The TumbleBit Setup Protocol

First draft of specification from July 31, 2017.

1 Introduction

TumbleBit [HAB+17] is a unidirectional unlinkable payment hub that allows parties to make fast, anonymous, off-blockchain payments through an untrusted intermediary called the Tumbler. TumbleBit's anonymity properties ensure that no one, not even the Tumbler, can link a payment from its payer to its payee. Every payment made via TumbleBit is backed by bitcoins, and comes with a guarantee that the Tumbler can neither violate anonymity, nor steal bitcoins, nor "print money" by issuing payments to itself. This document specifies the TumbleBit setup protocol which is run by the Tumbler itself. The goal of this protocol is to ensure the validity of the setup parameters so that there is no need to place any trust to 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 Z_N. In the absence of this assumption, the Tumbler can steal bitcoins.[1] At the same time, the cryptographic proofs of security for the TumbleBit protocol (which are in the real/ideal paradigm) rest on the assumption that payers and payees (Alice and Bob) verify a publicly-verifiable zero-knowledge proof of knowledge that the Tumbler knows the secret key corresponding to his RSA public key.

Because the Tumbler is charged with choosing its own RSA public key, it is important to ensure that the Tumbler did not choose an "adversarial" key that allows it to steal bitcoins. As such, the TumbleBit protocol has a setup phase that forces the Tumbler to prove that his key was chosen "properly". Specifically, the Tumbler must post his RSA public key[2] (N, e) and its configuration parameters, along with a pair of zero-knowledge proofs to the blockchain.[3] The first is a new zero-knowledge proof protocol that we designed. This protocol proves that the RSA public key (N, e) defines a permutation over Z_N. We design the protocol, prove its security, and provide a full specification. The second zero-knowledge proof protocol is by Poupard and Stern [PS00], and proves knowledge of the factorization of the RSA modulus N. For the Poupard-Stern protocol, we set parameters and provide a full specification. When any payer Alice or payee Bob wants to participate in the TumbleBit protocol with this Tumbler, they must first verify this pair of zero knowledge proofs. This document specifies the protocols for this pair of zero-knowledge proofs.

[1] Specifically, the Tumbler can provide Bob with a puzzle z that has two valid solutions $\epsilon_1 \neq \epsilon_2$ where $z = (\epsilon_1)^e = (\epsilon_2)^e \mod N$, where ϵ_1 allows Bob to decrypt the Tumbler's signature on the transaction that allows Bob to claim his bitcoin, but ϵ_2 does not. Then, to steal a bitcoin, the Tumbler gives Alice the solution ϵ_2, so that the Tumbler can claim Alice's bitcoin (because the puzzle-solver protocol will complete correctly, since $\epsilon_2^e = z$) without passing it on to Bob (because ϵ_2 cannot be used to decrypt the Tumbler's signature on the transaction that allows Bob to claim his bitcoin).

[2] Posting the Tumbler's public key to the blockchain prevents the Tumbler from presenting one key to some users, and a different key to other users, thus reducing the size of the anonymity set in a TumbleBit epoch.

[3] In practice, we can limit the amount of information that must actually be posted to the blockchain by having the Tumbler post only the hash of these values to the blockchain. The values themselves can be served to the users when they first contact the Tumbler.

2 Zero-Knowledge Proof that RSA is a Permutation over Z_N

Many applications use an RSA public key (N, e) that is chosen by an (potentially) adversarial party. In those applications, it is crucial that parties can verify that RSA public key was chosen "properly". While the definition of "properly" depends on the application, a common requirement for the many applications that use RSA solely as the instantiation of a trapdoor permutation, is to show that the RSA public key defines a permutation over Z_N. That is, for that every value ρ, where $\rho = 0, 1, 2, ..., N-1$, there is only one *unique* value σ that is its eth root modulo N, *i.e.,*

$$\rho = \sigma^e \pmod{N}. \tag{1}$$

We present the design and implementation of a simple zero-knowledge proof that allows any party holding an RSA public key (N, e) to verify that it defines a permutation over Z_N, without leaking information about the corresponding RSA private key. Our protocol is compliant with existing cryptographic specifications of RSA (*e.g.,* RFC8017 [MKJR16]) and can be used even when the RSA exponent e is small (which is useful for applications that require fast RSA encryption/verification). We designed this protocol for setup phase of the TumbleBit protocol for fast blockchain-backed private payments [HAB+17]. However, our protocol's simplicity and compliance with existing cryptographic standards and libraries may make it of independent interest. Indeed, are a variety of other protocols that rely on the assumption that an (adversarially-chosen) RSA public key (N, e) defines a permutation, see for instance [KKM12, Section 1.2].

2.1 Design decisions.

The design of our protocol was guided by the following considerations.

