{(2^{t/2})}$. Typically, this is done for d rounds, each halving the size of the exponent, until the verifier manually checks the remaining exponent of size $2^{t/2^d}$.

Boneh et al. [17] provide a detailed comparison of the two proof constructions. Wesolowski proofs are shorter (two group elements instead of $O(\log t)$) but more difficult to compute and rely on slightly stronger assumptions. In this work we observe a new property of both constructions, re-randomizability (§6.3), and introduce a new zero-knowledge variant of Wesolowski proofs (§7).

3 Related Work

Jakobsson et al. introduced the idea of using disjunctive statements to provide deniability [49]. Given a statement x to be proven to Bob in zero knowledge, the proof is transformed into the statement: {*either x or I know Bob's private key*}. A proof of this compound statement, which is called a *designated verifier proof*, is only convincing to Bob. If Bob is confident that nobody else knows his private key and that he did not compute the proof, then he knows the second clause is false and therefore x is true. Anybody else is unsure if x is true or if Bob *forged* the proof by satisfying the second clause. Another approach to constructing signatures with the designated-verifier property is *chameleon signatures* [52], which use a standard hash-and-sign construction but with a chameleon hash function whose trapdoor is known by the intended verifier.

Many of our constructions[3] follow a similar disjunctive template, with the essential statement being {*either x or someone solved a VDF of difficulty t on a*

[3] An initial version of this work appeared as a Masters thesis [30].

beacon value derived from b which was unknown before time T }. VDFs requires t sequential steps (which approximate elapsed time).

Several other works have used disjunctive proofs to provide different notions of deniability. Baldimtsi et al. showed the constructions and applications for *proofs-of-work-or-knowledge* (PoWorKs) of the form {*either x or someone solved a proof-of-work puzzle*} where the puzzle requires w units of parallelizable computation [8]. Time-traveling simulators, introduced by Goyal et al. [44], provide a similar deniability notion in which a proof is convincing until a blockchain grows to a certain length. Specter et al. proposed {*either x or someone has seen value v released at time T*} for a v to be published at a future time T [72]. This proposal is closest to our own work, as we discuss further in Sect. 10.2.

Similar time-based deniability notions for signatures specifically have been considered by several authors (we believe ours is the first to expand to general proofs). Ferradi et al. [38] in 2015 presented a protocol for what they call *fading signatures* using the RSW time-lock puzzle and a trusted authority to pre-compute some solutions using the trapdoor. Their notion is weaker in that verification is slow, requiring t sequential steps. In hindsight, with the benefit of modern VDFs the slow verification time of their approach could be fixed.

The connection between VDFs and time-based deniability was made by Wesolowski who presented an interactive identification scheme that becomes deniable after the passage of time [77, §8]. Wesolowski also described a time-limited signature protocol which improved on the Ferradi et al. construction in an unpublished 2016 manuscript [76]. Our Sign-Trapdoor construction (§8.1) improves on this protocol by making it transferable, non-interactive, and a true signature (rather than an authentication protocol).

Contemporaneous to our work, Beck et al. [10] propose a construction for what they call *time-deniable signatures*, which utilize time-lock puzzles. Colburn [30] described a folklore construction (called Folk+) in which a time-lock puzzle encapsulating the signing key is published along with a signature. Green et al.'s construction works similarly, except to enable continuous use of the key the time-lock puzzle encapsulates a *restricted signing oracle* which can only sign messages with a timestamp before a chosen expiration date. This construction appears inherently limited to signatures. It also utilizes completely different cryptographic techniques and as a result the reported signing time is over 4 s per message, orders of magnitude slower than our signature constructions.

Other time-based cryptographic primitives have been proposed including encryption, commitments, and signatures [15,66,75]. In the context of the cited literature, a *timed signature* [15] is a commitment to a signature that has been shared and can later be revealed. However if the committer aborts before revealing, the recipient can perform sequential work to uncover the signature. Dodis and Yum introduced a similar idea of *time-capsule signatures* [33] which become valid after a certain period of time when a time-server releases some information. We are essentially solving the inverse problem: instead of a signature being hidden for time Δt and then becomes unforgeable, a short-lived signature is unforgeable for Δt and then becomes deniable.

Definition 1 (Short-Lived Proofs) *Let λ be a security parameter. Let L_R be a language in* NP *and R be a relation such that $(x, w) \in R$ if and only if w is a witness showing $x \in L_R$. Let $\mathcal{B}$ be a space of beacon values where $|\mathcal{B}| \geq 2^\lambda$. A short-lived proof system Π_R^t with time delay $t \in \mathbb{Z}$ is a quartet of randomized algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Forge}, \mathsf{Verify})$*:*

- $\mathsf{Setup}(L, \lambda, t) \rightarrow pp$ *produces a set of public parameters pp*
- $\mathsf{Prove}(pp, x, w, b) \rightarrow \pi$ *produces a proof π if $(x, w) \in R$*
- $\mathsf{Forge}(pp, x, b) \rightarrow \pi$ *produces a proof π for any x*
- $\mathsf{Verify}(pp, x, \pi, b) \rightarrow \mathit{Accept}/\mathit{Reject}$

Π_R^t must satisfy the following properties:

- **Completeness:** *For all $(x, w) \in R$ and $b \in \mathcal{B}$, $\pi \leftarrow \mathsf{Prove}(pp, x, w, b)$ runs in time less than t and $\mathsf{Verify}(pp, x, \pi, b)$ outputs Accept.*
- **t-Forgeability:** *For all x, $b \in \mathcal{B}$, $\pi \leftarrow \mathsf{Forge}(pp, x, b)$ runs in time $(1 + \epsilon)t$ for some positive constant ϵ and $\mathsf{Verify}(pp, x, \pi, b)$ outputs Accept.*
- **t-Soundness:** *For all x and for any pair of adversary algorithms $\mathcal{A}_0$ (precomputation) which runs in total time $O(poly(t, \lambda))$ and $\mathcal{A}_1$ (online) which runs in parallel time $\sigma(t)$ with at most $p(t)$ parallel processors, if*

$$Pr\left[\begin{array}{c} \alpha \leftarrow \mathcal{A}_0(pp, x); \\ b \xleftarrow{\$} \mathcal{B}; \\ \pi \leftarrow \mathcal{A}_1(pp, x, b, \alpha); \\ \mathsf{Verify}(pp, x, \pi, b) = \mathit{Accept} \end{array}\right] > \mathsf{neg}(\lambda)$$

  then there exists an algorithm $\mathsf{Extract}$ *with rewinding access to $\mathcal{A}_1$ such that with probability $1 - \mathsf{neg}(\lambda)$ the algorithm $\mathsf{Extract}(pp, x, b)$ outputs a witness w such that $(x, w) \in R$. The probability is over the choice of b and the random coins used by each algorithm.*
- **Indistinguishability:** *For all $(x, w) \in R, b \in \mathcal{B}$ the distributions $\{\mathsf{Prove}(pp, x, w, b)\}$ and $\{\mathsf{Forge}(pp, x, b)\}$ (taken over the random coins used by each algorithm) are computationally (resp. statistically) indistinguishable.*

4 Definitions

Definition 1 provides our main definition of short-lived proofs. The public parameters *pp* potentially encapsulate both setup needed for an underlying proof system and setup needed for an underlying VDF. Either or both may represent a *trusted setup* if they require a secret parameter that can be used to break security assumptions if not destroyed. The Setup algorithm is also given both a description of the language L and delay parameter t. Some underlying proof systems may require setup specific to L (others may offer *universal setup*) and some underlying VDFs require hard-coding the delay parameter t. In the remainder of the paper, we will generally omit pp to keep notation simpler.

The critical security property, t-Soundness,[4] closely follows the security definition used for VDFs [16]. In our case, the (potentially long-running) preprocessing algorithm $\mathcal{A}_0$ receives not only the public parameters of a VDF function but also the statement x on which the adversary wishes to forge a proof. Once the random beacon value b is known, the attacker's clock starts running and the online algorithm $\mathcal{A}_1$ must attempt to forge a proof in fewer than $\sigma(t)$ time steps (which in all of our constructions reduces to the intractability of solving an underlying VDF in fewer than $\sigma(t)$ time steps).

Note that short-lived proof schemes are inherently zero-knowledge as the Forge algorithm serves as a simulator which produces valid proofs in polynomial-time without knowing a witness. Receiving a proof in time less than t breaks deniability as it must have been produced by Prove, but does not help the verifier break zero-knowledge as the same transcript could still be produced by Forge at a later time. Thus, t-Forgeability and Indistinguishability implies zero-knowledge and we do not to define a separate zero-knowledge property or an additional simulator[5].

