An implementation of the Paillier crypto system with threshold decryption without a trusted dealer

Thijs Veugen[1,2], Thomas Attema[1,2], and Gabriele Spini[1]

[1] TNO, Unit ICT, The Hague, The Netherlands
{thijs.veugen,thomas.attema,gabriele.spini}@tno.nl
[2] CWI, Cryptology group, Amsterdam, The Netherlands

Abstract. We consider the problem of securely generating the keys of the Paillier crypto system [11] with (t, n) threshold decryption, without a trusted dealer. Nishide and Sakurai [10] describe a solution, secure in the malicious model. We use their ideas to make a simpler solution for the semi-honest model, and further introduce a few optimisations. We implement the secure key generation protocol on a single computer, and consider its performance.

Keywords: Paillier · threshold decryption · secure key generation.

1 Introduction

Using threshold decryption in a public key cryptosystem with n parties, a minimal number of parties is required to decrypt a ciphertext. It can exclude the situation where a single party (holding the decryption key) is able to decrypt all sensitive information. In [5] it was shown how to build generic secure multiparty computation (MPC) protocols from additively homomorphic encryption with threshold decryption, where multiple parties can compute with encrypted sensitive inputs.

Although many cryptographic frameworks use additively homomorphic encryption with threshold decryption, the set-up of the key material is often neglected. This is either left to a trusted dealer, an external party assumed to be trusted by all players, or is considered as a secure computation to be done within any generic MPC framework, without specifying the computational steps. A trusted dealer is not an ideal situation, since it re-introduces a similar security risk that should be avoided by threshold decryption. Therefore, we only consider the setting where the n parties together generate the key material in a secure way.

In 2001, Boneh and Franklin described how to securely and efficiently generate RSA keys with two players [3]. Somewhat later, in 2010, Nishide and Sakurai [10] were the first ones describing these steps in detail for the Paillier crypto system. Their solution is secure in the malicious security model, but they did not

consider the performance of their method. We will describe and explain the different steps that are needed to securely generate the keys of Paillier with threshold decryption, and analyse its performance. We consider a semi-honest model, which leads to lower computational and communication complexity. While implementing the key generation protocol, we found a few optimisations to reduce the complexity, and slightly increase security. The authors are unaware of an earlier implementation of generating Paillier keys with threshold decryption, without a trusted dealer (and more than two players).

In addition, our implementation can be used to construct RSA-groups for which only a sufficiently large set of players can reconstruct the order. Groups of unknown order are in turn used to construct other cryptographic primitives such as RSA-accumulators [2], and verifiable delay functions [14]. See Table 1 for an overview of our notation throughout the paper.

σ	statistical security parameter	κ	bit size of RSA primes
n	number of parties	N	Paillier modulus
$\varphi(\cdot)$	Euler's totient function	p, q	large (RSA) primes
P	large public prime	g, θ	part of the public key
p	small prime	λ	private Paillier key
ν	n factorial	T	subset of parties
t	decryption threshold	s	secret
S	upper bound on s	c	cipher text
$\mathcal{P}_S$	set of polynomials	ℓ_i	integer Lagrange coefficients
$f(x), g(x), h(x)$	polynomials	B	algorithm parameter
$\langle\cdot\rangle_t$	Shamir secret sharing with threshold t	$\langle\cdot\rangle_t^{\mathbb{Z}}$	Secret sharing over the integers with threshold t

Table 1. Notation

1.1 Paillier with threshold decryption

In Paillier [11] the maximal plain text size N is the product of two large primes p and q. The number g is an element of $\mathbb{Z}_{N^2}^*$ with a nonzero multiple of N as order, typically $g = N + 1$. The public key is (N, g), the private key λ is, for example, Euler's totient $\varphi(N) = (p-1)(q-1)$.

To encrypt a message $m \in \mathbb{Z}_N$, a random number $r \in \mathbb{Z}_{N^2}$ is generated. The ciphertext c is computed as

$$c = g^m \cdot r^N \bmod N^2.$$

To decrypt, we use the property $(N+1)^x = N \cdot x + 1 \pmod{N^2}$ for any positive integer x, which is easily shown by means of binomial coefficients. Define the function $L(x)$ as $(x-1)/N$, then

$$m = L(c^\lambda \bmod N^2) \cdot \lambda^{-1} \bmod N,$$

where λ^{-1} is the multiplicative inverse of λ modulo N. The random factor $r^{N\lambda}$ disappears modulo N^2 because $\varphi(N^2) = N\lambda$.

