RSA modulus generation in the two-party case

Gérald Gavin

ERIC Lab - University of Lyon - France
E-mail: gavin@univ-lyon1.fr

Abstract. In this paper, secure two-party protocols are provided in order to securely generate a random k-bit RSA modulus n keeping its factorization secret. We first show that most existing two-party protocols based on Boneh's test are not correct: an RSA modulus can be output in the malicious case. Recently, Hazay et al. [13] proposed the first proven secure protocol against any polynomial active adversary. However, their protocol is very costly: several hours are required to output a 1024-bit RSA modulus on a standard platform. In this paper, we propose an other approach consisting of post-processing efficient existing Boneh's based protocols. The running time of this post-processing can be neglected with respect to the running time of the whole protocol.

Keywords: RSA modulus, Boneh's test, keys share.

1 Introduction

Many cryptographic schemes are based on RSA moduli, for example RSA and Paillier. Many threshold versions have been proposed [20], [19], [11], [9], [7]. In these versions, parties must generate an RSA modulus n and distribute public and private keys such that the factorization of n is kept secret. A solution consists of invoking a third party, called the trusted dealer, which generates RSA moduli and shared keys.

The natural question is to know whether the dealer can be efficiently removed. Boneh and Franklin [3] provided a positive answer when the number of parties is larger than 3. This result is based on an adaptation of the Fermat's primality test for RSA moduli, called Boneh's test. It is shown secure against passive adversaries. Frankel et al. [22], Algesheimer et al. [1] have proposed a robust protocol for the multi-party case in the presence of a minority of arbitrarily misbehaving malicious parties. Damgard and Mikkelsen [10] have proposed an other biprimality test with a better error estimate, but it cannot be used directly in the two-party setting with active adversaries. The security of threshold Paillier's scheme requires that moduli are safe, i.e the product of safe primes. Fouque and Stern [11] have shown that it suffices that the sub-group

QR(n) of quadratic residues is cyclic to ensure security. They provide secure multi-party protocols for RSA moduli having this property.

This paper deals with the two-party case. Gilboa [12] Poupard and Stern [18] and Straub [21] have proposed a solution based on Oblivious Transfert (OT). They propose a robust protocol ModulusGeneration which computes $n = (p_A + p_B)(q_A + q_B)$ where p_A, q_A and p_B, q_B are two k-bit integers randomly chosen respectively by Alice and by Bob. They then apply Boneh's test [4]. For concreteness, this biprimality test consists of randomly choosing $a \in Z_n^*$ such that $(\frac{a}{n}) = 1$, then Alice computes $v_A = a^{(n-p_A-q_A+1)/4} \bmod n$ and Bob computes $v_B = a^{(p_B+q_B)/4} \bmod n$. If $v_A/v_B \neq \pm 1 \bmod n$ then the test fails. If the test does not fail for several values of a there is a high probability that n is an RSA modulus. However, let us imagine that n is **not** an RSA modulus and that Alice knows the factorization of n while Bob does not (Alice could have greater resources or better factorization algorithms). In this case, Alice can guess (with non negligible probability) $p_B + q_B$ and thus v_B. She could therefore force Bob to believe that n is an RSA modulus. It proves that this solution is only secure against passive adversary (see appendix A for a concrete presentation and evaluation of this attack). In [2], the authors have proposed solutions against this attack. However their protocols have not been proven secure and above all their solution is not at all practical. For instance, $1,6.10^8$ El Gamal encryptions/decryptions are required to output a 1024-bit RSA modulus. Recently, Hazay et al. [13] proposed the first proven secure protocol against any polynomial active adversary. To reach this security level, authors propose a zero-knowledge proof to prove that v_A and v_B are correctly computed. However, this zero-knowledge proof is executed for each candidate n. This protocol based on a cut-and-choose approach is very costly: it requires $O(l)$ (l being a security parameter) exponentiations per loop in Boneh's test. As the number of candidates n which should be tested is about $O(k^2)$, the number of exponentiation of their protocol is $O(l^2k^2)$. We roughly estimate that several hours are needed to output a 1024-RSA modulus with this protocol (more than 10^7 modular exponentiations are required).

The key idea of this paper consists of post-processing existing (efficient) protocols based on Boneh's test with a running time in $O(l)$ (few seconds are requiring for 1024-bit numbers): this post-processing can be neglected with respect to the whole protocol. This idea was already suggested in [13] (section "Practical considerations") in order to drastically improve efficiency. However, security was not proven in this setting: authors claim: "*We cannot simply postpone all proofs; care must be taken* *to allow simulation and not to reveal information that would allow a malicious party to, e.g., fake some zero-knowledge proof at a later point*". While our approach differs from theirs, this paper can be seen as a positive answer to this claim.

For concreteness, the output n of classical Boneh's test based protocols is tested again. Here, it is assumed the existence of a correct and private protocol ModulusGeneration computing $n = (p_A + p_B)(q_A + q_B)$ (many versions can be found in the literature [12], [3], [18], [21]). The parties then test whether n is an RSA modulus with the classical Boneh's test (see [4]). As discussed before, this test may be incorrect against active adversaries. Thus, if (and only if) n has been determined an RSA modulus previously, it is tested again with a test directly derived from Boneh's test. This test is implemented in protocols Gamma and RSAModulusTest. The efficiency improvement comes from the fact that this second test is applied only once. The main protocol of this paper, RSAModulusGeneration is an implementation of the following generic scheme:

1. Each party generates a public key and a secret key of an **additively** homomorphic encryption scheme S. E_A (resp. E_B) refers to the encryption function generated by Alice (resp. Bob).
2. They invoke ModulusGeneration to get moduli n until n succeeds the classical Boneh's test. Each time the test fails, each party checks that the other party has behaved honestly.
3. Alice (resp. Bob) invokes Gamma to get an encryption $\Gamma_B = E_B((p-1)(q-1))$ (resp. $\Gamma_A = E_A((p-1)(q-1))$) where $n = pq$
4. Alice (resp. Bob) invokes RSAModulusTest (as party 1) on the input Γ_B (resp. Γ_A) to be convinced that n is really an RSA modulus. If the test fails, the protocol fails (it means that a party has behaved dishonestly in step 2.)

It is important to notice that an adversary does not gain any advantage when RSAModulusGeneration fails. This is used to simplify and improve protocols and their security proof. We propose original tools to prove that an adversary **cannot learn** the factorization of n without making the protocol fails: this is captured in the definition of the property nPrivate introduced in section 2. This property is dedicated to factorization problem but can be straightforwardly extended to other problems.

2 Problem and security statements

Let us make explicit the conjecture intrinsically made in Boneh's test based protocols. This conjecture is related to the difficulty of factorizing RSA moduli.

Conjecture 1. Let $k \in \mathbb{N}$ and $n = pq$ be an RSA modulus chosen at random such that $p \equiv q \equiv 3 \mod 4$, $p \in [p_A + 2^k, p_A + 2^{k+1}[\bigcap \mathbb{N}$ and $q \in [q_A + 2^k, q_A + 2^{k+1}[\bigcap \mathbb{N}$ where p_A, q_A are two arbitrary integers. There exists a negligible function δ such that for all probabilistic polynomial-time (p.p.t) algorithms A, A outputs the factorization of n with a probability smaller than $\delta(k)$ only given n, p_A and q_A.