4.1 Reusable Forgeability

A basic short-lived proof scheme allows the Forge algorithm time to perform a unique slow computation for each pair (x, b). In practice, this means that forging multiple proofs can be expensive, weakening the deniability as it becomes less plausible that somebody paid the cost of forging. To this end, some short-lived proof schemes may offer a stronger *reusable forgeability* property in which performing one slow computation for a beacon value b enables efficiently forging a proof for *any* statement x without performing a full additional slow computation. Even better, some schemes might allow forging a proof of any statement for any beacon value from a set $B = \{b_1, \ldots b_k\}$ after just one slow computation. We call this property k-reusable forgeability (with basic reusable forgeability being the special case of $k = 1$). In practice, the set B can comprise all prior beacon values, enhancing deniability as one slow computation at any point in the future could enable forgery of all prior proofs.

Definition 2 (k-Reusable Forgeability). *A k-reusably forgeable short-lived proof system Π^t_R is a short-lived proof with two additional functions:*

[4] Our notion of t-soundness corresponds to *knowledge soundness*, we denote it as t-soundness for conciseness.

[5] Computational and statistical indistinguishability imply computational and statistical zero-knowledge, respectively. Thus, it might be possible to design a simulator that achieves statistical zero-knowledge while Forge only achieves computational indistinguishability.

- $\mathsf{GenAdvice}(pp, B) \rightarrow \alpha$ *takes a set* B *of size* $|B| \leq k$ *and produces (in time* $(1+\epsilon)t$*) an advice string* α
- $\mathsf{FastForge}(pp, x, b, \alpha) \rightarrow \pi$ *produces a proof* π *for any* x

These new functions satisfy the following properties, in addition to all properties of a general short-lived proof system:

- **Reusable Forgeability:** *For all* x *and for all* $B \subseteq \mathcal{B}$ *and* $b \in B$*, given advice string* $\alpha \leftarrow \mathsf{GenAdvice}(pp, B)$ *the algorithm* $\mathsf{FastForge}(pp, x, b, \alpha) \rightarrow \pi$ *runs in parallel time less than* t *and* $\mathsf{Verify}(pp, x, \pi, b)$ *outputs* *Accept.*
- **Indistinguishability II:** *For all* $(x, w) \in R$*,* $B \subseteq \mathcal{B}$ *and* $b \in B$*, given advice string* $\alpha \leftarrow \mathsf{GenAdvice}(pp, b)$ *the distributions* $\{\mathsf{Forge}(pp, x, b)\}$ *and* $\{\mathsf{FastForge}(pp, x, b, \alpha)\}$ *(taken over the random coins used by each algorithm) are computationally (resp. statistically) indistinguishable.*

Our generic protocol (§5) offers 1-reusable forgeability immediately and extends easily to offer $k-$reusable forgeability for arbitrary k (with proving overhead logarithmic in k). Obtaining 1-reusable forgeability is also possible (though not as easy) for our Σ-constructions.

5 Short-lived Proofs from Generic Zero-Knowledge

Given any VDF scheme and a non-interactive zero-knowledge proof system Π for all languages in NP, we can produce a short-lived proof for any relation R for an NP language L_R. We do this by taking the disjunction ($\vee$) of R with the VDF relation R_{VDF}:

$$R_{\text{VDF}} = \{(x = b, w = (y, \pi)) \mid \mathsf{VDF.Verify}(b, y, \pi)\} \tag{1}$$

The language $L_{R_{\text{VDF}}}$ is in NP because VDF verification must run in polynomial time. Therefore, the disjunction $L_{R \vee R_{\text{VDF}}}$ is in NP and the proof protocol $\Pi_{R \vee R_{\text{VDF}}}$ is a short-lived proof for R:

Theorem 1 (SLP from Generic Zero-Knowlege). *Let* R *be a relation for a language in* NP*,* Π *be a zero-knowledge argument of knowledge system for languages in* NP *and* VDF *be a verifiable delay function with delay parameter* t*. Then* $\Pi_R^t = \Pi_{R \vee R_{\mathsf{VDF}}}$ *is a short-lived proof protocol with reusable forgeability for* R *with time delay* t

Proof. The required properties follow directly from definitions of the underlying primitives. Completeness of Π_R^t is due to the completeness of Π_R as a prover with a witness can satisfy relation R and ignore the VDF branch. t-Forgeability follows from the correctness property of the underlying VDF, ensuring that Forge can produce convincing forgeries in $(1+\epsilon)t$ steps by running $\mathsf{VDF.Eval}(b)$ and using the output (y, π) to satisfy the VDF branch of the disjunction. The Indistinguishability property followsmmediately from the zero-knowledge property of Π, preventing the adversary from knowing which half of the disjunction was satisfied and meaning an efficient simulator exists as required.

The <u>t-Soundness</u> property relies on the t-Sequentiality (Property 1) of the VDF. The restrictions on algorithms $\mathcal{A}_0, \mathcal{A}_1$ in the t-Soundness definition are identical to those in the t-Sequentiality definition, meaning such algorithms will not be able to solve the VDF with non-negligible probability. This means that any adversary able to produce proofs must know a witness w for x, which the extractor for Π_R can then efficiently extract.

Finally, to show that this scheme offers reusable forgeability, note that the exact same VDF computation $\mathsf{VDF.Eval}(b)$ required is independent of the statement x. Thus, it can be computed once and reused across proofs for that beacon. □

6 Short-lived Proofs from Σ-protocols

While our generic construction offers reusable forgeability and works for all NP-languages, generic zero-knowledge proof systems have practical drawbacks including complexity, high prover costs (§9) and trusted setup in some constructions. We would like to construct short-lived variants for Σ-protocols, an important class of efficient zero-knowledge proofs. They are also natural to consider given that Wesolowski proofs [77] are Σ-protocols and Pietrzak proofs are a multi-round generalization.

6.1 Non-solution: Σ-OR Proofs

A first, insecure attempt at a short-lived proof for a relation R with a Σ-protocol Σ_R is to simply combine Σ_R with the verification protocol Σ_{VDF} for some VDF scheme, for example using the classic Σ-OR construction outlined in Sect. 2.3.

Unfortunately, this generic composition does not yield a short-lived proof system because proofs are distinguishable from forgeries. Standard VDF proofs reveal the unique[6] value $y = \mathsf{VDF.Eval}(b)$ to the verifier as part of the proof statement. This means that the algorithm $\mathsf{VDF.Prove}$, which simulates the VDF half of a Σ-OR composition, must provide a fake value $y' \neq y$ as part of the proof whereas the Forge algorithm will simulate the R half of the proof and provide the genuine $y = \mathsf{VDF.Eval}(b)$. Our definition of Indistinguishability does not preclude the adversary from running for t steps, meaning they can simply compute the genuine value y themselves by running $\mathsf{VDF.Eval}(b)$ and then determine if a proof was constructed using Prove or Forge.

6.2 Short-lived Sigma Proofs from Precomputed VDFs

To ensure indistinguishability, we introduce the construction Σ-Precomp (Protocol 1) which works for any Σ-protocol by modifying the *input* to the VDF instead of the challenge. Assume the prover has precomputed a VDF on a random input value b^*. Just as in the Cramer et al. construction, given a challenge

[6] Proofs of sequential work do not have unique solutions, unlike VDFs, meaning they might be used directly in a Σ-OR composition.

```
Σ-Precomp

Setup
input: relation $R$, parameters $\lambda, t$
output: public parameters $pp$

  1. $pp \leftarrow \mathsf{VDF.Setup}(\lambda, t)$

Precompute
input: $pp$
output: $(b^*, y^*, \pi^*_{\mathrm{VDF}})$

  1. Sample $b^* \xleftarrow{\$} \mathcal{B}$
  2. $(y^*, \pi^*_{\mathrm{VDF}}) \leftarrow \mathsf{VDF.Eval}(b^*)$

Verify
input: $pp$, $x$, $b$, proof $(a, c_1, z, y, c_2, \pi_{\mathrm{VDF}})$
output: Accept/Reject

  1. Obtain $c = \mathcal{O}(x \parallel b \parallel a)$.
  2. Accept if $c = c_1 \oplus c_2$ and both:
       • $\Sigma_R.\mathsf{Verify}(x, a, c_1, z) \to \mathsf{Accept}$
       • $\mathsf{VDF.Verify}(b \oplus c_2, y, \pi_{\mathrm{V}}) \to \mathsf{Accept}$

Forge
input: $pp$, statement $x$, beacon value $b$
output: proof $(\tilde{a}, \tilde{c}_1, \tilde{z}, y, \pi_{\mathrm{VDF}}, c_2)$

  1. $(\tilde{a}, \tilde{c}_1, \tilde{z}) \leftarrow \Sigma_R.\mathsf{Simulate}(x)$
  2. Obtain $c = \mathcal{O}(x \parallel b \parallel \tilde{a})$
  3. Set $c_2 = c \oplus c_1$
  4. $(y, \pi_{\mathrm{VDF}}) \leftarrow \mathsf{VDF.Eval}(b \oplus c_2)$.

Prove
pre-computed: $(b^*, y^*, \pi^*_{\mathrm{VDF}})$
input: $pp$, $x$, $w$, $b$
output: proof $(a, c_1, z, y^*, \pi^*_{\mathrm{VDF}}, c_2)$

  1. $a \leftarrow \Sigma_R.\mathsf{Commit}(x)$
  2. Challenge $c = \mathcal{O}(x \parallel b \parallel a)$
  3. Set $c_2 = b^* \oplus b$ and $c_1 = c \oplus c_2$
  4. $z = \Sigma_R.\mathsf{Respond}(x, w, a, c_1)$.
```