To achieve threshold decryption with a trusted dealer [7], the dealer generates a random $\beta \in \mathbb{Z}_N^*$, and a secret sharing $\langle \lambda \cdot \beta \rangle_t$ over $\mathbb{Z}_{\varphi(N^2)}$ of the private key λ, multiplicatively blinded by β. This means that each player i receives $h(i) \bmod \varphi(N^2)$, where $h(x)$ is a t-degree polynomial with $h(0) = \lambda \cdot \beta$. The number $\theta = (\lambda \cdot \beta) \bmod N$ is added to the public key (N, g). Then, given a decryption set T of $t+1$ players, a ciphertext c can be decrypted as follows:

1. Each party of the decryption set T computes $c_i = c^{h(i)} \bmod N^2$, and reveals it.
2. The parties compute the Lagrange coefficients ℓ_i, $i \in T$ (see Subsection 2.1 for the integer variant) such that $h(0) = \sum_{i \in T} \ell_i h(i) \bmod \varphi(N^2)$, and compute $c^{\lambda \cdot \beta} = \prod_{i \in T} c_i^{\ell_i} \bmod N^2$.
3. The parties compute $m = L(c^{\lambda \cdot \beta} \bmod N^2) \cdot \theta^{-1} \bmod N$.

The $(\cdot)^{-1}$ in the last step denotes multiplicative inverse modulo N.

In order to eliminate the trusted dealer, we need an MPC protocol to securely generate N and a secret sharing of $\lambda \cdot \beta$. However, we cannot disclose $\varphi(N^2)$, so we use secret-sharing over the integers (see Section 2).

1.2 Related work

Quite some related work has been done in the secure generation of RSA numbers. In 1997 Boneh and Franklin [3] described a two-party protocol for RSA key generation. They used a third 'helper' party, which was overcome by Gilboa [8] in 1999. In 2010, Damgård and Mikkelsen [6] developed the first constant-round solution in the malicious model.

In 2010, Nishide and Sakurai provided a solution for generating threshold keys for Paillier in the malicious model [10]. Hazay et al. extended this to the dishonest majority case [9], and implement their key generation protocol in a two-party setting.

We closely follow the work of Nishide and Sakurai, and implement a semi-honest version, thereby slightly improving their results (not only due to the different security model), see Subsection 3.8 for the details. We first explain an important building block, namely secret sharing over the integers [12], before we describe the different computational steps of the secure key generation protocol. We end with the performance results, and the conclusions.

2 Secret sharing over the integers

Shamir secret sharing is typically used to secretly share a number s in a finite field. In our solution, we will construct a secret sharing of the private key λ, but we cannot afford carry-overs during decryption (see Section 3). Therefore, we need Shamir secret sharing over the integers [12]. Since the constant $n!$ is often used there, we define $\nu = n!$.

To generate a Shamir secret sharing over the integers $\langle s \rangle_t^{\mathbb{Z}}$, a random polynomial $f(x)$ of degree t is generated, with integer coefficients α_i, $1 \leq i \leq t$, uniformly chosen from the interval $[0, 2^\sigma \nu^2 S]$, where σ is the statistical security parameter, and S is an upper bound on the secret s: $0 \leq s < S$ [10]. The first coefficient $\alpha_0 = f(0)$ equals νs, and each party i is given share $f(i)$, $1 \leq i \leq n$. The extra factor ν in α_0 is needed for security reasons. Although standard Shamir secret sharing is information theoretically secure, its variant over the integers provides σ bits statistical security (see Subsection 2.3).

2.1 Revealing a secret

Given the shares $f(i)$, $i \in T$, for a subset T of parties, one can reconstruct the secret s by Lagrange interpolation, as long as $|T| > t$:

1. Compute the integer coefficients ℓ_i, $i \in T$, such that
$$\ell_i = \left(\prod_{j \in T, j \neq i} (-j) \right) \cdot \left(\frac{\nu}{\prod_{j \in T, j \neq i} (i-j)} \right).$$
2. Compute $\nu \cdot f(0) = \sum_{i \in T} \ell_i f(i)$, and divide $\nu \cdot f(0) = \nu^2 s$ by ν^2.

The extra factor ν assures that each ℓ_i is an integer, which is necessary for computing $c_i^{\ell_i} \bmod N^2$ later (see Section 3). To see that ℓ_i is integer, observe that $\left(\frac{\nu}{\prod_{j \in T, j \neq i} (i-j)} \right)$ is the product of the binomial coefficient $\binom{n}{i}$, and the integers $\frac{i!}{\prod_{j \in T, j < i} (i-j)}$ and $\frac{(n-i)!}{\prod_{j \in T, j > i} (i-j)}$.

2.2 Computing with shared secrets

Given two polynomials $f(x)$ and $g(x)$, with secrets $s_1 = f(0)$ and $s_2 = g(0)$, a secret sharing of the sum $s_1 + s_2$ is easily computed by adding the two polynomials, which means that each player i, $1 \leq i \leq n$, needs to locally add its shares $f(i)$ and $g(i)$.