In this paper we wish to design a two-party protocol for securely generating RSA moduli. Existing protocols for securely computing RSA moduli are decomposed in two parts. First, parties securely compute a candidate $n = pq$ by using a secure protocol called here ModulusGeneration, they then test n with Boneh's test. In this paper, we assume the existence of such a protocol ModulusGeneration whose the specifications are detailed in the next definition. In particular, it should be correct and private (see [14]).

Definition 1. *ModulusGeneration is protocol implementing the following ideal scheme. Alice chooses two secret positive integers p_A, q_A and Bob chooses two secret positive integers p_B, q_B. Then, they send these values to a trusted party which outputs $n = pq$ where $p = p_A + p_B$ and $q = q_A + q_B$ if and only if $p_A \equiv q_A \equiv 3 \mod 4$ and $p_B \equiv q_B \equiv 0 \mod 4$. ModulusGeneration is assumed to be secure against any active polynomial adversary in the stand-alone setting.*

ModulusGeneration checks that $p_A + p_B \equiv q_A + q_B \equiv 3 \mod 4$ because it is required for Boneh's test but ModulusGeneration does not need to verify the size of the input integers p_A, q_A, p_B and q_B. It will be done in the protocol Gamma only once, i.e. on the first modulus n which successfully passes Boneh's test. Versions of ModulusGeneration can be found in the literature (see [12], [18]). In section 3, we present a two-party protocol, called RSAModulusGeneration, which securely (we will clarify what it means below) generates RSA moduli. The idea consists of generating an RSA modulus n in the classical way (ModulusGeneration + Boneh's test). If n is determined to be an RSA modulus then it is tested again (Gamma + RSAModulusTest). The second test make RSAModulusGeneration fail if n is not an RSA modulus. Thus, the second test is applied only once. The

RSAModulusGeneration security proof exploits the fact that an adversary A does not gain any advantage when the protocol fails. For this reason, weaker security requirements will be considered by restricting the security analysis of RSAModulusGeneration to the two following issues :

1. RSAModulusGeneration is correct, i.e. the output n is not a k-bit RSA modulus with negligible probability with respect to k.
2. Assuming conjecture 1, for any polynomial adversary A, the probability that the protocol does not fail and that A factors the output n is negligible with respect to k.

In order to prove the second issue, we introduce a property dedicated to our problem, called nPrivate. This property can be interpreted as the difficulty for an adversary to **learn** the factorization of n during the execution of P without making P fail. This composable property is formalized in the next definition. In section 4, we will show that Gamma and RSAModulusTest satisfy this property. Roughly speaking, these protocols are private against adversaries which do not make the protocol fail with non-negligible probability.

Definition 2. *For any $k \in \mathbb{N}$, Γ_k refers to the set of k-bit integers. Let P be an arbitrary two-party protocol where a public integer $n \in \Gamma_k$ is input to P by both parties. Parties may have arbitrary other inputs. Let us assume that the party 1 is honest and the party 2 is controlled by a polynomial adversary A.*

1. *For any $k \in \mathbb{N}$, D_k denotes a probability distribution over $\Gamma_k \times \{0,1\}^*$. A family of such probability distributions $D = (D_k)_{k\in\mathbb{N}}$ is said nPrivate[1] if for any p.p.t algorithm F, the probability that F outputs the factorization of n given $(n, z_n) \leftarrow D_k$ is negligible.*
2. *Let $D = (D_k)_{k\in\mathbb{N}}$ be an arbitrary nPrivate probability distribution family. We propose the following experiment :*
   - *(a) a pair $(n, z_n) \in \Gamma_k \times \{0,1\}^*$ is randomly generated according to D_k, i.e. $(n, z_n) \leftarrow D_k$.*
   - *(b) The corrupted party receives (n, z_n)*
   - *(c) The honest party receives n and arbitrary information about n.*
   - *(d) Parties execute P where the honest party inputs n (and arbitrary other inputs).*

[1] Assuming conjecture 1, such distributions exist.

We say that P is nPrivate against A if the probability that P has not failed and A outputs the factorization of n at the end of the previous experiment is negligible

This definition is maybe difficult to understand because it is not only a security definition but it also depends on the functionality computed by P: for instance, P is not nPrivate against passive adversary if the factorization of n is expected to be output. On the other hand, classical Boneh's test is trivially nPrivate meaning that if an RSA modulus is tested, an adversary cannot learn the factorization of n without cheating the test (even by making it fail). Typically, the vector z_n contains the information learnt during the previous protocols. For instance, before the execution of Boneh's test (at the end of the execution of ModulusGeneration), (n, z_n) is computationally indistinguishable of (n, p_A, q_A) assuming ModulusGeneration is stand-alone secure. A can use z_n[2] to behave (maliciously) within P. Party 1 may know the factorization of n or may have partial information regarding n which he may use in the execution. At the end of the execution of P, A knows an information set $z_{n,A,P}$ "bigger" than z_n which intuitively contains z_n and its view in P. Roughly speaking, if an adversary cannot factor an integer n (chosen at random and input by the honest party) before the execution of an nPrivate protocol, it cannot collect enough information to factor n without making the protocol P fail. Let us see informally the link between the property nPrivate and UC-security[3]. In Gamma and RSAModulusTest parties output an encryption. Under the assumption of semantic security of the underlying encryption scheme, such output cannot "help" them to factor n. In such protocols (where the output is not informative about the factorization of n), it is easy to see that UC-security implies "*nPrivateness*": otherwise the ideal case and the real case could be distinguished with non negligible probability (factorization possible in the real-case but not the ideal-case). However the converse is not true. In particular, Gamma will be shown nPrivate but trivially not UC-secure. The property nPrivate especially dedicated to our problem allows to simplify the sub-protocols Gamma and RSAModulusTest and their security proofs. The price to pay is to understand why the next generic protocol GenFact is nPrivate. It will

[2] z_n can been interpreted as *prior* information about n known by A before the execution of P (see [17]).

[3] The UC-model was developed by R. Canetti [5]. Here UC stands for universally composable, which denotes that if a protocol is UC-secure according to the formal definition, then it is secure to use in any context (where it would have been secure to use the ideal functionality).

be used to prove that Gamma and RSAModulusTest, which are restricted implementations of this generic protocol, are nPrivate.

Definition 3. *Let k be a security parameter. GenFact is a protocol between an arbitrary oracle $\mathcal{O}$ and a party 2. The oracle $\mathcal{O}$ inputs a public k-bit integer n and it is assumed that $\mathcal{O}$ answers (correctly or not, depending of the characteristics of $\mathcal{O}$) to any binary question*[4] *(a question whose answer is "yes" or "no"). Party 2 first asks an arbitrary binary question. If the answer is "no" the protocol fails. If the protocol does not fail, party 2 can ask a second binary question. If the answer is "no" the protocol fails and so on.... The number of questions is assumed polynomial in k.* ***Moreover, it is assumed that party 2 receives arbitrary information (e.g. the factorization of n) when the protocol fails.***

The next result states that a polynomial adversary controlling party 2 cannot "learn" the factorization of n without making GenFact fails. Intuitively, if A wants to learn r bits, GenFact fails with probability $1-2^{-r}$.

