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
[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 formal 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. Our work also introduces two novel types of VDFs, re-randomizable VDFs and zero-knowledge VDFs, which may be of independent interest.

1 Introduction

A digital signature is forever. Or at least, until the underlying signature scheme is broken or the signature key is breached. This is often in contrast to what is strictly required in 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. When cryptographic proof of past actions lives on for decades, it violates the principle of least privilege. 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 Domain Keys Identified Mail (DKIM) protocol [61] 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 [32]. 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 [81]. 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 [51].

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) [18,87,72]. 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, as suggested for DKIM, is to publish secret information after running a protocol which enables any party to forge transcripts, undermining them as definitive evidence. Other approaches include deniable key exchange protocols (e.g., OTR messaging [23]) or designated verifier proofs/signatures [56] 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 Section 10. An example is sending signed email to a large group of individuals who do not have known public keys using a single signature. To our knowledge, ours is the first primitive to enable this.

To summarize our contributions, as outlined in Table 1:

- We introduce the notion of short-lived zero-knowledge proofs (§4), with short-lived signatures [35] as a special case (§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).
- 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 eliminate by introducing the concept of re-randomizable VDFs (§A).

Scheme	Section	Reusable forgeability / No pre-computation	VDF required	Proof/Sign type
		short-lived proofs		
Generic ZK	§5	● ●	any [18]	generic SNARK
Σ-Precomp	§6.2	○ ○	any [18]	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	§8.1	● ●	any [18]	generic SNARK
$\Sigma_{\text{sign}} \vee$ zkVDF	§8.1	◐ ●	zero-knowledge (§7.1, §C)	Σ signatures
Sign-Trapdoor	§8.2	○ ●	trapdoor ([87,72])	RSA
Sign-Watermark	§8.3	◐ ●	watermarkable ([87], §B)	RSA

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

- 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 (§8.1). We further introduce short-lived signature schemes (§8.2) from trapdoor VDFs [87] and watermarkable VDFs which are as compact as a single VDF proof and offer reusable forgeability.

2 Background and Related Work

2.1 Zero knowledge proofs and arguments

We start with a basic background on zero-knowledge proofs, while referring the reader to [83] 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 a Diffie-Hellman tuple, 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. If there is a polynomial-time algorithm to verify that $(x, w) \in R$, then L_R is an NP language. It has been shown that all NP-languages have a zero-knowledge proof system [50]. 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):

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

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 $(x, w) \in R$. 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 an efficient algorithm $\mathsf{Simulate}$ which, given any statement $x \in L_R$ and the ability to *program* the random oracle to give specified responses, can output simulated proofs $\tilde{\pi}$ indistinguishable from real proofs such that $\mathsf{Verify}(pp, x, \tilde{\pi})$ accepts.

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 [52,49,11].

2.2 Sigma Protocols

Outside of Section 5, our constructions utilize a special subclass of interactive zero-knowledge proof systems, called Σ-protocols [79]. A Σ-protocol [79] is a three-move interactive protocol between P and V that realizes a partially secure proof of knowledge which is called *honest verifier zero-knowledge*.

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 when 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 $\Sigma.\mathsf{Verify}(x, \tilde{a}, \tilde{c}, \tilde{z})$

accepts and the distribution of $(x, \tilde{a}, \tilde{c}, \tilde{z})$ is indistinguishable from transcripts of a genuine interaction between a verifier 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 [46], in which the challenge is generated as $c = \mathcal{O}(x, a)$. $\Sigma.\mathsf{Extract}$ 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 [37]. The classic protocol for disjunction of Σ-protocols, which we denote Σ-OR, is due to Cramer et al. [37].[4] 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 and 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 $\Sigma_{R_2}.\mathsf{Simulate}(x_2) \to (a_2, c_2, z_2)$
2. P runs $\Sigma_{R_1}.\mathsf{Commit}(x_1) \to a_1$
3. P send (a_1, a_2) to V
4. V sends $c = \Sigma.\mathsf{Challenge}()$ to P
5. P sets $c_1 = c \oplus c_2$
6. P runs $\Sigma_{R_1}.\mathsf{Respond}(x_1, w_1, a_1, c_1) \to z_1$
7. P sends (c_1, c_2, z_1, z_2) to V
8. V accepts if:
   - $c = c_1 \oplus c_2$
   - $\Sigma_{R_1}.\mathsf{Verify}(x_1, a_1, c_1, z_1)$ accepts
   - $\Sigma_{R_2}.\mathsf{Verify}(x_2, a_2, c_2, z_2)$ accepts

The core idea is that the prover is able to choose the challenge to one of the two Σ-protocols, but this induces a pseudorandom challenge for the other protocol which the prover must then be able to genuinely prove. We will use this construction in Section 7 and introduce a similar (but slightly different) construction in Section 6.2.

2.4 Beacons

A beacon [73] 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. 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.

[4] Ciampi et al. [30] later introduced a different Σ-OR protocol with certain advantages over the Cramer et al. construction.

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 seconds [2] using a multi-party randomness protocol [82]. Other potential beacons that have been analyzed include web server challenges [54], stock market prices [31], lottery draws [6] and blockchain data [22].

2.5 Verifiable Delay Functions

Verifiable delay functions (VDFs) are defined by a trio of algorithms:[5]

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

Boneh et al. first formalized VDFs and offer precise security definitions [18]. Informally, VDFs satisfy three important properties. VDFs must be verifiable, meaning that the verification algorithm is efficient (at most polylogarithmic in t and λ) and always accepts when given output from the genuine Eval algorithm.

Most importantly, 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. More precisely, VDF security is defined by a property called σ, p-sequentiality against an online adversary able to run for $\sigma(t)$ steps using up to $p(t)$ parallel processors [18]. The adversary may use an advice string from a precomputation algorithm that might run for much longer, i.e., $O(poly(t, \lambda))$. The exact values of σ and p depend on the underlying construction, as does the notion of a "step" of computation (typically squaring an element in a finite group or evaluating a one-way function). All of our constructions reduce forging a proof to evaluating a VDF on a random input. Our definition of t-Forgeability (§4) maps directly to the sequentiality definition for VDFs.

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. Our constructions don't rely on this property and we could instead use the weaker notion of a proof-of-sequential work (PoSW) [64,33]. 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.

VDFs constructions have been proposed from generic succinct proofs [18], repeated squaring in groups of unknown order [87,72], permutation polynomials [18], isogenies [44], and homomorphic encryption [57]. Earlier work proposed "weak" VDFs based on computing square roots mod p [41,63].

An important limitation of all VDF constructions is that they can only guarantee a certain number of steps of sequential computation are required. The

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

real-world or "wall-clock" time needed to execute this computation varies based on the speed of available computing platforms. In our work, this means that the period of time after which proofs become forgeable varies. 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.

2.6 VDFs from repeated squaring

We focus in particular on VDF constructions which utilize repeated squaring in a group of unknown order as the core sequential computation, as these have useful algebraic properties for building short-lived proofs and signatures. VDF evaluation is simply $y = \mathsf{VDF.Eval}(b) = b^{2^t}$. Note that the computation is $b^{(2^t)}$ (requiring t sequential squarings) and not $(b^2)^t$ (requiring only $\log t$ squarings); to avoid clutter we will omit parenthesis around the exponent 2^t which will appear frequently. Wesolowski [87] and Pietrzak [72] introduced two distinct approaches for efficiently proving that $y = b^{2^t}$ in a group of unknown order:

Wesolowski proofs Wesolowski proofs [87] work as follows. 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 $\tilde{y} = q^\ell b^r$. Note that this protocol is itself a Σ-protocol.

Pietrzak proofs Pietrzak proofs [72] are generated through a multi-round protocol. 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 then 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 send the value r to the prover, who can recursively prove that $\tilde{y}' = (b')^{(2^{t/2})}$. Typically, this is done for d rounds, each reducing the size of the exponent in half, until the verifier manually checks the remaining exponent of size $2^{t/2^d}$. A single round of Pietrzak is a Σ-protocol, the recursive version of the proof is a multi-round generalization.

Comparison Boneh et al. [19] 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). We also discuss watermarking (§8.3) which is possible for both proofs.

Concrete groups In this work, we will refer to an abstract group of unknown order $\mathbb{G}$ and assume some algorithm $\mathsf{GroupGen}$ which randomly samples suitably sized groups. The best-known family of groups of unknown order (often called *RSA groups*) is the multiplicative group $(\mathbb{Z}/N)^*$ of integers modulo a composite $N = pq$. Typically, N is chosen to be a product of two *safe primes* p, q, that is, $p = 2p' + 1, q = 2q' + 1$ for smaller primes p', q'. It is important to note that $(\mathbb{Z}/N)^*$ itself should not be used directly for VDFs, because the low-order assumption does not hold. Boneh et al. [19] suggest instead using the group $\mathbb{G}^+ = (\mathbb{Z}/N)^*/\{\pm 1\}$.