Protocol 1: Short-lived proofs using precomputed VDFs given a relation R with Σ-protocol Σ_R and a VDF scheme VDF

c, the prover chooses two values c_1, c_2 such that $c_1 \oplus c_2 = c$ and c_1 is the challenge used with Σ_R. Instead of using c_2 as the challenge for the VDF proof, it is used to modify the VDF input, evaluating on the point $b \oplus c_2$. The intuition is that the genuine prover can choose c_2 freely and thus set $c_2 = b \oplus b^*$, mapping the VDF input to a value b^* for which it has already precomputed the solution. However, a forger who is simulating Σ_R cannot choose c_1 freely and thus c_2 is an unpredictable random value, which requires the forger to solve a VDF on a random point $(b \oplus c_2)$.

Theorem 2 (Proof for Σ-Precomp). *Let R be a relation for a language in NP with Σ-protocol Σ_R and* VDF *be a verifiable delay function with delay parameter t. Protocol 1 is a short-lived proof scheme for relation R in the random oracle model.*

Proof. Completeness and t-forgeability of Σ-Precomp follow directly from the completeness of Σ_R and correctness of the VDF.

Indistinguishability: For an input x, witness w and beacon b, let (a, c_1, z, y, π, c_2) and $(\tilde{a}, \tilde{c_1}, \tilde{z}, \tilde{y}, \tilde{\pi}, \tilde{c_2})$ be the outputs of $\mathsf{Prove}(x, b; w)$ and $\mathsf{Forge}(x, b)$ and let $c, \tilde{c}$ be the respective challenges.

First, we note that $c_2, \tilde{c_2}$ are uniformly distributed in both algorithms. In Prove, $c_2 = b \oplus b^*$ where b is the beacon output produced only after b^* is sampled (as the honest prover must have pre-computed the VDF output on b^* first). In Force, $\tilde{c_2} = \tilde{c} \oplus \tilde{c_1}$ where $\tilde{c}$ is generated after $\tilde{c_1}$. This means the VDF proofs are generated on uniformly random inputs $(c_2 \oplus b)$ and $(\tilde{c_2} \oplus b)$, respectively, making the two VDF sub-transcripts (y, π, c_2) and $(\tilde{y}, \tilde{\pi}, \tilde{c_2})$ indistinguishable.

Next, we note that $c_1, \tilde{c_1}$ are both uniformly distributed, as well. In Prove, $c_1 = c \oplus c_2$ where c is a random oracle output independent of c_2. By the security of $\Sigma_R.\mathsf{Simulate}$, the value $\tilde{c_1}$ is generated in Forge must be indistinguishable from a randomly generated challenge. Again by the security of $\Sigma_R.\mathsf{Simulate}$, the sub-transcripts (a, c_1, z) and $(\tilde{a}, \tilde{c_1}, \tilde{z})$ are indistinguishable.

Finally, all pairs of sub-challenges (c_1, c_2) are equally likely to be generated by both Prove and Forge. This comes from the fact that each algorithm generates one sub-challenge that is uniformly distributed (c_2 for Prove and c_1 for Forge) and then creates the other challenge using c, which is a random oracle output.

t-Soundness: We define an extractor E which, given a pair of algorithm ($\mathcal{A}_0$, $\mathcal{A}_1$) which output accepting proofs, either (1) extracts a witness w from $\mathcal{A}_1$ for statement x and relation R or (2) computes a VDF output on a random input in fewer than t steps, violating the t-Sequentiality (Property 1) of the underlying VDF.

E first runs $\mathcal{A}_0$ and $\mathcal{A}_1$ to obtain an accepting transcript $(a, c, c_1, c_2, z, y, \pi)$. E then receives a random VDF input b_{chal} from a challenger for the VDF t-sequentiality game. Next, the extractor rewinds $\mathcal{A}_1$ to obtain a new transcript $(a, c', c_1', c_2', z', y', \pi')$ while programming the random oracle to fix $c' = b_{\mathsf{chal}} \oplus b \oplus c_1$. As $c \neq c'$, we have the following two cases:

Case 1: If $c_1 \neq c_1'$, then from the special soundness of Σ_R, a witness for x can be extracted by calling $\Sigma_R.\mathsf{Extract}(x, a, c_1, c_1', z, z')$.

Case 2: If $c_1 = c_1'$, the two VDF proofs are on inputs $d = b \oplus c_2 = b \oplus c \oplus c_1$ and $d' = b \oplus c_2' = b \oplus c' \oplus c_1' = b \oplus c' \oplus c_1$. As the extractor programmed the random oracle to ensure $c' = b_{\mathsf{chal}} \oplus b \oplus c_1$, we have $d' = b_{\mathsf{chal}}$. As both transcripts are accepting, $\mathsf{VDF.Verify}(b_{\mathsf{chal}}, y', \pi') = \mathsf{Accept}$. As $\mathcal{A}_1$ runs in fewer than t steps, E requires fewer than t steps to produce y' as it only rewound and reran the adversary *after* after obtaining b_{chal}. Thus, E can output y' and win the t-Sequentiality game for the underlying VDF.

Assuming the underlying VDF is t-Sequential, Case 2 cannot happen except with non-negligible probability. Therefore E correctly outputs a witness for x (Case 1) with overwhelming probability.

Size Overhead: Transforming a normal proof into a short-lived one using Σ-Precomp adds the VDF output, the VDF proof and the sub-challenge used in the VDF. In the case of Wesolowski's VDF [77], the output and proof are both a group element each and the challenge is λ (security parameter) bits long. For 2048-bit RSA groups with $\lambda = 128$, the total size overhead comes to 528 bytes.

The primary drawback of Σ-Precomp is that each precomputed VDF must only be used once. If the same VDF challenge b^* is visible in two proofs, an adversary can conclude (with overwhelming probability) that both proofs were generated by Prove, breaking Indistinguishability. Additionally it does not offer reusable forgeability as a new VDF evaluation on a random point is required for every run of Forge.

6.3 Optimization with Re-randomizable VDFs

The biggest drawback of this construction is that it requires precomputation before every call to Prove. However, the prover simply needs a fresh, random VDF input/output pair and not a solution on any specific point. We can greatly improve the practicality of this scheme if it is possible to quickly generate VDF solutions (and proofs) on random points. We introduce the notion of a *re-randomizable* VDF that has this property: given a VDF solution (b, y, π) and possibly some auxiliary data α, an efficient algorithm $\mathsf{VDF.Randomize}(b, y, \pi, \alpha) \rightarrow (b', y', \pi')$ outputs a randomly distributed solution.

Now, each time Prove is called, instead of precomputing a VDF solution (step 1 of Prove in Protocol 1), a new VDF solution on a random point is produced by calling VDF.Randomize. Indistinguishability of proofs and forgeries reduces to the indistinguishability of random VDF solutions and those generated by re-randomizing a known solution. We propose a definition for re-randomizable VDFs in Appendix A of the full version [7], capturing the necessary indistinguishability property.

For VDFs based on repeated squaring, a random exponent r is chosen and the input/output pair (b, y) is mapped to $(b' = b^r, y' = y^r)$, maintaining the relationship that $y' = (b')^{2^t}$. Unfortunately this homomorphism does not apply to proofs: given $(y, \pi) \leftarrow \mathsf{VDF.Eval}(b)$ and a randomized solution (b^r, y^r) we cannot obtain a correct proof by simply computing π^r. However, for repeated-squaring VDFs we can compute a proof for (b^r, y^r) in fewer than t steps using the same advice string α used to compute π when y was originally computed. Wesolowski [77] describes such an advice string of length $O(\sqrt{t})$ that allows a prover to compute a proof in $O(t/\log t)$ steps. This algorithm may still be to slow to re-randomize VDF proofs in reasonable time using commodity hardware. By contrast, Pietrzak proofs can be re-randomized in just $O(\sqrt{t})$ steps using an advice string of length $O(\sqrt{t})$. We provide the details of this re-randomization algorithm (originally suggested by Boneh et al. [17]) in Appendix A.1 of the full version [7].