Lemma 1. *GenFact is nPrivate against any polynomial adversary A controlling party 2.*

Proof. (Sketch.) Let $D = (D_k)_{k\in\mathbb{N}}$ be an nPrivate probability distribution family (see definition 2) and A be a polynomial adversary controlling party 2. Let $(n, z_n) \leftarrow D_k$. The probability that A factors n without making GenFact fail is denoted by P_A. Let us suppose that GenFact is not nPrivate by assuming that $P_A > p(k)$ where p is a polynomial.

GenFact fails if A receives at least one "no" to any of the arbitrary binary questions mentioned in definition 3. Thus, when A chooses its i^{th} question, it has received $i-1$ positive answers[5], i.e. ("yes", "yes",...,"yes"). Thus, the received messages are not informative (the transcript of the protocol can be exactly simulated in polynomial time without knowing the factorization of n) to choose its i^{th} question. Consequently, A can choose the polynomial-size list $\mathcal{L}$ of its questions *a priori*, i.e. before the execution of GenFact.

Let us show that there exists a polynomial adversary A', only given n, z_n, able to factor n with probability larger than $p(k)$ without interacting with the oracle $\mathcal{O}$. Indeed, A' chooses the same list $\mathcal{L}$ of questions *a priori*, assumes that the answers are "yes" and then computes the factorization

[4] For instance, "Does God exist?" is a binary question. In order to factor n, the party 2 should ask questions relative to n, e.g. n is a RSA-modulus $n = pq$? Is the i^{th} bit of p equal to 1? etc...

[5] Otherwise the protocol has failed previously.

of n as A would do (one could imagine that A' invokes A on the same list $\mathcal{L}$ of questions with positive answers). We consider the three following events S, F, F':

1. $S =$ Genfact does not fail.
2. $F = A$ factors n
3. $F' = A'$ factors n

The protocol does not fail when all the questions of $\mathcal{L}$ get a positive answer. In this case, A' correctly assumes that all the answers are positive implying that A and A' have the same information: consequently they factor n with the same probability (by construction of A'), i.e.

$$P(F|S) = P(F'|S)$$

By noting that $P_A = P(F|S)P(S)$, we have

$$P(F') \geq P(F'|S)P(S) = P(F|S)P(S) = P_A$$

It implies that $P(F') \geq P_A > p(k)$ meaning that A' factors n with non negligible probability. It contradicts that D is nPrivate. Consequently, $P_A < p(k)$ proving that GenFact is nPrivate.
□

As expected, this lemma implies that Boneh's test is nPrivate (but not correct) against any polynomial adversary. Indeed, this protocol can be seen as a restricted implementation of GenFact where the oracle is replaced by the honest party and the i^{th} binary question is $t_i a^{s_B} \overset{?}{\equiv} 1 \bmod n$, t_i being an arbitrary value chosen by A (recall that in GenFact, party 2 receives arbitrary information if the protocol fails: in Boneh's test, it receives $t_i a^{s_B}$).

The next intuitive result (see appendix C for the proof) states that the property nPrivate is stable by (serial) composition.

Proposition 1. ***(Composition.)*** *If R and S are two nPrivate protocols then the composed protocol $P = R \to S$ consisting of executing R and S successively (where honest parties input the same integer n in R and S) is also nPrivate.*

3 RSA Moduli Generation

In this section, we propose a secure (according to the security requirements presented in the previous section) implementation of the main protocol of this paper, i.e. RSAModulusGeneration. This protocol is based on the protocols RSAModulusTest and Gamma as schematically explained in the introduction. These protocols assume the existence of a semantically secure additively homomorphic public key encryption scheme $S = (G, E, D)$. It is also assumed the existence of the following UC-secure protocols:

1. CorrectKey is a ZK-proof proving correctness of keys.
2. EncryptProof is a ZK-proof allowing the private key owner to prove that a public encryption X encrypts a public value x without revealing anything about the private key.
3. Multiplication. Alice has 2 secret encryptions $X = E_B(x)$ and $Y = E_B(y)$ of 2 unknown values x and y. She outputs an encryption of Z of xy, i.e. $Z = E_B(xy)$, without learning and without revealing anything about x and y.
4. Bound is ZK-proof allowing Alice to prove that an encryption $X_A = E_A(x)$ is a valid encryption of a value x smaller than some public threshold B without revealing anything else about x.

Paillier's encryption scheme is a good candidate. Indeed to prove that X encrypts x, the private key owner sends the random value r of the encryption, i.e. $X = g^x r^n \mod n^2$, obtained in the decryption phase[6]. A version of CorrectKey (which consists of proving that n and $\phi(n)$ are co-prime) can be found in [13]. A version of Multiplication can be found in appendix B and A version of Bound can be found in [8]. The classic solution for the protocol Bound is to provide encryptions to the individual bits and prove in zero-knowledge that they are bits using a zero-knowledge proof (e.g. see [8]). A more sophisticated solution for Bound requiring only O(1) exponentiations is presented in [13].

Throughout the paper, the security of the protocols will be studied in the (CorrectKey, EncryptProof, Bound and Multiplication)-hybrid model where these protocols can be replaced by an oracle. According to the composition theorem (see [5]), provided that CorrectKey, EncryptProof, Bound and Multiplication are UC-secure, the security in the hybrid model ensures security in the classical model.

In this paper, Alice (resp. Bob) generates an encryption function E_A (resp. E_B) defined over Z_{n_A} (resp. Z_{n_B}) where n_A (resp. n_B) is a $5k$-size integer. Generally, X_A (resp. X_B) will refer to an encryption done with E_A (resp. E_B). Parties invoke CorrectKey to prove correctness of keys.

[6] El Gamal's encryption scheme does not satisfy this property.

3.1 Protocol Gamma.

In this section, we suppose that the parties have already computed $n = (p_A + p_B)(q_A + q_B)$ with ModulusGeneration assumed secure in the stand-alone setting: p_A, q_A (resp. p_B, q_B) refer to the secret values input by Alice (resp. Bob) in this protocol. Let us recall that p_A, q_A, p_B, q_B are expected to be k-bit integers such that $p_A \equiv q_A \equiv 3 \mod 4$ and $p_B \equiv q_B \equiv 0 \mod 4$. ModulusGeneration is not required to check the size of p_A, q_A, p_B, q_B because it is done here (and thus only once). Gamma aims to securely compute an encryption of $\gamma = (p-1)(q-1)/2$ such that $n = pq$. For concreteness, at the end of the execution of Gamma, Alice obtains an encryption $\mathit{\Gamma}_B = E_B(\gamma)$; Bob does not output anything ensuring correctness against any adversary controlling Alice. Let A be a polynomial adversary controlling Bob. If the parties honestly behave then $p = p_A + p_B$ and $q = q_A + q_B$. If the adversary modifies Bob's inputs, it might happen that $p \neq p_A + p_B$ or $q \neq q_A + q_B$ but it is ensured that $pq = n$ with p and q two k-bit such that $p \equiv q \equiv 3 \mod 4$. In this sense, Gamma is **correct** against any polynomial adversary Bob. We denote the secret $(k-2)$-bit size integers by p'_A, q'_A, p'_B, q'_B such that $p'_A = (p_A - 3)/4$, $q'_A = (q_A - 3)/4$, $p'_B = p_B/4$ and $q'_B = q_B/4$. Bob publicizes encryptions of p'_B and q'_B and proves that these encryptions encrypt $(k-2)$-bit integers with Bound.

