Digitally Watermarking RSA Moduli

Anna M. Johnston*

February 19, 2001

Abstract

The moduli used in RSA (see [5]) can be generated by many different sources. The generator of that modulus knows its factorization. They have the ability to forge signatures or break any system based on this moduli. If a moduli and the RSA parameters associated with it were generated by a reputable source, the system would have higher value than if the parameters were generated by an unknown entity. An RSA modulus is digitally marked, or digitally trade marked, if the generator and other identifying features of the modulus can be identified and possibly verified by the modulus itself. The basic concept of digitally marking an RSA modulus would be to fix the upper bits of the modulus to this tag. Thus anyone who sees the public modulus can tell who generated the modulus and who the generator believes the intended user/owner of the modulus is.

Two types of trade marking will be described here. The first is simpler but does not give verifiable trade marking. The second is more complex, but allows for verifiable trade marking of RSA moduli.

The second part of this paper describes how to generate an RSA modulus with fixed upper bits.

Key Words and Phrases public key cryptography, RSA, prime generation.

1991 Mathematics Subject Class Primary 11Y99; 11A07

*Sandia National Laboratories, Albuquerque, New Mexico 87185-0449. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.Email: {`ajohnst@sandia.gov`}.

1 Marking RSA Moduli

The moduli used in RSA (see [5]) can be generated by many different sources. The generator of that modulus (assuming a single entity generates the modulus) knows its factorization. They have the ability to forge signatures or break any system based on this moduli. If a moduli and the RSA parameters associated with it were generated by a reputable source, the system would have higher value than if the parameters were generated by an unknown entity. So for tracking, security, confidence and financial reasons it would be beneficial to know who the generator of the RSA modulus was. This is where digital marking comes in.

An RSA modulus is digitally marked, or digitally trade marked, if the generator and other identifying features of the modulus (such as it's intended user, the version number, etc.) can be identified and possibly verified by the modulus itself. The basic concept of digitally marking an RSA modulus would be to fix the upper bits of the modulus to this tag. Thus anyone who sees the public modulus can tell who generated the modulus and who the generator believes the intended user/owner of the modulus is. A digitally marked modulus N of $2k$-bits, with $2n$-bits reserved for the tag would have the following form:

$$N = PQ = t2^{2(k-n)} + r$$

where t is the $2n$-bit tag, r is $2(k-n)$ bits of random data.

Two types of trade marking will be described here. The first is simpler but does not give verifiable trade marking. Any enforcement of the moduli trade marks would have to be done in the legal system. The second is more complex, but allows for verifiable trade marking of RSA moduli.

Several problems arise in trying to generate trade marked moduli. First, what sort of a tagging scheme should be used in order to obtain the desired results? The tagging scheme must generate strong moduli and for verifiable trade marking, it must prevent others from creating forged moduli and incorporate non-repudiation for the intended user.

Second, how do you generate the modulus such that the upper bits of the modulus are set to the given tag? The scheme to generate them must create random moduli which are resistant to factoring attacks.

The rest of this paper will describe techniques to solve both these problems.

2 Simple Non-Verifiable Tag

The simplest idea for adding the trademark to the modulus would be to fix the high order bits to the companies trade mark. This would make checking the trademark as easy as looking at the modulus. Since the trade mark is not verifiable (anyone could generate it), enforcement of the trademark would have to be done at the legal level.

Although unlikely, one small concern with having the upper bits fixed to a known static pattern is that this may lead to specialized attacks against their moduli. To prevent this problem, random data can be added (mod 2) to the trademark. Both the trademark added with the random data and the random data itself must now be set as the high order bits of N. This technique would dilute the non-random property of the high order bits. Retrieving a trademark would only require an exclusive-or of the first two high order portions of the modulus.

The format for the upper $2n$ bits of the modulus would be:

Tag Bits	Data
2n-1	1 (forces t to be 2n-bits long)
2n-2 to n	low order n-1 bits of trade mark exclusive or'ed with low order n-1 bits of random pattern r
n-1	1, to be xor'ed with high order trademark bit
n-2 to 0	$(n-1)$ bits of random pattern r

3 Verifiable Tags

The tag used in the verifiable trademark must have several properties. It must identify (verifiably) the generator and intended user of the modulus. It must prevent others from generating forged moduli. It must prevent the generator of the modulus from later claiming that the modulus did not originate from them.

All three properties scream for digital signatures, though digital signatures alone can not put in place these properties. However digital signatures **and** a small, fixed amount of extra data can.