Small e. First, we want the protocol to work when the RSA encryption exponent e is small. Small e is vital for performance, because it ensures that RSA verification (see equation (1)) can be done quickly. Indeed, the TumbleBit protocol requires a large number of RSA verifications each time a payment is made, and therefore uses a small RSA exponent of $e = 65537$. As such, we cannot just use established approaches that choose large prime $e > N$ to ensure that RSA is a permutation. (See *e.g.,* the SNAPI protocol of [MPS00], or [CMS99] or [LMRS04].) 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 use reasonable complexity assumptions to show that when e is small, it is impossible to do verify that RSA is a permutation by providing only the values (N, e) and no additional information [KKM12, Section 1]. We circumvent this impossibility by having the Prover provide additional information (beyond just (N, e)) to the Verifier.

Standards compliance. Second, we want our protocol to work over all of Z_N, rather than just over Z_N^*. (Recall that Z_N is the set of integers $0 < \rho < N$, while Z_N^* is the set of integers $0 < \rho < N$ such that $\gcd(\rho, N) = 1$.) In other words, we want our protocol to prove that RSA is a permutation over all values in Z_N, including those values ρ that have $\gcd(\rho, N) > 1$ so that $\rho \in Z_N - Z_N^*$. This requirement is especially important because of standards that specify RSA signature and verification as performed on any integer $\rho \in Z_N$, not just on integers $\rho \in Z_N^*$. (See, for instance, the RSA PKCS #1, Version 2.2 specification in RFC8017 [MKJR16], Section 5.) For this reason, typical implementations of RSA do not first check that the input is in Z_N^* before performing an RSA operation; instead, they will perform cryptographic operations over any value in Z_N. (This is the case for the BouncyCastle cryptographic library used in TumbleBit [**?**].)

While our protocol works over all of Z_N, [WCZ03, Section 3.2] and [CPP07, Appendix D.2] present a protocol (similar to ours) that only works over Z_N^*.[4] Because the [WCZ03, Section 3.2] and [CPP07, Appendix D.2] protocols do not prove that RSA is a permutation over values in $Z_N - Z_N^*$, they should only be used in applications that additionally compute $\gcd(\rho, N) = 1$ each time an RSA verification is performed on input ρ. Apart from adding complexity and potentially harming performance, this is a deviation from cryptographic standards and thus requires modifications to cryptographic libraries, which is a non-trivial task for many developers and practitioners. By contrast, our protocol is carefully designed to prove that RSA is a permutation over all of Z_N, rather than just Z_N^*, so that it is compatible with existing cryptographic standards and libraries, and the applications that use them.

Simplicity. Third, we want our protocol to be simple and easy to implement. The protocol of [CM99, Section 5.2] proves that N is a product of two safe primes, which can be used to check easily that (N, e) specifies a permutation over all of Z_N even when e is small. However, this protocol is more powerful than what we need, and considerably less efficient. In particular, [CM99, Section 5.2] uses the entire protocol of Gennaro *et al.* [GMR98] as a subroutine; we will only use one part of [GMR98].

Performance. We intend this protocol to be used in applications (like TumbleBit) where users verify the RSA key only once during a setup phase, but the application itself requires many RSA operations to be performed with the RSA key. For this reason, the one-time computational cost of verifying our zero-knowledge proof can be amortized over all the RSA operations that are subsequently performed. Also, we have not focused on making the proof especially short. (This is because we have very few constraints on the length of the zero-knowledge proof for the TumbleBit setup protocol.[5])

That said, for 2048-bit RSA keys and a security parameter of $\kappa = 128$ the computational cost of our protocol amounts to about ten RSA signing operations for both prover and verifier, and the length of the proof corresponds to about ten RSA values.

2.2 Overview of our protocol

The starting point of our protocol is the observation that if the RSA modulus is *square free* and values in Z_N^* have eth roots modulo N, then it follows that all the values in $Z_N \backslash Z_N^*$ all have unique e^{th} roots modulo N. Thus, our protocol simultaneously tests that N is square free, and that RSA is permutation over Z_N.

Square-freeness. Recall that 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 will not be square free if it is divisible by p^2, where p is some prime.) It is important to note that RSA will not be a permutation over Z_N if N is not square free. Specifically, if N is not square free, there will be values in $Z_N \backslash Z_N^*$ that do no have unique eth roots modulo N.

For instance, suppose that N is not square free because $N = p^3 q$ for a small p (*e.g.,* around 1000) and let $e = 3$ so that $\gcd(e, \phi(N)) = 1$ (i.e. $e = 3$ does not divide $p - 1$ or $q - 1$). Then raising to the power of $e = 3$ is a permutation of Z_N^*, but not of Z_N. Why? Any $x \in Z_N \backslash Z_N^*$ that

[4] Indeed, using the [WCZ03, Section 3.2] or [CPP07, Appendix D.2] protocols with TumbleBit would require modifications to the TumbleBit protocol. Each time the TumbleBit protocol required an RSA verification of value ρ (per equation (1)), it would additionally need to check that that $\rho \in Z_N^*$ by checking that $\gcd(N, \rho) = 1$.

[5] For the TumbleBit setup, we need only post a *hash* of the proof the blockchain, and users can easily download the proof itself directly from the TumbleBit Tumbler when they become first become its clients.

is divisible by p will also become divisible by p^3 after this operation (*i.e.,* raising to the power of $e = 3$). Thus, such an x will *not* have a unique eth root modulo N which means that (N, e) is not a permutation. In fact, it's pretty far from a permutation: about $1/p$ fraction of Z_N will have no inverses, and $1/p^3$ fraction of Z_N will have three inverses.

