Attack Against Ibrahim's Distributed Key Generation for RSA

Błażej Brzeźniak, Lucjan Hanzlik, Przemysław Kubiak, and Mirosław Kutyłowski
(Corresponding author: Lucjan Hanzlik)

Faculty of Fundamental Problems of Technology, Wrocław University of Technology
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
(Email: lucjan.hanzlik@pwr.wroc.pl)
(Received June 9, 2011; revised and accepted Nov. 15, 2011)

Abstract

Distributed RSA key generation protocols aim to generate RSA keys in such a way that no single participant of the protocol can learn factorization of the RSA modulus. In this note we show that two recent protocols of this kind (Journal of Network Security, Vol. 7, No. 1, 2008, pp. 106-113 and Vol. 8, No. 2, 2009, pp. 139-150) fail their security target. We present an attack that can be launched by any protocol participant after terminating distributed key generation process.

Keywords: Attack, distributed RSA key generation, greatest common divisor

1 Introduction

Many papers (see for example [1, 2, 3, 4, 6, 7, 11, 14]) are devoted to distributed generation of RSA numbers. Their goal is to generate RSA keys (e, N), d (i.e., $N = pq$ for some primes p, q, and e, d chosen to satisfy congruence $e \cdot d \equiv 1 \bmod \operatorname{lcm}(p-1, q-1)$) in such a way that no single participant learns the private key d. One motivation for this kind of protocols is to generate RSA numbers in a situation that a user cannot fully trust any single hardware/software unit. Through distributing the process we make sure that leaking the private key requires at least some collusion between different units.

For $e \cdot d < N^2$ knowledge of the private key is known to be polynomially equivalent to knowledge of the factorization [5]. However, there is an efficient probabilistic algorithm that factors N given any integers e and d such that $e \cdot d \equiv 1 \bmod \operatorname{lcm}(p-1, q-1)$, i.e., the probabilistic algorithm does not impose constraints on the product $e \cdot d$ (for algorithm's description see e.g. [12, Sect. IV.2]).

Consequently, the output of distributed RSA key generation algorithms is usually a public key (e, N) and a set of private key components $d_1, \ldots, d_n$, where different key components d_i, $i = 1, \ldots, n$, are stored privately on n independent devices, and all of them (or some subset of them in case of a threshold scheme) must be used cooperatively to perform the RSA signature/decryption operation. However, disclosure of all but one component d_i (or a number of components that is lower than the threshold) must not lead to compromise of the private RSA key.

From the point of view of practical applications a protocol of distributed RSA key generation must be efficient regarding in particular its communication complexity, storage demands and computational complexity. This is a challenging task, since already RSA key generation by a single participant leads to nontrivial computational complexity (cf. efforts to accelerate the process on constrained devices [10, 13]).

This note concerns RSA key generation protocols proposed in papers [8, 9]. We present a simple attack that fully breaks these schemes. Namely, we show that after executing the key generation protocol any participant can factorize the modulus.

2 The Flaws

2.1 Protocol [8]

The protocol from [8] is a two-party protocol, it consists of four stages. In the first stage (see Section 2.3 for some details) numbers of the form $(a_1 + b_1)$ are checked for primality inside moduli of the form $N_1 = (a_1 + b_1)p_a p_b$, where a_1, p_a are chosen by Alice, b_1, p_b are chosen by Bob, and p_a, p_b are primes freshly chosen for each new candidate $(a_1 + b_1)$. Hence if $a_1 + b_1$ is prime, then N_1 is a three-prime modulus – to check if N_1 is really a three-prime the protocol executes distributed Fermat test. In the second stage a similar work is done and moduli of the form $N_2 = (a_2 + b_2)q_a q_b$ are examined. Next, during the third stage (see Section 2.3 for some details) for moduli N_1, N_2 that successfully completed Fermat test the RSA modulus $N = (a_1 + b_1)(a_2 + b_2)$ is calculated in a distributed way. The fourth stage is composed of computation of participant's private shares to a private exponent.

