Int. J. Appl. Math. Comput. Sci., 2023, Vol. 33, No. 1, 57–70
DOI: 10.34768/amcs-2023-0005

IMPLICATIONS OF THE ARITHMETIC RATIO OF PRIME NUMBERS FOR RSA SECURITY

ANDREY IVANOV [a], NIKOLAI STOIANOV [a,*]

[a]C4I Systems
Professor Tsvetan Lazarov Bulgarian Defence Institute
2 Prof. T. Lazarov Blvd., 1592 Sofia, Bulgaria
e-mail: andry.g.ivanov@gmail.com, n.stoianov@di.mod.bg

The most commonly used public key cryptographic algorithms are based on the difficulty in solving mathematical problems such as the integer factorization problem (IFP), the discrete logarithm problem (DLP) and the elliptic curve discrete logarithm problem (ECDLP). In practice, one of the most often used cryptographic algorithms continues to be the RSA. The security of RSA is based on IFP and DLP. To achieve good data security for RSA-protected encryption, it is important to follow strict rules related to key generation domains. It is essential to use sufficiently large lengths of the key, reliable generation of prime numbers and others. In this paper the importance of the arithmetic ratio of the prime numbers which create the modular number of the RSA key is presented as a new point of view. The question whether all requirements for key generation rules applied up to now are enough in order to have good levels of cybersecurity for RSA based cryptographic systems is clarified.

Keywords: public key cryptography, RSA encryption, public key generation rules, kleptography, fusion of number balance.

1. Introduction

Created by Rivest *et al*. (1978), RSA is the first public key cryptographic algorithm which continues to be the most commonly used today. Up to March 2022 more than 85% from certificate authorities (CAs) guarantee their root certificate's security by using the RSA encryption and signing scheme. Approximately 10% of CAs combine both RSA and the elliptic curve digital signature algorithm (ECDSA) to protect their public key infrastructures (PKI). This statement is based on our research, in which we analyzed the root certificates stored in operating systems (OSs) Windows, Android, and Linux. These OSs are the most commonly used in the world. We used in our research statistical results provided by the Certificate Transparency Organization. This is why RSA obviously continues to play a key role related to the information security of our electronic resources, which is why research on the cryptographic resilience of RSA is so important. The correctness of public key generation process and key management directly affects the resilience of cryptographic systems (Dodis *et al*., 2004; Alwen *et al*., 2009). All this makes key generation pairs for the RSA algorithm very significant about the security of information technologies that people use.

Over the years, many researchers have worked on the tasks of finding vulnerabilities or creating mathematical attack models to break the RSA-based cryptographic systems, but so far no attack approach has been developed which can break the RSA key with length of 2048 bits or bigger in acceptable time. The algorithm which implements key pair generation for RSA can be described as follows. Two different prime numbers p and q with a size approximately equal to $nBits/2$ each are generated, where $nBits$ is the desired key length in bits (for example, $nBits = 1024$). We denote the result of their multiplication with the number N ($N = p \cdot q$). Choose a small integer e and calculate $d \equiv e^{-1} \mod \varphi(N)$. This means that the numbers e and d satisfy the congruence

$$e \cdot d \equiv 1 \mod \varphi(N),$$

where $\varphi(N)$ is Euler totient function (Kaliski, 2011). In the case when p and q are prime numbers, the result of that Euler function is $\varphi(N) = (p-1) \cdot (q-1)$ that is the

*Corresponding author

order of the multiplicative group Z_N^*. We call N the RSA modulo number, e the exponent for encryption, and d the exponent for decryption. The pair (d, N) is the private key and pair (e, N) is the public key.

When a sender has to encrypt a message M, he/she has to use the recipient's public key. The mathematical operation

$$C \equiv M^e \mod N \tag{1}$$

is used as the encryption operation, where C is the encrypted message. To decrypt the cyphertext C, the recipient has to compute

$$C^d \equiv (M^e)^d \equiv M^{e \cdot d} \equiv M^1 \equiv M \pmod N, \tag{2}$$

where the last equality follows by Euler's theorem (Kaliski, 2011).

Other algorithms in public key cryptography widely used to protect information are based on DLP (Gordon, 2011; Adj *et al*., 2018) and ECDLP (Yan, 2019; Ahlswede, 2016). Signing data is a field in cryptography which has a very important role for information security. Such algorithms that implement this are ECDSA and the digital signature algorithm (DSA) (Menezes *et al*., 1996). Nowadays, many new approaches aim to improve digital signature algorithms and signing schemes. One of these new ways is the group signature scheme in the blockchain (Devidas *et al*., 2021).

The difference between RSA and the others is that in RSA large composite numbers are used when modulus operations have to been calculated (Yasuda *et al*., 2012). Among the most widely used cryptographic systems based on DLP are the Diffie–Hellman key exchange protocol (Diffie and Hellman, 1976), the El Gamal public key cryptosystem (Elgamal, 1985), signature schemes (Sako, 2011), etc. The mathematical challenge in DLP is based on computing discrete logarithms in finite fields of type Z_p which consist of integers obtained by computing modulo using a large prime number p. Given $g, h \in G_p$, and p (prime number) the Discrete Logarithm Problem consists in finding x (if it exists) such that

$$g^x \equiv h \mod p, \tag{3}$$

where G_p is a multiplicative Abelian group with generator g.

All used public key cryptographic algorithms are implemented based on the use of at least one secret parameter. In RSA-based systems, this secrecy is achieved by the inability to calculate d by the equation

$$e \cdot d + \varphi(N) \cdot i = 1,$$

where e and N are known but $\varphi(N)$ and i are difficult to calculate or find by exhaustive search.

2. General attacks on RSA

Over the years, the attacks against systems using RSA can be divided into three types. The first is based on a side channel attack, the second on incorrect generation of the private exponent and the third on decomposition of large numbers into prime factors. To overcome the weaknesses falling into the first two types, many additions have been made into the systems which use RSA. Due to that reason we have focused on exploring the possibility of attacks based on improperly generated p and q numbers or efficient factorization of modulus N which is part of the RSA public key.

Factorization algorithms can be divided into two groups: special-purpose and general-purpose. The first group consists of algebraic-group factorization algorithms, Fermat's or Euler's methods. The group of general-purpose factoring algorithms, also known as Kraitchik's family algorithms (after Maurice Kraitchik) (Bressoud and Wagon, 2000) has running time which depends solely on the size of the integer to be factored. The group of general-purpose algorithms includes the quadratic sieve (QS), the general number field sieve (GNFS), Shanks's square forms factorization, etc.

The most general-purpose factoring algorithm are based on the Fermat factorization method. In 1643 Fermat (Erra and Grenier, 2009) exposed an algorithm to factor odd integers by writing them as a difference of two squares. Fermat factorization is an iterative search method. Suppose that an integer N is a composite odd number that is not a perfect square. There is a one-to-one correspondence between factorization of $N = p \cdot q$, where $p > q > 0$ and the representation of N in the form $t^2 - s^2$, where s and t are non-negative integers. To execute the Fermat factorization algorithm for values of $t \geq \lfloor \sqrt{(N)} \rfloor + 1$, we compute $t^2 - N$ until we obtain a perfect square of s: $t^2 - N = s^2$.

The quadratic sieve algorithm was invented by Carl Pomerance in 1981 as an improvement to Schroeppel's linear sieve (Pomerance, 1982). The algorithm tries to create a congruence of number x squared modulo N that often leads to a factorization of N. The QS works in two main phases of execution. The first one has to prepare data that may lead to a congruence of squares and will be used in the second phase. The second phase puts all the data which were collected into a matrix and employs it to obtain a congruence of squares. The first phase allows the collecting process to be executed in parallel, but the second phase requires large amounts of memory, and is difficult to be parallelized efficiently.

How can we use congruence of squared numbers modulo N to factorize it? If we find at least one non-negative remainder of number x satisfying congruence

$$x^2 \equiv y^2 \mod N,$$

we know that

$$x^2 - y^2 \equiv 0 \mod N.$$

This means

$$(x - y) \cdot (x + y) = N.$$

Hence we can calculate the factors of N by computing $q = \text{GCD}(x - y, N)$ and $p = \text{GCD}(x + y, N)$ (if $q < p$).

The most important advantage of QS over the Fermat factorization method is the fact that the square of a congruence is more likely to find in a multiplicative Abelian group G_N than to find integers s and t which satisfy $t^2 - N = s^2$. We can demonstrate how QS works with an example.

Example 1. (*Quadratic sieve*) We have

$$330^2 \equiv 570 \mod 3\,611,$$
$$510^2 \equiv 108 \mod 3\,611,$$
$$196^2 \equiv 2\,036 \mod 3\,611.$$

None of the integers 570, 108 and $2\,306$ is a perfect square, but $570 \cdot 2\,306 \equiv 330^2 \cdot 196^2 \equiv 4^2 \mod 3\,611$ It follows that

$$(196 \cdot 330)^2 - 4^2 = k \cdot 3\,611.$$

Hence

$$k \cdot 3\,611 = (64\,680 - 4) \cdot (64\,680 + 4)$$
$$= 64\,676 \cdot 64\,684.$$

This is means

$$3\,611 = \text{GCD}(64\,676, 3\,611) \cdot \text{GCD}(64\,684, 3\,611),$$
$$3\,611 = 23 \cdot 157.$$

♦

We looked at the classical possible ways to attack public cryptography based on RSA, but there is a kind of attack which could be made throughout using a back door in the key generation process. To perform that kind of attack, an attacker has to modify cryptographic primitives in a such way that this could allow him to break the user key using the only available victim's public key.

