RSA Chips (Past/Present/Future)*

(Extended abstract)

Ronald L. Rivest
MIT Laboratory for Computer Science
Cambridge, Mass. 02139

Brief Abstract We review the issues involved in building a special-purpose chip for performing RSA encryption/decryption, and review a few of the current implementation efforts.

* This research was supported by NSF grant MCS-80-06938.

T. Beth, N. Cot, and I. Ingemarsson (Eds.): Advances in Cryptology - EUROCRYPT '84, LNCS 209, pp. 159-165, 1985.

I. Review of the RSA Cryptosystem

The "RSA cryptosystem" [RSA78] was the first published solution to the problem of implementing a public-key cryptosystem [DH76] – a concept invented by Diffie and Hellman. It remains today as the preeminent proposal for practical use. In this paper we review some of the considerations involved in implementing the RSA cryptosystem with special-purpose VLSI chips.

We begin by reviewing the RSA cryptosystem itself. The reader who wishes a more detailed review of public-key cryptography might consult [De82], [DH76], [DH79], or [RSA78].

A user A of the RSA cryptosystem creates his keys as follows:

- He first chooses at random two large (e.g. 100 decimal digit) prime numbers p and q.
- He then multiplies them together to get his public modulus $n = p \cdot q$.
- He then chooses at random a large integer d which has no divisors in common with either $p-1$ or $q-1$.
- He then computes e as the multiplicative inverse of d, modulo $(p-1)\cdot(q-1)$.
- He publishes as his secret key the pair (e, n), and keeps as his secret key the pair (d, n). (He may also wish to keep as part of his secret key the primes p and q.)

Anyone else can then encrypt a message M for A using A's public key, resulting in ciphertext C, using the equation:

$$C = M^e \pmod{n}.$$

Similarly, A can decrypt the ciphertext C using the equation:

$$M = C^d \pmod{n}.$$

As an example, if we choose $p = 47$ and $q = 59$, we have $n = 2773$. If we then choose $d = 157$ we can compute $e = 17$ using the technique given in [RSA78]. The public key is then $(e, n) = (17, 2773)$ and the secret key is $(d, n) = (157, 2773)$. The message $M = 31$ can be encrypted using the public key to obtain the ciphertext $C = 31^{17} = 587 \pmod{2773}$; decrypting yields the original message back: $31 = 587^{157} = 31 \pmod{2773}$.

II. Security of the RSA Cryptosystem

The security of the RSA cryptosystem depends on the difficulty for the enemy of factoring the published modulus n. If the enemy can factor the number n, he can compute the secret key (d, n) and read all of A's private mail (or forge A's digital signatures).

The security of the RSA cryptosystem is not known to be *equivalent* to the problem of factoring; it may be possible to break the RSA cryptosystem without factoring n. However, the most efficient attacks found to date are all provably equivalent to factoring. One can prove that computing the secret key is equivalent to factoring, and some variations on the basic RSA scheme are provably equivalent to factoring for some attacks (see [Ra79, Wi80]).

One interesting result, due to Andy Yao, is that the RSA system is "uniformly secure" in the sense that there can be no large sets of "weak messages": if an enemy can decrypt a significant fraction of messages encrypted with the RSA cryptosystem, then he could effectively decrypt all messages. Putting it another way, if the RSA cryptosystem offers security for the encrypted messages, then it offers uniformly high security for all messages. This follows from the multiplicative nature of the RSA scheme.

Even stronger results along this line have been proven by a number researchers (see [ACGS84] and its extensive list of references). The essence of these results is that if the RSA cryptosystem is secure, then the enemy will not even be able to get various kinds of partial information about the message from the ciphertext. (If he could, he would be able to get the whole message.)

IIA. How Hard is Factoring?

The best available algorithms for factoring large composite integers have a running time which is proportional to:

$$e^{\sqrt{\ln(n)\cdot\ln(\ln(n))}}.$$

for factoring a k-bit number n. A very crude approximation to this, in the range we are interested in, is:

$$5\cdot 10^{9+(\frac{k}{50})}.$$

In the range of interest, the difficulty of factoring seems to grow roughly one order of magnitude more difficult with each extra 50 bits (15 decimal digits) of modulus.