To prove the square-freeness of N, we adapt the protocol of Gennaro, Micciancio, and Rabin [GMR98, Section 3.1] which shows that random elements of Z_N^* have Nth roots modulo N. We modify their protocol, so that it works over all of Z_N, rather than just Z_N^*. This way, we avoid requiring a gcd computation each time a random element is chosen to ensure the random element is in Z_N^*, rather than Z_N (see Lemma 2.1); this allows allows us to easily combine this protocol with the protocol that "certifies" the permutation. Our modified protocol simply shows that random elements modulo N have Nth roots, as follows. First, the Verifier chooses random elements ρ_i of Z_N. The Prover responds by computes their Nth roots $\sigma_i = (\rho_i)^{1/N} \pmod N$. Finally, the Verifier accepts if $\rho_i = (\sigma_i)^N \pmod N$.

Certifying permutations. Next, we adapt the protocol of Bellare and Yung [BY96], which showed how to certify that any function is close to a permutation. We observe that for RSA with a square-free modulus N, raising to eth power is either a permutation or very far from one (see Lemma 2.2). Thus, the protocol from [BY96] can be used to certify that (N, e) specifies a permutation. Here, we simply show that random elements modulo N have eth roots.

Combining both protocols. 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 that random values modulo N have eNth roots. Specifically, the prover (who holds the RSA secret key) sends the Verifier $\sigma_1 ... \sigma_m$, where each σ_i is eNth root of a random values $\rho_i \in Z_N$ selected by the Verifier. The Verifier validates that each $\sigma_i^{(eN)} = \rho_i \pmod N$. Finally, we make the protocol non-interactive in the random oracle model by using the Fiat-Shamir [FS86] paradigm.

2.3 High-Level Protocol

Basic Protocol. To verify whether (N, e) defines a permutation with soundness error $2^{-\kappa}$, we proceed so as follows. We assume $e > 1$ and the smallest prime divisor e' of e is known (in most common RSA implementations, e is a fixed prime, anyway).

1. Start by choosing the system parameters: the security parameter is κ, and a prime number α. Let

$$m_1 = \left\lceil -(\kappa+1)/\log_2 \frac{1}{\alpha} \right\rceil \quad \text{and} \quad m_2 = \left\lceil -(\kappa+1)/\log_2 \left(\frac{1}{\alpha} + \frac{1}{e'} \left(1 - \frac{1}{\alpha} \right) \right) \right\rceil . \qquad (2)$$

   Notice that $m_2 > m_1$ since $e' > 1$. Regarding the choice of α: Any prime α works. Higher α will require more work for the Verifier in Step 0a, but less work all other steps (because a large α implies a smaller m_1 and m_2).

2. The Verifier chooses m_1 random values $\rho_i \in Z_N$ and m_2 random values $\rho_j \in Z_N$.

3. The Prover sends back $\sigma_i = (\rho_i)^{1/N} \pmod N$ for $i = 1 \ldots m_1$ and $\sigma_j = (\rho_j)^{1/e} \pmod N$ for $j = 1 \ldots m_2$.

4. The Verifier accepts that (N, e) defines a permutation if:

(a) Check that 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 for square-freeness of N with soundness error $2^{-(\kappa+1)}$, as follows. Verify that $\rho_i = (\sigma_i)^N \pmod N$ for $i = 1 \ldots m_1$. Square-freeness follows from Lemma 2.1.

(c) Check that (N, e) define a permutation assuming square-freeness of N, with soundness error $2^{-(\kappa+1)}$, as follows. Verify that $\rho_j = (\sigma_j)^e \pmod N$ for $j = 1 \ldots m_2$. Permutation follows from square-freeness and Lemma 2.2.

Improved Protocol. To improve performance, we observe that any value that has an (eN)th root also has an eth root and an Nth root, obtained by raising the (eN)th root to the power N or e, respectively). Thus, instead of requiring the protocol to test $m_1 + m_2$ distinct values, we can instead test for just m_2 values (because $m_1 \leq m_2$). Specifically, we modify Steps 2-4 of the basic protocol above as follows:

2. The Verifier chooses m_2 random values $\rho_i \in Z_N$.

3. The Prover sends back $\sigma_i = (\rho_i)^{1/(eN)} \pmod N$ for $i = 1 \ldots m_1$ (for convenience, we call σ_i a "weird RSA signature") and $\sigma_i = (\rho_i)^{1/e} \pmod N$ for $i = m_1 + 1 \ldots m_2$ (notice that σ_i is regular RSA signature).

4. The Verifier accepts that (N, e) defines a permutation if:

   (a) Check that N is not divisible by all the primes less than α, as before. (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 an eNth root exists for the first m_1 values, as follows. Verify that $\rho_i = (\sigma_i)^{eN} \pmod N$ for $i = 1 \ldots m_1$. (This is a "weird RSA verification".)