3.1 Why the Added Data

The obvious way for a company to trademark it's RSA moduli is to set the tag t to the signature of the modulus identifier (which contains the generating and recipient company's ID) by the generator. Unfortunately this signature is not tied to the remaining bits of the modulus. This means that all someone has to do to forge a 'genuine' modulus from company XYZ for company ABC is strip the tag t from the modulus and then go through the modulus generation process to generate a new modulus.

This is where the extra data comes in. Along with the trade marked modulus, a separate signature of the modulus itself would be given. Letting $S(m)$ be the signature (and all accompanying data required for verification) for data m, the trade marked modulus generated by company XYZ for company ABC would look like:

$$N = S(\text{XYZ for ABC 1})|r$$
$$S(N)$$

If the separate signature is needed for the modulus, the obvious question arises: "Why bother with a trade marked modulus?" There are two reasons for this:

1. The separate signature is not within the modulus itself. Though the signature in the modulus is not truly verifiable on it's own it does give a compact simple way to identify the modulus.

2. The signatures within later moduli can verify previous moduli so that only one external signature is required for all moduli a company needs. This significantly simplifies the data management of the system.

For example if company ABC bought v RSA data sets from company XYZ, the data would look like the following:

$$\begin{gathered}
N_1 = S(\text{XYZ for ABC 1})|r_1 \\
N_2 = S(N_1|\text{XYZ for ABC 2})|r_2 \\
\vdots \\
N_v = S(N_{v-1}|\text{XYZ for ABC v})|r_v \\
S(N_v)
\end{gathered}$$

If ABC buys another data set from XYZ (data set $(v+1)$), XYZ would then generate:

$$\begin{gathered}
N_{v+1} = S(N_v|\text{XYZ for ABC v+1})|r_{v+1} \\
S(N_{v+1})
\end{gathered}$$

The old separate signature, $S(N_v)$, could then be replaced by the new separate signature, $S(N_{v+1})$. No matter how many data sets are required, only one extra signature would be needed to validate the entire set!

3.2 Which Signature Scheme?

One question not yet dealt with is what signature scheme should be used for the tags? As the signature will be contained within the RSA modulus itself, the scheme must use appreciably less data than RSA modulus. DSA (over any seucre group) and elliptic curve El Gamal (see [4] and [2]) are both well suited to this task. For example, using a 160-bit field for the elliptic curve to trade mark a 1024 bit RSA modulus requires 322-bits for the signature data. This would leave room for the actual ID portion and still fix less then half the bits.

3.3 Tag format

The tag must be formatted for use in the modulus generation scheme. The initial requirements are simple:

- The modulus will be $2k$-bits long;
- The trade mark tag will be $2n$-bits long.

In order to insure these formatting requirements the following properties for the tag must be met:

1. In order to minimize modulus generation time, the ratio $\frac{k}{n} > 2$ (less than half the bits for the tag);

2. The $(2n-1)$-bit of the tag must be a one;

Using DSA or El Gamal, the bit format for the tag is described in the following table.

Modulus Bits	**Tag Bits**	**Data**
$2k-1$	$2n-1$	1
$2k-2$ to $2k-n$	$2n-2$ to n	signature
$2k-n-1$ to $2k-2n+1$	$n-1$ to 1	public version of random data (the x-coordinate if elliptic curves are used)
$2k-2n$	0	If using elliptic curve cryptosystem, this bit indicates which y-coordinate is used: 1 implies the larger, 0 implies the smaller. This simplifies the verification process but is not required.

4 Modulus Generation

A trade marked RSA modulus will contain the trademark for the company which generated it. This trademark consists of a $2n$-bit long pattern, t, described by one of the above methods. Anyone seeing the modulus (N) should be able to extract the trademark and identify the generator. The modulus generation technique described below allows for RSA moduli to be generated for both trade mark schemes.

4.1 Required Variables

The following variables are the basic variables needed for generating a trade marked modulus. They include the modulus, the primes composing the modulus, the approximate size N (divided equally between the two primes), the number of bits in the trademark and the trademark itself.

P A prime integer;

Q A prime integer;

N The RSA modulus, $N = PQ$, $2k$-bits in length;

k The approximate binary size of P and Q (i.e., $P \approx Q \approx 2^k$);

2n The number of upper bits fixed in N $(n < \frac{k}{2})$;

t The $2n$ bit trade mark pattern generated by one of the versions above;

4.2 Basic Idea

The basic idea of this algorithm is to generate N by first representing N as PQ with the upper (2n) bits fixed.

$$PQ = t2^{2(k-n)} + h \tag{1}$$

In this equation h is the lower $2(k-n)$ bits of the modulus. Assuming P and Q are about the same size (k-bits), the bits of t will be formed mostly by the upper n bits of P and Q. Writing P and Q out in terms of the upper n and lower $k-n$ bits gives:

$$P = p_1 2^{k-n} + p_0$$
$$Q = q_1 2^{k-n} + q_0$$
$$0 < p_0, q_0 < 2^{k-n}$$

When P, Q are chosen in this way, bounds on the p_1, q_1, p_0, q_0 values become apparent.