Anderson (1993) proposed a back door in the generator of the 512 bit RSA key. The security of the leaking information throughout the hidden channel was estimated as weak because Kaliski (1993) proposed a method for breaking such kind of back door.

Young and Yung (1996) introduced the concept of SETUP (secretly embedded trapdoor with universal protection) attacks. They defined kleptography as an extension to the SETUP concept. Using that mechanism, important secret information leaks throughout a hidden channel to the attacker. The attacker can use that information to factorize efficiently the modulus number N from the user's key.

Algorithm 1. SETUP attack on RSA.

Require: $(g, Y, P), m, K, W, e$ {e is a generated key public exponent}
1: $c_1 := randombits(\lfloor m/2 \rfloor)$
2: $z := g^{c_1 - W} \cdot Y^{-a \cdot c_1 - b} \mod P$
3: $l := H(z)$
4: $l := l \; or \; 1$
5: $numb := -2$
6: **repeat**
7: $numb := numb + 2$
8: $p := l + num$
9: **until** p is prime
10: $v := g^{c_1} \mod P$
11: **for** $i = 0;\ i < B_2;\ i++$ **do**
12: $U := G(v, K + i)$
13: $RND := randombits(\lfloor m/2 \rfloor)$
14: $T := [U][RND]$ {$[U][RND]$ is bit concat.}
15: $q := (T - r)/p$
16: **if** q is prime **then**
17: break
18: **end if**
19: **end for**
20: $N := p \cdot q$
21: $d := e^{-1} \mod \varphi(N)$
22: **return** N, e, d

To perform a SETUP attack, the kleptographic mechanism has to be embedded into hardware device firmware or into a computer software product by a developer or a person who participates in the development of the cryptographic algorithm; such a person we will call the attacker. According to the proposed SETUP attack, the attacker uses his/her own public key (g, Y, P) to protect important hidden information which is a part of the generated user's key, where number P is a prime with size $\lfloor m/2 \rfloor$ and m is the key size which have to be generated. The attacker keeps his/her private key x secret. In the algorithm execution, a function G is used. The result of the function is pseudorandom $a = G(b, c)$. This means that, when applied to the data b using the key c, a value of a with a binary size equal to or greater than P is produced. A non-secret symmetric key K is used in the algorithm. That key is a part of the device firmware or a computer software application. A hash function H whose binary size result has to be greater than P is used in the algorithm. The algorithm works with two constant parameters B_1 and B_2 which have to limit execution of some operations.

The pseudocode of the Young and Yung algorithm (YY97) is shown as Algorithm 1. We have to mention that the pseudocode steps correspond to the Young and Yung

algorithm described more descriptively in words in their publication (Young and Yung, 1997).

We have to stress that the values of a, b and W in row 2 of the above algorithm are not pointed in Young and Yung's publication (cf. Section 5). This means, that the values of a, b and W are constants and they are part of the attacker's private key; otherwise, it is impossible the attack successfully.

The maximum upper value of the range r (row 15 in Algorithm 1) is not pointed in Algorithm YY97. In Step 12 of the YY97 algorithm (solve for q in the equation $[U][\text{RND}] = pq + r$), r is an iterative variable. In that case the YY97 speed execution depends on the count of cycles limited from B_1, B_2 and r. In Appendix of the paper by Young and Yung (1997) it is shown that the average time of the 1024 bit public key generation is approximately 154.4 seconds by using GNU MP library version 1.3.2. The average time to generate such a key on a PC is approximately 2 seconds. Based on that argument, we could state that the algorithm of Young and Yung cannot be used practically and could not be embedded in RSA cryptographic systems. We state this because any user of a hardware device or a software cryptographic library will notice that key generation is slow and would be suspicious that something is wrong. That would be the reason which will lead to revealing the embedded Kleptographic algorithm.

At present days some new Kleptographic algorithms are created. They present good ideas and demonstrate the danger power of Kleptography could be. One of that algorithms was presented by Markelova (2021). She presented ideas of SETUP attacks on RSA in two variants. One of them is by using of elliptic curves as basis for the attack.

3. Kleptography attack on RSA by a new approach

In this part of the article, an idea for a new kind of kleptographic algorithm will be described and the importance of the rules for generating RSA keys and the need to supplement them will be discussed. The idea is based on a new way of representing whole numbers.

3.1. Mathematical fundamentals. To present how the mathematical ratio of p and q impacts RSA security, we will show a little bit different representation of N. Let $N = p \cdot q = t^2 + r$, hence $N \equiv r \mod t$ and $t = \lfloor \sqrt{N} \rfloor$. Another representation which we will use is $N + R = (t+1)^2$. From this we can derive

$$t^2 + 2 \cdot t + 1 = N + R, \tag{4}$$

$$2 \cdot t - r = R - 1. \tag{5}$$

The above formulas are easily represented as in Fig. 1.

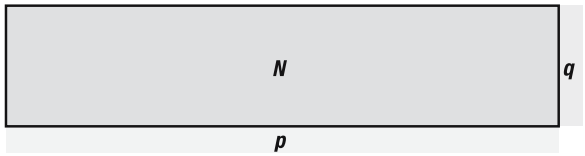

Fig. 1. Simple graphical representation of $N = p \cdot q$.

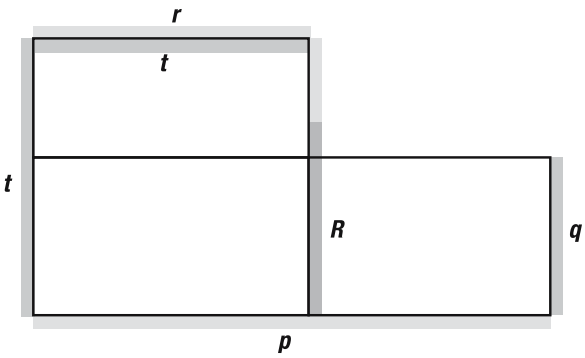

Fig. 2. Combined graphical representation of Eqns. (4), (5) and $N = t^2 + r$.

If we represent multipliers p and q of N as $p = t + x$ and $q = t - y$, where $p > q$ (hence $x > y$), we can derive the following equations:

$$N = (t + x) \cdot (t - y), \tag{6}$$

$$N = t^2 + t \cdot x - t \cdot y - x \cdot y, \tag{7}$$

$$N - t^2 = t \cdot (x - y) - x \cdot y, \tag{8}$$

$$r = t \cdot (x - y) - x \cdot y, \tag{9}$$

$$t = \frac{r + x \cdot y}{(x - y)}. \tag{10}$$

The above formulas result in

$$p + q = 2 \cdot t + x - y, \tag{11}$$

$$p - q = x + y. \tag{12}$$

If we form a square with side length $\frac{1}{2}(p + q)$ and from its area subtract the area of another square with side length t, we get

$$\left(\frac{p+q}{2}\right)^2 - t^2 = v,$$

hence

$$\left(\frac{p+q}{2}\right)^2 = v + t^2.$$

By definition, the modulo number N in the RSA key is a composite number which is a product of two very big prime numbers p and q. If they are prime and greater than 2 then these numbers are odd hence the sum of p and q is even and $\frac{1}{2}(p + q)$ is an integer number.

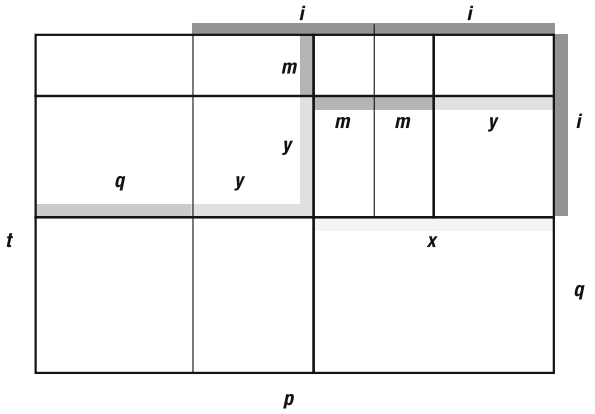

Fig. 3. Graphical representation of the connections between p, q, t, x, y and m, which are introduced and derived in the equations up to (15) and one new variable $i = m+y$.

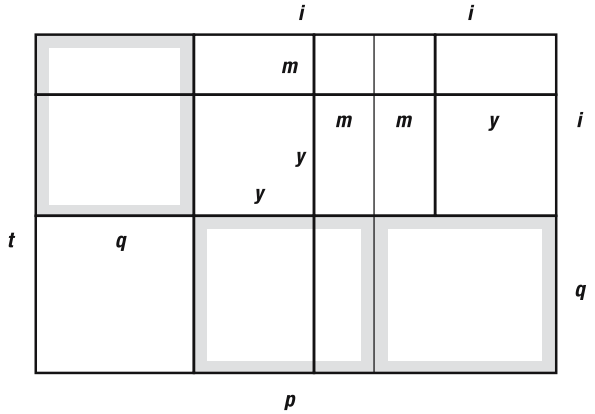

Fig. 4. Graphical representation of N as the difference of two squares: $(t+m)^2$ and i^2.

