Efficient Noninteractive Certification of RSA Moduli and Beyond

Sharon Goldberg[1(✉)], Leonid Reyzin[1], Omar Sagga[1], and Foteini Baldimtsi[2]

[1] Boston University, Boston, MA, USA
{goldbe,reyzin}@cs.bu.edu, osagga@bu.edu
[2] George Mason University, Fairfax, VA, USA
foteini@gmu.edu

Abstract. In many applications, it is important to verify that an RSA public key (N, e) specifies a permutation over the entire space $\mathbb{Z}_N$, in order to prevent attacks due to adversarially-generated public keys. We design and implement a simple and efficient noninteractive zero-knowledge protocol (in the random oracle model) for this task. Applications concerned about adversarial key generation can just append our proof to the RSA public key without any other modifications to existing code or cryptographic libraries. Users need only perform a one-time verification of the proof to ensure that raising to the power e is a permutation of the integers modulo N. For typical parameter settings, the proof consists of nine integers modulo N; generating the proof and verifying it both require about nine modular exponentiations.

We extend our results beyond RSA keys and also provide efficient noninteractive zero-knowledge proofs for other properties of N, which can be used to certify that N is suitable for the Paillier cryptosystem, is a product of two primes, or is a Blum integer. As compared to the recent work of Auerbach and Poettering (PKC 2018), who provide two-message protocols for similar languages, our protocols are more efficient and do not require interaction, which enables a broader class of applications.

1 Introduction

Many applications use an RSA public key (N, e) that is chosen by a party who may be adversarial. In such applications, it is often necessary to ensure that the public key defines a permutation over $\mathbb{Z}_N$: that is, raising to the power e modulo N must be bijective, or, equivalently, every integer between 0 and $N-1$ must have an eth root modulo N. An attacker who deliberately generates a bad key pair may subvert the security of other users—see for example, [MRV99, CMS99, MPS00, LMRS04]. In particular, our work was motivated by TumbleBit [HAB+17], a transaction-anonymizing system deployed [Str17] on top of Bitcoin, in which a bad key pair can lead to a devastating attack (see footnote 2 in Sect. 5 for the attack specifics).

Interactive proofs for correctness of RSA keys are available (see, for example, [AP18] and references therein), but interaction with the key owner is often not

S. D. Galbraith and S. Moriai (Eds.): ASIACRYPT 2019, LNCS 11923, pp. 700–727, 2019.
https://doi.org/10.1007/978-3-030-34618-8_24

possible in the application. Thus, the folklore solution, used, for example, in [MRV99, CMS99, MPS00, LMRS04], is to choose the public RSA exponent e such that e is prime and larger than N. This solution has two major drawbacks.

First, because the folklore solution requires $e > N$, e is not in the set of standard values typically used for e in RSA implementations *e.g.,* $e \in \{3, 17, 2^{16}+1\}$. Unless a large prime value for e is standardized, before using the public key, one would have to perform a one-time primality test on e, to ensure that e really is prime. This primality test is quite expensive (see Sect. 5).

Second, most RSA implementations choose a small value for e, typically from a set of standard values $e \in \{3, 17, 2^{16} + 1\}$. Choosing a small e significantly reduces the cost of performing RSA public key operations. However, this efficiency advantage is eliminated in the folklore solution, which requires $e > N$. Unlike the previous drawback, which results in a one-time cost for each public key used, this drawback makes *every* public-key operation about two orders of magnitude more expensive.

In addition, this solution is not compatible with existing RSA standards and off-the-shelf implementations. This is because the folklore solution does not ensure that the public key operation is a permutation over $\mathbb{Z}_N$, where $\mathbb{Z}_N = \{0, 1, ..., N - 1\}$. Instead, it ensures only that the public key operation defines a permutation over the set $\mathbb{Z}_N^*$, where $\mathbb{Z}_N^*$ is the set of values in $\mathbb{Z}_N$ that are relatively prime with N. Thus, there are no assurances about the values in the set $\mathbb{Z}_N - \mathbb{Z}_N^*$, *i.e.,* the set of values that are less than N but *not* relatively prime with N. (To see this, consider the example $N = 9$ and $e = 11$.) If the RSA public key is generated honestly, this is not a problem, because the set $\mathbb{Z}_N - \mathbb{Z}_N^*$ contains only a negligible fraction of $\mathbb{Z}_N$. However, if an adversary chooses the RSA public key (N, e) maliciously, then it could choose N so that the set $\mathbb{Z}_N - \mathbb{Z}_N^*$ is a large fraction of $\mathbb{Z}_N$.[1] To address this attack, the folklore solution additionally requires a gcd check along with *every* RSA public-key operation, to ensure that the exponentiated value is relatively prime with N.

1.1 Our Contributions

Proving that an RSA Key Specifies a Permutation over all of $\mathbb{Z}_N$ We present a simple noninteractive zero-knowledge proof (NIZK) in the random oracle model, that allows the holder of an RSA secret key to prove that the corresponding public key defines a permutation over all of $\mathbb{Z}_N$, without leaking information about the corresponding secret key. Our NIZK can be used even when the RSA exponent e is small, which is useful for applications that require fast RSA public key operations. In addition to the NIZK algorithm and a concrete security proof,

[1] It has been observed that such an N could be detected by checking if N has small divisors. However, the risk of being detected is not usually an adequate deterrent, unless implemented and deployed as part of a protocol. But if such a check is deployed, then the adversary, knowing what check has been deployed, could set divisors of N to be just slightly larger than the limits of the check, and thus still ensure that $\mathbb{Z}_N - \mathbb{Z}_N^*$ is a nonnegligible fraction of $\mathbb{Z}_N$.

we present a detailed specification of the prover and verifier algorithms, as well as production-quality implementation and an analysis of its performance. Because our NIZK is for all values in $\mathbb{Z}_N$, it is compliant with existing cryptographic specifications of RSA (*e.g.,* RFC8017 [MKJR16]).

For typical parameter settings, our NIZK consists of 9 elements of $\mathbb{Z}_N$. Generating the NIZK costs roughly 9 full-length RSA exponentiations modulo N. Meanwhile, each verifier pays the one-time cost of verifying our NIZK, which is also roughly equal to 9 full-length exponentiations. When compared to the folklore solution we described earlier, our solution (1) avoids the more expensive one-time primality test and (2) allows the verifier to continue using a small value of e, resulting in better performance for every public-key verification.

We view this result as of most immediate practical applicability (in fact, it has already been deployed). We therefore present not only a high-level explanation of this protocol (Sect. 3.3), but also its detailed specification (Appendix C) and implementation results (Sect. 5 and code at [cod]).

Suitability for Paillier and Other Properties of N. We also present simple NIZK proofs for several other properties of N, such as ensuring that N is square-free (Sect. 3.2), is suitable for Paillier encryption (required in [Lin17] and [HMRT12]; see Sect. 3.2), is a product of exactly two primes (Sect. 3.4), or is a Blum integer (i.e., product of two primes that are each 3 modulo 4; see Sect. 3.5). Most of these problems have been addressed only via interactive protocols in prior literature [AP18]. Noninteractive proofs have considerably broader applicability than interactive ones, because the owner of the public key can simply generate a nonineractive proof once and publish it once together with the public key, whereas in the interactive, the owner needs to be online, handle potentially high query loads, and be subject to denial of service attacks.

Our proofs for square-freeness and suitability for Paillier are of similar efficiency to the permutation proof, requiring only 8 elements in Z_N for typical parameter settings and 8 full-length modular exponentiations. Our proofs for products of two primes and Blum integers require the proof of square-freeness and a test for prime powers (same as in [AP18]), plus one more component, which is less efficient for the prover, but more efficient for the verifier. For 128-bit security, this additional component requires about 1420 square root operations $\mathbb{Z}_N$ by the prover (note, however, that this is done one-time during key generation). The verifier, on the other hand, needs to perform only Jacobi-symbol computations and modular squarings, which are much more efficient, making the verifier cost comparable to the cost of just a few full-length modular exponentiation. This additional component requires the publication of 1420 elements of Z_N.

All of our protocols are presented first as two-message public-coin honest-verifier protocols. We then convert them to noninteractive using the Fiat-Shamir heuristics, by obtaining the verifier's public-coin message through an application of the random oracle to the protocol input (see Sect. 4). They all have perfect completeness, perfect honest-verifier zero-knowledge, and statistical soundness, with the exception of the protocol for showing that N is a product of two primes, which has computational honest-verifier zero-knowledge under the quadratic residuosity assumption.

1.2 Related Work

Auerbach and Poettering [AP18] present two-message interactive protocols in the random oracle model for the same problems as we consider, with the exception of proving that (N, e) specifies a permutation over $\mathbb{Z}_N$ (they prove only that (N, e) specifies a permutation over $\mathbb{Z}_N^*$, which, would require users to modify their RSA implementations to add a gcd computation to every public-key operation). As already mentioned, noninteractive protocols have broader applicability than interactive ones. It is much more appealing to be able to post, say an RSA public key along with a NIZK proof of being well formed, as opposed to be expected to run an online, interactive protocol with each verifier. Their protocols for proving that (N, e) specifies a permutation, N is square-free, or is suitable for Paillier, are all considerably less efficient than ours, requiring 81–128 modular exponentiations for 128-bit security level. Their protocols for proving that N is a product of exactly two primes or is a Blum integer are also less efficient for the verifier (because the first step in those protocols is proving square-freeness); they are about 10 times more efficient for the prover if we consider only one-time use, but, because they are interactive, they must be run repeatedly by the prover, while in our noninteractive case, the prover needs to run them only once.

Kakvi, Kiltz, and May [KKM12] show how to verify that RSA is a permutation by providing only the RSA public key (N, e) and no additional information, as long as $e > N^{1/4}$. They also show that when e is small, it is impossible, under reasonable complexity assumptions, to verify that (N, e) is a permutation without any additional information [KKM12, Section 1]. Thus, their approach cannot be used when e is small. We circumvent their impossibility by having the prover additionally provide our NIZK (rather than just (N, e)) to the verifier.

Wong, Chan, and Zhu [WCZ03, Section 3.2] and Catalano, Pointcheval, and Pornin [CPP07, Appendix D.2] present interactive protocols (using techniques similar to ours) that, like the protocols of [AP18], also work only over $\mathbb{Z}_N^*$ rather than the entire $\mathbb{Z}_N$.

The protocols given by Camenisch and Michels [CM99, Section 5.2] and Benhamouda et al. [BFGN17] achieve much stronger goals. The former proves $N = pq$ is a product of two safe primes (*i.e.,* $p, q, (p-1)/2$, and $(q-1)/2$ are all prime); the second can prove that any prespecified procedure for generating the primes p and q was followed. These protocols can be used to prove that (N, e) specifies a permutation by imposing mild additional conditions on e (and the prime generation procedure for [BFGN17]). However, these stronger goals are not necessary for our purposes. Our protocol is considerably simpler and more efficient, and does not restrict p and q in any way.

