VICTOR
an efficient RSA hardware implementation

Holger Orup
Erik Svendsen
Erik Andreasen

Computer Science Department, Aarhus University
Ny Munkegade 116
DK-8000 Aarhus C, DENMARK
e-mail: orup@daimi.aau.dk

Abstract

The latest improvements of RSA chips are based on progress in implementation technology and strategy i.e. smaller circuits and higher clock frequencies. There has been no improvements in efficiency of the algorithms. The efficiency is here defined as the number of bits produced pr. 1000 clock cycles.

We present algorithms which improve the efficiency by 300%–400%. The main strategy is multiple bit scan and parallel execution of two multiplications. Using these algorithms and the presented hardware architecture a bit rate greater than 90 Kbit/sec. can be achieved encrypting 512 bit blocks.

1 Introduction

Several implementations or suggestions of how to implement the RSA protocol in hardware have been presented in the past. Brickell made an overview of existing RSA chips. The three implementations with the highest bit rate, when the length of an encryption block is 512 bits, are shown in table 1 [Bri89] [EMI88].

We have defined the efficiency as the number of bits produced pr. 1000 clock cycles. Note that the efficiency of the three implementations are approximately the same, and the difference in bit rate is due to the difference in clock speed. The efficiency as defined here is a performance measure of the algorithm used. On the other hand, the clock rate is a rough performance measure of the technology and methods used for realizing the algorithm in hardware.

I.B. Damgard (Ed.): Advances in Cryptology - EUROCRYPT '90, LNCS 473, pp. 245-252, 1991.

	Ω	Bit rate pr. 512 bits	efficiency
Thorn EMI	24 MHz	29.0 K	1.21
AT & T	15 MHz	19.0 K	1.27
Cryptech	14 MHz	17.0 K	1.21

Table 1: *The three fastest RSA chips to date*

Apparently there has been no development of more efficient algorithms suited for hardware implementation. In the following, we will present algorithms for exponentiation and multiplication which result in a higher efficiency than the above mentioned.

2 Exponentiation

The main operation in the RSA protocol is $M^E \bmod N$, where the length of each operand is at least 500 bits. Therefore it is essential to have an efficient exponentiation algorithm. The most commonly used algorithm is named *Russian Peasant* [Knu69]. Below is shown a variant in which E is read from the least significant bit. The i'th bit of E is denoted e_i.

Algorithm: Modulo exponentiation.
Stimulation: E, M, N, where $E \geq 0$ and $0 \leq M < N$.
Response: $X = M^E \bmod N$
Method:
$i := 0$
$X := 1$;
WHILE $i < n$ DO
    IF $e_i = 1$ THEN $X := (X \cdot M) \bmod N$ END;
    $M := (M \cdot M) \bmod N$;
    $i := i + 1$;
END;

Algorithm 1: *Variant of Russian Peasant for modulo exponentiation*

If we denote the length of M,E and N by n the time complexity is:

$$T[\text{Exp}, n] = \frac{3}{2} n T[\text{Mult}, n]$$

Assuming it is possible to perform *two* multiplications in parallel, this is indeed possible because the three statements in the loop do not depend on each other, the complexity is:

$$T[\text{Exp}, n] = n T[\text{Mult}, n]$$

This gives a 33% time reduction compared with the variant with one multiplication unit.

3 Multiplication

To implement the exponentiation algorithm mentioned above, we need an efficient way to perform modulo multiplication. Several algorithms have been presented [HDVG88] [Bar86] [ORSP86] [Bla83]. None of them are able to carry out a multiplication with fewer than n full additions of n-bit words.

The usual way of multiplying is by scanning the multiplier one bit at a time and conditionally accumulating the multiplicand parallelly. Assume we scan the multiplier k bits at a time, corresponding to base 2^k, we can express the serial-parallel multiplication scheme as in algorithm 2. In this algorithm S is the accumulator, n' is the number of

$S := 0$; $i := n' - 1$;
WHILE $i \geq 0$ DO
    $S := (2^k S + a_i B) \bmod N$;
    $i := i - 1$;
END;

Algorithm 2: *Serial-parallel multiplication with integrated modulo reduction*

digits in base 2^k, a_i is digit number i of the multiplier, B the multiplicand and N the modulus. The multiplier is scanned from the most significant digit.