As usual, multiplication is a bit more involved. However, if the players locally multiply their shares, they construct a secret-sharing of the product: $\langle \nu \cdot s_1 \cdot s_2 \rangle_{2t}^{\mathbb{Z}} = \langle s_1 \rangle_t^{\mathbb{Z}} \cdot \langle s_2 \rangle_t^{\mathbb{Z}}$. Besides from the extra factor ν, The main difference is that the resulting polynomial $f(x) \cdot g(x)$ now has degree $2t$, so we need at least $2t+1$ players to reveal a secret. Therefore, multiplication can be securely performed in case of honest majority ($t < n/2$).

2.3 Security

It is known that $t+1$ parties are able to retrieve s from $\langle s \rangle_t^{\mathbb{Z}}$, but it is less clear how much information on s is obtained by t (or fewer) parties. The following theorem shows that secret sharing over the integers is statistically secure.

Theorem 1. *Let $\mathcal{P}_S$ be the set of polynomials $f(x)$ of degree t with each coefficient being a non-negative integer, upper bounded by $2^\sigma \nu^2 S$, and $f(0) = 0$. Let s be a secret, $0 \leq s < S$. Let $f(x)$ be a polynomial, uniformly random chosen from $\mathcal{P}_S$. Let T be a subset of t parties, each party i having share $\nu \cdot s + f(i)$.*

Then, given any secret $\tilde{s}$, $0 \leq \tilde{s} < S$, the probability that polynomial $\tilde{f}(x) - \nu \cdot \tilde{s} \in \mathcal{P}_S$ is lower bounded by $1 - t2^{1-\sigma}$, where $\tilde{f}(x)$ is the t-degree polynomial such that $\tilde{f}(i) = f(i)$, $i \in T$, and $\tilde{f}(0) = \nu \cdot \tilde{s}$.

Proof. Let $p_T(x)$ be the polynomial $\prod_{i \in T}(x - i) \cdot \frac{\nu}{\prod_{i \in T}(-i)}$. This is a polynomial of degree t with integer coefficients, for which $p_T(i) = 0$, $i \in T$, and $p_T(0) = \nu$. Therefore, $\tilde{f}(x) = f(x) + \tilde{s} \cdot p_T(x)$ is the t-degree polynomial we are looking for. To be sure that $\tilde{f}(x) - \nu \cdot \tilde{s} \in \mathcal{P}_S$, we only need to show the right size of the coefficients.

Consider the polynomial $\prod_{i=1}^{n}(x - i)$. It is easily shown by induction that the absolute value of the coefficients of this polynomial are upper bounded by $n + 1$ factorial. It follows that the absolute values of the coefficients of p_T are upper bounded by ν^2. Finally, we have $\tilde{f}(x) - \nu \cdot \tilde{s} \in \mathcal{P}_S$, when the coefficients of $f(x)$ are in the interval $[\nu^2 S, (2^\sigma - 1)\nu^2 S]$. This occurs with probability $\left(\frac{2^\sigma - 2}{2^\sigma}\right)^t$, which is lower bounded by $1 - t2^{1-\sigma}$.

The lower bound $1 - t2^{1-\sigma}$ is exponentially (in σ) close to one, concluding that our secret sharing scheme is indeed statistically secure.

3 Distributed key generation

Inspired by the ideas of Nishide and Sakurai [10], our distributed key generation protocol for Paillier consists of the following steps:

1. Jointly generate secret-sharings $\langle p \rangle_t$ and $\langle q \rangle_t$ of random numbers p and q.
2. Compute $\langle N \rangle_{2t} = \langle p \rangle_t \cdot \langle q \rangle_t$, reveal the product $N = p \cdot q$, and test for small prime divisors. If the test fails, restart the protocol.
3. Test the biprimality of N. If the test fails, restart the protocol.
4. Securely compute $\langle \lambda \rangle_t^{\mathbb{Z}}$, jointly generate a secret sharing $\langle \beta \rangle_t^{\mathbb{Z}}$ of a random $\beta \in \mathbb{Z}_N^*$, and securely compute $\langle \nu \cdot \lambda \cdot \beta \rangle_{2t}^{\mathbb{Z}}$.
5. Securely compute and reveal $\theta = (\nu^2 \cdot \lambda \cdot \beta) \bmod N$.

The details of the above steps, and of decryption, are described in the next subsections. The public key is (N, g, θ), and the private key is (the secret-sharing $\langle \nu \cdot \lambda \cdot \beta \rangle_{2t}^{\mathbb{Z}}$ of) $\nu \cdot \lambda \cdot \beta$.

