# "Check-Before-you-Solve": Verifiable Time-lock Puzzles

***Abstract*—Time-lock puzzles are cryptographic primitives that guarantee to the *generator* that the puzzle cannot be solved in less than $\mathcal{T}$ sequential computation steps. They have recently found numerous applications, e.g., in fair contract signing and seal-bid auctions. However, *solvers* have no a priori guarantee about the solution they will reveal, e.g., about its "usefulness" within a certain application scenario. In this work, we propose *verifiable time-lock puzzles (VTLPs)* that address this by having the generator publish a succinct proof that the solution satisfies certain properties (without revealing anything else about it). Hence solvers are now motivated to "commit" resources into solving the puzzle. We propose VTLPs that support proving arbitrary NP relations $\mathcal{R}$ about the puzzle solution. At a technical level, to overcome the performance hurdles of the "naive" approach of simply solving the puzzle within a SNARK that also checks $\mathcal{R}$, our scheme combines the "classic" RSA time-lock puzzle of Rivest, Shamir, and Wagner, with novel building blocks for "offloading" expensive modular group exponentiations and multiplications from the SNARK circuit. We then propose a second VTLP specifically for checking RSA-based signatures and verifiable random functions (VRFs). Our second scheme does not rely on a SNARK and can have several applications, e.g., in the context of distributed randomness generation. Along the road, we propose new constant-size proofs for modular exponent relations over hidden-order groups that may be of independent interest. Finally, we experimentally evaluate the performance of our schemes and report the findings and comparisons with prior approaches.**

## 1. Introduction

Time-lock puzzles (TLPs) [1], [2], [3], [4], [5], [6], [7], [8], [9], [10] are cryptographic primitives that allow a *puzzle generator* to produce a puzzle such that anyone attempting to compute its solution will only be able to do so after devoting $\mathcal{T}$ sequential computational "steps" (even using multiple machines or parallelism). The key property of TLPs is the asymmetricity of the puzzle: the puzzle generator can generate the puzzle easily for any $\mathcal{T}$ while solving the puzzle takes time linear with $\mathcal{T}$. This finds many applications when such a "time delay" needs to be imposed, for example, fair contract signing [2], seal-bid auctions [1], [8], [11], [12], zero-knowledge proofs [13], [14], timed encryption [7], [15], [16], non-malleable timed commitments [2], [17], [18], distributed randomness generation [8], [19], or Byzantine consensus [9], [20]. With the popularity of cryptocurrencies, TLPs have attracted additional attention due to promising applications in Decentralized-Finance distributed systems, like preventing front-running [21], multi-signature transactions [6], and cross-chain atomic swaps [22].

However, the asymmetry of TLPs in terms of computational resources between the generator and the solvers is a "double-edged sword". It enables many interesting applications but, on the other hand, it imposes a simple limitation: puzzle solving takes effort but the solver cannot be sure it will retrieve valid, useful information unless the puzzle generator is trusted. E.g., in a blockchain setting, a malicious puzzle generator may abuse this to force other participants to "waste" their computational resources in hopes of benefiting from the solution, only to find out the latter is "bogus". As a solution to this problem, a recent work by Thyagarajan et al. [6], examined the case of *time-lock signatures*. These are time-lock puzzles where the solution is a signature of a public message under a public key (e.g., useful to impose a minimum delay before blockchain transactions can be processed). Their approach requires the puzzle generator to provide a proof about this, that can be checked by interested solvers before they solve the puzzle so that they decide whether to commit their resources.

In this work, we look into the more general version of this problem, aiming for time-lock puzzles with general provable guarantees regarding their solution. In particular, we ask whether it is possible to *efficiently prove a solution to a time-lock puzzle satisfies an arbitrary NP relation?*

**Verifiable Time-lock Puzzles.** We first introduce the notion of *verifiable time-lock puzzles for NP relations (VTLP for NP)*, which extends time-lock puzzles to support proving the solution fulfills an NP relation. Our definition enhances conventional time-lock puzzles by requiring the puzzle to be accompanied by proof for this claim. Hence, VTLPs must satisfy *soundness* (i.e., no cheating puzzle generator can convince verifiers about the validity of a false claim) and *zero-knowledge* (i.e., the proof and verification reveal nothing about the solution–or any additional witness detail in the NP relation). In practice, the former guarantees that, upon validating the proof, the verifier is convinced that the execution of exactly $\mathcal{T}$ computational steps will yield a solution that conforms to the specified NP relation. The latter guarantees that no information is leaked *until* the puzzle is solved. For example, the VTLP can prove that the solution is the preimage of a public hash value, and solving the puzzle guarantees a solution after $\mathcal{T}$ "steps". Another example is proving the time-lock puzzle hides a valid transaction signature, where the sender is the preimage of "some" leaf node of a public tree root, with enough balance to make the transaction. This directly enables delayed transactions

without relying on a trusted party or smart contracts. In that sense, verifiability for time-lock puzzles extends what can be achieved in the context of classical TLP applications for fair contract signing [2], auctions [1], [12] and more [6].

**Building Efficient VTLPs.** A rather straightforward, but terribly inefficient, VTLP construction for arbitrary relations can be achieved by combining a TLP with a *succinct non-interactive argument of knowledge (SNARK)* for NP [23], [24]. The puzzle generator provides a SNARK proof for a circuit that (i) checks solution $s$ satisfies the relation (using possibly a witness $w$), and (ii) proves the puzzle indeed opens to solution $s$ by performing $\mathcal{T}$ steps of computation. For some TLPs, it may be possible to "speed up" proving (ii). For instance, consider the "widely-used" RSA-based time-lock puzzle [1] that, given $(z, \mathcal{T}, N)$ accepts solution $z^{2^{\mathcal{T}}} \mod N$. With the RSA trapdoor $\phi(N)$ as witness input, the prover can perform the target exponentiation much faster in the SNARK. However, even this faster approach can have prohibitively large overheads for most applications. E.g., using xjSNARK [25], we observe that a single group operation—either multiplication or squaring—with a 2048-bit modulus necessitates approximately $18,000$ constraints. To further illustrate, verifying exponentiation of the form $z^{\tau} \mod N$ (where $\tau$ is a 2048-bit integer, i.e., after reducing modulo $\phi(N)$) would take roughly $2048$ squarings and multiplications, resulting in a SNARK circuit with approximately $72$ *million* constraints.

In this work, we propose a more efficient alternative for practical VTLPs following a modular approach for the proof. We first propose two $\Sigma$-*protocols* [26] for modular exponentiation relations in hidden order groups (Section 4). The first protocol is **PoKEModN** and it extends the protocol of [27] to the general moduli case, proving a committed integer $x$ modulo $n$ is $\hat{x}$. Our second protocol is **PoMoDE** and it proves a committed integer $x$ raised to the power of $e$ modulo $n$ is $\hat{x}$. Both constructions achieve $O(1)$ proof size and verification time, improving upon the prior state-of-the-art for hidden-order groups [28]. We then extend them to achieve zero-knowledge and propose two "SNARK-offloading" techniques, which allow a SNARK prover to offload parts of an elaborate computation from the SNARK circuit and prove them separately. The first one is **OffloadProd** and it works for computing products of sets of values, and the second is **OffloadExp**, for modular exponentiations for large integers.

Comparing with the "direct" computation of $u^x \mod N$ (where $x$ is a 2048-bit integer) inside the circuit that we mentioned above, our **OffloadExp** helps prove the same statement with merely $535,000$ constraints, yielding a circuit **$\sim$140 times smaller**! Finally, relying on these techniques, we build our VTLP from the RSA-based TLP [1].

*Comparison with Time-lock Signatures.* Besides supporting arbitrary NP relations, our solution also yields time-lock signatures that are both more efficient and more secure relative to the verifiable-time-lock signature than the scheme of Thyagarajan et al. [6]. At a high level, that work (as well as follow-ups [29], [30]) achieves verifiability by integrating a linearly-homomorphic time-lock puzzle [3] with multiple iterations of threshold secret sharing [31]. However, the soundness depends on having a substantial number of secret shares, which directly impacts the proving time and proof size, resulting in significant computational overhead and large proofs. Moreover, this approach suffers from a non-negligible soundness error for practical implementations of any time-lock signature. For instance, to verify a time-lock BLS signature [32] with a time parameter of $\mathcal{T} = 10^6$ using 40 shares, the proving procedure demands approximately *30 seconds*, the proof size scales with the number of shares, and the soundness error is at $10^{-12}$. Verification consumes *40 seconds*, also scaling with the number of shares. In comparison, our approach instantiated with Groth16 [23] enables the proving of a VTLP for a BLS signature in a mere *1.37 seconds*, yielding a proof whose size remains *constant* irrespectively of the time parameter $\mathcal{T}$. Verification is swift, requiring just 1ms, and this efficiency is maintained regardless of the time parameter—after a one-time transparent setup to establish $\mathcal{T}$. Crucially, our system's soundness error is negligible, overall substantially improving over [6].

*TLPs for sealed-bid auctions.* A recent work by Tyagi et al. [12] proposed Riggs, a system for sealed-bid auctions with guaranteed bid opening and provably valid price bids. To efficiently support simultaneous bids in multiple auctions, it uses Pedersen commitments for the bids which can then be homomorphically aggregated in a single commitment. Then, to avoid "over-bidding", bidders must provide a (range) proof that the pre-image of this aggregated commitment is smaller than their available assets. Computing such a proof for Pedersen is fairly efficient, however, to guarantee auction fairness (i.e., that all bids are eventually opened) bidders must also submit a TLP with the Pedersen pre-image as the solution. This raises the issue of ensuring consistency between TLPs and Pedersen commitments.

The authors of [12] consider using a SNARK, however, they deem this approach prohibitively costly. Instead, Riggs requires all bidders to put a significant amount per bid into escrow as collateral in case of misbehavior. If the bidder provides a TLP solution that is also the Pedersen pre-image within the bid opening phase, they can claim back the escrow; otherwise, after $\mathcal{T}$ time, the escrow is claimed by anyone providing a solution. While this guarantees consistency against rational adversaries that do not wish to lose their escrow, it imposes financial burdens on all honest bidders as well. Their funds must be held in escrow for a period, and if for any reason they cannot be online in time, they risk losing them altogether. Our VTLP can be used as a drop-in replacement of the above to provably "link" the TLP with the Pedersen commitment. Our experimental results demonstrate that this approach is around $150\times$ more efficient than the SNARK-based option considered in [12].

**Time-lock RSA signature and VRF.** We also propose a second VTLP for the specific case of verifying the solution with respect to an RSA signature, or RSA-based *verifiable random function (VRF)* evaluation [33], [34], [35]. For the former, the puzzle's solution is "guaranteed" to be a valid

RSA signature for a given message. Regarding the latter, at a high level, VRFs are analogous to pseudo-random functions in the public-key setting. Given an input and a private key, one can produce a unique pseudo-random value, accompanied by a publicly verifiable proof of correctness. Our VTLP can hide the VRF evaluation for given public input and key until the puzzle is solved. Our construction again combines our modular proof techniques with the RSA-based TLP but, crucially, *does not require a SNARK* to produce the proof. Below, we discuss possible applications of our second VTLP in the context of distributed random beacons and online lotteries.

*Applications of Time-lock RSA Signature/VRF.* The VRF use-case can directly apply to distributed random beacon generations [36]. A common strategy for creating a distributed random beacon involves multiple entities committing to their VRF values and subsequently revealing them after a designated accumulation phase. Nonetheless, this approach is vulnerable to manipulation by adversarial participants who may opt to withhold their commitments, e.g., in a DoS attack or to affect the outcome of the beacon. Our scheme would circumvent this critical issue by ensuring that the committed (time-lock) VRF values will *mandatorily* be exposed when the puzzle is solved, thus preserving the fairness and unpredictability of the random beacon. We consider this a meaningful advancement in developing reliable distributed randomness systems.

Another application of time-lock VRF is distributed online lottery and gaming, described in [37]. In this scenario, a VRF value is committed first, and after the bidding/gaming phase, the committed VRF value "should be" revealed when the game organizer is *honest*. However, if the organizer is malicious, they might abstain from revealing the committed VRF value, thereby stalling the game's resolution. To counter this problem and ensure a fair game, our VTLP for VRF can be integrated with smart contracts. Again, this enforces the eventual revelation of the VRF value, safeguarding the integrity of the game's outcome. In contrast, [37] focused on efficiently proving the legitimacy of a committed VRF but without achieving enforced openings.

Finally, our VTLP can facilitate RSA screening [38], a technique that for batch authentication of messages through a single proof (essentially the product of the individual signatures modulo $N$, where $N$ denotes the RSA group). Hence, our VTLP for RSA signatures can encompass an arbitrary amount of messages with a single proof.

**Offloading RSA Signature Verifications.** A "by-product" application of the building blocks we develop, besides VTLPs, is an efficient SNARK-offloading tool for RSA signature checking, e.g., to prove several elements of a SNARK witness are signed, crucially, without verifying all the signatures within the SNARK circuit. The most efficient existing tool for directly checking signatures inside a SNARK is BabyJubjub [39]. We measured that with Groth16 it takes $\sim$7 million constraints to check 1000 EdDSA signatures using BabyJubjub, while it takes $\sim$1.3 million constraints with our signature offloading technique for 1000 RSA 3072-bit signatures, resulting in 5-6$\times$ faster provers. A recent work by Fiore and Tucker [40] follows a different approach for signature-offloading, based on a scheme with homomorphic signatures called SPHinx. They target the problem of verifiable computation over streams of signed data, e.g., with applications in stock market oversight or forecasting services. Although SPHinx also deploys a SNARK, it leverages a special "tagged" signature, where each message is accompanied by a *unique* tag, that must be public and checked as part of the proof. This immediately makes the signature verification grow linearly with the number of signed messages. In comparison, our signature offloading focuses on the *standard* RSA signature with the PKCS 2.1 standard [41].

**Summary of Contributions.** Our overall contributions can be summarized as follows:

1) We propose definitions for VTLPs for NP and the first construction that satisfies them. Our scheme has efficient proof generation, and $O(1)$ proof size and verification cost. As a special case, it achieves very efficient verifiable time-lock signatures, for SNARK-friendly signatures, e.g., BabyJubjub, and BLS.
2) We propose a VTLP for efficient RSA signature/VRF checking, without relying on SNARKs.
3) We propose a SNARK-offloading scheme for batch verification of standard full-domain-hash RSA signatures.
4) As important building blocks to the above, we propose protocols for proving different modular exponent relations in hidden order groups and use them to enable offloading product and modular exponentiation computations from a SNARK circuit.
5) We experimentally evaluate the performance of our constructions and compare it with prior state-of-the-art approaches.

## 2. Related Work

**Verifiable Delay functions (VDFs), TLPs and VTLPs.** VDFs [42], [43] are functions requiring $\mathcal{T}$ steps to compute, even using parallelization. In this sense, VDF is a close notation to TLP, especially since they are usually based on similar cryptographic primitives and assumptions. However, their concepts differ in purpose: TLPs are designed to generate puzzles easily without guaranteeing a valid solution, while VDFs require a prover to demonstrate that they have performed $\mathcal{T}$ steps for a specific problem, which the verifier can check efficiently, often with less than $\log(\mathcal{T})$ steps. There is a line of works [2], [7], [8], [44], [45] that aims to combine both approaches to allow the solver (after performing $\mathcal{T}$ steps of computation) to convince third parties about the validity of a computed solution (thereby also proving the solver's computational effort). In contrast, VTLPs allow the generator to provide a proof about certain properties of the solution, without revealing the solution.

**TLPs with batch solving.** The fact that TLPs do not guarantee any specific property of the solution raises important challenges in many real-world TLP applications. For

instance, in a sealed-bid Vickrey auction, it is necessary to eventually reveal all bids to deduce the winner and the hammer price (the second-highest price). One way to enforce bid "openability" by the auctioneer is to use TLPs for the bids, in case adversarial bidders refuse to open their bids (e.g., to drive the price down). To make things easier for the auctioneer, [3], [4], [9], [46], [47] propose TLPs with efficient batch opening, where puzzles are time-locked towards a certain "key", and all the puzzles can be solved in $\mathcal{T}$ steps. While VTLPs do not directly solve this problem, they can offer a defense mechanism by asking each bidder to prove the validity of its bid. It is also an interesting problem to design VTLPs with batch solving.

**Proof of knowledge of Exponent (PoKE) and Double Exponent (PoKDE).** Various versions of proof-of-exponent protocols have been extensively studied in the literature [27], [48], [49], [50], [51], [52]. In this paper, we focus on the special case of **PoKDE** over hidden-order groups. This problem was first studied by Camenisch and Stadler [28] to build group signatures. The authors provide two protocols for **PoKDE**, with proof size and verification overhead either linear to the security parameter or linear to the bit-length of the "upper" exponent. Their protocol has since been adopted in Zerocoin [53]. In this paper, we propose a **PoKDE** with *constant proof size and verification time*, building upon the **PoKE** of [27] to extend it to the double-exponent case.

## 3. Preliminaries

**Notation.** Let $\lambda$ denote a security parameter. A function $\text{negl}(\lambda)$: $\mathbb{N} \to \mathbb{R}^+$ is negligible if for every positive polynomial $poly(\lambda)$ there exists a $\lambda_0 \in \mathbb{N}$, such that for all $\lambda > \lambda_0 : \epsilon(\lambda) < 1/poly(\lambda)$. We denote $PPT$ as "probabilistic polynomial time" with respect to $\lambda$ through our paper. Let $[j] = \{0, 1, \ldots, j-1\}$. We denote by $x \xleftarrow{\$} F$ sampling uniformly at random from domain $F$. We denote a vector by $\vec{a}$ and denote the size of vector $\vec{a}$ by $|\vec{a}|$.

**Proof of Exponentiation.** Given a hidden order group $\mathbb{G}_?$, one integer $x$ and generator $g$ and another group element $y$, a Proof of Exponentiation (PoE) protocol [42] proves the exponentiation relation $g^x = y \in \mathbb{G}_?$ with a constant-sized proof. The verifier first sends a prime challenge $l$, the prover calculates $q, r$ s.t. $x = ql + r, r \in [0, l)$, and sends $\pi = g^q \in \mathbb{G}_?$ to the verifier. The verifier computes $r = x \mod l$ and checks $\pi^l g^r \stackrel{?}{=} y$. Boneh *et al.* [27] extended the above idea to proof of knowledge of exponent for several different cases: **PoKE**$^*$ for the base $g$ encoded in the public parameter and **PoKE2** for any base $g \in \mathbb{G}_?$ and provided a Non-Interactive (**NI**) zero-knowledge variants [27]. Taking **PoKE2** for example, we refer to their protocols as (1) a prove function $\pi \leftarrow$ **NI-ZKPoKE2** $(g, G; x)$ where $\pi$ is the proof, $x$ is the witness and $G = g^x$, and (2) a verification function $1/0 \leftarrow$ **NI-ZKPoKE2**$(g, G, \pi)$.

**SNARKs.** A **S**uccinct **N**on-interactive **AR**gument of **K**nowledge (SNARK) for a relation $\mathcal{R}$ comprises of three algorithms as follows:

- $\Pi$.**Setup**$(1^\lambda, \mathcal{R}) \to$ crs. crs is a common reference string.
- $\Pi$.**Prove**$(\text{crs}, x; w) \to \pi$. $x$ is a statement and $w$ is the witness such that $\mathcal{R}(x, w)$ holds. It outputs $\pi$ as proof.
- $\Pi$.**Verify**$(\text{crs}, x, \pi) \to \{0, 1\}$. It inputs the crs, statement $x$ and proof $\pi$. It outputs 1 if the proof checks correctly; otherwise 0.

A SNARK is *complete, knowledge-sound, and succinct*. If a SNARK is also zero-knowledge, it leaks no information about the witness and is called zkSNARK [54]. In this paper, for the security and composition of different SNARK proofs, we employ Commit-and-Prove SNARKs (CP-SNARKs) [55]. Informally, the prover can commit the inputs and witnesses of a SNARK through some commitment scheme, e.g., extended Pedersen commitments which are perfect hiding and computationally binding. We denote $\boxed{c_{w1}}$ the committed witness $w1$ for a CP-SNARK proof $\pi \leftarrow \Pi$.**Prove**$(\text{crs}, \boxed{c_{w1}}, x; w2)$ where $w2$ is the non-committed part of the witness. As discussed in [55], CP-SNARK allows the modular composition of SNARKs through the committed witness.

**Time-lock puzzles (TLPs).** A time-lock puzzle scheme is a tuple of three $PPT$ functions (**KeySetup**, **GenPuz**, **SolvPuz**) between a puzzle generator and a solver. Function **KeySetup** takes in security parameter $\lambda$ and outputs a public/private key pair for the puzzle generator. Function **GenPuz** takes in its public/private key pair, a solution $s$, and a hardness parameter $\mathcal{T}$. It outputs a puzzle $z$. Function **SolvPuz** takes in the public key, a puzzle $z$, and the hardness parameter $\mathcal{T}$. It outputs the solution $s$. Following the definitions in [1], TLPs need to be correct and time-lock secure. Correctness means that a puzzle generated by honest execution of **GenPuz** can be solved using **SolvPuz** with respect to the public key and $\mathcal{T}$. Time-lock security requires that a puzzle $z$ generated under parameter $\mathcal{T}$ cannot be solved with time less than $\mathcal{T}$.

Due to space limitations, we defer the reader to Appendix A for extended preliminaries of RSA signatures and VRFs, as well as definitions of the order assumption, strong RSA assumption, adaptive root assumption, decisional modular subset sum assumption, and DDH-II assumption.