The square with side $\frac{1}{2}(p+q)$ has a greater area than the square with side t because, by definition, $p > q$, hence $x > y$ and if we refer to (11), we can deduce that $\frac{1}{2}(p+q) > t$. It follows that

$$\frac{p+q}{2} = t + m. \tag{13}$$

From (11) and (13) it is clear that

$$\frac{2 \cdot t + x - y}{2} = t + m. \tag{14}$$

Simplifying (14) yields

$$x - y = 2 \cdot m. \tag{15}$$

Figures 3 and 4 present more clearly all connections between the equations above.

Denoting the sum of y and m as $i = y + m$ and taking a look at Fig. 4 where three zones are marked with equal areas, we easily derive the formula which is a representation of the composite number N as the difference of two squares:

$$(t+m)^2 - i^2 = N. \tag{16}$$

This formula looks the same as Fermat's formula, which represents a composite number by the difference of two squares, but in it the new variable m is in use. Substituting $m = g + 1$ in (15) yields

$$g = \frac{x - y - 2}{2}. \tag{17}$$

Doing the same in (16), we get

$$(t + g + 1)^2 - i^2 = N. \tag{18}$$

If we express i by equation $i^2 = (t + g + 1)^2 - N$, the following simplified formula expressing i by g can be derived:

$$i^2 = (t+1)^2 + 2 \cdot g \cdot (t+1) + g^2 - N. \tag{19}$$

At the beginning N was expressed by equation $N + R = (t+1)^2$. If this is used to substitute N in (19),

$$i^2 = g^2 + 2 \cdot g \cdot (t+1) + R. \tag{20}$$

From (20) it follows that if $g = 0$ then R is an exact square.

Based on all derived formulas up to now, we could state two things: (i) if t and R have firmed values connected to a number N then the value of that N depends on the value of the variable g only; (ii) the minimal value of g we will be attained when the difference between p and q is equal to 2, which means $g = 0$.

By definition, we know the value of N and we compute t and R because from the first defined formulas it is known that $t = \lfloor \sqrt{(N)} \rfloor$ or $N \mod t = r$ and $R = (t^2 + 1) - N$. As a consequence, we can deduce that if we want to factorize N, we have to find out a value of g. From all formulas above it is true that g is in direct proportion of the difference of p and q. If the composite number N has more than two factors, then we will have a value of g for every pair of factors which, after multiplication, have as a result the value equal to the number N. As a consequence, we know that the lower the value of g, the closer the values of the multipliers in the pair of p and q.

The next example demonstrates what is the difference between Fermat's factorization and the way described above through g.

Example 2. (*Factorization of N: Fermat vs. finding g*) If we have to factorize N by the Fermat method, we have to find the smallest number that is squared and added to N to give an exact square. For $N = 3\,611$ that number is 67

$$3\,611 + 67^2 = 8\,100 = 90^2.$$

If we use our known values for t and R, through increasing the value of g by 1 starting from 0 and using that to

calculate i by (20), we will get that for $g = 29$ and $i = 67$,

$$t = \lfloor \sqrt{3\,611} \rfloor = 60,$$

$$R = (60+1)^2 - 3\,611 = 110,$$

$$i^2 = 29^2 + 2 \cdot (60+1) \cdot 29 + 110 = 4\,489$$

$$i = 67.$$

♦

The example demonstrates that by Fermat's method we will have different steps of searching for a proper value of i than to find the value of g. That statement is a consequence of (20).

We think that there is one essential question: Is there a way to decrease the value of g?. The answer is 'yes'. This could be done by multiplying N by $n = a \cdot b$ and if $a/b \approx p/q$ as a consequence of $M = n \cdot N = p \cdot q \cdot a \cdot b$ then we will have $g_M < g_N$. Even if the values of the relationship do not match completely, $\lfloor a/b \rfloor = \lfloor p/q \rfloor$ and the value of g_M could be 0. In other words, we will have a new bigger composite number M with factors $a \cdot p \approx q \cdot b = f$ for which $\lfloor \frac{a \cdot p}{q \cdot b} \rfloor \approx c$, where $c \ll q < p$ and $g_M < g_N$.

If we could find g_M easier than g_N and factorize M, then we can use the greatest common divisor calculation between N and factor f of M, and in this way we can obtain the values of p or q.

As a conclusion, we could say that the mathematical ratio of p to q is very essential to the security of systems which are based on RSA. The algorithm of finding a proper mathematical ratio of p to q which leads to decreasing g to very small values and that allows integer factorization will be called *Fusion of Number Balance*.

The process of key generation for RSA cryptographic systems follows several rules. We will consider official standards of two organizations: NIST (2019) and ETSI (2007). These standards regulate processes of RSA key generation. In the description of the rules, we will denote by $nBits = l/2$ the half of the binary size l of N.

First in SP 800-56Br2 of the NIST recommendations (page 37), we can find the following rules:

1. Primes p and q have to be truly prime numbers.
2. The sizes of p and q have to be in the range
$$\left]2^{nBits-1}, 2^{nBits}\right[.$$
3. Primes p and q have to be generated in different integer intervals to guarantee that the distance between them is really big in the binary representation,
$$|p-q| > 2^{nBits-100}.$$
4. Let N be the product of p and q.

In the second recommendation source, TS 102 176-1 V2.0.0 of ETSI in Annex C entitled "Generation of RSA modulus", we can find the following rules of generating number N:

1. Choose a random prime number p in the range
$$\left]2^{nBits-15}, 2^{nBits+15}\right[.$$
2. The sizes of p and q have to be in the range
$$\left[\frac{2^{nBits-1}}{p}, \frac{2^{nBits}}{p}\right[.$$
3. If the condition $0.1 < |\log_2 p - \log_2 q| < 30$ is not satisfied, go back to the first step.
4. Let N be the product of p and q.

Rules of RSA key primitives generation are also provided by ENISA. The document considers requirements for RSA key domains Smart *et al.* (2014,

Table 1. Example of RSA key domains with a low value of $n = a \cdot b$.

q	D29438CF85A6BE8549CB656EB89D0C0462100D3 772EE9983D0C153A201CF9A260F2254552F66DD F9A7E543F0183551FC28E9358259F40C30B4288 C376AA4FECACD8A8FE86E7BFCBE518373770682 9755FA574B7E9AA39DAAD4346B18A4B573E3F26 9B593FEED02BF22A859F70F1FEDB37C3F6D18ED 498F1C319B408CD6FFC3DD (size 1024 bits)
p	EDAF6345987F5AA44C34E84067BC34760CF8375 5141F254E210956A6A26AAC54B18FDAC5269DA5 BD1307F8B1C3BF7107F6FB692AC62434BEECF1B 2E21999FA27F95507B7D79EEC3471D1E2CB4E00 8614326A076F971B36C1684CA8DD7AEA1EEEA26 C64416BBB50A501A1C390FAA69E2D17239EF314 141A6AFE3E820920DF5475 (size 1024 bits)
N	C3837D9385F40FCECBF8D332BD166778D44A107 EE8738FED8CD291797AB9666AE09590C1AC6807 5ADE94F8D2620E52D8F8B84DDD090B566E0533B 7F0A3AE4B83204E23C0C6E1F048D8F24326FE8D 1E7095984C7F5ABB40F26E9FB81878665686CAF C1246AF5ADBB0ACFD65998A7C4A436C65BDF3EB 6474355CD005D21AA3F34B6AE783F7D2DCBBD5B 453BB0A4EED599BE49F1FC2200687FE552130A3 0BC193840B969365C73D65D9076150B1EF1C8D8 F342CEB0BF5AE5DA538BA06D340C3EA7DBBECFC 73C94A597B73A5A24AFEAEEF4BBACA5A5C66512 92909748AC217C5DD4C6A789B76383CEB568EE4 CE0C9491288BFFFB76424B10726298AC14F66CAC0801 (size 2048 bits)
p-q	1B1B2A7612D89C1F026982D1AF1F2871AAE82A1 DA1308BCA50480304A09B122EA26D866FF736C7 C36B22B4C1AB8A1F0BCE1233A86C30288E38C92 6AAAEF4FB5D2BCA77CF6922EF76204E6F54477D EEBE3812BBF0FC77991694183DC4D634AB0AB00 2AEAD6CCE4DE5DEF96999EB86B0799AE431DA26 CA8B4ECCA3417C49DF9098 (size 1021 bits)

Section 3.5). They are similar to those described and considered by the ETSI standard.

There are two variants of attacks which could affect negatively RSA cryptographic systems security harder than attacks or the mathematical basis of the algorithms. One of these ways concerns processes of public key generation. In the next part of the paper we will clarify how, according to us, all rules connected to key generation process which are enumerated above are not enough. This will be achieved by using mathematical basics described at the beginning of this section and will emphasize the importance of reliable and correct public key generation.

Table 1 contains domains of the RSA key example which presents highlights the importance of g.