Our protocol for showing that (N, e) specifies a permutation over $\mathbb{Z}_N$ builds on the protocol of Bellare and Yung [BY96], who showed how to prove that any function is “close” to a permutation. However, “close” is not good enough for our purposes, because the adversary may be able to force the honest parties to use the few values in $\mathbb{Z}_N$ at which the permutation property does not hold. Thus, additional work is required for our setting. This additional work is accomplished with the help of a simple sub-protocol from Gennaro, Micciancio, and Rabin [GMR98, Section 3.1] for showing the square-freeness of N (a similar sub-protocol in the interactive setting was discovered earlier by Boyar, Friedl, and Lund [BFL89, Section 2.2]). We demonstrate how to combine the ideas of [BY96] and [GMR98] to prove that (N, e) specifies a permutation over $\mathbb{Z}_N$.

2 Preliminaries

Some number-theoretic preliminaries are presented in Appendix A.

Here, we first recall the standard notion of honest-verifier zero-knowledge (HVZK).

Definition 1. *(Honest-Verifier Zero Knowledge (HVZK)) An interactive proof system between a prover and verifier (P, V) for a NP language L is said to be* honest-verifier zero knowledge *if the following properties hold:*

1. *(perfect) Completeness. For every $x \in L$ and every NP-witness w for x,*

$$Pr[\langle P(x,w), V(x)\rangle = 1] = 1.$$

2. *(statistical) Soundness. For every $x \notin L$ and every interactive algorithm P^**

$$Pr[\langle P^*(x), V(x)\rangle = 1] = \mathrm{negl}(|x|)$$

3. *HVZK. There exists a probabilistic polynomial-time simulator S such that for all $x \in L$ and all PPT distinguishers D we have:*

$$\mathtt{view}D^{\langle P(x,w), V(x)\rangle} \approx \mathtt{view}D^{S(x)}.$$

We say (P, V) is *public coin* if all the messages sent by verifier V to prover P are random coin tosses.

Promise Problems. We also recall the notion of a promise problem, which is a generalization of the notion of a language. A promise problem consists of two disjoint sets: $\mathrm{L_{yes}}$ and $\mathrm{L_{no}}$. In a language, $\mathrm{L_{no}} = \overline{\mathrm{L_{yes}}}$, but in a promise problem, there may be strings that are neither in $\mathrm{L_{yes}}$ nor $\mathrm{L_{no}}$, and we generally do not care what happens if such a string is input. Thus, in a ZK proof for a promise problem, completeness and zero-knowledge need to hold for inputs in $\mathrm{L_{yes}}$, while soundness needs to hold for inputs in $\mathrm{L_{no}}$.

3 HVZK Proofs for Properties of N and e

3.1 HVZK Proof for a Permutation over $\mathbb{Z}_N^*$

Bellare and Yung [BY96] showed how to certify that any function is close to a permutation. The idea is to simply ask the prover to invert the permutation on random points. It is a standard fact from number theory (Lemma 8) that raising to eth power is either a permutation of $\mathbb{Z}_N^*$ or very far from one—in fact, it is either a permutation or an e'-to-1 function, where e' is the smallest prime divisor of e. Here, we adapt the protocol of [BY96] to show that the RSA function is not just close to a permutation, but is actually a permutation over $\mathbb{Z}_N^*$: if we check that e' is high enough, then not many random points will be needed.

It is also a standard fact (recalled in Lemma 2) that raising to the power e defines a permutation over $\mathbb{Z}_N^*$ if and only if e is relatively prime to $\phi(N)$. Thus, let

$$\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*} = \{(N, e) \mid N, e > 0 \text{ and } \gcd(e, \phi(N)) = 1\}.$$

Let

$$\mathrm{L}_{e'} = \{(N, e) \mid N, e > 0 \text{ and no prime less than } e' \text{ divides } e\}.$$

(In typical RSA implementations, e is a fixed small prime, such as 3, 17, or 65537, and one would use $e' = e$.)

The following is an HVZK protocol for $\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*} \cap \mathrm{L}_{e'}$ with perfect completeness, perfect zero-knowledge, and statistical soundness error $2^{-\kappa}$. We emphasize that, while the protocol is similar to that of [BY96], it is not identical. Specifically, the addition of $\mathrm{L}_{e'}$ and the verifier check in Step 4a allow us to guarantee that the RSA function is a permutation over $\mathbb{Z}_N^*$ much more efficiently than the protocol of [BY96].

Protocol $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N^*}$

1. Both prover and verifier let $m = \lceil \kappa / \log_2 e' \rceil$.
2. The verifier chooses m random values $\rho_i \in \mathbb{Z}_N^*$ and sends them to prover.
3. The prover sends back eth roots of ρ_i modulo N:

$$\sigma_i = (\rho_i)^{e^{-1} \bmod \phi(N)} \bmod N$$

   for $i = 1 \ldots m$.
4. The verifier accepts that $N \in \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*} \cap \mathrm{L}_{e'}$ if all of the following checks pass.
   (a) Check that N, e, and σ_i for $i = 1 \ldots m$ are positive integers, and that e not divisible by all the primes less than e' (if e is a fixed prime as in typical RSA implementations, this check simply involves checking that $e = e'$).
   (b) Verify that $\rho_i = (\sigma_i)^e \bmod N$ for $i = 1 \ldots m$.

Theorem 1. *$\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N^*}$ is a 2-message public-coin protocol with perfect completeness, perfect honest-verifier zero-knowledge, and statistical soundness error $2^{-\kappa}$ for the language $\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*} \cap \mathrm{L}_{e'}$.*

Proof. It is a standard fact that raising to the power N is a permutation of $\mathbb{Z}_N^*$ whenever $e \in \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}$, and the inverse of this permutation is raising to the power $(e^{-1} \bmod \phi(N))$ (see Lemma 2). This fact gives perfect completeness and a perfect HVZK simulator who simply chooses σ_i and computes $\rho_i = (\sigma_i)^e \bmod N$ for $i = 1 \dots m$ (recall that by definition, completeness and HVZK apply only to $(N, e) \in \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*} \cap \mathrm{L}_{e'}$). Statistical soundness with error $2^{-\kappa}$ follows from the fact that if $(N, e) \notin \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}$ but $(N, e) \in \mathrm{L}_{e'}$, then size the image of the map $\sigma \mapsto \sigma^e$ is at most $|\mathbb{Z}_N^*|/e'$ by Lemma 8. Thus, the probability that a σ_i exists for every ρ_i is at most $1/(e')^m = 2^{-m \log_2 e'} \leq 2^{-\kappa}$. $\square$

3.2 HVZK Proofs for Paillier and Square-Free N

The Paillier cryptosystem requires a modulus N that is relatively prime with $\phi(N)$. Thus, let

$$\mathrm{L}_{\mathsf{paillier}\text{-}N} = \{N > 0 \mid \gcd(N, \phi(N)) = 1\}.$$

We emphasize that, unlike [AP18], we do not verify the properties of the generator g in the Paillier cryptosystem—but since the common choice is to use $g = N + 1$ per [DJ01], verifying that $N \in \mathrm{L}_{\mathsf{paillier}\text{-}N}$ is sufficient for the common case.

Let

$$\mathrm{L}_{\mathsf{square\text{-}free}} = \{N > 0 \mid \text{ there is no prime } p \text{ such that } p^2 \text{ divides } N\}.$$

Note that to be in $\mathrm{L}_{\mathsf{paillier}\text{-}N}$, N has to be in $\mathrm{L}_{\mathsf{square\text{-}free}}$ and also have no prime divisors p, q such that $p \mid q - 1$ (by definition of $\phi(N)$, as recalled in Appendix A), so $\mathrm{L}_{\mathsf{paillier}\text{-}N} \subset \mathrm{L}_{\mathsf{square\text{-}free}}$ (see Lemma 3).

Thus, letting

$$\mathrm{L}_{\mathsf{gap}} = \{N \in \mathrm{L}_{\mathsf{square\text{-}free}} | N \text{ has two prime divisors } p, q \text{ such that } p \text{ divides } q - 1\},$$

we know that $\mathrm{L}_{\mathsf{square\text{-}free}} - \mathrm{L}_{\mathsf{gap}} = \mathrm{L}_{\mathsf{paillier}\text{-}N}$.

Our protocols for proving suitability for Paillier or square-freeness will depend on a parameter α and the corresponding language

$$\mathrm{L}_\alpha = \{N > 0 \mid \text{no prime less than } \alpha \text{ divides } N\}.$$

We now describe the protocol $\mathcal{P}_{\mathsf{paillier}\text{-}N}$, an HVZK protocol for $\mathrm{L}_\alpha \cap \mathrm{L}_{\mathsf{paillier}\text{-}N}$ with perfect completeness, perfect zero-knowledge, and statistical soundness error $2^{-\kappa}$. This protocol builds on the protocol from [GMR98, Section 3.1], but is not identical to it: specifically, the addition of L_α and verifier's Step 4a gives better performance. Setting $\alpha = 2$ gives a protocol for $\mathrm{L}_{\mathsf{paillier}\text{-}N}$, but a higher setting of α will improve efficiency (see Sect. 5 for a discussion of how to pick α).

The idea of the protocol is to ask the prover to take Nth roots of random points—they will not exist for many points if $N \notin \mathrm{L}_{\mathsf{paillier}\text{-}N}$, because raising to the power N will be far from a permutation. The protocol is the same as the protocol $\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}$ described in Sect. 3.1, replacing e with N and e' with α.

Protocol $\mathcal{P}_{\mathsf{pailler}\text{-}N}$:

1. Both prover and verifier let $m = \lceil \kappa / \log_2 \alpha \rceil$.
2. The verifier chooses m random values $\rho_i \in \mathbb{Z}_N^*$ and sends them to prover.
3. The prover sends back Nth roots of ρ_i modulo N:
$$\sigma_i = (\rho_i)^{N^{-1} \bmod \phi(N)} \bmod N$$
for $i = 1 \dots m$.
4. The verifier accepts that $N \in \mathrm{L}_{\mathsf{pailler}\text{-}N} \cap \mathrm{L}_\alpha$ if all of the following checks pass.
   (a) Check that N is a positive integer and is not divisible by all the primes less than α.
   (b) Check that σ_i is a positive integer for $i = 1 \dots m$.
   (c) Verify that $\rho_i = (\sigma_i)^N \bmod N$ for $i = 1 \dots m$.

