On Incompressibility of RSA Public Keys

Oleg Mazonka, 2013

Abstract *Increasing computer power in time requires the increase of the size of public keys in order to retain the same level of cryptographic strength. This paper discusses the possibility of producing RSA public keys which can be represented in fewer numbers.*

1. Simple Public Key Generation

In this paper I would like to discuss a problem of representing RSA moduli in shorter form. RSA public keys have a deficiency that their moduli are big in comparison, for example, with shared cryptography keys. This is especially important for embedded systems with limited resources which are required to store many public keys of different parties.

While private keys can be efficiently represented in an arbitrarily short length, it seems that reducing public keys is somewhat difficult. The idea behind the reduction of an RSA modulus can briefly be described as following. Let P_W denote a word or phrase or any piece of information (a sequence of binary bits) revealed to public. Let publically known algorithm A_P generate a seed M for the RSA modulus – a big number.

$$M = A_P(P_W)$$

This algorithm must generate M in a deterministic way. Any usage of hashing functions is a good approach for this task.

Let S_W denote a password. Using an algorithm A_S (known or unknown to public) S_W can generate a prime number p – a private key – of a specified length:

$$p = A_S(S_W)$$

This is possible to do in $O(\ln p)$ time according to Prime Number Theorem. The simplest way is to generate a number and then check nearby numbers until a prime is found. Both algorithms A must obviously have complex enough structure so no simple mathematical function describes a set of generated numbers. This is easy to achieve by using, for example, hashing functions.

Let $|x|$ denote the length (number of bits) of a number x encoded in binary radix. Without losing generality one also may think of the length as a number of digits in decimal representation.

To construct an RSA modulus another prime q is required. It can be selected in such way that the product of p and q is close to M: $pq \approx M$. That can be done by sequentially checking numbers starting from $\lfloor M/p \rfloor$ and down until a prime is found, so that:

$$pq = M - m$$

where m is a small number in comparison to M. It is easy to see that p and q can be selected so that $m < q < p$ and $|m| \le |q| \le |p|$. However since there are no correlation between generation p and M (otherwise the private key is not secure), the length of m and the smaller factor q are very likely to be the same for reasonably big q[1].

Let us call a number of independent bits necessary to guess in order to efficiently recover p or q, cryptographic strength R. The length of q is $|q|$, and the cryptographic strength is less than

$$R < |q| - \ln|q|$$

because out of $|q|$ bits $|q| - \ln|q|$ bits can be predefined and fixed, and about $\ln|q|$ bits have to be altered to find a prime. The above inequality is quite strong because the public key *has* the complete information about the private key and recovering the private key is only limited by factorization algorithms. The situation is different with shared key cryptography where the cryptographic strength is close to the number of predefined bits.

At this moment public key can be identified by $|m| + |P_W| + |A_P|$ number of bits, and the private key can be regenerated from $|S_W| + |A_S|$ number of bits. $|A_P|$ and $|A_S|$ can be ignored here because these two algorithms might be in general knowledge and do not have to be defined explicitly; so that the public key is a pair of two entities (P_W, m) and the private key is just S_W. $|P_W|$ depends on user selection of the public phrase and can be arbitrary small without any security compromise. $|S_W|$ is a length of a user secret password. If the algorithm A_S is known to public, then a sufficient length of the password must be enforced to ensure the strength against brute force attack. In all other respect it can be as small as user wishes without compromising RSA security. At this point one can say that the private key is efficiently reduced because the user must remember only the secret password S_W to regenerate the private key. This is not

[1] Obviously the probability to find *m* shorter than *q* by *k* bits is 2^{-k}.

the case for the public key because it must include at least $|m|$ bits, number of which is equal to $|q|$ in the example above, which in turn cannot be smaller without reducing the cryptographic strength R as seen from the inequality for R above.

The question is now whether it possible to reduce $|m|$ retaining $|q|$ by cleverer selecting p and q?