RSA key domains presented in Table 1 satisfy all criteria of NIST and ETSI standards but we will use the value of number N to show how important the mathematical ratio of p to q. We used the values of p and q from Table 1 and calculated g_N. That value of g_N is as follows:

g_N	68FF4F9CB155A0990D31735E53E3EC68CC287239 77C4070BD4D45D81D3418BE962D7ACADCE71939C 7C8CB78C7FFD995EAE18E4B7D8CEE8672AE350F9 6EFD568B7BE5088B46C582452166DE1521DD9B16 FEC7258372F419AF7033A3C83C5156715EE97DBD 775FDC00AE8DFAEAD9996E6A7A541C24C861A3A3 2E98B5B84C5843

Applying the algorithm *Fusion of Number Balance* to N, after 41 minutes of its execution on a computer (i5 10th generation based CPU) we obtained a value of $n =$ DD62427(dec:232137767). With that value of n we calculated the new number $M = n \cdot N$. That M has $g_M = 0$, which means that we can calculate i_M through

$$i_M^2 = (t_M + g_M + 1)^2 - M. \tag{21}$$

Equation (21) can be used as a detector of g_M. If g_M has a proper value we could find a value of i_M^2 which has to be an exact square. Hence it is possible to calculate a multiplier f of M:

$$f = q_M = t_M - i_M + 1. \tag{22}$$

To obtain the factor of N we have to calculate the GCD of the values N and f:

$$z = \text{GCD}(N, f). \tag{23}$$

If we do all calculations with parameter values equal to our example numbers above, we will get the following:

Algorithm 2. *Fusion of Number Balance.*

Require: N, U_n, U_g
1: **for** $n = 0;\ i < U_n;\ n++$ **do**
2: $M := N \cdot n$
3: $t_M := \lfloor \sqrt{M} \rfloor$
4: **for** $g_M = 0;\ i < U_g;\ g_M++$ **do**
5: $l := (t_M + g_M + 1)^2 - M$
6: $i_M := \lfloor \sqrt{l} \rfloor$
7: **if** $i_M^2 = l$ **then**
8: $f := t_M - i_M + 1$
9: $z := \text{GCD}(N, f)$
10: **return** z
11: **end if**
12: **end for**
13: **end for**
14: **return** 0

n	DD62427
M	A9138E7BDD03F2271A1A1EA309F834D96BFFA02F D7E8AEF0A92BE66D7513FF2CF64F73E16CBB115A 825874F8A88300539D6680D1F4214CFCF1D9D110 C378A363A6CEAA7962E4868A21F81883BDFAB3FC 97D41629F06EAFDC1EF925A72CA5499E8BE9EB93 CF6EC891C35CA39B7462589143E9335FF111D22D 6F75B5EED7F072952A9CAC1BEB36E7708D7A9681 FAEF9A139AFC4879C9C3D71650C54EE92C9FECD0 834601A52DD70052579258C4D0EA5CA0C9D99A35 B5EB7E29EBA5E003C2ED5F8D2788881DE9DC8A43 0FB13C562AE0DA04817868C23F869667DEFDD0CA DB95A8D518C965423EA69CB41FD1802D3E4B9360 CD89E89952AEC365FC3A7677EAD6D097D2B5C27
t_M	3403022429B7D650B5E10FD69FF33226D9248CF3 B11034C4D1C0B02324C881C7E47EEFAA664C3D55 98DB1D8EBE4AB376F32ED1EA692DCA3ECB8F7FD9 F4042EBAA15278595DC1F9C615EC22B5E708A357 096C2DB28E3BE7826097CE6FBF369D56CE30E20C 44D8144CAB946AF088A957DAF3F13E774723CA2B EBB4A3AA895E7E7DE29B
i_M	585BAD
f	3403022429B7D650B5E10FD69FF33226D9248CF3 B11034C4D1C0B02324C881C7E47EEFAA664C3D55 98DB1D8EBE4AB376F32ED1EA692DCA3ECB8F7FD9 F4042EBAA15278595DC1F9C615EC22B5E708A357 096C2DB28E3BE7826097CE6FBF369D56CE30E20C 44D8144CAB946AF088A957DAF3F13E774723CA2B EBB4A3AA895E7E2586EF
z	D29438CF85A6BE8549CB656EB89D0C0462100D37 72EE9983D0C153A201CF9A260F2254552F66DDF9 A7E543F0183551FC28E9358259F40C30B4288C37 6AA4FECACD8A8FE86E7BFCBE5183737706829755 FA574B7E9AA39DAAD4346B18A4B573E3F269B593 FEED02BF22A859F70F1FEDB37C3F6D18ED498F1C 319B408CD6FFC3DD

The obtained value of z is equal to the factor value q of N.

We write down the algorithm *Fusion of Number Balance* with the pseudocode of Algorithm 2. It uses three input parameters, the number N, an upper limit number U_n for n and an upper limit number U_g for g_M. As a mathematical basis, Eqns. (21)–(23) are used.

Algorithm 2 is not a general factorization algorithm, which means that it cannot be used to perform an effective

attack against all generated RSA keys, but it can probably be used to quickly crack a significant number of them, and this raises a big question: *Should we add a new rule to evaluate the mathematical ratio of p to q in standards related to the key generation process for RSA-based cryptographic systems?*

3.2. Kleptographic algorithm. Everything we have described so far has forced us to think about whether, how and to what extent this can affect the security and resilience of RSA based cryptographic systems. One of the questions we asked ourselves was whether it was possible to create a kleptographic algorithm based on a predetermined fixed mathematical ratio of p to q which would be known only on the developer of a cryptographic system (we will call this developer an attacker). As a result, we created a kleptographic algorithm which will be described in this section. We called it *gBasedKleptoRSA2*.

The functionality of gBasedKleptoRSA2 is divided into two base phases. The first includes the attacker's key parameters which we can call the domains of her key. This is neither a symmetric nor an asymmetric key, because the attacker does not use a cryptographic algorithm to protect the data which leak throughout a hidden channel. The carrier of the information that will allow the attacker to factorize quickly a compromised domain N of an RSA key will be the user's public key certificate. The security of the hidden channel which helps the attacker to be invisible is based on a very big value of the ratio of p to q, which is known in advance to her only. These parameters of the attaker's key could be used no matter what RSA key size has to be created by the cryptographic system key generator (1024 bit size or lager). After these parameters have been created, the attacker puts part of them into hardware firmware or in a software library as a set of constants. The parameters which are used in this phase are the following:

1. Number l which determines the base length in bits of the other parameters it has to be in the range $126, \ldots, 192$;
2. Base parameters a and b with random values and approximately l bit length. The binary size difference between a and b have to be less than 4 bits ($|bitsize(a) - bitsize(b)| < 4$);
3. Base "modifier" integer K which has binary size $l - 63$;
4. Map matrix $R_{x,y}$ which is a two-dimensional array with 9×9 elements and each contains two values a_{sh} and b_{sh} ($R_{x,y} = R[x, y][a_{sh}, b_{sh}]$). Each of them is in the range $-4, \ldots, 4$ ($a_{sh}, b_{sh} \in -4, \ldots, 4$). Each combination of a_{sh} and b_{sh} is unique (see Appendix A);

Algorithm 3. Generate base parameters for gBasedKleptoRSA2.

Require: $l, a, b, R_{x,y}$ $\{bitsize(a) - bitsize(b) < 4\}$
1: $K := RandomBigInteger(l - 63)$
2: $S[0][a] := a$
3: $S[0][b] := b$
4: **for** $i = 1;\ i < 81;\ i++$ **do**
5: $D_{sh} := RandomRange(32 .. 54)$
6: $Z := K$ shl D_{sh}
7: $x := i$ div 9
8: $y := i \mod 9$
9: **if** $R[x, y][a_{sh}] \geq 0$ **then**
10: $S[i][a] := (a$ shl $R[x, y][a_{sh}]) + Z$
11: **else**
12: $S[i][a] := (a$ shr $R[x, y][a_{sh}]) + Z$
13: **end if**
14: **if** $R[x, y][b_{sh}] \geq 0$ **then**
15: $S[i][b] := (b$ shl $R[x, y][b_{sh}]) + Z$
16: **else**
17: $S[i][b] := (b$ shr $R[x, y][b_{sh}]) + Z$
18: **end if**
19: **end for**
20: $D_u := RandomRange(2^{16} .. 2^{32})$
21: **return** S, D_u

5. Parameter D_{sh} which is an integer in the range $32, \ldots, 54$.

As a result of that first phase two parameters are generated:

1. a constant D_u with a size of approximately 32 bits;
2. a table S with 81 rows which contain pairs a and b with a bit length approximately equal to l (plus or minus 4 bits relative to l).

The attacker uses all parameters above to produce table S. She will put S and D_u into the hardware device or software library which are created to be RSA key generators.

It is important to stress that all the following algorithms do not use procedures for fast generation of prime numbers when large random numbers are created. We will use two functions of random number generation. The first one $RandomBigInteger(int\ x)$, generates a big integer number with x bit size. The second, $RandomRange(lo..hi)$, generates randomly an integer in the range $lo, \ldots, hi$ where the maximum value is $2^{32} - 1$.

The algorithm of generating S and D_u is presented by the pseudocode of Algorithm 3.

Algorithm 3 of RSA key generation uses the following input parameters: k (key size, i.e., 1024, 2048 etc.), S, D_u, e (public exponent), d_a and d_b, where the values of last two parameters are results of functions

Algorithm 4. Generate RSA kleptographic key domains.

Require: k, S, D_u, e, d_a, d_b
1: $l := bitsize(S[0][b])$
2: $u := RandomBigInteger((k/2) - l)$
3: $m := RandomRange(0..80)$
4: **for** $i = 0;\ i < 81;\ i++$ **do**
5: $f := (m + i) \mod 81$
6: $d := (u - D_u) \cdot (u + D_u) \cdot S[f][a] \cdot S[f][b]$
7: **if** $bitsize(d) = k$ **then**
8: $D_a := RandomRange(1..171)$ shl d_a
9: $p := NextPrime((u - D_u) \cdot (S[f][a] + D_a))$
10: $D_b := RandomRange(1..171)$ shl d_b
11: $q := NextPrime((u + D_u) \cdot (S[f][b] + D_b))$
12: **if** $p < q$ **then**
13: $z := p$
14: $p := q$
15: $q := z$
16: **end if**
17: $break$
18: **end if**
19: **end for**
20: $N := p \cdot q$
21: $d := e^{-1} \mod \varphi(N)$
22: **return** N, p, q, d

Algorithm 5. Factorization of compromised RSA key.

