On the Security of Distributed Multiprime RSA

Ivan Damgård[1], Gert Læssøe Mikkelsen[2(✉)], and Tue Skeltved[3]

[1] Department of Computer Science, Aarhus University, Aarhus, Denmark
[2] The Alexandra Institute, Aarhus, Denmark
gert.l.mikkelsen@alexandra.dk
[3] Signaturgruppen A/S, Aarhus, Denmark

Abstract. Threshold RSA encryption and signing is a very useful tool to increase the security of the secret keys used. Key generation is, however, either done in a non-threshold way, or computationally inefficient protocols are used. This is not a big problem in a setup where one organization has a few high profile keys to secure, however, this does not scale well to systems with a lot of secret keys, like eID schemes where there exist one key pair per user, especially not if the we want the users' personal devices like smart phones to participate in the threshold setup. In this paper we present novel approaches to distributed RSA key generation which are efficient enough to let smart phones participate. This is done by generating keys consisting of more than two primes instead of generating standard RSA keys.

We present a 2-party protocol based on the ideas of [BH98] which produces a 3-prime modulo. We demonstrate that the protocol is efficient enough to be used in practical scenarios even from a mobile device which has not been demonstrated before. Then we show the first 2-party distributed multiprime RSA key generation protocol that are as efficient as standard centralized key generation, even if security against malicious adversaries is desired. Further, we show that RSA keys based on moduli with more than two prime factors and where part of the factorization is leaked to the adversary are useful in practice by showing that commonly used schemes such as PSS-RSA and OAEP-RSA is secure even if the adversary knows a partial factorization of the multiprime moduli. From all other parties the generated keys cannot be distinguished from standard RSA keys, which is very important as this make these protocols compatible with existing infrastructure and standards.

1 Introduction

Despite the introduction of elliptic curve cryptography and more recently lattice based cryptography, RSA remains one of the the most used public-key schemes. A very large number of e-commerce and net-banking transactions are protected using RSA. In many applications, a user's secret key resides on his own machine,

Gert Læssøe Mikkelsen—Supported by the Danish Council of technology and Innovation.

J. Lee and J. Kim (Eds.): ICISC 2014, LNCS 8949, pp. 18–33, 2015.
DOI: 10.1007/978-3-319-15943-0_2

and due to the fact that the security on private PC's is often very poor, keys as well as passwords can be stolen. It is well known that in net-banking, for instance, this has lead to a significant loss of of money.

Countermeasures proposed against this include using extra, special-purpose hardware which is often expensive, or storing secret keys on a central server while implementing some form of conventional access control to the secret keys. While a central server may certainly have better security, this also creates a single point of attack.

One approach that can lead to better solutions is to do *threshold RSA*, i.e., we split the secret key in two or more shares stored in different entities such that signing or decryption requires participation of at least some number of share-holders. The adversary now must break into more than one entity to steal the key. Whether this actually improves security in a real application depends, of course, on the implementation, but the threshold approach certainly creates new possibilities for designing a secure system. For instance, if the design involves a handheld mobile device, it may not be necessary to use a special-purpose high-security device if it will not be storing the entire key. A mobile phone, for instance, may be sufficient.

Threshold RSA is a well studied problem from a theoretical perspective, see for instance [GRJK07, DK01]. In this paper, we focus on the case of two share-holders. For concreteness the reader may think of a mobile device holding one share while the other is held by a server, run by the user himself, or by some organization. This case was studied in [DM09] where a formal model was given for a more realistic scenario where the human user is explicitly modeled as a player. This allows us to take passwords and login credentials into account when proving security. In [DM09] a protocol was given that is secure if the adversary can, at any one time, only corrupt the mobile or the server, but not both. However, this work, like most work on threshold RSA, does not directly consider the problem of generating keys in a distributed fashion, but assumes that shares of the key have been distributed by a trusted party.

To avoid a single point of failure, it is of course desirable to implement the trusted party using a secure protocol executed by the share-holders. Design of such a distributed key generation protocol has been studied in a long line of research. The first reasonably efficient solution to this problem is due to Boneh and Franklin [BF97, BF01]. Except for the work by Algesheimer et al. [ACS02] (which has prohibitively large round complexity), all other works (e.g. [BH98, FMY98, Gil99, DM10, HMRT12]) in this area are more or less variations of the original ideas from [BF97]. In short the idea is to generate a candidate RSA modulus $N = pq$, where p and q are random numbers that are additively shared among the players. They then execute a *distributed biprimality* test to check whether N is the product of two primes. This can be done efficiently because the players have shares of p and q. If N is indeed the product of two primes, then it is output, otherwise the protocol is restarted.

The main problem with this approach is that a candidate N can only be used if both p and q happen to be prime at the same time. This means that the expected number of attempts needed is quadratic in k, were k is the desired

length of the modulus, whereas standard centralized key generation is linear in k. This makes the distributed protocol several orders of magnitude slower than standard key generation for realistic values of k.

It was noted already in [BF97] that one can avoid this quadratic slowdown if one is willing to have RSA moduli with several prime factors and leak part of the factorization to the adversary. In particular, [BH98] presents a 3-party protocol secure against one corrupted player that generates a modulus with 3 prime factors. This protocol only requires that the parties have to find and generate a single additive shared prime, but on the other hand the adversary may learn one of the primes of the final modulus.

