Non-Interactive Provably Secure Attestations for Arbitrary RSA Prime Generation Algorithms

Fabrice Benhamouda[1], Houda Ferradi[2], Rémi Géraud[3], and David Naccache[3]

[1] IBM Research, Yorktown Heights, United States
fabrice.benhamouda@normalesup.org
[2] NTT Secure Platform Laboratories
3–9–11 Midori-cho, Musashino-shi, Tokyo 180–8585, Japan
ferradi.houda@lab.ntt.co.jp
[3] Département d'informatique de l'ENS, École normale supérieure,
CNRS, PSL Research University, Paris, France
given_name.family_name@ens.fr

Abstract RSA public keys are central to many cryptographic applications; hence their validity is of primary concern to the scrupulous cryptographer. The most relevant properties of an RSA public key (n, e) depend on the *factors* of n: are they properly generated primes? are they large enough? is e co-prime with $\phi(n)$? etc. And of course, it is out of question to reveal n's factors.
Generic non-interactive zero-knowledge (NIZK) proofs can be used to prove such properties. However, NIZK proofs are not practical at all. For some very specific properties, specialized proofs exist but such *ad hoc* proofs are naturally hard to generalize.
This paper proposes a new type of *general-purpose* compact non-interactive proofs, called *attestations*, allowing the key generator to convince any third party that n was properly generated. The proposed construction applies to *any* prime generation algorithm, and is provably secure in the Random Oracle Model.
As a typical implementation instance, for a 138-bit security, verifying or generating an attestation requires $k = 1024$ prime generations. For this instance, each processed message will later need to be signed or encrypted 14 times by the final users of the attested moduli.

Keywords: RSA key generation, random oracle, non-interactive proof.

1 Introduction

When provided with an RSA public key n, establishing that n is hard to factor might seem challenging: indeed, most of n's interesting properties depend on its secret factors, and even given good arithmetic properties (large prime factors, etc.) a subtle backdoor may still be hidden in n or e [1, 27, 28, 30, 31].

Several approaches, mentioned below, focused on proving as many interesting properties as possible without compromising n. However, such proofs are limited in two ways: first, they might not always be applicable — for instance [2, 3, 19] cannot prove that (n, e) define a permutation when e is too small. In addition, these *ad hoc* proofs are extremely specialized. If one wishes to prove some new property of n's factors, that would require modelling this new property and looking for a proper form of proof.

This paper proposes a new kind of general-purpose compact non-interactive proof ω_n, called *attestation*. An attestation allows the key generator to convince any third party that n was properly generated. The corresponding construction, called an *attestation scheme*, applies to *any* prime generation algorithm $\mathcal{G}(1^P, r)$ where r denotes $\mathcal{G}$'s random tape, and P the size of the generated primes. The method can, for instance, attest that n is composed of primes as eccentric as those for which $\lfloor 9393 \sin^4(p^3) \rfloor = 3939$.

More importantly, our attestation scheme provides the first efficient way to prove that (n, e) defines a permutation for a small e, by making $\mathcal{G}$ only output primes p such that e is coprime with $p - 1$.

Our construction is provably secure in the Random Oracle Model.

We present two variants: In the first, a valid attestation ω_n ensures that n contains at least two P-bit prime factors generated by $\mathcal{G}$ (if n is honestly generated, n must contain ℓ prime factors, for some integer $\ell \geq 2$ depending on the security parameter). In the second variant, a valid attestation $\omega_{\mathbf{n}}$ covers a set of moduli $\mathbf{n} = (n_1, \ldots, n_u)$ and ensures that at least one of these n_i is a product of two P-bit prime factors generated by $\mathcal{G}$.

Both variants are unified into a general attestation scheme (i.e., use several multi-factor moduli) to encompass the entire gamut of tradeoffs offered by the concept.

Prior Work. A long thread of papers deals with proving number-theoretic properties of composite moduli. The most general (yet least efficient) of these use non-interactive zero-knowledge (NIZK) proof techniques [8, 11, 15]. Recent work by Groth [16] establishes that there is a perfect NIZK argument for n being a properly generated RSA modulus. We distinguish between these *generic* proofs that can, in essence, prove anything provable [4] and *ad hoc* methods allowing to prove proper modulus generation in faster ways albeit for very specific $\mathcal{G}$s.

The first *ad hoc* modulus attestation scheme was introduced by Van de Graff and Peralta [26] and consists in proving that n is a Blum integer without revealing its factors. Boyar, Friedl and Lund [7] present a proof that n is square-free. Leveraging [7, 26], Gennaro, Micciancio and Rabin [14] present a protocol proving that n is the product of two "quasi-safe" primes[4]. Camenisch and Michels [9] give an NIZK proof that n is a product of two safe primes. Juels and Guajardo [18] introduce a proof for RSA key generation with verifiable randomness. Besides its complexity, [18]'s main drawback is that public parameters must be published by a trustworthy authority (TTP). Several authors [5, 10, 21, 22] describe protocols proving that n is the product of two primes p and q, without proving anything on p, q but their primality. Proving that $n = pq$ is insufficient to ascertain security

[4] A prime p is "quasi-safe" if $p = 2u^a + 1$ for a prime u and some integer a.

(for instance, p may be too short). Hence, several authors (e.g., [6,10,12,13,20,21]) introduced methods allowing to prove that p and q are roughly of identical sizes.

This work takes an *entirely different direction*: Given any generation procedure $\mathcal{G}$, we prove that $\mathcal{G}$ has been followed correctly during the generation of n. The new approach requires no TTPs, does not rely on n having any specific properties and attests that the correct prime generation algorithm has been used — with no restriction whatsoever on how this algorithm works.

As such, the concern of generating proper moduli (e.g. such that (N, e) define a permutation, but what constitutes a "proper" modulus may depend on the application) is entirely captured by the concern of choosing $\mathcal{G}$ appropriately. Our work merely attests that $\mathcal{G}$ was indeed used.

Cryptographic applications of attested RSA moduli abound. We refer the reader to [14] or [21] for an overview of typical applications of attested moduli. In particular, such concerns are salient in schemes where an authority is in charge of generating n (e.g., Fiat-Shamir or Guillou-Quisquater) and distributing private keys to users, or in the design of factoring-based verifiable secret-sharing schemes.

Another context in which this work has its place is to protect against the subversion of key generation procedures, as studied in e.g., [27,29–31]. A recent effort in that direction is [24].