   (c) Check that an eth root exists for the remaining $m_2 - m_1$ values as follows. Verify that $\rho_i = (\sigma_i)^e \pmod N$ for $i = m_1 + 1 \ldots m_2$. (This is a regular RSA verification.)

(Note: for many natural choices of parameters (e, κ, α), it follows that $m_1 = m_2$ so the the latter step is not necessary.)

Non-interactive protocol. To make the protocol non-interactive in the random oracle model, we use the Fiat-Shamir paradigm [FS86] as follows. Instead of having the Verifier select the random values $\rho_i \in Z_N$ in Step 2, the Prover samples $\rho_i \in Z_N$ by himself by computing the output of the random oracle over the concatenation of (1) the RSA public key (N, e) (2) a public string given as a system parameter `public string`, and (3) the index i. Thus, a given pair of RSA key (N, e) and `public string` determines a deterministic set of $\rho_i \in Z_N$. The Verifier can therefore compute $\rho_i \in Z_N$ on his own, by following the same procedure as the prover, and subsequently verification proceeds as in Step 4. The remainder of this paper provides a detailed specification of the non-interactive improved protocol, proves its security, and evaluates its performance.

2.4 Security Proofs

2.4.1 Completeness

If N is a proper RSA modulus $N = pq$ where p and q are large distinct primes, then the prover will always succeed.

2.4.2 Soundness

Let $\phi(N)$ denote $|Z_N^*|$. The notation $p|N$ means "p divides N".

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

Lemma 2.1. *Let $N > 1$ be an integer and p be a prime such that p^2 divides N. N is not square free. Then, the fraction of elements of Z_N that have an Nth root modulo N is 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 Z_N is also uniform modulo p^2, it suffices to consider what fraction of Z_{p^2} has Nth roots.

By Claim 2.3 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 $\phi(p^2)/p = p - 1$.

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 Z_N has Nth roots. □

The following lemma shows that if we know N is square free (which we can test using Lemma 2.1), then we can check whether raising to the power e is a permutation of Z_N, by checking if random values in Z_N have eth roots.

Lemma 2.2. *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 Z_N, then the fraction of elements of 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 Z_N).

Proof. By Chinese Remainder Theorem (CRT) [Sho09, Theorem 2.8], the ring 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 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 an permutation of Z_{p_i}, because $x^e \equiv y^e \pmod{p_i}$ (by CRT).

Since a uniformly random element x of Z_N is uniform modulo p_i, it suffices to consider what fraction of Z_{p_i} has eth roots. By Claim 2.3 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 both lemmas above relied on the claim below.

Claim 2.3. *For any integers $N > 1$ and $e > 1$, if raising to the power e modulo N is not a permutation over Z_N^*, then $\gcd(e, \phi(N)) > 1$ and the number elements of 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 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 $\phi(N)$, Z_N^* contains an element z of order e' [Sho09, Theorem 6.42]. Therefore, the homomorphism that takes each element of Z_N^* to the power e has kernel of size at least e' (because this kernel contains distinct values $z, z^2, \ldots, z^{e'}$ 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 $\phi(N)/e'$. □

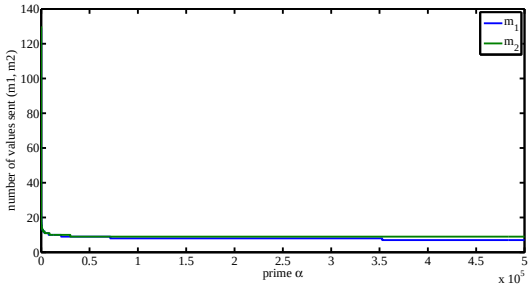
2.4.3 Zero-Knowledge

Roots of random values are the same as random values raised to the appropriate power, whenever you have a square free permutation.

2.5 Parameters and Performance for TumbleBit

For the TumbleBit Setup Protocol, $\kappa = 128$, $|N| = 2048$ and `public string` is the SHA256 hash of the blockchain's Genesis block. Also, $e' = 65537$, so that $m_1 \approx m_2$. Some sample values are given below, as well as a figure that plots m_1, m_2 versus the choice of prime α.

- If $\alpha = 41$ then $m_1 = m_2 = 25$.
- If $\alpha = 997$, then $m1 = m2 = 13$.
- If $\alpha = 4999$, then $m1 = m2 = 11$.
- If $\alpha = 7649$, then $m1 = 10$ and $m2 = 11$ which is a nice choice of parameters since we only need 10 weird RSA verifications (which are slow) and 1 regular RSA verification (which is fast).
- If $\alpha = 20663$ then $m1 = 9$ and $m2 = 10$ which is a nice choice of parameters since we only need 9 weird RSA verifications (which are slow) and 1 regular RSA verification (which is fast).
- If $\alpha = 30137$, this is the smallest α that gives us $m1 = m2 = 9$.
- If $\alpha = 33469$, then $m1 = m2 = 9$.