With any RSA group, the factors p, q act as a *trapdoor* enabling fast VDF evaluation (by reducing the exponent 2^t modulo $\varphi(N) = (p-1)(q-1)$). Hence this construction is described as a trapdoor VDF [87]. In our signature application, the trapdoor is in fact necessary (§8.2) while in other cases, the existence of the trapdoor is a risk. N can be computed via a trusted setup which deletes the prime factors, using a secure multi-party computation [55] or by choosing using a sufficiently large random modulus [77].

Alternately, class groups of imaginary quadratic fields [24] have been proposed [87,19] as a family of groups of unknown order which can be sampled efficiently without a trusted setup. However, secure parameter choices and optimized implementations are less well understood for class groups.

3 Related Work

Jakobsson, Sako and Impagliazzo originate the idea of using disjunctive statements to provide deniability [56]. Given a statement x to be proven to Bob in zero knowledge, the proof is transformed into the statement: {*either x or I know the 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* [60], which use a standard hash-and-sign construction but with a chameleon hash function whose trapdoor is known by the intended verifier, providing deniability if the verifier attempts to transfer the signature to another party.

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 [7]. Specter et al. propose {*either x or someone has seen value v released at time T*} for a v to be published at a future time T [81].

Many of our constructions follow the same disjunctive template, with the essential statement being {*either x or someone solved a VDF on a beacon value derived from b which was unknown before time T*}. VDFs requires t sequential

steps (which approximate elapsed time much more reliably than a parallelizable proof of work) and does not rely on future action for the second clause to be true. Of these related approaches, the Specter et al. KeyForge and TimeForge protocols are the closest in purpose to our own work—we discussed them deeper in Section 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. [45] 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.

An initial version of this work appeared as a Masters thesis in 2018 by Colburn [35] who was supervised with the third author of this paper (Clark). The thesis was written before the development of VDFs (using proof-of-work instead), and thus all of the constructions in this paper are new. However we do describe common applications of short-lived proofs and signatures, including the voting protocol in Section 10.3 which was originally proposed to Colburn by Clark.

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 [87, §8]. Interestingly, Wesolowski developed a time-limited signature protocol in 2016 which improved on the Ferradi et al. construction [86], which evolved into his seminal VDF paper. Sadly, Wesolowski's manuscript remained unpublished until the time of our own work. Wesolowski's protocol is essentially the same as our proposed Sign-Trapdoor construction (§8.2). Our version embeds a beacon value, making it transferable, non-interactive, and an actual signature (rather than an authentication protocol). While a simple change, the use of beacons are consequential. In fact, beacons can be used adversarially against his identification scheme—the scheme is still secure but requires new security bounds (see Appendix D).

Other time-based cryptographic primitives have been proposed including encryption, commitments, and signatures [74,17,84]. In the context of the cited literature, a *timed signature* [17] 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* [39] which become valid after a certain period of time when a time-server releases some information. A proof is offered that the commitment is a well-formed signature on a message, and recent work [84] has proposed new efficient constructions of the required proof. We are essentially solving the inverse problem: instead of a signature being hidden for time Δt and then unforgeable for the rest of time, a short-lived signature is unforgeable for Δt and then deniable for the rest of time.

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* (Setup, Prove, Forge, 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
- $\mathsf{Verify}(pp, x, \pi, b) \rightarrow (Accept, Reject)$

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*

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

  then there exists an algorithm Extract *with programmable access to the random oracle $\mathcal{O}$ and 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.*
- **Zero Knowledge:** *There exists an algorithm* Simulate *with programmable access to the random oracle $\mathcal{O}$ which runs in total time less than t such that given pp, for all $(x, w) \in R$ and $b \in \mathcal{B}$ the distributions $\{\mathsf{Simulate}(pp, x, b)\}$ and $\{\mathsf{Prove}(pp, x, w, b)\}$ (taken over the random coins used by each algorithm) are computationally (resp. statistically) indistinguishable.*
- **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, closely follows that used in defining security for VDFs [18]. 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).

The definitions of indistinguishability and zero-knowledge are closely related: the Forge algorithm is able to (slowly) produce proofs indistinguishable from those of the genuine Prove algorithm whereas Simulate produces indistinguishable proofs in fewer than t steps (but requires programmable random oracle access). In a sense, short-lived proof schemes are zero-knowledge without a simulator because the Forge algorithm is able to produce valid proofs without knowing a witness. However, it is important to require an efficient simulator (with respect to t) to be sure that the zero-knowledge property holds even if a distinguisher can measure the time used to create proofs.

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 of any individual proof. Some short-lived proof schemes 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.

For simplicity of presentation we denote the special case of 1-reusable forgeability as simply reusable forgeability. We provide a definition for k-reusable forgeability which allows an arbitrary set of size at most k. All of our schemes

either provide k-reusable forgeability for arbitrary k or are limited to 1-reusable forgeability.

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

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

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

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 proof 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. The completeness of Π_R^t is due to the completeness of Π_R. 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 half of the disjunction. The Indistinguishability and Zero-Knowledge properties both follow immediately 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 t-Soundness property relies on the t-Sequentiality 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 $\mathsf{VDF.Eval}(b)$ is required for proving any statement x. Thus, it can be computed once and reused across proofs. The Indistinguishability II property is ensured by the zero-knowledge property of the underlying proof scheme. □

5.1 Extending to k-reusable forgeability

The basic scheme offers only 1-reusable forgeability, as a VDF evaluation on a specific beacon value is required. We briefly outline a construction which provides k-reusable forgeability. This construction relies on a one-way accumulator scheme [12] which we denote Acc. As above, our scheme computes a proof on the disjunction of the original relation R with a new relation $R_{\mathrm{VDF}-k}$:

$$R_{\mathrm{VDF}-k} = \{ (x = b, w = (b^*, y^*, \pi^*, A, w_b)) \mid \mathsf{VDF.Verify}(b^*, y^*, \pi^*) \wedge b^* = \mathsf{Hash}(A) \wedge \mathsf{Acc.VerifyMembership}(A, b, w_b) \} \quad (2)$$

The core idea is that $\mathsf{GenAdvice}$ combines the set of past beacon values B into an accumulator value $A = \mathsf{Acc.AccumulateSet}(B)$ and evaluates a VDF on the value $b^* = \mathsf{Hash}(A)$. The advice string α output by $\mathsf{GenAdvice}$ includes the VDF solution (y^*, π^*) as well as the accumulator value A (and potentially some additional data to facilitate computing inclusion proofs). The $\mathsf{FastForge}$ algorithm can then satisfy $R_{\mathrm{VDF}-k}$ for any included beacon value b by providing the precomputed VDF solution (y^*, π^*) and an accumulator proof w_b showing that b is included in A. Computing w_b is the only required work per proof. In practice, this can be implemented efficiently using a classic Merkle Tree as the accumulator (using a circuit-friendly hash function), leading to membership proofs which are logarithmic in k (and independent of the delay parameter t).

The generic proof system's underlying zero-knowledge property ensures forgeries are indistinguishable from genuine proofs and furthermore that nothing

about the set B is revealed. Soundness is maintained because b^* depends on the entire set B and hence $\mathsf{GenAdvice}$ cannot be run until every beacon value $b \in B$ in the set is known. Because $\mathsf{GenAdvice}$ solves the VDF, it will not finish in fewer than t steps and hence cannot be used to win the t-Soundness game.

Proving security for this scheme requires an additional property of accumulators, namely that computing the accumulator value requires work linear in size of the accumulated set. Otherwise an attack would exist on soundness in which $\mathsf{GenAdvice}$ is precomputed for a large set $|B_{\text{large}}| = o(\mathsf{polylog}(\lambda))$, giving a non-negligible probability that a randomly chosen b is in B_{large}. While this is not a standard assumption about accumulators, it is true for common constructions (e.g., Merkle trees or RSA accumulators [12]).