2 Outline of the Approach

The proposed attestation method is based on the following idea: fix $k \geq 2$, generate k random numbers $r_1, \ldots, r_k$ and define $h_i = \mathcal{H}(i, r_i)$ where $\mathcal{H}$ denotes a hash function. Let $p_i = \mathcal{G}(h_i)$ and:

$$N = \prod_{i=1}^{k} p_i$$

Define $(X_1, X_2) = \mathcal{H}'_2(N)$, where $\mathcal{H}'_2$ is a hash function which outputs two indices $1 \leq X_1 < X_2 \leq k$. We later show how to construct such an $\mathcal{H}'_2$. This defines $n = p_{X_1} \times p_{X_2}$ and

$$\omega_n = \{r_1, r_2, \ldots, r_{X_1-1}, \star, r_{X_1+1}, \ldots, r_{X_2-1}, \star, r_{X_2+1}, \ldots, r_k\}$$

Here, a star symbol ($\star$) denotes a placeholder used to skip one index. The data ω_n is called the *attestation* of n. The algorithm $\mathcal{A}$ used to obtain ω_n is called an *attestator*.

The attestation process is illustrated in Figure 1: the choice of the r_i determines N, which is split into two parts: n and N/n. Splitting is determined by d, which is the digest of N, and is hence unpredictable for the opponent.

Verifying the validity of such an attestation ω_n is performed as follows: all (non-star) values r_i in ω_n are fed to $\mathcal{G}$ to generate primes, that are multiplied together and by n. This gives back N. If by hashing N and reading, as earlier, the digest of N (denoted d) as two values X_1 and X_2, we get the two exact starred

Figure 1. The approach used to generate and validate an attestation.

$$\begin{tikzcd}
r_1 \arrow[r, "\mathcal{H}(1, r_1)"] & h_1 \arrow[r, "\mathcal{G}(h_1)"] & p_1 & & & \\
\vdots & \vdots & \vdots & & & \\
r_{X_1} \arrow[r, "\mathcal{H}(X_1, r_{X_1})"] & h_{X_1} \arrow[r, "\mathcal{G}(h_{X_1})"] & p_{X_1} & & & \\
\vdots & \vdots & \vdots \arrow[r, phantom, "\Bigg\}"] & {} \arrow[r, "\times"] & N \arrow[r, "\mathcal{H}'_2(N)"] & d = \{X_1, X_2\} \arrow[lllllu, bend right] \arrow[lllll d, bend left] \\
r_{X_2} \arrow[r, "\mathcal{H}(X_2, r_{X_2})"] & h_{X_2} \arrow[r, "\mathcal{G}(h_{X_2})"] & p_{X_2} & & & \\
\vdots & \vdots & \vdots & & & \\
r_k \arrow[r, "\mathcal{H}(k, r_k)"] & h_k \arrow[r, "\mathcal{G}(h_k)"] & p_k & & &
\end{tikzcd}$$

positions X_1 and X_2 in ω_n, then ω_n is valid; else ω_n is invalid. The algorithm $\mathcal{V}$ we just described is called a *validator*. It is very similar to the attestator $\mathcal{A}$ mentioned above.

For a subtle reason, the r_i's are pre-processed into a set of values h_i before being fed into $\mathcal{G}$. The values h_i are generated by hashing the input r_is with their positions i. This serves two purposes: first, the hash welds together r_i and its position i in the list, which prevents the opponent from shuffling the p_is to his advantage; second, hashing prevents the opponent from manipulating the r_i's to influence $\mathcal{G}$'s output.

Evidently, as presented here, the method requires a very large k to achieve a high enough security level. The attacker, who chooses X_1, X_2, is expected to perform $k(k-1)/2$ operations to succeed. We circumvent this limitation using two techniques:

– The first technique uses ℓ indices $X_1, \ldots, X_\ell$ and not only $\ell = 2$. In RSA, security depends on the fact that n contains *at least* two properly formed prime factors. Hence we can afford to shorten k by allowing more factors in n. The drawback of using ℓ-factor moduli is a significant user slow-down as most factoring-based cryptosystems run in $O(\log^3 n)$. Also, by doing so, we prove that n *contains* a properly formed modulus rather than that n *is* a properly formed modulus.

– A second strategy consists in using $2u$ indices to form u moduli $n_1, \ldots, n_u$. Here, each user will be given u moduli and will process[5] each message u times. Thereby, total signature size and slow-down are only linear in ℓ. Encryption is more tricky: while for properly signing a message it suffices that *at least* one n_i is secure, when encrypting a message *all* n_i must be secure. Hence, to encrypt, the sender will pick u session keys κ_i, encrypt each κ_i using n_i, and form the global session-key $\kappa = \kappa_1 \oplus \ldots \oplus \kappa_u$. The target message will then be encrypted (using a block-cipher) using κ. In other words, it suffices

[5] Sign, verify, encrypt, or decrypt.

to have at least *one* factoring-resistant n_i to achieve message confidentiality. Interestingly, to be secure a signature conceptually behaves as a logical "or", while encryption behaves as a logical "and".

The size of ω_n is also a concern in this simple outline. Indeed, as presented here ω_n is $O(kR)$ bits large, where R represents the bitsize of the r_i[6]. Given the previous remark on k being rather large, this would result in very large attestations. Luckily, it turns out that attestation size can be reduced to $O(R \log k)$ using hash trees, as we explain in Section 5.

Note. Multiplication in $\mathbb{N}$ is one implementation option. All we need is a *completely multiplicative operation*. For instance, as we have:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\cdots\left(\frac{a}{p_k}\right),$$

the hash of the product of the Jacobi symbols of the p_i with respect to the first primes $a_j = 2, 3, 5, \ldots$[7] can equally serve as an index generator.

Before we proceed note that when generating a complete RSA key pair (n, e), it is important to ascertain that $\gcd(e, \phi(n)) = 1$. This constraint is easy to integrate into $\mathcal{G}$[8]. All in all, what we prove is that with high probability, *the key was generated by the desired algorithm* $\mathcal{G}$, whichever this $\mathcal{G}$ happens to be.

3 Model and Analysis

3.1 Preliminaries and Notations

We now formally introduce the tools necessary to rigorously describe and analyse the method sketched in Section 2.