3.1 Generating random 'primes'

To generate a additive sharing of a random number q (and similarly of a random number p), each party i does the following.

Generation of random number q. Let κ denote the minimum bit-length of $q = \sum_i q_i$. Party i generates a random number q_i of κ bits.

Furthermore, the biprimality test requires $q \equiv 3 \pmod 4$, which can be achieved by letting party 1 choose q_1, such that $q_1 \equiv 3 \pmod 4$, and the other parties choose q_i, such that $q_i \equiv 0 \pmod 4$.

Notice that the integer q is not uniformly distributed, but the κ least significant bits are, and this way of generating N does not simplify factoring [3, Lemma 1].

3.2 Securely computing N, and checking for small prime divisors

Once the 'primes' p and q have been securely generated, we need to compute their product $N = p \cdot q$. For this purpose, the additive shares of p and q are reshared to a Shamir secret sharing modulo a large prime P. To assure $P > N$, we choose a prime P of at least $2(\kappa + \log_2 n)$ bits.

Secure computation of N

1. The parties generate Shamir secret sharings modulo P (with threshold t) of their p_i and q_i, and distribute the shares.
2. The parties (locally) compute shares of $p = \sum_i p_i \bmod P$, $q = \sum_i q_i \bmod P$, and $\langle N \rangle_{2t} = \langle p \rangle_t \cdot \langle q \rangle_t$.
3. At least $2t + 1$ parties open their shares of N, and they reconstruct $N = (p \cdot q) \bmod P$.

As explained in the beginning of this section, the number N needs to be checked for small prime divisors (up to some upper bound B), to speed up the bi-primality test. Given that N is public, checking for prime divisor p is simply computing $N \bmod \mathsf{p}$. Each small prime test starts by checking for divisor $\mathsf{p} = 3$, up to the largest prime, smaller than B, until a divisor is found.

3.3 Biprimality test

Since the values of p and q should remain secret, we cannot use ordinary primality tests. Instead, we test whether N is a product of two primes, by verifying $g^{\varphi(N)/4} = \pm 1 \bmod N$ for a random g with Jacobi symbol $(\frac{g}{N}) = 1$ [3, 10].

Biprimality test

1. The parties agree on a random $g \in \mathbb{Z}_N^*$, such that its Jacobi symbol $(\frac{g}{N}) = 1$.
2. The first party computes $v_1 = g^{(N-p_1-q_1+1)/4} \bmod N$, the other parties compute $v_i = g^{(p_i+q_i)/4} \bmod N$.
3. They check whether $v_1 \equiv \pm v_2 v_3 \dots v_n \pmod N$.

The Jacobi symbol can be computed efficiently. The following theorem explains why a biprime N will succeed the test.

Theorem 2. *Let integers p and q be $3 \pmod 4$, and $N = p \cdot q$. Let $g \in \mathbb{Z}_N^*$, such that the Jacobi symbol $(\frac{g}{N}) = 1$. If N is biprime, then $g^{\varphi(N)/4} = \pm 1 \bmod N$.*

Proof. Because p and q are $3 \pmod 4$, the integers $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are odd. Furthermore, $\varphi(N)/4 = \frac{p-1}{2} \cdot \frac{q-1}{2}$, so if p is a prime, $g^{\varphi(N)/4} = (\frac{g}{p})^{\frac{q-1}{2}} = (\frac{g}{p}) \bmod p$, and similarly $g^{\varphi(N)/4} = (\frac{g}{q}) \bmod q$, if q is a prime. If N is biprime then $(\frac{g}{p}) \cdot (\frac{g}{q}) = (\frac{g}{N}) = 1$, so we have $(\frac{g}{p}) = (\frac{g}{q}) = \pm 1$, and therefore $g^{\varphi(N)/4} = \pm 1 \bmod N$.

If N is not biprime, the test will fail with probability at least $\frac{1}{2}$ [3], so the test needs to be repeated a few times. Similar to Fermat's primality test, with exponential small probability, N has a specific form and sneaks through the above test. An additional test, which we decided not to implement, is required to repair this small flaw [3, Subsection 3.1].

How often do we need to construct an N before we find a biprime number? The probability that a random integer x is prime, is roughly $\frac{1}{\log_2 x}$, so without additional measures it will take κ^2 tries on average! Fortunately, if we first check for small divisors, this number quickly decreases. For example, if we check p and q for prime divisors up to $B = 8103$, we need 'only' 484 probes on average (with $\kappa = 512$) [3].