It is not clear that using such a key in practice is secure. For instance, if the adversary sees public key (N, e) and ciphertext $c = x^e \bmod N$, he can compute a large amount of partial information about x. Say he knows one prime factor p, then he can compute $c^{e^{-1} \bmod (p-1)} \bmod p = x \bmod p$. To the best of our knowledge there has been no previous study of security of RSA based schemes in this scenario, which is perhaps the reason why this idea for key generation has received very little attention so far. In this article we demonstrate that such keys are in fact secure when used with appropriate padding schemes such as PSS-RSA and OAEP-RSA, which are the most widely used padding schemes, and which are an essential part of a secure scheme based on RSA.

1.1 Our Contributions

In this paper, we study the use of multiprime RSA moduli in distributed key generation and for encryption and signatures where the adversary may know part of the factorization. More precisely he may learn (or even get to choose) all but 2 of the prime factors. We concentrate on the 2-party case as this in many cases are a more realistic setup e.g., consiting of a user using a mobile device and a larger organization operating the server side. The 2-party case means that for a malicious adversary, we can only get security with abort: if one player stops prematurely, we cannot complete the protocol.

Our contributions are two-fold; We present two new 2-party distributed RSA key generation protocols and show that multiprime RSA keys used in combination with PSS-RSA or OAEP-RSA is secure even if the adversary knows part of the factorization. It is important to note, that this generalizes to all such keys, not just the ones produced by the protocols presented in this article.

The Protocols. We introduce two 2-party protocols. One is based on ideas from [BH98] which is a 3-party protocol secure against one corrupted player, where two parties generate a prime each locally (say p and q), whereas a random candidate number t is generated in secret-shared form. The players then compute $N = pqt$ securely and do a distributed test to check if N is the product of 3 primes. Since [BH98] assume honest majority, the secure computation needed could be done efficiently based on secret sharing. Here, we adapt the protocol to the two-party case using a homomorphic cryptosystem for two-party distributed computations

and we also adapt the primality tests from [BH98] to the 2-party scenario. In our particular implementation the Paillier cryptosystem [Pai99] is used and we demonstrate that this 2-party protocol is efficient enough to be useful even from mobile devices - a result that has not been demonstrated before. As in [BH98], we obtain passive security.

We then introduce a new approach where on the one hand we generate a larger modulus than before, namely with 4 prime factors, but on the other hand the protocol is much more efficient and can easily be made actively secure. The idea is to simply let each party do a normal RSA key generation locally where the only condition is that they agree on the public exponent e. They then exchange the public keys $(N_1, e), (N_2, e)$ and the final public key is $(N_1 N_2, e)$. It follows from the Chinese remainder theorem that the parties can use their locally generated secret exponents to do distributed signing or decryption. This can be made actively secure with very little overhead as long as we enforce that each player must know the factorization of his number, see more details within. It takes only seconds to generate a secure 2048-bit modulus, and thus only seconds to complete the protocol. Note that this system is very easy to build from existing RSA soft- or hardware, since standard key generation and encryption/decryption operations is essentially all that is required. Note also that any two (or even more) users who have the same public exponent can combine their keys in this way, even if they did not anticipate this at key generation time. This is the first 2-party protocol for multiprime RSA key generation that achieves active security while being as efficient as standard RSA key generation.

It is important to understand that this idea is very different from the trivial approach to "threshold" RSA signatures where we just let each shareholder sign with his own key. This would force parties who use or certify the public key to be aware that a certain person is actually "composed" of several entities, thus making practical implementation much more cumbersome. In our approach, we maintain that the public key is simply a standard RSA key (albeit with a longer modulus) and the fact that the key is shared is transparent to other users.

Security of PSS-RSA and OAEP-RSA in the Multiprime Setup. In practice RSA is never used without a secure padding scheme, such as PSS-RSA for signatures or OAEP-RSA for encryption. As show by Bleichenbacher's attack [Ble98] on the PCKS#1 v1.5 standard, provable security of RSA in combination with the padding scheme is very important. We show that both the PSS-RSA and OAEP-RSA padding schemes used with a multi-prime RSA key remain secure even if the adversary knows all but two of the prime factors, and therefore cannot completely factor the modulus but can extract some partial information of the preimage. The security level then corresponds to the security of the RSA modulus formed by the two unknown primes. We can therefore conclude that the keys output by the two key generation protocols presented in this article and similar protocols are indeed useful in application scenarios used today.

2 Preliminaries

Below in Assumption 1 we follow the standard definition of the security of "plain" RSA, by assuming that no efficient algorithm can invert the RSA function without knowledge of the private key.

Definition 1. *Let the algorithm $\mathcal{A}_{RSA}$ be specified as: Given $\{N, e, y\}$ s.t. N is the product of two k-bit primes, $gcd(\varphi(N), e) = 1$, and $y \in \mathbb{Z}_N$, then $\mathcal{A}_{RSA}$ outputs x s.t. $y \equiv x^e \pmod N$.*

Assumption 1 (Hardness of RSA). *We assume that no probabilistic polynomial Turing machine (PPT) exists that implements $\mathcal{A}_{RSA}$ for random input, with nonnegligible success rate.*

We now specify an adversary for breaking multiprime RSA (M-RSA), the RSA problem with a modulus consisting of more than two primes, where the adversary have chosen all but two of the prime factors of the modulus. This adversary actually consists of two algorithms, one for generating α the part of the M-RSA modulus known to the adversary and one inverting the RSA function using this modulus. We will see in Lemma 1 that the hardness of RSA implies hardness of M-RSA.