Throughout this paper, λ will denote a security parameter. The expression *polynomial time* will always refer to λ. The construction uses two cryptographic hash functions: a classical hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^R$ and a second hash function $\mathcal{H}'_d : \{0,1\}^* \to \mathcal{S}_d$ where $\mathcal{S}_d$ is the set of subsets of $\{1, \ldots, k\}$ of size d (for some positive integer d and k). $\mathcal{H}'$ can be constructed from a classical hash function using an unranking function [25] (see Appendix A). Both hash functions will be modelled as random oracles in the security analysis.

Let $k \geq 2$. Moreover our attestation and validation algorithms always implicitly take λ as input. We denote by $|a|$ the bitsize of a.

Let $\mathcal{G}(1^P, r)$ be a polynomial-time algorithm which, on input of a unary size P and of a random seed $r \in \{0,1\}^R$ produces a prime or a probably prime p of

[6] Because $\mathcal{G}$ may destroy entropy, R must be large enough to make the function $\mathcal{G}(\mathcal{H}(i, r))$ collision resistant.

[7] This product is actually an a_j-wise exclusive-or.

[8] A simple way to do so consists in re-running $\mathcal{G}$ with $r_i \| j$ (instead of r_i) for $j = 1, 2, \ldots$ until $\gcd(p_i - 1, e) = 1$.

size P. The argument 1^P is often omitted, for the sake of simplicity. The size P of the primes is supposed to be a function of λ. We write $r_1 \xleftarrow{\$} \{0,1\}^R$ to indicate that the seed r_1 is chosen uniformly at random from $\{0,1\}^R$.

An attestation scheme for $\mathcal{G}$ is a pair of two algorithms $(\mathcal{A}, \mathcal{V})$, where

- $\mathcal{A}$ is an *attestation algorithm* which takes as input k random entries ($(r_1, \ldots, r_k) \in \{0,1\}^R$, in the sequel) and which outputs a tuple of moduli $\mathbf{n} = (n_1, \ldots, n_u)$ along with a bitstring $\omega_{\mathbf{n}}$, called an *attestation*; u and k are integer parameters depending on λ; when $u = 1$, n_1 is denoted n;
- $\mathcal{V}$ is a *validation algorithm* which takes as input a tuple of moduli $\mathbf{n} = (n_1, \ldots, n_u)$ together with an attestation $\omega_{\mathbf{n}}$. $\mathcal{V}$ checks $\omega_{\mathbf{n}}$, and outputs True or False.

An *attestation scheme* must comply with the following properties:

- *Randomness.* If $r_1, \ldots, r_k$ are independent uniform random values, $\mathcal{A}(1^\lambda, r_1, \ldots, r_k)$ should output a tuple of moduli $\mathbf{n} = (n_1, \ldots, n_u)$ where each n_i is the product of ℓ random primes generated by $\mathcal{G}$. The positive integer $\ell \geq 2$ is a parameter depending on λ. More formally the two following distributions should be statistically indistinguishable:

$$\left\{ \mathbf{n} = (n_1, \ldots, n_u) \;\middle|\; \begin{array}{l} (r_1, \ldots, r_k) \xleftarrow{\$} \{0,1\}^R \\ (n_1, \ldots, n_u, \omega_n) \leftarrow \mathcal{A}(r_1, \ldots, r_k) \end{array} \right\}$$

$$\left\{ \mathbf{n} = (n_1, \ldots, n_u) \;\middle|\; \begin{array}{l} (r_1, \ldots, r_{\ell u}) \xleftarrow{\$} \{0,1\}^R \\ n_1 \leftarrow \mathcal{G}(r_1) \cdots \mathcal{G}(r_\ell), \ldots, n_u \leftarrow \mathcal{G}(r_{(u-1)\ell+1}) \cdots \mathcal{G}(r_{u\ell}) \end{array} \right\}$$

- *Correctness.* The validator $\mathcal{V}$ always accepts an attestation honestly generated by the attestator $\mathcal{A}$. More precisely, for all $r_1, \ldots, r_k$:

$$\mathcal{V}\left(\mathcal{A}(1^\lambda, r_1, \ldots, r_k)\right) = \mathsf{True}.$$

- *Soundness.* No polynomial-time adversary $\mathcal{F}$ can output (with non-negligible probability) a tuple $\mathbf{n} = (n_1, \ldots, n_u)$ and a valid attestation $\omega_{\mathbf{n}}$ such that no n_i contains at least two prime factors generated by $\mathcal{G}$ with two distinct random seeds. More formally, for any polynomial-time adversary $\mathcal{F}$, the soundness advantage $\mathsf{Adv}^{\mathrm{snd}}(\mathcal{F})$ defined as

$$\Pr\left[(\mathbf{n} = (n_1, \ldots, n_u), \omega_{\mathbf{n}}) \xleftarrow{\$} \mathcal{F}(1^\lambda) \;\middle|\; \begin{array}{l} \mathcal{V}(n_1, \ldots, n_u, \omega_n) = \mathsf{True} \text{ and} \\ \forall i = 1, \ldots, u,\ \nexists s_1, s_2 \in \{0,1\}^R, \\ \qquad s_1 \neq s_2 \text{ and } \mathcal{G}(s_1) \cdot \mathcal{G}(s_2) \text{ divides } n_i \end{array} \right]$$

is negligible in λ.
- *Non-revealing.* We formalise the property than an attestation does not leak sensitive information about the attested modulus as follows: An attestation algorithm $\mathcal{A}$ is said to be *non-revealing* if, for any $\mathbf{n}$, any PPT adversary $\mathcal{F}$ and any computable property $P(\mathbf{n}) \in \{0,1\}$ of $\mathbf{n}$ alone, the advantage of $\mathcal{F}$ in computing $P(\mathbf{n})$ knowing the output $\omega_{\mathbf{n}}$ of $\mathcal{A}$ is at most negligibly higher than without knowing $\omega_{\mathbf{n}}$.

Table 1. Summary of the various parameters

λ	security parameter (all the other parameters are function of λ)
P	size of prime numbers p_i generated by $\mathcal{G}$
R	size of the seed used by $\mathcal{G}$ to generate a prime number
k	number of primes generated by the attestator $\mathcal{A}$, which is the *dominating cost of* $\mathcal{A}$
u	number of moduli output by $\mathcal{A}$ ($u = 1$ in the multi-prime variant, and $u \geq 2$ in the multi-modulus variant)
ℓ	number of factors of each modulus n_i: $\lvert n_i \rvert = \ell P$

We remark that when it is hard to find two seeds s_1 and s_2 such that $\mathcal{G}(s_1) = \mathcal{G}(s_2)$, then soundness basically means that one of the n_i's contains a product of two distinct primes generated by $\mathcal{G}$. In addition, when $\ell = 2$, if $\mathcal{V}$ rejects moduli of size different from $2P$ (the size of an honestly generated modulus), one of the n_i's is necessarily exactly the product of two prime factors generated by $\mathcal{G}$.