This homomorphism was observed by Wesolowski, who warned that it was a potential security weakness to be prevented by hashing to a random group element as part of VDF computation [77, Remark 3]; here we use it in a constructive way. It has similarly been used by Thyagarajan et al. to build verifiable timed signatures [75] and by Malavolta and Thygarajan to construct additively homomorphic and fully homomorphic time-lock puzzles [57].

7 Short-lived Proofs from zkVDFs

The previous Σ-based constructions did not provide reusable forgeability (Definition 2). The fundamental problem is that (unlike our generic approach in

Sect. 5), they require Forge to solve the VDF on a new random value b^* derived from b for each forgery, rather than a solution on b itself which could be used for multiple forgeries. We cannot include a standard VDF proof for b in short-lived proofs because all known VDF proof schemes reveal the VDF output $y = \mathsf{Eval}(b)$ which would clearly distinguish proofs from forgeries.

To avoid this distinguishability problem, we propose using a novel *zero-knowledge VDF* (zkVDF) which proves knowledge of the output without revealing it. Of course, since VDF verification is (by definition) an NP statement, it is possible to construct a zkVDF from any VDF using a generic zero-knowledge proof system to prove knowledge of VDF solutions. Our construction in Sect. 5 essentially does this (embedded within a disjunction). Later in this section, we will present a more efficient construction based on Wesolowski proofs [77].

Given a Σ-protocol for R_{zkVDF} and any relation R for which we have a Σ-protocol Σ_R, we can use the standard Σ-OR construction to create a disjunction protocol $\Sigma_{R\vee R_{\text{zkVDF}}}$ which we call Σ-zkVDF. To obtain reusable forgeability, we set the VDF input to be the beacon value b. Thus, $\mathsf{VDF.Eval}(b)$ need be performed only once to generate advice to quickly forge others proofs with b.

Theorem 3 (SLP from zkVDF and Σ-OR). *Let R be a relation for a language in NP, Σ_R be a zero-knowledge Σ-protocol for R and $\Sigma_{R_{zkVDF}}$ be a Σ-protocol for a zkVDF with delay parameter t. Letting x and w be the statement and witness for relation R and b be the beacon, the following is a short-lived proof protocol with reusable forgeability for R with time delay t:*

Σ-zkVDF

- $\mathsf{Prove}(x, w, b)$*: perform $\Sigma_{R\vee R_{zkVDF}}$ by simulating $\Sigma_{R_{zkVDF}}$ with input b and running Σ_R with statement x and witness w*
- $\mathsf{Forge}(x, b)$*: perform $\Sigma_{R\vee R_{zkVDF}}$ by simulating Σ_R for statement x and running $\Sigma_{R_{zkVDF}}$ with input b*
- $\mathsf{Verify}(x, b, \pi)$*: verify $\Sigma_{R\vee R_{zkVDF}}$ with statement x for Σ_R and input b for $\Sigma_{R_{zkVDF}}$*

The proof is identical to that of Theorem 1. Owing to the security of Σ-OR compositions, the verifier cannot tell if the proof was generated by honestly computing the Σ_R branch (requiring knowledge of the witness) or the $\Sigma_{R_{\mathsf{zkVDF}}}$ branch (requiring the VDF solution on beacon b). Appendix D of the full version [7] contains the proof.

7.1 zkVDF Construction

In this section we present a Σ-based zkVDF construction built off of Wesolowski proofs [77]. To do so, we introduce a new zero-knowledge Σ-protocol for proof of knowledge of a power (Protocol 2) using an idea similar to that introduced by Boneh et al. [18, §3.2] for proof of knowledge of discrete log in a group of unknown order. Our zero-knowledge proof that $y = g^u$ sends a blinded value $y' = y \cdot h^v = g^u h^v$ (for a random v) instead of y itself. A proof of the following theorem is provided in Appendix C of the full version [7].

<u>zk-PoKP</u>

Parameters: security parameter λ, group of unknown order $\mathbb{G} \leftarrow GGen(\lambda)$, $h \xleftarrow{\$} \mathbb{G}$, $B \geq 2^{2\lambda}|\mathbb{G}|$; random oracle HashToPrime which outputs from the set $\mathsf{Primes}(\lambda)$ of the first 2^{λ} prime numbers

Prove
input: $g \in \mathbb{G}$, $u \in \mathbb{Z}$, witness $y \in \mathbb{G}$ such that $y = g^u$
output: proof $\pi = \langle a, Q, r_2 \rangle$

```
1. Sample $v \xleftarrow{\$} [-B, B]$
2. Compute $a = \mathsf{Commit}(y, v) = y \cdot h^v$
3. Compute $\ell = \mathsf{Challenge}(a) = \mathsf{HashToPrime}(a)$
4. Let $u = q_1\ell + r_1$, $v = q_2\ell + r_2$ such that $0 \leq r_1, r_2 \leq \ell$
5. Compute $Q = g^{q_1} h^{q_2}$
6. $\mathsf{Respond}(\ell) = Q, r_2$
```

Simulate
inputs: $g \in \mathbb{G}$, $u \in \mathbb{Z}$, simulated challenge $\tilde{\ell}$
output: simulated proof $\tilde{\pi} = \langle \tilde{a}, \tilde{Q}, \tilde{r_2} \rangle$

```
1. Sample $\tilde{q}_1, \tilde{v} \xleftarrow{\$} [-B, B]$
2. Let $\tilde{u} = \tilde{q}_1\tilde{\ell} + \tilde{r}_1$ and $\tilde{v} = \tilde{q}_2\tilde{\ell} + \tilde{r}_2$ such that $0 \leq \tilde{r}_1, \tilde{r}_2 \leq \tilde{\ell}$
3. Compute $\tilde{Q} = g^{\tilde{q}_1} h^{\tilde{q}_2}$
4. Compute $\tilde{a} = \tilde{Q}^{\tilde{\ell}} g^{r_1} h^{\tilde{r_2}}$
```

Verify
input: $g \in \mathbb{G}$, $u \in \mathbb{Z}$, proof $\pi = \langle a, Q, r_2 \rangle$
output: Accept/Reject

```
1. Compute $\ell = \mathsf{HashToPrime}(a)$
2. Let $u = q_1\ell + r_1$ such that $0 \leq r_1 \leq \ell$
3. Check that $a \stackrel{?}{=} Q^{\ell} g^{r_1} h^{r_2}$
```

Protocol 2: Σ-protocol for proof-of-knowledge of a power in a group of unknown order.

Theorem 4 (Zero-Knowledge Proof of Knowledge of Power). *Protocol 2 is an honest-verifier zero-knowledge argument of knowledge for the relation* $R_{\mathsf{PoKP}} = \{((g, u); y) : g^u = y\}$.

7.2 Efficiency of zk-PoKP and Σ-zkVDF

The zk-PoKP Simulate algorithm of Protocol 2 is efficient and takes time $O(\lambda \log |\mathbb{G}| + \mathsf{polylog}(t))$. The most significant cost is computing five group exponentiations with small exponents, each involving $O(\log B) = O(\lambda \log |\mathbb{G}|)$ steps. This makes the Σ-zkVDF prover efficient as it runs the zk-PoKP simulation algorithm.

The Forge algorithm for Σ-zkVDF must execute the Prove algorithm of zk-PoKP. This naively takes time $O(t)$, as it involves computing a large power Q^ℓ. For multiple forgeries, this can be improved significantly using a precomputed advice string, identical to that used for re-randomizable VDFs (Sect. 6.3). With an advice string of size $O(\sqrt{t})$, the Prove algorithm requires only $O(t/\log t)$ steps. Unlike the case for general re-randomizable VDFs, our zk-PoKP construction is inherently based off of Wesolowski proofs and cannot utilize the more efficient advice string approach used for re-randomizing Pietrzak proofs. Designing a Pietrzak-style zk-PoKP is an interesting open problem.

Proof Size: The zkVDF proof contains two group elements (a, Q) and the remainder r_2. When using 2048-bit RSA groups and $\lambda = 128$, the total size comes to 529 bytes: 512 bytes for the two group elements and 17 bytes ($\lambda \lg \lambda$) for the value r_2. The Σ-zkVDF construction additionally includes one sub-challenge (the other is implicit), which adds an extra λ bits (16 bytes), making the total overhead for transforming a normal proof into a short-lived one just 545 bytes. The algorithm Σ-zkVDF.Prove requires running the zk-PoKP simulator. The significant operations are raising a group element to a power of size B twice, where $B \approx 2^{2\lambda}|\mathbb{G}|$, and then raising two elements to a power of up to λ twice. In Sect. 9, we evaluate the cost of these operations for 2048-bit RSA groups.