3.4 Securely generating the private key

When the modulus N has been generated and checked, the parties need to generate a secret-sharing $\langle \nu \cdot \lambda \cdot \beta \rangle_{2t}^{\mathbb{Z}}$ (corresponding to a polynomial $h(x)$) over the integers of $\lambda \cdot \beta$, the decryption key multiplicatively hidden by a random number $\beta \in \mathbb{Z}_N^*$. An additive sharing of λ is easily obtained, as $\lambda = N - p - q + 1$. If each party i generates a random $\beta_i \in \mathbb{Z}_N$, we also have an additive sharing of $\beta = \sum_i \beta_i$.

Both β and λ need to be secret-shared over the integers to avoid carry-overs modulo N in the exponent while computing the c_i (see Subsection 3.6). Each party i can generate sharings of λ_i and β_i, and the secret sharing of $\lambda \cdot \beta$ is then easily constructed by local computations. The absolute value of each share

$h(i)$ will be upper bounded by $(2^\sigma \nu^2 n^{t+1} \frac{t+1}{n-1})^2 \lambda\beta$, requiring only slightly more bits than $\varphi(N^2) = \lambda N$, which is the share size in case of a trusted dealer (see Subsection 1.1).

To be precise, we summarize the steps in more detail:

Private key shares generation

1. Given λ_i, each party i generates a random polynomial f_i^λ with large integer coefficients (see Section 2) and $f_i^\lambda(0) = \nu\lambda_i$, and sends share $f_i^\lambda(j)$ to party j, $1 \leq j \leq n$.
2. The parties compute a secret sharing over the integers of λ: $\langle\lambda\rangle_t^{\mathbb{Z}} = \sum_i \langle\lambda_i\rangle_t^{\mathbb{Z}}$, by locally adding the shares.
3. The parties repeat both steps for the β_i, creating a secret-sharing $\langle\beta\rangle_t^{\mathbb{Z}}$.
4. The parties compute $\langle\nu \cdot \lambda \cdot \beta\rangle_{2t}^{\mathbb{Z}} = \langle\lambda\rangle_t^{\mathbb{Z}} \cdot \langle\beta\rangle_t^{\mathbb{Z}}$, by locally multiplying their shares.

The shares of $\langle\nu \cdot \lambda \cdot \beta\rangle_{2t}^{\mathbb{Z}}$ are used for computing $c_i = c^{h(i)} \bmod N^2$ during decryption (see Subsection 3.6).

3.5 Revealing the public key

Once secret sharing $\langle\nu \cdot \lambda \cdot \beta\rangle_{2t}^{\mathbb{Z}}$ has been generated, the parties need to compute and reveal $\theta = (\nu^2 \cdot \lambda \cdot \beta) \bmod N$. This can be easily done as follows:

Public key disclosure. Each party i reduces its share of $\langle\nu \cdot \lambda \cdot \beta\rangle_{2t}^{\mathbb{Z}}$ modulo N, transforming the secret sharing over the integers to a Shamir secret sharing $\langle\nu^2 \cdot \lambda \cdot \beta\rangle_{2t}$, and then the parties reveal the secret θ from $2t + 1$ shares by standard Lagrange interpolation over $\mathbb{Z}_N$.

As discussed in the beginning of this section, this method avoids the less efficient adding of $N \cdot R$, as suggested in [10].

In the unlikely case that θ is not invertible, we constructed a β that is not coprime with N (so we found a factor of N), and we need to redo our protocol from scratch.

3.6 Decryption

Once the keys have been securely generated, a ciphertext c can be decrypted as follows:

1. Each party of the decryption set T computes $c_i = c^{h(i)} \bmod N^2$, and reveals it.

2. The parties compute the coefficients ℓ_i, $i \in T$ (see Subsection 2.1) such that $\nu h(0) = \sum_{i \in T} \ell_i h(i)$, and compute $c^{\nu h(0)} = \prod_{i \in T} c_i^{\ell_i} \bmod N^2$.
3. The parties compute $m = L(c^{\nu h(0)}) \cdot \theta^{-1} \bmod N$.

The $(\cdot)^{-1}$ in the last step denotes multiplicative inverse modulo N. One could use vector addition chains [4] to speed up step 2. Alternatively, each party i can compute $c_i = c^{\ell_i h(i)} \bmod N^2$.

Since h is a polynomial of degree $2t$, it takes at least $2t+1$ parties to perform a decryption. However, this threshold can be lowered to t by resharing the $h(i)$ [1].

3.7 Security

We give an informal intuition on the security of our key generation protocol. By revealing θ, we might leak information on the decryption key. In fact, θ is (more or less) the remainder of the decryption key after division by N. Intuitively, this doesn't help decrypting, since $c^N = r^{N^2} \bmod N^2$, so the random factor doesn't disappear. The decryption key $\lambda \cdot \beta$ has size N^2, so the quotient has roughly size N.