2. Extended Euclidian Algorithm

The answer to the above question is positive. Consider that a pair p_0 and q_0 is selected:

$$p_0 q_0 = M - m_0$$

Ignore for now that the factors must be prime. We deal with it later. Let a new pair p_1 and q_1 be defined as

$$p_1 = p_0 + f$$
$$q_1 = q_0 + e$$

where *f* and *e* are some numbers:

$$p_1 q_1 = M - m_1$$
$$p_1 q_1 = (p_0 + f)(q_0 + e) = M - m_0 + p_0 e + q_0 f + ef$$
$$m_1 = m_0 - (p_0 e + q_0 f + ef)$$

If *f* and *e* can be chosen so that $m_1 < m_0$, then a new pair of p and q is selected and the process can be repeated. This seems not too difficult because using the Extended Euclidean Algorithm the expression $p_0 e + q_0 f$ can produce a small result with sufficiently small *f* and *e*.

Consider a Euclidean table:

a	b	c	d	e	f
a_0	b_0	c_0	d_0	e_0	f_0
a_1	b_1	c_1	d_1	e_1	f_1
.	.	.	.	.	.
a_n	b_n	c_n	d_n	e_n	f_n

with the following rules:

$a_0 = p_0$ $b_0 = q_0$	$c_i = a_i \bmod b_i$ $d_i = \lfloor a_i / b_i \rfloor$	$a_{i+1} = b_i$ $b_{i+1} = c_i$ $c_n = 0$	$\tilde{e}_j^{\,j} = 0$ $\tilde{f}_j^{\,j} = 1$	$\tilde{e}_i^{\,j} = \tilde{f}_{i+1}^{\,j}$ $\tilde{f}_i^{\,j} = \tilde{e}_{i+1}^{\,j} - d_i \tilde{e}_i^{\,j}$	$e_j = \tilde{e}_0^{\,j}$ $f_j = \tilde{f}_0^{\,j}$

The above rules ensure that

$$a_0 e_i + b_0 f_i = a_i \tilde{e}_j^{\,j} + b_i \tilde{f}_j^{\,j}$$
$$m_1 = m_0 - \left(a_i \tilde{e}_j^{\,j} + b_i \tilde{f}_j^{\,j} + e_i f_i\right) = m_0 - g_i$$

and the problem is reduced to finding an index *i* from the table such that $m_1 < m_0$, and a new pair of p and q is obtained by:

$$p_1 = p_0 + f_i$$
$$q_1 = q_0 + e_i$$

The task is not hard because the size of the Euclidean table is of $O(\ln p)$ order.

It seems that restricting *m* to be positive is artificial, and a better match can be found if *m* can positive or negative, because we are looking for only smaller *m* in absolute value. However since smaller *m* in absolute value is random, statistically is can be halved, or reduced in length by one bit. But exactly one bit is required to specify the sign of *m*. So overall, no gain is obtained by allowing *m* to be negative.

The rules above are straightforward as can be seen from a simple example when $p_0 = 91$ and $q_0 = 22$:

a	b	c	d	$\tilde{e}^2$	$\tilde{f}^2$	$\tilde{e}^1$	$\tilde{f}^1$	$\tilde{e}^0$	$\tilde{f}^0$	e	f
91	22	3	4	−7	29	1	−4	0	1	0	1
22	3	1	7	1	−7	0	1			1	−4
3	1	0	3	0	1					−7	29

The function g_i introduced above in the equation for m_1, becomes negative when *e* and *f* grow. Since we are interested only in positive values of g_i, each can be modified by changing value *f*. For example:

$$f \to f + 1 + \left\lfloor \frac{-g_i}{q + e_i} \right\rfloor$$

This replacement ensures that g_i is always positive and not greater than *q*.

As a more complex case let us select $M = 1234567890$ and $p_0 = 54321$, then $q_0 = 22727$ and $m_0 = 14523$:

a	b	c	d	e	f	$p_0 e + q_0 f + ef$
54321	22727	8867	2	0	1	22727
22727	8867	4993	2	1	-2	8865
8867	4993	3874	1	-2	5	4983
4993	3874	1119	1	3	-7	3853
3874	1119	517	3	-5	12	1059
1119	517	85	2	18	-43	-257
517	85	7	6	-41	98	-3933
85	7	1	12	264	-631	-166577
7	1	0	7	-3209	7670	-24613029