By using homomorphic properties, Alice gets encryptions P_B and Q_B of p and q. Alice then invokes Multiplication to get an encryption N_B of pq and checks (by asking Bob to decrypt it) that $pq \equiv n$. She then invokes Multiplication to securely compute an encryption of $\gamma = (p-1)(q-1)/2$.

Proposition 2. *Assume that the encryption scheme S is semantically secure. Gamma is correct and nPrivate against any polynomial adversary in the (Multiplication, Bound, EncryptProof)-hybrid model.*

Proof. The protocol is intrinsically correct against adversary controlling Alice because Bob is not expected to output anything. Let A be a polynomial adversary controlling Bob. In the (Multiplication, Bound, EncryptProof)-hybrid model, A does not interact with Alice except in the choice of the input values. Let us imagine that A inputs $(k-2)$-bit integers p''_B and q''_B such that $p''_B \neq p'_B$ or $q''_B \neq q'_B$. In this case, either the protocol fails (step 3) or $\mathit{\Gamma}_B$ encrypts $(p'-1)(q'-1)2^{-1} \mod n_B$ with $p' = 4(p'_A + p''_B) + 3$, $q' = 4(q'_A + q''_B) + 3$ and $p'q' \equiv n \mod n_B$ (recall that E_B is an encryption function over Z_{n_B}). As n_B was chosen sufficiently large ($5k$-bit integer), there are not modular reductions and the equalities remain true over Z).

Protocol 1 : Gamma.

Require : Let k be a security parameter. Let n be a public composite odd integer. Alice (resp. Bob) knows two $(k-2)$-bit integers p'_A, q'_A (resp. p'_B, q'_B) such that $n = (4(p'_A + p'_B) + 3)(4(q'_A + q'_B) + 3)$.

1. **Bob** publicizes encryptions $A = E_B(p'_B)$ and $B = E_B(q'_B)$. He then proves that A and B encrypt a $(k-2)$-bit integer by invoking Bound.
2. **Alice** computes, by using homomorphic properties, encryptions P_B, Q_B of respectively $p = 4(p'_B + p'_A) + 3$ and $q = 4(q'_B + q'_A) + 3$. By invoking Multiplication, she then computes an encryption N_B of pq.
3. **Alice** sends N_B and **Bob** decrypts it and he sends the decrypted value and proves the decryption with EncryptProof. If N_B does not encrypt n, the protocol **fails**.
4. **Alice** computes encryptions of $p-1$ and $q-1$ by using homomorphic properties and invokes Multiplication on these encryptions to compute an encryption of $(p-1)(q-1)$. By using homomorphic properties She computes and **outputs** an encryption Γ_B of $(p-1)(q-1)/2$, i.e. $2^{-1}(p-1)(q-1) \mod n_B$.

In the (Multiplication, Bound, EncryptProof)-hybrid model, the protocols Multiplication, Bound, EncryptProof can be replaced by a trusted party. Thus, the steps (1)+(2) and the step (4) can be seen as sub-protocols without direct interaction between parties (just with the trusted party). Moreover parties only receive encryptions (and "true" from the trusted party simulating Bound). Assuming S semantically secure, these sub-protocols are nPrivate. According to the composition property (see proposition 1), it suffices to prove that step (3) (interpreted as a sub-protocol) is nPrivate.

Let A be a polynomial adversary controlling Alice. In step 3, A can send any encryption N_B. However if N_B does not encrypt n, the protocol fails. This step can be seen as a restricted implementation of the nPrivate protocol GenFact, described in section 2, where A asks only one question, i.e. $n =^? D_B(N_B)$. It proves that Gamma is nPrivate against any polynomial adversary controlling Alice.

Let A be a polynomial adversary controlling Bob. In the same way, A can send encryptions of p''_B and q''_B such that $p''_B \neq p'_B$ or $q''_B \neq q'_B$. The protocol fails if N_B does not encrypt n. Thus Gamma can be also seen for A as a restricted implementation of GenFact, described in section 2, where A only asks if $n =^? (4(p_A + p''_B) + 3)(4(q_A + q''_B) + 3)$. According to lemma 1, Gamma is nPrivate against any polynomial adversary controlling Bob.

$\square$

It is easy to see that that Gamma is not UC-secure. Indeed, A can send an encryption N_B of p_B for instance. By doing this, A can learn the Bob's private value p_B but the protocol would fail in step 3. To avoid this, it suffices that Bob checks that $D_B(N_B) = n$ before to send $D_B(N_B)$.

3.2 Protocol RSAModulusTest

RSAModulusTest is used to check that a non-RSA modulus has not erroneously succeeded Boneh's test (because of a malicious behavior). Thus, RSAModulusTest simply consists of checking that n is an RSA modulus. The party 1, Alice by convention, wants to be convinced with a high probability that n is an RSA modulus: Alice (note that Bob does not output anything) outputs ok if n is an RSA modulus and otherwise RSAModulusTest fails (in RSAModulusGeneration, at the beginning of the execution of RSAModulusTest, n should be an RSA modulus if parties honestly behaved). RSAModulusTest implements the following test derived from the classical Boneh's test:

1. choose at random $a \in Z_n^*$ such that $(\frac{a}{n}) = 1$ and choose a random integer $\alpha \in \{n, n+1\}$
2. compute $v = a^{\alpha\gamma} \bmod n$ with $\gamma = (p-1)(q-1)/2$
3. If $v \neq 1$ then n is not an RSA modulus.

This test is less efficient than Boneh's test in the sense that a non-RSA modulus n has a larger probability of succeeding the test. Indeed, if $a^\gamma \neq 1 \bmod n$, the probability (over the choice of α) that $a^{\alpha\gamma/2} = 1 \bmod n$ is less than $1/2$ (because $a^{(n+1)\gamma/2} = a^{n\gamma}a^\gamma$). As the probability (over the choice of $a \in Z_n$ assuming that p, q are not Carmichael integers) that $a^\gamma \neq 1 \bmod n$ is larger than $1/2$ (see [4]), the probability (over the independent choices of α and a) that $a^{\alpha\gamma} \equiv 1 \bmod n$ is smaller than $3/4$.

In RSAModulusTest, $\alpha\gamma$ is shared by using homomorphic properties of the underlying homomorphic cryptosystem S. For concreteness, Alice has a secret encryption Γ_B of γ with $pq = n$. Alice and Bob jointly choose a random number a over Z_n^* such that $(\frac{a}{n}) = 1$. Then, she randomly chooses $\alpha \in \{n, n+1\}$, randomly chooses $s_1 \in [\frac{1}{8}\alpha n, \frac{3}{8}\alpha n] \bigcap \mathbb{N}$, computes $t_1 = a^{s_1} \bmod n$, commits it and sends an encryption of $s_2 = \alpha\gamma - s_1$.[7] Bob decrypts it and sends $t_2 = a^{s_2} \bmod n$. If $t_2 t_1 \neq 1 \bmod n$ the test fails. Let us suppose that n is not an RSA modulus and that an adversary A controlling Bob has guessed $\gamma \neq \lambda(n)/2$ before the execution

[7] Note that these numbers can be encrypted by E_B defined over Z_{n_B} because n_B is assumed to be large enough, i.e. a $5k$-bit integer.

of RSAModulusTest. As s_2 has been randomly chosen in a set statistically indistinguishable from $[\frac{1}{8}n^2, \frac{3}{8}n^2] \bigcap \mathbb{N}$, A is shown unable to choose $t_2 \in Z_n^*$, with probability larger than $3/4$, such that $t_1 t_2 = 1 \mod n$. This ensures correctness against any polynomial adversary controlling Bob. RSAModulusTest is trivially nPrivate against any polynomial adversary. Furthermore, RSAModulusTest can be seen, for Alice, as an instance of the generic protocol GenFact. It ensures that this protocol is nPrivate against any polynomial adversary controlling Alice (according to lemma 1).