Nishide and Sakurai [10, Appendix A] formally prove the security of their construction of θ. Our design is at least as secure, since we use Shamir secret-sharing modulo N, and only reveal the modular remainders. Even if t parties collude, θ does not leak information on λ, because for any $\lambda \in \mathbb{Z}_N^*$, a $\beta \in \mathbb{Z}_N^*$ can be found such that their product is $\theta \cdot \nu^{-2}$ modulo N.

3.8 Optimising efficiency

While implementing the distributed key generation protocol, we decided to slightly modify the approach of Nishide and Sakurai for efficiency reasons. Apart from the most important choice to take a semi-honest security model instead of a malicious model, we describe the main differences.

To reduce the number of iterations in the biprimality test, one should first check N for small prime divisors. Nishide and Sakurai decided to perform this check *before* securely computing the product N, so using the secret-sharings of p and q. To check whether q is divisible by p, the idea is to generate a random secret sharing $\langle r \rangle_t$ over $\mathbb{Z}_\mathsf{p}$, compute $\langle r \cdot q \rangle_{2t} = \langle r \rangle_t \cdot \langle q \rangle_t$, reveal $(r \cdot q) \bmod \mathsf{p}$, and check whether it is zero (see Appendix for the protocol).

The advantage is that each time one of the factors turned out not to be prime, one only needs to regenerate a secret-sharing of p, or q, and it's not necessary to securely multiply two large numbers: $N = p \cdot q$. However, it also introduces quite some overhead:

- The small prime test is probabilistic ($r \bmod \mathsf{p}$ might be zero), which means that it requires quite some iterations to be sure of the outcome.

– For performing the small prime test, (secret-sharings of) random numbers modulo a small prime need to be generated. In order to generate a uniformly distributed random number, which doesn't consist of a sequence of uniformly random bits, this generation needs to be repeated a couple of times (until $r < \mathbf{p}$).
– A lot of Shamir secret-sharings (modulo a small prime) need to be generated, which requires more random number generations, and additional communication.

During implementation, it turned out to be more efficient to check for small prime divisors *after* securely computing N. The main advantage is that N is public, so it can be directly checked for small prime divisors. This outweighed the fact that more different N needed to be computed securely.

We also optimised the final step of revealing public key θ. Given secret-sharing $\langle \nu \cdot \lambda \cdot \beta \rangle_{2t}^{\mathbb{Z}}$ over the integers, one needs to reveal $\theta = (\nu^2 \cdot \lambda \cdot \beta) \bmod N$. Nishide and Sakurai decided to first add a (secret) random multiple R of N, and then reveal the sum and reduce it modulo N. Our approach is to first transform the secret-sharing $\langle \nu \cdot \lambda \cdot \beta \rangle_{2t}^{\mathbb{Z}}$ to a Shamir secret-sharing $\langle \nu^2 \cdot \lambda \cdot \beta \rangle_{2t}$ (over $\mathbb{Z}_N^*$), by locally reducing the shares modulo N, and then revealing the shared secret θ. This not only avoids generating a secret-sharing over the integers of a large random number R, and computing with it, but also improves security, as we do no longer reveal $\nu^2 \cdot \lambda \cdot \beta + R \cdot N$, but only its modular reduction $(\nu^2 \cdot \lambda \cdot \beta) \bmod N$.

4 Performance

We implemented the secure key generation protocol on a single computer (one core, one thread) with $n = 3$ players (without actual communication), and threshold $t = 1$. The computer has a 2.4 GHz CPU and 8GB RAM, of which 220MB was actually used. The protocol was implemented in Python, using the Secrets library for generating randomness, Sympy for computing Jacobi symbols, and Gmpy2 for modular arithmetic [13]. Since the execution time is a random process, we run the entire protocol 1097 times, and plotted its performance in histograms.

4.1 Parameters

The key size parameter $\kappa = 1024$ is the bit length of the additive shares p_i, q_i $(1 \leq i \leq n)$ of the primes p and q, so the bit length of the generated RSA-modulus N is roughly $2(\kappa + \log_2 n)$. We chose $\sigma = 40$ as statistical security parameter for secret-sharing over the integers. The generated products N were biprime with probability at least $1 - 2^{-100}$, which required on average 1692 iterations of the biprimality test, as depicted in Figure 1.