## 4. Proofs of Modular Exponentiations

In this section, we present several new succinct proofs in hidden order groups. We first build the **PoKEModN** protocol and its zero-knowledge variant **ZK-PoKEModN**. Then we build the **PoKDE** protocol and its zero-knowledge variant **ZK-PoKDE**. Lastly, we build the **ZK-PoMoDE** protocol based on the above protocols. Let $\mathbb{G}_?$ be a hidden order group and $g$ its generator. The first two proofs extend the **PoKEMon** protocol from [27], which proves the committed exponent $x$ modulo a prime $p$ is $\hat{x}$:

$$\mathcal{R}_{\textbf{PoKEMon}} = \{(C \in \mathbb{G}_?, \hat{x} \in [p]; x \in \mathbb{Z}) : C = g^x, x \bmod p = \hat{x}\}.$$

Params: security parameter $\lambda$, $g \in \mathbb{G}_?$;
Inputs: $C \in \mathbb{G}_?$, $n \in \mathbb{Z}$, $\hat{x} \in [n]$; Witness: $x \in \mathbb{Z}$;
Claim: $C = g^x$ and $x \bmod n = \hat{x}$.

**PoKEModdN**

1. Verifier sends $\ell \xleftarrow{\$}$ **Prime**$(\lambda)$.
2. Prover computes the quotient $q \in \mathbb{Z}$ and residue $r \in [\ell \cdot n]$ such that $x = q(\ell \cdot n) + r$. It sends the pair $(Q \leftarrow g^q, r)$ to the verifier.
3. Verifier accepts if $r \in [\ell \cdot n]$ and $Q^{\ell \cdot n} g^r = C$ holds in $\mathbb{G}_?$ and $r \bmod n = \hat{x}$.

**ZK-PoKEModdN** with parameter $B > \text{maxord}(\mathbb{G})2^\lambda$

1. Prover samples $m \xleftarrow{\$} [B]$ and sends $D = g^m$ and $\pi = \textbf{NI-PoKE}^*(g, D; m)$.
2. Verifier checks if $1 = \textbf{NI-PoKE}^*(g, D, \pi)$. If not, it aborts. Then, it sends $\ell \xleftarrow{\$}$ **Prime**$(\lambda)$.
3. Prover computes the quotient $q \in \mathbb{Z}$ and residue $r \in [\ell \cdot n]$ such that $x + mn = q(\ell \cdot n) + r$. It sends the pair $(Q \leftarrow g^q, r)$ to the verifier.
4. Verifier accepts if $r \in [\ell \cdot n]$ and $Q^{\ell \cdot n} g^r = C \times D^n$ holds in $\mathbb{G}_?$ and $r \bmod n = \hat{x}$.

Figure 1: Protocol **PoKEModdN** and **ZK-PoKEModdN**.

We observe that the protocol can be further extended to a more general relation for any integer $n$, denoted as:

$$\mathcal{R}_{\textbf{PoKEModdN}} = \{(C \in \mathbb{G}_?, \hat{x} \in [n]; x \in \mathbb{Z}) : C = g^x, x \bmod n = \hat{x}\}.$$

We then provide an efficient zero-knowledge version of the protocol. Furthermore, we design a new protocol for the modular double exponent relation, which proves the committed exponent in $C$ when raised to the power of $e$ and taken modulo $n$, equals $\hat{x}$:

$$\mathcal{R}_{\textbf{PoMoDE}} = \{(C \in \mathbb{G}_?, e, n, \hat{x} \in \mathbb{Z}; x \in \mathbb{Z}) : C = g^x, x^e \bmod n = \hat{x}\}.$$

We provide an efficient zero-knowledge version of the protocol. All protocols in this section are public-coin and can be made non-interactive using the Fiat-Shamir heuristic.

### 4.1. Proof of modular exponent

We observe that the original proof of modular exponent is limited to prime moduli because the soundness of the protocol relies on Shamir's trick [56], which requires the exponents to be co-prime with the modulus. We first extend Shamir's trick by the following simple lemma. We do not claim any novelty for this lemma because we consider it straightforward.

**Lemma 1.** *For any integer $N$, given integer $g, h \in \mathbb{Z}_N^*$ and $a, b \in \mathbb{Z}$, such that $g^a = h^b \bmod N$ and $\gcd(a, b) < \min(a, b)$, one can efficiently compute $D \in \mathbb{Z}_N^*$ and $\alpha \in \mathbb{Z}$ such that $D^\alpha = h$ and $\alpha > 1$ ($D$ is a non-trivial root of $h$).*

Based on Lemma 1, we can extend **PoKEMon** to **PoKEModdN**, from prime moduli only to a general moduli $N$. We depict our **PoKEModdN** protocol in Figure 1.[1]

**Zero-knowledge PoKEMod.** Our protocols fall within the family of $\Sigma$-protocols [26]. A common way to make such protocols zero-knowledge is by using additional random "blinding" factors. However, this method does not work easily for the **PoKEModdN** protocol for two reasons: (1) if we add a random number $m$ to hide the witness $x$, the remainder modulo $n$ changes according to the prover's intent and this can compromise soundness; (2) if we blind the witness by multiplying the challenge value $c$, the remainder modulo $n$ changes and correctness doesn't hold.

Instead, we can cloak the exponent as $z = mn + x$, which makes the randomizer $mn$ divide $n$ so does not affect soundness. To correctly extract the value $x$, we additionally require $m$ to be committed first, together with a zero-knowledge proof of knowledge of $m$ as the first step. To further reduce the overhead of the protocol, we show that a simple **PoKE**$^*$ is enough to provide a zero-knowledge proof of knowledge for the random number $m$. We present the **ZK-PoKEModdN** protocol in Figure 1. We introduce an additional parameter $B > \text{maxord}(\mathbb{G}_?)2^\lambda$ where $\text{maxord}(\mathbb{G})$ is the upper bound for the group order and can be set to the modulus of $\mathbb{G}_?$ in practice.

### 4.2. Proof of modular double exponent

Before we present **PoMoDE**, we introduce as a building block the proof of knowledge of double exponent (**PoKDE**) for the following relation. It proves the committed exponent in $C_1$, when raised to the power of $e$ equals the committed exponent in $C_2$:

$$\mathcal{R}_{\textbf{PoKDE}} = \{(C_1, C_2 \in \mathbb{G}_?, e \in \mathbb{Z}; x \in \mathbb{Z}) : C_1 = g^x, C_2 = g^{x^e}\}.$$

We present our **PoKDE** protocol in Figure 2. Intuitively, the soundness of **PoKDE** relies on the observation that:

- Fact 1 For a prime $\ell$, if $x_1 \bmod \ell = r_1$, then $x_1^e \equiv r_1^e \bmod \ell$. For committed $x_1, x_2$ and prime challenge $\ell$, if $r_1 = x_1 \bmod \ell$, $r_2 = x_2 \bmod \ell$ and $r_1^e \equiv r_2 \bmod \ell$, then $x_2 = x_1^e$ with overwhelming probability.

**Zero-knowledge version of ZK-PoKDE.** Similar to before, we cannot directly build a zero-knowledge **PoKDE** by "naively" adding randomizers. This is because once we cloak the exponent with random numbers, the remainder changes accordingly, and correctness does not hold ($r_1^e \neq r_2 \bmod \ell$). We turn to another approach based on the following observation:

- Fact 2 For committed $x, c$ and a prime challenge $\gamma$, the binomial-type polynomial of the form $(x+\gamma)^n - c^n$ divides $\gamma$ when $c = x$. If $c \neq x$, $(x + \gamma)^n - c^n$ divides $\gamma$ with probability less then $1/\gamma$.

1. Due to space limitations, we include all proofs in the extended version of this paper [57] (anonymized for submission) and only provide intuitions of proofs in the submitted version.

Params: security parameter $\lambda$, $g \in \mathbb{G}_?$;
Inputs: $C_1, C_2 \in \mathbb{G}_?$, $e \in \mathbb{Z}^+$; Witness: $x \in \mathbb{Z}$;
Claim: $C_1 = g^x$, $C_2 = g^{x^e}$.

**PoKDE**

1. Verifier sends $\ell \xleftarrow{\$}$ **Prime**$(\lambda)$ to the prover.
2. Prover computes $r_1 = x \bmod \ell$, $q_1 = \lfloor x/\ell \rfloor$, $Q_1 = g^{q_1}$, $r_2 = x^e \bmod \ell$, $q_2 = \lfloor x^e/\ell \rfloor$, $Q_2 = g^{q_2}$. It sends $(Q_1, r_1, Q_2, r_2)$ to the verifier.
3. Verifier accepts if $r_1, r_2 \in [\ell]$, $Q_1^\ell g^{r_1} = C_1$, $Q_2^\ell g^{r_2} = C_2$ and $r_1^e \mod \ell = r_2$.

**ZK-PoKDE** with extra parameters: range of non-smooth randomizer $[B]$, $S = \max(2^{2\lambda}|\mathbb{G}_?|, 2^{2\lambda}M, B)$

1. Prover picks $m \xleftarrow{\$} [S]$, calculates $D = g^m$ and $\pi_1 = \textbf{NI-PoKE}(g, D; m)$.
It sends $(D, \pi_1)$ to the verifier.
2. Verifier checks if $1 = \textbf{NI-PoKE}(g, D, \pi_1)$. If not, it aborts. Then, it picks $\gamma \xleftarrow{\$} [B]$, $\ell \xleftarrow{\$}$ **Prime**$(\lambda)$ and sends $(\gamma, \ell)$ to the prover.
3. Prover computes $z = x\ell + m + \gamma$, $E = g^{z^e}$, $K = C_2^{\ell^e}$, $\omega = z^e - (x\ell)^e$, $\omega' = \omega/(m+\gamma)$, $F = g^\omega = g^{\omega'(m+\gamma)}$, $\pi_2 = \textbf{NI-PoE}(C_2, K, \ell^e)$, $\pi_3 = \textbf{NIZK-PoKE2}(g^{m+\gamma}, F; \omega')$, $\pi_4 = \textbf{NI-PoKDE}(C_1^\ell \times D \times g^\gamma, E, e; z)$. It sends $(E, K, F, \pi_2, \pi_3, \pi_4)$.
4. Verifier accepts if all the following are correct: $E/K \stackrel{?}{=} F$, $1 = \textbf{NI-PoE}(C_2, K, \ell^e, \pi_2)$, $1 = \textbf{NIZK-PoKE2}(D \times g^\gamma, F, \pi_3)$ and $1 = \textbf{NI-PoKDE}(C_1^\ell \times D \times g^\gamma, E, e, \pi_4)$.

Figure 2: Protocol **PoKDE** and **ZK-PoKDE**.

To protect the privacy of committed $x$ and $c$, the prover needs to involve a randomizer $m$ to cloak $x$ and $c$, leading to the equation of $(x+m+\gamma)^n - (c+m)^n$. This is a stringent condition because if $\gamma$ is picked randomly, $m+\gamma$ is *not prime* with probability $1 - 1/\log|m+\gamma|$. To address this issue, we relax the constraint that requires $\gamma$ to be a prime number. Instead, we require $\gamma$ to be ***non-smooth***. Informally, if $\gamma$ is $y$-smooth, then $\gamma$ contains a prime factor larger than $y$. We also observe that a random number with long enough bits is non-smooth with overwhelming probability. We introduce more backgrounds regarding smooth numbers in the extended version of our paper. The following lemma relies on the fact that the larger $a$, the harder to find a $y$-smooth random number in $[1, a]$:

**Lemma 2.** *Denote $P_1$ the probability that a uniform random integer between one and $a$ is $y$-smooth. For an integer $\Delta > a$, denote $P_2$ the probability that a uniform random integer between $\Delta$ and $\Delta + a$ is $y$-smooth. Then, $P_2 \leq P_1$.*

Suppose the prover uses a random number $m$ to hide the exponent. If the verifier sends the challenge $\gamma$, then $m+\gamma$ has a bit length at least close to $\gamma$. If $\gamma$ is large enough to be non-smooth with overwhelming probability, then $m+\gamma$ is also non-smooth with overwhelming probability. We extend Fact 2 to the following fact with two practical parameters of $\gamma$ as examples based on Lemma 2 and the analysis of Coron and Naccache [58]:

Params: security parameter $\lambda$, $g \in \mathbb{G}_?$, $B > \text{maxord}(\mathbb{G})2^\lambda$; Inputs: $C_1 \in \mathbb{G}_?$, $n, e \in \mathbb{Z}$, $\hat{x} \in [n]$; Witness: $x \in \mathbb{Z}$; Claim: $C_1 = g^x$ and $x^e \bmod n = \hat{x}$.

**ZK-PoMoDE**

1. Prover samples $m \xleftarrow{\$} [B]$ and computes $D = g^m$, $\pi_1 = \textbf{NI-PoKE}^*(g, D; m)$, $C_2 = g^{(x+mn)^e}$, $\pi_2 = \textbf{NIZK-PoKDE}(C_1 \times D^n, C_2, e; x+mn)$, $\pi_3 = \textbf{NIZK-PoKEModN}(g, C_2, n, \hat{x}; (x+mn)^e)$.
It sends $(D, C_2, \pi_1, \pi_2, \pi_3)$ to the verifier.
2. Verifier accepts if all the following are correct: $1 = \textbf{NI-PoKE}^*(g, D, \pi_1)$, $1 = \textbf{NIZK-PoKDE}(C_1 \times D^n, C_2, e, \pi_2)$ and $1 = \textbf{NIZK-PoKEModN}(g, C_2, n, \hat{x}, \pi_3)$.

Figure 3: Protocol **ZK-PoMoDE**.

- Fact 3 For committed $x, c$ and a 1024-bit (2048-bit) challenge $\gamma$ with smoothness $2^{80}$ ($2^{120}$), the binomial-type polynomial of the form $(x+\gamma)^n - c^n$ divides $\gamma$ when $c = x$. If $c \neq x$, then $(x+\gamma)^n - c^n$ divides $\gamma$ with probability less then $1/2^{80}$ ($1/2^{120}$).

We describe the **ZK-PoKDE** protocol in Figure 2 based on the intuition introduced in Fact 3, and capture its security by the following theorem:

**Theorem 1.** *Protocol* **ZK-PoKDE** *is a statistically special-honest verifier zero-knowledge proof of knowledge for relation $\mathcal{R}_{\textbf{PoKDE}}$ if protocol* **NI-PoE** *is an argument of exponent for the relation $\mathcal{R}_{\textbf{PoE}}$, protocol* **NI-PoKDE** *is an argument of knowledge for relation $\mathcal{R}_{\textbf{PoKDE}}$, protocol* **NIZK-PoKE2** *is a statistically zero-knowledge argument of knowledge of exponent for the relation $\mathcal{R}_{\textbf{PoKE2}}$, uniform random numbers in $[B]$ are $2^\lambda$-smooth.*

The intuition to prove Theorem 1 is that: (1) because of the bitlength of $\gamma$, $(m+\gamma)$ is non-smooth with overwhelming probability; (2) based on **NIZK-PoKE2**$(g^{m+\gamma}, F; \omega')$, the extractor can extract the integer committed in $F$ that divides $m+\gamma$; (3) $E/K = F = g^{\omega' \times (m+\gamma)}$, because $E = g^{(x\ell+m+\gamma)^e}$ and $K = C_2^{\ell^e}$, $(m+\gamma)|(\log E/K)$, based on Fact 3, the integer committed in $C_2$ is $x^e$ with overwhelming probability. We present our **ZK-PoMoDE** protocol in Figure 3 by combining the **ZK-PoKEModN** protocol and the **ZK-PoKDE** protocol with additional random numbers. The intuition of the **NIZK-PoMoDE** protocol is: (1) committing random number $D = g^m$; (2) using **ZK-PoKDE** protocol to prove $C_2$ commits $(x+mn)^e$ in zero-knowledge; (3) using **NIZK-PoKEModN** protocol to show $(x+mn)^e \bmod n = \hat{x}$.

## 5. Verifiable Time-lock puzzles

In this section, we present our construction for a *verifiable time-lock puzzle (VTLP)* for general NP relations. Compared to "classic" time-lock puzzles (e.g., [1], [3]), a VTLP is enhanced by facilitating the puzzle generator to prove that the solution to a puzzle is compliant with a predetermined NP relation without leaking the solution. Upon the validation of such a proof, the verifier is assured

that the execution of exactly $\mathcal{T}$ computational steps will yield a solution to the puzzle that, moreover, *conforms to the specified NP relation*. In practice, such relations may capture validated signatures on the solution, checking that it is a well-formed VRF evaluation or hash pre-image, or any arbitrary computation on the solution in the general case.

First, we present a formal definition of VTLPs and their security properties. We then provide a "strawman" solution from SNARKs and discuss limitations for use in practice. We then describe *offloading techniques* to offload the computations of set products and modular exponentiations from the computed circuit as an essential building block. Subsequently, in Subsection 5.3, we provide our VTLP construction that can be used to prove solutions belong to any NP relation—while this still entails a SNARK, our above building blocks provide improved performance compared to the strawman approach. Finally, in Subsection 5.4, we delineate a "simple" VTLP construction for the specific problem of proving that the solution is valid VRF-evaluation/RSA signature. Compared to our general-purpose VTLP, this scheme does not rely on a SNARK to check this relation, instead utilizing our proofs for modular exponentiations in hidden order groups from Section 4.

### 5.1. Definitions and Security

Our VTLP definition adapts those of classic time-lock puzzles [1], [3] with suitable modifications to capture the verifiability of solutions for arbitrary NP relations. Formally, a verifiable time-lock puzzle (VTLP) for NP comprises the following six algorithms:

- **KeySetup**$(1^\lambda) \to (pk, sk)$: a probabilistic algorithm that takes as input security parameter $1^\lambda$ and outputs a public/private key pair $(pk, sk)$.
- **GenPuz**$(s, \mathcal{T}, pk, sk) \to z$: a probabilistic algorithm that takes as input a solution $s$, hardness parameter $\mathcal{T}$ and public/private key pairs $(pk, sk)$, and outputs a puzzle $z$.
- **SolvPuz**$(pk, z, \mathcal{T}) \to s$: a deterministic algorithm that takes as input the public key $pk$, puzzle $z$ and hardness parameter $\mathcal{T}$, and outputs a solution $s$.
- **FuncSetup**$(1^\lambda, \mathcal{R}) \to$ crs: a deterministic algorithm that takes as input security parameter $1^\lambda$, the relation $\mathcal{R}$ and outputs a common reference string crs to descript the relation. This function is run by a trusted party. Note that this step is optional when using a SNARK that does not require pre-processing or the relation is easy to check.
- **Prove**$(\text{crs}, z, s, \mathcal{T}, pk, sk, x, w) \to (\pi_\mathcal{T}, \pi_\mathcal{R})$: a probabilistic algorithm that takes as input a common reference string crs, a puzzle $z$, a solution $s$, a hardness parameter $\mathcal{T}$, a pair of key $(pk, sk)$, a public input $x$ and a private witness $w$ s.t. $\mathcal{R}(x, s; w) = 1$. It outputs a pair of proofs $(\pi_\mathcal{T}, \pi_\mathcal{R})$. $\pi_\mathcal{R}$ is used for verifying the relation $\mathcal{R}$. The verifier can verify $\mathcal{R}(x, s, \pi_\mathcal{R}) = 1/0$ without seeing the witness $w$.
- **Verify**$(\text{crs}, z, \mathcal{T}, pk, x, \pi_\mathcal{T}, \pi_\mathcal{R}) \to 1/0$: a deterministic algorithm that takes as input a common reference string crs, a puzzle $z$, a hardness parameter $\mathcal{T}$, a public key $pk$, a public input $x$ and a proof $\pi$. It outputs 1 if all proofs are verified and 0 otherwise.

VTLPs must satisfy *correctness* and *time-lock security*. Their definitions are straightforward adaptations of those of classic time-lock puzzle [1], [3], [5], so we defer them to the extended version of our paper. Additionally, verifiable time-lock puzzles must satisfy *soundness* and *zero-knowledge*.

The soundness property ensures that if a proof $\pi$ is verified against a puzzle $z$, then the solution $s$ after solving the puzzle for time $\mathcal{T}$, is in the NP relation $\mathcal{R}$ s.t. $\mathcal{R}(x, s, \pi_\mathcal{R}) = 1$. In practice, this is extremely useful to stop an adversary from launching a Denial of Service (DoS) attack, that fools the verifier into solving a "*fake*" time-lock puzzle and getting a solution not in the desired relation.

**Definition 1** (Soundness)**.** *For all $\mathcal{T} = poly(\lambda)$ and any relation $\mathcal{R}$ in NP language with additional input and witness $x, w$ s.t. $\mathcal{R}(x, s; w) = 1$, for all $PPT$ adversaries $\mathcal{A}$ that runs in time $poly(\mathcal{T})$, a VTLP is sound if*

$$\Pr\left[\begin{array}{c} \text{crs} \leftarrow \textbf{FuncSetup}(1^\lambda, \mathcal{R}), \\ (z, \mathcal{T}, pk, x, \pi) \leftarrow \mathcal{A}(1^\lambda, \text{crs}), \\ 1 \leftarrow \textbf{Verify}(\text{crs}, z, \mathcal{T}, pk, x, \pi_\mathcal{T}, \pi_\mathcal{R}), \\ \bar{s} \leftarrow \textbf{SolvPuz}(pk, z, \mathcal{T}), \\ 0 \leftarrow \mathcal{R}(x, \bar{s}, \pi_\mathcal{R}) \end{array}\right] \leq \text{negl}(\lambda)$$

Zero-knowledge ensures that the proof does not reveal additional information about the solution or the witness (clearly, this holds *before* the solution is computed).