Let us observe that none of the "blinding" primes p_a, p_b, q_a, q_b is incorporated into modulus N. We have

$$\gcd(N_1, N) = \gcd((a_1 + b_1)p_a p_b, (a_1 + b_1)(a_2 + b_2)),$$

hence $a_1 + b_1 | \gcd(N_1, N)$. For large random numbers we expect that $\gcd(p_a p_b, a_2+b_2) = 1$, thus $\gcd(N_1, N) = a_1+b_1$. As N_1 and N are known to each protocol participant, they can perform this computation and factorize N.

2.2 Protocol [9]

Paper [9] is an extension of [8] to the (t, n) threshold multiparty key generation. That is, the number of participants is $n > 3t$, $t \geq 1$, and the protocol is claimed to be t-private. This means that an adversary that successfully eavesdrops no more than t participants must not be able to factor N.

This protocol also proceeds in stages. In the first stage moduli of the form $N_1 = p_1(\Pi_{i=1}^{t+1} q_i)$ are examined, where for each tested N_1 value p_1 is a distributively generated candidate for a factor of N, q_i is a prime number chosen by the ith participant. The numbers q_i are freshly generated for each successive candidate p_1. It is assumed in [9] that only participants from a subset of cardinality $t + 1$ generate primes in the product $\Pi_{i=1}^{t+1} q_i$. The first stage terminates, if the test on N_1 indicates that the current candidate p_1 is prime. The participants do not know complete factorization of N_1 (in particular they do not know p_1), although they know each tested N_1. Note that each participant knows at most one q_i – a factor of N_1, but as long as no more than t participants collude, they cannot extract all factors of N_1. The second stage is analogous to the first one: moduli N_2 are examined, and the final N_2 contains a prime factor p_2. Again, no participant knows complete factorization of N_2 (in particular, no participant knows p_2), although each of them knows N_2. In the next stage RSA modulus $N = p_1 p_2$ is calculated in a distributed way.

It is easy to see that the protocol from [9] inherits the flaw from its predecessor. Namely, in practice it suffices to compute $\gcd(N_1, N)$ to find a nontrivial factor of a large RSA number N.

2.3 A Toy Example of the Attack on Protocol [8]

According to Subsect. 2.1 the protocol [8] has two participants, say Alice and Bob. Below we present Alice's view to an exemplary execution of protocol's components relevant for the attack – we indicate data known to Alice. Subsequently we show how Alice can perform the attack.

generating N_1 the procedure applies mul-to-sum routine (see Appendix):

1) Alice chooses at random a number $a_1 = 25$ and a prime number $p_a = 17$. Bob chooses b_1 and p_b in a similar way.
2) Alice computes locally $A = a_1 p_a = 425$. Bob computes a similar value $B = b_1 p_b$.
3) Alice and Bob both perform a mul-to-sum routine to share Ap_b in an additive way $x_a + x_b$, where $x_a = 543$ is held by Alice and x_b by Bob.
4) Alice and Bob both perform a mul-to-sum routine to share $Bp_a = y_a + y_b$, where $y_a = 378$ is held by Alice and y_b is held by Bob.
5) Alice sends $x_a + y_a = 921$ to Bob and receives $x_b + y_b = 14362$ from Bob.
6) Alice computes $N_1 = x_a + x_b + y_a + y_b = 15283$. This means that $15283 = (a_1 + b_1)p_a p_b = (25 + b_1) \cdot 17 \cdot p_b$.

testing N_1**:** the idea is to apply Fermat test: if $a_1 + b_1$ is a prime, then $\varphi(N_1) = (a_1 + b_1 - 1)(p_a - 1)(p_b - 1)$. So we may choose g at random and test if $g^{(a_1+b_1-1)(p_a-1)(p_b-1)}$ equals 1:

1) Alice and Bob agree on an $g \in \mathbb{Z}_{N_1}^*$, suppose that they agreed on $g = 14$.
2) Alice sends $G_a = g^{(p_a-1)(a_1-1)} \mod N_1 = 11102$ to Bob and receives $G_b = g^{(p_b-1)b_1} \mod N_1 = 13718$ from Bob.
3) Alice sends $G_b' = G_b^{p_a-1} \mod N_1 = 12819$ to Bob and receives $G_a' = G_a^{p_b-1} \mod N_1 = 4931$ from Bob.
4) Alice and Bob compute $G = G_a' G_b' \mod N_1$ and they check if $G = 1$. Since indeed $G = 1 \mod N_1$ then, according to [8, Subsect. 8.1], both parties assume that $a_1 + b_1$ is prime.

choosing and testing N_2**:** Alice chooses at random a new number $a_2 = 2$ and a prime number $q_a = 31$. Bob chooses new values b_2 and q_b. Together they compute $N_2 = 10013$. Then Alice and Bob perform similar steps as steps 1-4 of the test procedure.

computing N**:** step by step Alice and Bob remove the blinding factors p_a, p_b, q_a, q_b from $N_1 N_2$:

1) Alice sends $N_a = (N_1 N_2)/(p_a q_a)$ to Bob.
2) Bob sends $N_b = (N_1 N_2)/(p_b q_b) = 310403$ to Alice.
3) Alice computes $N = N_b/(p_a q_a) = (a_1 + b_1)(a_2 + b_2) = 589$.

After those steps Alice has enough data to factorize N: she computes $\gcd(N_1, N) = \gcd(15283, 589) = 31$ and $\gcd(N_2, N) = \gcd(10013, 589) = 19$.

3 Final Comments

The algorithms from [8, 9] are based on the principle of separate generation of factors of $N = p_1 p_2$. In order to prevent the participants from learning the factors multiplicative blinding is applied: in this way the players see

N_1 and N_2, but not the factors p_1 and p_2 of, respectively, N_1 and N_2. Unfortunately, multiplicative blinding MUST fail, as existence of any protocol value that contains the factor p_1 but not p_2 (or vice versa) leads immediately to factorization of N by computing $\gcd(N, N_1)$. Finally, we are afraid that there is no way to avoid the presented flaws, as long as these protocols are based on separate primality testing of the factors of N.

Acknowledgments

This work has been supported by Polish Ministry of Science and Higher Education, project O R00 0015 07 and by Foundation for Polish Science, Programme "MISTRZ".

References

[1] J. Algesheimer, J. Camenisch, and V. Shoup, "Efficient computation modulo a shared secret with application to the generation of shared safe-prime products," in *Crypto* (M. Yung, ed.), vol. LNCS 2442, pp. 417–432. Spriger-Verlag, 2002.

[2] S. R. Blackburn, S. Blake-Wilson, M. Burmester, and S. D. Galbraith, "Weaknesses in shared RSA key generation protocols," in *IMA International Conference*, vol. LNCS 1746, pp. 300–306. Spriger-Verlag, 1999.

[3] D. Boneh and M. K. Franklin, "Efficient generation of shared RSA keys (extended abstract)," in *Crypto* (B. S. K. Jr., ed.), vol. LNCS 1294, pp. 425–439. Spriger-Verlag, 1997.

[4] C. Cocks, "Split knowledge generation of RSA parameters," in *IMA International Conference* (M. Darnell, ed.), vol. LNCS 1355, pp. 89–95. Spriger-Verlag, 1997.

[5] J. S. Coron and A. May, "Deterministic polynomial-time equivalence of computing the RSA secret key and factoring," *Journal of Cryptology*, vol. 20, pp. 39–50, 2007.

[6] I. Damgård and G. L. Mikkelsen, "Efficient, robust and constant-round distributed RSA key generation," in *TCC* (D. Micciancio, ed.), vol. LNCS 5978, pp. 183–200. Spriger-Verlag, 2010.

[7] Y. Frankel, P. D. MacKenzie, and M. Yung, "Robust efficient distributed RSA-key generation," in *PODC*, p. 320, 1998.

[8] M. H. Ibrahim, "Eliminating quadratic slowdown in two-prime RSA function sharing," *International Journal of Network Security*, vol. 7, pp. 106–113, 2008.