4.2 Good choice of B

The key generation algorithm contains two primality tests. Firstly, the small prime test determines whether the resulting modulus N is divisible by primes

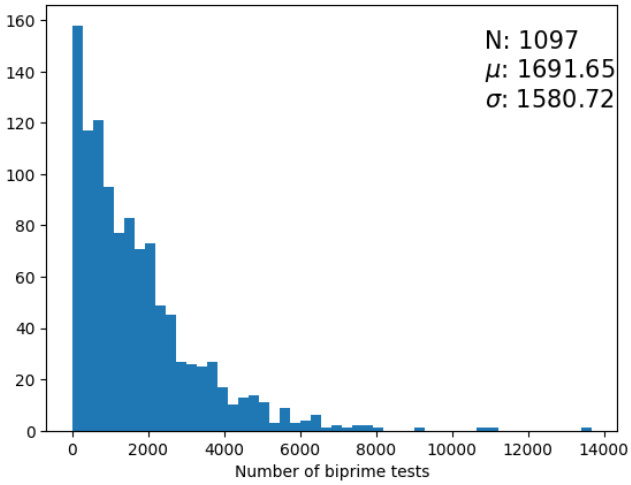

Fig. 1. Number of biprimality tests

smaller than upper bound B. Secondly, if the small prime tests succeed (i.e. N is not divisible by primes smaller than B) the biprime test determines whether N is the product of two primes.

To theoretically optimize the performance, we should minimize these costs over B, but in practice this optimization depends on the implementation, the communication network, and the hard- and software used. Boneh and Franklin came to $B = 8103$ [3], although their aim was an N of only 1024 bits. Using experimental results, we chose $B = 20,000$ for our setting. There are 2261 primes between 3 and $20,000$. Figure 2 shows the total time spent on both types of prime tests for this choice.

The number of biprime tests over the 1097 runs is shown in Figure 1, and the number of failed small prime tests in Figure 3 (the number of succeeded small prime tests equals the number of biprime tests). On average $130,137$ times a small prime divisor of N was found.

4.3 Execution time and communication complexity

The average execution time was 66 seconds for the entire key distribution protocol, which is reasonable for most applications that require only one key generation phase. Figure 4 depicts the distribution of the run times.

Although we did not actually implement communication, we can determine the communication complexity of the key generation protocol, which is dominated by the generation of a suitable N. Each time a fresh N has to be generated, the parties need to generate Shamir secret sharings of new p and q. On average, this occurs 1692 times due to a failed biprimality test, and $130,137$ times due to a failed small prime test. The field size of the Shamir secret sharings is

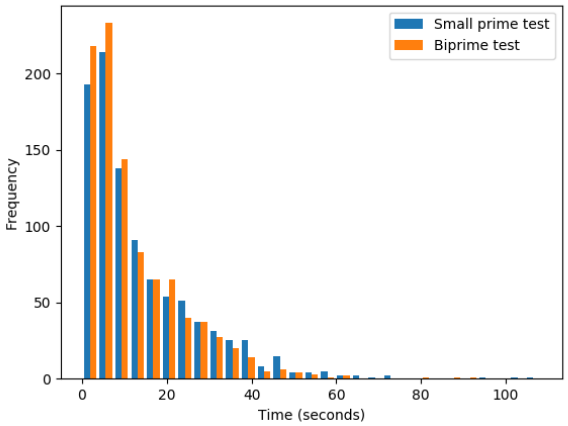

Fig. 2. Execution times of primality tests

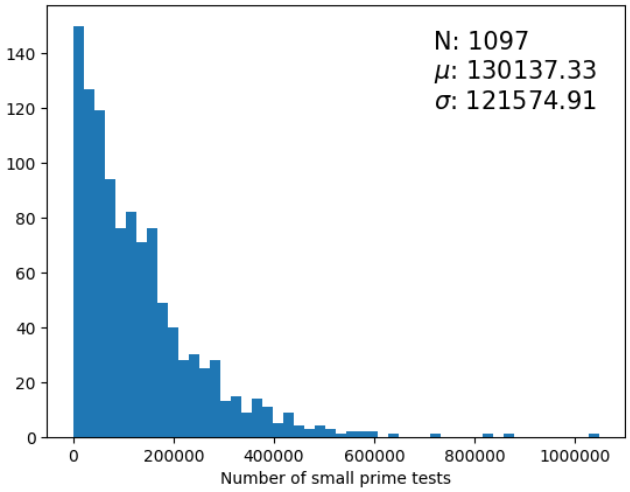

Fig. 3. Number of failed small prime tests

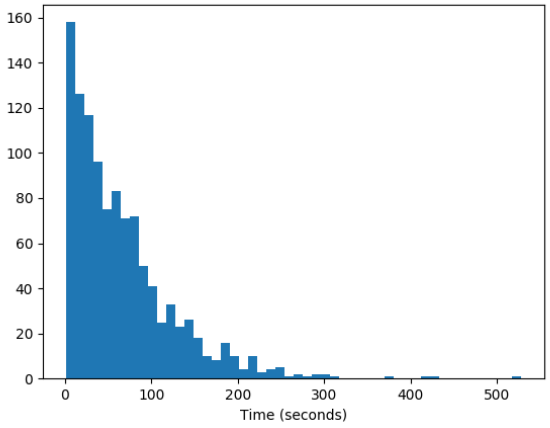

Fig. 4. Run times

$2(\kappa + \log_2 n)$ bits. Therefore, the total communication complexity is linear in the number of players n, with roughly $(1962 + 130,137) \cdot 2048 = 270,538,752$ bits, or 32.3 MB per player (for $\kappa = 1024$).