Require: S, N, e, d_a, d_b
1: **for** $i = 0;\ i < 81;\ i++$ **do**
2: **for** $wL = 1;\ wL < 172;\ wL++$ **do**
3: $aV := S[i][a] + (wL$ shl $d_a)$
4: **for** $wH = 1;\ wH < 172;\ wH++$ **do**
5: $bV := S[i][b] + (wH$ shl $d_b)$
6: $n := aV \cdot bV$
7: $M := n \cdot N$
8: $t := \lfloor \sqrt{M} \rfloor$
9: $z := (t + 1)^2 - M$
10: $i := \lfloor \sqrt{z} \rfloor$
11: **if** $i^2 = z$ **then**
12: $f := t - i + 1$
13: $q := GCD(f, N)$
14: $p := N$ div q
15: $d := e^{-1} \mod \varphi(N)$
16: **return** p, q, d
17: **end if**
18: **end for**
19: **end for**
20: **end for**
21: **return** $p = 0, q = 0, d = 0$

$G_a(id, time)$ and $G_b(id, time)$, i.e., $d_a = G_a(id, time)$ and $d_b = G_b(id, time)$. The input parameters of these functions are the identifier of the certificate and the start date of validity. They have different results when their input parameters are the same. Results of functions G_a and G_b are in the range $1, \ldots, 103$. In the next algorithm we will use another function $NextPrime(BigInteger\ x)$. That function returns the value of the next prime number bigger than x.

The algorithm of RSA key generation by gBasedKleptoRSA2 is represented as Algorithm 4.

The formulas for p and q which are used in Algorithm 4 are as follows:

$$p = (u - D_u) \cdot (S[f][a] + D_a) + r_p, \tag{24}$$

$$q = (u + D_u) \cdot (S[f][a] + D_b) + r_q. \tag{25}$$

The algorithm ends when suitable values for $S[f][a]$ and $S[f][b]$ are found, i.e., the product of p and q has a bit length equal to the needed key size. The product of r_p and r_q is the count of iterations which are needed to generate the final key domains. This is the biggest part of key generation time because for each iteration r_p and r_q the primality test is performed.

Several examples of RSA generated key domains with different key sizes from 1024 to 4096 bits are presented in Appendix B. In the generation process for all keys contained in Appendix B, the same table S was used as a key generation parameter.

The next algorithm, represented as Algorithm 5, contains the steps of execution which the attacker makes in order to decompose N (the modulo number of the compromised RSA public key). To perform a successful attack, an attacker needs three things: the contents of the table S, D_u and the certificate, which is the user's public key. When the attacker obtains a user certificate, she first calculates $d_a = G_a(id, time)$ and $d_b = G_b(id, time)$ and then performs the steps of Algorithm 5. It is important to note that this is applicable when the bit size of the product of a and b is less than the half of the key length.

If the attacker is successful, the result of this algorithm are values of (p, q, d) greater than zero and this is true if the input parameter N is a part of the compromised RSA key which is generated with predefined S and D_u; otherwise, the result is $(p = 0, q = 0, d = 0)$.

In cases when half of the key size (k) is smaller than the product of a_f and b_f ($a_f = S[f][a]$ and $b_f = S[f][b]$), we will use another algorithm to obtain factors p and q. When this is true, $u < a_f$ and $u < b_f$. We have

$$\begin{aligned} N &\approx ((u - D_u) \cdot a_f + r_p) \cdot ((u + D_u) \cdot b_f + r_q) \\ &= (u^2 - D_u^2) \cdot a_f \cdot b_f + (u - D_u) \cdot a_f \cdot r_q \\ &\quad + (u + D_u) \cdot b_f \cdot r_p + r_p \cdot r_q. \end{aligned}$$

Hence

$$u \approx \lfloor \sqrt{N \text{div}\ a_i \cdot b_i} \rfloor. \tag{26}$$

This means that the result of the computation $u = \lfloor \sqrt{N \text{div}\ a_f \cdot b_f} \rfloor - D_u$ is a float value on the least

significant bit if $u << a_f \cdot b_f$ (the least significant bit of u can be 0 or 1). As a consequence, if half of the key size (k) is smaller than the product of a_f and b_f, then the attacker could obtain $p = (u - D_u) \cdot (S[f][a] + D_a \operatorname{shl} d_a)$, where calculation is performed for every $D_a \in \{1, \ldots, 171\}$ combined with all pairs a_f and b_f. If the multiplier p is found, then $N \mod p = 0$.

3.3. Finding if gBasedKleptoRSA2 could be hidden.

In order to exist and be used in practice, a kleptographic algorithm must have characteristics that allow it to go unnoticed. That is why we have to analyze whether it is possible to detect gBasedKleptoRSA2 if it is embedded into cryptographic systems.

If a kleptographic algorithm is not hidden enough, the first feature which could be obvious is the domain generation time of a key. If there is an abnormal time difference of key generation between non-kleptographic and kleptographic algorithms that could be a reason to suspect something is wrong in the key generating tool, this will raise the question of whether kleptography is used in it.

To estimate the generation time through gBasedKleptoRSA2, we performed more than 2000 key generations with lengths from 1024 to 4096. In that process we used two S tables S_1 and S_2. The root parameters a and b which were used to create table S_1 had a length of 152 bits each and for S_2 this was 232 bits. We made a comparison of the average times of key generation through gBasedKleptoRSA2 and the openSSL library (version 1.1.1h22 Sep 2020). For key generations, the same computer was used. The results of performed tests are as follows:

- 1024 bit key:
  - using: S_1 362 ms, S2 293 ms
  - openSSL 285 ms
- 2048 bit key:
  - using: S_1 2.04 sec, S_2 1.7 sec
  - openSSL 350 ms
- 4096 bit key:
  - using: S_2 33 sec
  - openSSL 911 ms

We have to emphasize that in the openSSL library an RSA key is generated trough fast prime number generation algorithms. That is why an obvious time difference exists when 2048 and 4096 bit keys are generated. If other cryptographic tools are used instead of openSSL, the time required to generate 1024 and 2048 bit keys is comparable to that of gBasedKleptoRSA2. That sets a task for future work to find out why openSSL owns a feature like that. One of the possible reasons could be that the intervals where p and q are generated are not random enough. This could lead to of set of limited mathematical ratios of p to q. As a consequence, if that is found, then *Fusion of Number Balance* can be used for factorization.

The next part of our analysis concerns the possibility of finding a pair of a_i and b_i. If this is proven, it is enough to state that the kleptographic algorithm gBasedKleptoRSA2 is embedded into the system which we analyze. To do that, we need to have one pair p and q from a key produced by the system which we analyze and to generate $(171 \cdot 103)^2 \cdot 81 = 25\,127\,639\,289$ other keys by the same system. We have to do that to be sure that we have a 100% possibility to find out the same a_i, b_i, d_a, d_b, D_a and D_b in other keys which are being used in obtaining our base pair p and q. This is the first difficulty because if a single 1024 bit key generation time is the lowest one 290 ms the that will take 84 340 days (231 years). Of course, if the key generation process is parallelized, this would take less time but money investments have to increase. For example, the Microsoft cloud HSM could be rented for about \$4.85 per hour or the IMB cloud HSM will cost \$1250 per month. This is not a big investment for an organization but it needs to invest one million dollars to perform that amount of key generation per week.

The next difficulty is a bigger obstacle. We have to perform 25 127 639 289 checks of $\mathrm{GCD}(w_i, w_j) = A$ to find out an existence of at least one value for A which has at least 126 bit length. In that calculation $i \neq j$ and $w_x = p_x - r_x$, where p_x is an RSA key domain parameter (a factor of N_x) and the maximum value r_x is the difference between p_x and previous prime number lower than p_x. All calculations described above took at least 10 min for 1024 bit key when i5 10th generation with 60% load and 2.7GHz maximum clock is used. The time it will take for this analysis to be successful will be 478 075 years.

Here is the place to recall that the values of D_a and D_b are in the range $1, \ldots, 171$ and theses of d_a and d_b are in the range $1, \ldots, 103$. These ranges could be bigger. The time required for an attacker to factorize N for a 1024-bit key with these parameters is less than 10 seconds, for a 4096 bit-key is less than 35 seconds. These are very small values of time for attack execution.

All the parameters related to gBaseKleptoRSA2 make it dangerous if used in an RSA cryptographic system, and we can say that it is very likely to remain hidden if no serious analysis is done on the keys generated by such a system.

Speaking of kleptography and after the introduction of gBasedKleptoRSA2, we want to draw attention to a study entitled "The million key question—Exploring the origins of RSA public keys", which was done by Czech team developers (Svenda *et al.*, 2016). In their study,

they evaluated the process of RSA key generation. They have investigated over 39 hardware devices and software libraries. One of the results shows a very interesting fact. Among all analyzed tools only the Microsoft key generation tool has a good probability distribution of the most significant byte of the prime p and its corresponding prime q. A bad distribution could be a sign of a poor ratio of p to q. This may be due to a hidden unintentional form of kleptography, which is as consequence of the prime number generating principle. In other words, this could be due to the use of fast generating prime numbers algorithms for example. But all the results mentioned could be a sign of very dangerous vulnerability which affects seriously resilience of RSA based cryptographic systems.