5.2 Supporting variable delay parameters

Some VDF constructions (including those based on repeated squaring) do not require hardcoding t at the time Setup is called. Instead, any value of t can be passed as a parameter to Prove and Verify. In this case, we can extend the notion of reusability such that one VDF evaluation with any delay $t' \geq t$ allows forging a proof for with delay parameter t. This enhances deniability if different provers use different values for t. This is straightforward with the generic construction, simply adding an extra parameter to the VDF relation:

$$R_{\text{VDF}_{\geq t}} = \{(x = (b, t); w = (y, \pi, t')) | \mathsf{VDF.Verify}(b, y, \pi, t')) \wedge t' \geq t\} \quad (3)$$

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 [87] 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 Section 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

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

statement. This means that the algorithm 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 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). *Protocol 1 is a short-lived proof scheme.*

Proof. The completeness and t-forgeability of Σ-Precomp follow directly from the completeness of Σ_R and correctness of the VDF. Indistinguishability similarly follows from the underlying schemes. In the output of Prove, the challenge $c_2 = b^* \oplus b$ will be randomly distributed as both b and b^* are random by definition. This means c_1 will similarly be random, as $c_1 = c \oplus c_2$ where c is the result of a random oracle and c_2 is randomly distributed. In the output of Forge, the zero-knowledge property of Σ_R implies that the values $(\tilde{a}, \tilde{c}_1, \tilde{z})$ output by Simulate are indistinguishable from the genuine values (a, c_1, z) output by Prove (with c_1 being randomly distributed). This further implies that the value c_2 output by Forge will be randomly distributed, since $c_2 = c \oplus \tilde{c}_1$, c is the result of a random oracle and $\tilde{c}_1$ must be randomly distributed to be indistinguishable from c_1. Finally, the values y, π_{VDF} are determined uniquely by $b \oplus c_2$ as VDF.Eval is a deterministic function and hence can add no new distinguishing information.

The Zero-Knowledge property can be shown by construction. The simulator first calls $\Sigma_R.\mathsf{Simulate}(x)$ to generate a sub-proof $(\tilde{a}, \tilde{c}_1, \tilde{z})$. The simulator then programs the random oracle such that $\mathcal{O}(x \parallel b \parallel \tilde{a}) \to b^* \oplus \tilde{c}_1$, where b^* is a value for which the VDF solution $(y^*, \pi^*_{\mathrm{VDF}})$ is known, allowing the simulator to complete the proof quickly.

Finally, to demonstrate 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

Σ-Precomp

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

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

Prove
input: statement x, witness: w, beacon value b, precomputed $(b^*, y^*, \pi^*_{\mathrm{VDF}})$
output: proof $(a, c_1, z, y^*, \pi^*_{\mathrm{VDF}}, c_2)$

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

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

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

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

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

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

output on a random input in fewer than t steps, violating the t-Sequentiality 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 once 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. □

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. Later, we present improved constructions aimed at reducing precomputation and/or providing reusable forgeability.

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 [87], 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.

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 $\mathsf{VDF.Randomize}(b, y, \pi, \alpha) \to (b', y', \pi')$ outputs a randomly distributed solution.

We can use a re-randomizable VDF scheme to improve the efficiency of Protocol 1, enabling efficient proof of any (true) statement after a single VDF precomputation. 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 formalize the notion of a re-randomizable VDF in Appendix A, 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 [87] 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. [19]) in Appendix A.1.

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 [87, Remark 3]; here we use it in a constructive way. It has similarly been used by Thyagarajan et al. to build verifiable timed signatures [84] and by Malavolta and Thygarajan to construct additively homomorphic and fully homomorphic time-lock puzzles [65].

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 Section 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 Section 5 essentially does this (embedded within a disjunction). Later in this section, we will present a more efficient construction based on Wesolowski proofs [87].

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

Proof. Our proof closely follows the proof for Theorem 1 and stems from the definition of a short-lived proof and Σ-protocols. The completeness of $\Sigma_{R \vee R_{\mathsf{zkVDF}}}$ is due to the completeness of Σ_R. t-Forgeability follows from the correctness property of the underlying zkVDF: Forge can produce convincing forgeries in $(1 + \epsilon)t$ steps by running zkVDF.$\mathsf{Eval}(b)$ and using the output (y, π) to satisfy the $\Sigma_{R_{\mathsf{zkVDF}}}$ half of the disjunction. The Indistinguishability and Zero-Knowledge properties both follow immediately from the zero-knowledge property of Σ-OR construction, preventing the adversary from knowing which half of the disjunction was satisfied and meaning an efficient simulator exists as required.

The t-Soundness property relies on the t-Sequentiality of the zkVDF. 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 zkVDF with non-negligible probability. This means that any adversary able to produce proofs must know a witness w for x, which the extractor for Σ_R can then efficiently extract.

Finally, to show that this scheme is reusable, note that the exact same zkVDF computation zkVDF.$\mathsf{Eval}(b)$ is required for proving any statement x. Thus, it can be computed once and immediately reused across proofs. □

7.1 zkVDF Construction

In this section we present a Σ-based zkVDF construction built off of Wesolowski proofs [87]. 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. [20, §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.

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

A proof of Theorem 4 is provided in Appendix C.

zk-PoKP

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 $\mathsf{HashToPrime}$ 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. $\mathsf{Respond}(\ell) = Q, r_2$

$\mathsf{Simulate}$
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.

7.2 Efficiency of zk-PoKP and Σ-zkVDF

The zk-PoKP $\mathsf{Simulate}$ 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 honest 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 precom-

puted advice string, identical to that used for re-randomizable VDFs (Section 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 528 bytes. The Σ-zkVDF construction additionally includes one sub-challenge (the other is implicit), which adds an extra λ bits, making the total overhead for transforming a normal proof into a Σ-zkVDF short-lived proof 544 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.

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:*

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

The following properties are satisfied:

- **Correctness:** *For all m, $b \in \mathcal{B}$, if $\sigma \leftarrow \mathsf{Sign}(pp, sk, m, b)$, then $\mathsf{Verify}(pp, pk, m, b, \sigma)$ accepts.*
- **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* $\mathsf{Verify}(pp, pk, m_*, b, \sigma_*) = Accept$.

- **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 [75], BLS [21]). In this case, it is necessary that Sign and Forge produce the same exact signature with overwhelming probability.

8.1 Constructions from Short-lived Proofs

We observe that our generic constructions in Section 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 Section 6 can also be used for Σ-based signature schemes such as Schnorr [79] or DSA[1,58].

8.2 Construction from Trapdoor VDFs

We present a short-lived signature construction from *trapdoor VDFs* [87] 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.

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

Sign-Trapdoor

KeyGen
input: $\varnothing$
output: public key pk, private key sk

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

Sign
input: message m, beacon value b
output: signature σ

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

Forge
input: message m, beacon value b
output: signature σ

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

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

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

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

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

Sign-Watermark

KeyGen
input: $\varnothing$
output: public key pk, private key sk

1. Generate keys $(pk, sk) \leftarrow \mathsf{wtVDF.Setup}()$

Sign
input: message m, beacon value b
output: signature σ

1. Compute watermark $\mu = \mathsf{Hash}(m)$
2. Compute $\sigma = (y, \pi) \leftarrow \mathsf{wtVDF.WatermarkTrapdoorEval}(sk, \mathsf{Hash}(b), \mu)$

GenAdvice
input: beacon value b
output: VDF output y, advice α

1. Compute (with delay t) $y, \alpha = \mathsf{wtVDF.Eval}(\mathsf{Hash}(b))$

FastForge
input: message m, VDF output y, advice α
output: signature σ

1. Compute watermark $\mu = \mathsf{Hash}(m)$
2. Compute proof $\pi = \mathsf{wtVDF.WatermarkEvalWithAdvice}(\mathsf{Hash}(b), \mu, y, \alpha)$
3. Output $\sigma = (y, \pi)$

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

1. Compute watermark $\mu = \mathsf{Hash}(m)$
2. Check that $\mathsf{wtVDF.WatermarkVerify}(pk, \mathsf{Hash}(b), \mu, y, \pi)$

Protocol 4: Reusably Forgeable Short-Lived Signatures from a Watermarkable Trapdoor VDFs. $\mathsf{wtVDF.WatermarkTrapdoorEval}(sk, b, \mu)$ takes secret trapdoor sk, input b and watermark μ; the latter two are used to generate the challenge.

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. □

8.3 Construction from Watermarkable VDFs

The construction in Protocol 3 does not offer reusable forgeability, as the Forge algorithm works by evaluating a VDF on a message-dependent value $x = \mathsf{Hash}(m \parallel b)$. We construct an efficient signature scheme (Protocol 4) with reusable forgeability using *watermarkable VDFs* which embed a prover-chosen *watermark* (μ) during proof generation. Watermarkable VDFs were presented informally by Wesolowski [87, §7.2]; we provide a formal definition with the essential security property (*watermark unforgeability*) in Appendix B.

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 Section 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. Constructing an efficient short-lived signature scheme with reusable forgeability between different signers is an interesting open problem.

Theorem 6 (Short-Lived Sigs from Watermarkable Trapdoor VDFs). *Assuming that* Hash *is a random oracle, Protocol 4 is a short-lived signature scheme with reusable forgeability.*

Proof. The proof is substantially similar to that of Theorem 5. Correctness follows the correctness of the underlying watermarkable trapdoor VDF. Reusable forgeability comes from the fact that computing different watermarked proofs requires fewer than t steps once the VDF solution y and advice α have been precomputed. Indistinguishability and Indistinguishability II are trivial as signing, forgery and fast forgery all produce the exact same VDF output.