$$t2^{2(k-n)} + h = p_1 q_1 2^{2(k-n)} + 2^{k-n}(p_1 q_0 + p_0 q_1) + p_0 q_0 \tag{2}$$

The algorithm begins by choosing p_1. This choice and the value of t will determine q_1 (or very tight bounds for q_1). These choices will determine the bound for p_0, which in turn determines the bound for q_0. Standard primality testing is done on P and Q make sure they are prime. The main focus of this paper is to define the bounds on p_1, q_1 given p_1, p_0 and q_0 given the previous values, and to describe a final algorithm for generating trade marked composite moduli.

4.3 Bounds for p_1

The bound for p_1 depends only on the desired size of P with respect to Q. The following equation forces P to be within $(2s)$-bits of Q.

$$2^{n-1+s} \leq p_1 < 2^{n+s}. \tag{3}$$

Setting $s = 0$ ($2^{n-1} \leq p_1 < 2^n$) generates P and Q of equivalent bit lengths. Randomly choose p_1 from this range.

4.4 Bounds for q_1

The choice of p_1 imposes strict upper and lower bounds on q_1. The upper bound for q_1 is very simple. The product of $p_1 q_1$ must be less than or equal to t. If this does not hold the top bits will NOT be equivalent to t (see equation 2). So: $q_1 \leq \frac{t}{p_1}$

The lower bound is a little more complicated. The bound is found here by solving equation 2 for q_1, then using it to compute a lower bound.

$$\begin{aligned}
q_1 &= \frac{2^{2(k-n)}t + h - 2^{k-n}p_1q_0 - p_0q_0}{2^{2(k-n)}p_1 + 2^{k-n}p_0} \\
q_1 &> \frac{2^{2(k-n)}t - 2^{2(k-n)}p_1 - 2^{2(k-n)}}{2^{2(k-n)}(p_1+1)} \\
q_1 &> \frac{t}{p_1+1} - 1
\end{aligned}$$

So the bounds on q_1 will be:

$$\frac{t}{p_1+1} - 1 < q_1 \leq \frac{t}{p_1} \tag{4}$$

In most cases this bound determines q_1.

4.5 Bounds on p_0

There are two requirements for p_0: it must lie within the required bounds and it must be chosen such that P is prime. The primality question will be handled with basic primality testing (see [1]). The numeric bounds follow.

As with the bounds for q_1, the bounds for p_0 are found by solving equation 2 for p_0 then, using this equation, finding the upper and lower bounds. There is one added restriction on both p_0 and q_0: they must both be greater than zero and less than 2^{k-n}.

$$p_0 = \frac{2^{2(k-n)}(t - p_1q_1) + h - 2^{k-n}p_1q_0}{2^{k-n}q_1 + q_0}$$

Lower Bound

$$\begin{aligned}
\frac{2^{2(k-n)}(t - p_1q_1 - p_1)}{2^{k-n}(q_1+1)} &< p_0 \\
MAX\left(0, \frac{2^{k-n}(t - p_1q_1 - p_1)}{q_1+1}\right) &< p_0
\end{aligned}$$

Upper Bound

$$\begin{aligned}
p_0 &< \frac{2^{2(k-n)}(t - p_1q_1 + 1)}{2^{k-n}q_1} \\
p_0 &< MIN\left(\frac{2^{k-n}(t - p_1q_1 + 1)}{q_1}, 2^{k-n}\right)
\end{aligned}$$

The combined bounds for p_0 are:

$$MAX\left(0, \frac{2^{k-n}(t - p_1q_1)}{q_1+1}\right) < p_0 < MIN\left(\frac{2^{k-n}(t - p_1q_1 + 1)}{q_1}, 2^{k-n}\right) \tag{5}$$

4.6 Bounds for q_0

As with p_0, there are two requirements for q_0: it must lie within the required bounds and it must be chosen such that Q is prime. The primality question will be handled with basic primality testing (see [1]). The numeric bounds follow.