4. Conclusion

Based on our experience in the field of information security and our knowledge in number theory, we aim to increase the cryptographic security of systems that people use widely. The presented approach to security assessment on one of the most commonly used encryption algorithms aims to focus efforts on creating algorithms that will increase the security of RSA based cryptographic systems. We have to stress that the suggested algorithm *Fusion of Number Balance* is not a factorization algorithm in general.

This article shows the need for new requirements and rules to be part of worldwide standards for RSA key generation. These new rules should serve to prevent the creation of RSA keys that are easy to attack as a result of a small mathematical ratio of p to q. Without such a new part of the rules, it is very possible that a form of kleptographic attack is part of existing systems based on RSA. This may be due to the desire to use algorithms to quickly generate large prime numbers, which is a consequence of wanting the key generation process to be as fast as possible. Practically speaking, if a form of kleptographic attack based on the idea described by gBasedKleptoRSA2 is implemented in a cryptographic system, the information security of the organization in which this system is operated seriously threatens its security and work resilience.

Our future efforts will be to develop an algorithm which will increase RSA security by suggesting a better calculated ratio of p to q. The other future work will include an improvement of the algorithm which deals with estimation of the probable ratio of p to q and will aim at speeding up *Fusion of Number Balance*.

Acknowledgment

This work has been funded by the European Union's Horizon 2020 Research and Innovation Programme under the grant agreement no. 830943, the *ECHO (European Network of Cybersecurity Centres and Competence Hub for Innovation and Operations)* project.

References

Adj, G., Canales-Martínez, I., Cruz-Cortés, N., Menezes, A., Oliveira, T., Rivera-Zamarripa, L. and Rodríguez-Henríquez, F. (2018). Computing discrete logarithms in cryptographically-interesting characteristic-three finite fields, *Advances in Mathematics of Communications* **12**(4): 741–759.

Ahlswede, R. (2016). Elliptic curve cryptosystems, *in* A. Ahlswede *et al.* (Eds), *Hiding Data Selected Topics: Foundations in Signal Processing, Communications and Networking*, Vol. 12, Cham, pp. 225–336, DOI: 10.1007/978-3-319-31515-7_4.

Alwen, J., Dodis, Y. and Wichs, D. (2009). Leakage-resilient public-key cryptography in the bounded-retrieval model, *in* S. Halevi (Ed.), *Advances in Cryptology, CRYPTO 2009*, Springer, Berlin, pp. 36–54.

Anderson, R.J. (1993). Practical RSA trapdoor, *Electronics Letters* **29**(11): 995.

Bressoud, D.M. and Wagon, S. (2000). *Course in Computational Number Theory*, Key College Publishing, Emeryville.

Devidas, S., Rao Y.V., S. and Rekha, N.R. (2021). A decentralized group signature scheme for privacy protection in a blockchain, *International Journal of Applied Mathematics and Computer Science* **31**(2): 353–364, DOI: 10.34768/amcs-2021-0024.

Diffie, W. and Hellman, M. (1976). New directions in cryptography, *IEEE Transactions on Information Theory* **22**(6): 644–654, DOI: 10.1109/TIT.1976.1055638.

Dodis, Y., Franklin, M., Katz, J., Miyaji, A. and Yung, M. (2004). A generic construction for intrusion-resilient public-key encryption, *in* T. Okamoto (Ed.), *Topics in Cryptology, CT-RSA 2004*, Springer, Berlin/Heidelberg, pp. 81–98.

Elgamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* **31**(4): 469–472, DOI: 10.1109/TIT.1985.10570748.

Erra, R. and Grenier, C. (2009). The Fermat factorization method revisited, *Cryptology ePrint Archive*, Report 2009/318, `https://eprint.iacr.org/2009/318.pdf`.

ETSI (2007). Electronic signatures and infrastructures (ESI): Algorithms and parameters for secure electronic signatures. Part 1: Hash functions and asymmetric algorithms, *TS 102 176-1—V2.1.1*, European Telecommunications Standards Institute, Valbonne, `https://www.etsi.org/deliver/etsi_ts/102100_102199/10217601/02.01.01_60/ts_10217601v020101p.pdf`.

Gordon, D. (2011). Discrete logarithm problem, *in* H.C.A. van Tilborg and S. Jajodia (Eds), *Encyclopedia of Cryptography and Security*, Springer, Boston, pp. 352–353, DOI: 10.1007/978-1-4419-5906-5_445.

Kaliski, B. (2011). Euler's totient function, *in* H.C.A. van Tilborg and S. Jajodia (Eds), *Encyclopedia of Cryptography and Security*, Springer, Boston, pp. 430–430.

Kaliski, B.S.J. (1993). Anderson's RSA trapdoor can be broken, *Electronics Letters* **29**(15): 1387–1388.

Markelova, A.V. (2021). Embedding asymmetric backdoors into the RSA key generator, *Journal of Computer Virology and Hacking Techniques* **17**(1): 37–46, DOI: 10.1007/s11416-020-00363-x.

Menezes, A.J., Van Oorschot, P.C. and Vanstone, S.A. (1996). *Handbook of Applied Cryptography*, CRC Press, Boca Raton.

NIST (2019). Recommendation for pair-wise key establishment using integer factorization cryptography, *NIST SP 800-56Br2*, National Institute of Standards and Technology, Gaithersburg, DOI: 10.6028/NIST.SP.800-56Br2.

Pomerance, C. (1982). Analysis and comparison of some integer factoring algorithms, *in* H.W. Lenstra and R. Tijdeman (Eds), *Computational Methods in Number Theory*, Math Centrum, Amsterdam, pp. 89–139.

Rivest, R. L., Shamir, A. and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* **21**(2): 120–126, DOI: 10.1145/359340.359342.

Sako, K. (2011). Digital signature schemes, *in* H.C.A. van Tilborg and S. Jajodia (Eds), *Encyclopedia of Cryptography and Security*, Springer, Boston, pp. 343–344.

Smart, N., Rijmen, V., Gierlichs, B., Paterson, K., Stam, M., Warinschi, B. and Watson, G. (2014). Algorithms, key size and parameters report 2014, European Union Agency for Network and Information Security (ENISA), Brussels, `https://www.enisa.europa.eu/publications/algorithms-key-size-and-parameters-report-2014`.

Svenda, P., Nemec, M., Sekan, P., Kvasnovsky, R., Formanek, D., Komarek, D. and Matyas, V. (2016). The million-key question—Investigating the origins of RSA public keys, *25th USENIX Security Symposium (USENIX Security 16), Austin, USA*, pp. 893–910.

Yan, S.Y. (2019). *Cybercryptography: Applicable Cryptography for Cyberspace Security*, Springer, Cham, chapter "Elliptic curve cryptography", pp. 343–398.

Yasuda, M., Shimoyama, T., Kogure, J. and Izu, T. (2012). On the strength comparison of the ECDLP and the IFP, *in* I. Visconti and R. De Prisco (Eds), *Security and Cryptography for Networks*, Springer, Berlin, pp. 302–325.

Young, A. and Yung, M. (1996). The dark side of "black-box" cryptography or: Should we trust capstone?, *in* N. Koblitz (Ed.), *Advances in Cryptology, CRYPTO'96*, Springer, Berlin/Heidelberg, pp. 89–103.

Young, A. and Yung, M. (1997). Kleptography: Using cryptography against cryptography, *in* W. Fumy (Ed.), *Advances in Cryptology, EUROCRYPT'97*, Springer, Berlin/Heidelberg, pp. 62–74.

Andrey Ivanov is an engineer. He graduated in automated management systems from P. Volov Higher Military Artillery School in Shumen, Bulgaria, in 2000. His main research interests focus on cryptography and cybersecurity.

Nikolai Stoianov is an assistant professor and a deputy director of the Professor Tsvetan Lazarov Bulgarian Defence Institute. In 2003 he received a PhD degree in the Institute for Perspective Research for Defence (IPRD), G.S. Rakovski National Defence College. He is an expert in communication and information systems and information protection. In 2020 he graduated in strategic management of defence and the armed forces from G.S. Rakovski National Defence College.