8 Short-lived Signatures

A key special case of zero-knowledge proofs is digital signatures. We define a short-lived signature scheme as follows:

Definition 3 (Short-Lived Signatures). *Let λ be a security parameter and $\mathcal{B}$ be a space of beacon values where $|\mathcal{B}| \geq 2^\lambda$. A short-lived signature scheme with time delay t is a tuple of algorithms:*

- Setup$(\lambda, t) \rightarrow pp$
- KeyGen$(pp) \rightarrow (pk, sk)$
- Sign$(pp, sk, m, b) \rightarrow \sigma$ *takes a message m and beacon b and outputs (in time less than t) a signature σ.*
- Forge$(pp, m, b) \rightarrow \sigma$ *takes a message m and beacon b and outputs (in time less than $(1 + \epsilon)t$) a signature σ.*
- Verify$(pp, pk, m, b, \sigma) \rightarrow$ *Accept/Reject*

The following properties are satisfied:

- **Correctness:** *For all m, $b \in \mathcal{B}$, if $\sigma \leftarrow$ Sign(pp, sk, m, b), then* Verify$(pp, pk, m, b, \sigma) \rightarrow$ *Accept.*
- **Existential Unforgeability:** *For all pairs of adversary algorithms $\mathcal{A}_0$ (precomputation) which runs in total time $O(poly(t, \lambda))$ and $\mathcal{A}_1$ (online) which runs in parallel time $\sigma(t)$ with at most $p(t)$ processors, the probability that $(\mathcal{A}_0, \mathcal{A}_1)$ win the following game is negligible:*

1. *Challenger C runs $pp \leftarrow$* $\mathsf{Setup}(\lambda, t)$ *and* $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$. *$C$ sends pp, pk to $(\mathcal{A}_0, \mathcal{A}_1)$.*
2. *The adversary runs $A_0(pp, pk) \leftrightarrow C$ interactively with the challenger, adaptively sending chosen message/beacon queries (m_i, b_i) to the challenger and receiving $\sigma_i \leftarrow$* $\mathsf{Sign}(pp, sk, b_i, m_i)$ *in response.*
3. *A_0 outputs an advice string α.*
4. *C samples a random beacon value $b \xleftarrow{\$} \mathcal{B}$ and sends it to the adversary.*
5. *The adversary runs $A_1(pp, pk, \alpha, b) \leftrightarrow C$ interactively with the challenger, adaptively sending chosen message/beacon queries (m_i, b_i) to the challenger and receiving $\sigma_i \leftarrow$* $\mathsf{Sign}(pp, sk, b_i, m_i)$ *in response.*
6. *A_1 outputs a claimed forgery (m_*, b, σ_*) and wins if $(m_*, b) \neq (m_i, b_i)$ for all i and* $\mathsf{Verify}(pp, pk, m_*, b, \sigma_*) = \mathit{Accept}$.

- **Indistinguishability:** *For all $m, b \in \mathcal{B}$, given a random $(pk, sk) \leftarrow$* $\mathsf{KeyGen}(pp)$ *the distributions* $\{\mathsf{Sign}(pp, sk, m, b)\}$ *and* $\{\mathsf{Forge}(pp, m, b)\}$ *(taken over the random coins used by each algorithm and randomly generated private key) are computationally (resp. statistically) indistinguishable.*

This definition closely follows our definition of short-lived proofs and standard security properties for signatures. We present a game-based definition for short-lived signature unforgeability, in contrast with our probabilistic soundness definition for short-lived proofs, to more closely match standard unforgeability definitions for signature schemes. The primary distinction is that the second adversary A_1 is required to run in fewer than t steps (otherwise it could simply run the provided Forge algorithm).

Note that while our Indistinguishability definition compares distributions of output, some signature schemes are deterministic(e.g., RSA [68], BLS [19]). In this case, it is necessary that Sign and Forge produce the same exact signature with overwhelming probability.

We observe that our generic constructions in Sect. 5 can be used to transform any signature scheme into short-lived signature scheme by implementing a zero-knowledge proof for knowledge of a signature. Furthermore, our Σ-based constructions in Sect. 6 can also be used for Σ-based signature schemes such as Schnorr [70] or DSA [1,51].

8.1 Construction from Trapdoor VDFs

We present a short-lived signature construction from *trapdoor VDFs* [77] in Protocol 3. Trapdoor VDFs require a trusted setup which yields a secret evaluation key (the trapdoor) enabling efficient evaluation. Normally, this trapdoor represents a security risk if not destroyed. However, we observe that in the case of short-lived signatures, the trapdoor can serve as a signing key. Repeated-squaring VDFs in RSA groups are trapdoor VDFs: the public parameters include an RSA modulus N and the trapdoor is the factors p, q such that $N = p \cdot q$. With the trapdoor, raising an element to any large exponent z (e.g. $z = 2^t$) is efficient, as z can be reduced modulo $\varphi(N) = (p-1)(q-1)$ into an equivalent exponent of size less than N. Note that this trapdoor is equivalent to the private key used for traditional RSA signatures.

Sign-Trapdoor

```
KeyGen
input: $\lambda$, delay parameter $t$
output: key pair $(pk, sk)$

  1. Generate keys
     $(pk, sk) \leftarrow \mathsf{tdVDF.Setup}(\lambda, t)$
```

```
Sign
input: message $m$, beacon value $b$
output: signature $\sigma$

  1. $x = \mathsf{Hash}(m \parallel b)$
  2. $\sigma = (y, \pi) \leftarrow$
     $\mathsf{tdVDF.TrapdoorEval}(sk, x)$
```

```
Forge
input: message $m$, beacon value $b$
output: signature $\sigma$

  1. $x = \mathsf{Hash}(m \parallel b)$
  2. Compute with delay:
     $\sigma = (y, \pi) \leftarrow \mathsf{tdVDF.Eval}(x)$
```

```
Verify
input: message $m$, beacon $b$, $\sigma = (y, \pi)$
output: Accept/Reject

  1. $x = \mathsf{Hash}(m \parallel b)$
  2. Check that $\mathsf{tdVDF.Verify}(pk, x, y, \pi, t)$
```

Protocol 3: Short-Lived Signatures from a trapdoor VDF scheme

Theorem 5 (Short-Lived Signatures from Trapdoor VDFs). *Assuming that* Hash *is a random oracle and* tdVDF *is a trapdoor VDF, Protocol 3 is a short-lived signature scheme.*

Proof. The correctness of this scheme comes from the correctness of the underlying trapdoor VDF. Indistinguishability is trivial as signing and forgery produce the exact same VDF output, given that VDFs are deterministic.

Existential unforgeability comes from the definition of a trapdoor VDF and modeling Hash as a random oracle. Since the challenger chooses b randomly during the existential forgery game after the precomputation of $\mathcal{A}_0$, the value $x_* = \mathsf{Hash}(m_* || b)$ will be randomly distributed for any message m_*. Thus, the online algorithm A_1 must evaluate the VDF on a random input in fewer than t steps. The adversary's ability to query for signatures on chosen pairs (m_i, b_i) is new from the traditional VDF security model. However since each such pair leads to a VDF evaluation by the challenger on $x_i = \mathsf{Hash}(m_i || b_i)$, the adversary can only learn VDF evaluations on a polynomial number of random inputs. This ability could be simulated by $\mathcal{A}_0$ precomputing the VDF on a polynomial number of random inputs and passing the results as part of the advice string α. Thus, any pair $(\mathcal{A}_0, \mathcal{A}_1)$ which make queries could be converted into an equivalent pair $(\mathcal{A}'_0, \mathcal{A}'_1)$ which make no queries but rely on $\mathcal{A}'_0$ to precompute random VDF solutions instead. Winning the signature forgery game with no querying capability is then equivalent to evaluating the VDF on a random input. The VDF security definition states that no suitably bounded algorithms $(\mathcal{A}'_0, \mathcal{A}'_1)$ can do so with non-negligible probability. □

Table 2. Additional costs to transform a standard proof/signature into a short-lived proof/signature. λ is the security parameter, $\langle\mathbb{G}\rangle$ denotes the size of a group element, $\mathsf{exp}_{\mathbb{G}}(e)$ is the cost of raising a group element to a power of size e, and t is the VDF delay parameter. For concrete evaluations, $\lambda = 128$ and $\mathbb{G}$ is a 2048-bit RSA group. The Generic zk-SNARK method was implemented using Groth16 [47]. All evaluations were performed on a 2.3 GHz 8-Core Intel Core i9 laptop with 16 GB memory.