At the moment, using available supercomputers, numbers with 71 digits can be factored in a reasonable length of time. Numbers with up to 100 decimal digits are plausibly factorable in the future using the best available algorithms and special-purpose hardware.

If we take as a bench-mark data point that a 75-digit number can be factored in about one day with today's technology, and using the above formulas, we can derive the following table:

75 digits – $9\cdot 10^{12}$ operations – 1 day
100 digits – $2\cdot 10^{15}$ operations – 255 days
125 digits – $3\cdot 10^{17}$ operations – 103 years
150 digits – $3\cdot 10^{19}$ operations – 9,755 years
175 digits – $2\cdot 10^{21}$ operations – 70 thousand years
200 digits – $1\cdot 10^{23}$ operations – 36 million years
225 digits – $5\cdot 10^{24}$ operations – 1 billion years
250 digits – $2\cdot 10^{26}$ operations – 60 billion years
300 digits – $1\cdot 10^{29}$ operations – $5\cdot 10^{13}$ years

In our original paper [RSA78] we proposed that 200 decimal digits (around 664 bits) would be a reasonable modulus size; we still feel that this is a reasonable choice.

III. Implementation Basics and the Need for Special-Purpose VLSI

III.A. Implementation Basics

Multiplication of two k-bit integers takes time:

- $O(k^2)$ on a microcomputer using a standard algorithm,
- $O(k)$ with special-purpose serial/parallel multiplication hardware ($O(k)$ gates),
- $O(\log k)$ with special-purpose parallel-parallel multiplication hardware ($O(k^2)$ gates).

Using today's technology, the serial-parallel approach seems the best trade-off point.

Modular multiplication of two k-bit integers modulo a third k-bit integer takes time:

- $O(k^2)$ on a microcomputer using standard algorithms,
- $O(k)$ with special-purpose hardware ($O(k)$ gates),
- $O((\log k)^{1+\epsilon})$ with special-purpose hardware ($O(k^2)$ gates).

Again, with today's technology, the $O(k)$-time, $O(k)$-hardware approach seems best.

Modular exponentiation is an interesting computational problem in that it seems intrinsically "sequential": using extra hardware or extra parallelism doesn't seem to help beyond the amount it helps to speed up the underlying modular multiplications. To raise a k-bit number to a k-bit power modulo a k-bit modulus thus seems to require $O(k)$ multiplications. We have the available time/hardware tradeoff choices:

- $O(k^3)$ time on a microcomputer (e.g. 2 minutes for 200 digits),
- $O(k^2)$ time using $O(k)$ gates (e.g. 0.5 seconds for 200 digits),
- $O(k \cdot \log k)$ using $O(k^2)$ gates (e.g. 7 milliseconds for 200 digits).

The corresponding data rates would then be

- 5 bits/second on a microcomputer,
- 1330 bits/second with $O(k)$ gates, and
- 95K bits/second with $O(k^2)$ gates.

Key generation has two parts: finding large primes and computing e from d. The first part is the most expensive; it requires approximately $O(k)$ primality tests to locate a k-bit prime, and each primality test requires one modular exponentiation. We thus have that the expected time to find two large prime numbers is:

- $O(k^4)$ on a microcomputer (e.g. 20 minutes for 100-digit primes),
- $O(k^3)$ using $O(k)$ gates (e.g. 5 seconds for 100-digit primes),
- $O(k^2 \log k)$ using $O(k^2)$ gates (e.g. 70 milliseconds for 100-digit primes).

We note that so-called "strong" primes are not intrinsically more difficult to find that random primes. (See the paper by J. Gordon in this proceedings.)

The second step of generating an RSA key-set, finding e from d, is not harder than modular exponentiation, since we have the relation:

$$e = d^{\phi(\phi(n))-1} \pmod{n}.$$

Another approach, using the extended Euclidean algorithm for finding greatest common divisors, can also be used (see [RSA78] for details). The algorithm chosen here doesn't matter much since the bulk of work for key-generation will be in finding the large prime numbers.

III. B. Implementation ideas for speed.

The following ideas may help speed up an implementation, over and above the basic approach outlined above.

A *fast clock rate* may of course be very helpful.