2.6 Detailed Spec

The input is the RSA public key $PK = (N, e)$. The following specification is for the improved protocol made non-interactive, as described in Section 2.3. This specification assumes that the RSA exponent e is prime.

2.6.1 System Parameters

The system parameters are the RSA key length $|N|$, 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 `public string`.

2.6.2 Proving

System parameters:

1. `public string` (a octet string),
2. α (a prime number)
3. κ (the security parameter, use 128 by default)

Input: RSA exponent e and distinct primes p and q such that the RSA modulus is $N = pq$.

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

Algorithm:

1. Set m_1 and m_2 as in equation 2.
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
$$d_{NP} = (eN)^{-1} \bmod (p-1) \qquad d_{NP} = (eN)^{-1} \bmod (q-1) \tag{3}$$
5. For integer $i = 1 \dots m_2$
   (a) Sample ρ_i, a random element of Z_N, as
   $$\rho_i = \text{getRho}((N, e), |N|, \texttt{public string}, i, |N|, m_2)$$
   (b) If $i \leq m_1$, 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 $(N, e), \{\sigma_1, ..., \sigma_{m_2}\}$.

2.6.3 Verifying.

System parameters:

1. `public string` (a octet string),
2. α (a prime number)
3. κ (the security parameter, use 128 by default)
4. $|N|$, the RSA key length

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 $N \geq 2^{|N|-1}$ and $N < 2^{|N|}$. If not, output INVALID and stop.
2. Compute m_1 and m_2 per equation (2).
3. Check that there are exactly m_2 values $\{\sigma_1, ..., \sigma_{m_2}\}$ in the input. If not, output INVALID and stop.
4. 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 $\gcd(P, N) = 1$. If not, output INVALID and stop.

   (Alternatively, Let P be the primorial of $\alpha - 1$, ı.e., the product of all prime numbers up to (but not including) α. P should be computed once and should be a system parameter. Check that $\gcd(P, N) = 1$.)
5. For integer $i = 1 \ldots m_2$
   (a) Sample ρ_i, a random element of Z_N, as
   $$\rho_i = \text{getRho}((N, e), |N|, \texttt{public string}, i, |N|, m_2)$$
   (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 $\rho_i = \sigma_i^e \bmod N$).
6. Output VALID.

2.6.4 Auxiliary function: getRho

Rejection sampling of a pseudorandom element $\rho_i \in Z_N$.
This is a "deterministic" function that always produces the same output for a given input.
Input:

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

Output: ρ_i

Algorithm:

1. Let
$$|m_2| = \lceil \tfrac{1}{8}(\log_2(m_2 + 1)) \rceil$$
be the length of m_2 in octets. (Note: This is an octet length, not a bit length!)
2. Let $j = 2$.
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 `public string` be an octet string.
   (c) Let $EI = \text{I2OSP}(i, |m_2|)$ be the $|m_2|$-octet long string encoding of the integer i. (The I2OSP primitive is specified in [MKJR16, Sec. 4.2].)
   (d) Let $EJ = \text{I2OSP}(j, |j|)$ be the $|j|$-octet long string encoding of the integer j, where $|j| = \lceil \frac{1}{8} \log_2(j + 1) \rceil$.
   (e) Let $s = PK || \texttt{public string} || EI || EJ$ be the concatenation of these octet strings.
   (f) Let $ER = \text{MGF1-SHA256}(x, |N|)$ 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 $|N|$ bits long.
   (g) Let $\rho_i = \text{OS2IP}(ER)$ be an integer.
   (That is, convert ER to an $|N|$ bit integer ρ_i using the OS2IP primitive specified in [MKJR16, Sec. 4.1].)
   (h) If $\rho_i \geq N$, then let $j = j + 1$ and continue; Else, break.
   (Note: This step tests if $\rho_i \in Z_N$.)
4. Output integer ρ_i.

3 Proof-of-knowledge of RSA-Modulus Factorization from [PS00]

We now adapt the zero-knowledge proof-of-knowledge of factorization of an RSA modulus N by Poupard and Stern [PS00, Fig 1].[6] We make the protocol non-interactive (using the Fiat-Shamir Heuristic) and select its parameters so that we satisfy the security requirements in [PS00].

The protocol excerpted from [PS00, Fig 1] is shown below. We only need a the single "elementary round" shown in the excerpt (so $\ell = 1$). In our discussion, we will rename the variable e from [PS00] to variable w (to avoid confusion with the RSA exponent).

Let k be a security parameter. Let n be an integer whose number of digits in its binary expansion is denoted $|n|$. Let A, B, ℓ and K be integers which depend *a priori* on k and $|n|$. Let $z_1,...z_K$ be K elements randomly chosen in $\mathbb{Z}_n{}^*$. We describe an interactive proof of knowledge for the factorization of n.

A round of proof (see figure 1) consists for the prover in randomly choosing an integer r in $[0, A[$ and computing, for $i = 1..K$, the *commitments* $x_i = z_i^r \bmod n$. Then he sends the x_is to the verifier who answers a *challenge* e randomly chosen in $[0, B[$. The prover computes $y = r + (n - \varphi(n)) \times e$ (in $\mathbb{Z}$) and sends it to the verifier who checks $0 \leq y < A$ and, for $i = 1..K$, $z_i^{y-n\times e} = x_i \bmod n$. A complete proof consists in repeating ℓ times the elementary round.