Protocol		Proof Size Overhead		Proving Time Overhead	
zk-SNARKs	(§5)	0 for Groth16		∼60 s	
Σ-Precomp	(§6.2)	$2\langle\mathbb{G}\rangle + \lambda$	528 bytes	$O(T)$ precomputation	
Σ-rrVDF	(§6.3)	$2\langle\mathbb{G}\rangle + \lambda$	528 bytes	$O(T/k)$ precomputation	
Σ-zkVDF	(§7)	$2\langle\mathbb{G}\rangle + 2\lambda$	545 bytes	$2\mathsf{exp}_{\mathbb{G}}(2^{2\lambda}\lvert\mathbb{G}\rvert) + 2\mathsf{exp}_{\mathbb{G}}(2^{\lambda})$	120 ms
Sign-Trapdoor	(§8.1)	$\langle\mathbb{G}\rangle + \lambda$	272 bytes	$\mathsf{exp}_{\mathbb{G}}(2^{\lambda})$	10 ms
Sign-Watermark	(§8.2)	$\langle\mathbb{G}\rangle + \lambda$	272 bytes	$\mathsf{exp}_{\mathbb{G}}(2^{\lambda})$	10 ms

8.2 Construction from Watermarkable VDFs

The construction in Protocol 3 does not offer reusable forgeability, as the Forge evaluates a VDF on a message-dependent value $x = \mathsf{Hash}(m \parallel b)$. We construct an efficient signature scheme (Protocol 5) with reusable forgeability using *watermarkable VDFs* which embed a prover-chosen *watermark* (μ) during proof generation. Watermarkable VDFs were presented informally by Wesolowski [77, §7.2]; we propose a definition capturing the essential security property of *watermark unforgeability* in Appendix B of the full version [7]. The key idea to construct a watermarkable VDF is to embed the watermark into the Fiat-Shamir challenge, computing $\ell \leftarrow \mathsf{HashToPrime}(y \parallel \mu)$ instead of $\ell \leftarrow \mathsf{HashToPrime}(y)$.

Short-lived Signatures from Watermarkable VDFs To build a short-lived signature scheme using a watermarkable VDF, we use the beacon value as the input to the VDF and the message as the watermark. This enables reusable forgeability, as once a forger has computed $y = \mathsf{VDF.Eval}(b)$ for a specific beacon value, along with its associated advice string α, they can sign a new message by computing a new proof using the same advice string. This is equivalent to proof re-randomization, which can be done in significantly fewer than t steps as discussed in Sect. 6.3. We note that reusability is more limited for this signature scheme as the precomputation is specific to an individual user's public key. A single VDF evaluation enables efficient forgery of any statement by a given signer, but will not work between different signers.

9 Implementation and Performance Evaluation

9.1 zk-SNARK Construction

We implement the generic ZK algorithm using zk-SNARKs, which produce succinct non-interactive proofs for large computations. With zk-SNARKs, the statement is represented in a format similar to algebraic circuits and prover efficiency depends on the size of the circuit in gates. Given a base circuit for relation R and an efficient circuit representation of $\mathsf{VDF.Verify}$, it is straightforward to compile a circuit that is the disjunction of the two as outlined in Sect. 5. We implemented a VDF circuit using Wesolowski VDF proofs [77] using a 2048-bit RSA modulus and the "bellman-bignat" library [62].

The total size of the VDF verification circuit is just over 5 million gates. The large size is due to the costly "hash-to-prime" involved in Wesolowski verification. We composed this circuit with an elliptic curve signature verification circuit (acting as the base relation R) of size under 1000 gates. All proofs were generated using the Groth16 construction [47] which produces proofs of constant size around 300 bytes and with verification time under 10 ms. As proofs are constant size and the verification cost is minimal, there is no added overhead on verifiers for short-lived proofs. However, proof generation incurs a significant added cost of around 60 s.

9.2 Σ-based Constructions

Table 2 compares the performance of our algorithms. Our Σ-constructions, which require only a few exponentiations in a group of unknown order, are significantly more efficient than the zk-SNARK method. We evaluated them using Wesolowski proofs in a 2048-bit RSA group, which is conjectured to provide close to $\lambda = 128$ bits of security [9]. We denote the cost of raising a group element to an exponent of magnitude $2^{2\lambda}|\mathbb{G}|$ as $\mathsf{exp}_{\mathbb{G}}(2^{2\lambda}|\mathbb{G}|)$ (this is the value of B in Protocol 4). A single exponentiation takes around 40 ms. The costliest of the Σ-constructions is Σ-zkVDF which takes two $\mathsf{exp}_{\mathbb{G}}(2^{2\lambda}|\mathbb{G}|)$ operations and three $\mathsf{exp}_{\mathbb{G}}(2^{\lambda})$ operations (<10 ms each), leading to a total overhead of under 0.12 s. Section 7.2 contains more details on the size overhead of the Σ-zkVDF construction. Our signature constructions add one or two more group elements to the size of the base proof/signature. With 2048-bit RSA groups, each element is of size $\langle\mathbb{G}\rangle = 256$ bytes.

9.3 Re-randomization Improvements

Three of our constructions (based on re-randomizable VDFs, zero-knowledge VDFs, watermarkable VDFs) can utilize a precomputed advice string α to speed up computation. For Σ-rrVDF, α is used to speed up the Prove algorithm; for Σ-zkVDF and Sign-Watermark, α speeds up $\mathsf{FastForge}$.

In Appendix A.1 of the full version [7], we outline such an advice string of size $O(\sqrt{t})$ for Pietrzak proofs which enables proof computation in $O(\sqrt{t})$ steps. This advice string is applicable to Σ-rrVDF and Sign-Watermark protocols, for which Pietrzak proofs can be used. Table 3 highlights the practical performance of this approach for different delay parameters. We provide numbers for 2048-bit and 1024-bit RSA groups and assume that hardware implementations (e.g.

Table 3. The time taken for re-randomizing a Pietrzak proof on commodity hardware for RSA groups with delay parameters and specialized hardware speed assumptions. The lengths of the advice strings range from 1 MB to 16 MB. The lengths of the proofs range from 2.6 KB to 8.0 KB. The proof verification time is under 50 ms and 150 ms for 1024-bit and 2048-bit RSA groups, respectively.

			Re-Rand Time	
Hardware Speed	Delay	$\log t$	RSA-2048	RSA-1024
2^{25} ops/s	1 min	31	28 s	8 s
	15 min	35	110 s	35 s
2^{30} ops/s	1 min	36	145 s	58 s
	15 min	40	720 s	240 s

FPGAs) for RSA arithmetic can perform up to 2^{25} and 2^{30} operations per second.[7] The improvements achieved by our advice string enable Pietrzak proofs to be rerandomized in minutes with commodity hardware, whereas Wesolowski proofs would still take days with comparably sized advice strings ([77], Section 4.1).

10 Applications

10.1 Deniable Messaging and Email

Deniable messaging protocols aim to ensure that a purported transcript of a secure communication session between Alice and Bob, along with copies of all cryptographic keys used, does not provide convincing evidence of what Alice and Bob actually communicated or (in some cases) that they communicated at all. Generally, a secure messaging (chat) protocol is run between two participants, identified by public keys bound somehow to their real-world identities. When both participants are online, they can use a key agreement protocol to establish an ephemeral shared MAC key for message integrity. Even if the long-term keys are compromised, the transcript could be forged by either party [69], as popularized by Off-the-Record messaging (OTR) [21]. Deliberate publication of the MAC key after the session can extend forgeability to anyone.

Deniability can be extended to offline recipients in a store-and-forward system through non-interactive designated verifier signatures [49] or ring signatures formed between a sender and a recipient [21,69]—however both require prior knowledge of each recipient's public key. Email is a particularly challenging environment, as in addition to being asynchronous and unidirectional the sender cannot assume knowledge of the recipient's public key. OTR's authors believed email is too difficult for an OTR-like protocol [21].

[7] Öztürk [78] reported a speed for $\sim 2^{24}$ squarings/second in 2019 in an optimized FPGA implementation used to break the RSA LCS-35 timelock puzzle [67].

Our work suggests a different (and complementary) approach to deniability: a sender with an identifiable public key can provide a short-lived signature on their messages. Recipients within the validity period of the signature can validate the message's authenticity, while the message becomes indistinguishable from a forgery after a period of time and therefore deniable by the original sender. Short-lived messages do not require any knowledge about the recipient, interaction, or follow-up steps, making them very versatile. They can be broadcast asynchronously to a group of unidentified recipients with a single communication, and even forwarded with no additional cryptographic effort, making them suitable for email as well as messaging protocols.