Using a *short encryption exponent* (e.g. $e = 3$, as suggested by Knuth [Kn81, p. 386]) gives a 300-fold or so improvement in the speed of encryption and signature verifications (operations which use the public key), but does not help with decryption or signing (operations which use the secret key). This trick can not be used on d as well, since the length of e plus the the length of d should be approximately the length of n. Furthermore, if d is short it could be guessed, so a short d provides little security.

Using the *Chinese Remainder Theorem* – working modulo p and modulo q separately – can help speed up decryption and signing by a factor of 4 on a microcomputer and a factor of 2 to 4 using $O(k)$ hardware.

There are two basically different *exponentiation algorithms* one may use: the *left-to-right* algorithm and the *right-to-left* algorithm. These algorithms examine the bits of the exponent in different orders. Suppose the exponent e has a binary representation of $e_{k-1}e_{k-2}\ldots e_1e_0$. Then the algorithms for computing a ciphertext C from a message M both begin by setting C to 1, and then proceed as follows:

- The *Left-to-Right* Algorithm: for i from $k-1$ down to 0, this algorithm first sets C to C^2 (mod n) and then, if $e_i = 1$, sets C to $C \cdot M$ (mod n).
- The *Right-to-Left* Algorithm: for i from 0 up to $k-1$, this algorithm first sets C to $C \cdot M$ (mod n) if $e_i = 1$, and then (in any case) sets M to M^2 (mod n).

If the *left-to-right* algorithm is used, then the number of modular multiplications required in the worst case can be reduced from $2 \cdot k$ to $k + (\frac{k}{t})$ by precomputing a table of $M^1, \ldots, M^{2^t-1}$ (i.e. by modifying the left-to-right algorithm to consider the exponent e in radix 2^t instead of radix 2).

If the *right-to-left* algorithm is used, then by using twice as much hardware one can obtain a two-fold speed-up, since each squaring modular multiplication can be performed *in parallel* with the "accumulation" modular multiplication.

We note that the above two optimization techniques are incompatible, since they require different underlying exponentiation algorithms.

An elegant approach for speeding up the computation is to perform modular multiplication *directly*, rather than first performing an integer multiplication and then reducing the result modulo n as a separate step. This can yield a six-fold (approximately) speed-up, since the modular multiplication of two k-bit numbers can now be performed in approximately k clock cycles instead of approximately $6 \cdot k$. (see [Br82]).

IV. Overview of Existing/Planned Chips

In this section we review briefly six designs for RSA chips. These reviews are brief, and only itended to give the reader a feel for the kinds of chips possible with today's technology. For more details the reader should consult the references. Also, there are other chips in the design stage for which no references exist; these chips are not listed here.

IV.A. The "first" RSA chip

This chip was designed by Rivest, Shamir, and Adleman, and is described in [Ri80].

It was a single-chip nMOS design; using 4-micron design rules, the chip occupied 42 mm^2. It contained a 512-bit ALU in bit-slice design with eight 512-bit registers for storage of intermediate results, carry-save adder logic, and up-down shifter logic. The 224-word microcode ROM contained control routines for encryption, decryption, finding large primes, gcd, etc. It used a 5V supply, and drew approximately 1 watt of power. It contained approximately 40,000 transistors. It communicated with a host microprocessor using an 8-bit I/O port. The encryption rate was designed to be slightly in excess of 1200 bits/second. Due to an as yet undiagnosed error in the memory cell design, this chip never worked reliably.

IV.B. The NEC/Miyaguchi Design

This chip design was described in [Mi82]; I do not know if it was ever fabricated.

The design was for a cascadable chip set, with each chip having a 2-bit slice. (So 333 chips would be needed for a 200 decimal digit modulus.) Each chip would contain a 2 by 8 multiplier;

multiplication would be done byte-wise (8 by n). An encryption rate of 50,000 bits/second was claimed possible for a 512-bit modulus using this design, or 29,000 bits/second using a 200 decimal-digit modulus.

IV.C. The First Sandia Design

This chip, described in [RSWB82], used a two-chip set to work with numbers up to 336 bits in length. Each of the two chips is identical and could perform a modular multiplication of 336-bit numbers. Using the right-to-left exponentiation algorithm, one chip repeatedly squared the message while the other chip accumulated the product of the desired powers.