Table 1 summarizes the various parameters used in our construction (all are assumed to be function of λ). We now describe the following two variants:

- The multi-prime variant, where $\mathcal{A}$ only outputs one modulus (i.e., $u = 1$);
- The multi-modulus variant, where $\mathcal{A}$ outputs $u \geq 2$ two-factor moduli (i.e., $\ell = 2$).

3.2 Multi-Prime Attestation Scheme ($u = 1$)

We now describe the algorithms $\mathcal{A}$ and $\mathcal{V}$ that generate and verify, respectively, an attestation along with an RSA public key, when $u = 1$ (only one modulus is generated). Algorithms in this Section are given for $\ell = 2$ (corresponding to the common case where $n = pq$) for the sake of clarity and as a warm-up.

Algorithms for arbitrary ℓ are particular cases of the general algorithms described in Section 3.4.

In Algorithms 1 and 2, a star symbol ($\star$) denotes a placeholder used to skip one index.

Generating an Attestation. The attestator $\mathcal{A}$ is described in Algorithm 1. $\mathcal{A}$ calls $\mathcal{H}$ and $\mathcal{G}$.

Algorithm 1: Attestator $\mathcal{A}$ for the attestation scheme $(u = 1,\ \ell = 2)$

Input: $r_1, \ldots, r_k$.
Output: n, ω_n.

```
1. $N \leftarrow 1$
2. for all $i \leftarrow 1$ to $k$
3.     $h_i \leftarrow \mathcal{H}(i, r_i)$
4.     $p_i \leftarrow \mathcal{G}(h_i)$
5.     $N \leftarrow N \times p_i$
6. $X_1, X_2 \leftarrow \mathcal{H}_2'(N)$
7. $\omega_n \leftarrow \{r_1, \ldots, r_{X_1-1}, \star, r_{X_1+1}, \ldots, r_{X_2-1}, \star, r_{X_2+1}, \ldots, r_k\}$
8. $n \leftarrow p_{X_1} \times p_{X_2}$
9. return $n, \omega_n$
```

In this setting, the attestation has size k. This size is reduced to $\log k$ using hash trees as described in Section 5.

Verifying an Attestation. The validator $\mathcal{V}$ is described in Algorithm 2.

Algorithm 2: Validator $\mathcal{V}$ for the attestation scheme $(u = 1,\ \ell = 2)$

Input: n, ω_n.
Output: True or False.

```
1. $N \leftarrow n$
2. for all $r_i \neq \star \in \omega_i$
3.     $h_i \leftarrow \mathcal{H}(i, r_i)$
4.     $p_i \leftarrow \mathcal{G}(h_i)$
5.     $N \leftarrow N \times p_i$
6. $X_1, X_2 \leftarrow \mathcal{H}_2'(N)$
7. if $r_{X_1} = \star$ and $r_{X_2} = \star$ and $\#\{r_i \in \omega_n \text{ s.t. } r_i = \star\} = 2$ and $|n| = \ell P$
8.     return True
9. return False
```

Correctness: The h_is are generated deterministically, therefore so are the p_is, and their product times n yields the correct value of N.

Randomness: In the Random Oracle Model (for $\mathcal{H}$), the scheme's *randomness* is proven later in Section 4.1, as a particular case of the general scheme's soundness (see Section 3.4).

3.3 Multi-Modulus Attestation Scheme $(u \geq 2,\ \ell = 2)$

The second variant consists in generating in a batch $u = \ell/2$ bi-factor moduli. The corresponding attestator and validator are given in Algorithms 3 and 4.

Algorithm 3: Attestator $\mathcal{A}$ for the attestation scheme $(u \geq 2,\ \ell = 2)$

Input: $r_1, \ldots, r_k$.
Output: $\boldsymbol{n} = (n_1, \ldots, n_u), \omega_{\boldsymbol{n}}$.

```
1.  $N \leftarrow 1$
2.  for $i \leftarrow 1$ to $k$
3.        $h_i \leftarrow \mathcal{H}(i, r_i)$
4.        $p_i \leftarrow \mathcal{G}(h_i)$
5.        $N \leftarrow N \times p_i$
6.  $X_1, \ldots, X_{2u} \leftarrow \mathcal{H}'_{2u}(N)$
7.  $\omega_{\boldsymbol{n}} \leftarrow \{r_1, \ldots, r_{X_1-1}, \star, r_{X_1+1}, \ldots, r_{X_{u\ell}-1}, \star, r_{X_{u\ell}+1}, \ldots, r_k\}$
8.  for $j \leftarrow 1$ to $u$
9.        $n_j \leftarrow p_{X_{2j}} \times p_{X_{2j+1}}$
10. return $\boldsymbol{n} = (n_1, \ldots, n_u), \omega_{\boldsymbol{n}}$
```

Algorithm 4: Validator $\mathcal{V}$ for the attestation scheme $(u \geq 2,\ \ell = 2)$

Input: $\boldsymbol{n} = (n_1, \ldots, n_u), \omega_{\boldsymbol{n}}$.
Output: True or False

```
1.  $N \leftarrow n_1 \times \cdots \times n_u$
2.  for $r_i \neq \star \in \omega_{\boldsymbol{n}}$
3.        $h_i \leftarrow \mathcal{H}(i, r_i)$
4.        $p_i \leftarrow \mathcal{G}(h_i)$
5.        $N \leftarrow N \times p_i$
6.  $X_1, \ldots, X_{2u} \leftarrow \mathcal{H}'_{2u}(N)$
7.  if $r_j = \star$ for all $j = 1$ to $u$ and $\#\{r_i \text{ s.t. } r_i = \star\} = 2u$ and $|n_1| = \cdots = |n_u| = 2P$
8.        return True
9.  return False
```

3.4 General Attestation Scheme

Algorithms 5 and 6 describe our general attestation scheme, for any $u \geq 1$ and $\ell \geq 2$. The previous multi-prime and multi-modulus schemes are illustrative particular cases of this scheme.