10.2 Deniable Domain Authentication

A specific case of deniable authentication arises with the DomainKeys Identified Mail (DKIM) standard for email. Originally proposed to address email forgeries and spam, DKIM requires that the sender's mail server sign every outbound email with a domain-bound key. For example, all email originating from the mail server for `example.com` would be signed with a key bound to the DNS record of `example.com`, however (unlike the use case above) the signature will not distinguish between mail from `alice@example.com` and `bob@example.com`. By 2015, DKIM headers were present in 83% of all inbound mail to Gmail [34].

Over the past two decades, email dumps—the public release of private email messages from breached servers—have received extensive news coverage [27]. DKIM signatures increase the value of email dumps by certifying their authenticity. The call to periodically release past DKIM private signing keys was popularized by Matthew Green [45]. DKIM signatures do not require validity beyond the network latency of reaching a recipient's mail server.

Specter et al. proposed KeyForge and TimeForge [72] to replace DKIM with *Forward Forgeable Signatures (FFS)* that become *non-attributable* after a specified time (e.g., 15 min). Both KeyForge and TimeForge require future action to ensure deniability: respectively, a secret value released by the signer, or a future signed update to a beacon-like service called a *publicly verifiable timekeeper*. If the time-keeper's private key is lost then all signatures become permanently attributable. Alternately, if the time-keeper is silently compromised then signatures are immediately forgeable. Short-lived signatures can fulfill the same role as a drop-in replacement for DKIM, while requiring no follow-up action by *anyone* and hence deniability is guaranteed at the time of signing. Both TimeForge and short-live signatures expand the current length of a 2048-bit RSA DKIM signature. Our trapdoor RSA-based short-lived signature adds a single group element (200% expansion) while TimeForge signatures expand by 329% [72].

TimeForge also has advantages: no costly VDFs need to be evaluated (or threatened) to provide deniability and the timing of deniability is precise, whereas for short-lived signatures the deniability time period depends on how fast VDFs can be evaluated. Our approaches are complementary: a signature could be both short-lived and forgeable after the release of information as in TimeForge, attaining the advantages of both.

10.3 Receipt-Free Voting

Numerous cryptographic voting protocols involve encoding a voter's selection with an additively homomorphic encryption scheme. A voter wants *ballot casting assurance* [14] that a posted ciphertext decrypts to her choice, however she should not be able to transfer this assurance to anyone else. As the literature moves toward a more realistic view of voters as humans casting ballots at polling places, vote casting needs to be accessible and bare-handed (i.e., no assumption of an additional device at casting time). The dominant approach (exemplified in Helios [5], STAR-Vote [11], and Microsoft's ElectionGuard [3]) is the *Benaloh challenge* [14]: (1) a voter asks for an encryption of a candidate s_i, (2) the voting machine commits (e.g., on paper) a ciphertext c, (3) the voter chooses to audit the ballot or cast it, and (4) if auditing, the voting machine produces the plaintext and randomness (s_i, r) such that $c = \mathsf{Enc}(s_i, r)$; and the voter restarts at (1). Later, aided by a computer, the voter validates all transcripts. This protocol has two drawbacks. If a voter asks for an encryption of candidate Bob, receives one for candidate Alice instead, audits it and receives a proof for Alice, the voter is convinced the machine is malicious (she knows she asked for Bob) but the transcript will not convince a third party that the machine misbehaved. The second drawback is that auditing is probabilistic (and a low audit frequency is observable by the machine itself).

Alternatives to the Benaloh challenge mitigate these drawbacks but add complexity for the voter. A collection of techniques [49,59,60] use quite different protocols to produce a similar outcome: the voter leaves with a receipt that contains the ciphertext c and n proofs that $c = \mathsf{Enc}(s_i)$ for each of the n candidates. One proof is real and the rest are forgeries, but the transcript does not reveal which one is real. These protocols vary, but at a high level, either the machine prepares the forgeries and the voter releases a value (e.g., a challenge) for construction of the real proof; or the machine prepares the real proof, and the voter releases a value (e.g., a trapdoor or private key) for the forgeries.

A short-lived proof can be used in this second paradigm to eliminate all the pre-constructed values (i.e., challenges, keys, trapdoors) the voter must bring into the polling place, replacing them with a simple clock. The voter experience is as follows: (1) a voter selects candidate s_i, (2) the voting machine commits (e.g., on paper) the time T, the name of s_i (in plaintext), a ciphertext c, and a short-lived (e.g., 60 s) proof that $c = \mathsf{Enc}(s_i)$, (3) the voter checks that T and s_i are correct, (4) after two minutes, the machine (possibly a different machine at a different station within the polling place) produces $n-1$ forged receipts for each leftover s_i with the same c and same (and now outdated) T.

Commonly used encryption schemes for voting, like exponential Elgamal and Paillier, have efficient Σ-protocol proofs of plaintext values and be adapted to use any of our Σ-protocol-based short-lived proof constructions. A time parameter of 60 s provides reasonable assurance if the beacon and voter's clock are synchronized to within one second, while not delaying the time to vote substantially. Voters could choose to shred their initial receipt or wait for a set of forged receipts. Attention is required to mitigate side-channel information (like forensics of the paper) to infer the order in which the proofs are printed.

11 Concluding Remarks

We observe that the existence of the Forge algorithm for short-lived proofs circumvents Pass' observation [63] that non-interactive zero-knowledge proofs in the random oracle model are not deniable. Normally, because the simulator requires programmable access to the random oracle, verifiers cannot simulate proofs and hence possession of a proof demonstrates interaction with a genuine prover. In our case, the Forge algorithm does not require programmable random oracle access, only the ability to compute a slow function. Short-lived proofs therefore can offer deniability as they can be forged with no special ability except time.

In practice, this is an important limitation if the time taken to compute a proof is known and is insufficient to produce a forgery. For example, if a short-lived proof π is convincingly timestamped at time T_1 and the beacon value b used to compute π was not known until time T_0, with $T_1 - T_0 < t$, then π could not have been computed via Forge. Thus, deniability for short-lived proofs relies on the assumption that it is not feasible to convincingly timestamp all data. For example, in the case of deniable DKIM signatures, it must be the case that signed emails are not routinely timestamped en masse. We note that other solutions to this problem, including key expiry/rollover or the KeyForge/TimeForge schemes of Specter et al. [72], have the exact same limitation.

Recall that we offer constructions for proofs and signatures based on Σ-protocols, where deniability is added by combining the original statement with a VDF-related statement in a disjunction. As noted in Sect. 3, designated verifier proofs, proofs of work-or-knowledge, and KeyForge/TimeForge also add deniability via disjunction with a second statement. It is straightforward to combine these approaches. For example, a statement could be proven in zero knowledge to be true only if the proof is received by a specific recipient before a signed timekeeper statement is released or a VDF could have been computed. In this way, a proof can gain deniability in the absence of any trusted third party action in the future (as with short-lived proofs) while also gaining deniability without requiring anybody to solve a VDF if the third party acts faithfully.

We conclude with several open problems arising from our work:

- Short lived proofs require *someone* to evaluate a VDF. It might be possible to piggyback off of an existing party computing VDFs, such as a computational time-stamping service [54], some other party computing a long-running continuous VDF [36] or a decentralized protocol using chained VDFs [29,40].
- Our Σ-zkVDF construction (§7) only provides 1-reusable forgeability. It might be possible to extend this to k-reusable forgeability using an RSA-style accumulator to combine past beacon values (while keeping the accumulator value secret to avoid undermining deniability).

- Our zk-PoKP construction in Protocol 2 is based on Wesolowski proofs. Constructing a zk-PoKP algorithm based on Pietrzak proofs would allow a Σ-zkVDF forger to leverage the more efficient re-randomization algorithm for Pietrzak proofs, enabling significantly faster forging times.
- While our watermarkable VDF signature construction (§8.2) offers reusable forgeability, it requires a VDF computation per-signer (as each signer uses unique public parameters). An ideal scheme might use similar accumulator techniques to forge proofs from a set of signers after just one VDF evaluation.
- Another, potentially much more efficient, approach to achieve generic short-lived proofs is to take an existing generic proof scheme which relies on a Σ-protocol and replace that component with a short-lived equivalent (e.g. our Σ-zkVDF construction). While this would not retain the k-reusability of our generic approach in §5, it would avoid the cost of verifying a VDF within the proof system itself. This approach potentially applies to many popular zero-knowledge proof systems, including Bulletproofs [22], Marlin [24], PLONK [41], Sonic [58], Spartan [71], Supersonic [23], or STARKs [12].
- Another approach to obtain short-lived SNARKs is through composition techniques that construct zero-knowledge proofs of disjunctions of SNARKs with Σ-protocols (zkVDFs in our case) as proposed by Agrawal et al. [6]. This would avoid the costly encoding of VDFs as arithmetic circuits (which require millions of gates) leading to interesting tradeoffs between shorter proving times and larger proof sizes.