Theorem 2 (GMR98). $\mathcal{P}_{\mathsf{pailler}\text{-}N}$ *is a 2-message public-coin proof with perfect completeness, perfect honest-verifier zero-knowledge, and statistical soundness error* $2^{-\kappa}$ *for the language* $\mathrm{L}_\alpha \cap \mathrm{L}_{\mathsf{pailler}\text{-}N}$.

Note that choosing elements in $\mathbb{Z}_N^*$ in step 2 of the protocol requires a gcd computation by the verifier (because the verifier cannot be sure that the difference between $\mathbb{Z}_N$ and $\mathbb{Z}_N^*$ is negligible). To avoid this computation, the verifier can choose values in $\mathbb{Z}_N$ instead. Then the verifier may have a lower probability of rejecting inputs outside of $\mathrm{L}_\alpha \cap \mathrm{L}_{\mathsf{pailler}\text{-}N}$, but is still guaranteed to reject inputs outside of $\mathrm{L}_\alpha \cap \mathrm{L}_{\mathsf{square\text{-}free}}$ with probability $1 - 2^{-\kappa}$, as we show in Lemma 6. Perfect completeness and zero-knowledge still hold for $\mathrm{L}_\alpha \cap \mathrm{L}_{\mathsf{pailler}\text{-}N}$, and thus for an honestly generated RSA modulus. Let us call this modified protocol $\mathcal{P}_{\mathsf{square\text{-}free}}$.

Protocol $\mathcal{P}_{\mathsf{square\text{-}free}}$: Same as the protocol $\mathcal{P}_{\mathsf{pailler}\text{-}N}$ described above, replacing Z_N^* with Z_N in step 2 and $N \in \mathrm{L}_{\mathsf{pailler}\text{-}N} \cap \mathrm{L}_\alpha$ with $N \in \mathrm{L}_{\mathsf{square\text{-}free}} \cap \mathrm{L}_\alpha$ in step 4. Specifically,

1. Both prover and verifier let $m = \lceil \kappa / \log_2 \alpha \rceil$.
2. The verifier chooses m random values $\rho_i \in \mathbb{Z}_N$ and sends them to prover.
3. The prover sends back Nth roots of ρ_i modulo N:
$$\sigma_i = (\rho_i)^{N^{-1} \bmod \phi(N)} \bmod N$$
for $i = 1 \dots m$.
4. The verifier accepts that $N \in \mathrm{L}_{\mathsf{square\text{-}free}} \cap \mathrm{L}_\alpha$ if all of the following checks pass.
   (a) Check that N is a positive integer and is not divisible by all the primes less than α.
   (b) Check that σ_i is a positive integer for $i = 1 \dots m$.
   (c) Verify that $\rho_i = (\sigma_i)^N \bmod N$ for $i = 1 \dots m$.

Theorem 3. $\mathcal{P}_{\mathsf{square\text{-}free}}$ *is a 2-message public-coin proof with perfect completeness, perfect honest-verifier zero-knowledge, and statistical soundness error* $2^{-\kappa}$ *for the promise problem* $(\mathrm{L}_{\mathrm{yes}} = \mathrm{L}_\alpha \cap \mathrm{L}_{\mathsf{paillier}\text{-}N}, \mathrm{L}_{\mathrm{no}} = \overline{\mathrm{L}_\alpha \cap \mathrm{L}_{\mathsf{square\text{-}free}}})$.

Proof. It is a standard fact that raising to the power N is a permutation of $\mathbb{Z}_N$ whenever $N \in \mathrm{L}_{\mathsf{paillier}\text{-}N}$, and the inverse of this permutation is raising to the power $(N^{-1} \bmod \phi(N))$ (see Lemmas 3 and 4, setting $f = N$). This fact gives perfect completeness and a perfect HVZK simulator who simply chooses σ_i and computes $\rho_i = (\sigma_i)^N \bmod N$ for $i = 1 \ldots m$ (recall that by definition, completeness and HVZK apply only to $N \in \mathrm{L}_{\mathrm{yes}}$). Statistical soundness with error $2^{-\kappa}$ follows from the fact that if $p \geq \alpha$ is a prime such that $p^2 | N$, then the map $\sigma \mapsto \sigma^N$ is at least α-to-1 over Z_N (per Lemma 6); thus, the probability that a σ_i exists for every ρ_i is at most $1/\alpha^m = 2^{-m \log_2 \alpha} \leq 2^{-\kappa}$. □

3.3 HVZK Proof for Permutation over Entire $\mathbb{Z}_N$

As explained in the introduction, ensuring that raising to the power e is a permutation over the entire $\mathbb{Z}_N$ is more desirable than ensuring only that it is a permutation over $\mathbb{Z}_N^*$. In this section, we show that a careful combination of protocols $\mathcal{P}_{\mathsf{square\text{-}free}}$ and $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N^*}$ gives an efficient two-message public-coin HVZK protocol for proving that an RSA public key defines a permutation over the entire $\mathbb{Z}_N$.

Let $\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}$ and $\mathrm{L}_{e'}$ be as in Sect. 3.1, and L_α, $\mathrm{L}_{\mathsf{square\text{-}free}}$, $\mathrm{L}_{\mathsf{paillier}\text{-}N}$, and $\mathrm{L}_{\mathsf{gap}}$ be as in Sect. 3.2, except defined not just on integers N, but on pairs (N, e) for an arbitrary $e > 0$.

Let $\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N} = \{(N, e) \,|\, N, e > 0$ and raising to the power e is a permutation over $\mathbb{Z}_N\}$.

Note that

$$\left(\mathrm{L}_{\mathsf{paillier}\text{-}N} \cap \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}\right) \subset \left(\mathrm{L}_{\mathsf{square\text{-}free}} \cap \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}\right) \subset \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N} .$$

(the first $\subset$ property follows from Lemma 3; the second $\subset$ property follows from Lemma 4). Note that the only pairs (N, e) in $\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N} - \left(\mathrm{L}_{\mathsf{square\text{-}free}} \cap \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}\right)$ are those for which $e = 1$ and N is not square-free (per Lemma 5).

We want to design a protocol for $\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N}$. For efficiency reasons, we will focus instead on $\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N} \cap \mathrm{L}_\alpha \cap \mathrm{L}_{e'}$, i.e., require N and e to not have divisors smaller than α and e', respectively. Moreover, just like in protocol $\mathcal{P}_{\mathsf{square\text{-}free}}$ of Sect. 3.2, we will consider slightly weaker completeness: if N is square-free, but has two prime divisors p, q such that $p \,|\, (q-1)$ (i.e., falls into $\mathrm{L}_{\mathsf{gap}}$), the verifier will be permitted to reject N. Thus, let

$$\begin{aligned} \mathrm{L}_{\mathrm{yes}} &= \mathrm{L}_{\mathsf{paillier}\text{-}N} \cap \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*} \cap \mathrm{L}_\alpha \cap \mathrm{L}_{e'} \\ \mathrm{L}_{\mathrm{no}} &= \overline{\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N}} \cup \overline{\mathrm{L}_\alpha} \cup \overline{\mathrm{L}_{e'}} \end{aligned}$$

The gap between $\mathrm{L}_{\mathrm{yes}}$ and L_{no} (i.e., the only pairs (N, e) not in $\mathrm{L}_{\mathrm{yes}} \cup \mathrm{L}_{\mathrm{no}}$) is almost the same as in Theorem 3: namely, $\mathrm{L}_{\mathsf{gap}} \cap \mathrm{L}_{e'} \cap \mathrm{L}_\alpha$, as well as some pairs

(N, e) with $e = 1$. Naturally occurring RSA moduli should never fall into this gap. Every (N, e) in the gap still defines a permutation over the entire $\mathbb{Z}_N$, but the prover may be unable to show this fact.

We now present a protocol $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N}$ for the promise problem $(\mathrm{L}_{\mathrm{yes}}, \mathrm{L}_{\mathrm{no}})$. The protocol $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N}$ is not simply a combination of $\mathcal{P}_{\mathsf{square\text{-}free}}$ and $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N^*}$: we save space by using the same ρ_i for both eth roots and Nth roots. Because any value that has an (eN)th root also has an eth root and an Nth root, we combine the two protocols simply by checking the ρ_i values have eNth roots.

The protocol $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N}$ depends on two parameters α and e', which are both primes, at most about 16 bits long. The verifier will reject any N that is divisible by a prime less than α and any e that is divisible by a prime less than e'. Any setting of α and e' is valid for security; varying these parameters affects only efficiency. An optimal setting of these parameters is implementation-dependent, since larger e' and α will result in some additional work for the verifier, but will also reduce work for the prover and verifier since m_1 and m_2 in Eq. (1) below become smaller. When e is a fixed prime like 3, 17, or $2^{16} + 1$, as is standard for many RSA implementations, then we set e' equal to e. We further discuss parameter settings in Sect. 5.

The prover's witness is the prime factorization of N. Recall that κ is a security parameter. The protocol will achieve statistical soundness error $2^{-\kappa}$.

Protocol $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N}$:

1. Both prover and verifier let
$$m_1 = \lceil \kappa / \log_2 \alpha \rceil \text{ and } m_2 = \left\lceil -\kappa / \log_2 \left(\frac{1}{\alpha} + \frac{1}{e'} \left(1 - \frac{1}{\alpha} \right) \right) \right\rceil . \qquad (1)$$
Notice that $m_2 \geq m_1$ since $e' > 1$.
2. The verifier chooses m_2 random values $\rho_i \in Z_N$ and sends them to Prover.
3. The Prover sends back
$$\sigma_i = (\rho_i)^{(eN)^{-1} \bmod \phi(N)} \pmod N$$
for $i = 1 \ldots m_1$ (for convenience, we call this a "weird RSA signature") and
$$\sigma_i = (\rho_i)^{e^{-1} \bmod \phi(N)} \pmod N$$
for $i = m_1 + 1 \ldots m_2$ (which is just a regular RSA signature).
4. The verifier accepts that (N, e) defines a permutation over all of $\mathbb{Z}_N$ if all of the following checks pass.
   (a) Check that $N > 0$ and N is not divisible by all the primes less than α. (Equivalently, one can let P be the product of all primes less than α (also known as $\alpha - 1$ primorial) and verify that $\gcd(N, P) = 1$).
   (b) Check that $e > 0$ and is e not divisible by all the primes less than e'. (In most implementations of RSA, e is a fixed prime, in which case the verifier can just check that $e = e'$).
   (c) Verify that $\rho_i = (\sigma_i)^{eN} \pmod N$ for $i = 1 \ldots m_1$.
   (d) Verify that $\rho_i = (\sigma_i)^{e} \pmod N$ for $i = m_1 + 1 \ldots m_2$.

Note that for many natural choices of parameters (e, κ, α), we have $m_1 = m_2$, and so step 4d disappears.