The *correctness* and *randomness* arguments are similar to those of Section 3.2. In addition, the attestation has size k. This size is brought down to $\ell u \log k$ using hash-trees as described in Section 5.

Algorithm 5: Attestator $\mathcal{A}$ for the general scheme ($u \geq 1$, $\ell \geq 2$)

Input: $r_1, \ldots, r_k$.
Output: $\boldsymbol{n} = (n_1, \ldots, n_u), \omega_{\boldsymbol{n}}$.

```
1. $N \leftarrow 1$
2. for $i \leftarrow 1$ to $k$
3.     $h_i \leftarrow \mathcal{H}(i, r_i)$
4.     $p_i \leftarrow \mathcal{G}(h_i)$
5.     $N \leftarrow N \times p_i$
6. $X_1, \ldots, X_{u\ell} \leftarrow \mathcal{H}'_{u\ell}(N)$
7. $\omega_{\boldsymbol{n}} \leftarrow \{r_1, \ldots, r_{X_1-1}, \star, r_{X_1+1}, \ldots, r_{X_{u\ell}-1}, \star, r_{X_{u\ell}+1}, \ldots, r_k\}$
8. for $j \leftarrow 1$ to $u$
9.     $n_j \leftarrow p_{X_{(\ell-1)j+1}} \times \cdots \times p_{X_{\ell j}}$
10. return $\boldsymbol{n} = (n_1, \ldots, n_u), \omega_{\boldsymbol{n}}$
```

Algorithm 6: Validator $\mathcal{V}$ for the general scheme ($u \geq 1$, $\ell \geq 2$)

Input: $\boldsymbol{n}, \omega_{\boldsymbol{n}}$.
Output: True or False

```
1. $N \leftarrow n_1 \times \cdots \times n_u$
2. for $r_i \neq \star$ in $\omega_{\boldsymbol{n}}$
3.     $h_i \leftarrow \mathcal{H}(i, r_i)$
4.     $p_i \leftarrow \mathcal{G}(h_i)$
5.     $N \leftarrow N \times p_i$
6. $X_1, \ldots, X_{2u\ell} \leftarrow \mathcal{H}'_{u\ell}(N)$
7. if $r_{X_j} = \star$ for $j = 1$ to $\ell$ and $\#\{r_i \text{ s.t. } r_i = \star\} = u\ell$ and $|n_1| = \cdots = |n_u| = \ell P$
8.     return True
9. return False
```

4 Security and Parameter Choice

4.1 Security

In this section, we prove that for correctly chosen parameters u, ℓ, k, the general attestation scheme defined in Section 3.4 (Algorithms 5 and 6) is sound. We recall that the two other properties required by an attestation scheme (namely correctness and randomness) were proven in previous sections.

More formally, we have the following theorem:

Theorem 1. *In the Random Oracle Model, the soundness advantage of an adversary making $q_{\mathcal{H}}$ queries to $\mathcal{H}$ and $q_{\mathcal{H}'}$ queries to $\mathcal{H}'$ is at most:*

$$(q_{\mathcal{H}'} + 1) \cdot \left(\frac{\ell u}{k - (\ell - 1)u + 1}\right)^{(\ell-1)u} + \frac{q_{\mathcal{H}} \cdot (q_{\mathcal{H}} - 1)}{2} \cdot p_{\mathcal{G}-\text{col}} \, ,$$

where $p_{\mathcal{G}-\text{col}}$ is the probability that $\mathcal{G}(r) = \mathcal{G}(s)$, when $r, s \xleftarrow{\$} \{0,1\}^R$.

We point out that $p_{\mathcal{G}-\mathrm{col}}$ must be small, otherwise the generated primes are unsafe in any case.

Proof. First, we denote by S_i the set of all prime numbers $\rho = \mathcal{G}(\mathcal{H}(i,r))$, for which (i,r) has been queried to $\mathcal{H}$ (for $i = 1,\ldots,k$). We remark that the probability that two such primes ρ are equal is at most $\frac{q_{\mathcal{H}}\cdot(q_{\mathcal{H}}-1)}{2} \cdot p_{\mathcal{G}-\mathrm{col}}$. This is the second term in the security bound.

In the sequel, we suppose that there are no collisions between the primes. Thus the sets S_i are pairwise disjoint.

Now assume that the adversary $\mathcal{F}$ has been able to forge a valid attestation $\omega_{\mathbf{n}}$ for $\mathbf{n} = (n_1,\ldots,n_u)$ and let $N = \beta\prod_{i=1}^{u} n_i$, where β stands for the product of all the primes generated from the elements of $\omega_{\mathbf{n}}$. As the attestation is valid, $|n_1| = \cdots = |n_u| = \ell P$. Let $N = \prod_{i=1}^{L}\rho_i$ be the prime decomposition of N. Up to reordering the sets S_i, there exists an integer t such that:

– none of $S_1,\ldots,S_t$ contains a factor ρ_i;
– each of $S_{t+1},\ldots,S_k$ contains a factor ρ_i. We arbitrarily choose a prime $p_i \in S_i$ for $i = t+1,\ldots,k$.

We distinguish two cases:

– if $t < (\ell-1)\cdot u$, then this means that N is divisible by $m = p_{t+1}\times\cdots\times p_k$. But we also know that N is divisible by $n_1\times\cdots\times n_u$. As $|n_1\times\cdots\times n_u| = \ell uP$, $|m| = (k-t)P \geq kP - (\ell-1)uP + P$, and $|N| = kP$, we have

$$|\gcd(n_1\cdots n_u, m)| \geq |n_1\cdots n_u| + |m| - |N| \geq (u+1)P.$$

This implies that $n_1\times\cdots\times n_u$ is divisible by at least $u+1$ distinct primes among $p_{t+1},\ldots,p_k$. By the pigeon-hole principle, at least one of the n_i's is divisible by two distinct primes generated as $\mathcal{G}(r_i)$ for two distinct seeds r_i (seeds have to be distinct, otherwise the two primes would be equal).