Definition 2. *Let algorithm $\mathcal{A}_{M\text{-}RSA\text{-}Gen}$ be specified as: Given N s.t. N is the product of two k-bit primes, then $\mathcal{A}_{M\text{-}RSA\text{-}Gen}$ outputs $\{\alpha, M, \varphi(\alpha), state\}$, where α is an arbitrary k-bit positive integer, $M = N\alpha$, and state is an arbitrary string.*

Let $\mathcal{A}_{M\text{-}RSA}$ be an algorithm taking as input $\{M, e, y, state\}$ s.t. M and state is the output of $\mathcal{A}_{M\text{-}RSA\text{-}Gen}$, $gcd(\varphi(M), e) = 1$ and $y \in \mathbb{Z}_M$. The output of $\mathcal{A}_{M\text{-}RSA}$ is x s.t. $y \equiv x^e \pmod M$.

3 Protocol for Two Players with a Three-Prime Modulus

In this section we present a two-party protocol generating a three-prime RSA modulus. To enable distributed computations between two players the Paillier Cryptosystem [Pai99] is used. The protocol is designed and optimized to run between a mobile device and a server and in particular only the server has to generate a Paillier key pair. The protocol is based on the ideas from [BH98].

To test whether a modulus N is well formed, the parties need an additive sharing of the following two numbers: $\Phi(N) = (p-1)(q-1)(r-1)$ and $\Psi(N) = (p+1)(q+1)(r+1)$, where $N = pqr$, p and q are primes chosen by the two parties respectively and r is a number that is additively shared between them as $r = r_1 + r_2$. The tests ensure that if a N is output, then r is prime except with negligible probability.

In the following, E_k, D_k denotes the Paillier encryption/decryption function with modulus k. Recall that the Paillier scheme uses computation modulo k^2 for the ciphertexts, and is additively homomorphic modulo k. This modulus must be large enough to accommodate without overflow the product of two primes plus room for some added randomness. In the following we denote the two parties S and M for server and mobile device. The Paillier keys are generated by S. We first give a short overview of the main steps in the protocol:

3.1 Protocol Steps

1. **Generate possible candidate N.** The parties jointly generates the public RSA moduli $N = p \cdot q \cdot (r_1 + r_2)$, using primes p, q and random integers r_1, r_2 as input.
2. **Fermat test.** By utilizing Fermat's little theorem, the two parties test if $g^{\phi_a + \phi_b} = 1 \pmod N$, for a random element $g \in_R Z_N^*$. Here $\Phi(N) = \phi_a + \phi_b$ denotes the additive shares of $\Phi(N)$ generated by the two parties during the previous step.
3. **Twisted group Fermat test.** The parties perform a Fermat test in the Twisted group T_N, picking a random element $g \in_R T_N$, and testing if $g^{\psi_a + \psi_b} = 1 \pmod N$. Here $\Psi(N) = \psi_a + \psi_b$ denotes the additive shares of $\Psi(N)$ generated by the two parties during the first protocol step. For more details on the Twisted Group, see [DMS14].
4. **Check that $N = p^a q^b r^c$, for three distinct primes p, q and r.**
5. **Zero knowledge test that $gcd(N, p + q) = 1$.**
6. **Generate the private key distributed as additive shares.**

3.2 The Protocol

We now give a more detailed account of the first part of the protocol:

i. S generates a random $(n-1)$-bit integer r_2 and sends the encryption $E_k(r_2)$ to M.
ii. M generates a random n-bit prime p, where $p \equiv 3 \pmod 4$ and a random $(n-1)$-bit integer r_1 and sends $E_k(r \cdot p) = (E_k(r_2) \cdot E_k(r_1))^p \bmod k^2$ to S. Note that the randomness in $E_k(r_2)$ will ensure that $E_k(r \cdot p)$ is a random encryption containing $r \cdot p$.
iii. S decrypts $r \cdot p$, optionally runs a trial division test on $r \cdot p$ using small primes and generates a random n-bit prime q, where $q \equiv 3 \pmod 4$. If the division test fails then S aborts.
iv. $N = r \cdot p \cdot q$ is sent back to M.

The two parties now have a candidate N. To test if N can be used, they need additive shares of $\Phi(N)$ and $\Psi(N)$. As $\Phi(N) = (p-1)(q-1)(r_1 + r_2 - 1)$ they can exploit the fact that:

$$\Phi(N) = N - pr - qr - pq + p + q + r - 1$$

Here we note that $N = p \cdot q \cdot (r_1 + r_2)$, so M has $qr = N/p$ and S has $pr = N/q$. They just need an additive share of pq in order to have all the additive components which they share as follows:

i. S sends $E_k(q)$ to M
ii. M then creates a blinded encryption of $p \cdot q$ by: $E_k(pq + t) = E_k(q)^p \cdot E_k(t) \pmod{k^2}$, where t is a random integer with a maximum bit-length such that $p \cdot q + t$ cannot overflow modulo k
iii. $E_k(pq + t)$ is sent to S

iv. M calculates $qr = N/p$ and then $\phi_a = t - 1 - qr + p + r_1$
v. S calculates $pr = N/q$ and then $\phi_b = N + (-1) \cdot (pq + t) - pr + q + r_2$

Now, $\Phi(N) = \phi_a + \phi_b$. And similar for the sharing of $\Psi(N)$

Due to space limitations, further details of the protocol for doing steps 2–6 above can be found in the full version of this work [DMS14], these steps follow the flow of [BH98], transfered to the two-party setting.