Appendix A

Matrix R_x, y : 9×9 values:

```
       0       1           4       5           8
0: [ 0, 0] [ 0, 1] ... [ 0, 4] [ 0,-1] ... [ 0,-4]
1: [ 1, 0] [ 1, 1] ... [ 1, 4] [ 1,-1] ... [ 1,-4]
      :                                     :
4: [ 4, 0] [ 4, 1] ... [ 4, 4] [ 4,-1] ... [ 4,-4]
5: [-1, 0] [-1, 1] ... [-1, 4] [-1,-1] ... [-1,-4]
      :                                     :
8: [-4, 0] [-4, 0] ... [-4, 4] [-4,-1] ... [-4,-4]
```

Appendix B

ID: 1	1024 bit size generation time: 156 ms
q	6E2F8AECBE4C50A206B9F6427D9247EE240EE43 7615736B822C7E8C36FFD77749B424903EF72E1 048C0606EFB6C4AF8C8108534D68A96DE43961F 35198E31EFD (size 511 bits)
p	167902C95E4646A4A703E15F01C8E2F6BA918B8 94D025C9007E96A66B5B1828135FF112740C055 F03113EDAAAA115348DD28BDA58ECD0A52FA163 275B64992FCB (size 513 bits)
N	9AC2B9C0D3E22C8D4A635AFE84A2B6511A30578 47664FD7638CA1AFABE8D906F3A5D010F0C3E8B 969101FA5EA119AEA4F3CC043CFBA05CDAFB07B 7E3BBBEF9D43F09F1E5D9CEFBC2F999BF31FEE3 33386B194E439C9BE1BADF1BF4854C397EC5CB2 FC430076EF45D5DF88DD627FAE7D2D43D962687 2B3D29D320938D47FF059F (size 1024 bits)
p-q	F960A1A9261819A869841FAD9EFBE77D8509D45 D6ECE92485BCEBDA7EB1AB09EC4AEC9701C927D FE8538D3BAEA5085015183870B8427374B68013 409CBB610CE (size 512 bits)

ID: 2	1024 bit size generation time: 250 ms
q	417386A7E084955B94BF33983102AA263A8C01E 4FE36899182C148B9805375F3B3ABC28296B0B6 881FAB8EF8928B9E001032EBA199FFAC1BF3A4D FCB43353B71 (size 511 bits)
p	281D3C6E65D0E7DEBB9CB8813F5FAF9887F8122 ABA1FF099436F5C2FF89E9F6B1A03A47EA6AE26 0F3211E37ACF61593BBC8D264057542F5FFF58D 0A71410A129D (size 514 bits)
N	A418693C95458629B39CEB0EF7E54E1036FEBBF 1EB0CF8B147E33F7E176B4AFD39B767EBA85C66 3AA458059BAE847495391FBD90F87A525F66DB4 1614081569FFAB9FD4B97FC82C06A5D7AC795BA E8B5124E23E49607406A2BE9CD593B5919ED7C5 732FBCE93E9393C45ADBE5A09718D7BD74159E0 B0EFA5FEE9556EF93D664D (size 1024 bits)
p-q	24060403E7C89E890250C547BC4F84F6244F520 C6A3C88002B4347A46099680BDEC8E8567D431A A6B0172A8B46389F5BBB89F7863DB4349E401E8 2AA5FDD4D72C (size 514 bits)

ID: 4	2048 bit size generation time: 1859 ms
q	97866EF5B259F782BAFF2D9896B7054A24256CB4585F CB0686F33E8140F9A10120362A1AD5770DA0D6C2F60E EA141EB867B2C0A510F9ACCFD47E62C44C63CA5FF490 2AC264D14FDC6D6D9BD5E61FE3FF6F75A8E188158B02 01E69C0B4D8BAE8E89D56592306DF085C9F1232AD907 38EEFE535E43F4E26EC41273D721E309695B (size 1024 bits)
p	17378890FB02C3E47840057B5967C07AE35BE472FB77 E223C0E51238499BC0DAB853E91CA59DF1059CD09010 864FA9595218DEAC2742D5398EB787CBF5B10FC8C5D3 84F0CF4F172C1B9F5DC1E53E6274C4DB9EC3C55918C5 EABF33EF295459F1F2DEEAF04EA7BF8A44523B3735F3 DDFF6F428F6F3E8562EE099FD279A14D03EFB (size 1025 bits)
N	DBDF2AF24F2C94F6F3243B848D039E6EB0393287FF63 20EE7F2518D078980B021BC3E76360F7576C0E5515DD 4D21880271412FF623A285047A28D69EA9783293D202 4D3FF9B524A6AE2AE26288B7DFA1D86DEFD452374687 C4EB2C85DBD698D9CD36CAF360500257F668D1747572 A5F1C4FEE9B226FE9850C3C47679C8F494DA8F66096F 4B00A1C3E8B43ED9C971AD2241E81F1C571567AD4BA0 ED054049E8E06E65BEA7B16E8DA91828AF6CC5EA5E48 081F6CE7878FA2A7CC06F8B795E0A31543D9932FA6E0 CDDD1D2AFE9C00C7901555FDFB3E334530C525EF692A 93067B70F23876C469CD6CAA0C8A6497C873509C995A 17DA13F63F811B9894D497AE5639 (size 2048 bits)
p-q	DBF21A19FDD246C4C9012A1CFFC502641198DA7B5F1E 5735875DE50358C26CAA650867AF846802B8F6460AF9 7AE676DCB9DB2A1D6333A6C916FA19FB0EAD322C68A8 244A902022E54C8840486DC6434CDE44435ACD7C015C AA0CA2E747B9F090A4194972BA0E081E7B3290488636 A707F5D598AFF373C01C8789507831C6D5A0 (size 1024 bits)

ID: 3	2048 bit size generation time: 1523 ms
q	282F500ADB641C6E1E3A1EAD896F1723BD3D1411B9FB F5CD86199D82EDC9D3BF46D0F7296AE83724952B50DC FC89E77A690E7E501040EB8D6F9B7F3C968B39B91D15 E4AF9AE7EF8C02E439335ADBEA8F95D289F9D1CA58EF 34DB9BA65C7DFB7A1D7AF347185E3BE1DA742EF52D66 D43AAA7D9269E7DC98C76ECEF20A2BEF5697 (size 1022 bits)
p	4191052CAC4DC232D487F3291C109EF834FD9D911D70 C3BE8A4973DD120B5B435746D16D8480283113131F29 7792C697B60EDC7BE0007DEB2BDE65AE1ED02BB80B63 DB4DAEB9EB159B3724859E9AEB79790A3ADA69D17F70 B162B78E8AC4584894C6A49C8322C09163D07C97BB08 DA162FE39E1452E201CB73C2EB487860DE233 (size 1027 bits)
N	A4AC6EF0798A590EDBC8C3895AC5CCE715251F4BB8DB E431EED37F8DC2635448FF4E733E4154978415B3F4CE 0B5D9F3DE9BA36E798020B375CF8908D48199BEDBB7E D49C8C3B6F42335FD455478F4C13D0F0A9DCE86C79B8 63930449649362FC7F78F2793141B07A591563C97AC1 040B570A6C4505BB2CE7D474E5109A3A294D88721D88 64B9EC4FF5D2514EDE84A1F98592ED45708D542AB7B9 D31C8B05CD33290F67E5B7718D34C93D0EEA019E143A C324CDB7DFA6A19EEDD9AAFF36343366929BA28D3115 9A448F785493C73DFE289550E13DCC36C3C88DF54654 A9294B204A2E05FBF39F285664E373D2067519F17D61 CC3571D3A440C9A58AEB7ACA8E15 (size 2048 bits)
p-q	3F0E102BFE97806BF2A4513E4379AD85F929CC5001D1 0461B1E7DA04E32EBE0762D9C1FAEDD1A4BEC9C06A1B A7CA28200F7DF496DEFC6F3254E4ADBA5567781C7992 7D02B50B6C1CDB08E0F268ED2CD07FAD123ACCB4D9E1 BE14FDD424FC7890F2EEF568119CDCD3462939A86832 6CD2853BC4EDB464383EFCD5FC27D5A1E8B9C (size 1026 bits)

ID: 5	2048 bit size generation time: 1407 ms
q	52574FDA16AF6CED61C27A2271B2D8F1313AD570E51E 6E0F6D329CB8058B6E3D20AECD2DB199C1044C480F12 417220A454C9B3683CB76F76A678E2A67D615E73F50E 8FA5B62627C940B13B73B931CFE7D95007529A9A011C FD0E46BD22A387291AE4599F09E43B0DBF1C7D62612F 574AA288B6627959E9A69564C91012C9FBE7 (size 1023 bits)
p	21965FEDB3780DABC59E49DC2E6E509F82580FAE84B7 3E162D2C675FC3EB0C6B1B84F96602FD219A59A3D0DD CA3BCAF8A71CC24FCBBF491763476BCD53563DD59325 A9FF80D9A35DA86C1D8CCCE2AAD4E08D4833ED44C5EB 86278A45D7E03CF43E8CC9F68E6308753DDA5D64FFBB 14121403CD1EFDAA92EE1BD0F8F55D6D663AD (size 1026 bits)
N	ACD9F4EC1E66D4F218D180D9B5720B4DAF36DD88FCA4 D919CB5E0992850EE4020214BE00F168FF39C1B74C16 D92B2E940C220E75DF1C55A2EC80251AFB4C37838726 EFEC1B350BDA671A6626C74E32E3403D5D1CE0ADD802 BA118F1707BAD05DB9D997457D14F3F3B38A89BB8F1A CDD3BF7549FBBF8B8180895044453D127087FBB99EB2 4A2FC588BE277D7D538DC62FF59A9C11F97729BB054E 62472F05058B2AD0B1483576065FAABE7E7F077D4AEF 4DA7801BCF61C824B29FD6B78174D2760DD1352E441B 6A57D5E3E5B072CD102133C57BB2FE8C2AFF9DA31300 7B4EAD59144E614585387049F9CD9AA93D41A7BBBECA 8113B6215157D051513F7C03901B (size 2048 bits)
p-q	1C70EAF0120D16DCEF82223A075323106F4462577665 573536593D9443925587497A0C9327E3858A14DF4FEC A624A8EE61D0271947F3D21FF8DFDDA2EB8027EE53D4 C105257740E1146109D5914F8DD662F847BEC39B25D9 B656A5DA05B60481ACDE845C9DC4C4C461E8958ED9A8 1E9D69DB41B8D614F453B27AAC645C40C67C6 (size 1025 bits)