– if $t \geq (\ell-1)\cdot u$, the adversary will only be able to generate a valid attestation if none of the indices $X_1,\ldots,X_{u\ell}$ (obtained by $\mathcal{H}'_{u\ell}(N)$) falls in $\{1,\ldots,t\}$. As $\{1,\ldots,k\}\setminus\{X_1,\ldots,X_{u\ell}\}$ is a random subset of $\{1,\ldots,k\}$ with $k-\ell u$ elements, the previous bad event ($\mathcal{F}$ is able to generate a valid attestation) corresponds to this set being a subset of $\{t+1,\ldots,k\}$ and happens with probability:

$$\begin{aligned}\frac{\binom{k-t}{k-\ell u}}{\binom{k}{k-\ell u}} &= \frac{(k-t)\cdot(k-t-1)\cdots(k-\ell u+1)}{k\cdot(k-1)\cdots(k-\ell u+1)}\cdot\frac{(\ell u)!}{(\ell u-t)!}\\ &\leq \frac{1}{(k-t+1)^t}\cdot(\ell u)^t \leq \left(\frac{\ell u}{k-(\ell-1)u+1}\right)^{(\ell-1)\cdot u}.\end{aligned}$$

Since $\mathcal{F}$ makes $q_{\mathcal{H}'}$ queries to $\mathcal{H}'$, we get the theorem's bound (where the $+1$ corresponds to the query necessary to verify $\mathcal{F}$'s attestation if he did not do it himself).

□

Theorem 2. *In the programmable random oracle model, our attestations are non-revealing.*

Proof. The proof strategy consists in replacing the hash functions by a random oracle, resulting in attestations which are in particular completely unrelated to the modulus' factorization. We give the proof in the multi-prime $\ell = 2$ case. The more general case is similar.

Let n be an RSA modulus, and let $\omega_n = (r_1, \ldots, r_k)$, where there are exactly two values $r_{X_1} = r_{X_2} = \star$, be an attestation.

Assume that there exists a PPT adversary $\mathcal{A}$ that can compute some property $P(n)$ from the knowledge of n and ω_n, with access to the hash functions $\mathcal{H}$ and $\mathcal{H}'_2$, with non-negligible advantage. Since $\mathcal{A}$ uses $\mathcal{H}'_2$ as a black box, we can replace $\mathcal{H}'_2$ by a programmable random oracle as follows.

We compute

$$N = n \times \prod_{i=1, i \neq X_1, X_2}^{k} \mathcal{G}\left(\mathcal{H}\left(i, r_i\right)\right).$$

Now $\mathcal{H}'_2$ is replaced by a random oracle that returns $\{X_1, X_2\}$ if its input equals N, and a couple of random distinct integers in $\{1, \ldots, k\}$ otherwise. In particular, note that $\mathcal{H}'_2$ is not given the factorization of n. With this choice of $\mathcal{H}'_2$, ω_n is a valid attestation for n.

However, by design, ω_n is chosen independently from n. Thus it is clear that if $\mathcal{A}$ can compute $P(n)$ from the knowledge of n and ω_n, in fact $\mathcal{A}$ can compute $P(n)$ from n alone. □

4.2 Typical Parameters and Complexity Analysis

Algorithms 5 and 6 have the following properties:

- Attestation size $|\omega_n| = 2u\ell R \log k$, using the hash-tree compression technique in Section 5
- λ-bit security approximatively when:

$$\left(\frac{\ell u}{k - (\ell - 1)u + 1}\right)^{(\ell-1)u} \leq 2^{-\lambda}$$

  (according to the soundness bound given by Theorem 1, omitting the second part, which is negligible in practice);
- Attestation and validation times mostly consist in generating (or re-generating) the k primes. Validation time is very slightly faster than attestation time.

5 Compressing the Attestation

As mentioned above, providing an attestation ω_n "as is" might be cumbersome, as it grows linearly with k. However, it is possible to drastically reduce ω_n's size using the following technique.

The tree of Figure 2 is constructed as follows: Let h be some public hash function. Each non-leaf node C of the tree has two children, whose value is computed by $r_{x0} \leftarrow h(r_x, 0)$ and $r_{x1} \leftarrow h(r_x, 1)$ for the left child and the right child respectively, where r_x is the value of C. Given a root seed r, one can therefore reconstruct the whole tree. The leaf values can now be used as r_i's for the attestation procedure.

To compress ω_n we proceed as follows:

- Get the indices X_1 and X_2 from the attestation procedure;
- Identify the paths from X_1 up to the root, and mark them;
- Identify the paths from X_2 up to the root, and mark them;
- Send the following information:

$$\omega_n = \{\text{for all leaves } L\text{, highest-ranking unmarked parent of } L\}$$

This requires revealing at most $2\log_2 k$ intermediate higher-rank hashes[9] instead of the $k-2$ values required to encode ω_n when naively sending the seeds directly.

Generalization to $u\ell \geq 2$ is straightforward.

6 Parameter Settings

Table 2 shows typical parameter values illustrating different tradeoffs between security (λ), attestation size ($2u\ell R \log k$), modulus size (ℓ), the number of required moduli (u), and the work factors of $\mathcal{A}$ and $\mathcal{V}$ ($kt_{\mathcal{G}}$ where $t_{\mathcal{G}}$ is $\mathcal{G}$'s average running time). Table 3 provides the same information for the multi-modulus variant.

We (arbitrarily) consider that reasonable attestations and validations should occur in less than ten minutes using standard HSM such as the IBM 4764 PCI-X Cryptographic Coprocessor [17] or Oracle's Sun Crypto Accelerator SCA 6000 [23]. When run with 7 threads in the host application, the 4764 generates on average 2.23 key-pairs per second (1,024 bits). The SCA 6000 (for which average key generation figures are not available) is about 11 times faster than the 4764 when processing RSA 1,024-bit keys. Hence we can assume that the SCA 6000 would generate about 24 key-pairs per second. We thus consider that average-cost current-date HSMs generate 10 key-pairs per second, i.e., 20 primes per second.

Spending ten minutes to generate or validate an attestation might not be an issue given that attestation typically occurs only once during n's lifetime. This means that a "reasonable" attestation implementation would use $k = 10 \times 60 \times 20 = 12{,}000$. This gives $\ell = 10$ and $\ell = 6$ for the multi-prime and multi-modulus $\mathcal{A}$ (respectively) for $\lambda = 128$.

Note that in practical field deployments an attestation would be verified *once* by a trusted *Attestation Authority* and replaced by a signature on n (or $\mathbf{n}$).