**Definition 2** (Zero-knowledge)**.** *For any NP relation $\mathcal{R}$, denote* $\textbf{Real}_{\textbf{Adv}}(1^\lambda)$, $\textbf{Ideal}_{\textbf{Adv,Sim}}(1^\lambda)$ *as games between a challenger, an adversary* **Adv** *and a simulator* **Sim***, defined as:*

$\textbf{Real}_{\textbf{Adv}}(1^\lambda)$*:*

- ***Setup.*** *The challenger runs* $\text{crs} \leftarrow \textbf{FuncSetup}(1^\lambda, \mathcal{R})$ *and* $(pk, sk) \leftarrow \textbf{KeySetup}(1^\lambda)$ *and forwards* $\text{crs}, pk$ *to* **Adv**. **Adv** *chooses* $(x, s, w)$ *s.t.* $\mathcal{R}(x, s; w) = 1$ *and sends* $(s, x, w)$ *to the challenger. The challenger runs* $z \leftarrow \textbf{GenPuz}(s, \mathcal{T}, pk, sk)$*, and* $(\pi_\mathcal{T}, \pi_\mathcal{R}) \leftarrow \textbf{Prove}(\text{crs}, z, s, \mathcal{T}, pk, sk, x, w)$ *and replies with* $(z, \pi_\mathcal{T}, \pi_\mathcal{R})$.
- ***Respond.*** **Adv** *outputs a bit $b$.*

$\textbf{Ideal}_{\textbf{Adv,Sim}}(1^\lambda)$*:*

- ***Setup.*** *The simulator* **Sim** *inputs* $(1^\lambda)$ *and forwards* $(pk, \text{crs})$ *to* **Adv**. **Adv** *chooses* $(s, x, w)$ *s.t.* $\mathcal{R}(x, s; w) = 1$ *and sends* $(s, x)$ *to the simulator. The simulator responds with* $(z, \pi_\mathcal{T}, \pi_\mathcal{R})$ *to the* **Adv**.
- ***Respond.*** **Adv** *outputs a bit $b$.*

*A VTLP is zero-knowledge if there exists a $PPT$ simulator* **Sim** *such that for all adversaries* **Adv** *running in in less than $\mathcal{T}$ computational steps:*

$$|\Pr[\textbf{Real}_{\textbf{Adv}}(1^\lambda) = 1] - \Pr[\textbf{Ideal}_{\textbf{Adv,Sim}}(1^\lambda) = 1]| \leq \text{negl}(\lambda).$$

We say a VTLP is *succinct* if checking the proof is more efficient than solving the puzzle (and checking the relation).

Public params $pp = (\text{security parameter } \lambda,\ g \in \mathbb{G}_?)$
**Setup**$(\lambda)$
**return** crs $\leftarrow \Pi.\textbf{Setup}(1^\lambda, \mathcal{R}_{\textbf{AllMod}})$
**Prove**$(pp, \text{crs}, \vec{v}, \boxed{c_{\vec{v}}}, A, x)$
$\ell = \textbf{HashToPrime}(pp, \text{crs}, A, \boxed{c_{\vec{v}}})$,
$q = \lfloor x/\ell \rfloor, r = x \bmod \ell, Q = g^q$, $\pi_1 = (Q, r)$,
$\pi_2 \leftarrow \Pi.\textbf{Prove}(\text{crs}, \boxed{c_{\vec{v}}}, \ell, r)$, **return**$(\pi_1, \pi_2)$.
**Verify**$(pp, \text{crs}, \boxed{c_{\vec{v}}}, A, x, \pi_1, \pi_2)$
$\ell = \textbf{HashToPrime}(pp, \text{crs}, A, \boxed{c_{\vec{v}}})$, parse $\pi_1$ as $(Q, r)$
Check if $1 \leftarrow \Pi.\textbf{Verify}(\text{crs}, \boxed{c_{\vec{v}}}, \ell, r, \pi_2)$ and $Q^\ell g^r = A$. **return** 1 if all check passed and 0 otherwise.

Figure 4: Protocol **OffloadProd**, a protocol for offloading set product from SNARK, proving $A = g^x, x = \prod_{\forall v_i \in \vec{v}} v_i$.

**Definition 3** (Succinctness)**.** *A VTLP is succinct if $\pi_\mathcal{T}, \pi_\mathcal{R}$ have size and executing* **Verify** *takes time at most polylogarithmic in $(\mathcal{T}, w)$.*

**Strawman solutions based on black-box SNARK use.** As already discussed in our introduction, a straightforward approach to build a VTLP is to combine any TLP with a general-purpose SNARK, for a circuit running **SolvPuz** in $\mathcal{T}$ steps and checking the solution satisfies $\mathcal{R}$. Unfortunately, the performance of this approach quickly deteriorates as $\mathcal{T}$ grows. For instance, a recent benchmark [6] with the RSA TLP [1] shows that securing a puzzle for just one hour requires setting $\mathcal{T}$ to approximately $2^{31}$. A circuit encoding a 2-billion-step computation is arguably beyond the practical limitations of most modern SNARKs.

For some TLPs, there may be faster ways to check the correct execution of **SolvPuz** inside the SNARK, exploiting the trapdoor information available to the generator. For instance, consider the widely-used repeated square-based RSA TLP [1]. For RSA public key $N = pq$, checking $z^{2^\mathcal{T}} \bmod N$ is *equivalent* to checking $z^{2^\mathcal{T} \bmod \phi(N)} \bmod N$ (assuming the circuit tests the correctness of $\phi(N)$ for $N$).[2] However, even with this optimization, the SNARK circuit is still concretely large. As discussed in the introduction, using the SNARK circuit to check roughly 2048 squaring and multiplications takes approximately 72 *million* constraints.

## 5.2. Building Block: Offloading set products and modular exponentiations from SNARK

This subsection presents two protocols for the specific problem "offloading" set product and modular exponentiation computation from within a SNARK circuit. We employ "commit-and-prove" CP-SNARK techniques [55] to get a commitment of the witnesses as the first step and then combine different SNARKs. We assume the commitment scheme that is computational binding and perfect hiding, e.g., like Pedersen commitment [59] and most vector commitments used for CP-SNARK [60]. For simplicity of presentation, we describe non-interactive versions of the protocols, by using a **HashToPrime** function that is collision-resistant, modeled as a random oracle, and outputs $2\lambda$-bit prime numbers.[3]

**Offloading set products.** The general idea of our offloading technique is to employ a hidden order group to generate a commitment $A$ (with a single base $g$ for efficiency; this is similar to an RSA accumulator [27], [55], [61], but we will not need membership proofs) to commit to the set. We then prove that $A$ and $\boxed{c_{\vec{v}}}$ commit the same set, where $\boxed{c_{\vec{v}}}$ is a CP-SNARK-compliant commitment scheme committing to $\vec{v}$, and $\vec{v}$ is an appropriate encoding of the set for the chosen SNARK (e.g., as vector of elements for [62]). We implement this by checking the products of the two pre-images modulo a prime challenge $\ell$ have the same remainder $r$. Similar to Fact 1 (for case $e = 1$), if they are not the same, they have the same remainder with probability less than $1/\ell$. Note that $A$ directly encodes the product of all accumulated integers in the set. We provide our protocol details in Figure 4. More specifically, we "offload" by proving the following relation using the combination of **PoKE** protocols and CP-SNARKs:

$$\mathcal{R}_{\textbf{OffloadProd}} = \{(\boxed{c_{\vec{v}}}, A \in \mathbb{G}_?; x) : A = g^x, x = \prod_{\forall v_i \in \vec{v}} v_i\}.$$

The SNARK circuit itself only needs to check: $r = \prod_{\forall v_i} v_i \mod \ell$. Without loss of generality, we assume the prime challenge $\ell$ resides within the SNARK system's native prime field. Compared to the strawman solution, this simplified circuit design reduces the computational overhead by only checking a single remainder within the prime field for each integer and calculating their cumulative product modulo $\ell$. We denote the above SNARK relation as $\mathcal{R}_{\textbf{AllMod}}$, where the SNARK circuit computes:

1) Initiate $r = 1$;
2) Ranging all $v_i$, $r = r \times v_i \bmod \ell$.

We note that this trick was used in [63] to prove a set of integers committed in an RSA accumulator is the same as the one in CP-SNARK in the context of a MultiSwap application. Note that the **OffloadProd** protocol described above can check integer products more efficiently than naive methods. However, when the integers are larger than the native field of SNARKs (particularly in the RSA setting), the checking process remains expensive. In the next part, we will demonstrate how to address this issue in the context of integer modular exponentiations.

**Offloading modular exponentiations.** We present a protocol to offload modular exponentiations from within a SNARK. More specifically, we prove the following relation using the combination of **PoKE** protocol, **ZK-PoKEModN** protocol, and a SNARK, where $x$ is committed by $\boxed{c_x}$ using any CP-SNARK commitment scheme:

$$\mathcal{R}_{\textbf{OffloadExp}} = \{(u, C \in \mathbb{Z}_N^*, \boxed{c_x}) : C = u^x \bmod N\}.$$

2. Note that such an "optimized" approach for checking **SolvPuz** may not be possible for any *arbitrary* TLP. E.g., there is nothing in the TLP definition that precludes the possibility that an adversarial generator can provide a "solution" as SNARK witness that passes the check but is different from what is computed by honestly running **SolvPuz**.

3. We stress that this is just for presentation purposes; we provide the proof of the interactive versions of our schemes. Since their verification is public-coin the Fiat-Shamir heuristic for the hash function yields their non-interactive versions.

Public params $pp =$ (security parameter $\lambda$, $g \in \mathbb{G}_?$)
**Setup**($\lambda$)
**return** crs $\leftarrow \Pi$.**Setup**$(1^\lambda, \mathcal{R}_{\textbf{SelectMod}})$
**Prove**$(pp, \text{crs}, N, u, \boxed{c_x}, C)$
Parse $x$ to binaries $(x_0, \ldots, x_{|N|-1})$
For $i \in [|N|]$, $v_i = u^{2^i} \mod N$, $\omega = \prod v_i^{x_i}$, $A = g^\omega$,
$\ell =$ **HashToPrime**$(pp, \text{crs}, N, u, C, A, \boxed{c_x})$.
For $i \in [|N|]$, $w_i = v_i \mod \ell$,
$q = \lfloor \omega / \ell \rfloor, r = \omega \mod \ell, Q = g^q$, $\pi_1 = (Q, r)$,
$\pi_2 \leftarrow \Pi$.**Prove**$(\text{crs}, \boxed{c_x}, \ell, r, w_0, \ldots, w_{|N|-1})$,
$\pi_3 \leftarrow$ **NIZK-PoKEModN**$(g, A, N, C; \omega)$
**return**$(\boxed{c_x}, A, \pi_1, \pi_2, \pi_3)$,
**Verify**$(pp, \text{crs}, N, u, \boxed{c_x}, C, A, \pi_1, \pi_2, \pi_3)$
$\ell =$ **HashToPrime**$(pp, \text{crs}, N, u, C, A, \boxed{c_x})$,
For $i \in [|N|]$, $v_i = u^{2^i} \mod N$, $w_i = v_i \mod \ell$.
Parse $\pi_1$ as $(Q, r)$, check if $Q^\ell g^r = A$.
Check if $1 \leftarrow \Pi$.**Verify**$(\text{crs}, \boxed{c_x}, \ell, r, w_0, \ldots, w_{|N|-1}, \pi_2)$
Check if $1 \leftarrow$ **NIZK-PoKEModN**$(g, A, N, C, \pi_3)$.
**return** 1 if all check passed and 0 otherwise.

Figure 5: Protocol **OffloadExp**, a protocol for offloading modular exponentiations from SNARK, proving $C = u^x \mod N$.

As we explain next, such a protocol becomes especially challenging when $N$ is an RSA modulus *generated by the prover*, even if we ignore the requirement of $x$ being committed using SNARK-friendly commitment. Although this relation looks similar to $\mathcal{R}_{\textbf{PoKE}}$, it turns out it is very hard to prove using directly **PoKE**-style protocols. This is because the prover knows the order of the group $\mathbb{Z}_N^*$; hence, the adaptive root assumption does not hold for $\mathbb{Z}_N^*$. For our VTLP, we will only use this protocol for an RSA modulus $N$, we focus on this case during the presentation (our analysis trivially extends to arbitrary natural numbers).

We first simplify $u^x \mod N$ to prove this relation efficiently using the square-and-multiply algorithm. We denote $x$ using its binary form: $x = x_0 + x_1 \times 2 + x_2 \times 4 + \cdots + x_{|N|-1} \times 2^{|N|-1}$ with each $x_i \in \{0, 1\}$. Because $x \in \phi(N)$, all $x_i = 0$ when $i > |\phi(N)|$. We write till $|N-1|$ for simplicity of explanation. $u^x \mod N$ can then be computed as $(u^{2^0})^{x_0} \times (u^{2^1})^{x_1} \times \cdots \times (u^{2^{|N|-1}})^{x_{|N|-1}} \mod N$. Thus, the prover can input $(v_0 = u^{2^0} \mod N, v_1 = u^{2^1} \mod N, \ldots, v_{|N|-1} = u^{2^{|N|-1}} \mod N)$ into the SNARK circuit that applies the above equation to compute $u^x \mod N$ with $N$ multiplications. Based on our set product offloading techniques above, we can further offload this computation, combining it with a hidden order group element $g \in \mathbb{G}_?$, **PoKE**$^*$, **ZK-PoKEModN** protocol and a simple SNARK circuit checking subset selection based on $x$ and their cumulative product modulo $\ell$. The prover first commits to $x$ using $\boxed{c_x}$ and then commits $\omega = \prod v_i^{x_i}$ using a hidden order group $A = g^\omega$. It then inputs $(\boxed{c_x}, A)$ and the remaining parts of the statement to **HashToPrime** to get the prime challenge $\ell$. The SNARK circuit needs to check the product of the "selected" $v_i$ modulo $\ell$ is $r$, where $r$ is also the remainder of $\omega$ modulo $\ell$.

Moreover, because $\ell$ and all $v_i$ are public, $v_i$ modulo $\ell$ can be computed outside the SNARK to simplify the circuit further. More specifically, both the prover and the verifier compute $w_i = v_i \mod \ell$, and the SNARK circuit takes as input public values $(\ell, r, w_0, w_1, \ldots, w_{|N|-1})$ with witness $x$. Finally the circuit for relation $\mathcal{R}_{\textbf{SelectMod}}$ computes:

1) Initiate $r = 1$; parse $x$ as its binary form;
2) Ranging $x_i$ from $x_0$ till $x_{|N|} - 1$, if $x_i = 1$, $r = r \times w_i \mod \ell$.

We denote the protocol as **OffloadExp** and depict its details in Figure 5. It is now evident that the circuit checks at most $|N| - 1$ *native field* modular multiplications. Recall that we assume $\ell$ falls within the native field size of the SNARK (from the analysis of [27], this does not impact the protocols' security).

Finally, we further extend protocol **OffloadExp** to support the relation $\mathcal{R}_{\textbf{OffloadExpCom}}$ where the exponentiation result is committed using a hidden order group:

$$\mathcal{R}_{\textbf{OffloadExpCom}} = \{(u \in \mathbb{Z}_N^*, D \in \mathbb{G}_?, \boxed{c_x}) : D = g^{u^x \mod N}\}.$$

This modified protocol **OffloadExpCom** has similar steps and security properties as **OffloadExp**, except that $C = u^x \mod N$ is no longer provided but committed in a hidden order group element $D$.

### 5.3. Our construction

Next, we present our VTLP protocol for general NP relations. Without loss of generality, we assume the bit-length of the solution $s$ is less than $2\lambda$ (larger $s$ can always be hashed down to $2\lambda$ bits and the SNARK in our VTLP will also "open" the hash).

In the context of a SNARK for a general relation $\mathcal{R}(x, s; w) = 1$, we denote $s$ as the SNARK "output" to specifically separate it from statement $x$. Our methodology unfolds in two primary parts. The initial part involves the secure encoding of the SNARK output within an integer commitment scheme that is compatible with SNARKs [27], [63], ensuring that the actual output' $s$ remains confidential. The second part employs our **OffloadExpCom** protocol to prove that, after executing exactly $\mathcal{T}$ computational steps of the puzzle, the outcome is an integer that matches the one previously committed. It is noteworthy that for most SNARKs, the output is conventionally encoded via a commitment scheme such as Pedersen [59] or polynomial commitments [60]. However, directly integrating these commitments with our **OffloadExp** to get the committed version of the protocol presents significant challenges. For efficiency, we encode the SNARK output using a hidden order group element, which is consistent with protocol **OffloadExp**.

**Committing SNARK output with hidden order groups.** For a CP-SNARK that checks $\mathcal{R}(x, s; w) = 1$, the puzzle generator additional samples a random number that is well-spread: $\nu \xleftarrow{\$} [2^{4\lambda}]$. The puzzle generator commits the

Public params $pp = (\text{security parameter } \lambda, \mathcal{T}, g \in \mathbb{G}_?)$
**Setup**$(\lambda)$
**return** $\text{crs}_1 \leftarrow \Pi.\textbf{Setup}(1^\lambda, \mathcal{R}_{\textbf{SelectMod}})$
**KeySetup**$(\lambda)$
Sample large safe primes $p' = 2p+1, q' = 2q+1$, $N = p'q'$, **return** $pk = N$, $sk = (p, q)$
**TimeParaSetup**$(\lambda, sk, \mathcal{T})$
Compute $\tau = 2^{\mathcal{T}} \bmod pq$, $u' = H(\lambda||\mathcal{T}||N) \bmod N, u = (u')^2 \bmod N$. Commit $\tau$ in $\boxed{c_\tau}$, compute $\mu = u^\tau \bmod N$, $\pi_{\textbf{Setup}} \leftarrow \textbf{OffloadExp}(\text{crs}_1, N, u, \boxed{c_\tau}, \mu)$
**return** $(\boxed{c_\tau}, \pi_{\textbf{Setup}})$
**GenPuz**$(s, \mathcal{T}, pk, sk)$
Sample $\nu \xleftarrow{\$} [2^{4\lambda}]$, s.t. $s' = s||\nu, s' \in QR_N$, $\eta = \tau^{-1} \bmod pq$, $z = (s')^\eta \bmod N$, **return** $z$
**SolvPuz**$(pk, z, \mathcal{T})$
$s = \lfloor (z^{2^{\mathcal{T}}} \bmod N)/2^{4\lambda} \rfloor$
**return** $s$
**FuncSetup**$(1^\lambda, \mathcal{R})$
$\text{crs}_2 \leftarrow \Pi.\textbf{Setup}(1^\lambda, \mathcal{R}^+)$
**return** $\text{crs}_2$
**Prove**$(\text{crs}_1, \text{crs}_2, \boxed{c_\tau}, z, s', \mathcal{T}, pk, x, w)$
\\ $(x, w)$ is reserved for any potential relation $\mathcal{R}$
Commit $s'$ in $\boxed{c_{s'}}$, $D = g^{s'}$,
$\ell = \textbf{HashToPrime}(pp, \text{crs}_1, \text{crs}_2, D, \boxed{c_{s'}}, pk, x, z)$,
$q = \lfloor s'/\ell \rfloor, r = s' \bmod \ell, Q = g^q$, $\pi_1 = (Q, r)$,
$\pi_{\mathcal{R}^+} \leftarrow \Pi.\textbf{Prove}(\text{crs}_2, \boxed{c_{s'}}, x, \ell, r; w)$,
$\pi_2 \leftarrow \textbf{OffloadExpCom}(\text{crs}_1, N, z, \boxed{c_\tau}, D; \tau)$
\\ Proving $D = g^{z^\tau \bmod N}$ where $\tau = 2^{\mathcal{T}} \bmod pq$
**return** $(\pi_{\mathcal{T}} = (\pi_1, D, \pi_2), \pi_{\mathcal{R}^+})$
**Verify**$(\text{crs}_1, \text{crs}_2, z, \boxed{c_{s'}}, \mathcal{T}, pk, x, \pi_{\mathcal{T}}, \pi_{\mathcal{R}^+})$
$\ell = \textbf{HashToPrime}(pp, \text{crs}_2, D, \boxed{c_{s'}}, pk, x, z)$.
Parse $\pi_1$ as $(Q, r)$, check if $Q^\ell g^r = D$.
Check if $1 \leftarrow \textbf{OffloadExpCom}(\text{crs}_1, N, z, \boxed{c_\tau}, D, \pi_2)$.
Check if $1 \leftarrow \Pi.\textbf{Verify}(\text{crs}_2, \boxed{c_{s'}}, x, \ell, r, \pi_{\mathcal{R}^+})$.
**return** 1 if all check passed and 0 otherwise.

Figure 6: Protocol **VTLP** for NP.

SNARK "output" by concatenating $s$ and $\nu$: $C = g^{s||\nu}$ without eliminating potential leading '0's of $\nu$. In other words, the commitment $C = g^{s||\nu} = g^{s \times 2^{4\lambda} + \nu}$. We additionally require $\text{minord}(\mathbb{G}_?) > s||\nu$ holds, which is typically true for all reasonable choices of RSA-style parameters for hidden order groups when $s$ has $2\lambda$ bits. We can "connect" $C$ and $s$ inside the SNARK using the **PoKE**$^*$ protocol, similar to our offloading technique, to prove the integer committed in $C$ is the same as in the CP-SNARK. We denote $s' = s||\nu$. In the interactive version, the prover commits $C$ and $\boxed{c_{s'}}$ first, then the verifier samples prime challenge $\ell$. The prover responds with $Q$, $r$ and a SNARK proof showing $s' \bmod \ell = r$. We show this randomization is sufficient to hide $s$, based on the DDH-II assumption over hidden order groups [63].