3.3 Passive Security of the Protocol

In the full version of this paper [DMS14], we show that the error probability of our primality tests are as good as the similar tests from [BH98] which implies that the desired probability of N being the product of three large primes can be achieved by repeating the primality tests a certain number of times. Thus correctness is ensured s.t. when the protocol is completed, the two parties have produced a modulus of three large primes except with negligible probability. In [DMS14] we prove that the protocol achieves passive security when the underlying homomorphic cryptosystem is secure.

4 Four-Prime Distributed RSA

This section introduces a new approach for generating a distributed RSA key between two parties constructing a public modulus with four prime factors formed as a product of standard RSA moduli. We use as subprotocol a protocol for proving knowledge of discrete logarithm modulo a composite, this protocol is due to Girault [Gir91] and is essentially the Schnorr protocol [Sch91] done modulo a composite. We note that this protocol can be made non-interactive and zero-knowledge in the random oracle model We denote this protocol the PK-CDL protocol (Proof of Knowledge of Composite DL) in the following.

Key Generation

1. The two parties S and M agree on a public exponent e. They then generate a standard RSA key pair each, denoted by $((N_S, e), (N_S, d_S))$ and $((N_M, e), (N_M, d_M))$, respectively.
2. They exchange the public keys, set $N = N_S N_M$ and the joint public key is defined to be (N, e). S and M store d_S and d_M as their shares of the secret key. N_S, N_M are stored for practical reasons, but are not considered secret.
3. S convinces M that (N_S, e) is well formed as follows:
   (a) M chooses a random $x \in Z^*_{N_S}$ and sends x and $y = x^e$ to S.
   (b) Using the PK-CDL protocol S proofs knowledge of d s.t., $x \equiv y^d \bmod N_S$.
4. The above step is repeated with the roles of M and S interchanged. If any proof fails, the parties abort, otherwise they output the key material defined above.

This idea clearly extends to more than two parties, of course at the expense of having a larger modulus. If only passive security is desired, the last two steps can be omitted. One applies the public key as usual by raising to power e modulo N. In a standard threshold RSA set-up, one would usually secret-share the private exponent additively, we present a protocol for this in [DMS14]. However, in our case it is easier to use the local private exponents that are available anyway. Therefore applying the secret key is done using Chinese remaindering as follows:

Distributed Decryption/Signing

1. On input $y \in Z_N$ to which the secret key should be applied, S and M compute $x_S = y^{d_S} \bmod N_S$ respectively $x_M = y^{d_M} \bmod N_M$ and exchange these values.
2. Both players use the Chinese Remainder Theorem to compute $x \in Z_N$ such that $x \bmod N_S = x_S$, $x \bmod N_M = x_M$. They check that $x^e \bmod N = y$ and output x if this is the case. Otherwise, they abort.

4.1 Security of Four-Prime Distributed RSA

These protocols are secure for sequential composition, even if one parties are malicious. This is proven via a simulation proof and below we outline the functionality that we prove is implemented by the key generation protocol. We emphasize that we only claim security for sequential composition so that the simulator is allowed to rewind, however, using standard techniques the protocol can be made secure for general composition.

Key Generation Functionality

1. Receive public exponent e as input from the honest party (or parties).
2. If both parties are honest, generate all key material honestly and send it to the parties.
3. If S is corrupt, generate N_M honestly and send N_M, e to the adversary. Receive from the adversary either N_S and the prime factors $p_1, ..., p_t$ in N_S, where e is relatively prime to $\phi(N)$; or "abort". In the first case, output N_S, N_M, d_M to M. In the second case output "abort". If M is corrupt, do the same with S and M interchanged.

Theorem 1. *The* Key Generation Protocol *for* Four-prime Distributed RSA *securely realizes the* Key Generation Functionality *presented above, for sequential composition (allowing rewinding).*

Proof. We assume that S is corrupt. A simulator for the above key generation protocol would then receive N_M, e from the functionality and then execute the protocol with the corrupted S (the adversary). It can simulate M's part of the protocol using N_M, e by simulating the PK-CDL protocol, which is zero-knowledge. When the corrupt S executes the PK-CDL protocol to prove

knowledge of d_S, the simulator extracts the witness d_S. If N_S, e was well formed, then the simulator with the knowledge of both e and d_S can easily factor N_S and input these factors to the functionality. In case N_S, e is malformed i.e., $gcd(e, \phi(N_S)) = \alpha \neq 1$, then no inverse d_S of e exists modulo $\phi(N_S)$. Therefore PK-CDL would fail, as the corrupt S cannot know d_S. In this case the simulator input "abort" to the functionality.

As for the protocol for distributed decryption/signing, we can think of it as being executed in a model where the key material has been generated by the functionality we just described. Therefore we know that e is relatively prime to $\phi(N)$ and hence there is a well defined decryption exponent d. First note that if both parties are honest, the result x always equals $y^d \bmod N$. This is because $x \bmod N_S = x_S = y^{d_S} \bmod N_S$ and hence $x^e \bmod N_S = y \bmod N_S$. Similarly we also see that $x^e \bmod N_M = y \bmod N_M$ and hence by the Chinese remainder theorem we have $x^e \bmod N = y$. If one party is corrupt the protocol trivially outputs the correct result or abort, and furthermore, if S is corrupt, it can simulate M's contribution when given the output $x = y^d \bmod N$, simply by computing $x \bmod N_M$.