Theorem 4. $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N}$ *is a 2-message public-coin proof with perfect completeness, perfect honest-verifier zero-knowledge, and statistical soundness error* $2^{-\kappa}$ *for the promise problem*

$$\mathrm{L}_{\mathrm{yes}} = \mathrm{L}_{\mathsf{paillier}\text{-}N} \cap \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*} \cap \mathrm{L}_\alpha \cap \mathrm{L}_{e'}$$
$$\mathrm{L}_{\mathrm{no}} = \overline{\mathrm{L}_{\mathsf{perm}\mathbb{Z}_N}} \cup \overline{\mathrm{L}_\alpha} \cup \overline{\mathrm{L}_{e'}}$$

Proof. It is a standard fact (per Lemmas 3 and 4) that raising to a power f is a permutation of $\mathbb{Z}_N$ whenever $N \in \mathrm{L}_{\mathsf{paillier}\text{-}N}$ and $\gcd(f, \phi(N)) = 1$, and that the inverse of this permutation is raising to the power $(f^{-1} \bmod \phi(N))$. This fact, when we set $f = eN$ for $i = 1 \ldots m_1$ and $f = N$ for $i = m_1 + 1, \ldots, m_2$, gives perfect completeness. It also gives a perfect HVZK simulator who simply chooses σ_i and computes $\rho_i = (\sigma_i)^{eN} \bmod N$ for $i = 1 \ldots m_1$ and $\rho_i = (\sigma_i)^e$ for $i = m_1 + 1, \ldots, m_2$ (recall that, by definition, the simulator needs to work only for $(N, e) \in \mathrm{L}_{\mathrm{yes}}$).

To show soundness, suppose $(N, e) \in \mathrm{L}_{\mathrm{no}}$. If $x \in \overline{\mathrm{L}_\alpha} \cup \overline{\mathrm{L}_{e'}}$, the verifier will reject in steps 4a or 4b, and soundness holds. Therefore, assume $(N, e) \in \mathrm{L}_\alpha \cap \mathrm{L}_{e'}$. This means $(N, e) \notin \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N}$.

Suppose $(N, e) \notin \mathrm{L}_{\mathsf{square\text{-}free}}$. Since the smallest prime divisor of N is at least α, by applying Lemma 6, we know at most $1/\alpha$ fraction of $\mathbb{Z}_N$ will have an Nth root. By choosing m_1 elements of $\mathbb{Z}_N$ and verifying that they have Nth roots, we ensure that the chances that the prover passes Step 4c with N that is not square-free are at most $(1/\alpha)^{m_1} \leq 2^{-\kappa}$.

Now suppose $(N, e) \in \mathrm{L}_{\mathsf{square\text{-}free}}$ but $(N, e) \notin \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N}$. Since N is square free, the smallest prime divisor of N is at least α, and the smallest prime divisor of e is at least e', we can apply Lemma 7 to conclude that at most $1/\alpha + (1 - 1/\alpha)/e'$ fraction of $\mathbb{Z}_N$ have an eth root. By choosing m_2 elements of $\mathbb{Z}_N$ and verifying that they have eth roots, we ensure that the chances that the prover passes Steps 4c and 4d are at most

$$\left(\frac{1}{\alpha} + \frac{1}{e'}\left(1 - \frac{1}{\alpha}\right)'\right)^{m_2} \leq 2^{-\kappa}.$$

□

A Possible Optimization. Instead of choosing the ρ_i values from $\mathbb{Z}_N$, the prover could choose ρ_i values from $\mathbb{Z}_N^*$ (this requires m_2 gcd computations), and set a potentially lower $m_2 = \max\left(\lceil \kappa / \log_2 e' \rceil, m_1\right)$. The proofs of completeness and zero-knowledge proofs remain the same (because if (N, e) define a permutation over $\mathbb{Z}_N$, they also define a permutation when restricted to $\mathbb{Z}_N^*$). The proof of soundness changes in the last paragraph. Observe that, since $\left(\mathrm{L}_{\mathsf{square\text{-}free}} \cap \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}\right) \subset \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N}$, if $(N, e) \in \mathrm{L}_{\mathsf{square\text{-}free}}$ but $(N, e) \notin \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N}$, then $(N, e) \notin \mathrm{L}_{\mathsf{perm}\mathbb{Z}_N^*}$. Thus, per Lemma 8, the chances that the prover passes steps 4c and 4d are at most

$$\left(\frac{1}{e'}\right)^{m_2} \leq 2^{-\kappa}\,.$$

For example, for $\kappa = 128$ and $e = \alpha = 65537$, this optimization reduces the value of m_2 from 9 to 8. This reduction in m_2 is at the expense of $\gcd(\rho_i, N)$ computations, and so it may or may not improve overall performance, depending on the implementation and the parameter values. We emphasize, however, that the lower m_2 value will not give security $2^{-\kappa}$ without the gcd computations on the part of the verifier, so implementers of this optimization should ensure the verifier rejects if $\gcd(\rho_i, N) \neq 1$ for some i.

3.4 HVZK Proof for a Product of Two Primes

In this section, we consider the language

$$\mathrm{L}_{\mathsf{ppp}} = \{N > 0 \mid N \text{is odd and has exactly two distinct prime divisors}\}\,.$$

Note that the more interesting language is

$$\mathrm{L}_{\mathsf{pp}} = \{N > 0 \mid N \text{is odd and is a product of two distinct primes }\} = \\ (\mathrm{L}_{\mathsf{ppp}} \cap \mathrm{L}_{\mathsf{square\text{-}free}}) \supset (\mathrm{L}_{\mathsf{ppp}} \cap \mathrm{L}_{\mathsf{pailler}\text{-}N})\ ,$$

because it rules out prime powers as factors of N.

We obtain a two-round public-coin HVZK proof for the promise problem $\mathrm{L}_{\mathrm{yes}} = \mathrm{L}_{\mathsf{pp}}$ and $\mathrm{L}_{\mathrm{no}} = \overline{\mathrm{L}_{\mathsf{ppp}}}$ (note that only N not in $\mathrm{L}_{\mathrm{yes}} \cup \mathrm{L}_{\mathrm{no}}$ are those that have exactly two distinct odd prime divisors and are not square-free). We can obtain an HVZK proof for L_{pp} (with a similar gap for the case $p|q-1$) by combing the protocol in this section with the protocol for and $\mathrm{L}_{\mathsf{square\text{-}free}}$, similar to [AP18]. The combination can be space-saving, similar to Protocol $\mathcal{P}_{\mathsf{perm}\mathbb{Z}_N}$ in Sect. 3.3.

Let J_N denote the subset of $\mathbb{Z}_N^*$ with Jacobi symbol 1. Let QR_N denote the subset of J_N that consists of quadratic residues in $\mathbb{Z}_N^*$. The following is an HVZK protocol for for the promise problem ($\mathrm{L}_{\mathrm{yes}} = \mathrm{L}_{\mathsf{pp}}$, $\mathrm{L}_{\mathrm{no}} = \overline{\mathrm{L}_{\mathsf{ppp}}}$). Let κ be the statistical security parameter.

Protocol $\mathcal{P}_{\mathsf{ppp}}$

1. Both the Prover and the Verifier let $m = \lceil \kappa \cdot 32 \cdot \ln 2 \rceil$.
2. The Verifier chooses m random values $\rho_i \in J_N$ and sends them to Prover.
3. For every $\rho_i \in QR_N$, the Prover sends back $\sigma_i \in Z_N^*$ such that $\sigma_i^2 \bmod N = \rho_i$. Of the four square roots, the Prover chooses one at random. For other ρ_i, the prover sends back 0.
4. Verifier first checks that N is a positive odd integer and is not a prime or a prime power (see [Ber98, BLP07] and references therein). If these checks pass, then the Verifier accepts if the number of nonzero responses is at least $3m/8$, and for every nonzero σ_i, it holds that $\rho_i = (\sigma_i)^2 \bmod N$.

Note that our design choice to have the verifier pick values in J_N rather than in all of Z_N^* results in improved efficiency by a factor of four as compared to the hash-then-solve protocol presented in [AP18]. This is because when the verifier chooses elements in J_N, at least 1/2 of them have square roots for $N \in \mathrm{L}_{\mathsf{pp}}$, vs. 1/4 for $N \notin \mathrm{L}_{\mathsf{ppp}}$. In contrast, when the verifier chooses elements in all of Z_N^*, the fractions change to 1/4 and 1/8, respectively. But the number of repetitions m required to distinguish 1/4 from 1/8 is four times greater than the number of repetitions required to distinguish 1/2 from 1/4, for any fixed confidence level $2^{-\kappa}$ (this follows from bounds on the tail of the binomial distribution; see the proof of Theorem 5).

Theorem 5. *$\mathcal{P}_{\mathsf{ppp}}$ is a 2-message public-coin protocol for the promise problem ($\mathrm{L}_{\text{yes}} = \mathrm{L}_{\mathsf{pp}}$, $\mathrm{L}_{\text{no}} = \overline{\mathrm{L}_{\mathsf{ppp}}}$) with statistical completeness error $2^{-\kappa}$, computational honest-verifier zero-knowledge, and statistical soundness error $2^{-\kappa}$.*

Proof. In order to show completeness, we need to show that the honest prover will be able to carry out Step 3, and the verifier's checks in Step 4 will pass. Since the prover knows the factorization of $N = pq$, it can efficiently check if $\rho_i \in QR_N$ by determining if it is a quadratic residue module each prime divisor p and q of N.

Then, given that $\rho_i \in QR_N$, it is easy for the prover to compute σ_i such that $\sigma_i^2 \bmod N = \rho_i$. To do so, the prover computes $\beta_i = \rho_i \bmod p$ and $\gamma_i = \rho_i \bmod q$. Then the prover finds solutions $\pm b$ to $\sigma_i^2 \bmod p = \beta$, and $\pm c$ to $\sigma_i^2 \bmod q = \gamma$, using any of the available algorithms for finding square roots modulo primes. Finally, the prover uses the Chinese Remainder Theorem to obtain four solutions (corresponding to pairs $(b, c), (-b, c), (b, -c), (-b, -c)$) to $\sigma_i^2 \bmod N = \rho_i$. Thus, the prover can indeed carry out Step 3.

Let us now discuss why the verifier's checks in Step 4 will pass with probability close to 1. As discussed above if $\rho_i \in QR_N$ the prover can always send back valid σ_i's. So in order to achieve completeness, we need to make sure that among the ρ_i's sent from the Verifier to the Prover in Step 2, at least $3m/8$ of them are in QR_N. Since $N \in \mathrm{L}_{\mathsf{pp}}$, $|J_N| = \phi(N)/2$ while $|QR_N| = \phi(N)/4$ (it is in this step that we use the fact that that $N \in \mathrm{L}_{\mathsf{pp}}$ and not just in $\mathrm{L}_{\mathsf{ppp}}$; because $|J_N|$ when is N is a product of two prime powers can be more than twice $|QR_N|$ if one or both the powers is even).