Prover		Verifier
$r \in_R [0, A[$		
$x_i = z_i^r \bmod n$, for $i = 1..K$	$\xrightarrow{(x_i)_{i\in[1,K]}}$	
	$\xleftarrow{e}$	$e \in_R [0, B[$
$y = r + (n - \varphi(n)) \times e$	$\xrightarrow{y}$	check $0 \leq y < A$ and $z_i^{y-ne} \stackrel{?}{=} x_i \bmod n$, for $i = 1..K$

Fig. 1. Interactive proof of knowledge for factoring (elementary round)

Figure 1: Except from [PS00].

The protocol, as written in the excerpt of Poupard and Stern [PS00, Fig 1] is interactive, and therefore only statistically complete (*i.e.,* there is some probability that the Prover will fail to generate a valid proof, even if he knows the factorization of N). Specifically, the protocol fails to be complete if the Verifier chooses a w that causes y to be such that $y < 0$ or $y > A$. We make the protocol non-interactive using the Fiat-Shamir heuristic, as also suggested in [PS00, Section 3.4] . In doing this, we also use rejection sampling to make the protocol deterministically sound. Specifically, in our version of the protocol, the Prover randomly samples the value r, and then computes the value w (using the Fiat-Shamir heuristic) as the hash of the commitments $x_1 \ldots x_K$ and the modulus N. The Prover then computes y and confirms that $0 < y < A$. If not, the Prover simply chooses a new value of r and tries again.

Also, in the excerpt of [PS00] above, the values $z_1 \ldots z_K$ are integers randomly chosen from Z_N^*. (Since $z_i \in Z_N^*$ we require that $\gcd(z_i, N) = 1$ and $z_i < N$.) We shall select these z_i via rejection sampling using the output of a random oracle computed over a public octet string `public string` and the RSA modulus N. That is, the Prover $z_i \in Z_N^*$ by computing the output of the random

[6] We can either implement Fig 1 or Fig 2 from this paper. The computational cost is the same but size of the proof is smaller in Fig 2 because of the use of the hash function H(); however, we don't care about the length of the proof so we just use the protocol in Figure 1.

oracle over the concatenation of (1) the modulus N and (2) a public string given as a system parameter `public string`, and (3) the index i. Thus, a given modulus N and `public string` determines a deterministic set of $\rho_i \in Z_N$. The Verifier can therefore compute $\rho_i \in Z_N$ alone, by following the same procedure as the Prover.

Parameter selection. Suppose the security parameter is κ and the RSA key size is $|N|$. How can we set parameters (namely A, B, K) for the protocol?

1. [PS00] shows that zero-knowledge requires that B is polynomial in κ and

$$(N - \phi(N))B/A \tag{4}$$

   is "negligible", *i.e.,* assymptotically smaller than $\frac{1}{k^c}$ for any constant c.

2. [PS00] shows that completeness requires that equation (4) is "negligible". Even though we use rejection-sampling to reject values r that cause the prover to fail completeness, we shall still choose A and B so that equation (4) is negligible, since we anyway require this in order to satisfy zero-knowledge. As such, the probability that the Prover will have to reject r and try again is negligible.

3. [PS00] shows that soundness requires that

$$\log B = \theta(\kappa) \tag{5}$$

   and $A < n$ and $\log(A)$ is polynomial in k and $|N|$ and

$$K = \theta(k + \log |N|)$$

   More precisely, we require that $\frac{\eta}{2^K}$ is negligible, where η is the number of prime factors of N (see the bottom of page 7 of [PS00], which argues that $\eta < \log |N|$ for any N).

Following similar setting of parameters as [PS00, Sec. 5], we can take $B = 2^k$ to satisfy equation (5), which is the most stringent requirement on B. We set $A = 2^{|N|-1}$ so that we satisfy $|A| = \theta(|N|)$ and $A < N$ since N is an RSA modulus where $N > 2^{|N|-1}$.[7] Finally, we could set

$$K = \lceil (\kappa + \log_2(|N|) \rceil \tag{6}$$

for the general case when N could be the product of more than two prime factors.

However, in our specification that follows, we will assume that $N = pq$ is the product of exactly two prime factors p and q so that $\eta = 2$. Thus, we can slightly optimize this by taking

$$K = \kappa + 1 \tag{7}$$

Implementations that do not make this assumption should instead choose K as in equation (6). When $N = pq$ where p and q are large primes so that $|p| = |q| = \frac{|N|}{2}$, our choice of parameters fulfils completeness and zero-knowledge, since equation (4) becomes

$$\frac{(N - \phi(N))B}{A} = \frac{(N - \phi(N))2^\kappa}{2^{|N|-1}} = \frac{(pq - (p-1)(q-1))2^\kappa}{2^{|N|-1}} = \frac{p + q - 1}{2^{|N|-1}} 2^\kappa \approx \frac{2^{|N|/2+1}}{2^{|N|-1}} 2^\kappa = 2^{-(|N|/2-\kappa-2)}$$

which is negligible for any "reasonable" choice of RSA key length $|N|$ and security parameter κ.