Existential unforgeability comes directly from the essential security property of watermarkable VDFs [87]. In the security game for a watermarked VDF, a t-Sequential adversary is given a VDF output $y = \mathsf{VDF.Eval}(x)$ on a random point x and the ability to query a number of watermarks (μ_i) from the challenger, receiving $(y_i, \pi_i) = \mathsf{VDF.WatermarkEval}(x, \mu_i)$ in response. The adversary wins by producing a new proof (y, π_*) for a watermark $\mu_* \neq \mu_i$ for any queried w_i, such that $\mathsf{VDF.WatermarkVerify}(x, y, \pi_*, \mu_*)$ accepts. This game maps exactly to our unforgeability definition for short-lived signatures, therefore if no adversary is able to win this game then our signature scheme is existentially unforgeable. Similar to the proof for our trapdoor signature scheme, the extra capability of the adversary to query for watermarks on any values $b_i \neq b$ can be simulated by A_0 precomputing random VDF solutions and therefore cannot help to win the watermark forgeability game. □

The key idea to construction a watermarkable VDF is to embed the watermark into the Fiat-Shamir challenge, computing $\ell \leftarrow \mathsf{HashToPrime}(y \parallel \mu)$ in-

Protocol		Size Overhead		Proving Time Overhead	
zk-SNARKs	(§5)	none		60-70 seconds	
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)	$2\langle\mathbb{G}\rangle + 2\lambda$	544 bytes	$2\mathsf{exp}_{\mathbb{G}}(2^{2\lambda}\lvert\mathbb{G}\rvert) + 2\mathsf{exp}_{\mathbb{G}}(2^{\lambda})$	120 ms
Sign-Trapdoor	(§8.2)	$\langle\mathbb{G}\rangle + \lambda$	272 bytes	$\mathsf{exp}_{\mathbb{G}}(2^{\lambda})$	10 ms
Sign-Watermark	(§8.3)	$\langle\mathbb{G}\rangle + \lambda$	272 bytes	$\mathsf{exp}_{\mathbb{G}}(2^{\lambda})$	10 ms

Table 2. Additional costs required 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, t is the VDF delay parameter. For concrete evaluations, $\lambda = 128$ and $\mathbb{G}$ is an 2048-bit RSA group. Evaluation of the Generic zk-SNARK method was done using Groth16 zk-SNARKs [53].

stead of $\ell \leftarrow \mathsf{HashToPrime}(y)$. This approach is possible for both Wesolowski [87] and Pietrzak proofs [72]. Watermark unforgeability is claimed to hold for Wesolowski proofs, though proving security is an open problem because the proof reveals a value b^q for a large q which may speed up computing $y = b^{2^t}$. We advocate a more conservative approach, employing a watermarked version of our zkVDF (§7) which is guaranteed not to reveal any such information.

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 outline in Section 5. We implemented a VDF circuit using Wesolowski VDF proofs [87] using a 2048-bit RSA modulus and the "bellman-bignat" library [70].

The total size of the VDF verification circuit is just over 5 million gates. The large size is due to the number of RSA group operations involved in the verification. SNARK proofs were generated using the Groth16 construction [53] 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 60–70 seconds. The exact time depends on the base circuit that the VDF circuit is combined with.

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 [8]. 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 seconds. Our 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.

10 Applications

Deniability is a foundational subject in cryptography that has been studied from many angles, including group signatures [28], designated verifier signatures/proofs [56], deniable encryption [27], ring signatures [76], secure messaging [23], deniable authentication [42], and receipt-free [16] and coercion-resistant [59] voting protocols. Terminology varies between deniable, reputable, signer-ambiguous, non-transferable, and non-attributable. We highlight three potential applications of short-lived proofs/signatures here.

10.1 Deniable Messaging and Email

Assume an adversary produces a purported transcript of a secure communication session between Alice and Bob, along with copies of all cryptographic keys used: is this sufficient to prove Alice participated? Does it prove what was said? Does it prove, more specifically, what Alice said? Deniable messaging protocols aim to answer no to as many of these questions as possible.

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, with a synchronous, bidirectional channel, 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 [76], as popularized by Off-the-Record messaging (OTR) [23]. Deliberate publication of the MAC key after the session can extend forgeability to anyone. In group chat protocols, communication complexity generally scales (at least linearly) with the number of participants. Much research (see Unger et al. [85] for a survey) has focused on achieving the properties of OTR in a group scenario.

Deniability can be extended to offline recipients in a store-and-forward system through non-interactive designated verifier signatures [56] or ring signatures

formed between a sender and a recipient [76,23]—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 [23].

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 [40].

Over the past two decades, email dumps—the public release of private email messages from breached servers—have received extensive news coverage [32]. 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 [51]. DKIM signatures do not require validity beyond the network latency of reaching a recipient's mail server.

Specter et al. proposed KeyForge and TimeForge [81] to replace DKIM with *Forward Forgeable Signatures (FFS)* that become *non-attributable* after a specified time (e.g., 15 minutes). 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% [81].

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 [13,38]. Assuming the voter operates a computational device she does not fully trust, she wants *ballot casting assurance* [15] 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 [9], and Microsoft's ElectionGuard [3]) is the *Benaloh challenge* [14,15]: (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 [56,68,67] 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 second) 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 seconds 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. One drawback, common to all of the other the ballot assurance methods, is that the process becomes onerous as the ballot complexity increases (more candidates, multiple contests, ranked choice voting, etc.).

11 Concluding Remarks

We observe that the existence of the Forge algorithm for short-lived proofs circumvents Pass' observation [71] 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. [81], 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 Section 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 [62], some other party computing a long-running continuous VDF [43] or a decentralized protocol using chained VDFs [78,34,47].
- Our Σ-zkVDF construction (§7) only provides 1-reusable forgeability. It might be possible to extend this to k-reusable forgeability (as in §5.1) 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 (§8.3) 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 [25], Marlin [29], PLONK [48], Sonic [66], Spartan [80], Supersonic [26], or STARKs [10].

Acknowledgments

We give a special acknowledgement to Michael Colburn who initially pursued the idea of short-lived signatures and proofs in his MASc thesis [35] supervised by Jeremy Clark. The constructions in this paper are new, and any ideas or text from [35] incorporated into this paper were originated by Clark. We thank Benjamin Wesolowski, Justin Thaler, Bünz, Ben Fisch, Dan Boneh, Matthew Green, Michael Specter, and Michael Walfish and for helpful feedback and discussion. We thank Alex Ozdemir and Riad Wahby 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. `github.com/microsoft/electionguard` (2021)
4. NIST Randomness Beacon Version 2.0. `beacon.nist.gov/home` (2021)
5. Adida, B.: Helios: Web-based Open-audit Voting. In: USENIX Security (2008)
6. Baignères, T., Delerablée, C., Finiasz, M., Goubin, L., Lepoint, T., Rivain, M.: Trap Me If You Can – Million Dollar Curve. Cryptology ePrint Archive, Report 2015/1249 (2015)
7. Baldimtsi, F., Kiayias, A., Zacharias, T., Zhang, B.: Indistinguishable Proofs of Work or Knowledge. In: Eurocrypt (2016)
8. Barker, E., Dang, Q.: Recommendation for Key Management. NIST Special Publication **800-57** (2015)
9. Bell, S., Benaloh, J., Byrne, M.D., Debeauvoir, D., Eakin, B., Kortum, P., McBurnett, N., Pereira, O., Stark, P.B., Wallach, D.S., Fisher, G., Montoya, J., Parker, M., Winn, M.: STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In: JETS (2013)
10. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/46 (2018)
11. 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)
12. Benaloh, J., de Mare, M.: One-way Accumulators: A Decentralized Alternative to Digital Signatures. In: Eurocrypt (1993)
13. Benaloh, J.: Verifiable secret-ballot elections. Ph.D. thesis, Yale University (1987)
14. Benaloh, J.: Simple Verifiable Elections. In: EVT (2006)
15. Benaloh, J.: Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In: EVT (2007)
16. Benaloh, J., Tuinstra, D.: Receipt-Free Secret-Ballot Elections. In: STOC (1994)
17. Boneh, D., Naor, M.: Timed Commitments. In: CRYPTO (2000)
18. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable Delay Functions. In: CRYPTO (2018)
19. Boneh, D., Bünz, B., Fisch, B.: A Survey of Two Verifiable Delay Functions. Cryptology ePrint Archive, Report 2018/712 (2018)
20. Boneh, D., Bünz, B., Fisch, B.: Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In: CRYPTO (2019)
21. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Eurocrypt (2001)
22. Bonneau, J., Clark, J., Goldfeder, S.: On Bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015 (2015)
23. Borisov, N., Goldberg, I., Brewer, E.: Off-the-record communication, or, why not to use PGP. In: ACM WPES (2004)
24. Buchmann, J., Hamdy, S.: A survey on IQ cryptography. In: Public-Key Cryptography and Computational Number Theory (2011)
25. 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)
26. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: CRYPTO (2020)