4.2 Efficiency of Four-Prime Distributed RSA

We now consider the efficiency of this set-up compared to standard RSA with a 2-prime modulus. The Key Generation takes time equivalent to a local key generation plus the time needed for the PK-CDL protocols. The PK-CDL protocol takes time essentially equivalent to 1 exponentiation for both parties. In practice it will actually be less because we can choose e significantly smaller than n_S, and S can optimize her computations using Chinese Remaindering. Note also that the last two steps of the protocol (where S, resp. M plays the role of the prover) can be done in parallel. The local key generation requires a few exponentiations due to the primality tests needed. Therefore we can expect that the full key generation takes time about twice that of standard local key generation.

The time for applying the secret key is clearly equivalent to applying a secret key for a standard modulus, since this is exactly what both parties are doing. The time to apply the public key is larger than in the standard case because the public modulus is twice as long. However, this makes little difference in practice since first, we can use a value of e that is much smaller than the modulus ($e = 2^{16} + 1$ is a standard choice); and second if N_S and N_M are known (which would not hurt security) exponentiation modulo N can be done modulo N_S and N_M using Chinese Remaindering.

5 Implementation Results

In the three-prime protocol one prime has to be found by random trial-and-error computation and a distributed primality test. By the Prime Number Theorem

(see [DMS14]) the number of rounds needed on average as well as execution time grow with the target modulus size. In this section we show implementational results from the three-prime protocol presented in this article demonstrating that the protocol is efficient enough to be useful even from a mobile device. Further it demonstrates that computing the needed number of random primes for this and similar protocols is a dominant factor in the overall processing time and thus provides a natural lower bound for this type of protocol.

The following two setups have been used to run the three-prime protocol between a smartphone and a laptop (server):

1. i-7 Q 820 4 x 1.73 GHz, 8 GB RAM and Samsung Galaxy s-II, 2 x 1.2 GHz
2. i7-4712HQ @ 2.3 GHz, 16 GB RAM and HTC ONE Quad-core 1.7 GHz

We present the average running time measured for the protocol to complete between the mobile device and the laptop to illustrate that the protocol is indeed efficient enough to be used in practical scenarios. Then we present results of the protocol being run entirely on the laptop to illustrate that the protocol will finish in just seconds if run between two desktop computers for a 2000 bit key. Further we present the running time of a single thread computing the expected number of random primes needed to complete the protocol. Note that the implementation utilizes all cores available on the devices, so the time needed to generate all the primes for a protocol round on a single thread takes about a quarter when running on a quad core device. Little data is sent between the parties (expected

Table 1. Results from the implementation of the three-prime protocol

Three-prime protocol between phone (no precomputation) and laptop			
Modulus size	Rounds$^{\alpha}$	First setup$^{\beta}$	Second setup$^{\beta}$
1000 bit	117	17 s	7.85 s
2000 bit	234	150 s	75 s
Laptop running both parties without precomputation			
Modulus size	Rounds$^{\alpha}$	Intel i-7 Q 820	Intel i7-4712HQ
1000 bit	117	3.7 s	1.57 s
2000 bit	234	34.4 s	13.84 s
Laptop running both parties with precomputation			
Modulus size	Rounds$^{\alpha}$	Intel i-7 Q 820	Intel i7-4712HQ
1000 bit	117	2.26 s	1 s
2000 bit	234	14.32 s	7.6 s
Single thread computational times for all primes, one protocol execution			
Modulus size	Intel i-7 Q 820	Samsung Galaxy SII 1,2Ghz	Intel i7-4712HQ
1000 bit	2.9 s	27 s	1.45 s
2000 bit	30.9 s	222 s	16.5 s

$^{\alpha}$Expected number of rounds.
$^{\beta}$Avg. time used on setup.

1 MB for the entire protocol for a 2000 bit key), and the protocol is thus very prone to optimizations using precomputation (computing a set of random primes before execution begins), crypto-hardware etc. We also present results demonstrating the effect of precomputing a large set of the random primes needed in each protocol round (Table 1).

Note that the Samsung smartphone uses 222 s to generate all the expected number of primes needed to complete the protocol for a 2000 bit key on a single thread. Running this on the two cores available on the Samsung device takes an expected 111 s to complete the prime number generation, which accounts of 74 % of the total average time for the protocol to complete. Also note that the time needed on the i7-4712HQ for the generation of a shared key will reduce the expected running time by a factor 2 if two similar computers are running as one of the parties each. The amount of data exchanged in each direction during execution of the three-prime protocol is on average less than 0,5 MB for the 1000 bit modulus and 1 MB for the 2000 bit modulus.

The four-prime protocol. In comparison to the three-prime protocol, the four-prime protocol presented in this article just needs all parties to agree on the public exponent and have each party generate a standard RSA key, which can be done in seconds (or less) - even on mobile devices.

6 Security of the M-RSA Trapdoor Permutation

In this section we will start taking a closer look at the security implications of utilizing multiprime RSA (M-RSA) moduli, generated by our protocols. First, in the lemma below, we look at the security of the general plain M-RSA function in the subsequent sections we analyze the security when M-RSA moduli are used in different specific protocols. It is important to note that the security of these protocols does not follow directly from the lemma below.