As with the previous bounds for q_1, the bounds for q_0 are found by solving equation 2 for q_0 then, using this equation, finding the upper and lower bounds. Since all other variables have been chosen at this point, the bounds will be a little more restrictive. There is also the added restriction for q_0: it must be greater than zero and less than 2^{k-n}.

$$q_0 = \frac{2^{2(k-n)}(t - p_1 q_1) + h - 2^{k-n} p_0 q_1}{2^{k-n} p_1 + p_0}$$

Lower Bound

$$\frac{2^{2(k-n)}(t - p_1 q_1) - 2^{k-n} p_0 q_1}{P} < q_0$$

Upper Bound

$$q_0 < \frac{2^{2(k-n)}(t - p_1 q_1 + 1) - 2^{k-n} p_0 q_1}{P}$$

The combined bounds for q_0 are:

$$\begin{aligned} q_0 \quad &> \quad MAX\left(0, \frac{2^{2(k-n)}(t - p_1 q_1) - 2^{k-n} p_0 q_1}{P}\right) \\ &< \quad MIN\left(\frac{2^{2(k-n)}(t - p_1 q_1 + 1) - 2^{k-n} p_0 q_1}{P}, 2^{k-n}\right) \end{aligned} \tag{6}$$

4.7 Algorithm

In the following algorithm the variables p_1, q_1, p_0 and q_0 are chosen, in this order, to generate the primes P and Q. The bounds for choosing the first three variables should insure that the range for q_0 is not empty. However it is possible that no primes lie in that range. In which case the algorithm should be backed up (find a new p_0) and a new q_0 searched for. The generated primes will both be k-bits long.

Algorithm to Generate Signature RSA Modulus

1. Choose random p_1 with $2^{n-1} \leq p_1 < 2^n$;
2. Choose random q_1 with $\frac{t}{p_1+1} - 1 < q_1 \leq \frac{t}{p_1}$;

3. Choose a random p_0 value such that $P = p_1 2^{k-n} + p_0$ passes primality tests and satisfies the bounds from equation 5:

$$MAX\left(0, \frac{2^{k-n}(t - p_1 q_1)}{q_1 + 1}\right) < p_0 < MIN\left(2^{k-n}, \frac{2^{k-n}(t - p_1 q_1 + 1)}{q_1}\right)$$

4. Choose q_0 such that $Q = q_1 2^{k-n} + q_0$ passes primality tests satisfies the bounds from equation 6:

$$MAX\left(0, \frac{2^{2(k-n)}(t - p_1 q_1) - 2^{k-n} p_0 q_1}{P}\right) < q_0 < MIN\left(\frac{2^{2(k-n)}(t - p_1 q_1 + 1) - 2^{k-n} p_0 q_1}{P}, 2^{k-n}\right)$$

5. If no q_0 can be found, choose a new p_0. If no p_0 can be found choose a new q_1. If no q_1 can be found choose a new p_1.

5 Example

In the following example an 80-bit RSA modulus will be generated with a 20-bit fixed pattern. Let $t = 0xfafab$. Other initial input variables are $k = 40$ and $n = 10$.

1. Compute the bounds for and choose p_1:

$$2^9 \geq p_1 < 2^{10}$$

$$\mathbf{p_1 = 0x22f}$$

2. Compute the bounds for and choose q_1:

$$\begin{aligned} \frac{0xfafab}{0x22f + 1} - 1 < \quad & q_1 \quad \leq \frac{0xfafab}{0x22f} \\ 0x72a < \quad & q_1 \quad \leq 0x72f \end{aligned}$$

$$\mathbf{q_1 = 0x72c}$$

3. Compute the bounds for and choose a trial p_0.

$$0x274c81e1 < p_0 0x3ad74910$$

$$\mathbf{p_0 = 0x2dedaef5}$$

4. $P = 2^{30} p_1 + p_0 = 0x8bededaef5$ is not prime. Advance p_0 within range until a prime P is found.

$$\begin{aligned} \mathbf{p_0} &= \mathbf{0x2dedaf03} \\ \mathbf{P} &= \mathbf{0x8bededaf03} \end{aligned}$$

5. Compute the bounds for and choose a trial q_0.

$$0x2a3dd904 < q_0 < 0x2a5b1e9e$$

$$\mathbf{q_0 = 0x2a577515}$$

6. $Q = 0x1cb2a577515$ is not prime. Advance q_0 within range until a prime Q is found.

$$\begin{aligned} \mathbf{q_0} &= \mathbf{0x2a57751d} \\ \mathbf{Q} &= \mathbf{0x1cb2a57751d} \end{aligned}$$

7. Multiply the two primes to create the trademarked RSA modulii:

$$\mathbf{N = PQ = 0xfafabdffa8758cee3257}$$

References

[1] D. M. Bressoud, *Factorization and Primality Testing*, Springer-Verlag, 1989.

[2] T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, in Advances in Cryptology – CRYPTO, no. 196 in Lecture Notes in Computer Science, 1985.

[3] G. Hardy and W. Wright, *An Introduction to the Theory of Numbers*, Oxford Science Publications, New York, fifth ed., 1979.

[4] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[5] R. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, (1978), pp. 120–126.