By applying the classic Hoeffding bound [Hoe63, Theorem 2] for $m = \lceil \kappa \cdot 32 \cdot \ln 2 \rceil$, we see that Pr[the number of ρ_i's $\in QR_N < 3m/8] < e^{-2m(1/2-3/8)^2} = 2^{-2m/(64 \ln 2)} \leq 2^{-\kappa}$. Thus we conclude that our protocol has statistical completeness with error probability at most $2^{-\kappa}$.

To show soundness, suppose that $N \notin \mathrm{L}_{\mathsf{ppp}}$, i.e., N is even, a prime, a prime power, or has at least three prime divisors. If N is even, a prime, or a prime power, the verifier will reject. If N has at least three prime divisors, then at most 1/4 of the elements of J_N have square roots. But the prover can cheat only if $3m/8$ of the ρ_i values have square roots. Thus, probability of cheating is Pr[the number of squares is $\geq 3m/8] \leq e^{-2m(3/8-1/4)^2} \leq 2^{-\kappa}$ by the Hoeffding bound.

Finally, we argue that our protocol is computational honest-verifier zero-knowledge. We first recall the QR assumption [GM84].

Assumption 6 (QR assumption). *For any $N = pq$, a randomly chosen $\rho \in J_N$, and any PPT algorithm A,*

$$\Pr[\sigma = QR(\rho) \mid N = pq,\ \rho \leftarrow J_N,\ A(\rho, N) \to \sigma \in \{\pm 1\}] \leq 1/2 + negl(\kappa).$$

The HVZK simulator (which, by definition, needs to work only when $N \in \mathrm{L}_{\mathsf{pp}}$) will pick random values σ_i and square them getting ρ_i. For each number, it will flip a coin and, depending on the coin's output the simulator will either output (σ_i, ρ_i) or $(0, \rho_i')$ for a random $\rho_i' \in J_N$. Because of the QR assumption (the distributions of J_N and QR_N are computationally indistinguishable) the view of the simulator is computationally indistinguishable from that of an honest verifier interacting with a prover. □

3.5 HVZK Proof for a Blum Integer

In this section we consider the language $\mathrm{L}_{\mathsf{blum-powers}} = \{N > 0 \mid N = p^a q^b$ for primes $p \equiv q \equiv 3 \pmod 4\}$. Note, similar to Sect. 3.4, that the more interesting language is the language of Blum integers $\mathrm{L}_{\mathsf{blum}} = \mathrm{L}_{\mathsf{square\text{-}free}} \cap \mathrm{L}_{\mathsf{blum-powers}}$.

In this section we obtain a two-round public-coin HVZK protocol for the promise problem $(\mathrm{L}_{\mathrm{yes}} = \mathrm{L}_{\mathsf{blum}}, \mathrm{L}_{\mathrm{no}} = \overline{\mathrm{L}_{\mathsf{blum-powers}}})$. We can obtain a protocol for $\mathrm{L}_{\mathsf{blum}}$ (with a similar gap for the case $p \mid q-1$ as in Sect. 3.2) by combing the proofs for $\mathrm{L}_{\mathsf{blum-powers}}$ and $\mathrm{L}_{\mathsf{square\text{-}free}}$. Remarks at the beginning of Sect. 3.4 apply here, as well.

The protocol for $\mathrm{L}_{\mathsf{blum-powers}}$ is very similar to the protocol for $\mathrm{L}_{\mathsf{ppp}}$ but instead of considering square roots, we now consider 4th roots. Note that if N is a Blum integer then among the four roots of $\rho_i \in QR_N$, one and only one is a quadratic residue.

Protocol $\mathcal{P}_{\mathsf{blum-powers}}$
Same as protocol $\mathcal{P}_{\mathsf{ppp}}$ described in Sect. 3.4 but in step 3 the prover computes 4th roots instead and in step 4 the verifier checks 4th roots.

1. Both the Prover and the Verifier let $m = \lceil \kappa \cdot 32 \cdot \ln 2 \rceil$.
2. The Verifier chooses m random values $\rho_i \in J_N$ and sends them to Prover.
3. For every $\rho_i \in QR_N$, the Prover sends back $\sigma_i \in Z_N^*$ such that $\sigma_i^4 \bmod N = \rho_i$, choosing one at random from among four possibilities. For other ρ_i, the prover sends back 0.
4. Verifier first checks that N is a positive odd integer and is not a prime or a prime power (see [Ber98, BLP07] and references therein). The Verifier accepts if the number of nonzero responses is at least $3m/8$, and for every nonzero σ_i, it holds that $\rho_i = (\sigma_i)^4 \bmod N$.

Theorem 7. *$\mathcal{P}_{\mathsf{blum-powers}}$ is a 2-message public-coin protocol with statistical completeness error $2^{-\kappa}$, perfect honest-verifier zero-knowledge, and statistical soundness error $2^{-\kappa}$, for the promise problem* $(\mathrm{L}_{\mathrm{yes}} = \mathrm{L}_{\mathsf{blum}}, \mathrm{L}_{\mathrm{no}} = \overline{\mathrm{L}_{\mathsf{blum-powers}}})$.

Proof. Similar to the proof of Theorem 5, we get statistical completeness with error $2^{-\kappa}$. The prover knowing the factorization of N can efficiently compute the 4th roots for $N \in \mathrm{L}_{\text{yes}}$, and completeness relies on receiving enough ρ_i's $\in QR_N$. Also we get the same statistical soundness error $2^{-\kappa}$.

Finally, $\mathcal{P}_{\mathsf{blum-powers}}$ achieves perfect honest-verifier zero-knowledge since -1 is always a Jacobi symbol 1 non-square. Then we can construct a simulator that, after computing ρ_i by raising a random σ_i to the fourth power, flips a coin and sends either $(0, -\rho_i)$ or (σ_i, ρ_i). $\square$

4 Making Our Protocols Noninteractive via Fiat-Shamir

We use the Fiat-Shamir paradigm [FS86] to convert each of the 2-message public-coin HVZK interactive protocols presented above into a non-interactive zero-knowledge (NIZK) protocol. The transformation is very simple, because the first message in every protocol we present always consists of the verifier sending some challenges $\rho_1, \ldots, \rho_m$ to the prover. The challenges are uniformly distributed in some space with easy membership testing (such as $\mathbb{Z}_N$ or $\mathbb{Z}_N^*$, for example).

Thus, to make our protocols noninteractive, Prover samples ρ_i by herself using the random oracle. To make sure values ρ_i are in the correct space, such as $\mathbb{Z}_N$ or $\mathbb{Z}_N^*$, the prover performs rejection sampling for each ρ_i using a counter, trying multiple random-oracle outputs until obtaining the first one that lands in the desired space. Thus, each ρ_i is obtained by computing the output of the random oracle over the concatenation of (1) the protocol input—e.g., the RSA public key (N, e); (2) a salt given as a system parameter; (3) the index i; and (4) the counter value. If the result is in the correct space, the prover uses this ρ_i; if not, she increments the counter and tries again.

Thus, the protocol input and the salt determine the set of $\rho_i \in \mathbb{Z}_N$. The verifier can therefore compute ρ_i on his own, by following the same procedure as the prover, and subsequently perform verification. Note that the verifier, just like the prover, will need to perform rejection sampling.

The noninteractive proof then is simply the message that the prover sends to the verifier in the interactive protocol.

The security of this transformation is standard; we provide some formal details in Appendix B.

5 Specification, Implementation and Performance for NIZK of Permutations over $\mathbb{Z}_N$

Specification. Here we provide a more precise specification the protocol of Sect. 3.3 made non-interactive using the Fiat-Shamir paradigm as described in Sect. 4. The goal of this specification is to make the protocol precise enough for implementation and compatibility. The full specification is available in Appendix C. It assumes e is a fixed prime and thus sets $e' = e$. It takes in

α and the salt as system parameters. The random oracle used to deterministically select the ρ_i values is a "full-domain hash" [BR93] instantiated with the industry-standard MGF1 Mask Generation Function as defined in [MKJR16, Sec. B.2.1]. We use the industry-standard I2OSP and OS2IP to convert between octet strings and integers [MKJR16, Sec. 4.1] and the industry-standard RSASP to perform an RSA secret key operations [MKJR16, Sec. 5.2.1], and RSAVP for RSA public-key operations [MKJR16, Sec. 5.2.2].

Implementation. An open-source implementation of our specification in C#, based on the bouncycastle cryptographic library [bou], is publicly available [cod]. We hope that our implementation will become a part of bouncycastle.

Integration with TumbleBit. Our implementation has already been integrated into the open-source reference implementation of TumbleBit, which is currently being developed for production use [Ntu,Str17]. TumbleBit [HAB+17] is a unidirectional Bitcoin payment hub that allows parties to make fast, anonymous, off-blockchain payments through an untrusted intermediary called the Tumbler. The security of the TumbleBit protocol rests on the assumption that the Tumbler's RSA public key (N, e) defines a permutation over $\mathbb{Z}_N$. In the absence of this assumption, the Tumbler can steal bitcoin from payers.[2] Thus, in addition to publishing (N, e), a Tumbler publishes our NIZK proof that (N, e) defines a permutation, which is verified, during a setup phase, by any payer or payee who wants to participate in the protocol with this Tumbler. Integration with TumbleBit was easy. No modification to the existing TumbleBit protocol or codebase were required; instead, our NIZK was simply added to TumbleBit's setup phase.

Parameters and Performance for TumbleBit. When used with TumbleBit, our NIZK has parameters $\kappa = 128$, the RSA key length is $|N| = 2048$, the public RSA exponent is $e = e' = 65537$, and the salt is the SHA256 hash of the Genesis block of the Bitcoin blockchain.