Acknowledgments. We give a special acknowledgement to Michael Colburn who initially pursued the idea of short-lived signatures and proofs in his MASc thesis [30] supervised by Jeremy Clark. The constructions in this paper are new, and any ideas or text from [30] incorporated into this paper were originated by Clark. We thank Benjamin Wesolowski, Justin Thaler, Riad Wahby, Benedikt Bünz, Ben Fisch, Dan Boneh, Matthew Green, Michael Specter, and Michael Walfish and for helpful feedback and discussion. We thank Alex Ozdemir for help with implementing our SNARK-based construction.

Arasu Arun and Joseph Bonneau were supported by DARPA under Agreement No. HR00112020022. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government or DARPA.

Jeremy Clark acknowledges support for this research project from (i) the National Sciences and Engineering Research Council (NSERC), Raymond Chabot Grant Thornton, and Catallaxy Industrial Research Chair in Blockchain Technologies, and (ii) NSERC through a Discovery Grant.

References

1. The digital signature standard: communications of the ACM **35**(7), 36–40 (1992)
2. Drand Randomness Beacon. drand.love (2021)
3. ElectionGuard. https://github.com/microsoft/electionguard (2021)
4. NIST Randomness Beacon Version 2.0. https://beacon.nist.gov/home (2021)
5. Adida, B.: Helios: Web-based Open-audit Voting. In: USENIX Security (2008)

6. Agrawal, S., Ganesh, C., Mohassel, P.: Non-interactive zero-knowledge proofs for composite statements. In: CRYPTO (2018)
7. Arun, A., Bonneau, J., Clark, J.: Short-lived zero-knowledge proofs and signatures. Cryptology ePrint Archive, Paper 2022/190 (2022)
8. Baldimtsi, F., Kiayias, A., Zacharias, T., Zhang, B.: Indistinguishable Proofs of Work or Knowledge. In: Eurocrypt (2016)
9. Barker, E., Dang, Q.: Recommendation for Key Management. NIST Special Publication 800–857 (2015)
10. Beck, G., Choudhuri, A.R., Green, M., Jain, A., Tiwari, P.R.: Time-deniable signatures. Cryptology ePrint Archive, Paper 2022/1018 (2022)
11. Bell, S., et al.: STAR-Vote: a secure, transparent, auditable, and reliable voting system. In: JETS (2013)
12. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/46 (2018)
13. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: CRYPTO (2013)
14. Benaloh, J.: Ballot casting assurance via voter-initiated poll station auditing. In: EVT (2007)
15. Boneh, D., Naor, M.: Timed commitments. In: CRYPTO (2000)
16. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (2018)
17. Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712 (2018)
18. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: CRYPTO (2019)
19. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Eurocrypt (2001)
20. Bonneau, J., Clark, J., Goldfeder, S.: On Bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015 (2015)
21. Borisov, N., Goldberg, I., Brewer, E.: Off-the-record communication, or, why not to use PGP. In: ACM WPES (2004)
22. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: IEEE Security & Privacy (2018)
23. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: CRYPTO (2020)
24. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: CRYPTO (2020)
25. Ciampi, M., Persiano, G., Scafuro, A., Siniscalchi, L., Visconti, I.: Improved OR Composition of Sigma-Protocols. In: TCC (2016)
26. Clark, J., Hengartner, U.: On the use of financial data as a random beacon. In: EVT/WOTE (2010)
27. Clark, J., van Oorschot, P.C., Ruoti, S., Seamons, K., Zappala, D.: SoK: securing email: a stakeholder-based analysis. In: Financial Cryptography (2021)
28. Cohen, B., Pietrzak, K.: Simple Proofs of Sequential Work. In: CRYPTO (2018)
29. Cohen, B., Pietrzak, K.: The Chia network blockchain (2019)
30. Colburn, M.: Short-lived signatures. Master's thesis, Concordia University (2018)
31. Couteau, G., Klooß, M., Lin, H., Reichle, M.: Efficient range proofs with transparent setup from bounded integer commitments. In: Eurocrypt (2021)

32. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: CRYPTO (1994)
33. Dodis, Y., Yum, D.H.: Time capsule signature. In: Financial Cryptography (2005)
34. Durumeric, Z., et al.: Neither snow nor rain nor MITM: an empirical analysis of email delivery security. In: ACM CCS (2015)
35. Dwork, C., Naor, M.: Pricing via Processing or Combatting Junk Mail. In: CRYPTO (1992)
36. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: CRYPTO (2020)
37. Feo, L.D., Masson, S., Petit, C., Sanso, A.: Verifiable Delay Functions from Supersingular Isogenies and Pairings. Cryptology ePrint Archive, Report 2019/166 (2019)
38. Ferradi, H., Géraud, R., Naccache, D.: slow motion zero knowledge identifying with colliding commitments. In: ICISC (2015)
39. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
40. Foundation, E.: Ethereum 2.0 Beacon Chain (2020)
41. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019)
42. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: CRYPTO (2013)
43. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. JACM 38(3) (1991)
44. Goyal, V., Raizes, J., Soni, P.: Time-traveling simulators using blockchains and their applications. Cryptology ePrint Archive, Paper 2022/035 (2022)
45. Green, M.: Ok Google: please publish your DKIM secret keys (November 2020)
46. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Eurocrypt (2010)
47. Groth, J.: On the size of pairing-based non-interactive arguments. In: CRYPTO (2016)
48. Halderman, J.A., Waters, B.: Harvesting verifiable challenges from oblivious online sources. In: CCS (2007)
49. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: Eurocrypt (1996)
50. Jaques, S., Montgomery, H., Roy, A.: Time-release cryptography from minimal circuit assumptions. Cryptology ePrint Archive, Report 2020/755 (2020)
51. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. 1(1) (2001)
52. Krawczyk, H., Rabin, T.: Chameleon signatures. In: NDSS (2000)
53. Kucherawy, M., Crocker, D., Hansen, T.: DomainKeys identified mail (DKIM) signatures. RFC 6376 (2011)
54. Landerreche, E., Stevens, M., Schaffner, C.: Non-interactive cryptographic timestamping based on verifiable delay functions. In: Financial Cryptography (2020)
55. Lenstra, A.K., Wesolowski, B.: Trustworthy public randomness with sloth, unicorn, and trx. Int. J. Appl. Crypto. **3**(4), 330–343 (2017)
56. Mahmoody, M., Moran, T., Vadhan, S.: Publicly verifiable proofs of sequential work. In: ITCS (2013)
57. Malavolta, G., Thyagarajan, S.A.K.: homomorphic time-lock puzzles and applications. In: CRYPTO (2019)

58. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: ACM CCS (2019)
59. Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: CRYPTO (2006)
60. Neff, C.A.: Practical high certainty intent verification for encrypted votes. Tech. rep, VoteHere Whitepaper (2004)
61. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: Eurocrypt (1998)
62. Ozdemir, A., Wahby, R., Whitehat, B., Boneh, D.: Scaling verifiable computation using efficient set accumulators. In: USENIX Security (2020)
63. Pass, R.: On deniability in the common reference string and random oracle model. In: CRYPTO (2003)
64. Pietrzak, K.: Simple verifiable delay functions. In: ITCS (2018)
65. Rabin, M.: Transaction protection by Beacons. J. Comput. Syst. Sci. 27(2) (1983)
66. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. Rep. TR-684, MIT (1996)
67. Rivest, R.L.: Description of the LCS35 time capsule crypto-puzzle. https://people.csail.mit.edu/rivest/lcs35-puzzle-description.txt (1999)
68. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2) (1978)
69. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Asiacrypt (2001)
70. Schnorr, C.P.: Efficient signature generation by smart cards. J. Crypto. **4**(3), 161–174 (1991)
71. Setty, S.: Spartan: efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO (2020)
72. Specter, M.A., Park, S., Green, M.: KeyForge: non-attributable email from forward-forgeable signatures. In: USENIX Security (2021)
73. Syta, E., et al.: Scalable bias-resistant distributed randomness. In: IEEE Security & Privacy (2017)
74. Thaler, J.: Proofs, arguments, and zero-knowledge (2021)
75. Thyagarajan, S.A.K., Bhat, A., Malavolta, G., Döttling, N., Kate, A., Schröder, D.: Verifiable timed signatures made practical. In: ACM CCS (2020)
76. Wesolowski, B.: A proof of time or knowledge. https://hal.archives-ouvertes.fr/hal-03380471
77. Wesolowski, B.: Efficient verifiable delay functions. In: Eurocrypt (2019)
78. Öztürk, E.: Modular multiplication algorithm suitable for low-latency circuit implementations. Cryptology ePrint Archive, Paper 2019/826 (2019)