[9] M. H. Ibrahim, "Efficient dealer-less threshold sharing of standard RSA," *International Journal of Network Security*, vol. 8, pp. 139–150, 2009.

[10] M. Joye and P. Paillier, "Fast generation of prime numbers on portable devices: An update," in *CHES* (L. Goubin and M. Matsui, eds.), vol. LNCS 4249, pp. 160–173. Spriger-Verlag, 2006.

[11] M. Joye and R. Pinch, "Cheating in split-knowledge RSA parameter generation," in *Workshop on Coding and Cryptography* (D. Augot and C. Carlet, eds.), pp. 157–163, 1999.

[12] N. Koblitz, *A course in number theory and cryptography*, vol. 2nd edition. Springer, 1994.

[13] C. Lu, A. L. M. dos Santos, and F. R. Pimentel, "Implementation of fast RSA key generation on smart cards," in *Proceedings of the 2002 ACM Symposium on Applied Computing*, pp. 214–220, 2002.

[14] E. Ong and J. Kubiatowicz, "Optimizing robustness while generating shared secret safe primes," in *Public Key Cryptography* (S. Vaudenay, ed.), vol. LNCS 3386, pp. 120–137. Spriger-Verlag, 2005.

Appendix-The Mult-to-Sum Subroutine from [8]

Let $\mathcal{R}$ be a publicly known ring and let $\rho = \log|\mathcal{R}|$. Let $\ell \leq \rho$. Alice holds an ℓ-bit secret value $a \in \mathcal{R}$ and Bob holds an ℓ-bit secret value $b \in \mathcal{R}$. Alice and Bob want to additively share ab with no information revealed about a or b. The protocol is as follows:

- Bob selects uniformly at random ℓ ring elements $c_0, \ldots, c_{\ell-1}$ and defines ℓ pairs of ring elements $(t_0^{(0)}, t_0^{(1)}), \ldots, (t_{\ell-1}^{(0)}, t_{\ell-1}^{(1)})$. Namely, he sets $t_i^{(0)} = c_i$ and $t_i^{(1)} = 2^i b + c_i$ for $i = 0, \ldots, \ell - 1$.
- Let the binary representation of a be $(a_{\ell-1} \ldots a_0)_2$, Alice and Bob perform ℓ invocations of OT_2^1 (1-out-of-2 Oblivious Transfer protocol). In the i-th invocation Alice chooses $t_i^{(a_i)}$ from the pair $(t_i^{(0)}, t_i^{(1)})$.
- Alice sets $x = \sum_{i=0}^{\ell-1} t_i^{(a_i)}$ while Bob sets $y = -\sum_{i=0}^{\ell-1} c_i$.

As a result $x + y = ab$ over $\mathcal{R}$.

Błażej Brzeźniak received engineer degree in computer science in 2011. He is currently studying at Wrocław University of Technology. He is interested in cryptography, mathematics and astronomy.

Lucjan Hanzlik received master degree in computer science from Wrocław University of Technology in 2011. Currently, he is an Ph.D. student at Wrocław University of Technology. His main interest is Cryptography. More specifically, working on the design of efficient and secure cryptographic algorithms.

Przemysław Kubiak received master degree in mathematics in 1997 and Phd in 2001. He is an assistant professor for computer science at Wrocław University of Technology. His research interests are in public key cryptography.

Mirosław Kutyłowski is a full professor for computer science at Wrocław University of Technology. He received master degree in mathematics in 1980, PhD in 1986, Habilitation in 1992, from University of Wrocław. In 1999 he got professor title. Humboldt Fellow in TH Darmstadt in 1987-88. He was affiliated with Wrocław University, University of Paderborn, Adam Mickiewicz University in Poznań. He is involved in e-government issues, serving as adviser of different public institutions. In 2009 he received prize "Mistrz" from Foundation for Polish Science in technical sciences. Mirosław Kutyłowski is a member of Reasearch Council of Institute of Computer Science, Polish Academy of Sciences.