The chip was fabricated using 3-micron CMOS technology; the total area of the chip is 41 mm^2. With a 20Mhz clock rate the chip can encrypt one block in 0.8 second – a rate of 420 bits/second. The chip works correctly.

IV.D. The Second Sandia Design

This design is still in progress. The mathematics involved are described in [Br82]; the chip performs modular multiplications directly.

The chip will be cascadable; the first chips made are likely to be a 128-bit slice of the set. (For 512-bit moduli, four chips would be needed.)

IV.E. The "RSA Security" Design

RSA Security, Inc., a new start-up in the data-encryption area, is designing an RSA chip for commercial use [RSA84]. Currently in the design stage, the chip should be available in sample quantities in mid-1985.

Using 3-micron CMOS design rules, the chip should be approximately 47 mm^2 in size.

It will be able to handle numbers up to 200 decimal digits (664 bits) in length, and should be able to do one encryption in under 65 milliseconds (i.e. the data rate should be in excess of 9600 bits/second for a full-size modulus).

V. The Future...

It is interesting to observe that seven years ago, when the RSA cryptosystem was invented, the task of implementing the RSA scheme in a reasonably secure manner was quite expensive. (For example, we built a $3000 TTL implementation that could only handle numbers slightly over 300 bits in length.) Today, a very secure implementation (664 bits) fits nicely on one chip. Seven years from now we may move from a 3-micron technology to a submicron (say 0.3 micron) technology, giving a 100-fold reduction in area. In this case the same RSA implementation will take only 1% of a typical chip. The steady progess of technology will clearly make cryptography so cost-effective that no information system that handles data that is at all sensitive or that needs to be authenticated can afford to do without it.

REFERENCES

[ACGS84] Alexi, W., B. Chor, O. Goldreich, and C. P. Schnor, "RSA/Rabin Bits are $1/2 + \frac{1}{poly(\log N)}$ Secure," *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, (Singer Island, 1984).

[Br82] Brickell, E. F., "A Fast Modular Multiplication Algorithm with Applications to Two-Key Cryptography," *Advances in Cryptology - Proceedings of CRYPTO 82*, (ed. by Chaum et. al) (Plenum 1983), 51-60.

[De82] Denning, D. CRYPTOGRAPHY AND DATA SECURITY, (Addison-Wesley, Reading, Mass., 1982).

[DH76] Diffie, W. and M. E. Hellman, "New Directions in Cryptography", *IEEE Trans. Info. Theory* IT-22 (Nov. 1976), 644-654.

[DH79] Diffie, W. and M. E. Hellman, "Privacy and Authentication: An Introduction to Cryptography, *Proc. of the IEEE* 67,3 (March 1979), 397-427.

[Kn81] Knuth, Donald E., SEMINUMERICAL ALGORITHMS - The Art of Computer Programming (Vol. 2 - Second Edition), (Addison-Wesley 1981).

[Mi82] Miyaguchi, S., "Fast Encryption Algorithm for the RSA Cryptographic System," *Proceedings COMPCON 82.*

[Ra79] Rabin, Michael. "Digitalized Signatures as Intractable as Factorization," MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-212 (Jan. 1979).

[Ri80] Rivest, R. L., "A Description of a Single-Chip Implementation of the RSA Cipher," Lambda 1 (Fourth Quarter 1980), 14-18.

[RSA78] Rivest, R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. of the ACM* (Feb. 1978), 120-126.

[RSA84] RSA Security, Inc. (1717 Karameos Drive, Sunnyvale, CA 94087) "Preliminary Data Sheet for the RSA Cryptochip," (1984).

[RSWB82] Rieden, R. F., J. B. Snyder, R. J. Widman, and W. J. Barnard, "A Two-Chip Implementation of the RSA Public-Key Encryption Algorithm," Digest of Papers for the 1982 Government Microciruit Applications Conference (November 1982), 24-27.

[Wi80] Williams, H. C., "A Modification of the RSA Public-Key Cryptosystem," *IEEE Trans. Info. Theory* IT-26 (Nov. 1980), 726-729.

[Wi84] Williams, H. C., "Some Public-Key Crypto-Functions as Intractable as Factorization," *Proceedings of CRYPTO 84* (Springer 1984).