**Construction details.** A "classic" RSA-style time-lock puzzle [1] is secure in any subgroup of $\mathbb{Z}_N^*$ with *large enough* order. For an RSA public key $N = p'q'$ where $p' = 2p+1$ and $q' = q+1$, its subgroups with order $pq, 2pq, 4pq$ are all considered to be secure for time-lock puzzles. For simplicity of construction and analysis, we use the subgroup of $QR_N$ with order $pq$ to build our VTLP. Figure 6 presented our detailed construction. Initially, we require a one-time setup for each puzzle generator to set up its VTLP keys $pk, sk$. For each hardness parameter $\mathcal{T}$ and an RSA public key $N = p'q'$, the puzzle generator additionally runs the function **TimeParaSetup** to prove that a commitment $\boxed{c_v}$ commits $2^{\mathcal{T}} \bmod pq$. Because checking this directly using a SNARK is very inefficient, we simplify this by a reduction to fixed-sized exponentiation. First, the puzzle generator samples a random element in $u \in QR_N$ by hashing $u' = H(\lambda||\mathcal{T}||N) \bmod N$ where $H(\cdot)$ is a cryptographically secure hash function (modeled as a random oracle) which outputs to a domain larger than $N$ and $u = (u')^2 \bmod N$. This step stops the puzzle generator from maliciously picking a generator $u \notin QR_N$. As proved in Lemma 1 in [64], $u$ is a generator in $QR_N$ if and only if $\gcd(u-1, N) = 1$. Then, it computes $\mu = u^{2^{\mathcal{T}}} \bmod N$, which can be simplified by first calculating $\tau = 2^{\mathcal{T}} \bmod pq$ first and then computing $\mu = u^\tau \bmod N$. Finally, it commits to $\tau$ using CP-SNARK commitment $\boxed{c_\tau}$ and further uses our **OffloadExp** protocol to prove $u^\tau \bmod N = \mu$. $\boxed{c_\tau}$ and $\pi_{\textbf{Setup}}$ are published as public parameter and $\pi_{\textbf{Setup}}$ needs to be verified at least once. Verifiers can compute $u$ in the same way, compute $\mu = u^{2^{\mathcal{T}}} \bmod N$ and verify $\pi_{\textbf{Setup}}$. To prove the puzzle solution satisfies the chosen relation, we rely on a SNARK that checks the relation $\mathcal{R}(x, s; w) = 1$ and checks $s||\nu \bmod \ell = r$. We denote this SNARK relation as $\mathcal{R}^+$, which is set up by **FuncSetup**. The following theorem captures the security of our construction:

**Theorem 2.** *The protocol presented in Figure 6 satisfies the VTLP security definitions for relation $\mathcal{R}$ if protocol* **PoKE**$^*$ *is an argument of knowledge for the relation $\mathcal{R}_{\textbf{PoKE}^*}$,* **OffloadExp** *is an argument of knowledge for relation $\mathcal{R}_{\textbf{OffloadExp}}$,* **OffloadExpCom** *is an argument of knowledge for relation $\mathcal{R}_{\textbf{OffloadExpCom}}$, $\Pi.\mathcal{R}^+(\boxed{c_{s'}}, x, \ell, r; w, s)$ is a secure CP-SNARK and DDH-II assumption and order assumption hold for group $\mathbb{G}_?$.*

The intuition of Protocol **VTLP** for NP is using the **OffloadExpCom** protocol to show the solution raised to the power of $\tau \bmod N$, which is equivalent to raising the power of $2^{\mathcal{T}} \bmod N$, is the exponent $s'$, committed in $D$.

## 5.4. Verifiable Time-lock for VRFs/Signatures

In addition to our above general-purpose VTLP construction that accommodates any NP language, we propose a specialized and efficient protocol tailored specifically for statements involving RSA-based verifiable random functions (VRFs) or signatures over the solution to the puzzle. This customized VTLP can be used to prove the puzzle conceals a valid VRF evaluation or signature for a given context/message related to a public RSA key. Notably, this protocol achieves this *without the need for SNARKs*.

The intuition behind this construction is as follows. Assume the standard RSA-based VRF [33], [34], [35] or RSA signature schemes with full domain hash [65]. The

Public params $pp =$ (security parameter $\lambda$, $g \in \mathbb{G}_?$)
**Setup**($\lambda$)
**return** crs $\leftarrow \Pi$.**Setup**$(1^\lambda, \mathcal{R}_{\textbf{SelectMod}})$
**KeySetup**($\lambda$)
Sample large safe primes $p' = 2p+1, q' = 2q+1$, $N = p'q'$, **return** $pk = N$, $sk = (p, q)$
**TimeParaSetup**($\lambda, sk, \mathcal{T}$)
Compute $\tau = 2^{\mathcal{T}} \bmod pq$, $u' = H(\lambda||\mathcal{T}||N) \bmod N, u = (u')^2 \bmod N$. Commit $\tau$ in $\boxed{c_\tau}$, compute $\mu = u^\tau \bmod N$, $\pi_{\textbf{Setup}} \leftarrow$ **OffloadExp**$(\text{crs}_1, N, u, \boxed{c_\tau}, \mu)$
**return** ($\boxed{c_\tau}$, $\pi_{\textbf{Setup}}$)
**KeySetupForVRF/Sig**($\lambda$)
Samples two large safe primes $\bar{p}, \bar{q}$, $\bar{N} = \bar{p}\bar{q}$, $\phi(\bar{N}) = (\bar{p}-1)(\bar{q}-1)$, find $\bar{d}$, $\bar{e}$ s.t. $\bar{d} = \bar{e}^{-1} \bmod \phi(\bar{N})$.
**return** $\bar{pk} = (\bar{N}, \bar{e})$, $\bar{sk} = (\bar{p}, \bar{q}, \bar{d})$
**GenVRF/Sig**($\bar{sk}, m$)
$s = \textbf{FDH}(m)^{\bar{d}} \bmod \bar{N}$, **return** $s$
**GenPuz**($s, \mathcal{T}, pk, sk$)
$\eta = (2^{\mathcal{T}})^{-1} \bmod \phi(N)$
**return** $z = s^\eta \bmod N$
**SolvPuz**($pk, z, \mathcal{T}$)
$s = z^{2^{\mathcal{T}}} \bmod N$
**return** $s$
**Prove**(crs, $z, s, \mathcal{T}, pk, m, \bar{pk}$)
$D = g^s$, $\pi_1 \leftarrow$ **NIZK-PoMoDE**$(D, \bar{e}, \bar{N}, \textbf{FDH}(m); s)$
\\ Proving $s^{\bar{e}} \bmod \bar{N} = \textbf{FDH}(m)$
$\pi_2 \leftarrow$ **OffloadExpCom**$(\text{crs}, N, z, \boxed{c_\tau}, D; \tau)$
\\ Proving $D = g^{z^\tau \bmod N}$ where $\tau = 2^{\mathcal{T}} \bmod pq$
**return** $(\pi_{\mathcal{T}} = \pi_2, \pi_{\mathcal{R}} = (D, \pi_1))$
**Verify**(crs, $z, \mathcal{T}, pk, m, \bar{pk}, \pi_{\mathcal{T}}, \pi_{\mathcal{R}}$)
Check if $1 \leftarrow$ **NIZK-PoMoDE**$(D, \bar{e}, \bar{N}, \textbf{FDH}(m), \pi_1)$.
Check if $1 \leftarrow$ **OffloadExpCom**$(\text{crs}, N, z, \boxed{c_\tau}, D, \pi_2)$.
**return** 1 if all check passed and 0 otherwise

Figure 7: Protocol **VTLP** for RSA VRFs/signatures. **FDH** denotes a secure full-domain hash function.

prover evaluates the VRF $s$ (resp. computes the signature) for a specific message $m$ and commits to it as a hidden-order group element $D = g^s$. It then uses our **PoMoDE** to show the integer committed in $D$ is indeed a valid VRF/signature for a given context/message, which can be presented by the following relation:

$$\mathcal{R}_{\textbf{VTLPVRF}} = \{((g, D) \in \mathbb{G}_?, (\bar{e}, \bar{N}, m) \in \mathbb{Z}; s) : D = g^s, s^{\bar{e}} \bmod \bar{N} = \textbf{FDH}(m)\}.$$

We use the public key $\bar{e}$ as the exponent and $\bar{N}$ as the modulus, and the **PoMoDE** protocol to prove $s^{\bar{e}} \bmod \bar{N} = \textbf{FDH}(m)$. Then, we use the same technique (**OffloadExpCom**) to show the execution of exactly $\mathcal{T}$ computational steps yields the committed VRF/signature. We provide the construction details in Figure 7. Its security is captured by the following theorem:

**Theorem 3.** *The protocol in Figure 7 satisfies the VTLP security definitions for RSA VRF and signatures relations* $\mathcal{R}_{\textbf{VTLPVRF}}$ *if* **NIZK-PoMoDE** *is a zero-knowledge argument of knowledge for relation* $\mathcal{R}_{\textbf{PoMoDE}}$, **OffloadExp** *is an argument of knowledge for relation* $\mathcal{R}_{\textbf{OffloadExp}}$, **OffloadExpCom** *is an argument of knowledge for relation* $\mathcal{R}_{\textbf{OffloadExpCom}}$, *and order assumption holds for* $\mathbb{G}_?$.

The intuition to achieve the zero-knowledge property is using the existential pseudorandomness properties of VRF, which says the VRF value is indistinguishable from a random value unless you can run the verification of VRF. Because the VRF value is committed and time-locked, its commitment is indistinguishable from a random group element unless the puzzle is first solved.

Due to space limitations, we discuss new techniques for offloading RSA signature verification from SNARK based on **OffloadProd** protocol in Appendix B.

## 6. Experimental Evaluation

In this section, we experimentally evaluate the performance of our constructions. We test the performance of **PoMoDE**, VTLP for general NP, VTLP for RSA VRF/signatures, as well as our RSA signatures offloading techniques as a standalone tool. Our code is available online and anonymized for submission.[4] Our implementations are in Golang, and we used gnark [66] for Groth16-type SNARKs [23], optimized with DIZK [67] with the BN254 elliptic curve. Our Groth16 is not CP-SNARK but can be modified easily to a CP-SNARK by committing the witness first, with slight impacts on the performance [55]. We conducted on a laptop with AMD Ryzen 7 5800H, 8 3.2GHz cores, and 16GB RAM running Windows 10. We use MiMC [68] hash as the SNARK-friendly hash function. First, we benchmark our modular exponentiation protocols, focusing on prover and verifier time, as well as proof size. Specifically for **PoMoDE**, we benchmark its proving time with different exponents. Next, we evaluate our two VTLP schemes for general NP-relations (with our exponentiation offloading technique) and RSA VRF/signatures (based on our **PoMoDE** protocol). For the former, we chose specific relations to be checked by the SNARK circuit. Namely, checking the solution with (i) a MiMC hash and (ii) an EdDSA signature [69]. We also compare its performance with the time-lock signatures of Thyagarajan et al. [6]. Finally, we evaluate our RSA signatures offloading techniques, with and without zero-knowledge guarantees, as a stand-alone tool and compare its performance with other approaches for batch verification of signatures within a SNARK.

### 6.1. Proofs of Modular Exponent

We first summarize the proof sizes of our various protocols from Section 4 and show the results in Table 1. Recall that protocol **PoKDE** and **ZK-PoKDE** have a remainder $r \in [\ell \cdot n]$ in the proof where $\ell$ is a $2\lambda$-bit challenge and $n$ is the modulus. We take the example of a 128-bit $\lambda$ and 2048-bit $n$, which is also the case we use in the offloading exponentiations and signatures. Protocol **PoKDE** and **ZK-PoKDE** have constant-sized proofs regardless of the

4. https://anonymous.4open.science/r/VTLP-7578/

| Protocol | # $\mathbb{G}_?$ elem. | # Field elem. | ProofSize(Bytes) |
|---|---|---|---|
| **PoKEModN** | 1 | 1 | 544 |
| **ZK-PoKEModN** | 3 | 2 | 1088 |
| **PoKDE** | 2 | 2 | 576 |
| **ZK-PoKDE** | 13 | 6 | 3520 |
| **PoMoDE** | 17 | 9 | 4896 |

Table 1: Proof sizes of different protocols. We consider $\mathbb{G}_?$ with 2048 bits, $\lambda = 128$ and modulo $n$ with 2048 bits.

exponent to prove. Because **PoMoDE** protocol calls all of **ZK-PoKEModN**, **PoKDE**, **ZK-PoKDE**, it is the most time-consuming protocol for the prover and will be used directly for our VTLPs, we next focus on it and benchmark its performance with 2048-bit modulus $n$ and variable exponent. Table 2 showcases the prover time, averaged over five runs. The result indicates that the prover time correlates almost linearly with the size of the exponent. This empirical finding aligns with observations from the **PoMoDE** protocol, where the prover is tasked with calculating $g^{x^e}$, for exponent $e$.

It is important to note that the primary constraint in the **PoMoDE** proving process is the computation of a fixed base $g$ raised to a substantial integer power. This can be heavily optimized using precomputation tables (at the cost of increased storage) and with multiple threads; numbers reported in Table 2 do not use either of these optimizations. Finally, the verification time of **PoMoDE** protocol is constant, in practice it was measured within $29 - 32$ms.

## 6.2. VTLP Performance

Next, we evaluate the performance of VTLP for NP and VTLP for RSA VRF/signatures, which are based on our offloading modular exponentiation from the SNARK technique.

*Checking the Puzzle Solution.* First, we report the time to prove a solution is correctly concealed within a time-lock puzzle on our test machine, using the RSA-based time lock puzzle [1] (see Section 3). Checking a puzzle solution assuming a 2048-bit modulus takes $535,048$ constraints and approximately $2.08$ seconds to prove (averaged over five executions). The proof size is constantly 256 bytes (a Groth16 proof [23]) and takes around 1 ms to verify. We note that the proving/verification time as well as the proof size are independent of $\mathcal{T}$.

*Additional Relations.* Recall that our VTLP for NP is built in a modular way, such that it can be combined with any SNARK to prove the SNARK result is encoded in the puzzle with the same overhead. For example, when combined with checking the preimage of a MiMC Hash (takes around 330 constraints and 7ms to prove), we can build a VTLP for Hash-preimage with $2.087$ seconds prover time. When combined with checking one EdDSA signature (it takes around 7000 constraints and 50ms to prove), we can build a VTLP for EdDSA with a prover time of $2.13$ seconds.

*Comparison with Time-lock Signatures [6].* Specifically focusing on the VTLP for general signatures, we compare our construction with the state-of-the-art time-lock signatures

| Exponent | $3^*$ | 8 | $17^*$ | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|
| Prover time (sec) | 0.2 | 0.48 | 1.0 | 1.9 | 3.5 | 7.2 | 14.3 |

Table 2: Prover time of **PoMoDE** protocol with various exponents under 2048-bit modulus $n$. Because $3, 17$ are usually used as RSA signature/VRF public keys, we especially test them in place of $4$ and $16$.

presented by Thyagarajan et al. [6]. To maintain consistency with their parameters and reported results, we also benchmarked our VTLP using a 1024-bit modulus. For BLS signatures, our result shows that our VLTP requires $267,784$ (for RSA time-lock) plus $14,918$ (to validate a single BLS signature) constraints and approximately $1.37$ seconds in total to generate a proof, based on an average of five trials. In contrast, the estimated results presented in [6] suggest a proving time for $\mathcal{T} = 10^6$ of approximately 30 seconds and a verification time of 41 seconds. For EdDSA signatures, ours takes around $274,000$ constraints and $1.27$ seconds to prove, while the estimation of [6] shows around 10 seconds to prove and verify. Their proof sizes and verification scale linearly with the number of shares (unlike ours, which are constant). We stress that their solution suffers from non-negligible soundness error ($7.25 \times 10^{-12}$). Note that the estimates from [6] assume perfect parallelization across two threads. On the other hand, ours are derived from actual code execution on an eight-core setup, but the benefits of parallelization for Groth16 are not linear.

*Comparison with Riggs [12].* In [12], the authors benchmarked the verification of the TLP from [3] using Groth16 and a Pedersen commitment over its native field. As discussed in [3], a randomizer value of at least 2048-bits (chosen from $[1, 2^\lambda N]$, where $N$ is the hidden order modulus) should be applied as the exponent to achieve security. Riggs [12] only reports tests for a randomized of 128 bits and takes approximately 21 seconds to compute the proof. We run their code on a Linux machine with 6 cores and 40 GB of memory and we also observe $20.8$ seconds. Increasing the randomized length yields $41.2$ seconds for 256 bits, and $82.1$ for 512 bits; we were not able to run their code for larger randomizers. Erring on the conservative side, we can estimate that it will take close to 328 seconds to check a secure TLP with a 2048-bit randomizer with their approach. For comparison, our VTLP takes around $2.13$ seconds to check the solution is the pre-image of a Pedersen commitment, which is about $150\times$ more efficient than the result of [12].

*Performance of our VRF/Signatures VTLP.* Finally, we test the performance of our tailor-made VTLP for RSA-based VRFs/signatures from Section 5. For an RSA public key equal to 3 and a 2048-bit modulus, the proving time is $2.28$ seconds, while for a public key equal to 17 it is $3.08$ seconds. Recall that verifying an RSA VRF/RSA signature inside a SNARK is hard. Although it might not be as fast as checking an EdDSA/BLS/ECDSA signature inside SNARK (depending on the public key size, SNARK choices, and optimization), verifying an RSA signature is the same as verifying an RSA screening proof, which enables the

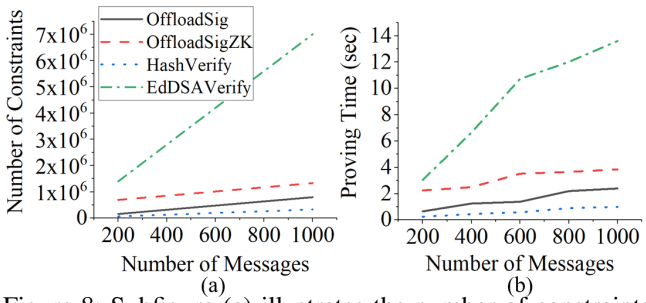


Figure 8: Subfigure (a) illustrates the number of constraints for our offloading RSA signatures techniques with and without zero-knowledge, alongside the constraints for MiMC hash verifications and EdDSA signature verifications across different numbers of messages to prove; Subfigure (b) displays the corresponding proving times for these cases.

authentication of a batch of messages via a single proof, making our VTLP for RSA signatures encompass an arbitrary amount of messages with a single proof.

### 6.3. Offloading Signatures

Finally, we evaluate the performance of our RSA signature offloading techniques as a stand-alone tool. We consider both variants, with and without zero-knowledge property. We also compare our results with the efficient batch verifications of signatures inside a SNARK, which is based on combining the EdDSA with SNARK-friendly curves, also known as BabyJubjub [39]. In both cases, messages are first hashed using MiMC. Our implementation of RSA signature verification adheres to a Hash-then-padding paradigm, also leveraging the MiMC hash function. We use RSA at 3072 bits so that the RSA signatures have the same estimated level of security with EdDSA according to NIST standards [70]. However, we stress that the choice of RSA modulo size has minimal effect on the performance of our construction (as the SNARK circuit performs a single modular operation and the overhead is strongly dominated by other parts). We use 2048 random numbers as the randomizer set for the zero-knowledge version so that the DMSS assumption holds with respect to the randomizer set and modulus 3072-bits $N$. To provide a baseline, we additionally show the cost of simply verifying the MiMC hashes in the SNARK (without signature verification). The results are presented in Figure 8, where we report the size of the circuit as number of constraints (left) and the prover time (right).

The experimental results indicate our technique is extremely efficient for offloading RSA signature verification. In practice, it remains around $2\times$ slower than simply checking the messages' MiMC hashes. Checking each RSA signature takes around 800 constraints, where 350 of the constraints are contributed by checking the MiMC hash. This result is consistent with our theoretical analysis since the prover only needs to prove one modular multiplication for each hash result. To achieve zero knowledge for the offloaded signatures, the signature offloading requires an additional $530,000$ constraints for each verification batch (approximately 1.6 seconds more for the prover). This is only related to the randomizer set size and not the number of signatures. Compared to the EdDSA+BabyJubJub approach, our prover is $1.5-4$ times faster throughout the experiment. For fairness, we note that there also exist batch verification techniques for EdDSA, but we are unaware of any batch verification of EdDSA that can easily be embedded into a SNARK with significant improvement in efficiency. There do exist such in-SNARK batch-verification techniques for ECDSA signatures [71] that can provide a 2-3$\times$ speed-up. However, due to the structure of ECDSA, it takes about 15 million constraints to verify 32 signatures, which is not as efficient as the EdDSA+BabyJubJub we tested and is entirely incomparable to our own offloading approach.

*Non-hash-based Signatures.* A recent work by Fiore and Tucker [40] proposed SPHinx, a SNARK-friendly homomorphic signature scheme, focusing on verifiable computation over authenticated (signed) data streams. Their implementation assumes messages are signed directly (without hashing them first), allowing easy computation within the SNARK. Due to this, their batched signature verification is extremely efficient. Benchmarked on a Linux server with 8 cores Xeon-Gold-6154 clocked at 3GHz and with 98 GB of RAM, their reported prover time is 0.6 seconds for 1000 messages (on our testbed our prover with RSA-3072 and 1000 messages takes 3.8 seconds). On the other hand, assuming an application that requires first hashing the messages (e.g., if messages correspond to blockchain transactions that do not fit in the underlying field of SPHinx) would impose a similar overhead as the one we reported for our MiMC baseline above. Also, the verification time of SPHinx grows with the number of messages, e.g., it requires 41ms to verify 1000 signatures (vs. one Groth16 verification in 1ms for ours).

## 7. Conclusion

In this paper, we introduce the notion of verifiable time-lock puzzles that support efficiently checking that the puzzle solution satisfies certain properties, before solvers commit resources towards solving the puzzle. We propose two constructions, one for arbitrary NP relations and one for tying the solution to an RSA signature or VRF evaluation. Our schemes are concretely efficient, as we demonstrate experimentally, and are useful in various application scenarios.

## References

[1] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.

[2] D. Boneh and M. Naor, "Timed commitments," in *CRYPTO*, 2000.