According to the bounds of Theorem 1, we have

$$\lambda \geq -(\ell - 1)u \log_2 \left(\frac{\ell u}{k - (\ell - 1)u + 1} \right)$$

[9] I.e., we essentially only publish co-paths.

$$\begin{tikzcd}
 & & & r_{111} = h(r_{11}, 1) \\
 & & r_{11} = h(r_1, 1) \arrow[ru] \arrow[rd] & \\
 & & & r_{110} = h(r_{11}, 0) \\
 & r_1 = h(r, 1) \arrow[ruu] \arrow[rdd] & & \\
 & & & r_{101} = h(r_{10}, 1) \\
 & & r_{10} = h(r_1, 0) \arrow[ru] \arrow[rd] & \\
 & & & r_{100} = h(r_{10}, 0) \\
r \arrow[ruuuu] \arrow[rdddd] & & & \\
 & & & r_{011} = h(r_{01}, 1) \\
 & & r_{01} = h(r_0, 1) \arrow[ru] \arrow[rd] & \\
 & & & r_{010} = h(r_{01}, 0) \\
 & r_0 = h(r, 0) \arrow[ruu] \arrow[rdd] & & \\
 & & & r_{001} = h(r_{00}, 1) \\
 & & r_{00} = h(r_0, 0) \arrow[ru] \arrow[rd] & \\
 & & & r_{000} = h(r_{00}, 0)
\end{tikzcd}$$

Figure 2. Compressing ω_n using a hash tree.

Table 2 is read as follows: we can see that taking for instance $\ell = 10$ and $\log_2 k = 13$ with the multi-factor version gives 156-bit security. In Table 3, taking $\ell = 10$ and $\log_2 k = 13$ with the multi-modulus version gives 285-bit security.

7 Conclusion and Further Research

The construction described in this paper attests in a non-interactive way that n was properly generated using an arbitrary (publicly known) prime generator $\mathcal{G}$. The attestation is compact and publicly verifiable. As a result, any entity can convince herself of the modulus' validity before using it. Even though computation times may seem unattractive, we stress that attestation generation and verification only need to be performed once.

This work raises a number of interesting questions.

Committing to the primes p_i's might also be achieved using more involved tools such as pairings. For instance, given the commitments g^{p_1} and g^{p_2}, it is easy to check that $e(g^{p_1}, g^{p_2}) = e(g, g)^n$.

An interesting research direction consists in hashing $N \bmod v$ (instead of N) for some public v, to speed-up calculations. However, the condition $v > n$ must be enforced by design to prevent an opponent from using ω_n as the "attestation" of $n + tv$ for some $t \in \mathbb{N}$. Note that we did not adapt our security proof to this (overly?) simplified variant.

In general, any strategy allowing to reduce k without impacting λ would yield more efficient attestators. Also, generalizing and applying this approach to the parameter generation of other cryptographic problems, such as the discrete logarithm, may prove useful.

Finally, to date, no attestation method proves (without resorting to TTPs) that the random tape used for forming the primes was properly drawn. Like all other prior work articles cited in Section 1, we do not address this issue and assume that the random number that feeds $\mathcal{G}$ was not biased by the attacker.

Acknowledgements. The first author was supported by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO) under Contract No.W911NF-15-C-0236.

References

1. Anderson, R.: Practical RSA trapdoor. Electronics Letters 29(11), 995–995 (1993)
2. Bellare, M., Yung, M.: Certifying cryptographic tools: The case of trapdoor permutations. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 442–460. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1993)
3. Bellare, M., Yung, M.: Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. Journal of Cryptology 9(3), 149–166 (1996)
4. Ben-Or, M., Goldreich, O., Goldwasser, S., Håstad, J., Kilian, J., Micali, S., Rogaway, P.: Everything provable is provable in zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO'88. LNCS, vol. 403, pp. 37–56. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 21–25, 1990)

Table 2. Some typical parameters for multi-factor attestation ($u = 2$). Each table entry contains λ for the corresponding choice of k and ℓ.

$\log_2 k$	Time	$\ell = 6$	$\ell = 8$	$\ell = 10$	$\ell = 12$	$\ell = 14$	$\ell = 16$	$\ell = 18$	$\ell = 20$
8	25 s	43	54	64	72	79	84	89	93
9	51 s	53	69	83	95	107	117	126	135
10	1.7 min	64	83	101	118	134	148	162	175
11	3.4 min	74	97	119	140	160	179	197	214
12	6.8 min	84	111	138	162	186	209	231	253
13	13.7 min	94	125	156	185	212	239	266	291
14	27.3 min	104	139	174	207	238	269	300	329
15	54.6 min	114	153	192	229	264	299	334	367
16	1.8 hrs	124	167	210	251	290	329	368	405
17	3.6 hrs	134	181	228	273	317	359	402	443
18	7.3 hrs	144	195	246	295	343	389	436	481
19	14.6 hrs	154	209	264	317	369	419	470	519
20	1.2 d	164	223	282	339	395	449	504	557
21	2.4 d	174	237	300	361	421	479	538	595

Table 3. Some typical parameters for multi-modulus attestation ($u = \ell/2$). Each cell contains λ for the corresponding choice of k and ℓ. Some choices of parameters are incompatible and are hence indicated by a dash.

$\log_2 k$	Time	$\ell = 6$	$\ell = 8$	$\ell = 10$	$\ell = 12$	$\ell = 14$	$\ell = 16$	$\ell = 18$	$\ell = 20$	$\ell = 30$	$\ell = 40$	$\ell = 50$	$\ell = 60$	$\ell = 70$	$\ell = 80$
7	12 s	39	46	33	-	-	-	-	-	-	-	-	-	-	-
8	25 s	56	79	93	92	69	11	-	-	-	-	-	-	-	-
9	51 s	71	109	145	173	191	194	176	131	-	-	-	-	-	-
10	1.7 min	87	138	193	246	295	338	371	391	169	-	-	-	-	-
11	3.4 min	102	167	239	315	393	469	542	611	801	519	-	-	-	-
12	6.8 min	117	195	285	383	487	594	704	814	1315	1600	1470	655	-	-
13	13.7 min	132	223	330	450	579	717	861	1011	1786	2505	3036	3248	2989	2064
14	27.3 min	147	251	375	516	671	838	1016	1204	2239	3342	4410	5347	6065	6468
15	54.6 min	162	279	420	582	762	959	1170	1396	2682	4150	5705	7267	8768	10143
16	1.8 hrs	177	307	465	648	853	1079	1324	1586	3121	4944	6964	9109	11319	13540
17	3.6 hrs	192	335	511	714	944	1199	1477	1777	3558	5731	8205	10914	13800	16814
18	7.3 hrs	207	363	556	780	1036	1319	1630	1967	3994	6514	9439	12702	16248	20030
19	14.6 hrs	222	391	601	846	1127	1439	1783	2157	4430	7296	10668	14480	18679	23217
20	1.2 d	237	419	646	912	1218	1559	1936	2347	4865	8076	11895	16255	21102	26391
21	2.4 d	252	447	691	978	1309	1679	2089	2537	5300	8857	13121	18027	23521	29558