Lemma 1 (Security of Plain M-RSA). *Under Assumption 1 (Security of RSA) there does not exist a couple of PPT algorithms implementing $\mathcal{A}_{M\text{-}RSA\text{-}Gen}$ and $\mathcal{A}_{M\text{-}RSA}$ with nonnegligible probability of success.*

Proof. Assuming there exist two PPT's implementing $\mathcal{A}_{\text{M-RSA-Gen}}$ and $\mathcal{A}_{\text{M-RSA}}$, we can use these to implement $\mathcal{A}_{\text{RSA}}$ in the following way: Given a two-prime RSA public key $\langle e, N\rangle$ and y, run $\mathcal{A}_{\text{M-RSA-Gen}}$ to obtain α and $\varphi(\alpha)$. We assume $(\alpha, N) = 1$, otherwise factoring N is trivial, then we run $\mathcal{A}_{\text{M-RSA}}(N' = \alpha N, e, y, state)$. If $\mathcal{A}_{\text{M-RSA}}$ returns x' such that $y = (x')^e \bmod N'$, then our reduction returns $x = x' \bmod N$. Since $y = (x')^e \bmod N' = (x')^e \bmod \alpha N$ and $y \bmod N' = y \bmod N$, then $y = x^e \bmod N = (x' \bmod N)^e \bmod N$ Therefore the existence of $\mathcal{A}_{\text{M-RSA-Gen}}$ and $\mathcal{A}_{\text{M-RSA}}$ violates Assumption 1.

It is easy to see that the above reduction is tight, meaning if an adversary can break M-RSA in time t, then we can use this adversary to break RSA in time t plus a little overhead. We will formulate more exact security in the following way: If an algorithm A in time $t(k)$ and with probability $\epsilon(k)$ can break a scheme, for

example RSA, we say that A (t, ϵ)-breaks the scheme. If for given functions t and ϵ no algorithm that (t, ϵ)-breaks a scheme exists we call the scheme (t, ϵ)-secure. Regarding to M-RSA the time t describes the running time of both $\mathcal{A}_{\text{M-RSA-Gen}}$ and $\mathcal{A}_{\text{M-RSA}}$. We let k denote the bit-length of the primes in the modulus.

Corollary 1 (Tightness of Plain M-RSA). *If RSA is (t', ϵ')-secure then M-RSA is (t, ϵ)-secure with:*

$$t(k) \geq t'(k) - O(k^2)$$
$$\epsilon(k) \leq \epsilon'(k)$$

Proof. It is easy to see that if $\mathcal{A}_{\text{M-RSA-Gen}}$ and $\mathcal{A}_{\text{M-RSA}}$ break M-RSA then the reduction of Lemma 1 breaks RSA, and therefore $\epsilon(k) \leq \epsilon'(k)$.

The overhead of the reduction of Lemma 1 is the modulo reduction $x = x' \bmod N$, which gives the overhead of $O(k^2)$.

7 Security of Multiprime PSS-RSA Signatures

For various reasons hashing is normally applied to a message before it is signed with the RSA function, this makes RSA signatures semantic secure and in addition it enables signing of messages of arbitrary length. Hashing alone, however, does not give a tight bound on the security of the digital signature scheme, because it cannot be reduced to inverting the RSA function. The same holds for full domain hashing, where hashing is done such that it hits the complete preimage of the RSA function. To get a tighter bound Bellare and Rogaway [BR96] describes a randomized hashing and padding scheme known as PSS-RSA. [BR96] also gave a proof of a tight bound for PSS-RSA in the *Random Oracle* (RO) model.

The PSS-RSA scheme of Bellare and Rogaway has later been augmented and standardized as part of PKCS #1 v2 [RSA02]. This scheme is also known as PSS-RSA. Although the two PSS-RSA schemes have differences, reductions from forging both schemes to inverting plain RSA are analogues. From hereinafter we will concentrate on the PSS-RSA scheme by Bellare and Rogaway, whereas the results will also be valid for the PKCS version.

7.1 Signing with PSS

When signing with PSS-RSA two cryptographic hash functions $h : \{0,1\}^* \mapsto \{0,1\}^{k_1}$ and $g : \{0,1\}^{k_1} \mapsto \{0,1\}^{k-k_1-1}$ are used for hashing and padding the message m. In addition a uniform random value $r \in_{\text{R}} \{0,1\}^{k_0}$ is used. The uniform randomness of r is crucial for the security proof, even though it is sometimes omitted in real world applications, see [RSA02]. After hashing and padding m, the (private) RSA function f^{-1} is applied to the result, and the output is the signature. Let $||$ denote bit-wise concatenation.

SignPSS(m)

$$
\begin{aligned}
r &\xleftarrow{r} \{0,1\}^{k_0} \\
w &\leftarrow h(m||r) \\
y &\leftarrow 0||w||(g(w) \oplus (r||0\ldots 0)) \\
x &\leftarrow f^{-1}(y)
\end{aligned}
$$

To verify m and signature x, the RSA function with the public key is applied s.t. $y \leftarrow f(x)$ is first calculated, afterward r can be reconstructed from y and the hashing and padding of m can be verified. The following theorem, due to [BR96], states the tightness of the PSS-RSA construction, q_{hash} specifies the number of times the adversary is allowed to invoke the hash algorithm (the random oracle) and q_{sig} the number of signed messages he can see before the forgery.