[3] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *CRYPTO*, 2019.

[4] Z. Brakerski, N. Döttling, S. Garg, and G. Malavolta, "Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles," in *Theory of Cryptography Conference (TCC)*, 2019.

[5] A. Abadi and A. Kiayias, "Multi-instance publicly verifiable time-lock puzzle and its applications," in *International Conference on Financial Cryptography and Data Security (FC)*, 2021.

[6] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder, "Verifiable timed signatures made practical," in *Proceedings of the ACM CCS*, 2020.

[7] J. Katz, J. Loss, and J. Xu, "On the security of time-lock puzzles and timed commitments," in *Theory of Cryptography Conference (TCC)*, 2020.

[8] C. Freitag, I. Komargodski, R. Pass, and N. Sirkin, "Non-malleable time-lock puzzles and applications," in *Theory of Cryptography Conference (TCC)*, 2021.

[9] S. Srinivasan, J. Loss, G. Malavolta, K. Nayak, C. Papamanthou, and S. A. Thyagarajan, "Transparent batchable time-lock puzzles and applications to byzantine consensus," in *Public Key Cryptography (PKC)*, 2023.

[10] K. Eldefrawy, B. Terner, and M. Yung, "Composing timed cryptographic protocols: Foundations and applications," *Cryptology ePrint Archive*, 2024.

[11] D. Deuber, N. Döttling, B. Magri, G. Malavolta, and S. A. K. Thyagarajan, "Minting mechanism for proof of stake blockchains," in *Applied Cryptography and Network Security (ACNS)*, 2020.

[12] N. Tyagi, A. Arun, C. Freitag, R. Wahby, J. Bonneau, and D. Mazières, "Riggs: Decentralized sealed-bid auctions," in *Proceedings of the ACM CCS*, 2023.

[13] C. Dwork, M. Naor, and A. Sahai, "Concurrent zero-knowledge," *Journal of the ACM (JACM)*, 2004.

[14] N. Bitansky and H. Lin, "One-message zero knowledge and non-malleable commitments," in *Theory of Cryptography Conference (TCC)*, 2018.

[15] L. Baird, P. Mukherjee, and R. Sinha, "i-tire: Incremental timed-release encryption or how to use timed-release encryption on blockchains?" in *Proceedings of the ACM CCS*, 2022.

[16] N. Döttling, L. Hanzlik, B. Magri, and S. Wohnig, "Mcfly: verifiable encryption to the future made practical," in *International Conference on Financial Cryptography and Data Security (FC)*, 2023.

[17] H. Lin, R. Pass, and P. Soni, "Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles," *Annual IEEE Symposium on Foundations of Computer Science FOCS*, 2017.

[18] P. Chvojka and T. Jager, "Simple, fast, efficient, and tightly-secure non-malleable non-interactive timed commitments," in *Public Key Cryptography (PKC)*, 2023.

[19] S. A. K. Thyagarajan, G. Castagnos, F. Laguillaumie, and G. Malavolta, "Efficient cca timed commitments in class groups," in *Proceedings of the ACM CCS*, 2021.

[20] A. B. Alexandru, J. Loss, C. Papamanthou, G. Tsimos, and B. Wagner, "Sublinear-round broadcast without trusted setup," *Cryptology ePrint Archive*, 2024.

[21] C. Baum, J. Hsin-yu Chiang, B. David, T. K. Frederiksen, and L. Gentile, "Sok: Mitigation of front-running in decentralized finance," in *International Conference on Financial Cryptography and Data Security (FC)*, 2022.

[22] Y. Manevich and A. Akavia, "Cross chain atomic swaps in the absence of time via attribute verifiable timed commitments," in *European Symposium on Security and Privacy (EuroS&P)*, 2022.

[23] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT*, 2016.

[24] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," *Cryptology ePrint Archive*, 2019.

[25] A. E. Kosba, C. Papamanthou, and E. Shi, "xjsnark: A framework for efficient verifiable computation," in *IEEE Symposium on Security and Privacy, SP*, 2018.

[26] R. Cramer, "Modular design of secure yet practical cryptographic protocols," *Ph. D.-thesis, CWI and U. of Amsterdam*, 1996.

[27] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *CRYPTO*, 2019.

[28] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *CRYPTO*, 1997.

[29] X. Zhou, D. He, J. Ning, M. Luo, and X. Huang, "Efficient construction of verifiable timed signatures and its application in scalable payments," *IEEE Transactions on Information Forensics and Security*, 2023.

[30] S. A. Thyagarajan, G. Malavolta, F. Schmid, and D. Schröder, "Verifiable timed linkable ring signatures for scalable payments for monero," in *ESORICS*, 2022.

[31] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979.

[32] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *ASIACRYPT*, 2001.

[33] S. Goldberg, M. Naor, D. Papadopoulos, L. Reyzin, S. Vasant, and A. Ziv, "NSEC5: provably preventing DNSSEC zone enumeration," in *Network and Distributed System Security Symposium NDSS*, 2015.

[34] M. Naor and A. Ziv, "Primary-secondary-resolver membership proof systems," in *Theory of Cryptography Conference (TCC)*. Springer, 2015, pp. 199–228.

[35] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Včelák, "Verifiable Random Functions (VRFs)," RFC 9381, Aug. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9381

[36] K. Choi, A. Manoj, and J. Bonneau, "Sok: Distributed randomness beacons," in *IEEE Symposium on Security and Privacy, SP*, 2023.

[37] A. Kate, E. V. Mangipudi, S. Maradana, and P. Mukherjee, "Flexirand: Output private (distributed) vrfs and application to blockchains," in *Proceedings of the ACM CCS*, 2023.

[38] M. Bellare, J. A. Garay, and T. Rabin, "Fast batch verification for modular exponentiation and digital signatures," in *EUROCRYPT*, 1998.

[39] B. WhiteHat, J. Baylina, and M. Bellés, "Baby jubjub elliptic curve," *Ethereum Improvement Proposal, EIP-2494*, vol. 29, 2020.

[40] D. Fiore and I. Tucker, "Efficient zero-knowledge proofs on signed data with applications to verifiable computation on data streams," in *Proceedings of the ACM CCS*, 2022.

[41] "Pkcs 2.1," https://www.rfc-editor.org/rfc/rfc8017.html, 2016, accessed: 2023-12-12.

[42] B. Wesolowski, "Efficient verifiable delay functions," in *EUROCRYPT*, 2019.

[43] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions," *Cryptology ePrint Archive*, 2018.

[44] C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner, "Tardis: a foundation of time-lock puzzles in uc," in *EUROCRYPT*, 2021.

[45] M. Ambrona, M. Beunardeau, and R. R. Toledo, "Timed commitments revisited," *Cryptology ePrint Archive*, 2023.

[46] J. Burdges and L. De Feo, "Delay encryption," in *EUROCRYPT*, 2021.

[47] J. Dujmovic, R. Garg, and G. Malavolta, "Time-lock puzzles with efficient batch solving," in *EUROCRYPT*, 2024.

[48] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *CRYPTO*, 1992.

[49] M. Girault, "Self-certified public keys," in *EUROCRYPT*, 1991.

[50] C. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, 1991.

[51] F. Boudot, "Efficient proofs that a committed number lies in an interval," in *EUROCRYPT*, 2000.

[52] I. Damgård and E. Fujisaki, "A statistically-hiding integer commitment scheme based on groups with hidden order," in *ASIACRYPT*, 2002.

[53] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *IEEE Symposium on Security and Privacy, SP*, 2013.

[54] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Innovations in Theoretical Computer Science*, 2012.

[55] M. Campanelli, D. Fiore, and A. Querol, "Legosnark: Modular design and composition of succinct zero-knowledge proofs," in *Proceedings of the ACM CCS*, 2019.

[56] A. Shamir, "On the generation of cryptographically strong pseudorandom sequences," *ACM Transactions on Computer Systems (TOCS)*, 1983.

[57] https://anonymous.4open.science/r/VTLP-7578/extended_version.pdf.

[58] J.-S. Coron and D. Naccache, "Security analysis of the gennaro-halevi-rabin signature scheme," in *EUROCRYPT*, 2000.

[59] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, 1992.

[60] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT*, 2010.

[61] A. Ozdemir, R. Wahby, B. Whitehat, and D. Boneh, "Scaling verifiable computation using efficient set accumulators," in *USENIX Security Symposium*, 2020.

[62] E. Ghosh, O. Ohrimenko, D. Papadopoulos, R. Tamassia, and N. Triandopoulos, "Zero-knowledge accumulators and set algebra," in *ASIACRYPT*, 2016.

[63] M. Campanelli, D. Fiore, S. Han, J. Kim, D. Kolonelos, and H. Oh, "Succinct zero-knowledge batch proofs for set accumulators," in *Proceedings of the ACM CCS*, 2022.

[64] D. Micciancio, "The RSA group is pseudo-free," in *EUROCRYPT*, 2005.

[65] D. Boneh *et al.*, "Twenty years of attacks on the rsa cryptosystem," *Notices of the AMS*, 1999.

[66] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, "Consensys/gnark: v0.8.0," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.5819104

[67] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica, "DIZK: A distributed zero knowledge proof system," in *USENIX Security Symposium*, 2018.

[68] M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *ASIACRYPT*, 2016.

[69] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of cryptographic engineering*, 2012.

[70] NIST, "Sp 800-52 rev. 1. nist special publication (2013)," https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-52r1.pdf/, 2013, accessed: 2023-01-02.

[71] 0xPARC Foundation, "Batch ecdsa in snarks," https://0xparc.org/blog/batch-ecdsa, 2022, accessed: 2023-01-02.

[72] S. Micali, M. O. Rabin, and S. P. Vadhan, "Verifiable random functions," in *Annual IEEE Symposium on Foundations of Computer Science FOCS*, 1999.

[73] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys," in *Public Key Cryptography (PKC)*, 2005.

[74] G. L. Miller, "Riemann's hypothesis and tests for primality," in *Proceedings of the seventh annual ACM symposium on Theory of computing*, 1975.

[75] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *EUROCRYPT*, 1997.

[76] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient nonmembership proofs," in *ACNS*, 2007.

[77] I. A. Tomescu Nicolescu, "How to keep a secret and share a public key (using polynomial commitments)," Ph.D. dissertation, Massachusetts Institute of Technology, 2020.

[78] A. Tomescu, Y. Xia, and Z. Newman, "Authenticated dictionaries with cross-incremental proof (dis) aggregation." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1239, 2020.

[79] R. Impagliazzo and M. Naor, "Efficient cryptographic schemes provably as secure as subset sum," *Journal of cryptology*, vol. 9, no. 4, pp. 199–216, 1996.

[80] V. Lyubashevsky, A. Palacio, and G. Segev, "Public-key cryptographic primitives provably as secure as subset sum," in *Theory of Cryptography Conference (TCC)*, 2010.

[81] J. Maire and D. Vergnaud, "Commitments with efficient zero-knowledge arguments from subset sum problems," in *Computer Security. ESORICS 2023*, ser. Lecture Notes in Computer Science. Springer, 2023.

[82] M. J. Coster, A. Joux, B. A. LaMacchia, A. M. Odlyzko, C.-P. Schnorr, and J. Stern, "Improved low-density subset sum algorithms," *Computational complexity*, 1992.

[83] K. Koiliaris and C. Xu, "Faster pseudopolynomial time algorithms for subset sum," *ACM Transactions on Algorithms (TALG)*, 2019.

[84] M. Bellare and P. Rogaway, "The exact security of digital signatures - how to sign with RSA and rabin," in *EUROCRYPT*, 1996.

[85] J. Coron, "On the exact security of full domain hash," in *CRYPTO*, 2000.

[86] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," in *USENIX Security Symposium*, 2021.

[87] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno, "Cinderella: Turning shabby x. 509 certificates into elegant anonymous credentials with the magic of verifiable computation," in *IEEE Symposium on Security and Privacy, SP*, 2016.

[88] G. Ateniese, E. De Cristofaro, and G. Tsudik, "(if) size matters: Size-hiding private set intersection," in *Public Key Cryptography (PKC)*, 2011.

[89] E. R. Canfield, P. Erdös, and C. Pomerance, "On a problem of oppenheim concerning "factorisatio numerorum"," *Journal of number theory*, 1983.

[90] D. Bernhard, O. Pereira, and B. Warinschi, "How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios," in *ASIACRYPT*, 2012.

[91] A. Ivić and G. Tenenbaum, "Local densities over integers free of large prime factors," *The Quarterly Journal of Mathematics*, 1986.

[92] "Perpetual trading," https://docs.starkware.co/starkex/perpetual/oracle-price-tick.html, 2023.

# Appendix A. Extended Preliminaries

**RSA signatures.** RSA signatures are classical public key digital signature schemes. It involves three steps: key generation, sign, and verification. During key generation, two large primes are chosen and multiplied to form the modulus

$N = pq$, while the private and public exponents $d, e$ are chosen such that $d \cdot e = 1 \mod \phi(N)$, $\phi(\cdot)$ is the Euler's totient function. To sign a message $m$, the signer computes the signature $\sigma = FDH(m)^d \mod N$ where $FDH(\cdot)$ is a full-domain hash [65]. To verify a signature $\sigma'$ for message $m$, the verifier checks if $\sigma'^e = FDH(m) \mod N$. For faster verification, $e$ is usually chosen to be a small prime number, e.g., $3, 17, 2^{16}+1$.

**Verifiable Random functions (VRFs).** VRFs are cryptographic functions that, for a given input and public key, generate a unique random number and a corresponding proof, which can be verified to confirm that the number was produced deterministically and honestly in relation to the public key and specific input. We follow the VRF definitions provided by [72], [73]. VRFs are delineated by three properties: provability, uniqueness, and existential pseudorandomness. Provability ensures a prover can convince a verifier that the random number was generated correctly. Uniqueness guarantees that for a given input and public key, there is only one random number that the verifier will accept as valid. Existential pseudorandomness ensures that to any $PPT$ adversary, the output appears indistinguishable from a truly random value, except if the adversary executes the verification function.

**Order assumption.** Given a hidden order group $\mathbb{G}_?$, and a random group element $g \in \mathbb{G}_?$, it is hard to find any multiples of its order [74]. It can be formally stated as:

**Assumption 1.** *Order assumption. For every $PPT$ algorithm $\mathcal{A}$*

$$\Pr\left[\begin{array}{c|c} g^e = 1_{\mathbb{G}_?} & \begin{array}{c}\mathbb{G}_? \leftarrow GGen(1^\lambda) \\ g \xleftarrow{\$} \mathbb{G}_? \\ e \leftarrow \mathcal{A}(\mathbb{G}_?, g)\end{array}\end{array}\right] < \text{negl}(\lambda).$$

**Strong RSA assumption.** Given a hidden order group $\mathbb{G}_?$ and a random $x \xleftarrow{\$} \mathbb{G}_?$, it is hard to find a non-trivial root $e \in [2, N-1]$ of a random group element $y \in \mathbb{G}_?$ such that $y^e = x$. The strong RSA assumption was introduced in [75] and used in many cryptographic schemes (e.g., [27], [76], [77], [78]). It can be formally stated as follows:

**Assumption 2.** *Strong RSA assumption. For every $PPT$ algorithm $\mathcal{A}$, and $GGen(1^\lambda)$ is an algorithm that generates a group $\mathbb{G}$ with hidden order.*

$$\Pr\left[\begin{array}{c|c} y^e = g \wedge e > 1 & \begin{array}{c}\mathbb{G}_? \leftarrow GGen(1^\lambda) \\ g \xleftarrow{\$} \mathbb{G}_? \\ (y, e) \leftarrow \mathcal{A}(\mathbb{G}_?, g)\end{array}\end{array}\right] < \text{negl}(\lambda).$$

**Adaptive Root assumption.** The adaptive root assumption was introduced by Wesolowski [42]. It implies that adversaries cannot compute the order of any non-trivial element. We denote $1_{\mathbb{G}}$ as the identity element in group $\mathbb{G}$. Identity elements are excluded as challenges since any root of the identity element is itself. It can be formally stated as follows:

**Assumption 3.** *Adaptive Root assumption. For every $PPT$ algorithm $(\mathcal{A}_0, \mathcal{A}_1)$, and given a $GGen(1^\lambda)$ algorithm generating a group $\mathbb{G}_?$ of hidden order*

$$\Pr\left[\begin{array}{c|c} u^e = g \wedge \\ g \neq 1_{\mathbb{G}_?} & \begin{array}{c}\mathbb{G}_? \leftarrow GGen(1^\lambda) \\ (g, state) \leftarrow \mathcal{A}_0(\mathbb{G}_?) \\ e \xleftarrow{\$} \text{Primes}(\lambda) \\ u \leftarrow \mathcal{A}_1(g, state, e)\end{array}\end{array}\right] < \text{negl}(\lambda).$$

**Decisional Modular Subset Sum (DMSS) assumption.** This decisional modular variant of the subset sum assumption was initially proposed by Impagliazzo and Naor in [79] and later employed in [80], [81]. The DMSS assumption posits that distinguishing between a sum of a randomly chosen subset of integers modulo a given modulus and a random number drawn from a uniform distribution modulo the same modulus is a computationally infeasible task. It can be formally stated as follows:

**Assumption 4.** *Decisional modular subset sum assumption. For every $PPT$ algorithm $\mathcal{A}$, and given a* **modulus**$(1^\lambda)$ *algorithm generating a random integer modulus $n$ and $\ell = poly(\lambda)$,*

$$\Pr\left[\begin{array}{c|c} b = b' & \begin{array}{c} n \leftarrow \textbf{modulus}(1^\lambda) \\ \vec{\gamma} \xleftarrow{\$} [n]^l, \vec{x} \xleftarrow{\$} \{0,1\}^l \\ t_0 =< \vec{\gamma}, \vec{x} > \mod n \\ t_1 \xleftarrow{\$} [n], b \xleftarrow{\$} \{0,1\} \\ b' \leftarrow \mathcal{A}(1^\lambda, n, \vec{\gamma}, t_b)\end{array}\end{array}\right] < 1/2 + \text{negl}(\lambda).$$

The hardness of the DMSS assumption depends on the random set size $l$, the modulo $n$, and the *"density"* $l/|n|$. The DMSS problem becomes easy when the density is too small ($k/|N| < 1/k$) [82]. $l$ and $n$ need to be larger than 256 to achieve 128 bits of security [82], [83].

**DDH-II assumption.** DDH-II assumption for hidden order groups was first introduced in [63] for a well-spread distribution. A distribution $\mathcal{WS}_{2\lambda}$ with domain $\mathcal{X}_{2\lambda}$ is called well-spread if $\Pr[X = x | X \xleftarrow{\$} \mathcal{WS}_{2\lambda}] \leq 2^{-2\lambda}$ for each $x \in \mathcal{X}_{2\lambda}$. In other words, elements sampled from this distribution are sufficiently random. Because the order of the group is unknown, we denote the upper bound of the group order as maxord($\mathbb{G}$) and the lower bound of the group as minord($\mathbb{G}$). It can be formally stated as follows:

**Assumption 5.** *DDH-II assumption. For every $PPT$ algorithm $\mathcal{A}$, and $GGen(1^\lambda)$ is an algorithm that generates a group $\mathbb{G}$ with hidden order, $g \xleftarrow{\$} \mathbb{G}$. Let $\mathcal{WS}_{2\lambda}$ be a well-spread distribution with domain $\mathcal{X}_{2\lambda} \subseteq [1, \text{minord}(\mathbb{G})]$. Then for any $PPT$ adversary $\mathcal{A}$*

$|\Pr[\mathcal{A}(g^x, g^y, g^{xy}) = 0] - \Pr[\mathcal{A}(g^x, g^y, g^r) = 0]| < \text{negl}(\lambda)$ *where* $x \xleftarrow{\$} \mathcal{WS}_{2\lambda}$, $y, r \xleftarrow{\$} [1, \text{maxord}(\mathbb{G})2^\lambda]$.

## Appendix B. Offloading signatures verifications

This section describes new techniques for offloading RSA signature checks from SNARK. We first describe a

basic solution that verifies whether all private inputs to a SNARK, termed "messages" from the signature perspective or "witnesses" from the SNARK perspective, are signed by a known RSA public key. We detail an efficient construction that utilizes RSA screening and SNARK-friendly full-domain hashes. Then, we describe how to offload the RSA screening part of the SNARK using our **OffloadProd** protocol. Our method is universally applicable to any SNARK framework that supports CP-SNARK. Moreover, we show how to achieve zero knowledge while offloading signatures.

## B.1. RSA signatures inside SNARK

**RSA screening.** For standard "hash-then-sign" RSA signatures, there exists a batching technique called RSA screening [38]. This technique enables an aggregator to produce a *constant-sized* proof that validates a set of unrepeated messages as being signed by a public key $(N, e)$. The verifier can then ascertain the authenticity of the entire message batch by verifying this single proof, obviating the need to check each signature individually. This technique not only streamlines the verification process but also reduces the data transmission load and lessens the computational burden on the verifier. We briefly introduce this technique below and refer to [38] for more details. Given an RSA public key pair $(N, e)$, consider a sequence of $k$ unrepeated messages $m_0, m_1, \ldots, m_k$ accompanied by their RSA signatures $\sigma_0, \sigma_1, \ldots, \sigma_k$. The aggregator computes the proof, denoted as $\pi_{sig}$, simply by computing the product of all signatures modular $N$: $\pi_{sig} = \prod_{i\in[k]} \sigma_i \bmod N$. The verifier checks $\pi_{sig}^e \stackrel{?}{=} \prod_{i\in[k]} H(m_i) \bmod N$ where $H(\cdot)$ is the Hash function used for the RSA signature.