The greatest number in the last column which is less than $m_0 = 14523$ is in the second row, hence a new pair of p and q is $p_1 = 54319$, $q_1 = 22728$ and $m_1 = 5658$. This process can be repeated again to reduce m even further, for example is $p = 54877$ and $m = 21$. When m cannot be reduced the final pair of p and q is tested against cryptographic criteria[2] and repeated if necessary with different p_0 until the criteria are satisfied. This repetition process is still polynomial efficient because of Prime Number Theorem. To make this process work faster each time p and q can be tested for division by small numbers. If a number is found, then the original equation cancels out that factor. For example, if

$$p = kp'$$

then

$$p'q = \left\lfloor \frac{M}{k} \right\rfloor - \frac{m - M \bmod k}{k} = \lfloor M/k \rfloor - m'$$

Here the amount of information in public key is increased by ln*k* as *k* has to be specified, but at the same time *m* is reduced by exactly this amount. So, overall amount information stays at the same level. Of course, *M* and *p* are reduced and the cryptographic strength is reduced. This does not seem to be a problem though, because they can be set arbitrary large in the first place. The probability that both p and q are not primes after this procedure is greatly diminished making this algorithm work much faster.

Numbers $\tilde{e}_j^j$ and $\tilde{f}_j^j$ may not necessary be fixed to 0 and 1. If a matrix of their possible values is selected then chances to reduce *m* increase with the size of the matrix. But the benefit is at the expense of computational effort.

3. Gloomy Considerations

There are two drawbacks in the method described at the previous section. First, finding smaller m becomes more difficult when $|m| << |q|$ because larger *f* and *e* numbers required fitting smaller linear combination but the product *ef* grows at the

[2] The first and obvious criterion is that both factors are prime. The other might be conditions such as smoothness of +1 and -1 numbers, and primality of numbers made by taking off the one last and two last bits from both factors.

same time reducing chances of the expression $g = pe + qf + ef$ to become close to *m*. If that would not be the case, this method would be an efficient factorization method. Figure 1 shows the behaviour of the last column in the Euclidean table. The value shown is the average of the logarithm of squared expression: $\langle \ln g^2 \rangle$ depending on row index in the table. We can see that this function has a minimum corresponding to the expected lower limit of *m*'s size. When *m* becomes smaller than the minimum of the function the probability of finding $g < m$ dies out.

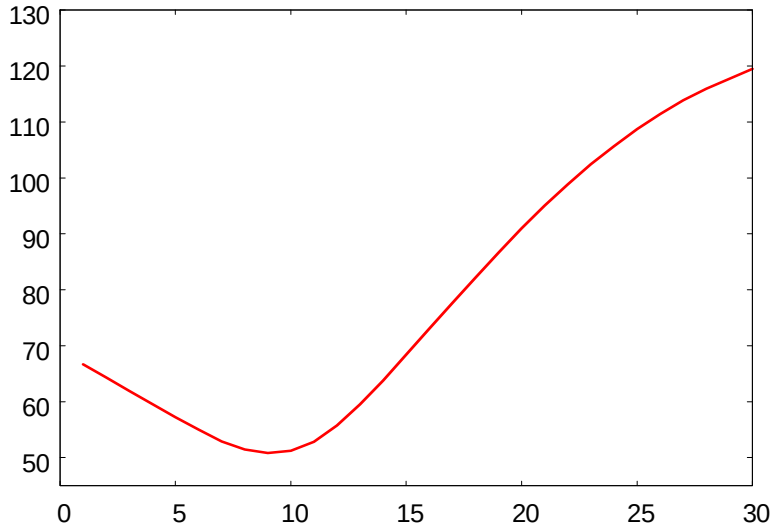

Figure 1 The value of the logarithm of the squared expression *g* depending on the row index at Euclidean table. This is the last step in the iterations reducing *m*. The plot shows the average value over 6000 samples of randomly selected *M* and *p* of exactly 100 and 50 bit sizes correspondingly.

For randomly chosen M and p the algorithm efficiently converges to roughly the third length of the modulus $|m| \approx |M|/3$ when factors are balanced: $|p| \approx |q|$. For unbalanced factors the size of m is bound and closer to q: $|m| \approx |q|$. Obviously m can be shrunk even further by seeding different p_0, but that process is exponential to the size of m hence inefficient.