Theorem 2 (From [BR96], Security of PSS-RSA). *In the random oracle (RO) model: If RSA is (t', ϵ')-secure, then for any q_{sig}, q_{hash} the signature scheme* PSS-RSA$[k_0, k_1]$ *is $(t, q_{sig}, q_{hash}, \epsilon)$-secure, where*

$$t(k) = t'(k) - [q_{sig}(k) + q_{hash}(k) + 1] \cdot k_0 \cdot \Theta(k^3), \text{and} \tag{1}$$

$$\epsilon(k) = \epsilon'(k) + [2(q_{sig}(k) + q_{hash}(k))^2 + 1] \cdot (2^{-k_0} + 2^{-k_1}). \tag{2}$$

Proof (short sketch). We assume to have a forger $\mathcal{F}_{\text{PSS-RSA}}$ that has access to an oracle $\mathcal{O}$ that will sign up to q_{sig} messages $m_1, \ldots, m_{q_{sig}}$ and answer up to q_{hash} queries for h or g. Furthermore we assume that $\mathcal{F}_{\text{PSS-RSA}}$ in time less than t and with probability ϵ can output a message m which has not been signed by $\mathcal{O}$.

We will construct an attacker $\mathcal{A}_{\text{RSA}}$ capable of inverting RSA, say given N, e and y can output x s.t. $y = f(x) = x^e \bmod N$. To construct $\mathcal{A}_{\text{RSA}}$, $\mathcal{F}_{\text{PSS-RSA}}$ is instantiated, however, instead of access to $\mathcal{O}$, $\mathcal{A}_{\text{RSA}}$ will answer all queries. A signing request on m_i is answered by randomly selecting $x_i \in_{\text{R}} \mathbb{Z}_N$, calculating $y_i = f(x_i)$ and due to the RO model $\mathcal{A}_{\text{RSA}}$ can specify h and g in a way that makes y_i consistent with m_i. On h or g requests $\mathcal{A}_{\text{RSA}}$ returns a value consistent with $f(x_i)y$ for a random x_i. If $\mathcal{F}_{\text{PSS-RSA}}$ later on makes a forgery s.t. $\tilde{x} = f^{-1}(f(x_i)y)$, then due to the multiplicative homomorphic property of RSA $\tilde{x} = x_i f^{-1}(y)$. Therefore $\mathcal{A}_{\text{RSA}}$ can return $x = \tilde{x}x_i^{-1} = f^{-1}(y)$.

We refer to [BR96] for a full and formal proof.

It is important to note that the proof of Theorem 2 uses the RSA permutation in a blackbox way except for its multiplicative homomorphic property. Therefore the proof will work for PSS used with any multiplicative homomorphic oneway permutation including M-RSA. We can now formulate the security of using M-RSA in connection with PSS-RSA.

Corollary 2 (Security of PSS-M-RSA). *In the RO model: If RSA is (t', ϵ')-secure. Then for any q_{sig}, q_{hash} the signature scheme* PSS-M-RSA$[k_0, k_1]$ *is $(t, q_{sig}, q_{hash}, \epsilon)$-secure, where*

$$t(k) = t'(k) - [q_{sig}(k) + q_{hash}(k) + 1] \cdot k_0 \cdot \Theta(k^3), \text{and} \tag{3}$$

$$\epsilon(k) = \epsilon'(k) + [2(q_{sig}(k) + q_{hash}(k))^2 + 1] \cdot (2^{-k_0} + 2^{-k_1}). \tag{4}$$

Proof. A reduction from PSS-M-RSA to M-RSA follows analogously the reduction from Theorem 2 with the exact same overhead. The overhead of the reduction from M-RSA to RSA (Corollary 1) is dominated by the overhead from PSS-M-RSA to M-RSA.

8 Security of Multi-Prime OAEP-RSA Encryption

In this section we will see how some of the techniques from the previous section also applies to cryptosystems. Lemma 1 and Corollary 1 states the security of plain RSA and therefore also the plain RSA cryptosystem, however, as in the case of digital signatures, and due to some of the same problems plain RSA encryptions are very seldom used in practice. We therefore investigate the security of what is known as OAEP-RSA, a cryptosystem widely used in practice.

Bellare and Rogaway [BR94] introduces Optimal Asynchronous Encryption Padding (OAEP) as a way to achieve CCA2 security[1] for RSA in the random oracle model. They proved that OAEP-RSA is plaintext aware 1 (PA1) secure based only on the one wayness of RSA. However, as pointed out and formally proved by Shoup [Sho02] PA1 security does not imply CCA2 security, contrary to this Shoup [Sho02] also proved that if 3 is used as the public exponent then RSA-OAEP is actually CCA2 secure. This result was further extended by Fujisaki et al. [FOPS04] to RSA-OAEP being CCA2 secure regardless of the public exponent. This result is based on additional properties than one wayness, namely that RSA is *Set Partial Domain One Way*.

A permutation is *Partial Domain One Way*, if no adversary is able to extract a certain number of the most significant bits of the preimage, and *Set Partial Domain One Way*, if no adversary is able to compute a set where one of the elements is equal to a certain number of the most significant bits of the preimage.

Definition 3 (Set-Partial-Domain-One-Way). *We define a permutation f as being (ℓ, t, ϵ)-set-partial-domain-one-way if no adversary A outputting a set with ℓ elements and running in time t exists s.t. $Pr(s \in A(f(s||t)) > \epsilon$, where the length of s is $k - k_0$, with k_0 being defined by the size of the hashing used in OAEP padding.*

To prove the exact security of OAEP-M-RSA we first need to show that M-RSA is *Set Partial Domain One Way*. This is done by taking a lemma which originates from [FOPS04], and states that RSA is *Set Partial Domain One Way*, and see that it also covers M-RSA.