Protocol 2 : RSAModulusTest

Require : Let k be a security parameter and $l \in \mathbb{N}$ be indexed by k. Let $n = pq$ be an odd composite integer where p and q are k-bit integers such that $p \equiv q \equiv 3 \mod 4$. Party 1 (Alice by convention) has a secret encryption Γ_B of $\gamma = (p-1)(q-1)/2$.

```
For $i = 1$ to $l$
    1. Alice and Bob jointly choose $a \in Z_n^*$ at random such that $(\frac{a}{n}) = 1$
    2. Alice randomly chooses $\alpha \in \{n, n+1\}$ and $s_1 \in [\frac{1}{8}\alpha n, \frac{3}{8}\alpha n] \bigcap \mathbb{N}$ and sends $U = E_B(\alpha\gamma - s_1)$ where $U$ is computed only by using homomorphic properties. She computes an encryption $T_1$ of $t_1 = a^{s_1} \mod n$ with the encryption function $E_A$, i.e. $T_1 = E_A(t_1)$ and sends it ($t_1$ is committed).
    3. Bob decrypts $U$, i.e $s_2 = D_B(U)$ and sends $t_2 = a^{s_2} \mod n$
    4. Alice sends $t_1 = D_A(T_1)$ and proves the decryption with EncryptProof.
    5. Alice and Bob compute $v = t_1 t_2 \mod n$. If $v \neq 1$ then the protocol fails.
End for
Alice outputs ok
```

Proposition 3. *Assuming the encryption scheme S is semantically secure, RSAModulusTest is correct and nPrivate against any polynomial adversary in the (EncryptProof)-hybrid model.*

Proof. First, let us note that the encryption domain Z_{n_B} of E_B was chosen large enough (n_B is a $5k$-bit integer and all the considered integers are at most $4k$-bit integers) to avoid modular reductions[8] (modulo n_B). Let A be a polynomial adversary controlling Bob. Let us prove that RSAModulusTest is **correct** against A. As the only possible output is ok, RSAModulusTest is intrinsically correct if n is an RSA modulus. Let us suppose

[8] It can be imagined that the encryption domain of E_B is $\mathbb{N}$.

that n is **not** an RSA modulus such that A knows its factorization, $\lambda(n)$ and $\gamma \neq \lambda(n)/2$. RSAModulusTest is correct if Alice outputs ok with a negligible probability. Let [9] $a \in Z_n^*$ s.t. $a^\gamma \neq 1 \mod n$. In order to make Alice output ok, A should guess, at each iteration, $t_1^{-1} \mod n$ (and send it to Alice at step 3). If A knows γ, it knows that

$$t_1^{-1} \in \{a^{s_2 - i\gamma} \mod n | i \in \{n, n+1\}\}$$

By assuming $a^\gamma \neq 1 \mod n$, this set contains two distinct elements m_1 and m_2. A receives a value s_2 such that $s_1 + s_2 = \alpha\gamma$ (the equality holds because the domain size Z_{n_B} of E_B is sufficiently large) where s_2 is a random number over a distribution statistically indistinguishable to the uniform distribution over $[\frac{1}{8}n^2, \frac{3}{8}n^2] \bigcap \mathbb{N}$: the statistical indistinguishability holds by noticing that $\gamma - n/2 = O(n^{\frac{1}{2}})$ (here, it is assumed that p and q are k-bit integers) is negligible implying that for any $\alpha \in \{n, n+1\}$

$$[\alpha\gamma - \frac{3}{8}\alpha n, \alpha\gamma - \frac{1}{8}\alpha n] \bigcap \mathbb{N} \overset{s}{\equiv} [\frac{1}{8}n^2, \frac{3}{8}n^2] \bigcap \mathbb{N}$$

Consequently, A cannot distinguish between $\alpha = n$ or $\alpha = n+1$ implying that it cannot distinguish between $t_1^{-1} \equiv m_1 \mod n$ or $t_1^{-1} \equiv m_2 \mod n$ with probability significantly larger than 1/2. The probability to choose $a \in Z_n^*$ such that $a^\gamma \neq 1 \mod n$ is larger than 1/2 see [4][10]. Therefore the probability that A guess $t_1^{-1} \mod n$ is smaller than 3/4. Also, the probability to do this at each iteration is smaller than $(3/4)^k$.

RSAModulusTest is trivially nPrivate against A. Indeed, the view of A (when the protocol does not fail) can be simulated by a list of k independent triplets of values (T_1, a^{-s_2}, s_2) where T_1 is randomly chosen in the ciphertext domain of E_A, s_2 is chosen at random in a set statistically indistinguishable from $[\frac{1}{8}n^2, \frac{3}{8}n^2] \bigcap \mathbb{N}$. Assuming the semantic security of S, the simulation and the real view are computationally indistinguishable.

Let A be a polynomial adversary controlling Alice. Correctness is implicit because Bob does not output anything. RSAModulusTest can be seen, for A, as a restricted implementation of GenFact, described in section 2, where the k questions are $a^{s_2} \equiv^? t_1^{-1} \mod n$ (the commitment of t_1 is required to get this). According to lemma 1, RSAModulusTest is nPrivate against any polynomial adversary controlling Alice.

$\square$

[9] Throughout this paper, we neglect the probability that n is a Carmichael modulus.
[10] By supposing that n is not a Carmichael RSA modulus.

3.3 Protocol RSAModulusGeneration

The following protocol RSAModulusGeneration implements the functionality described in section 2. As mentioned in section 2, ModulusGeneration denotes a stand-alone secure (in the malicious case) protocol computing $n = (p_A + p_B)(q_A + q_B)$. This protocol is used to build an RSA modulus candidate $n = pq$. The parties then use the classical Boneh's test to test if n is an RSA modulus. If not, the parties exchange the current values p_A, p_A, p_B, q_B (which do not need to be kept secret anymore) and check that the other party behaved honestly before generating an other modulus n (by executing a new iteration of the loop). If Boneh's test claims that n is an RSA modulus, the parties then invoke Gamma to get an encryption of $(p-1)(q-1)/2$. They then check that n is really an RSA modulus by executing RSAModulusTest twice (exchanging their positions each time). The idea is to execute RSAModulusTest only if n is an RSA modulus or if the adversary did not behave honestly. RSAModulusTest will distinguish these two cases. Note that either RSAModulusGeneration outputs the first RSA modulus n_0 output by ModulusGeneration in step 1.a.: it will be used in the security analysis of RSAModulusGeneration.