ID: 6	4096 bit size generation time: 3828 ms
q	2E22F5E3104A7D9C56FA5DCAED770D4C9EB833F7C9DC 5E85F0F5135967B5BB6034BCDC25E7417BE246099192 DFE612E1D7AACB3B7734D8E3778F0DB6CCCFEA5536F0 75F3AE87679C995551777FEC9DD9C3A219C28F51C731 C792149AB7E04CADF597483B1E246FA272756D100341 35CA6F61974D2C45CB3F05CED8ED3028D3D538E6D66F 897DD35BF64AA3B80077A66C590564BBDFB01332AAEB 94D7F357C2FB82D632A0654D4F96DE4607A31BD16DEC 208663EF9248FD815CF45D1C4A83CDDD87A2BFA245DF ABACA4545DADC3497ED58A6413159734979AC3383BF8 CD87E35391AD9BEF45C0377F08297451334D82D4F60E 3D16F42ADA18A16935B6A7298A7F (size 2046 bits)
p	4B470A28F736AFB3D698F83565570AF0C7B661EE42AC BF92FE4D52FBC79562923AA4667AA89DC38590DFFB10 3E5D92022523DF0D2EDDFF8D35B1396FA8689D17B82C C79937028E94A16C105023CCC92AA5E6699456C0BD7E AC94C9F28026ADECCE73EBFC48DC433312DF7FDBFF12 A048CE57FEB6BD642D8B0E98AB7011E04FF0321A76C9 4747144EC8FF3E815A46D4BD8A8BC4C1BB5970888A37 1764BD32FC37022FBA4F0D77337482670FBBBA92A508 EDA285CF7930B7BC5FA68BFB7EF6309137AE9C066CCC 72962C31D8269EC9B2ABDF7A0BAB65AB6979B590ED4B 2B3C7A453519A783205F73412BDF94707BCCEC8C920D 2029DF7A6CC055CB8D77FD1407DE5 (size 2051 bits)
N	D910B90775E37DB5A357496C8A03ADD2594115C15F73 7ADC38C2C4D9B19DB19500684948E9856B6B1949A013 6AA817C5D8B75C89269B7C10B5DF48ADC6C106AED5F8 5A050930C4518CF1C85DD9FBD5B84DFAE1F44D6D59D0 B49B8CC1AB7E942D45E6F3584396A20A3E41F27DD273 89B05765567C266241A9D4FA95058067ABF9976D3F7F 8F1C73475EFF342B53F42ECA82A13E78EAD6F936FE4A 44B701507E2852E23464734F3D49A29EF64AE5A6946D 71C6DCD42DE09D0ABD60A094F52DAE83BE3F12A11884 4C2FCD09C2BB85772A1728DE8FE51E8265F3E0678AF4 38B72A768AD81DD6894A903C54CA8462E8028DC653D8 DDBC161D02EF7D5064D84C5750CF9462D1462F961FD8 30A7E538EFCE0ED9D42FC1065F2B312E0B56227830EF 34E64EA0A81258B51EF26A365A3B3B4FBB74CA8D5608 670B5F51834D051BF1103ABA09432E8B59435FBBFECB 62E1FB4381291DDF2382C4FD49A6722786BF1384BA7F 547B05D5B0CAA3E6E6ED987A2B77233B30068AB3A847 8D37BDBCBF8AC1B1C7D6CA5A1FE1009F1C52FFE07CC3 9D9237672BCF7B2AFEBACE400DBE6B815D49300E9C57 029F11E0EF38C91A8B3738F0556BAF3D0CCDBADA4D35 4D4F4F770FD1559DC1DA6AD128CED4A744B85036C762 BDAB5A11D8C3CF59523FE05810AEEFCA191AE6C616F9 512A9456DB0F1898B6FA58492D1997735CF2661D4EAE 4E781F88E69B (size 4096 bits)
p-q	4864DACAC63207DA11295258B67F9A1BFDCADEAEC60E F9AA9F3E01C6311A06DC375898B84A29ABC76C7F61F7 105F30D407A93259776AB1FEFE3848943B9B9E7264BD C039FC1A181AD7D6BB38ABCDFF4D09AC47F82DCBA10B 901BA8A8D4A8A921EF1A777896F9FC38EBB8290AFEDE 8CEC2761E541EA9FD0D71E3BBDE13EDDC2B2DE8C0962 4EAF3719099A9445DA3F5A56C4FB6E75FD5E6F555F88 5E173DFD80074A02572507225E7B1482AF4188D58E2A 2B9A1F90800C27E449D74629BA4DF3B35F34700C486E 77DB61EC924BC2951ABE86D3CA7A0C382000095D698B 9E63FC0FFBFECDC42C036FC93B5CFD2B6898145F42AC 3C587037BF1ECBB4FA1C92A16F366 (size 2051 bits)

ID: 7	4096 bit size generation time: 3547 ms
q	4B7D924158A52D138BB8300A49E8785756D34FBC7E68 072696DD3AD1C3B411AD705B7F3D32D032947AB93922 D9939E6C4D8D7E7BC594A23EE2610F7A809F92C43BBF 7A09DE172051AD234071F68813F4F196BDDCE784D045 B27D39D45196EDD23BB437E854D1954DDC3583BB8680 2E6E4AD9ED78C82EF630088CD3BEF10BD4E7196D00B1 16443A80C26D5C578C6CDC135664908B0B4FC0F200DC ADCE277656C7FFA6CCD89471A0B415AFFD6C4249736E CF0173D39E28F8F276193D57D1F11F19A49877F2746F 64F540DC95BEAB0CCF49809715B9F4BE3E665A5D26BD 204DE8A4446473951F6EF646CA3F1FC8C12A112BC43E 2EE0ED98ADEFEB91C797B79098F5 (size 2047 bits)
p	2E4461D75820652D109ECD574C16F41E350BE2179C5C 45043D396182F65DDAB585A13525D671EDE7C65E5033 5A347376E6DB370BE5BF8D6A6BA1705BAC18CC3E091A 8C013900943CDA7E867325959ACBC38D04AE10666920 BD717FC0947CBBB2BCD5955FA55131AEB1FA3DE52759 96DA6D7C9DD8413ADCBBC467468E58D1471C50662D20 9D63FEBD192731B285AEEFBD25D5C5AFF7C8876CC539 1AC4D9D6452BDF8A7946CAC5BBF9A508FA6A8C057125 54E3E1FC4BF6F6BD33216DBB164FBAF7424F74576589 53C092246472963B7EECAA238BFD247ED23181EF4150 E5F6C9626191742166EEE05DB7EF8832C074437F3A36 673A0AE75A501128AD0845568405B (size 2050 bits)
N	DA4BA7CAC268940186A7FB57E4C2D449560246135BFA 1F12A4A859B2EFD4DB65A4284D985756351997 88ECCC 2D0A13F4C84D768FCE061992E09C20A4F98B1AEDC3CA 2D2371E65124DF53D65D6D543E5109090D1386C80678 21B1B13F8D754BB8DEFE4C282EBE3144CE9703525518 AC0184908327E6E1B14050839E03D7A504FBC18DABD6 9A581BAFBD3F88BCCFC28CDF8D826EC2085D657E0BD7 82A2126F7EBBBA5051FDF2CA482D79624A346F8DE472 4F31206FC37536FB530A654ABD07E5AD6615BB850DDE 0A04F0BA9A582A2C38F85600A2D6DB99739B4BD9A287 1C9A4F7D35B64C4707FD5A4C66FD9150154A05763AE5 03AFE77B7F60C97A88CB77728E9788235A89E725470B 9C6E278F4A7B009A125D8CA931B1CED67A796545661E 80C6C8021C326443C70E34C8968E71C1FDD22FF78BAA 99CD83B559D441657022E1BDB10BC76AC6AC5E24CCD0 33179DD4B8D4A74A02BABCBF39E6D8EF586E40446AD4 4C98D013CC226D262DFF2E845978C3C494D2A099F5BE 586420D26BB7F9768E61F3DE77EDA2AFEEF4005975F6 1363F88109B75093B10B3D34637ED4CAC3A013BB9B2F 6BCC9B7CB34D9069A7F9439EFB3048D847210EC1CBD8 E1602DAB9E0288D3B83ACB0E4E2B03EE2DE03B9C814A 13340AC4B423ACDC12F791295A0F19B1675269D90DCF C6FB2F970E0C9D62DE0F26E8120E5EF07797B9B47535 64ADE32B9F17 (size 4096 bits)
p-q	298C88B34296125BD7E34A56A7786C98BF9EAD1BD475 C491D3CB8DD5DA22999AAE9B7D320344EABE7EB2BCA1 2C9B399022025F24296643467D7B5F64040ED311C55E 94609B1F2237BFAC526C062D198C747398D041EE1C1C 6249AC234F634CD5991A51E120041859D436E5A96EF1 93F388CEFF00B4B7ED58C3DE795269C089CDDECF5D15 8BFFBB150D005BED0CE821FBF06F7CA747138B5DA52B 4FE7F75EDFBF5F900C79417EA1EE63ADFA93C7E0D9EE 67F3CABF1214672E0BBFD9E59930A905A805ECD83E42 5D713E169B16AB8AB1F8121A1AA18532EE4B1C496EE5 13F1EAD81D4B2CE814F7F0F94B4B96363461A26C7DF2 844BFC0DCF71126F908EC9DD7A766 (size 2050 bits)

Received: 7 March 2022
Revised: 14 October 2022
Accepted: 9 November 2022