Lemma 2 (Modified [FOPS04, Lemma 4]). *Let A be an algorithm, which in time t and with probability ϵ, is capable of computing a q-set containing $k - k_0$ ($k > 2k_0$) of the most significant bits of the e'th root of its input. Then there exists an algorithms B capable of inverting M-RSA.*

[1] Security against adaptively chosen ciphertext attacks.

Proof (short sketch). This proof is analogous to the proof of [FOPS04, Lemma 4]. This proof is based on the self-reducibility of RSA, which is the same for M-RSA. The algorithm B runs A twice to compute two sets containing the a partial preimage of x and $r^e x$, for a random value r. Then B utilizes a special designed lattice version of Gaussian reduction to solve a set of linear modular equations defined by these partially preimages. We refer to [FOPS04] for a full and formal proof.

The theorem bellow, due to [FOPS04], states the tightness of the OAEP-RSA construction. The value q_D specifies the number of times the adversary is allowed to invoke a decryption oracle, and q_H and q_G is the number of allowed invocations to the random oracle simulating the two hash algorithms H and G used in OAEP.

Theorem 3 (Modified [FOPS04, Theorem 2]). *In the RO model, for any q_D,q_G and q_H if there exists an adversary A $(t, q_D, q_G, q_H, \epsilon)$-breaking OAEP-M-RSA (with $k > 2k_0$) then there exists and algorithm B (t', ϵ')-inverting RSA, with:*

$$\epsilon' > \frac{\epsilon^2}{4} - \epsilon \cdot \left(\frac{q_D q_G + q_D + q_G}{2^{k_0}} + \frac{q_D}{2^{k_1}} + \frac{32}{2^{k-2k_0}} \right)$$
$$t' \leq 2t + q_H \cdot (q_H + 2q_G) \cdot O(k^3)$$

Proof. From Fujisaki et al. [FOPS04] we have a Theorem (Theorem 1) stating the security of OAEP used in connection with a *set-partial-one-way* permutation, this in connection with Lemma 2 gives us the above result.

Since the bounds of [FOPS04] (Theorem 1) are less tight compared with the reduction from M-RSA to RSA (Corollary 1), the bound for breaking OAEP-RSA and OAEP-M-RSA is the same. So we achieve the same level of security (bit-length of the individual primes) using M-RSA as using standard RSA.

We note that in addition to the (above) bound equal to Theorem 2 in [FOPS04, FOPS04] also presents a slightly tighter bound in their Theorem 3. This result is also based on the partial domain one-wayness of RSA and therefore proving this bound for OAEP-M-RSA follows analogously from their proof.

References

[ACS02] Algesheimer, J., Camenisch, J.L., Shoup, V.: Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 417–432. Springer, Heidelberg (2002)

[BF97] Boneh, D., Franklin, M.K.: Efficient generation of shared RSA keys. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 425–439. Springer, Heidelberg (1997)

[BF01] Boneh, D., Franklin, M.K.: Efficient generation of shared RSA keys. J. ACM **48**(4), 702–722 (2001)

[BH98] Boneh, D., Horwitz, J.: Generating a product of three primes with an unknown factorization. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 237–251. Springer, Heidelberg (1998)

[Ble98] Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, p. 1. Springer, Heidelberg (1998)

[BR94] Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995)

[BR96] Bellare, M., Rogaway, P.: The exact security of digital signatures - how to sign with RSA and rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)

[DK01] Damgård, I.B., Koprowski, M.: Practical threshold RSA signatures without a trusted dealer. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, p. 152. Springer, Heidelberg (2001)

[DM09] Damgård, I., Mikkelsen, G.L.: On the theory and practice of personal digital signatures. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 277–296. Springer, Heidelberg (2009)

[DM10] Damgård, I., Mikkelsen, G.L.: Efficient, robust and constant-round distributed rsa key generation. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 183–200. Springer, Heidelberg (2010)

[DMS14] Damgård, I., Mikkelsen, G.L., Skeltved, T.: On the security of distributed multiprime RSA. IACR ePrint Archive (2014)

[FMY98] Frankel, Y., MacKenzie, P.D., Yung, M.: Robust efficient distributed RSA-key generation. In: Vitter, J.S. (ed.) STOC, pp. 663–672. ACM (1998)

[FOPS04] Fujisaki, E., Okamoto, T., Pointcheval, D., Stern, J.: RSA-OAEP is secure under the RSA assumption. J. Cryptol. **17**(2), 81–104 (2004)

[Gil99] Gilboa, N.: Two party RSA key generation. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 116. Springer, Heidelberg (1999)

[Gir91] Girault, M.: Self-certified public keys. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 490–497. Springer, Heidelberg (1991)

[GRJK07] Gennaro, R., Rabin, T., Jarecki, S., Krawczyk, H.: Robust and efficient sharing of RSA functions. J. Cryptol. **20**(3), 393 (2007)

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

[Pai99] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 223. Springer, Heidelberg (1999)

[RSA02] RSA Laboratories. PKCS #1 v2.1: RSA cryptography standard. Technical report (2002)

[Sch91] Schnorr, C.-P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)

[Sho02] Shoup, V.: OAEP reconsidered. J. Cryptol. **15**(4), 223–249 (2002)

http://www.springer.com/978-3-319-15942-3

Information Security and Cryptology – ICISC 2014
17th International Conference, Seoul, South Korea,
December 3–5, 2014, Revised Selected Papers
Lee, J.; Kim, J. (Eds.)
2015, XIII, 448 p. 83 illus., Softcover
ISBN: 978-3-319-15942-3