Theorem 1. *Let A be a polynomial adversary controlling Alice or Bob. Assume that the encryption scheme S is semantically secure, the following assertions are true in the (CorrectKey, EncryptProof, Bound and Multiplication)-hybrid model.*

1. *RSAModulusGeneration is correct against A, i.e. the probability that the output n is not an RSA modulus is negligible.*
2. *Assuming conjecture 1, the probability that the protocol does not fail and that A factors the output n is negligible.*

Proof. Let k be the security parameter. Let us suppose that Bob is honest and that Alice is controlled by an active adversary A.

ModulusGeneration is assumed to correctly compute $n = (p_A+p_B)(q_A+q_B)$. If n is not an RSA modulus and A cheats Boneh's test in step (1.b), the protocol fails in step (4) according to proposition 1. Thus, RSAModulusGeneration cannot output a non-RSA Modulus. If p_A or q_A is not a k-bit size integer, if $p_A \not\equiv 3 \mod 4$ or if $q_A \not\equiv 3 \mod 4$ then the protocol fails in step 2. Thus, the output is a well-formed RSA modulus. It proves correctness.

Let us show that either the first RSA modulus generated by ModulusGeneration is output or RSAModulusGeneration fails. Let $n_0 = pq = (p_A + p_B)(q_A + q_B)$ be the **first** (by assuming the protocol has not failed

Protocol 3 : RSAModulusGeneration

Require : Let k be a security parameter. Alice (resp. Bob) generates an encryption function E_A (resp. E_B) defined over Z_{n_A} (resp. Z_{n_B}) where n_A (resp. n_B) is a $5k$-size integer. Parties invoke CorrectKey to prove correctness of keys.

1. *While Boneh's test fails*
   (a) **Alice and Bob** randomly choose k-bit numbers, respectively p_A, q_A and p_B, q_B such that $p_A \equiv q_A \equiv 3 \mod 4$ and $p_B \equiv q_B \equiv 0 \mod 4$. They then invoke ModulusGeneration to get $n = (p_A + p_B)(q_A + q_B)$.
   (b) **Alice and Bob** invoke Boneh's test to decide if n is an RSA modulus or not.
   (c) if n failed Boneh's test then parties broadcast p_A, q_A, p_B, q_B and verify that $n = (p_A + p_B)(q_A + q_B)$ and $p_A + p_B$ or $q_A + q_B$ are not prime. If n is an RSA modulus then the protocol fails.

   End while
2. **Alice and Bob** invoke Gamma twice (by exchanging their position) on the private inputs $((p_A - 3)/4, (q_A - 3)/4)$, $(p_B/4, q_B/4)$ and the public input n. Alice gets the encryption Γ_B and Bob gets the encryption Γ_A.
3. **Alice** invokes RSAModulusTest on the input n, Γ_B (Alice is party 1)
4. **Bob** invokes RSAModulusTest on the input n, Γ_A (Bob is party 1)
5. If **Alice and Bob** have output ok in the previous steps, n **is output.**

before) RSA modulus output by ModulusGeneration in loop 1. If A cheats Boneh's test in step (1.b) then the protocol fails in step (1.c) else the step 2 is executed. If p or q are not k-bit size integers then the execution of Gamma makes RSAModulusGeneration fails in step 2. As RSAModulusTest is correct, either n_0 is output or the protocol fails in step 4. Thus, the output RSA modulus n_0 is randomly chosen such that $p \equiv q \equiv 3 \mod 4$, $p \in [p_A + 2^k, p_A + 2^{k+1}[\bigcap \mathbb{N}$ and $q \in [q_A + 2^k, q_A + 2^{k+1}[\bigcap \mathbb{N}$ where p_A, q_A are two arbitrary k-bit integers (chosen by the adversary). According to conjecture 1 and assuming that ModulusGeneration is secure (in the stand-alone setting), A cannot factor n_0, with non negligible probability, at the end of step 1.a.. Indeed, in the ideal case (see [17]) an adversary A_I knows n, p_A, q_A and in the real case the adversary A knows $n_0, z_{n_0,A}$ ($z_{n_0,A}$ being the view of A in ModulusGeneration). Indeed, according to conjecture 1, there there does not exists any polynomial adversary A_I able to factor n_0 with non negligible probability in the ideal case (given n_0, p_A, q_A). Thus, if there exists a polynomial adversary A able to factor n_0 with non negligible probability, then the real view and the ideal view can be distinguished with non negligible probability: it contradicts that ModulusGeneration is secure. Thus, such an adversary A does not exist. In other words, by denoting D_k the distribution of $n_0, z_{n_0,A}$, the family of distributions $(D_k)_{k \in \mathbb{N}}$ is nPrivate. Boneh's test is nPrivate and according to proposition 2 and 3, Gamma and RSAModulusTest are nPrivate. According to the composition property (see proposition 1), the sub-protocols composed by steps (1b.)(2),(3) and (4) is nPrivate. According to the definition 2, A cannot output the factorization of n_0 without making the nPrivate sub-protocol composed of the steps (1.b.)(2),(3),(4) (thus RSAModulusGeneration) fail with non negligible probability.

$\square$

Efficiency Analysis. Classical Protocols generating RSA moduli correspond to the steps (1a) (1b) of RSAModulusGeneration. The steps (1c) (2), (3), (4) were introduced in order to make the protocol robust against active adversaries. Let us evaluate the supplementary cost of these steps.

In a first analysis, we only take into account modular exponentiations, encryptions and decryptions (ME/E/D) that Alice (and symmetrically Bob) must compute (neglecting, for instance, modular multiplications). The underlying encryption scheme S is assumed to be the Paillier cryptosystem. The number of iterations of the loop is approximatively equal to $\log^2 n$ on average. As (1c) consists of doing $\log^2 n$ primary tests, the cost of (1c) is approximately equal to $\log^2 n$ exponentiations. However, (1c) should be added in all classical Boneh's test protocols to not allow an adversary to cheat Boneh's test on RSA moduli. Furthermore, (1c) does not need to be executed at each iteration but only with a given probability.

Gamma and RSAModulusTest are executed twice. Gamma requires $O(\log n)$ ME/E/D (assuming Bound requires $O(1)$ ME/E/D) and RSAModulusTest requires $O(1)$ ME/E/D. Thus, steps (2),(3),(4) can be neglected with respect to step (1) (which requires $O(\log^2 n)$ ME/E/D) when $\log n$ is large. Let us detail our analysis.

In Gamma, the protocol Bound is invoked twice. For classical security level, approximatively 10^3 ME/E/D are required. Without any optimization, the total running time of steps (2), (3), (4) is a few minute on a standard platform.

However, as far as we know, there does not exist two-party really efficient Boneh's test based protocols (while there exists such efficient protocol in the multi-party case [16]): hours are required to output a 1024-bit RSA modulus. It is very challenging to find such a protocol.

Complementary approaches to reach efficiency consist of modifying the distribution of the inputs (see [15]). For instance, it should be asked to Alice to choose p_A, q_A as multiples of some small primes and it should be asked to Bob to choose p_B, q_B as multiples of **other** small primes. By doing this $p_A + p_B$ and $q_A + q_B$ are not multiple of the involved small primes increasing their probability to be prime.