5. Boneh, D., Franklin, M.K.: Efficient generation of shared RSA keys (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 425–439. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 1997)
6. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg, Germany, Bruges, Belgium (May 14–18, 2000)
7. Boyar, J., Friedl, K., Lund, C.: Practical zero-knowledge proofs: Giving hints and using deficiencies. In: Quisquater, J.J., Vandewalle, J. (eds.) EUROCRYPT'89. LNCS, vol. 434, pp. 155–172. Springer, Heidelberg, Germany, Houthalen, Belgium (Apr 10–13, 1990)
8. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. Journal of Computer and System Sciences 37(2), 156–189 (1988)
9. Camenisch, J., Michels, M.: Separability and efficiency for generic group signature schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 413–430. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999)
10. Chan, A.H., Frankel, Y., Tsiounis, Y.: Easy come - easy go divisible cash. In: Nyberg, K. (ed.) EUROCRYPT'98. LNCS, vol. 1403, pp. 561–575. Springer, Heidelberg, Germany, Espoo, Finland (May 31 – Jun 4, 1998)
11. Cramer, R., Damgård, I.: Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 424–441. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 23–27, 1998)
12. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 1997)
13. Fujisaki, E., Okamoto, T.: A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In: Nyberg, K. (ed.) EUROCRYPT'98. LNCS, vol. 1403, pp. 32–46. Springer, Heidelberg, Germany, Espoo, Finland (May 31 – Jun 4, 1998)
14. Gennaro, R., Micciancio, D., Rabin, T.: An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In: ACM CCS 98. pp. 67–72. ACM Press, San Francisco, CA, USA (Nov 2–5, 1998)
15. Goldreich, O., Micali, S., Wigderson, A.: How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 171–185. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 1987)
16. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 339–358. Springer, Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006)
17. IBM: 4764 PCI-X Cryptographic Coprocessor, see http://www-03.ibm.com/security/cryptocards/pcixcc/overperformance.shtml
18. Juels, A., Guajardo, J.: RSA key generation with verifiable randomness. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 357–374. Springer, Heidelberg, Germany, Paris, France (Feb 12–14, 2002)
19. 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, Germany, Beijing, China (Dec 2–6, 2012)
20. Liskov, M., Silverman, B.: A statistical-limited knowledge proof for secure RSA keys (1998), manuscript

21. Mao, W.: Verifiable partial sharing of integer fractions. In: Tavares, S.E., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 94–105. Springer, Heidelberg, Germany, Kingston, Ontario, Canada (Aug 17–18, 1999)
22. Micali, S.: Fair public-key cryptosystems. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 113–138. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1993)
23. Oracle: Sun Crypto accelerator SCA 6000, see http://www.oracle.com/us/products/servers-storage/036080.pdf
24. Russell, A., Tang, Q., Yung, M., Zhou, H.: Cliptography: Clipping the power of kleptographic attacks. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10032, pp. 34–64 (2016), https://doi.org/10.1007/978-3-662-53890-6_2
25. Stanton, D., White, D.: Constructive combinatorics. Springer-Verlag New York, Inc. (1986)
26. van de Graaf, J., Peralta, R.: A simple and secure way to show the validity of your public key. In: Pomerance, C. (ed.) CRYPTO'87. LNCS, vol. 293, pp. 128–134. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1988)
27. Young, A., Yung, M.: The dark side of "black-box" cryptography, or: Should we trust capstone? In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 89–103. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996)
28. Young, A., Yung, M.: Kleptography: Using cryptography against cryptography. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 62–74. Springer, Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997)
29. Young, A., Yung, M.: The prevalence of kleptographic attacks on discrete-log based cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 264–276. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 1997)
30. Young, A., Yung, M.: Malicious cryptography: Kleptographic aspects (invited talk). In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 7–18. Springer, Heidelberg, Germany, San Francisco, CA, USA (Feb 14–18, 2005)
31. Young, A., Yung, M.: A space efficient backdoor in RSA and its applications. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 128–143. Springer, Heidelberg, Germany, Kingston, Ontario, Canada (Aug 11–12, 2006)

A Implementing the Second Hash Function $\mathcal{H}'$

To implement the second hash function $\mathcal{H}'_d$ from a classical hash function, we can apply an unranking hash function [25], which maps an integer (in some interval) to a subset $\{X_1, \ldots, X_u\} \subset \{0, \ldots, k-1\}$.

As an example, we describe here a simple (natural) unranking function. Let $\mathcal{H}''$ be a classical hash function with range $\{0, \ldots, M\}$, where $M = k(k-1)\cdots(k-u+1)-1$. To hash a value N, we first compute $r \leftarrow \mathcal{H}''(N)$. Then we compute the integers $r_1, \ldots, r_d$ as in Algorithm 7.

Algorithm 7: Unranking Algorithm

```
Input: $r, k, u$.
Output: $r_1, \dots, r_u$.

1. for all $i = 1$ to $u$
2.     $r_i \leftarrow r \bmod (k - i + 1)$
3.     $r \leftarrow r$ div $(k - i + 1)$
4. return $r_1, \dots, r_u$
```

Algorithm 7 generates a mixed radix representation of r, hence any $r \in [0, M]$ can be represented this way. We now generate the unranking $X_1, \dots, X_d$ iteratively as follows:

- $X_1 \leftarrow r_1$
- $X_{i+1} \leftarrow r_{i+1} + \#\{X_j \text{ s.t. } X_j \leq r_{i+1} \text{ for } j \leq i\}$

In other terms, we have a pool of M values, and for each i, one of these values is drawn and assigned to X_i. Hence it is easy to check that this provides a list of pairwise distinct integers.

This algorithm is simple and illustrates how unranking may be implemented. Alternative unranking methods can be found in [25].