The performance of our NIZK largely depends on our choice of the parameter α. A shorter α means that the verifier has to spend less time trying to divide N by primes less than α, but also increases m_1 and m_2, the number of RSA values

[2] Specifically, in TumbleBit, the Tumbler provides the payee Bob with a value z called a "puzzle," and a proof that its solution will transfer some of Tumbler's money to Bob. This solution is a value ϵ such that $z = \epsilon^e \bmod N$. The protocol crucially relies on uniqueness of ϵ, because the proof that the solution will unlock money applies to only one of the solutions of z. When Alice wants to pay Bob, she learns the solution to the puzzle in exchange for paying money to the Tumbler, and then gives that solution to Bob as payment. If RSA is not a permutation, then a malicious Tumbler can provide the payee Bob with a puzzle z that has two valid solutions $\epsilon_1 \neq \epsilon_2$, where $z = (\epsilon_1)^e = (\epsilon_2)^e \mod N$, and a proof that ϵ_1 transfers money. Then, to steal money, the Tumbler gives payer Alice the solution ϵ_2 in exchange for her money, which does not permit Bob to obtain the Tumbler's money and complete the transaction.

in the NIZK. The relationship between α and m_1, m_2 is determined by Eq. (1). Specifically for the TumbleBit parameters, we show this relationship in Fig. 1. To evaluate the performance of our NIZK, we choose the smallest value of α that corresponds a given pair of (m_1, m_2) values, and benchmark proving and verifying times for our NIZK for the RSA key length $|N| = 2048$ bits in Table 1 on a single-core of an Intel Xeon processor. We can see from the table that choosing $\alpha = 319567$ (so that $m_1 = 7$ and $m_2 = 9$) gives optimal performance, though performance for $\alpha = 65537$ is roughly similar and the optimal choice is likely implementation-dependent.

For the optimal choice of α, proving takes about 237 ms (a small fraction of the key generation cost, which is 2022 ms) and verifying takes about 713 ms. For comparison, verification of our NIZK is about 8 times faster than the folklore solution discussed in Sect. 1, which requires the verifier to spend 5588 ms to perform the Rabin-Miller primality test on a 2048-bit RSA exponent, and also slows down every public-key operation by a factor of about 60 because e is 2048 bits long (instead of $e = 65537$, which is 17 bits long). We should note that even though our solution is much faster than the folklore one, and adds only 12% to the prover's normal RSA key generation cost, it is still relatively expensive for the verifier: for comparison, the public key operation (encryption or signature verification) with $e = 65537$ takes only about 1.4ms.

From Table 1 we also see that verifying is generally slower than proving (until α gets so big that divisibility testing takes too long for the verifier). This follows because proving involves m_1 modular exponentiations (using RSASP), which can be done separately modulo p and modulo q for $N = pq$ (with the exponent reduced modulo $p-1$ and $q-1$), and then combined using the Chinese Remainder Theorem (CRT). Meanwhile, the verifier does not know p and q, and so cannot use (CRT); moreover, the exponent used for modular exponentiations (using RSAVP) is slightly longer than $\phi(n)$, but the verifier does not know $\phi(N)$ and so cannot reduce it. Thus, exponentiations performed by the verifier are slower than those performed by the prover.

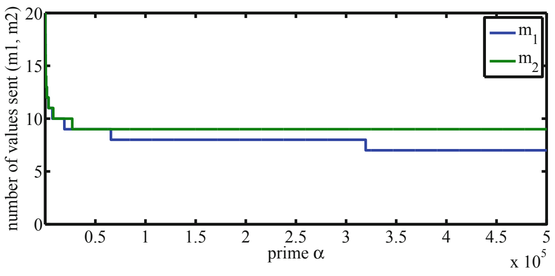

Fig. 1. Values of m_1 and m_2 versus the choice of parameter α for our NIZK, when $\kappa = 128$ and $e = e' = 65537$.

Table 1. Proving and verifying times for our C# implementation as observed on an Azure DS1 v2 virtual machine running Windows Server 2016 Datacenter (single-core 2.4 GHz Intel Xeon E5-2673 v3 Haswell processor, 3.5GiB RAM). Time is given in ms. Public exponent is $e = e' = 65537$ and security parameter is $\kappa = 128$.

Parameters			Permutation Proof	
α	m_1	m_2	Prove	Verify
41	24	24	632	2326
89	20	20	518	1925
191	17	17	443	1612
937	13	13	334	1216
1667	12	12	311	1127
3187	11	12	308	1042
3347	11	11	281	1025
7151	10	11	284	943
8009	10	10	256	948
19121	9	10	254	853
26981	9	9	233	854
65537	8	9	230	768
319567	7	9	237	713
2642257	6	9	234	956
50859013	5	9	230	6756

Acknowledgements. The authors thank Ethan Heilman, Alessandra Scafuro and Yehuda Lindell for useful discussions. This research was supported, in part, by US NSF grants 1717067, 1350733, and 1422965.

A Number-Theoretic Lemmas

We present number-theoretic lemmas that are useful for proving security of our protocols. Some of them are standard and are presented here only to make the presentation self-contained.

Let $\mathbb{Z}_N = \{0, 1, ..., N-1\}$ for any positive integer N and $\mathbb{Z}_N^*$ be the multiplicative group modulo N, i.e., the set of values in $\mathbb{Z}_N$ that are relatively prime to N, or else $\{x \in \mathbb{Z}_N | \gcd(x, N) = 1\}$. We use notation $p|N$ to denote that "p divides N".

Euler's phi or totient function (see, e.g., [Sho09, Section 2.6] for the relevant background) is defined for all positive integers N as:

$$\phi(N) = |\mathbb{Z}_N^*|.$$

If $N = pq$ where p, q are two distinct primes it holds that $\phi(N) = (p-1)(q-1)$. More generally, if the prime factorization of N is $N = p_1^{\alpha_1} \times \cdots \times p_k^{\alpha_k}$, then $\phi(N) = (p_1^{\alpha_1 - 1} \times \cdots \times p_k^{\alpha_k - 1}) \times ((p_1 - 1) \times \cdots \times (p_k - 1))$, with $\phi(1) = 1$ [Sho09, Theorem 2.11]. The following theorem is standard [Sho09, Theorem 2.13]:

Lemma 1 (Euler's theorem). *Let N be a positive integer and $a \in \mathbb{Z}_N^*$. Then $a^{\phi(N)} \bmod N = 1$.*

Given positive integers N and e, consider the map $x \mapsto x^e \bmod N$. We will first consider this map as restricted to $\mathbb{Z}_N^*$. The following lemma is standard.

Lemma 2. *The map* $x \mapsto x^e \bmod N$ *is a permutation of* $\mathbb{Z}_N^*$ *if and only if* $\gcd(e, \phi(N)) = 1$. *If the map is a permutation of* $\mathbb{Z}_N^*$*, then its inverse is the map* $x \mapsto x^d \bmod N$ *for* $d = e^{-1} \bmod \phi(N)$ *(which exists by [Sho09, Theorem 2.5] because* $\gcd(e, \phi(N)) = 1$*).*

Proof. Suppose $\gcd(e, \phi(N)) = 1$. Then let $d = e^{-1} \bmod \phi(N)$. Thus, $de = k\phi(N) + 1$ for some integer k. For every $x \in \mathbb{Z}_N^*$, $(x^e)^d \bmod N = x^{ed} \bmod N = (x^{\phi(N)})^k \cdot x \bmod N = 1^k \cdot x = x$, where the second-to-last equality follows from Lemma 1.

Now suppose $\gcd(e, \phi(N)) = g \neq 1$. Let p be a prime divisor of g. Then $p \mid \phi(N)$, and therefore $\mathbb{Z}_N^*$ contains an element $x \neq 1$ such that $x^p \bmod N = 1$ [Sho09, Theorem 6.42]. Therefore, $x^e \bmod N = (x^p)^{e/p} \bmod N = 1^{e/p} = 1$, and thus the map is not a permutation. □

A number N is *square free* if it can be written as $N = p_1 p_2 \dots p_k$ for *distinct* prime numbers p_i. (N is not square free if it is divisible by p^2, where p is some prime.)

Lemma 3. *For a positive integer* N*, if* $\gcd(N, \phi(N)) = 1$*, then* N *is square-free.*

Proof. Indeed, suppose $p^2 \mid N$ for some prime p. Then $p \mid \phi(N)$, so $\gcd(N, \phi(N)) \geq p > 1$. □

We now extend one direction of Lemma 2 to all of $\mathbb{Z}_N$ for the case of square-free N.

Lemma 4. *If for some positive integers* N *and* f*,* N *is square-free and* $\gcd(f, \phi(N)) = 1$*, then the map* $x \mapsto x^f \bmod N$ *is a permutation on* $\mathbb{Z}_N$*. Its inverse is computed as follows: for* $g = f^{-1} \bmod \phi(N)$ *(which exists by [Sho09, Theorem 2.5]) and for all* $x \in \mathbb{Z}_N$*,* $x^{gf} \bmod N = x$*.*

Proof. Let $N = p_1 p_2 \dots p_k$ for distinct prime numbers p_i. By the Chinese Remainder Theorem (CRT) [Sho09, Theorem 2.8], the ring $\mathbb{Z}_N$ is isomorphic to the product of rings $Z_{p_1} \times \cdots \times \mathbb{Z}_{p_k}$. It therefore suffices to show that $x^{ef} \bmod p_i = x$ for each i. Indeed, $fg = t\phi(N) + 1$ for some integer t, and therefore $x^{fg} = (x^{p_i - 1})^s \cdot x$ for some integer s, and the result follows by Fermat's little theorem [Sho09, Theorem 2.14] when $x \bmod p_i \neq 0$, and trivially when $x \bmod p_i = 0$. □

To extend the other direction of Lemma 2 to all of $\mathbb{Z}_N$ is a little more complicated.

Lemma 5. *If for some positive integers* N *and* f*, the map* $x \mapsto x^f \bmod N$ *is a permutation on* $\mathbb{Z}_N$*, then* $x \mapsto x^f \bmod N$ *is a permutation on* $\mathbb{Z}_N^*$ *(and thus* $\gcd(f, N) = 1$ *by Lemma 2) and either:*

– *N is square-free, or*
– *$f = 1$*

Proof. The first part of the lemma follows from the fact that when raised to the power f modulo N, elements of Z_N^* stay within Z_N^* (because if $\gcd(x, N) = 1$, then $\gcd(x^f \bmod N, N) = 1$). The second part of the lemma is proven as follows. Suppose N is not square-free and $f > 1$. Then let $p^2 \mid N$ for some prime p. The set $\{x \in \mathbb{Z}_N : x \text{ is divisible by } p\}$ contains N/p elements. The image of this set is contained in $\{x \in \mathbb{Z}_N : x \text{ is divisible by } p^2\}$, which contains only N/p^2 elements. Thus, the map is not injective.

The following lemma shows that one can validate if an integer N is square-free by checking if random values in $\mathbb{Z}_N$ have Nth roots. This lemma generalizes the result of Gennaro, Micciancio, and Rabin [GMR98, Section 3.1], which worked over $\mathbb{Z}_N^*$ and thus required a gcd computation every time a random value was selected.