4 Conclusion and future work

We first showed that existing two-party protocols based on Boneh's test are not secure against actives adversaries. In this paper, we have provided a provably secure (with security requirements dedicated to our application) protocol to solve this problem. We propose a simple and efficient "patch" making most Boneh's test-based protocols correct against any polynomial active adversaries. We develop *ad-hoc* security tools to prove that a polynomial adversary cannot learn the factorization of n without making the protocol fail. These new security tools could be re-used for other applications. A natural extension would consist of considering the multi-party case in presence of an arbitrary number of misbehaving parties.

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*, pages 417–432, 2002.
2. Simon R. Blackburn, Simon Blake-Wilson, Mike Burmester, and Steven D. Galbraith. Weaknesses in shared rsa key generation protocols. In *IMA Int. Conf.*, pages 300–306, 1999.
3. D. Boneh and M.K. Franklin. Efficient generation of shared rsa keys (extended abstract). In *CRYPTO*, pages 425–439, 1997.
4. D. Boneh and M.K. Franklin. Efficient generation of shared rsa keys. *J. ACM*, 48(4):702–722, 2001.
5. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
6. R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.
7. I. Damgård and K. Dupont. Efficient threshold rsa signatures with general moduli and no extra assumptions. In *Public Key Cryptography*, pages 346–361, 2005.
8. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136, 2001.
9. Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In *EUROCRYPT*, pages 152–165, 2001.
10. Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed rsa key generation. In *TCC*, pages 183–200, 2010.
11. P. Fouque and J. Stern. Fully distributed threshold rsa under standard assumptions. In *IACR Cryptology ePrint Archive: Report 2001/2008*, February 2001.
12. N. Gilboa. Two party rsa key generation. In *CRYPTO*, pages 116–129, 1999.
13. Carmit Hazay, Gert Lsse Mikkelsen, Tal Rabin, and Tomas Toft. Efficient rsa key generation and threshold paillier in the two-party setting. Cryptology ePrint Archive, Report 2011/494, 2011. http://eprint.iacr.org/.
14. Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *CRYPTO*, pages 36–54, 2000.
15. Daniel Loebenberger and Michael Nüsken. Analyzing standards for rsa integers. In *AFRICACRYPT*, pages 260–277, 2011.
16. Michael Malkin, Thomas D. Wu, and Dan Boneh. Experimenting with shared generation of rsa keys. In *NDSS*, 1999.
17. O.Goldreich, S.Michali, and A.Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
18. G. Poupard and J. Stern. Generation of shared rsa keys by two parties. In *Asiacrypt'98, LNCS 1514, Springer-Verlag*, pages 11–24, 1998.
19. Tal Rabin. A simplified approach to threshold and proactive rsa. In *CRYPTO*, pages 89–104, 1998.
20. Victor Shoup. Practical threshold signatures. In *EUROCRYPT*, pages 207–220, 2000.
21. T. Straub. Efficient two party multi-prime rsa key generation. In *IASTED International Conference on Communication, Network, and Information Security*, 2003.
22. P. MacKenzie Y. Frankel and M. Yung. Robust efficient distributed rsa key generation. In *STOC'98*, pages 663–672, 1998.

A Attack on Boneh's test

Boneh and Franklin [3] have proposed a test to decide if $n = pq$ is an RSA modulus assuming that $p \equiv q \equiv 3 \mod 4$. This test consists of randomly choosing $a \in Z_n^*$ such that $(\frac{a}{n}) = 1$, computing[11] $v = a^{(p-1)(q-1)/2} \mod n$ and checking if $v \neq 1$. For a value n which passes the test t times, n is not an RSA modulus with probability smaller than 2^{-t}. This test is not a complete probabilistic test in the sense that there are some integers n, which are not RSA moduli, but which succeed the test for any basis a. However, this probability can be considered as negligible (around 10^{-60}) for the number sizes considered in our application. In existing protocols [4], [12], [18], [21], Alice and Bob generate $n = (p_A + p_B)(q_A + q_B)$ with a secure protocol ModulusGeneration where $p_A \equiv q_A \equiv 3 \mod 4$ and $p_B \equiv q_B \equiv 0 \mod 4$. Alice then computes $s_A = (n - p_A - q_A + 1)/2$ and Bob computes $s_B = -(p_B + q_B)/2$ in order to additively share $(p-1)(q-1)/2$, i.e. $s_A + s_B = (p-1)(q-1)/2$. Finally, Alice computes $a^{s_A} \mod n$, Bob computes $a^{s_B} \mod n$ and they check that $a^{s_A}a^{s_B} = 1 \mod n$. This test is secure against passive adversary. However, this protocol may be insecure against an active adversary because there is nothing to ensure that parties send the correct value $a^s \mod n$ in Boneh's test. To secure the protocol, each party should prove that it sends $a^s \mod n$. Classical methods (based on zero-knowledge proofs which ensure two values $a^s \mod n$ and $g^s \mod n$ have the same discrete logarithm s) cannot be used because the shared values s are chosen before n (they are used to build n). In the next section, an attack in a dissymmetric environment is presented and evaluated (note that nothing excludes that other attacks exist, even in a symmetric environment).

A.1 Presentation of a brute force attack

This attack works in a dissymmetric environment where one party has more resources than the other.[12] Let us suppose that an active polynomial adversary A controls Alice. We suppose that A has more resources than Bob. We denote the ratio of resources between Alice and Bob by $r > 1$.

[11] The original Boneh test consists of checking if $a^{(p-1)(q-1)/4} = \pm 1 \mod n$. This formulation is equivalent to ours. Indeed, as it is assumed that $p \equiv q \equiv 3 \mod 4$, 1 and -1 are the two square roots of 1 with a Jacobi symbol equal to 1. Thus, $a^{(p-1)(q-1)/2} = 1 \Leftrightarrow a^{(p-1)(q-1)/4} = \pm 1$.

[12] The "power " party ressource can be, for instance, only linear with respect to the resource of the other. It is obviously assumed that the "power" party is not able to factor output RSA moduli, i.e. 1024-bit RSA moduli.

Typically, A computes r times faster than Bob. In Boneh's test, each party computes a modular exponentiation. We denote the time needed by Bob to compute the modular exponentiation, by t_e. A can try to factorize n during $(1 - 1/r)t_e$ time units and keeps t_e/r time units to compute a modular exponentiation. Let us suppose that $q = q_A + q_B$ is prime and that Alice manages to decompose n in prime factors, i.e. $n = p_1 p_2 ... p_m q$. This situation can be identified by A with probability $1/2$ (A cannot distinguish it from the symmetric case where $p = p_A + p_B$ is prime) because $|q| = k > |p_i|$. Thus, A can deduce $p_B = p_1 p_2 ... p_m - p_A$ and $q_B = q - q_A$. Consequently A can compute Bob's secret share s_B. By sending $a^{-s_B} \mod n$ instead of $a^{s_A} \mod n$, Boneh's test succeeds and Bob is convinced that n is an RSA modulus. A first naive approach would consist of letting time t_B to Bob to test if n is a RSA modulus after the Boneh's test. In the next section, we experiment with this type of attack against this naive approach.