27. Canetti, R., Dwork, C., Naor, M., Ostrovsky, R.: Deniable Encryption. In: CRYPTO (1997)
28. Chaum, D., van Heyst, E.: Group Signatures. In: Eurocrypt (1991)
29. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: CRYPTO (2020)
30. Ciampi, M., Persiano, G., Scafuro, A., Siniscalchi, L., Visconti, I.: Improved OR Composition of Sigma-Protocols. In: TCC (2016)
31. Clark, J., Hengartner, U.: On the Use of Financial Data as a Random Beacon. In: EVT/WOTE (2010)
32. Clark, J., van Oorschot, P.C., Ruoti, S., Seamons, K., Zappala, D.: SoK: Securing email: A stakeholder-based analysis. In: Financial Cryptography (2021)
33. Cohen, B., Pietrzak, K.: Simple Proofs of Sequential Work. In: CRYPTO (2018)
34. Cohen, B., Pietrzak, K.: The Chia network blockchain (2019)
35. Colburn, M.: Short-lived signatures. Master's thesis, Concordia University (August 2018)
36. Couteau, G., Klooß, M., Lin, H., Reichle, M.: Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments. In: Eurocrypt (2021)
37. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: CRYPTO (1994)
38. Cramer, R., Gennaro, R., Schoenmakers, B.: A Secure and Optimally Efficient Multi-authority Election Scheme. In: Eurocrypt (1997)
39. Dodis, Y., Yum, D.H.: Time Capsule Signature. In: Financial Cryptography (2005)
40. Durumeric, Z., Adrian, D., Mirian, A., Kasten, J., Bursztein, E., Lidzborski, N., Thomas, K., Eranti, V., Bailey, M., Halderman, J.A.: Neither snow nor rain nor MITM: An empirical analysis of email delivery security. In: ACM CCS (2015)
41. Dwork, C., Naor, M.: Pricing via Processing or Combatting Junk Mail. In: CRYPTO (1992)
42. Dwork, C., Naor, M., Sahai, A.: Concurrent Zero-Knowledge. Journal of the ACM (JACM) **51**(6) (2004)
43. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous Verifiable Delay Functions. In: CRYPTO (2020)
44. Feo, L.D., Masson, S., Petit, C., Sanso, A.: Verifiable Delay Functions from Supersingular Isogenies and Pairings. Cryptology ePrint Archive, Report 2019/166 (2019)
45. Ferradi, H., Géraud, R., Naccache, D.: Slow Motion Zero Knowledge Identifying with Colliding Commitments. In: ICISC (2015)
46. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Eurocrypt. Springer (1986)
47. Foundation, E.: Ethereum 2.0 Beacon Chain (2020)
48. 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)
49. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: CRYPTO (2013)
50. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM (JACM) **38**(3) (1991)
51. Green, M.: Ok Google: please publish your DKIM secret keys (November 2020)
52. Groth, J.: Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In: Eurocrypt (2010)

53. Groth, J.: On the Size of Pairing-Based Non-interactive Arguments. In: CRYPTO (2016)
54. Halderman, J.A., Waters, B.: Harvesting Verifiable Challenges from Oblivious Online Sources. In: CCS (2007)
55. Hazay, C., Chen, M., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., abhi shelat, Wang, R., Venkitasubramaniam, M.: Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. IEEE Security & Privacy (2021)
56. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated Verifier Proofs and their Applications. In: Eurocrypt (1996)
57. Jaques, S., Montgomery, H., Roy, A.: Time-release Cryptography from Minimal Circuit Assumptions. Cryptology ePrint Archive, Report 2020/755 (2020)
58. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security **1**(1), 36–63 (2001)
59. Juels, A., Catalano, D., Jacobsson, M.: Coercion-Resistant Electronic Elections. In: WPES (2005)
60. Krawczyk, H., Rabin, T.: Chameleon Signatures. In: NDSS (2000)
61. Kucherawy, M., Crocker, D., Hansen, T.: DomainKeys identified mail (DKIM) signatures. RFC 6376 (2011)
62. Landerreche, E., Stevens, M., Schaffner, C.: Non-interactive Cryptographic Timestamping based on Verifiable Delay Functions. In: Financial Cryptography (2020)
63. Lenstra, A.K., Wesolowski, B.: Trustworthy public randomness with sloth, unicorn, and trx. International Journal of Applied Cryptography **3**(4), 330–343 (2017)
64. Mahmoody, M., Moran, T., Vadhan, S.: Publicly Verifiable Proofs of Sequential Work. In: ITCS (2013)
65. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic Time-Lock Puzzles and Applications. In: CRYPTO (2019)
66. 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)
67. Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: CRYPTO (2006)
68. Neff, C.A.: Practical High Certainty Intent Verification for Encrypted Votes. Tech. rep., VoteHere Whitepaper (2004)
69. Okamoto, T., Uchiyama, S.: A New Public-Key Cryptosystem as Secure as Factoring. In: Eurocrypt (1998)
70. Ozdemir, A., Wahby, R., Whitehat, B., Boneh, D.: Scaling Verifiable Computation Using Efficient Set Accumulators. In: USENIX Security (2020)
71. Pass, R.: On Deniability in the Common Reference String and Random Oracle Model. In: CRYPTO (2003)
72. Pietrzak, K.: Simple Verifiable Delay Functions. In: ITCS (2018)
73. Rabin, M.: Transaction protection by beacons. Journal of Computer and System Sciences **27**(2) (1983)
74. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release Crypto. Tech. Rep. TR-684, MIT (1996)
75. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
76. Rivest, R.L., Shamir, A., Tauman, Y.: How to Leak a Secret. In: Asiacrypt (2001)
77. Sander, T.: Efficient Accumulators Without Trapdoor. In: ICICS (1999)

78. Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., Weippl, E.: RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. Cryptology ePrint Archive, Report 2020/942 (2020)
79. Schnorr, C.P.: Efficient Signature Generation by Smart Cards. Journal of Cryptology **4**(3), 161–174 (1991)
80. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO (2020)
81. Specter, M.A., Park, S., Green, M.: KeyForge: Non-Attributable Email from Forward-Forgeable Signatures. In: USENIX Security (2021)
82. Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable Bias-Resistant Distributed Randomness. In: IEEE Security & Privacy (2017)
83. Thaler, J.: Proofs, Arguments, and Zero-Knowledge (2021)
84. Thyagarajan, S.A.K., Bhat, A., Malavolta, G., Döttling, N., Kate, A., Schröder, D.: Verifiable Timed Signatures Made Practical. In: ACM CCS (2020)
85. Unger, N., Dechand, S., Bonneau, J., Fahl, S., Perl, H., Goldberg, I., Smith, M.: SoK: Secure messaging. In: IEEE Security & Privacy (2015)
86. Wesolowski, B.: A proof of time or knowledge, `https://hal.archives-ouvertes.fr/hal-03380471`
87. Wesolowski, B.: Efficient Verifiable Delay Functions. In: Eurocrypt (2019)

A Re-randomizable VDFs

We provide a complete definition of re-randomizable VDFs, as proposed in Section 6.3 as an optimization to avoid precomputation in Protocol 1.

Definition 4 (Re-randomizable VDFs). *A re-randomizable VDF is a set of algorithms* $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify}, \mathsf{Randomize})$ *with the following functionality:*

- $\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify}$ *are the same as basic VDFs, though* Eval *may output an additional advice string* α*.*
- $\mathsf{Randomize}(pp, b, y, \alpha) \to (b', y', \pi')$ *is a randomized function that takes as input* b, y, α *such that* $(y, \alpha) = \mathsf{VDF.Eval}(pp, b)$ *and outputs a tuple* (b', y', π')*.*

Additionally, the following two properties are satisfied:

- **Re-randomizability:** *For all* b*, given* $(y, \alpha) = \mathsf{VDF.Eval}(pp, b)$*,* $\mathsf{Randomize}(pp, b, y, \alpha) \to (b', y', \pi')$ *runs in fewer than* t *sequential steps and* $\mathsf{Verify}(pp, b', y', \pi')$ *accepts.*
- **Indistinguishability:** *No pair of polynomial time algorithms* $(\mathcal{A}_0, \mathcal{A}_1)$ *can win the following game with probability non-negligibly greater than* $\frac{1}{2}$*:*
  1. *Challenger* C *runs* $pp \leftarrow \mathsf{Setup}(\lambda)$*, chooses an element* $b_0 \xleftarrow{\$} \mathbb{G}$*, runs* $(y, \alpha) \leftarrow rrVDF.\mathsf{Eval}(pp, b_0)$ *and sends* pp, b_0, y, α *to* $\mathcal{A}_0$
  2. *Adversary* A *generates* $\alpha' \leftarrow \mathcal{A}_0(pp, b_0, y, \alpha)$
  3. C *flips a coin* $c \xleftarrow{\$} \{0, 1\}$*. If* $c = 0$*,* C *generates a random* $b' \xleftarrow{\$} \mathbb{G}$ *and computes* $y', \pi' \leftarrow rrVDF.\mathsf{Eval}(pp, b')$*. If* $c = 1$*,* C *computes* $b', y', \pi' \leftarrow rrVDF.\mathsf{Randomize}(pp, b_0, y, \alpha)$*.* C *sends* b', y', π' *to the adversary.*

4. *The adversary computes* $c_* \leftarrow \mathcal{A}_1(b_0, y, \alpha, \alpha', b', y', \pi')$ *and wins if it correctly guesses* $c_* = c$.