For large numbers M one can assume:

$$|e| = |f|$$

Since $g = b + ef$, $|g|$ reduces with the row in the Euclidean table as $|b|$, but then grows as $2|e|$. The optimal value is when $|g| = |m|$ and $|b| = 2|e|$. On the other hand, $|e|$ grows exactly as much as $|b|$ decreases, so that $|p| = |b| + |e|$. Hence

$$|p| = 2|e| + |e| = 3|e| \quad \Rightarrow \quad |e| = |p|/3 \quad \Rightarrow \quad |m| = |M|/3$$

Figure 2 shows the average $|m|$ for 100 bit modulus and different size of factors. It is interesting to see that *m* is quite sharply plateaued, so its size can roughly be described as: $|m| \approx \min(|p|,|q|,|M|/3)$

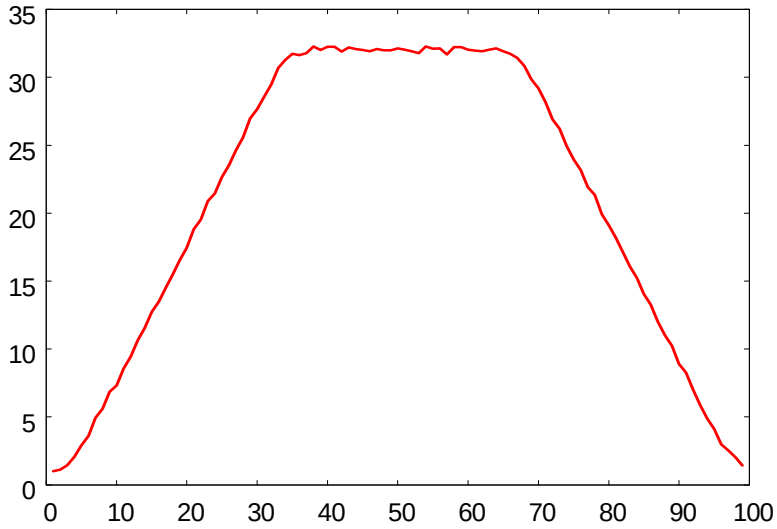

Figure 2 The *m*'s size depending on *p*'s size. *M* is a randomly selected 100 bit number. X-axis represent the size of randomly selected *p*. Y-axis is the average size of *m* out of 250 samples.

The second problem is that the cryptographic strength *R* is reduced since different p_0 generated by A_S may result in the same final p. In other words higher order bits are altered in the process of finding better matching pair of p and q. Since the amount of information is preserved, at least $|m|$ bits of p_0 have to be guessed by interloper to recover p. Hence the cryptographic strength *R* is limited by m:

$$R < |m|$$

That means that the original idea to reduce m is not quite appealing as it seemed at the beginning. To retain a predefined level of cryptographic strength *R*, the reduction of the public key is limited by a well-defined boundary.

This is, however, to some extent a controversial question because the factorization algorithms are developed with the aim to factorize big numbers. Having longer factors but selected by a well-defined and known algorithm may reduce or may increase the cryptographic strength. That is difficult to estimate without detailed analysis of the algorithm and factorization methods. The next section tries to shed light on the question from the empirical point of view.

4. Numerical Factorization

As a simple algorithm for generating numbers lets us define a number obtained by taking N bits from a very big number $7^{7^7} = {}^3 7 = 7\uparrow\uparrow 3$ at the position *u*, and denoting a pair $P_W = (N,u)$. For example, (10,0)= 721[1011010001] and (7,5)= 69[1000101]. That big number has more than two millions of binary digits, so there is plenty space to randomly choose from.

For example, for

$$P_W = (100,0) = M = 893000717014499785872557917781 \text{ [100 bits]}$$

$$S_W = (50,100) = p = 785606124574568 \text{ [50 bits]}$$

the algorithm finds

$$q = 785592493014349 \text{ [50 bits]}$$

$$m = 2385274828 \text{ [32 bits]}$$

The public key in this case is $\{100,0,2385274828\}$ and the private password is $\{50,100\}$. The program MSIEVE [1] factorizes the number $M - m$ in less than a second on my desktop computer.