**Secure hashing within SNARK.** Most established RSA signature schemes rely on a sophisticated structure of the message to be signed to resist attacks [65]. The principle is that the Hash output needs to span the full domain of the RSA modulus. There are two common approaches for RSA signatures as listed by PKCS 2.1 [41]:

*Full Domain Hash* [84], [85]. In this approach, the hash function outputs a value that can directly cover the entire domain space. There are two main methods to achieve this. The first one is concatenating multiple hash outputs: shorter hash functions, e.g., SHA-256, are repeatedly applied to the input together with a counter, and their outputs are concatenated together until the combined length matches the bit length of the domain. For example, to compute a 2048 bits Full Domain Hash, we can compute $H_{2048}(m) = SHA256(m||8)||\ldots||SHA256(m||1)$.
The second one uses sponge hashes, e.g., SHA3, which output to an arbitrary length and directly fulfills the Full Domain requirement.

*EMSA-PKCS1-v1_5*. In this approach, a standard hash function like SHA-256 is applied to the input to generate an initial output. This hash output is then concatenated with a fixed padding string to extend the total length beyond the size of the target domain. This approach can be viewed as computing the final output by adding a large constant number so that the result is larger than the RSA modulus $N$. For example, to compute a 2048 bits Hash-then-Padding hash, we compute $H_{2048}(m) = \Delta + SHA256(m)$ where $\Delta$ is a fixed integer with 2048 bits.

Given the computational intensity of evaluating hash functions within SNARKs, two efficient methods stand out for encoding RSA signatures within these constraints: (1) using SNARK-friendly sponge hashes like Poseidon [86] to construct a Full Domain Hash; (2) using SNARK-friendly hashes and adding a large fixed integer to construct Hash-then-Padding. In this work, we prefer method (2) for encoding within SNARKs, following the precedent set by [87]. This preference is due to method (2)'s practical speed advantages (we can encode the integers within the native field of the SNARK) and potential compatibility with existing PKI standards when incorporating SHA256 as the hash function.

**SNARK-friendly RSA signatures.** Based on the above information, we present a basic solution to authenticate a set of messages inside SNARK. We first illustrate the system model to clarify the problem definition and security goals. Then, we provide a basic solution based on RSA screening.

This model has three entities: signer, verifier, and aggregator. Their roles are summarized as follows:

- Signer. The signer owns an RSA public/private key pair, sequentially signs a stream of messages $M = (m_0, \ldots, m_k)$ and sends the respective RSA signatures $\vec{\sigma} = (\sigma_0, \ldots, \sigma_k)$ to the aggregator.
- Aggregator. The aggregator is malicious. It receives all the signed messages and picks a subset of messages $M'$ for a public function $f(x; w)$. For the function $f(x; w)$, $x$ denotes public inputs, e.g., data labels and public parameters, and $w$ denotes private inputs, which are the messages signed by the signer. It proves with SNARK to show that all private inputs to function $f(\cdot)$ are signed by the signer and $f(x; M') = y$.
- Verifier. The verifier is curious but honest. It knows the public key, function $f(\cdot)$, and public input $x$ for $f(\cdot)$. It checks the SNARK proof to verify that all the private inputs to function $f(\cdot)$ are signed by the signer and $f(x; M') = y$.

The aggregator implements the SNARK prove function, and the verifier implements the SNARK verify function for the following relation $\mathcal{R}_{\textbf{OffloadSig}}$ for any function $f(\cdot)$:

$$\mathcal{R}_{\textbf{OffloadSig}} = \{(f(\cdot), x, y, N, e, \pi_{sig}; M') : f(x; M') = y \wedge$$
$$\forall m_i \in M', m_i < m_{i+1} \wedge \pi_{sig}^e \equiv \prod_{m_i \in M'} H(m_i) \bmod N\}.$$

The completeness, soundness, zero-knowledge, and succinctness follow directly from the same definition of SNARK. Setting aside the generic function $f(\cdot)$, the SNARK circuit checks:

1) $\forall m_i \in M', m_i < m_{i+1}$;
2) $\pi_{sig}^e \equiv \prod_{m_i \in M'} H_{2048}(m_i) \bmod N$.

Checking (1) guarantees that the messages are not repeated, as required by RSA screening [38]. This check is straightforward to implement inside SNARK. We employ our **OffloadProd** technique to streamline the process of

Public params $pp = (\text{security params } \lambda,\ g \in \mathbb{G}_?,\ \Delta)$
**Setup**$(\lambda,\ f(\cdot))$
**return** $\text{crs} \leftarrow \Pi.\textbf{Setup}(1^\lambda, \mathcal{R}_{\textbf{HashMod}}, \Delta)$
**Prove**$(pp, \text{crs}, N, e, M', \vec{\sigma}')$
$\forall \sigma_i \in \vec{\sigma}', \pi_{sig} = \prod \sigma_i \bmod N,\ \forall m_i \in M', u_i = \Delta + H(m_i),\ \boxed{c_{\vec{u}}} = \textbf{Commit}(\vec{u}),\ x = \prod_{u_i \in \vec{u}} u_i,\ A = g^x,$
$\xi = x \bmod N,\ \ell = \textbf{HashToPrime}(pp, \text{crs}, A, \boxed{c_{\vec{u}}}),$
$q = \lfloor x/\ell \rfloor, r = x \mod \ell, Q = g^q,\ \pi_1 = (Q, r),$
$\pi_2 \leftarrow \Pi.\textbf{Prove}(\text{crs}, \boxed{c_{\vec{u}}}, \ell, r; M'),$
$\pi_3 \leftarrow \textbf{NI-PoKEModN}(g, A, N, \xi; x),$
**return**$(\pi_{sig}, \boxed{c_{\vec{u}}}, A, \xi, \pi_1, \pi_2, \pi_3)$
**Verify**$(pp, \text{crs}, N, e, \pi_{sig}, \boxed{c_{\vec{u}}}, A, \xi, \pi_1, \pi_2, \pi_3)$
$\ell = \textbf{HashToPrime}(pp, \text{crs}, A, \boxed{c_{\vec{u}}})$, parse $\pi_1$ as $(Q, r)$
Check if $\pi_{sig}^e = \xi \bmod N$ and $Q^\ell g^r = A$.
Check if $1 \leftarrow \Pi.\textbf{Verify}(\text{crs}, \boxed{c_{\vec{u}}}, \ell, r, \pi_2)$.
Check if $1 \leftarrow \textbf{NI-PoKEModN}(g, A, N, \xi, \pi_3)$.
**return** 1 if all check passed and 0 otherwise.

Figure 9: Protocol **OffloadSig**, a protocol for offloading RSA signatures from SNARK without privacy guarantees.

checking (2). We can ask the signer to pick a small public key (e.g., 17) for faster verification.

We begin by presenting a basic variant of the protocol that doesn't incorporate privacy considerations; instead, it only uses the succinctness offered by SNARK. In this variant, messages are treated as SNARK witnesses, which serves to compress the size of the statement and communication. Given RSA public key pair $(N, e)$, set of messages $M'$ and their corresponding signatures $\vec{\sigma}'$, the general idea of offloading RSA signatures scheme is the following:

- Use an additional hidden order group element $A$ to commit the hashes of $M'$ and prove they are the same as the witnesses inside SNARK (by checking they have the same remainder $r$ modulo prime challenge $\ell$);
- By the analysis of **OffloadProd** protocol, we directly have the product of all accumulated hashes;
- Employ our **PoKEModN** protocol to prove the product of hashes committed in the $A$ modulo $N$ is $\xi$;
- Provide a $\pi_{sig}$ and show $\pi_{sig}^e = \xi \bmod N$.

Besides the function $f(\cdot)$, the SNARK circuit checks:

1) $\forall m_i \in M', m_i < m_{i+1}$;
2) $\Delta' = \Delta \bmod \ell,\ \forall m_i \in M', u_i = \Delta' + H(m_i) \bmod \ell$;
3) $r = \prod_{\forall u_i} u_i \bmod \ell$.

Ensuring that the messages are sorted and that their corresponding hash outputs match is typically indispensable for the integrity of these schemes. In comparison, the simplified circuit design reduces the computational overhead by only checking a single addition and multiplication within the prime field. We denote the above SNARK relation as $\mathcal{R}_{\textbf{HashMod}}$, the committed Hash outputs as $\boxed{c_{\vec{u}}}$ and **Commit**$(\cdot)$ the commitment function. We provide the protocol details in Figure 9. We further discuss the applications of **OffloadSig** in the extended version of our paper.

**Zero-knowledge RSA signature offloading.** To fulfill the zero-knowledge property in the offloading signature verification protocol, we additionally require (1) $k$ random numbers (denoted by $\beta_j$) as public parameters; (2) the signer additionally signs the random numbers and publishes the signatures as one-time setup; (3) use **NIZK-PoKEModN** instead of the **NI-PoKEModN**. The general idea of achieving zero knowledge is allowing the aggregator to pick a random subset of the public random numbers to randomize the accumulator and proofs to achieve computational zero knowledge. Because the whole set of random numbers is already signed by the signer, it is easy to calculate the $\pi_{sig}$ even if the set is randomized. Similar to our **OffloadExp** protocol, because $\beta_j$ are public, $\varsigma_j = \beta_j \bmod \ell$ can be input to the SNARK circuits to decrease the prover's overhead further. We denote the enhanced relation as $\mathcal{R}_{\textbf{HashMod}'}$, which additionally checks the cumulative remainder $r$ respect to $\varsigma_j$ and the subset selection. More specifically, $r = \prod_{\forall u_i} u_i \times \prod_{\forall \varsigma_j} \varsigma_j^{v_j} \bmod \ell$, where $v_j$ denotes the subset selection of random numbers. We present protocol details in Figure 10 and capture its security by the following theorem:

Public params $pp = (\text{security params } \lambda, k,\ g \in \mathbb{G}_?,\ \Delta)$
**Setup**$(\lambda,\ f(\cdot))$
$D = \max(\text{maxord}(\mathbb{G}_?), 2^{|N|}),\ |\vec{\alpha}| = k, \forall \vec{\alpha}, \alpha_i \xleftarrow{\$} [D],$
$\text{crs} \leftarrow \Pi.\textbf{Setup}(1^\lambda, \mathcal{R}_{\textbf{HashMod}'}, \Delta)$, **return** crs
**SignerSetup**$(\vec{\alpha}, N, d)$
$|\vec{\beta}| = k, \forall \vec{\beta}, \beta_j \leftarrow \textbf{Sign}(\alpha_i, N, d)$, **return** $\vec{\beta}$
**Prove**$(pp, \text{crs}, N, e, M', \vec{\sigma}', \vec{\alpha}, \vec{\beta})$
For $j \in [k],\ v_j \xleftarrow{\$} \{0, 1\}$,
$\pi_{sig} = \prod_{\forall \sigma_i \in \vec{\sigma}'} \sigma_i \times \prod_{\forall \beta_j \in \vec{\beta}} \beta_j^{v_j} \bmod N,$
$\forall m_i \in M', u_i = \Delta + H(m_i)$, if $v_j = 1$, $u_{|M'|+i} = \alpha_j$
$\setminus\setminus\ \vec{u}$ contains all the messages and the random subset.
$\boxed{c_{\vec{u}}} = \textbf{Commit}(\vec{u}),\ x = \prod_{u_i \in \vec{u}} u_i,\ A = g^x,$
$\ell = \textbf{HashToPrime}(pp, \text{crs}, A, \boxed{c_{\vec{u}}}),\ \xi = x \bmod N,$
$\varsigma_j = \beta_j \bmod \ell,\ q = \lfloor x/\ell \rfloor, r = x \bmod \ell, Q = g^q,$
$\pi_1 = (Q, r),\ \pi_2 \leftarrow \Pi.\textbf{Prove}(\text{crs}, \boxed{c_{\vec{u}}}, \ell, \vec{\varsigma}, r; M'),$
$\pi_3 \leftarrow \textbf{NIZK-PoKEModN}(g, A, N, \xi; x).$
**return**$(\pi_{sig}, \boxed{c_{\vec{u}}}, A, \xi, \pi_1, \pi_2, \pi_3)$
**Verify**$(pp, \text{crs}, N, e, \pi_{sig}, \boxed{c_{\vec{u}}}, A, \xi, \pi_1, \pi_2, \pi_3)$
$\ell = \textbf{HashToPrime}(pp, \text{crs}, A, \boxed{c_{\vec{u}}})$, parse $\pi_1$ as $(Q, r)$
Check if $\pi_{sig}^e = \xi \bmod N$, $Q^\ell g^r = A \bmod N$ and $\varsigma_j = \beta_j \bmod \ell$.
Check if $1 \leftarrow \Pi.\textbf{Verify}(\text{crs}, \boxed{c_{\vec{u}}}, \ell, \vec{\varsigma}, r, \pi_2)$.
Check if $1 \leftarrow \textbf{NIZK-PoKEModN}(g, A, N, \xi, \pi_3)$.
**return** 1 if all check passed and 0 otherwise.

Figure 10: Protocol **OffloadSigZK**, a zero-knowledge protocol for offloading RSA signatures from SNARK.

**Theorem 4.** *Let $\Pi$ be secure CP-SNARK for relation $\mathcal{R}_{\textbf{HashMod}}$, then the protocol in Figure 10 is a succinct computational zero-knowledge argument of knowledge for relation $\mathcal{R}_{\textbf{OffloadSig}}$ if* **NIZK-PoKEModN** *is a zero-knowledge argument of knowledge for relation $\mathcal{R}_{\textbf{PoKEModN}}$, the order assumption holds for $\mathbb{G}_?$ and DMSS assumption holds for the set size $k$ and modulus* **Order**$(\mathbb{G}_?)$ *and $N$ respectively.*

# Appendix C. Proofs for Section 4

## C.1. Proof for Lemma 1

*Proof.* Denote $\gcd(a,b) = k$. Because $\gcd(a,b) = k$, we can calculate $\alpha = a/k, \beta = b/k$ such that $\gcd(\alpha,\beta) = 1$, and $g^\alpha = h^\beta \mod N$. Because $k < \min(a,b)$, $\alpha = a/k > 1$ and $\beta = b/k > 1$.

Because $\gcd(\alpha,\beta) = 1$, we can use the extended Euclidian Algorithm to find $x, y \in \mathbb{Z}$ such that $\alpha x + \beta y = 1$. Let $D = g^y \times h^x \mod N$, then $D^\alpha = g^{\alpha y} \times h^{\alpha x} = h^{\beta y} \times h^{\alpha x} = h$. □

## C.2. Proofs for the PoKEModN

**Theorem 5.** *Protocol* **PoKEModN** *is an argument of knowledge for the relation* $\mathcal{R}_{\textbf{PoKEModN}}$ *if protocol* **PoKE**$^*$ *is an argument of knowledge for the relation* $\mathcal{R}_{\textbf{PoKE}^*}$ *and the strong RSA assumption holds for the group* $\mathbb{G}_?$.

*Proof.* The correctness of the protocol directly follows. To prove the special soundness, we use the extractor **Ext**$^*$ of **PoKE**$^*$ to build an extractor **Ext** for **PoKEModN**, which extracts $x$ such that $g^x = C$ and $x = \hat{x} \mod n$ with overwhelming probability against any **PoKEModN** adversary that has a non-negligible success rate.

Similar to Theorem 9 in [27], **Ext** runs a copy of **Ext**$^*$ and simulates both the **PoKE**$^*$ challenges and a **PoKE**$^*$ adversary's response. When **Ext** receives the challenge $\ell$ from **Ext**$^*$ and the **PoKEModN** adversary's response $(Q, r)$, it computes $q' = \lfloor r/\ell \rfloor$ and $r' = r \mod \ell$ so that $r = q'\ell + r'$ and sets $Q' = Q^n g^{q'}$. It forwards $(\ell, Q', r')$ to **Ext**$^*$. If the **PoKEModN** adversary's response is valid, then $Q^{\ell n} g^e = C$. It implies $Q'^\ell g^{r'} = C$. If **PoKEModN** adversary responds with a non-negligible success rate, **Ext**$^*$ extracts $x$ with overwhelming probability such that $g^x = C$.

Note the above step is the same as the Theorem 9 in [27], we put it here for completeness of proof. Now we prove $x = r \mod \ell \cdot n$.

We consider two cases: (1) $x - r > \ell \cdot n$ and (2) $x - r \leq \ell \cdot n$. For case (1), we prove by contradiction. Because $x \neq r \mod \ell \cdot n$, $\ell \cdot n \nmid x - r$, $\gcd(\ell \cdot n, x - r) < \ell \cdot n < \min(\ell \cdot n, x - r)$. Recall that $Q^{\ell \cdot n} = g^{x-r}$, according to Lemma 1, we can efficiently compute $D^\alpha = g \mod N$, where $\alpha > 1$. This breaks the strong RSA assumption. This contradicts our assumption. Therefore, $x = r \mod \ell \cdot n$, which implies $r \mod n = x$ with overwhelming probability for any verified transcript. Moreover, for any accepting transcript, $\hat{x} = r \mod n$, we have $\hat{x} = x \mod n$.

For case (2), $x - r \leq \ell \cdot n$. Because $\ell$ is a prime challenge picked after seeing $x$ and $r$, $\Pr[(x-r)|\ell] \leq \text{negl}(\lambda)$. We consider two sub-cases: (2.1) $(x-r) \nmid n$ and (2.2) $(x-r)|n$. For case (2.1), because $(x-r) \nmid n$ and because $(x-r) \nmid \ell$ with overwhelming probability, $\gcd(\ell \cdot n, x - r) < x - r < \min(\ell \cdot n, x - r)$. Similar to case (1), according to Lemma 1, we can efficiently compute $D^\alpha = g \mod N$, where $\alpha > 1$. This breaks the strong RSA assumption, which contradicts our assumption. For case (2.2), because $(x-r)|n$, there exists an integer $k$ s.t. $x - r = k \times n$. Thus, $x - r \mod n = 0$, $x \equiv r \mod n$. Because $\hat{x} = r \mod n$, $\hat{x} = x \mod n$, which is the statement. □

We capture the security of the Protocol **ZK-PoKEModN** by Theorem 6. Note that we require trapdoor simulation, a technique previously used in other hidden-order group proofs [88].

**Theorem 6.** *Protocol* **ZK-PoKEModN** *is a statistical zero-knowledge argument of knowledge for the relation* $\mathcal{R}_{\textbf{PoKEModN}}$ *if protocol* **PoKE**$^*$ *is an argument of knowledge for the relation* $\mathcal{R}_{\textbf{PoKE}^*}$ *and the strong RSA assumption holds for the group* $\mathbb{G}_?$.

*Proof.* The correctness of the protocol directly follows.

**Special Soundness.** We denote $C \times D^n = g^\beta$. Based on the same analysis in Theorem 5, if the strong RSA assumption holds and protocol **PoKE**$^*$ is a valid argument of knowledge, we can extract the exponent $\beta$ s.t. $\beta \mod n = \hat{x}$. We then use the extractor of $D$ to extract its exponent $m$ and calculate the $x$ by $x = \beta - mn$ s.t. $C = g^x$. Because $mn|n$, and $\beta \equiv \hat{x} \mod n$, we can get $x \equiv \beta - mn \equiv \hat{x} \mod n$.

**Zero knowledge.** Recall that **Order**$(\mathbb{G}_?)$ denotes the order of the hidden order group. We build a simulator **Sim**$(\tilde{D}, \tilde{\pi}, \tilde{Q}, \tilde{r})$ as follow:

- $\tilde{m} \xleftarrow{\$} [B]$, $\tilde{D} = g^{\tilde{m}}$, $\tilde{\pi} =$ **NI-PoKE**$^*(g, \tilde{D}; \tilde{m})$
- $\iota = (\ell \cdot n)^{-1} \mod$ **Order**$(\mathbb{G}_?)$
- $\rho \xleftarrow{\$} [\ell]$, $\tilde{r} = \hat{x} \cdot \rho$, $\tilde{Q} = (C \times \tilde{D}^n \times g^{-\tilde{r}})^\iota$

We can calculate $\iota$ efficiently using the extended Euclidean algorithm because **Order**$(\mathbb{G}_?)$ is the product of two large primes that are co-prime with $(\ell \cdot n)$ with overwhelming probability.

We now argue that $(\tilde{D}, \tilde{\pi}, \tilde{Q}, \tilde{r})$ is statistically indistinguishable from a transcript between an honest prover and verifier $(D, \pi, Q, r)$. Because $(\tilde{m}, \tilde{D})$ are generated in the same way as the honest prover and verifier. Thus, $(\tilde{m}, \tilde{D})$ are perfect zero-knowledge. Because **NI-PoKE**$^*$ is made non-interactive using Fiat-Shamir heuristic, $\tilde{\pi}$ is deterministic based on $(\tilde{m}, \tilde{D})$ and also perfect zero-knowledge.

It remains to prove $(\tilde{Q}, \tilde{r})$ have the correct distribution. We first introduce the following facts:

- Fact 4 [27]. If a distribution $Z$ is a uniform random variable over $N$ consecutive integers and $m < N$, then $Z \mod m$ has statistical distance at most $m/N$ from the uniform distribution over $[m]$.
- Fact 5 (ring isomorphism based on Chinese Remainder Theorem). Given two positive integers $a, b$ that are co-prime, then $x \mod ab \longmapsto (x \mod a, x \mod b)$.
- Fact 6. Given two positive integers $a, b$ that are co-prime, then $xb \mod ab \longmapsto x \mod a$.

Fact 6 can be checked based on Fact 5 in the following way: $xb \mod ab \longmapsto (xb \mod a, xb \mod b)$ where $xb \mod b = 0$ is fixed. Because $b$ is a fixed positive integer, $xb \mod a \longmapsto x \mod a$.