5 Conclusion

We described a way of securely generating the keys of Paillier with threshold decryption. The solution is secure in the semi-honest model, and requires no trusted dealer. Because of the use of secret sharing over the integers, the solution is statistically secure. As long as not more than t players collude, the security is guaranteed. We need $2t + 1$ decryption shares while decrypting a cipher text, but this can set to t by resharing once.

We closely followed the ideas of [10] and [3], and further optimised its efficiency. Although secure key generation takes considerable more effort without a trusted dealer, the effort for decrypting is comparable to the setting with a trusted dealer.

Acknowledgements

The research activities that have led to this paper were funded by the Shared Research Program Cyber Security: a research collaboration between TNO, ABN AMRO, Rabobank, ING and Achmea.

References

1. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC. pp. 1–10 (1988)
2. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. Cryptology ePrint Archive, Report 2018/1188 (2018)
3. Boneh, D., Franklin, M.K.: Efficient generation of shared RSA keys. In: CRYPTO. pp. 425–439 (1997)
4. Bos, J.: Practical Privacy. Ph.D. thesis, Eindhoven University of Technology (1992)
5. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Eurocrypt, International Conference on the Theory and Application of Cryptographic Techniques. Lecture Notes on Computer Science, vol. 2045, pp. 280–299. Springer (2001)
6. Damgård, I., Mikkelsen, G.L.: Efficient, robust and constant-round distributed RSA key generation. In: Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings. pp. 183–200 (2010). https://doi.org/10.1007/978-3-642-11799-2_12, https://doi.org/10.1007/978-3-642-11799-2_12
7. Fouque, P., Poupard, G., Stern, J.: Sharing decryption in the context of voting or lotteries. In: Frankel, Y. (ed.) Financial Cryptography, 4th International Conference, FC 2000 Anguilla, British West Indies, February 20-24, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1962, pp. 90–104. Springer (2000). https://doi.org/10.1007/3-540-45472-1_7, https://doi.org/10.1007/3-540-45472-1_7
8. Gilboa, N.: Two party RSA key generation. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 116–129 (1999). https://doi.org/10.1007/3-540-48405-1_8, https://doi.org/10.1007/3-540-48405-1_8
9. Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T.: Efficient RSA key generation and threshold paillier in the two-party setting. In: Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings. pp. 313–331 (2012). https://doi.org/10.1007/978-3-642-27954-6_20, https://doi.org/10.1007/978-3-642-27954-6_20
10. Nishide, T., Sakurai, K.: Distributed Paillier cryptosystem without trusted dealer. In: Information Security Applications - 11th International Workshop WISA. pp. 44–60 (August 2010). https://doi.org/10.1007/978-3-642-17955-6_4, https://doi.org/10.1007/978-3-642-17955-6_4
11. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Eurocrypt. pp. 223–238. Springer (1999)
12. Rabin, T.: A simplified approach to threshold and proactive rsa. In: CRYPTO. pp. 89–104 (1998)
13. Spini, G., Attema, T.: Implementation source code. `https://github.com/TNO/Distributed-Paillier-Cryptosystem` (2019)
14. Wesolowski, B.: Efficient verifiable delay functions. In: Eurocrypt. Lecture Notes in Computer Science (2019)

Appendix

Checking for small prime divisors of secret-shared values

In order to reduce the number of iterations of the biprimality test, we need to check q for small divisors, for example all primes up to $B = 8103$ [3]. The following protocol checks whether small prime p is a divisor of q:

1. Each party generates a random number $r_i \in \mathbb{Z}_{\mathsf{p}}$, so they additively share $r = \sum_i r_i$.
2. Each party i generates Shamir secret sharings modulo p (with threshold t) for both q_i and r_i.
3. They (locally) compute a Shamir secret sharing modulo p for $r \cdot q$.
4. They open at least $2t+1$ shares to reconstruct $(rq) \bmod \mathsf{p}$, and check whether it is zero.

If $(rq) \bmod \mathsf{p} = 0$, it might be the case that p is a divisor of r, so we have to repeat the protocol a few times to be sufficiently sure (reusing the shares of q).

Nishide and Sakurai [10] propose a similar protocol, but it uses secret-sharing over the integers to ensure robustness. Since we deploy a semi-honest model, the above, more efficient, protocol could be used.