Testing this program with many random numbers selected by changing u reveal the following interesting properties. First, there is no difference in time required to factor modulus obtained by the simple method described in Section 1 and the reduction method described in Section 2. This is obviously the result that the factoring program and the reduction method are independent and have no incidental correlations. Second, the time required to factor modulus, for this particular factoring program, does not depend on the size of the factors given that one is not too small. Figure 3 shows the comparison of average time required to factor 200-bit number with different length of factors. If the smallest factor is larger than 40-bit the time is almost constant.

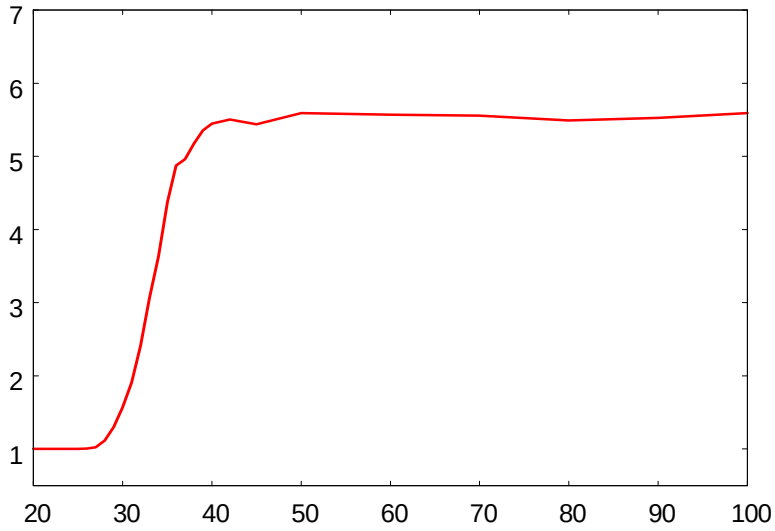

Figure 3 Geometric average time required to factor 200-bit number. X-axis is the size in bits of one factor; Y-axis is time in seconds (on a particular desktop PC). 1000 numbers sampled per each point.

Third, the average factoring time becomes significant after 200-bit size of the modulus. As expected it grows exponentially with the size of the number.

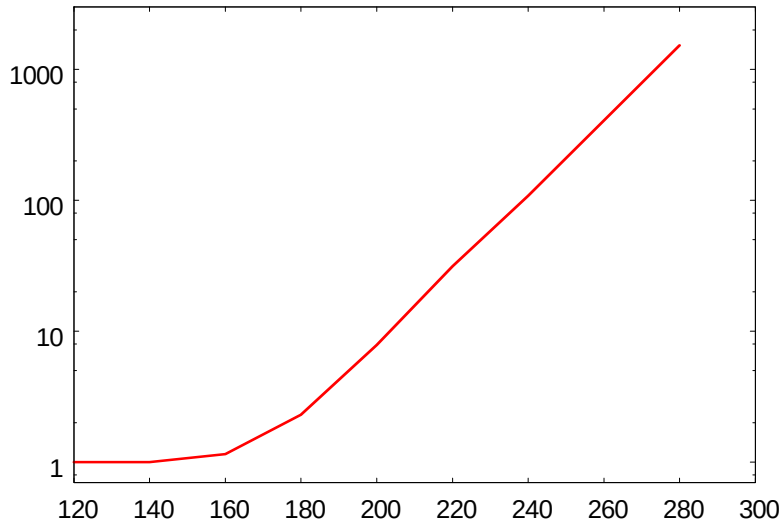

Figure 4 Geometric average time required to factor a number. X-axis is a size of *M*, and Y-axis is time in seconds (on a particular desktop PC). Factors are taken to be half of the *M* size and reduction algorithm from Section 2 is applied.

Figure 4 shows the average time required to factor a number of a particular size.

5. Final remarks

The simple algorithm of getting RSA public key presented with half of its digits (or bits) predefined by some arbitrary algorithm is also described in [2]. The algorithm described in Section 2 allows making one small step further in reducing the representation of a public key. However, there is a limitation to what extent the public key can be reduced as well as no theoretical base as for the retained cryptographic strength of the reduced public key.

References

1. Jason Papadopoulos, MSIEVE: A Library for Factoring Large Integers, www.boo.net/~jasonp
2. Arjen K. Lenstra, “Generating RSA Moduli with a Predetermined Portion”, Lecture Notes in Computer Science Volume 1514, 1998, pp 1-10.