Repeated-squaring VDFs can be re-randomized via the homomorphism $(b, y) \to (b^r, y^r)$ for a random exponent r. Sampling $r \xleftarrow{\$} [0, \mathsf{order}(\mathbb{G})]$ ensures that the distribution of b^r is statistically indistinguishable from a random element of $\mathbb{G}$ when b is a generator. However, this requires r to be large – for example, it would be 2048 bits long when using RSA2048. Under the S-bounded short exponent indistinguishability (SEI) assumption [36, Definition 11], it is computationally hard to distinguish between the distributions $\{g^z \mid z \xleftarrow{\$} [0, \mathsf{order}(\mathbb{G}]\}$ and $\{g^z \mid z \xleftarrow{\$} [0, S]\}$ when $S \geq 2^{2\lambda}$. Thus, it suffices to sample $r \in [0, 2^{2\lambda}]$ to ensure a negligible probability of distinguishing between b^r for a fixed b and a random element in $\mathbb{G}$.

A.1 Efficiently Re-Randomizing Pietrzak Proofs

Protocol 5 recaps Pietrzak proofs and describe our advice string and re-randomization algorithm. Let (g, h) where $g, h \in \mathbb{G}$ be the input-output pair for which a Pietrzak proof is to be produced. Let v_i be the value produced by the Prover and r_i be the corresponding challenge given by the Verifier in round i. The intuition behind the advice string, informally presented in [19], is the following observation about the structure of the value of v_i:

$$
\begin{aligned}
v_1 &= g^{2^{t/2}} \\
v_2 &= g_1^{2^{t/4}} = (g^{r_1} v_1)^{2^{t/4}} = (g^{2^{t/4}})^{r_1} g^{2^{3t/4}} \\
v_3 &= g_2^{2^{t/8}} = (g_1^{r_1} v_2)^{2^{t/8}} = (g^{2^{t/8}})^{r_1 r_2} (g^{2^{3t/8}})^{r_1} (g^{2^{5t/8}})^{r_2} g^{2^{t/8}} \\
&\cdots \\
v_i &= \prod_{\text{odd } j}^{j<2^i} (g^{2^{t/2^i \cdot j}})^{\text{combination of } r \text{ values}}
\end{aligned}
$$

The parts not involving r_i can be pre-computed and used as advice for future proofs. Thus, the advice string will thus consist of $\{g^{2^{t/2^i \cdot j}} \mid i \leq k, \text{ odd j} < 2^i\}$. This is equivalent to $\{g^{2^{t/2^k \cdot j}} \mid j \in 1, \ldots, 2^k - 1\}$.

Theorem 7 (Re-Randomizing Pietrzak). *The advice string specified in Protocol 5 computed for element* $g \in \mathbb{G}$ *and parameter* k *enables computing Pietrzak proofs for any power pair* (g^u, h^u) *with just* $(2^k + \frac{t}{2^k})\lambda$ *group multiplications. Moreover, the size of the advice string is* 2^k *elements long. In particular, when* $k = (\log t)/2$ *re-randomization takes time* $O(\sqrt{t})$.

Proof. The proof is by construction. At the start of round i of the Pietrzak proof protocol, let $r_1, \ldots, r_{i-1}$ be the random values sent so far. At round i, the Prover

Pietrzak Proof and Re-Randomization

Prove
parameters: delay t; $d < \log t$
input: $g, h \in \mathbb{G}$
claim: $h = g^{2^t}$

1. Let $(g_0, h_0) = (g, h)$
2. For round $i = 1, \ldots, d$:
   - Prover computes and sends $v_i = g_{i-1}^{2^{t/2^i}}$
   - Verifier samples and sends $r_i \xleftarrow{\$} \{0,1\}^\lambda$
   - Both parties compute $g_i = g_{i-1}^{r_i} v_i, h_i = v_i^{r_i} h_{i-1}$.
3. Verifier manually checks that $h_d = g_d^{2^{t/2^d}}$

GenAdvice
parameters: delay t, k such that $k < \log t$
input: $g \in \mathbb{G}$
output: advice α

The advice string is $\alpha = g^{2^{(t/2^k)\cdot i}}$ for $i = 1, 2, \ldots, 2^k - 1$.

We enumerate the elements of the advice string as follows:

$$\forall i \in [1,k], \text{odd } j < 2^i : \quad \alpha_{i,j} = \mathsf{pow}(g, \mathsf{pow}(2, \frac{t}{2^i} j))$$

Randomize
parameters: delay t, k, d st $k < d < \log t$;
input: $g, h \in \mathbb{G}$
claim: $h = g^{2^t}$
private input: The prover has access to advice string α for element g_* and knows u such that $g = g_*^u$. Advice α and g_* may be public, but u is private.

Let $g_0, h_0 = g, h$: For rounds $i = 1, \ldots, k$:

1. Prover sends v_i, computed as follows. Let $R_{i,j} = \prod_{j' \notin \mathsf{bin}(j)}^{j'<i} r_{j'}$

$$\text{Compute } v_i = \prod_{j=1}^{2^i} ((\alpha_{i,2j+1})^u)^{R_{i,j}}$$

2. Verifier samples and sends $r_i \xleftarrow{\$} \{0,1\}^\lambda$
3. Both parties compute $g_i = g_{i-1}^{r_i} v_i, h_i = v_i^{r_i} h_{i-1}$.

Conduct rounds $k+1, \ldots, d$ as in protocol Prove to complete the proof π.

Protocol 5: Pietrzak proof re-randomization. $\mathsf{bin}(j)$ denotes the set of indices of the binary representation of j that are 1, with index 0 the least significant.

sends v_i and the Verifier sends r_i. Using this, we derive:

$$g_i = g_{i-1}^{r_i} v_i = g_{i-1}^{r_i} g_{i-1}^{2^{t/2^i}} = (g_{i-1})^{r_i + 2^{t/2^i}}$$

This leads to the following recursive formula:

$$g_i = g^{\prod_{j=1}^{i}(R_{i,j} + 2^{t/2^j})}$$

We can expand the exponent as follows: (let $\mathsf{PS}([i])$ denote the power set of $\{1, \ldots, i\}$).

$$\begin{aligned} g_i &= g^{(r_1 + 2^{t/2})(r_2 + 2^{t/4}) \ldots (r_i + 2^{t/2^i})} \\ &= \prod_{J \in \mathsf{PS}(i)} (g^{\prod\limits_{j \in J} 2^{t/2^j}})^{\prod\limits_{j \notin J} r_i} \end{aligned}$$

The exponent of g is $\prod\limits_{j \in J} 2^{t/2^j} = 2^{\frac{t}{2^i} \cdot \sum\limits_{j \in J} 2^{i-j}}$. As J goes through all of $\mathsf{PS}(i)$, the whole expression is equivalent to:

$$g_i = \prod_{j=1}^{2^i} (g^{\mathsf{pow}(2, (\frac{t}{2^i} \cdot j))})^{\prod\limits_{j' \notin \mathsf{bin}(j)}^{j' < i} r_{j'}}$$

where $\mathsf{bin}(j)$ is the set of indices in the binary representation of j (starting with index 0 being the least significant bit).

Let $R_{i,j} = \prod\limits_{j' \notin \mathsf{bin}(j)}^{j' < i} r_{j'}$. We can now represent v_{i+1} as:

$$\begin{aligned} v_i &= g_i^{2^{t/2^{i+1}}} \\ &= (\prod_{j=1}^{2^i} (g^{2^{(\frac{t}{2^i} j)}})^{R_{i,j}})^{2^{t/2^{i+1}}} \\ &= \prod_{j=1}^{2^i} ((g^{2^{(\frac{t}{2^i} j)}})^{2^{t/2^{i+1}}})^{R_{i,j}} \\ &= \prod_{j=1}^{2^i} (g^{2^{(\frac{t}{2^{i+1}}) \cdot (2j+1)}})^{R_{i,j}} \\ &= \prod_{j=1}^{2^i} (\alpha_{i,2j+1})^{R_{i,j}} \end{aligned}$$

The key idea is that $\alpha_{i,2j+1}$ can be precomputed as it depends only on g, t and not on the r_i values sampled in the proofs. Thus, it can be stored as part of the advice string.