Next, we analyze the distribution of $r = x + mn \mod \ell \cdot n$. Based on Fact 4 and $B >$ maxord$(\mathbb{G})2^\lambda$, the distribution

$m$ mod $\ell$ has statistical distance at most $1/2^\lambda$ from uniform random in $[0, \ell-1]$. Because $\ell$ is a prime challenge randomly selected after seeing the statement including $n$, $\ell$ is co-prime with $n$ with overwhelming probability. Based on Fact 6, the distribution $mn$ mod $\ell \cdot n$ is isomorphism with the distribution $mn$ mod $\ell$. Because $n$ is fixed in the statement, the distribution $mn$ mod $\ell$ is the same as the distribution $m$ mod $\ell$ and has statistical distance at most $1/2^\lambda$ from uniform random in $[0, \ell-1]$.

We rewrite $x$ as $\hat{x}+kn$. The distribution $x+mn = \hat{x}+kn+mn = \hat{x}+(k+m)n$. Based on Fact 6, the distribution $\hat{x}+(k+m)n$ mod $\ell \cdot n$ forms a ring isomorphism with the distribution $(k+m)$ mod $\ell$. Because $k$ is fixed before choosing $\ell$, the distribution $(k+m)$ mod $\ell$ is same as the distribution $m$ mod $\ell$, and has statistical distance at most $1/2^\lambda$ from uniform random in $[0, \ell-1]$. Therefore, $r = x+mn$ mod $\ell n \in \{0, \hat{x}, 2\hat{x}, \ldots, (\ell-1)\hat{x}\}$ and has statistical distance at most $1/2^\lambda$ from uniform random. According to the definition of the **Sim**, the distribution of $\tilde{r}$ is the uniform distribution of $\{0, \hat{x}, 2\hat{x}, \ldots, (\ell-1)\hat{x}\}$, the distribution of $r$ and $\tilde{r}$ has statistical distance at most $1/2^\lambda$.

Because $\tilde{Q} = (C \times \tilde{D}^n \times g^{-\tilde{r}})^\iota$, which is computed based on $(C, \tilde{D}, \ell, n, \tilde{r})$ uniquely. $(C, \ell, n)$ are the same for both the simulator and the verifier, and $(\tilde{D}, \tilde{r})$ are statistically indistinguishable from $(D, r)$, thus, $Q$ and $\tilde{Q}$ are also statistically indistinguishable. □

## C.3. Backgrounds on smooth numbers and proofs for PoKDE and PoMoDE

**Smooth numbers.** For a positive integer $z$, it is $y$-smooth if all prime factors of $z$ are less than $y$. Denote by $\psi(a, y)$ the number of integers $1 \le z \le a$ that are $y$-smooth. The following theorem [89] can estimate the density of smooth numbers:

**Theorem 7.** *If $\epsilon$ is an arbitrary positive constant then uniformly for $a \ge 10$ and $y \ge (\log a)^{e+\epsilon}$,*

$$\psi(a, y) = xu^{-u+o(u)} \text{ as } a \to \infty$$

*where $u = (\log a)/(\log y)$.*

Setting $y = L_a[\beta] = \exp((\beta + o(1))\sqrt{\log a \log\log a})$, we can describe the probability that a uniform random integer $z$ between one and $a$ is $y$-smooth is:

$$\Pr[z \text{ is } y-\text{smooth}] = \frac{\psi(a, y)}{a} = L_a[-\frac{1}{2\beta}].$$

Coron and Naccache [58] analyzed and tested Theorem 7 for condition $\beta = \sqrt{2}/2$. The time complexity to find a smooth number for uniform random numbers $[1, a]$ is:

$$\exp\left((\sqrt{2}/2 + o(1))\sqrt{\log a \log\log a}\right).$$

For $a = 2^{1024}$, it takes time $2^{86}$ to find a number that is $2^{86}$-smooth, and for $a = 2^{2048}$, it takes time around $2^{120}$ to find a number that is $2^{120}$-smooth.

**Proof for Lemma 2.**

*Proof.* According to Theorem 7, we have

$$P_2 = \frac{\psi(\Delta+a, y) - \psi(\Delta, y)}{\Delta + a} \le \frac{\psi(\Delta+a, y) - \psi(\Delta, y)}{a}$$
$$\le \frac{\psi(\Delta+a, y)}{a} \le \frac{\psi(a, y)}{a} = P_1.$$

□

**Theorem 8.** *Protocol* **PoKDE** *is an argument of knowledge for relation $\mathcal{R}_{\textbf{PoKDE}}$ if adaptive root assumption and order assumption holds for $\mathbb{G}_?$.*

*Proof.* The correctness can be checked based on Fact 1.

**Special soundness.** First, we show we can build an extractor to extract the exponent of $C_1$ and $C_2$. The **PoKDE** is running the **PoKE** protocol in parallel for both $C_1$ and $C_2$, we can extract $x_1 = \log_g C_1$ and $x_2 = \log_g C_2$ similar to the **PoKE** protocol with overwhelming probability if adaptive root assumption holds for $\mathbb{G}_?$.

Next, we prove $x_2 = x_1^e$. We prove by contraposition supposing $x_2 \ne x_1^e$. Because $x_1 \mod \ell = r_1$ and $r_1^e = r_2 \mod \ell$, we know $x_1^e \mod \ell = r_2$. We rewind the extractor for $K = poly(\lambda)$ times, and compute $x_1^e$ using Chinese Remainder Theorem for $K = poly(\lambda)$ times based on $\ell$ and $r_2$, to get the only solution within range $[0, 2^{\lambda K}]$ s.t. $x_1^e \mod \ell = r_2$. Similarly, we and compute $x_2$ using Chinese Remainder Theorem for $K = poly(\lambda)$ times based on $\ell$ and $r_2$, to get the only solution within range $[0, 2^{\lambda K}]$ s.t. $x_2 \mod \ell = r_2$. Because there is only one solution within $[0, 2^{\lambda K}]$ that modulo $\ell$ equals to $r_2$, and because we can extract a $x_2$ s.t. $x_2 \ne x_1^e$, we have two different integer representation of $C_2$. In other words, we have $C_2 = g^{x_2} = g^{x_1^e}$, which breaks the order assumption. □

We prove the security of Theorem 1 by the following proof:

*Proof.* **Correctness.** We can check that $C_1^\ell \times D = g^{x\ell+m} = g^{z-\gamma}$, $K = C_2^{\ell^e} = g^{x^e \ell^e} = g^{(\ell x)^e}$, $E/K = g^{z^e - (\ell x)^e} = g^\omega = g^{\omega' \times (m+\gamma)}$.

**Special soundness.** To show the special soundness, we build an extractor **Ext** to extract valid witness $x, x^e$, s.t. $C_1 = g^x$ and $C_2 = g^{x^e}$. **Ext** can use the extractor of **NI-PoKDE**$(C_1^\ell \times D \times g^\gamma, E, e)$ together with $\gamma$ to extract the witness $x\ell+m$. **Ext** can use the extractor of **NI-PoKE**$(g, D)$ to extract the random number $m$. Then, **Ext** can extract $x$ based on $x\ell+m$, $\ell$ and $m$.

We show the extracted $x$ raised to the power of $e$ is the witness of $C_2$. We prove $g^{x^e} = C_2$ by contradiction. Under the generic group model, there is a unique integer to generate $C_2$. We denote $C_2 = g^\sigma$. Because **NI-PoKDE** is an argument of double exponent, and $g^z = C_1^\ell \times D \times g^\gamma$, given a valid proof for **NI-PoKDE**$(C_1^\ell \times D \times g^\gamma, E)$, we have $E = g^{z^e}$. Given a valid **NI-PoE**$(C_2, K, \ell^e)$ proof, we

know $K = C_2^{\ell^e} = g^{\ell^e \times \sigma}$. When $\sigma \neq x^e$, by the binomial expansion:

$$\begin{aligned}\log_g E/K &= z^e - \ell^e \times \sigma = (x\ell + m + \gamma)^e - \ell^e \times \sigma \\ &= \binom{e}{0}(x\ell)^e + \binom{e}{1}(x\ell)^{e-1} \times (m+\gamma) + \dots \\ &+ \binom{e}{0}(m+\gamma)^e - \ell^e \times \sigma \\ &= (x^e - \sigma)l^e + \binom{e}{1}(x\ell)^{e-1} \times (m+\gamma) + \dots \\ &+ \binom{e}{0}(m+\gamma)^e \\ &= (x^e - \sigma)l^e + (m+\gamma) \times (\binom{e}{1}(x\ell)^{e-1} + \dots \\ &+ \binom{e}{0}\gamma^{e-1}).\end{aligned}$$

Because **NIZK-PoKE2** is an argument of knowledge of exponent for the relation $\mathcal{R}_{\textbf{PoKE2}}$, given a valid proof **NIZK-PoKE2**$(g^{m+\gamma}, F, \pi_4)$, we know there is a valid exponent $\eta$ s.t. $(g^{m+\gamma})^\eta = g^{(m+\gamma)\eta} = F$. Because $E/K = F$, we have

$$\begin{aligned}\log_g F =&(m+\gamma)\eta = (x^e - \sigma)l^e + (m+\gamma)\times \\ &(\binom{e}{1}(x\ell)^{e-1} + \dots + \binom{e}{0}\gamma^{e-1}).\end{aligned}$$

Since $(m+\gamma)|(x^e-\sigma)l^e + (m+\gamma) \times (\binom{e}{1}(x\ell)^{e-1} + \dots + \binom{e}{0}\gamma^{e-1})$, we have $(m+\gamma)|(x^e-\sigma)l^e$. Because $\ell$ is a prime with $\lambda$ bits, $(m+\gamma) \nmid l^e$ with overwhelming probability. Thus, $(m+\gamma)|(x^e-\sigma)$. When $m < B$, $(m+\gamma)$ is still uniformly random in $[B]$, and according to our assumption, $(m+\gamma)$ is non-smooth. When $m \geq B$, and according to Lemma 2, $(m+\gamma)$ is also non-smooth.

Since $(x^e-\sigma)$ are fixed before the verifier chooses $\gamma$, and $(m+\gamma)$ is non-smooth, $\gamma|(x^e-\sigma)$ happens with negligible probability and contradicts our assumption. Therefore, $x^e = \sigma$ with overwhelming probability.

**Zero-knowledge.** To prove the statistically special-honest verifier zero-knowledge, we build a simulator **Sim**, which computes $(\tilde{D}, \tilde{\pi}_1, \tilde{E}, \tilde{K}, \tilde{F}, \tilde{\pi}_2, \tilde{\pi}_3, \tilde{\pi}_4)$ as follow:

- $\tilde{m} \xleftarrow{\$} [S]$, $\tilde{D} = g^{\tilde{m}}$, $\tilde{\pi}_1 =$ **NI-PoKE**$(g, \tilde{D}; \tilde{m})$
- $\tilde{z} \xleftarrow{\$} [S], \tilde{E} = g^{\tilde{z}^e}$, $\tilde{K} = C_2^{\ell^e}$, $\tilde{F} = g^{\omega} = g^{\omega'(m+\gamma)}$
- $\tilde{\pi}_2 =$ **NI-PoE**$(C_2, K; \ell^e)$
- $\tilde{\pi}_3 =$ **NIZK-PoKE2**$(g^{m+\gamma}, F; \omega')$
- $\tilde{\pi}_4 =$ **NI-PoKDE**$(C_1^\ell \times \tilde{D} \times g^\gamma, \tilde{E}, e; \tilde{z})$

Because $\tilde{D}, \tilde{\pi_1}, \tilde{K}$ are computed in the same way as the transcript between an honest prover and verifier, they are perfect zero-knowledge. Recall that $\pi_2$ is the proof of **PoE**$^*$ protocol and provided for the verifier's efficiency. Because the distribution of $\tilde{z}$ is the same as $z$, thus, $\tilde{E}, \tilde{F}$ have the same distribution as the transcript between an honest prover and verifier. $\tilde{\pi}_3$ is the proof of the **NIZK-PoKE** protocol, we can directly use the simulator of **NIZK-PoKE** to simulate them, which is statistically zero-knowledge. Because **NI-PoKDE**$^*$ is made non-interactive using Fiat-Shamir heuristic, $\tilde{\pi}_4$ is deterministic based on $(C_1, \ell, \tilde{D}, \gamma, \tilde{E}, e)$, where the simulator's transcript is the same as the honest prover and verifier's transcript, $\tilde{\pi}_4$ is also perfect zero-knowledge.

In summary, the protocol is statistically special-honest verifier zero-knowledge. □

**Non-interactive variants and their security.** All of the protocols in this section can be made non-interactive using the Fiat-Shamir transform. In these non-interactive variants, the prover solely computes the protocol transcripts. This is achieved by substituting the interactive challenges with hash values generated by a collision-resistant hash function, denoted as $H$, which can be modeled as a random oracle. To compute these challenges, the hash function $H$ intakes as its argument the concatenation of all prior protocol transcripts. It is imperative that this input also includes any relevant protocol statements and any public parameters. This ensures that the challenge is intrinsically bound to the specific instance of the protocol to resist certain attacks [90].

More specifically for our protocols, the challenge is either a large integer or a prime. Hashing to large integers can be implemented by repeating the hash functions and concatenating them together. We denote hash to the range $[B]$ by $\mathbf{H}_B(\cdot)$. For an input $x$, generating a Hash to prime can be implemented by finding the smallest counter $i$ such that $H(x||i)$ is a prime. Especially, in the interactive protocols, the prime challenges are required to have $\lambda$ bits, while in non-interactive cases, they are required to have $2\lambda$ bits. We denote hash to prime with $2\lambda$ bits by $\mathbf{HPrime}_{2\lambda}(\cdot)$.

Both Wesolowski proof, PoE protocol, and PoKE protocol suffer from a sub-exponential attack, especially in non-interactive cases [27], [42], [43]: the adversary sets the exponent to be the product of $O(2^{\lambda/2})$ random primes with $2^\lambda$ bits so that the exponents divide the prime challenge with $O(2^{-\lambda/2})$ probability. Therefore, the prime challenges in non-interactive cases are set to $2\lambda$-bit to resist this attack in these protocols.

We analyze the same attack and show it *does not* apply to our **ZK-PoKDE** protocol for the smooth number part, not even in the non-interactive cases. We describe the attack as shown below:

- Adversary computes $\Psi$ to be the product of $2^{\lambda/2}$ random primes in $[2, 2^\lambda]$.
- Adversary finds a random $x$ and compute $\sigma$ such that $x^e - \sigma = \Psi$, $C = g^\sigma$, $C_{root} = g^x$, $\gamma = \mathbf{H}_B(g||\mathbb{G}_?||N||e||C||C_{root})$, $\gamma$ divides $(x^e - \sigma)$.

Suppose $\mathbf{H}_B$ can be modeled as a random oracle that outputs uniform random numbers in the range $[B]$, such that numbers are $2^\lambda$-smooth. To simplify our analysis, we will first introduce a fundamental principle of square-free numbers. Square-free numbers are defined as those numbers that, in their prime factorization, each prime number appears at most once. This means that no prime number is squared or appears multiple times in the factorization. According to [91], random numbers are square-free asymptotically with probability $6/\pi^2$. If $\gamma$ is not square-free, $\gamma \nmid (x^e-\sigma)$ because the repeated prime factor $p$ is "included" in $\Psi$ twice, with probability $1/2^\lambda$.

Suppose $\gamma$ is square-free, we can express $\gamma$ as the product of its distinct prime factors in ascending order, written as $\gamma = p_1 p_2 \dots p_k$. Because $\gamma$ is non-smooth with respect to $2^\lambda$, $p_k \geq 2^\lambda$. If $p_k > 2^\lambda$, the adversary fails. If $p_k$ has $2^\lambda$ bits, $\Pr[p_k \text{ divides } \Psi] \leq 1/2^{\lambda/2}$. Because $\gamma > 2^\lambda$, it has at least another prime factor $p_{k-1} \leq 2^\lambda$. Because each prime is included into $\Psi$ with same probability, then $\Pr[p_{k-1} \text{ divides } \Psi] \leq 1/2^{\lambda/2}$. In summary,

$$\begin{aligned}\Pr[\gamma \text{ divides } \Psi] &\leq \Pr[\gamma \text{ divides } \Psi | \gamma \text{ is square-free}] \\ &\leq \Pr[p_{k-1} \text{ divides } \Psi] \times \Pr[p_k \text{ divides } \Psi] \\ &\leq 1/2^{\lambda/2} \times 1/2^{\lambda/2} < 1/2^\lambda.\end{aligned}$$

The above analysis also suggests the time complexity of a trivial attack: the adversary: (1) compute $\Psi$ to the product of all primes less than $2^\lambda$ and (2) repeatedly queries a new challenge $\gamma$ until $\gamma$ is $2^\lambda$-smooth. The step (1) takes time at least $O(2^{2\lambda})$ and step (2) happen with probability less than $1/2^\lambda$.

**Theorem 9.** *Protocol* **NIZK-PoMoDE** *is a statistical zero knowledge argument of knowledge for the relation* $\mathcal{R}_{\textbf{PoMoDE}}$ *if protocol* **PoKE**$^*$ *is an argument of knowledge for the relation* $\mathcal{R}_{\textbf{PoKE}^*}$*, protocol* **NIZK-PoKEModN** *is a statistical zero-knowledge argument of knowledge for the relation* $\mathcal{R}_{\textbf{PoKEModN}}$ *and the order assumption holds for the group* $\mathbb{G}_?$.

*Proof.* The correctness of the protocol follows directly from the correctness of the **NIZK-PoKDE** protocol and **NIZK-PoKEModN** protocol.

**Special Soundness.** To prove the statistical zero-knowledge, we build an extractor **Ext** to extract the witness $x$ so that $x^e \bmod n = \hat{x}$. **Ext** can use the extractor of **PoKE**$^*$ to extract $m$. Then, **Ext** can use the extractor of **PoMoDE** to extract $x + mn$ and find $x$. **Ext** can relay on the extractor of **PoKEModN** to guarantee that $x^e \equiv (x + mn)^e \equiv \hat{x} \bmod n$.

**Zero-knowledge.** To prove the statistical zero-knowledge, we build a simulator $\textbf{Sim}(\tilde{D}, \tilde{C_2}, \tilde{\pi_1}, \tilde{\pi_2}, \tilde{\pi_3})$ as follow:

- $\tilde{m} \xleftarrow{\$} [B]$, $\tilde{D} = g^{\tilde{m}}$, $\tilde{\pi_1} = \textbf{NI-PoKE}^*(g, \tilde{D}; \tilde{m})$
- $\tilde{r} \xleftarrow{\$} [\textbf{Order}(\mathbb{G}_?)], \tilde{C_2} = g^{r^e}$
- $\tilde{\pi_2} = \textbf{NIZK-PoKDE}(C_1 \times \tilde{D}^n, \tilde{C_2}, e; \tilde{r})$
- $\tilde{\pi_3} = \textbf{NIZK-PoKEModN}(g, \tilde{C_2}, n, \hat{x}; \tilde{r})$

**Sim** can simulate $\tilde{D}$ and $\tilde{\pi_1}$ similar to our analysis in Theorem 6. We prove $\tilde{C_2}$ and $C_2$ are indistinguishable. We start with showing $x + mn \bmod \textbf{Order}(\mathbb{G}_?)$ is indistinguishable from uniform random in $[\textbf{Order}(\mathbb{G}_?)]$. Because $m$ is uniformly sampled from $[B]$, according to Fact 1 in Theorem 6, it is indistinguishable from uniform random in $[\textbf{Order}(\mathbb{G}_?)]$. Recall that $\textbf{Order}(\mathbb{G}_?)$ equals to the product of two large primes $p$ and $q$. Given that $n$ is coprime to $pq$ with overwhelming probability, we can efficiently compute its multiplicative inverse modulo $pq$. Let this inverse be denoted as $n'$. By applying the extended Euclidean algorithm, we obtain $n \times n' + b \times pq = 1$, where $b$ is an integer such that the equality holds. For any element in the group $\mathbb{Z}_{pq}$, dubbed as $t$, it can be generated uniquely by $t = mn \bmod pq$ where $m = n' \times t \bmod pq$. Because $m$ is indistinguishable from uniform random in $[pq]$, $mn \bmod pq$ is also indistinguishable from uniform random in $[pq]$. Thus, $x + mn \bmod pq$ is indistinguishable from uniform random in $[pq]$ and $C_1 \times D^n$ is indistinguishable from $g^r$. Because $C_2 = g^{(x+mn)^e}$ is deterministic computed from $C_1$ and $\tilde{C_2}$ is deterministic computed from $g^r$, $\tilde{C_2}$ and $C_2$ are statistical indistinguishable.