[7] For some reason, [PS00, Sec 5.] recommends setting $A = 2^{1024}$ when N is a 1024-bit integer, and then subsequently states that this parameter selection ensures that $A < N$. We assume that this is a typo, and instead set $A = 2^{|N|-1}$. Indeed, soundness in [PS00] requires that $A < N$ to ensure that $y < N$; if $y > N$, the prover can cheat.

3.1 Choice of parameters for TumbleBit

For the TumbleBit Setup Protocol, $\kappa = 128$, $|N| = 2048$ and `public string` is the SHA256 hash of the blockchain's Genesis block. This means that $K = \kappa + 1 = 129$.

This protocol, as written here, also requires that the TumbleBit public key (N, e) to be such that (1) N is the product of two large primes p, q, *i.e.,* $N = pq$ and (2) $N > 2^{|N|-1} = 2^{2047}$ and (3) that

$$(N - (p-1)(q-1)) \times 2^{\kappa} \ll N$$

(which is guaranteed if (1) holds).

3.2 Detailed specification

The input is the RSA public key $PK = (N, e)$.

The system parameters are the RSA key length $|N|$, the security parameter κ in bits (where by default $\kappa = 128$), and a publicly-known octet string `public string`.

Assumptions. The following specification assumes that both κ and $|N|$ are divisible by 8. If that is not the case, set $\kappa = 8\lceil \kappa/8 \rceil$.

3.2.1 Proving.

Assumption: The algorithm assumes that PK is an RSA public key such that the modulus N satisfies $N > 2^{|N|-1}$ (this is a common requirement from an RSA modulus) and $(N-\phi(N))2^{\kappa} \ll N$ (which will be the case when N is an RSA modulus that is the product of two large primes p, q such that $N = pq$). If these assumptions are not satisfied, then verification could fail even if the Prover behaves honestly.

System parameters:

1. Octet string `public string`,
2. κ the security parameter in bits, use 128 by default
3. $|N|$ the length of the RSA modulus, in bits

Input: Public key $PK = (N, e)$ and distinct primes p and q such that the RSA modulus is $N = pq$.

Goal: Prove that you know prime integers p, q such that $pq = N$.

Algorithm:

1. Check that $N \geq 2^{|N|-1}$ and that $N < 2^{|N|}$ and that

$$\frac{2^{|N|-1}}{2^{\kappa}(N-(p-1)\cdot(q-1))} > 2^{\kappa}$$

   If not, stop and output "Bad RSA modulus N".

2. Let K be as in equation (7).
3. Use cryptographic randomness to sample a random $|N|-1$-bit integer r. (Note that $0 \leq r < 2^{|N|-1}$.)
4. For integer $i = 1...K$
   (a) Sample a random element $z_i \in Z_N^*$ by taking

$$z_i = \text{sampleFromZnStar}(PK, \texttt{public string}, i, K, |N|)$$

   (b) Compute $x_i = (z_i)^r \mod N$.
5. Let

$$w = \text{getW}(PK, \texttt{public string}, x_1, ..., x_K, \kappa, |N|)$$

   be a κ bit integer.
6. Let $y = r + (N-(p-1)\cdot(q-1))\cdot w$. (Note: This is integer arithmetic.)
7. If $y \geq 2^{|N|-1}$ or $y < 0$, return to Step 3.
8. Output $PK, \{x_1, ..., x_K\}, y$.

3.3 Verifying.

System parameters:

1. Octet string `public string`,
2. κ the security parameter in bits, use 128 by default
3. $|N|$ the length of the RSA modulus, in bits

Input: $PK, \{x_1, ..., x_K\}, y$, where $PK = (N, e)$

Output: VALID or INVALID.

1. Check that $0 \leq y < 2^{|N|-1}$. If not, output INVALID and stop.
2. Check that $N \geq 2^{|N|-1}$ and that $N < 2^{|N|}$. If not, output INVALID and stop.
3. Let K be as in equation (7).
4. If the number of x_i values in the input does not equal K, output INVALID and stop.
5. Let
$$w = \text{getW}(PK, \texttt{public string}, x_1, ..., x_K, \kappa, |N|)$$
be a κ bit integer.
6. Let $r' = y - N \cdot w$. (Note: This is integer arithmetic.)
7. For integer $i = 1...K$
   (a) Let $z_i = \text{sampleFromZnStar}(PK, \texttt{public string}, i, K, |N|)$
   (b) Check that $x_i = (z_i)^{r'} \mod N$. If not, output INVALID and stop.
8. Output VALID.

3.3.1 Auxiliary function: sampleFromZnStar

This function performs rejection sampling of a pseudorandom element $z_i \in Z_N^*$.
This is a "deterministic" function that always produces the same output for a given input.

Input:

1. RSA public key $PK = (N, e)$.
2. Octet string `public string`
3. Index integer i.
4. Integer K
5. Length of RSA modulus $|N|$ in bits.