The algorithm to compute a proof with the advice string is given in Protocol 5. The required operations in each round i are:

1. Querying 2^{i-1} advice elements: $\alpha_{i,2j+1}$ for odd $j < 2^i$
   $\rightarrow 2^i$ advice elements
2. Raising each advice element to the power $u \times \prod_{\text{subset } j} r_j$
   $\rightarrow 2^i \times (\log u + \lambda i)$ per advice element, as $r_j < 2^\lambda$
3. Multiplying the elements from step 2 together to obtain v_i:
   $\rightarrow 2^i$ multiplications per round
4. Computing g_i, h_i by raising both v_i, g_i to the power r_i:
   $\rightarrow \lambda$ multiplications at the end of each round

The remaining rounds $i = k+1, \ldots, d$ take up to 2^{k+1} group multiplications. Thus, the total number of group multiplications involved is:

$$\begin{aligned} 2^{k+1} + \sum_{i=1}^{k}(2^i(\log u + \lambda i) + 2^i + \lambda) &= 2^{k+1} + \sum_{i=1}^{k}(2^i \log u) + \sum_{i=1}^{k}(2^i \lambda i) + \sum_{i=1}^{k} 2^i + \sum_{i=1}^{k} \lambda \\ &= 2^{k+1} + \log u 2^{k+1} + \lambda(k+1)2^{k+1} + 2^{k+1} + \lambda(k) \\ &< 2^{k+1}(\log u + (k+1)\lambda + 3) \end{aligned}$$

With $k = (\log t)/2$, we get that this takes under $2\sqrt{t}(\log u + \lambda \log t + \lambda + 3) < 4\sqrt{t}(\log u + \lambda(\log t + 1) + 3)$, which is asymptotically $O(\sqrt{t}(\log u + \lambda \log t) = \tilde{O}(\sqrt{t})$. □

Concretely, with $\log u = 4098, \lambda = 128, \log t \leq 40$, this is $2\sqrt{t}(2^{12} + 2^7 20 + 3) < 2^{34}$. This is a 64x speed-up. The advice string would be of size $\sqrt{t} = 2^{20}$ elements, which is 256 GB for 2048-bit RSA groups.

B Watermarkable VDFs

The notion of watermarkable VDFs was proposed at the 2018 Stanford VDF workshop and presented informally by Wesolowski [87]. Motivated by our reusably forgeable signature scheme (Protocol 4, §8.3) we present a formal definition and security properties here:

Definition 5 (Watermarkble VDFs). *A watermarkable VDF is a set of algorithms* (Setup, Eval, WatermarkProve, Verify) *with the following functionality:*

- Setup, Eval *are the same as basic VDFs, though* Eval *may output an additional advice string* α.

- $\mathsf{WatermarkProve}(pp, b, \mu, y, \alpha) \to (y, \pi_\mu)$ *outputs a proof for* (b, y) *with embedded watermark* μ.
- $\mathsf{Verify}(pp, b, \tilde{\mu}, y, \pi_\mu) \to \mathsf{Accept/Reject}$ *works as for basic VDFs but in addition to verifying that* $y = \mathsf{Eval}(pp, b)$ *checks that the watermark* $\tilde{\mu}$ *is embedded in* π_μ.

A watermarkable VDF scheme must satisfy the security properties of a basic VDF scheme (correctness, uniqueness, sequentiality) as well as the following property:

- **Watermark 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 sends* pp *to* $(\mathcal{A}_0, \mathcal{A}_1)$.
  2. *The adversary precomputes its advice string* $\alpha \leftarrow A_0(pp)$.
  3. C *samples a random VDF input value* b*, computes* $y, \alpha \leftarrow \mathsf{Eval}(pp, b)$ *and sends* (b, y) *to the adversary.*
  4. *The adversary runs* $A_1(pp, b, \alpha) \leftrightarrow C$ *interactively with the challenger, adaptively sending chosen watermark queries* μ_i *to the challenger and receiving* $\pi_{\mu_i} \leftarrow \mathsf{WatermarkProve}(pp, b, \mu_i, y, \alpha)$ *in response.*
  5. A_1 *outputs a forgery pair* (μ_*, π_{μ_*}) *and wins if* $\mu_* \neq \mu_i$ *for all* i *and* $\mathsf{Verify}(pp, b, \mu_*, y, \pi_{\mu_*}) = Accept$

Watermarkable VDFs can be constructed for any Fiat-Shamir based VDF by generating the challenge as $\ell = \mathsf{HashToPrime}(y, \mu)$ instead of $\ell = \mathsf{HashToPrime}(y)$ for a suitable hash-to-prime function. This approach is possible for repeated-squaring VDFs using Wesolowski proofs [87] or Pietrzak proofs [72]. We leave proving that this approach satisfies watermark unforgeability as an open problem and note that it may not hold for either proof system as both proofs reveal some information which might help compute the solution y faster. However, it should hold for our zero-knowledge variant of Wesolowski's proof.

C Proof of zero-knowledge proof-of-knowledge of a power

We now prove Theorem 4, originally stated in Section 7.1.

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

Proof. Completeness follows trivially. We can reduce soundness to the Adaptive Root Assumption [19], which holds if no pair of polynomial time algorithms $(\mathcal{B}_0, \mathcal{B}_1)$ can win the following game with non-negligible probability: First, a group $\mathbb{G} \leftarrow \mathsf{GroupGen}(\lambda)$ is given to $\mathcal{B}_0$, who outputs $(z, \alpha) \leftarrow \mathcal{B}_0(\mathbb{G})$. Then, a random prime ℓ is chosen and $\mathcal{B}_1(z, \alpha, \ell)$ outputs $z^{1/\ell} \in \mathbb{G}$. More formally, for all polynomial time $(\mathcal{B}_0, \mathcal{B}_1)$ we have:

$$Pr\left[u^\ell = z : \begin{array}{c} \mathbb{G} \leftarrow \mathsf{GroupGen}(\lambda) \\ z, \alpha \leftarrow \mathcal{B}_0(\mathbb{G}) \\ \ell \leftarrow \mathsf{Primes}(\lambda) \\ u \leftarrow \mathcal{B}_1(\ell, \alpha) \end{array}\right] < \mathsf{neg}(\lambda)$$

After the first step of Protocol 2 (where the Prover provides a), the rest of the protocol is a special case of the Proof-of-knowledge-of-representation protocol (PoKRep) of Boneh et al. [20] for $n = 2$ with input a and in which one of the exponents, u is public. Thus, we can use PoKRep extractor [20, Theorem 7] to obtain u', v' such that $a = g^{u'}h^{v'}$. From the same theorem, we also have that $r_2 = v' \mod \ell$ with overwhelming probability.

Let $(\mathcal{A}_0, \mathcal{A}_1)$ be an adversary for zk-PoKP where $\alpha \leftarrow \mathcal{A}_0(\mathbb{G})$ and $\mathcal{A}_1(\alpha, \mathbb{G}, g, h, u)$ causes the Verifier to accept with non-negligible probability. We will show that $\mathcal{A}_1$ either knows $y = g^u$ or can be used to construct a successful adaptive-root adversary $(\mathcal{B}_0, \mathcal{B}_1)$ as follows:

1. $\alpha \leftarrow \mathcal{A}_0(\mathbb{G})$.
2. $\mathcal{B}_0$ interacts with $\mathcal{A}_1(\mathbb{G}, g, h, u)$ to obtain a and runs the PoKRep extractor to obtain u', v' such that $a = g^{u'}h^{v'}$.
3. If $u' = u$ then we have that $\mathcal{A}_1$ indeed knows $y = g^u$.
4. Otherwise, $\mathcal{B}_0$ outputs $z = a/(g^u h^{v'})$ to the Adaptive Root challenger. As $u' \neq u$, we have that $z \neq 1$.
5. $\mathcal{B}_1$ receives the adaptive root challenge ℓ.
6. $\mathcal{B}_1$ rewinds $\mathcal{A}_1$ to provide the new challenge ℓ and obtains response (Q, r_2). If the verifier accepts, we know $Q^l g^{r_1} h^{r_2} = a$. The PoKRep also guarantees that $r_2 = v' \mod \ell$ with overwhelming probability.
7. $\mathcal{B}_1$ calculates q_1, q_2 such that $u = q_1\ell + r$ and $v' = q_2\ell + r_2$ and outputs $Q/(g^{q_1}h^{q_2})$, which is a correct response in the Adaptive Root game for $z^{\frac{1}{\ell}}$:

$$\begin{aligned} \left(\frac{Q}{g^{q_1}h^{q_2}}\right)^\ell &= \frac{Q^\ell}{g^{q_1\ell}h^{q_2\ell}} \\ &= \frac{a}{g^{q_1\ell}h^{q_2\ell}g^{r_1}h^{r_2}} \\ &= \frac{a}{g^u h^{v'}} = z \end{aligned}$$

Thus, these algorithms must either know g^u or win the Adaptive Root game with overwhelming probability.