Lemma 6. *Let N be a positive integer and p be a prime such that p^2 divides N (i.e., N is not square free). Then, the fraction of elements of $\mathbb{Z}_N^*$ that have an Nth root modulo N is at most $1/p$, and the fraction of elements of $\mathbb{Z}_N$ that have an Nth root modulo N is also at most $1/p$.*

Proof. Suppose x has an Nth root modulo N. Then there is a value r such that $r^N \equiv x \pmod{N}$. Hence, N divides $r^N - x$, which means p^2 divides $r^N - x$ (since p^2 divides N), and therefore r is the Nth root of x modulo p^2. Thus, in order to have an Nth root modulo N, x must have an Nth root modulo p^2. Since a uniformly random element x of $\mathbb{Z}_N$ is also uniform modulo p^2, and a uniformly random element x of $\mathbb{Z}_N^*$ is also uniform in $Z_{p^2}^*$ when reduced modulo p^2, it suffices to consider what fractions of $Z_{p^2}^*$ and of Z_{p^2} have Nth roots.

By Lemma 8 below, the number of elements of $Z_{p^2}^*$ that have Nth roots is at most $\phi(p^2)/e'$, where e' is the largest prime divisor of $\gcd(N, \phi(p^2)) = \gcd(N, p(p-1))$. Since $p|N$, we have $e' = p$. Thus, the number of elements of $Z_{p^2}^*$ that have Nth roots is at most $\phi(p^2)/p = p - 1$. This shows the first half of the conclusion.

If $x \in Z_{p^2} - Z_{p^2}^*$, then $p|x$. If x has an Nth root r modulo p^2, then $p^2|(r^N - x)$, hence $p|(r^N - x)$, hence $p|r^N$ (because $p|x$ and $p|(r^N - x)$), hence $p|r$ (because p is prime), hence $p^2|r^2$, hence $p^2|r^N$ (because $N > 1$), and hence $p^2|x$ (because $p^2|(r^N - x)$ and $p^2|r^N$). We therefore have that $x \in Z_{p^2}$ and $p^2|x$, which means that $x = 0$.

Thus, the total number of elements of Z_{p^2} that have an Nth root is at most $p - 1$ elements from $Z_{p^2}^*$ and one element from $Z_{p^2} - Z_{p^2}^*$ (namely, the element $x = 0$), for a total of at most p elements from Z_{p^2}. Thus, at most a $p/|Z_{p^2}| = 1/p$ fraction of elements of Z_{p^2} have Nth roots. It follows that at most a $1/p$ fraction of elements of $\mathbb{Z}_N$ has Nth roots. □

The following lemma shows that if we know that N is square free (which we can test using Lemma 6), then we can check whether raising to the power e is a permutation of $\mathbb{Z}_N$, by checking if random values in $\mathbb{Z}_N$ have eth roots.

Lemma 7. *Suppose $N > 0$ is a square-free integer so that $N = p_1 p_2 \dots p_k$ for distinct prime numbers p_i, and $e > 0$ is an integer. If raising to the power e modulo N is not a permutation over $\mathbb{Z}_N$, then the fraction of elements of $\mathbb{Z}_N$ that have a root of degree e is at most*

$$\frac{1}{p} + \frac{1}{e'}\left(1 - \frac{1}{p}\right),$$

where e' is the smallest prime divisor of e and p is the smallest prime divisor of N (these are well-defined, because if $N = 1$ or $e = 1$, then raising to the eth power is a permutation over $\mathbb{Z}_N$).

Proof. By Chinese Remainder Theorem (CRT) [Sho09, Theorem 2.8], the ring $\mathbb{Z}_N$ is isomorphic to the product of rings $Z_{p_1} \times \dots \times Z_{p_k}$. Note that if raising to the power e modulo N is not a permutation over $\mathbb{Z}_N$, then there exist $x \not\equiv y \pmod{N}$ such that $x^e \equiv y^e \pmod{N}$. Let i be such that $x \not\equiv y \pmod{p_i}$ (it must exist by CRT); then raising to the power e modulo p_i is not a permutation of Z_{p_i}, because $x^e \equiv y^e \pmod{p_i}$ (by CRT).

Since a uniformly random element x of $\mathbb{Z}_N$ is uniform modulo p_i, it suffices to consider what fraction of Z_{p_i} has eth roots. By Lemma 8 below, the number of elements of $Z^*_{p_i}$ that have eth roots is at most $\phi(Z^*_{p_i})/e' = (p_i - 1)/e'$. The only element in $Z_{p_i} - Z^*_{p_i}$ is the element 0. So, in total, at most $(p_i - 1)/e' + 1$ elements of Z_{p_i} have eth roots. Since $p_i \geq p$,

$$\frac{(p_i - 1)/e' + 1}{p_i} = \frac{1}{e'} + \frac{1}{p_i}\left(1 - \frac{1}{e'}\right) \leq \frac{1}{e'} + \frac{1}{p}\left(1 - \frac{1}{e'}\right) = \frac{1}{p} + \frac{1}{e'}\left(1 - \frac{1}{p}\right).$$

□

The proofs of two lemmas above relied on the lemma below.

Lemma 8. *For any positive integers N and e, if raising to the power e modulo N is not a permutation over $\mathbb{Z}^*_N$, then $\gcd(e, \phi(N)) > 1$ and the number of elements of $\mathbb{Z}^*_N$ that have a root of degree e is at most $\phi(N)/e'$, where e' is the largest prime divisor of $\gcd(e, \phi(N))$.*

Proof. Suppose there exist x and y in $\mathbb{Z}^*_N$ such that $x^e \equiv y^e \pmod{N}$ but $x \not\equiv y \pmod{N}$. Then $x/y \not\equiv 1 \pmod{N}$ but $(x/y)^e \equiv 1 \pmod{N}$. Therefore, the multiplicative order of (x/y) is greater than 1 and divides e [Sho09, Theorem 2.12] and $\phi(N)$ [Sho09, Theorem 2.13], which implies that $\gcd(e, \phi(N)) > 1$. Let e' be the largest prime divisor of $\gcd(e, \phi(N))$.

Because e' is a prime that divides $\phi(N)$, $\mathbb{Z}^*_N$ contains an element z of order e' [Sho09, Theorem 6.42]. Therefore, the homomorphism that takes each element of $\mathbb{Z}^*_N$ to the power e has kernel of size at least e' (because this kernel contains distinct values $z, z^2, \dots, z^{e'}$ which are all eth roots of 1 because e' divides e). The image of this homomorphism contains exactly the elements that have roots of degree e, and the size of this image is equal to $\phi(N)$ divided by the size of the kernel [Sho09, Theorem 6.23], i.e., at most $\phi(N)/e'$. □

B Background on the Fiat-Shamir transform

Any efficient, interactive constant-round, public-coin, honest-verifier zero knowledge (HVZK) proof system can be converted into a noninteractive ZK argument[3] (NIZK) through the so called Fiat-Shamir (FS) transformation [FS86]. Applying FS allows us to replace the verifier V by instead calling a hash function on input the current transcript. The security of the resulting scheme holds in the random oracle [BR93] (RO), where a hash function H is evaluated through calls to an oracle that acts as a random function. The main idea in the security proof is that the simulator for HVZK can "program" the RO (i.e., the simulator decides the answer to each specific query). This allows the simulator to convert the entire transcript of a public-coin HVZK proof into a single message that is indistinguishable from the message computed by an honest NIZK prover. We first recall the definition of NIZKs in the RO and then state the Fiat-Shamir transformation theorem (definitions slightly modified from [FKMV12]).

Let S be a simulator that operates in two modes: $(h_i, st) \leftarrow \mathrm{S}(1, st, q_i)$ which on input a random oracle query q_i it responds with h_i (usually by lazy sampling), and $(\pi, st) \leftarrow \mathrm{S}(2, st, x)$ which simulates simulates the actual proof. (Note that calls to $\mathrm{S}(1, \cdots)$ and $\mathrm{S}(2, \cdots)$ share the common state st that is updated after each operation).

Definition 2 (NIZK). *Let* $(\mathrm{S}_1, \mathrm{S}_2)$ *be oracles such that* $S_1(q_i)$ *returns the first output of* $(h_i, st) \leftarrow \mathrm{S}(1, st, q_i)$ *and* $S_2(x, w)$ *returns the first output of* $(\pi, st) \leftarrow \mathrm{S}(2, st, x)$ *if* $(x, w) \in RL$.

A protocol $\langle \mathrm{P}^H, \mathrm{V}^H \rangle$ *is said to be a NIZK proof for language* L *in the random oracle model, if there exists a PPT simulator* S *such that for all PPT distinguishers D we have*

$$\mathtt{view}D^{H(\cdot),\mathrm{P}^H(\cdot,\cdot)} \approx \mathtt{view}D^{\mathrm{S}_1(\cdot),\mathrm{S}_2^H(\cdot,\cdot)}.$$

We now state and prove the following theorem for the Fiat-Shamir transformation [FKMV12]:

Theorem 8 (Fiat-Shamir NIZK). *Let κ be a security parameter. Consider a non-trivial constant round, public-coin, honest-verifier zero-knowledge (HVZK) interactive proof system* $\langle \mathrm{P}, \mathrm{V} \rangle$ *for a language* L. *Let $H()$ be a function with range equal to the space of the verifier's coins. In the random oracle model the proof system* $\langle \mathrm{P}^H, \mathrm{V}^H \rangle$, *derived from* $\langle \mathrm{P}, \mathrm{V} \rangle$ *by applying the Fiat-Shamir transform, is a noninteractive ZK argument.*

Proof. (sketch) All we need to show is that there exists a simulator S as required in Definition 2. This can be done by invoking the HVZK simulator associated with the underlying interactive proof system.

[3] As opposed to a proof system where soundness needs to hold unconditionally, in an argument system it is sufficient that soundness holds with respect to a computationally bounded adversary P^*.

We design S to work as follows:

- To answer to a query q to S_1, $\mathrm{S}(1, st, q)$ lazily samples a lookup table kept in state st. It checks whether an answer for q was already defined. If this is the case, it returns the previously assigned value; otherwise it returns a fresh random value h and stores the pair (q, h) in the table.
- To answer to a query x to S_2, $\mathrm{S}(2, st, x)$ calls the HVZK simulator of $\langle \mathrm{P}, \mathrm{V} \rangle$ on input x to obtain a proof π. Then, it updates the look up table by storing x, π. If the look up table happens to be already defined on this input, S returns failure and aborts.

Given that the protocol is non-trivial, the probability of failure in each of the queries to S_2 is negligible. □

C Detailed Specification for the NIZK of Permutations over $\mathbb{Z}_n$

The following specification is for the NIZK of Permutations over $\mathbb{Z}_n$, as described in Sect. 5. This specification assumes that the RSA exponent e is prime.

C.1 System Parameters

The system parameters are the RSA modulus length `len`, the security parameter κ (where by default $\kappa = 128$), a small prime α (about 16 bits long or less), and a publicly-known octet string `salt`.