Output: Integer z_i

Overview: We iterate over index j until the value $z_i = H(PK||\texttt{public string}||i||j)$ is such that $0 < z_i < N$ and $\gcd(N, z_i) = 1$, so that $z_i \in Z_N^*$.

Algorithm:

1. Let $|i| = \left\lceil \frac{\log_2(K+1)}{8} \right\rceil$ be the length of the indices i in octets. (Note: This is an octet length, not a bit length!)
2. Let $j = 2$.
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 `public string` be an octet string.
   (c) Let $EI = \text{I2OSP}(i, |i|)$ be the $|i|$-octet long string encoding of the integer i. (The I2OSP primitive is specified in [MKJR16, Sec. 4.2].)
   (d) Let $EJ = \text{I2OSP}(j, |j|)$ be the $|j|$-octet long string encoding of the integer j, where $|j| = \lceil \frac{1}{8} \log_2(j+1) \rceil$.
   (e) Let $s = PK||\texttt{public string}||EI||EJ$ be the concatenation of these octet strings.
   (f) Let $Z_i = \text{MGF1-SHA256}(x, |N|)$ 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 $|N|$ bits long.
   (g) Let $z_i = \text{OS2IP}(Z_i)$ be an integer.
   (That is, convert Z_i to an $|N|$ bit integer z_i using the OS2IP primitive specified in [MKJR16, Sec. 4.1].)
   (h) If $z_i \geq N$ or $\gcd(z_i, N) \neq 1$, then let $j = j + 1$ and continue; Else, break.
   (Note: This step tests if $z_i \in Z_N^*$.)
4. Output integer z_i.

3.3.2 Auxiliary function: getW

This function computes the κ-bit integer $w = H(PK||\texttt{public string}||\{x_1, ..., x_K\})$.
This is a "deterministic" function that always produces the same output for a given input.

Input:

1. RSA public key $PK = (N, e)$.
2. Octet string `public string`
3. Set of integers $\{x_1, \ldots, x_K\}$.
4. Security parameter κ in bits.
5. Length of RSA public key $|N|$.

Output: Integer w

Algorithm:

1. Let PK be the ASN.1 octet string encoding of the RSA public key (N, e) as specified in [MKJR16, Appendix A].
2. Let `public string` be an octet string.
3. Let $EX_i = \text{I2OSP}(x_i, \lceil \frac{1}{8}|N| \rceil)$ be the octet string encoding of the integer x_i, for each $i = 1 \ldots K$. (Note: EX_i is $|N|$ bits long, and consists of $\lceil \frac{1}{8}|N| \rceil$ octets. The I2OSP primitive is specified in [MKJR16, Sec. 4.2].)
4. Let $s = PK||\texttt{public string}||EX_1|| \ldots ||EX_K$ be the concatenation of these octet strings.
5. Let $W = H_2(s)$ where H_2 is SHA-256 with output truncated to κ bits.
6. Let $w = \text{OS2IP}(W)$ be an integer.
7. Output integer w.

References

[BY96] Mihir Bellare and Moti Yung. Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. *J. Cryptology*, 9(3):149–166, 1996. `https://cseweb.ucsd.edu/~mihir/papers/cct.html`.

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 1999. `http://www.brics.dk/RS/98/29/BRICS-RS-98-29.pdf`.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Eurocrypt*, volume 99, pages 402–414. Springer, 1999.

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

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[GMR98] Rosario Gennaro, Daniele Micciancio, and Tal Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In Li Gong and Michael K. Reiter, editors, *CCS '98, Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, CA, USA, November 3-5, 1998.*, pages 67–72. ACM, 1998. http://eprint.iacr.org/1998/008.

[HAB+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. NDSS, 2017. https://eprint.iacr.org/2016/575.pdf.

[KKM12] Saqib A. Kakvi, Eike Kiltz, and Alexander May. Certifying RSA. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 404–414. Springer, 2012. http://www.cits.rub.de/imperia/md/content/may/paper/main.pdf.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *Eurocrypt*, volume 3027, pages 74–90. Springer, 2004.

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

[MPS00] Philip D. MacKenzie, Sarvar Patel, and Ram Swaminathan. Password-authenticated key exchange based on RSA. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 599–613. Springer, 2000. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.3089&rep=rep1&type=pdf.

[PS00] Guillaume Poupard and Jacques Stern. Short proofs of knowledge for factoring. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography (PKC), Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings*, volume 1751 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 2000. https://link.springer.com/chapter/10.1007%2F978-3-540-46588-1_11?LI=true.

[Sho09] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, second edition, 2009. http://www.shoup.net/ntb/ntb-v2.pdf.

[WCZ03] Duncan S. Wong, Agnes Hui Chan, and Feng Zhu. More efficient password authenticated key exchange based on RSA. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings*, volume 2904 of *Lecture Notes in Computer Science*, pages 375–387. Springer, 2003. `http://www.ccs.neu.edu/home/ahchan/wsl/papers/pake_indocrypt03.pdf`.