**Sim** can simulate $\pi_2$ independently using the simulator of the **NIZK-PoKEModN** protocol. **Sim** can simulate $\pi_3$ independently using the simulator of the **NIZK-PoKEModN** protocol. In conclusion, $(\tilde{D}, \tilde{C_2}, \tilde{\pi_1}, \tilde{\pi_2}, \tilde{\pi_3})$ is statistically indistinguishable from the transcript between an honest prover and verifier □

## Appendix D. Security properties of VTLP

**Definition 4** (Correctness)**.** *For all* $\mathcal{T} = poly(\lambda)$ *and any relation* $\mathcal{R}$ *in NP language with additional input and witness* $x, w$ *s.t.* $\mathcal{R}(x; w) = s$*, a VTLP is correct if*

$$\Pr\left[\begin{array}{c}(pk, sk) \leftarrow \textbf{KeySetup}(1^\lambda), \\ \text{crs} \leftarrow \textbf{FuncSetup}(1^\lambda, \mathcal{R}), \\ z \leftarrow \textbf{GenPuz}(s, \mathcal{T}, pk, sk), \\ (\pi_\mathcal{T}, \pi_\mathcal{R}) \leftarrow \textbf{Prove}(\text{crs}, z, s, \mathcal{T}, pk, sk, x, w), \\ s \leftarrow \textbf{SolvPuz}(pk, z, \mathcal{T}), \\ 1 \leftarrow \textbf{Verify}(\text{crs}, z, \mathcal{T}, pk, x, \pi_\mathcal{T}, \pi_\mathcal{R}), \\ 1 \leftarrow \mathcal{R}(x, s, \pi_\mathcal{R})\end{array}\right] \geq 1 - \text{negl}(\lambda)$$

**Definition 5** (Time-lock Security)**.** *For all* $\mathcal{T} = poly(\lambda)$ *and all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *where* $\mathcal{A}_2$ *runs in time less than* $\mathcal{T}$ *in parallel using at most* $\beta(\mathcal{T})$ *processors, a VTLP is time-lock secure if*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c}(pk, sk) \leftarrow \textbf{KeySetup}(1^\lambda), \\ (s_0, s_1, state) \leftarrow \mathcal{A}_1(1^\lambda, pk, \mathcal{T}), \\ b \xleftarrow{\$} \{0, 1\}, \\ z \leftarrow \textbf{GenPuz}(s_b, \mathcal{T}, pk, sk), \\ b' \leftarrow \mathcal{A}_2(pk, z, state)\end{array}\right] < 1/2 + \text{negl}(\lambda).$$

## Appendix E. Security proofs for Section 5

**Theorem 10.** *Let* $\Pi$ *be secure CP-SNARK for relation* $\mathcal{R}_{\textbf{AllMod}}$*, then the protocol in Figure 4 is a succinct argument of knowledge for relation* $\mathcal{R}_{\textbf{OffloadProd}}$ *if* **PoKE**$^*$ *is an argument of knowledge for relation* $\mathcal{R}_{\textbf{PoKE}}$.

*Proof.* The completeness and succinctness come directly from the completeness and succinctness of CP-SNARK and **PoKE**$^*$ protocol. Because $\boxed{c_{\vec{v}}}$ and $A$ are input to **HashToPrime** to generate a prime challenge $\ell$ with $2\lambda$-bit, $\boxed{c_{\vec{v}}}$ and $A$ commits the same set of integers by Fact 1 with probability larger than $1 - 1/\ell$. Based on the nature of $A$, committing the set is raising the base $g$ to the power of the

product of the set elements. Thus, $A = g^x$ where $x$ is the product of set of $\vec{v}$. We can extract the product using either the extractor of the SNARK or $A$ under the generic group model [27]. □

**Theorem 11.** *Let $\Pi$ be secure CP-SNARK for relation $\mathcal{R}_{\textbf{SelectMod}}$, then the protocol in Figure 5 is an argument of knowledge for relation $\mathcal{R}_{\textbf{RSAExp}}$ if* **NIZK-PoKEModN** *is an argument of knowledge for relation $\mathcal{R}_{\textbf{PoKEModN}}$ and the order assumption holds for $\mathbb{G}_?$.*

*Proof.* The correctness can be checked by $\omega \bmod N = (u^{2^0})^{x_0} \times (u^{2^1})^{x_1} \times \cdots \times (u^{2^{|N|-1}})^{x_{|N|-1}} \bmod N = u^x \bmod N = C$.

We prove the special soundness by building an extractor **Ext** that extracts the correct witness for the relation $\mathcal{R}_{\textbf{RSAExp}}$. **Ext** first uses the extractor of the CP-SNARK to extract the witness $x$. Based on the soundness of CP-SNARK, $\boxed{c_x}$ commits $x$. Because $\boxed{c_x}$, $u$, $A$ are inputs of **HashToPrime** to generate the prime challenge, $A$ commits the same set of squares of $u$ selected according to $x$ with overwhelming probability, similar to our analysis in Theorem 10. Based on the nature of the RSA accumulator, committing the set is raising the base $g$ to the power of the product of the set elements. Thus, the RSA accumulator is $g^\omega$ where $\omega$ is the product of selected squares of $u$. Based on the special soundness of the **NI-PoKEModN** protocol, **Ext** can extract an exponent $\omega'$ s.t. $\omega' \bmod N = C$. If $\omega' \neq \omega$, then **Ext** finds two different integer $\omega, \omega'$ s.t. $g^\omega = g^{\omega'}$, which breaks the order assumption. □

**Theorem 12.** *The commitment of form $C = g^{s||\nu}$ presented in Section 5.3 and proofs $(C, Q, r)$ are computationally indistinguishable from uniformly random group elements in $\mathbb{G}_?$ and uniform random numbers in $[\ell]$ under the DDH-II assumption.*

*Proof.* Because $\mathbb{G}_?$ is a hidden order group, the distribution of $s||\nu$ is well-spread, minord$(\mathbb{G}_?) > s||\nu$, according to Theorem 4.1 in [63], $C$ is computationally indistinguishable from a uniformly random group element in $\mathbb{G}_?$ under the DDH-II assumption. According to Fact 4, $\nu \bmod \ell$ has statistical distance at most $2^{2\lambda}/2^{4\lambda} = 1/2^{2\lambda}$ from uniform in $[2^{2\lambda}]$. Therefore, $r = s \times 2^{4\lambda} + \nu \bmod \ell$ has statistical distance at most $= 1/2^{2\lambda}$ from uniform in $[\ell]$. Because $Q$ is uniquely determined by $C$ and $r$, $(C, Q, r)$ are computationally indistinguishable from uniformly random group elements in $\mathbb{G}_?$ and uniform random numbers in $[\ell]$ under the DDH-II assumption. □

## E.1. Protocol OffloadExpCom

We depict the protocol details in Figure 11.

## E.2. Proof for Theorem 2

*Proof.* The puzzle generator can determine if $s'$ in $QR_N$ using Legendre symbol based on its knowledge of $p'$ and $q'$. The puzzle generator can compute $\eta = \tau^{-1} \bmod pq$ because

Public params $pp =$ (security parameter $\lambda$, $g \in \mathbb{G}_?$)
**Setup**$(\lambda)$
**return** crs $\leftarrow \Pi$.**Setup**$(1^\lambda, \mathcal{R}_{\textbf{SelectMod}})$
**Prove**$(pp, \text{crs}, N, u, \boxed{c_x}, D)$
Parse $x$ to binaries $(x_0, \ldots, x_{|N|-1})$.
For $i \in [|N|]$, $v_i = u^{2^i} \bmod N$, $\omega = \prod v_i^{x_i}$, $A = g^\omega$,
$\ell =$ **HashToPrime**$(pp, \text{crs}, N, u, D, A, \boxed{c_x})$.
For $i \in [|N|]$, $w_i = v_i \bmod \ell$,
$q = \lfloor \omega/\ell \rfloor, r = \omega \bmod \ell, Q = g^q$, $\pi_1 = (Q, r)$,
$\pi_2 \leftarrow \Pi$.**Prove**$(\text{crs}, \boxed{c_x}, \ell, r, w_0, \ldots, w_{|N|-1}; x)$,
$k = \lfloor \omega/N \rfloor$, $E = g^{kN}$,
$\pi_3 \leftarrow$ **NIZK-PoKE2**$(g^N, E; k)$.
**return**$(\boxed{c_x}, A, \pi_1, \pi_2, E, \pi_3)$
**Verify**$(pp, \text{crs}, N, u, \boxed{c_x}, D, A, \pi_1, \pi_2, \pi_3)$
$\ell =$ **HashToPrime**$(pp, \text{crs}, N, u, D, A, \boxed{c_x})$.
For $i \in [|N|]$, $v_i = u^{2^i} \bmod N$, $w_i = v_i \bmod \ell$.
Parse $\pi_1$ as $(Q, r)$, check if $Q^\ell g^r = A$.
Check if $1 \leftarrow \Pi$.**Verify**$(\text{crs}, \boxed{c_x}, \ell, r, v_0, \ldots, v_{|N|-1}, \pi_2)$.
Check if $1 \leftarrow$ **NIZK-PoKE2**$(g^N, E, \pi_3)$.
Check if $D \times E = A$.
**return** 1 if all check passed and 0 otherwise.

Figure 11: Protocol **OffloadExpCom**, a protocol for offloading RSA exponentiations from SNARK, proving a commitment $D$ satisfies $D = g^{u^x \bmod N}$.

$\tau = 2^\mathcal{T} \bmod pq$ and coprime with $pq$ with overwhelming probability. The rest of correctness and time-lock security follow directly from the original time-lock puzzle protocol in [1].

**Soundness.** Because $\Pi.\mathcal{R}^+(\boxed{c_{s'}}, x, \ell, r; w, s)$ is a secure CP-SNARKs, $\boxed{c_{s'}}$ commits the encoded $s$ s.t. $\mathcal{R}(x, s; w) = 1$. Because protocol **PoKE**$^*$ is an argument of knowledge for the relation $\mathcal{R}_{\textbf{PoKE}^*}$, $D$ and $\boxed{c_{s'}}$ are presented before seeing the challenge $\ell$, $D$ commits the same integer $s'$ as committed in $\boxed{c_{s'}}$, following the same analysis by Theorem 10.

Because $u'$ is sampled uniformly from random in $[N]$, $u = (u')^2 \bmod N$, $u$ is a generator of $QR_N$ with overwhelming probability. Because **OffloadExp** protocol is an argument of knowledge for relation $\mathcal{R}_{\textbf{RSAExp}}$, $\boxed{c_\tau}$ commits $\tau$ s.t. $\tau = 2^\mathcal{T} \bmod pq$. Because **OffloadExpCom** is an argument of knowledge for relation $\mathcal{R}_{\textbf{OffloadExpCom}}$, $D$ commits $\tau$ s.t. $D = g^{z^\tau \bmod pq}$, which is $s'$. If there exists an adversary $\mathcal{A}$ that outputs $z$ and valid proof s.t. $\bar{s} \leftarrow$ **SolvPuz**$(pk, z, \mathcal{T})$, $0 \leftarrow \mathcal{R}(x, \bar{s}, \pi_\mathcal{R})$, we can build another adversary $\mathcal{B}$ that finds another exponent $\bar{s}$ for group element $D$, which breaks the order assumption for $\mathbb{G}_?$.

**Zero-knowledge.** We can simulate $(D, Q, r)$ according to Theorem 12 under the DDH-II assumption. Because $p', q'$ are selected after seeing $2^\mathcal{T}$, $\tau$ is indistinguishable from uniform random in $[pq]$. We can simulate $\pi_{\mathcal{R}^+}$ if $\Pi.\mathcal{R}^+(\cdot)$ is a secure CP-SNARK.

We simulate $\pi_{\textbf{Setup}}$ by sampling $\tilde{x}$ following the same distribution of $\mu$. **Sim** generates $\pi_2$ and $\pi_3$ using the original

simulators of the SNARK and the **NI-PoKEModN** protocol. It generates $\tilde{A}$ for $\pi_{\textbf{Setup}}$ by $g^{\tilde{x}}$. $\tilde{A}$ is indistinguishable from $A$ based on the DMSS assumption over the set $v_i$ and modular $\phi(N)$. It generates $\tilde{r}$ by sampling from $[\ell]$ uniformly at random. It generate $\tilde{Q} = (\tilde{Acc}/g^r)^{\ell^{-1}}$. $\tilde{r}$ is indistinguishable from $r$ based on the DMSS assumption over the set $v_i$ and modular $\ell$. $\tilde{Q}$ is then uniquely determined based on $\tilde{A}$ and $\tilde{r}$.

Similarly for $\pi_2$, by sampling $\tilde{x}$ following the same distribution of $\tau$. Additionally, $(\tilde{A}, \tilde{Q}, \tilde{r})$ can be simulated in the same way as above and $\tilde{E}$ is uniquely generated by $\tilde{A}/D$.

$\boxed{c_{s'}}$ is usually perfect hiding if implemented by a Pedersen commitment or polynomial commitment and independent from $(D, \boxed{c_u}, Q, r)$. Therefore, the protocol presented in Figure 6 achieves computational zero-knowledge. □

### E.3. Proof for Theorem 3

*Proof.* Most parts of the protocol are similar to the protocol **VTLP** for NP, which is analyzed in Theorem 2. We focus on the different parts.

**Soundness.** Because **NIZK-PoMoDE** is an argument of knowledge for relation $\mathcal{R}_{\textbf{PoMoDE}}$, $\pi_{\mathcal{R}}$ shows $D$ commits a valid RSA VRF/signature for the message $m$. Otherwise, we can find another exponent $x'$ for $D$, which breaks the order assumption.

**Zero-knowledge.** $\pi_1$ can be simulated independently using its own simulators. It remains to show that $D$ is simulatable. We prove this by considering $s$ as a VRF value. We simulate $s$ by sampling a random value $s'$ in the VRF space $\mathbb{V}$, more specifically in $\mathbb{Z}^*_{\tilde{N}}$. Because $s$ is committed using $g$, the adversary cannot run the verification of VRF unless solving the puzzle first. Based on the Pseudorandomness of the VRF, the adversary has a negligible advantage in distinguishing $s$ from $s'$. □

## Appendix F. Security analysis for offloading signature verifications and applications

**Theorem 13.** *Let* $\Pi$ *be secure CP-SNARK for relation* $\mathcal{R}_{\textbf{HashMod}}$*, then the protocol in Figure 9 is a succinct argument of knowledge for relation* $\mathcal{R}_{\textbf{OffloadSig}}$ *if* **NI-PoKEModN** *is an argument of knowledge for relation* $\mathcal{R}_{\textbf{PoKEModN}}$.

*Proof.* The completeness and succinctness come directly from the completeness and succinctness of CP-SNARK, **PoKE**$^*$ protocol, and **NI-PoKEModN** protocol.

Now, we prove the special soundness by building an extractor **Ext** that extracts the correct witness for the relation $\mathcal{R}_{\textbf{Sig}}$. **Ext** first uses the extractor of $\Pi$ to extract the witness $M'$. Based on the soundness of CP-SNARK, $\boxed{c_{\vec{u}}}$ is the vector of hash outputs of $M'$. Because $\boxed{c_{\vec{u}}}$ and $A$ are input to **HashToPrime** to generate a prime challenge with $2\lambda$-bit, $\boxed{c_{\vec{u}}}$ and $A$ commits the same set of hash outputs with overwhelming probability. We refer to Theorem B.1 from [55] for a detailed proof. Based on the nature of the RSA accumulator, committing the set is raising the base $g$ to the power of the product of the set elements. Thus, the RSA accumulator is $g^x$ where $x$ is the product of hash outputs of $M'$. Based on the special soundness of the **NI-PoKEModN** protocol, **Ext** can extract an exponent $x'$ s.t. $x' \bmod N = \xi$. If $x' \neq x$, then **Ext** finds two different integer $x, x'$ s.t. $g^x = g^{x'}$, which breaks the order assumption. If $x' = x$, then $x \bmod = \xi$. Because $\pi^e_{sig} = \xi \bmod N$, $\pi_{sig}$ is a valid RSA signature screening proof for the messages $M'$. □

Proof for Theorem B.1.

*Proof.* The completeness and succinctness come directly from the completeness and succinctness of CP-SNARK, **PoKE**$^*$ protocol, and **NIZK-PoKEModN** protocol. The special soundness is the same as the analysis in Theorem 13 based on the fact that each $\beta$ is a valid signature for the randomizer $\alpha$.

We prove the protocol is computational zero-knowledge by building a simulator $\textbf{Sim}(\tilde{\pi_{sig}}, \boxed{\tilde{c}_{\vec{u}}}, \tilde{A}, \tilde{\xi}, \tilde{\pi_1}, \tilde{\pi_2}, \tilde{\pi_3})$:

- $j \in [k]$, $v_j \xleftarrow{\$} \{0,1\}$, $\tilde{\pi_{sig}} = \prod_{\forall \beta_j \in \vec{\beta}} \beta_j^{v_j} \bmod N$
- If $v_j = 1$, $u_i = \alpha_j$, $\boxed{\tilde{c}_{\vec{u}}} = \textbf{Commit}(\vec{u})$
- $\tilde{x} = \prod_{u_i \in \vec{u}} u_i$, $\tilde{A} = g^{\tilde{x}}$, $\tilde{\xi} = \tilde{x} \bmod N$
- $q = \lfloor \tilde{x}/\ell \rfloor, \tilde{r} = \tilde{x} \bmod \ell, \tilde{Q} = g^q$, $\pi_1 = (\tilde{Q}, \tilde{r})$
- $\tilde{\pi_2} \leftarrow \Pi.\textbf{Prove}(\text{crs}, \boxed{c_{\vec{u}}}, \ell, r; M')$
- $\tilde{\pi_3} \leftarrow \textbf{NIZK-PoKEModN}(g, \tilde{A}, N, \xi; x)$

Additionally, **Sim** computes the challenge $\ell$ in the same way as an honest verifier. We first prove $\pi_{sig}$ and $\tilde{\pi_{sig}}$ are indistinguishable. Because $\vec{\alpha}$ is uniformly sampled from $[D]$ where $D > N$ before the signer generates $N$, and $N$ is generated by the trusted signer, each element in $\vec{\alpha}$ modular $N$ is uniformly random in $[N]$. Because each signature is signed by the honest signer, there is a bilinear mapping between $\alpha_j \bmod N$ and $\beta_j \bmod N$, each element in $\vec{\beta}$ modular $N$ is uniformly random in $[N]$. Because DMSS assumption holds for the set size $k$ and modulus $N$, and each element in $\vec{\beta}$ modular $N$ is uniformly random, $\tilde{\pi_{sig}}$ is computational indistinguishable from uniform in $[N]$. Similarly, for the honest prover, the integer $\omega = \prod_{\forall \beta_j \in \vec{\beta}} \beta_j^{v_j} \bmod N$ is computational indistinguishable from uniform in $\mathbb{Z}_N$. Because $\pi_{sig} = \prod_{\forall \sigma_i \in \vec{\sigma'}} \sigma_i \times \omega \bmod N$, $\pi_{sig}$ is computational indistinguishable from uniform in $\mathbb{Z}_N$ and computational indistinguishable from $\tilde{\pi_{sig}}$. Similarly, $\xi$ and $\tilde{\xi}$ are computational indistinguishable.

Because $\vec{\alpha}$ is uniformly sampled from $[D]$, $D > \text{maxord}(\mathbb{G}_?)$, and the DMSS assumption holds for the set size $k$ and modulus $\textbf{Order}(\mathbb{G}_?)$, $\rho = \prod_{\forall \alpha_j \in \vec{\alpha}} \alpha_i^{v_i} \bmod \textbf{Order}(\mathbb{G}_?)$ is computationally indistinguishable from uniform in $[\textbf{Order}(\mathbb{G}_?)]$. Because

$$x \equiv \rho \times \prod_{\forall m_i \in M'} (\Delta + H(m_i)) \bmod \textbf{Order}(\mathbb{G}_?),$$

$x \bmod \textbf{Order}(\mathbb{G}_?)$ is computationally indistinguishable from uniform in $[\textbf{Order}(\mathbb{G}_?)]$. Similarly, $\tilde{x}$

$\bmod$ **Order**$(\mathbb{G}_?)$ is also computationally indistinguishable from uniform in $[$**Order**$(\mathbb{G}_?)]$ and computationally indistinguishable from $x \bmod$ **Order**$(\mathbb{G}_?)$. Because $A = g^x = g^{x \bmod \textbf{Order}(\mathbb{G}_?)}$, $\tilde{A} = g^{\tilde{x} \bmod \textbf{Order}(\mathbb{G}_?)}$, $A$ and $\tilde{A}$ are computationally indistinguishable.

Because the exponent $\tilde{x}$ is computationally indistinguishable from $x$, and **PoKE**$^*$ is a deterministic protocol, $\pi_1$ and $\tilde{\pi_1}$ are also computationally indistinguishable. $\boxed{c_{\vec{u}}}$ and $\boxed{\tilde{c}_{\vec{u}}}$ are statistically indistinguishable based on the perfect hiding property of the commitment scheme. $\tilde{\pi_2}$ and $\tilde{\pi_3}$ can be simulated independently using the simulator of SNARK and the **NIZK-PoKEModN** protocol. □

**Statistical zero-knowledge.** While our protocol achieves computational zero knowledge, we point out that it is possible to achieve statistical zero-knowledge with additional cost. To achieve this, the signer must generate and sign a large random number of size $2^\lambda \text{maxord}(\mathbb{G}_?)$ *for each individual proof*. This random number is then utilized by the aggregator to randomize $A$ and $\pi_1$. The computational burdens placed on the signer while the verifier remain almost consistent between the computational and statistical zero-knowledge scenarios.

## F.1. Applications of signature offloading

**On-chain Perpetual trading**, as delineated in [92], is an innovative decentralized exchange mechanism for cryptocurrencies. This system facilitates the trading of digital assets where users rely on pricing information provided by trusted sources known as Oracle Price Ticks (OPTs). OPTs provide signed price tuples in real-time *in data streams*, which consist of the asset's current price, a precise timestamp, and the asset's name. The transactions are then organized by a prover and posted on blockchains through a SNARK proof *for succinct proof sizes*. Inside the SNARK circuit, each transaction must match the tuple provided and signed by the OPT for the same timestamp.

To resist a single point of failure of OPTs, provers usually collect the prices from **at least three** different OPTs. This redundancy means for each transaction, the SNARK circuit checks not only one signature of the asset owner but also at least three signatures from OPTs. However, this requirement to validate multiple signatures introduces a significant computational burden on the prover, often emerging as a system bottleneck. To address this, compromises are made that can affect the precision and granularity of price data. For instance, provers may simplify computations by verifying within the zk-SNARK circuit that transactions are bounded by the minimum and maximum prices for the time interval for a batch of transactions. While this approach reduces the computational load, it can also attenuate the degree of fairness, as it does not account for the exact prices at the time of each transaction.

Because the perpetual trading system uses SNARK only for its proof *succinctness*, all transactions and price tuples are *naturally public*, we can solve the prover bottleneck using our basic RSA-offloading protocol to offload all the signatures of OPTs, at the cost of additional constant-sized proofs.