C.2 Proving

System Parameters:

1. `salt` (an octet string),
2. α (a prime number)
3. κ (the security parameter, use 128 by default)
4. e, the fixed prime RSA exponent
5. `len`, the RSA key length.

Auxiliary Function: getRho, defined in Sect. C.4.

Input: Distinct equal-length primes p and q greater than α such that the RSA modulus is $N = pq$ is of length `len`, and e does not divide $(p-1)(q-1)$.

Output: $(N, e), \{\sigma_1, ..., \sigma_{m_2}\}$.

Algorithm:

1. Set m_1 and m_2 as in Eq. 1, Sect. 3.3, with $e' = e$.
2. Set $N = pq$.
3. Obtain the RSA secret key K as specified by [MKJR16, Sec. 3.2]:

$$K = (p, q, d_{NP}, d_{NQ}, q_{Inv})$$

4. Compute the "weird RSA" secret key corresponding to public key (N, eN) (with exponent eN and modulus N) in the [MKJR16, Sec. 3.2] as

$$K' = (p, q, d_{NP}, d_{NQ}, q_{Inv})$$

   where p, q, q_{Inv} are the same as in the normal RSA secret key K and

$$d_{NP} = (eN)^{-1} \bmod (p-1) \qquad d_{NP} = (eN)^{-1} \bmod (q-1) \tag{2}$$

5. For integer $i = 1 \ldots m_2$
   (a) Sample ρ_i, a random element of Z_N, as

$$\rho_i = \text{getRho}((N, e), \texttt{salt}, i, \texttt{len}, m_2)$$

   (b) If $i \le m_1$, let
$$\sigma_i = \text{RSASP1}(K', \rho_i)$$
   where RSASP1 is the RSA signature primitive of [MKJR16, Sec. 5.2.1]. In other words, σ is the RSA decryption of ρ_i using the "weird RSA" secret key K'.
   (It follows that σ_i is (eN)th root of ρ_i.)
   (c) Else let
$$\sigma_i = \text{RSASP1}(K, \rho_i)$$
   where RSASP1 is the RSA signature primitive of [MKJR16, Sec. 5.2.1]. In other words, σ is the RSA decryption of ρ_i using the regular RSA secret key K.
   (It follows that σ_i is eth root of ρ_i.)
6. Output $(N, e), \{\sigma_1, ..., \sigma_{m_2}\}$.

C.3 Verifying

System Parameters:

1. `salt` (an octet string),
2. α (a prime number)
3. κ (the security parameter, use 128 by default)
4. e, the fixed prime RSA exponent
5. `len`, the RSA key length

Auxiliary Function: getRho, defined in Sect. C.4.

Input: RSA public key (N, e) and $\{\sigma_1, ..., \sigma_{m_2}\}$.

Output: VALID or INVALID

Algorithm:

1. Check that N is an integer and $N \geq 2^{\texttt{len}-1}$ and $N < 2^{\texttt{len}}$. If not, output INVALID and stop.
2. Check that e is prime. If not, output INVALID and stop.
3. Compute m_1 and m_2 per Eq. (1), Sect. 3.3, with $e' = e$.
4. Check that there are exactly m_2 values $\{\sigma_1, ..., \sigma_{m_2}\}$ in the input. If not, output INVALID and stop.
5. Generate the vector Primes$(\alpha - 1)$, which includes all primes up to and including $\alpha - 1$. (This can be efficiently implemented using the Sieve of Eratosthenes when α is small.)
   For each $p \in$ Primes$(\alpha - 1)$:
   - Check that N is not divisible by p. If not, output INVALID and stop.
     (Alternatively, let `primorial` be the product of all values in Primes$(\alpha - 1)$. `primorial` should be computed once and should be a system parameter. Check that gcd$(\texttt{primorial}, N) = 1$.)
6. For integer $i = 1 \ldots m_2$
   (a) Sample ρ_i, a random element of Z_N, as
   $$\rho_i = \text{getRho}((N, e), \texttt{salt}, i, \texttt{len}, m_2)$$
   (b) If $i \leq m_1$, check that
   $$\rho_i = \text{RSAVP1}((N, eN), \sigma_i)$$
   where RSAVP1 is the RSA verification primitive of [MKJR16, Sec. 5.2.2]. In other words, check that ρ_i is the RSA encryption of σ_i using the "weird RSA" public key (N, eN). If not, output INVALID and stop.
   (Thus, check that $\rho_i = \sigma_i^{eN} \bmod N$).
   (c) Else check that
   $$\rho_i = \text{RSAVP1}(PK, \sigma_i)$$
   In other words, check that the ρ_i is the RSA encryption of σ_i using the RSA public key (N, e). If not, output INVALID and stop.
   (Thus, check that $\rho_i = \sigma_i^{e} \bmod N$).
7. Output VALID.

C.4 Auxiliary function: getRho

This function is for rejection sampling of a pseudorandom element $\rho_i \in Z_N$. It is "deterministic," always producing the same output for a given input.

Input:

1. RSA public key (N, e).
2. `salt` (an octet string)
3. Index integer i.
4. Length of RSA modulus `len`
5. Value m_2, with $i \le m_2$.

Output: ρ_i

Algorithm:

```
1. Let
       $$|m_2| = \lceil \frac{1}{8}(\log_2(m_2 + 1)) \rceil$$
   be the length of $m_2$ in octets. (Note: This is an octet length, not a bit length!)
2. Let $j = 1$.
3. While true:
   (a) Let $PK$ be the ASN.1 octet string encoding of the RSA public key $(N, e)$
       as specified in [MKJR16, Appendix A].
   (b) Let $EI$ = I2OSP$(i, |m_2|)$ be the $|m_2|$-octet long string encoding of the
       integer $i$. (The I2OSP primitive is specified in [MKJR16, Sec. 4.2].)
   (c) Let $EJ$ = I2OSP$(j, |j|)$ be the $|j|$-octet long string encoding of the integer
       $j$, where $|j| = \lceil \frac{1}{8} \log_2(j + 1) \rceil$.
   (d) Let $s = PK||$salt$||EI||EJ$ be the concatenation of these octet strings.
   (e) Let $ER$ = MGF1-SHA256$(s, $len$)$ where $H_1$ is the MGF1 Mask Generation
       Function based on the SHA-256 hash function as defined in
       [MKJR16, Sec. B.2.1], outputting values that are len bits long.
   (f) Let $\rho_i$ = OS2IP$(ER)$ be an integer.
       (That is, convert $ER$ to an len bit integer $\rho_i$ using the OS2IP primitive
       specified in [MKJR16, Sec. 4.1].)
   (g) If $\rho_i \ge N$, then let $j = j + 1$ and continue; Else, break.
       (Note: This step tests if $\rho_i \in Z_N$.)
4. Output integer $\rho_i$.
```

References

[AP18] Auerbach, B., Poettering, B.: Hashing solutions instead of generating problems: on the interactive certification of RSA moduli. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10770, pp. 403–430. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76581-5_14

[Ber98] Bernstein, D.J.: Detecting perfect powers in essentially linear time. Math. Comput. 67, 1253–1283 (1998)

[BFGN17] Benhamouda, F., Ferradi, H., Géraud, R., Naccache, D.: Non-interactive provably secure attestations for arbitrary RSA prime generation algorithms. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10492, pp. 206–223. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66402-6_13

[BFL89] Boyar, J., Friedl, K., Lund, C.: Practical zero-knowledge proofs: giving hints and using deficiencies. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 155–172. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-46885-4_18

[BLP07] Bernstein, D.J., Lenstra, H.W., Pila, J.: Detecting perfect powers by factoring into coprimes. Math. Comput. **76**(257), 385–388 (2007)

[bou] bouncycastle c# api. https://www.bouncycastle.org/csharp/index.html

[BR93] Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, pp. 62–73. ACM (1993)

[BY96] Bellare, M., Yung, M.: Certifying permutations: noninteractive zero-knowledge based on any trapdoor permutation. J. Cryptol. 9(3), 149–166 (1996). https://cseweb.ucsd.edu/~mihir/papers/cct.html

[CM99] Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 107–122. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_8

[CMS99] Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_28

[cod] Tumblebit setup implementation. https://github.com/osagga/TumbleBitSetup

[CPP07] Catalano, D., Pointcheval, D. and Pornin, T. Trapdoor hard-to-invert group isomorphisms and their application to password-based authentication. J. Cryptol. **20**(1), 115–149, 2007. http://www.di.ens.fr/~pointche/Documents/Papers/2006_joc.pdf

[DJ01] Damgård, I., Jurik, M.: A Generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_9

[FKMV12] Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_5

[FS86] Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12

[GM84] Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)

[GMR98] Gennaro, R., Micciancio, D., Rabin, T.: An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In: Gong, L., Reiter, M.K. (eds.) CCS 19, Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, CA, USA, 3–5 November 1998, pp. 67–72. ACM (1998). http://eprint.iacr.org/1998/008

[HAB+17] Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A. and Goldberg, S.: Tumblebit: an untrusted bitcoin-compatible anonymous payment hub. In: 24th Annual Network and Distributed System Security Symposium, NDSS. The Internet Society (2017). https://eprint.iacr.org/2016/575.pdf

[HMRT12] Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T.: Efficient RSA key generation and threshold paillier in the two-party setting. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 313–331. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27954-6_20

[Hoe63] Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)

[KKM12] Kakvi, S.A., Kiltz, E., May, A.: Certifying RSA. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 404–414. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_25

[Lin17] Lindell, Y.: Fast secure two-party ECDSA signing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 613–644. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_21

[LMRS04] Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_5

[MKJR16] Moriarty, K., Kaliski, B., Jonsson, J., Rusch, A.: RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2. Internet Engineering Task Force (IETF) (2016). https://tools.ietf.org/html/rfc8017

[MPS00] MacKenzie, P., Patel, S., Swaminathan, R.: Password-authenticated key exchange based on RSA. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 599–613. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_46

[MRV99] Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science, FOCS 1999, 17–18 October 1999, New York, NY, USA, pp. 120–130. IEEE Computer Society (1999)

[Ntu] Tumblebit implementation in.net core. https://github.com/NTumbleBit/NTumbleBit/

[Sho09] Shoup, V.: A Computational Introduction to Number Theory and Algebra, 2nd edn. Cambridge University Press (2009). http://www.shoup.net/ntb/ntb-v2.pdf

[Str17] Stratis Blockchain: Bitcoin privacy is a breeze: tumblebit successfully integrated into breeze, August 2017. https://stratisplatform.com/2017/08/10/bitcoin-privacy-tumblebit-integrated-into-breeze/

[WCZ03] Wong, D.S., Chan, A.H., Zhu, F.: More efficient password authenticated key exchange based on RSA. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 375–387. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-24582-7_28