Zero-Knowledge: To show that the protocol is zero-knowledge, we must show that the simulated transcript $(\tilde{a}, \tilde{\ell}, \tilde{Q}, \tilde{r}_2)$ is indistinguishable from a real transcript (a, ℓ, Q, r_2) between an honest prover and verifier. To show this, we first note that a and $\tilde{a}$ are both uniquely defined by the rest of the transcript and hence cannot be used in a distinguishing attack. The distributions of ℓ and $\tilde{\ell}$ are identical, as are the distributions of r_2 and $\tilde{r}_2$, because in both cases the honest prover and simulator generate them in the same manner. Specifically, ℓ

and $\tilde{\ell}$ are both sampled uniformly at random from $\mathsf{Primes}(\lambda)$ while r_2 and $\tilde{r}_2$ are computed by sampling v (resp. $\tilde{v}$) uniformly at random and taking a remainder modulo ℓ (resp. $\tilde{\ell}$).

Additionally, we require that the joint distribution of (Q, r_2) and $(\tilde{Q}, \tilde{r}_2)$ are indistinguishable. This is indeed the case as the joint distributions of (q_2, r_2) and $(\tilde{q}_2, \tilde{r}_2)$ are identical and q_1 and $\tilde{q}_1$ are independent of r_2 and $\tilde{r}_2$, respectively. As $Q, \tilde{Q}$ are uniquely defined by (q_1, q_2) and $(\tilde{q}_1, \tilde{q}_2)$ respectively, the joint distributions with (Q, r_2) and $(\tilde{Q}, \tilde{r}_2)$ are thus identical.

It remains to be shown only that the distributions of Q and $\tilde{Q}$ are indistinguishable. We first claim that the distribution of h^{q_2} is statistically indistinguishable from the uniform distribution over the subgroup $\mathbb{G}_h$ generated by h. The distribution of h^{q_2} depends on the distribution of $q_2 \mod \mathsf{ord}(\mathsf{h})$, where $\mathsf{ord}(h) = |\mathbb{G}_h|$. As v is uniformly sampled from $[-B, B]$, the distribution of $q_2 = \lfloor v/\ell \rfloor$ is statistically indistinguishable from the uniform distribution U_Y over $Y = [-\lfloor B/\ell \rfloor, \lfloor B/\ell \rfloor]$ as $B > 2^{2\lambda} n|\mathbb{G}|$ and $\ell < 2^{\lambda}$. Note that Y has at least $\frac{2B}{\ell} - 1$ elements. This means that the distance between the distribution of q_2 and $U_{|\mathbb{G}_h|}$ is at most $\mathbb{G}_h/Y$ more than the distance between the distribution of q_2 and $|U_Y|$[20, Theorem 10, Fact 1]. This additional distance is also negligible with given our choice of B. Thus, the distribution of q_2 is statistically indistinguishable from $U_{|\mathbb{G}_h|}$ which implies that h^{q_2} is statistically indistinguishable from a randomly chosen element from the subgroup $\mathbb{G}_h$.

As $\tilde{v}$ is generated in the same manner as v, the distributions of q_2 and $\tilde{q}_2$ are indistinguishable. Therefore by an equivalent argument to that for h^{q_2}, it follows that $h^{\tilde{q}_2}$ is statistically indistinguishable from a randomly chosen element from the subgroup $\mathbb{G}_h$.

Statistical Indistinguishability: We obtain statistical indistinguishability of $Q, \tilde{Q}$ when h can generate g (in particular, when $\mathbb{G}_h = \mathbb{G}_g$). In this case, there must exist some $k \in \mathbb{Z}$ for which $h^k = g$. This means that $Q = g^{q_1} h^{q_2} = h^{kq_1 + q_2}$. As the distribution of $q_2 \mod |\mathbb{G}_h|$ is statistically indistinguishable from $U_{|\mathbb{G}_h|}$, so is $(kq_1 + q_2) \mod |\mathbb{G}_h|$, as kq_1 is independent of q_2. The same argument applies to $\tilde{Q} = g^{\tilde{q}_1} h^{\tilde{q}_2} = h^{k\tilde{q}_1 + \tilde{q}_2}$. Thus, both Q and $\tilde{Q}$ are both statistically indistinguishable from the uniform distribution over $\mathbb{G}_h$ (and therefore from each other).

It is possible to efficiently generate pairs (g, h) such that h generates g within an RSA group with a modulus N generated using safe primes. In such a group, we can efficiently sample a random generator of QR_N, the subgroup of quadratic residues modulo N, along with a random member of QR_N. By sampling h to be a generator of QR_N and g to be a random element of QR_N, we obtain statistical indistinguishability as h always generates g.

Computational Indistinguishability: In general, we can obtain computational indistinguishability of Q and $\tilde{Q}$ when it is difficult to efficiently check if g can be generated by h. This can be seen using a hybrid argument where the first hybrid creates $Q, \tilde{Q}$ with (g, h) such that g is generated by h. We have seen that $Q, \tilde{Q}$ are statistically indistinguishable in this case. We then swap g with g' where g' cannot be generated by h. If there exists an efficient distinguisher of Q and $\tilde{Q}$

in the new hybrid, then we can use it to distinguish between the two hybrids, violating the assumption.

We can also obtain computational indistinguishability when elements from different cosets of $\mathbb{G}_h$ in $\mathbb{G}$ cannot be efficiently distinguished (which is true under the p-subgroup assumption for RSA groups [69]). Since we have shown $h^{\tilde{q}_2}$ to be indistinguishable from $\mathbb{G}_h$, it holds that Q and $\tilde{Q}$ are indistinguishable from the distributions of $g^{q_1}\mathbb{G}_h$ and $g^{\tilde{q}_1}\mathbb{G}_h$, respectively. Equivalently, Q and $\tilde{Q}$ are each uniformly distributed over a different coset of $\mathbb{G}_h$ depending on the choices of q_1 and $\tilde{q}$. Thus, if it is computationally difficult to decide if two elements belong to the same coset of $\mathbb{G}_h$ in $\mathbb{G}$, then Q and $\tilde{Q}$ are computationally indistinguishable. □

D Wesolowski's Deniable Identification

We point out a subtle issue in the deniable identification scheme proposed by Wesolowski [87]. The protocol enables Alice to identify herself to a server, while a third-party adversary Judy tries to determine if an identification attempt has been made. The protocol is deniable if Judy cannot reliably distinguish real authentications from forgeries, even when Judy corrupts the server.

Alice generates a key pair $\mathsf{tdVDF.Setup}() \to (pk, sk)$ for a trapdoor VDF and the server registers pk. Alice sends pk to the server. An honest server starts a timer and responds with an unpredictable random challenge c. Alice computes $\mathsf{tdVDF.Eval}(c) \to (y, \pi)$ and sends (y, π) for verification. The server accepts (y, π) if the timer is less than time parameter Δ where Δ is the estimated elapsed time of computing the sequential steps of $\mathsf{tdVDF.Eval}$.

Assume for simplicity that Alice and the server have negligible distance between them, while Judy's distance from both is such that she cannot communicate faster than $\Delta/2$ with either of them. Consider one strategy Judy might use to break deniability: she corrupts the server so she can program the challenges it will send to Alice. If she sees a response from Alice of (y, π) any time earlier than Δ after she sent the challenge to the server, she can correctly identify it as a valid authentication attempt. However her distance from the server ensures that the challenge takes at least $\Delta/2$ time to travel to the server and once Alice's response is available to the server, it takes another $\Delta/2$ to return. The critical property of Wesolowski's security argument is the following property: "the challenge c is independent of anything Judy sent after point in time $t_0 - \Delta/2$" [87].

We make a small observation: the existence of beacons imply the distance between Judy needs to be doubled from $\Delta/2$ to Δ. Assume the beacon is negligibly close to server and outputs b at a particular time t_0. While Judy will not learn the beacon value immediately because of her distance, we assume that when she comes to know b, she also knows it was known to the beacon (and thus the server and Alice) at t_0. With these assumptions, she corrupts the server to use beacon values as the challenge. Thus if she can hear Alice's response in time $\Delta/2$ (as in Wesolowski's original proposal) for a trapdoor VDF with delay Δ, she concludes it is a legitimate authentication request. However if she is greater

than Δ time-distance, she cannot distinguish it as legitimate. The broader point is that using beacons, which are in all of our constructions, is not a minor addition to these kinds of time-delay protocols, but is a foundational primitive that affects security properties in a meaningful way.