A.2 Concrete scenario of the attack

In this section, Bob's simulations were implemented on a standard PC (Pentium 2 GHz) with the java class *BigInteger*. The factorization algorithm is ECM. This algorithm is particularly efficient for discovering *small* factors in large integers. In a sense this algorithm is very relevant for Bob's problem which consists of checking whether n is an RSA modulus or not. In our simulations, A computes on the same platform but the obtained computations times are divided r.

In order to correctly evaluate this attack, we should evaluate the probability of getting "malicious" values of n. A malicious $n = p_1 ... p_m q$ can be factored in a small amount of time by A while Bob cannot find any factor p_i. Typically, $p_1, .., p_m$ should be large enough but not too large. In the following experiment, we simulate the attack with the following parameters $|n| = 1024$, $r = 10^5$ and $t_e = 100ms$ (this is approximately the time that Bob needs to compute a modular exponentiation considering 1024-bit size integers). We denote the time available to Bob after applying Boneh's test in order to check if n is an RSA modulus by t_B. We wish to estimate the probability of finding n such that A is able to factorize it in $(1 - 1/r)t_e \approx t_e$ time units and such that Bob is not able to find a factor within t_B time units. In our experiments, if $|p_i| \in \{80, ..., 110\}$ (and $|q| = 512$) then A is able to factorize n within $t_e = 100ms$ (3h with our platform) and Bob needs at least 3mins to find a factor.

Now, let us evaluate the ratio between the probability of getting an RSA modulus and the probability of finding such malicious values of n.

This ratio provides a lower bound of the probability to output a non-RSA modulus. A short analysis [13] based on prime theorem shows that this ratio is approximately equal to 10^{-4}.

To summarize, if the ratio of performance between Bob and A is larger than 10^5 and if Bob is given less than 3 minutes after applying Boneh's test in order to verify whether n is an RSA modulus, then the adversary is able to convince Bob that a non-RSA modulus is a RSA-modulus with probability larger than 10^{-4}.

In this paper, we propose a *"patch"* to the classical Boneh's test in order to ensure that a value n is an RSA modulus against any polynomial adversary, in particular against a more powerful adversary (whatever the performance ratio r is). The computation of this patch requires only few seconds.

B Protocol Multiplication

MultProof. A party builds 3 encryptions X, Y, Z from 3 values x, y, z such that $z = xy$. MultProof is a $\Sigma-$ protocol which proves that Z encrypts xy without revealing anything about x and y.

A version of MultProof is dedicated to Paillier's encryption scheme. It can be found in [6] (generic versions can also be found). We have as inputs encryption $C_x = g^x r^n \mod n^2$, $C_y = g^y s^n \mod n^2$, $D = C_x^y \gamma^n \mod n^2$ and a player P_i who knows in addition s, y, γ. What we need is a proof that D encrypts $xy \mod n$. We proceed as follows:

1. P_i chooses $a \in Z_n$ and $b, c \in Z_n^*$ at random, computes and sends
$$A = C_x^a b^n \mod n^2, B = g^a c^n \mod n^2$$
2. The verifier sends a random challenge e
3. P_i computes and sends
$$w = a + ey \mod n, z = cs^e g^t \mod n^2, z' = bC_x^t \gamma^e \mod n^2$$
where t is defined by $a + ey = w + tn$.

[13] We assume that the number of primes smaller than any integer t is equal to $t/\log t$. By using this approximation, we can estimate the number P_k of primes of size k. To estimate the number of 512-bit size integers $p = p_1 ... p_m$ with $|p_i| \in \{80, ..., 110\}$, it suffices to consider each possible value of $7 \geq m \geq 5$, each possible size (without taking into account permutations) of $p_1, p_2, ..., p_m$ belonging to $\{80, ..., 110\}$. Given m and $|p_1|, ..., |p_m|$, it is easy to count the numbers of 512-bit size integers p knowing $P_{|p_1|}, ..., P_{|p_m|}$. By considering all size configurations, we approximately obtain the number of 512-bit size integers which can be written as product of $\{80, ..., 110\}$-bit size primes p_i.

4. The verifier checks that

$$g^w z^n = BC_y^e \mod n^2, C_x^w y^n = AD^e \mod n^2$$

and accepts if and only if it is the case.

This zero-knowledge proof is shown secure under composition. Based on such proofs, a secure protocol Multiplication can be easily built.

Multiplication

Require : Alice has two encryptions $E_B(x), E_B(y)$ of unknown values $x, y \in Z_{n_B}$

1. **Alice** sends $E_B(r+x), E_B(s+y)$ where r and s are random numbers chosen in Z_{n_B}
2. **Bob** computes and sends $Z = E_B((x+r)(y+s))$. He then proves that Z encrypts the correct value with MultProof.
3. **Alice** computes an encryption of $(x+r)(y+s)-sx-ry-rs$ by using homomorphic properties and outputs it.

C Proof of proposition 1

Let $D = (D_i)_{i\in\mathbb{N}}$ be an nPrivate probability distribution family (see definition 1) and k be a security parameter. Let A be a polynomial adversary and $(n, z_n) \leftarrow D_k$. Without loss of generality, an adversary A in P can be decomposed in two adversaries A_R, A_S, denoted by $A = A_R \to A_S$ as follows: A_R receives $(n, z_n) \leftarrow D_k$ before the execution of R, outputs $(n, z_{n,A,R})$ and sends it to A_S which starts the protocol S given $(n, z_{n,A,R})$.

Let us change the rules of interaction with the adversary A. If R fails, A is given "another chance" by letting it execute S but by deleting the information it learns about n. More formally, this can be interpreted as the interaction between party 1 and a new polynomial adversary $A' = A'_R \to A_S$ in a new composed protocol $P' = R' \to S$: R' is the same protocol as R except that the A'_R outputs z_n (the honest party outputs an arbitrary value) when R fails and A'_R outputs $z_{n,A,R}$ otherwise (note that R' never fails). By summary

$$z_{n,A',R'} = \begin{cases} z_{n,A,R}, & \text{if } R \text{ does not fail;} \\ z_n, & \text{otherwise.} \end{cases}$$

It is clear that the probability that A' outputs the factorization of n without making P' fail is larger than the probability that A outputs the factorization of n without making P fail. Let us show that this probability is negligible. Let D'_k be the probability distribution of $(n, z_{n,A',R'})$. As S is $\mathsf{nPrivate}$, it suffices to prove that $(D'_k)_{k\in\mathbb{N}}$ is $\mathsf{nPrivate}$.

Let us suppose that it is not the case and that there exists a p.p.t algorithm F able to factor n, with non negligible probability, given $n, z_{n,A',R'}$. This implies that one of the two following issues is satisfied with non negligible probability:

1. F factors n given n, z_n
2. F factors n given $n, z_{n,A,R}$ and R does not fail.

If F cannot factor n, with non negligible probability, given n, z_n because $(D_k)_{k\in\mathbb{N}}$ is $\mathsf{nPrivate}$. Let A'' be a polynomial adversary behaving like A in R and applying F on $n, z_{n,A,R}$. Issue 2 implies that the probability that A'' factors n without making R fail is not negligible. Thus, issue 2 cannot hold assuming that R is $\mathsf{nPrivate}$. It proves that $(D'_k)_{k\in\mathbb{N}}$ is $\mathsf{nPrivate}$.