The modulo reduction can be carried out by subtracting N from S until $S \in [0; N[$. The maximal number of subtractions will be $2^k + 2^k - 2$, because $a_i \in [0; 2^k - 1]$ and $B \in [0; N]$. Even though the number of subtractions is limited, this method is rather slow. Instead we can estimate the quotient, S div N, belonging to $[0; 2^{k+1} - 2]$ and carry out the reduction in *one* subtraction. This is shown in algorithm 3. Note that this method

$S := 0$; $i := n' - 1$;
WHILE $i \geq 0$ DO
    $q :=$ estimate(S div N);
    $S := 2^k S + a_i B - 2^k qN$;
    $i := i - 1$;
END;
Correction of S;

Algorithm 3: *Modulo multiplication with quotient estimation*

is only feasible if we are able to generate the products $a_i B$ and qN rapidly. We could for example precalculate all the possible values of $a_i B$ and qN and save them in a table. According to Barrett [Bar86], the quotient estimate can be found by multiplying a few

of the most significant bits of the dividend S and the reciprocal of the divisor N. We assume that the necessary amount of bits of $\frac{1}{N}$ is part of the input to the chip. If q is to have an accuracy of x bits, then by using $x+2$ bits from S and $\frac{1}{N}$ we get an estimate which at the most is one less than the exact quotient.

In algorithm 3 the accumulator S does not necessarily belong to the interval $[0;N[$ after each iteration. This does not matter, as long as S belongs to the same residue as S mod N, and S does not diverge. But after the loop S has to be corrected by subtracting N until S belongs to the correct interval. It is proven in [OS90] that $S \in [0;(3 \cdot 2^k - 1)N]$ after each iteration. The range of q is therefore $[0;3 \cdot 2^k - 1]$. As we shall see later, we can construct hardware that generates qN efficiently if the range of q belongs to $[0;42]$, this means the scan factor k is limited to 3.

We are able to reduce the range of q, the idea is as follows: if we estimate $\frac{S}{2}$ div N instead of S div N, the range of the quotient is apparently halved. The modulo reduction is performed by subtracting $2 \cdot 2^k qN$ instead of $2^k qN$. However, the accuracy of the quotient estimate is hereby reduced, implying an increase of the range of S. A closer analysis [OSA90] shows that if we estimate $\frac{S}{2^r}$ div N we get a minimal range of q when $k \leq r$: $[0;2^{k+1}]$. This means that the scan factor can be increased to 4. The final algorithm for modulo multiplication is shown as algorithm 4. Note that the final corrections can be made by iterating two extra times while setting $a_i = 0$ and further more assuming $r = k$.

Algorithm: Modulo multiplication.

Stimulation:
$A = a_{n'-1}a_{n'-2}\cdots a_0 a_{-1}a_{-2}$,
where $a_i \in [0;2^k-1]$, $a_{-1} = a_{-2} = 0$ and $n' = \lceil \frac{n+1}{k} \rceil$;
B, where $B \in [0;2N[$;
N, where $N \in]2^{n-1};2^n[$;
k, where $k \geq 3$;
r, where $r = k$.

Response:
S div $2^{2k} \equiv_N AB$ and
S div $2^{2k} \in [0;2N[$.

Method:
```
S := 0; i := n' - 1;
WHILE i ≥ -2 DO
        q := estimate(S div 2^r, N);
        S := 2^k S + a_i B - 2^{k+r} qN;
        i := i - 1;
END;
```

Algorithm 4: *Modulo multiplication*

The final result is read from S discarding the $2k$ least significant bits. Note that this result belongs to the interval $[0;2N[$. A further reduction is not necessary. When the exponentiation algorithm terminates, the result will also belong to $[0;2N[$, here a

reduction is necessary. This reduction is easily carried out while outputting the result serially. The correctness of the algorithm is proven in [OS90]. The time complexity is:

$$T[\text{Mult}, n] = (\lceil \frac{n+1}{k} \rceil + 2)T[\text{loop}]$$

In the rest of this paper we will describe how to perform the central operation of the loop: $S := 2^k S + a_i B - 2^{k+r} qN$. To do this we have to take a closer look at the hardware architecture of the multiplication unit.

3.1 Hardware architecture

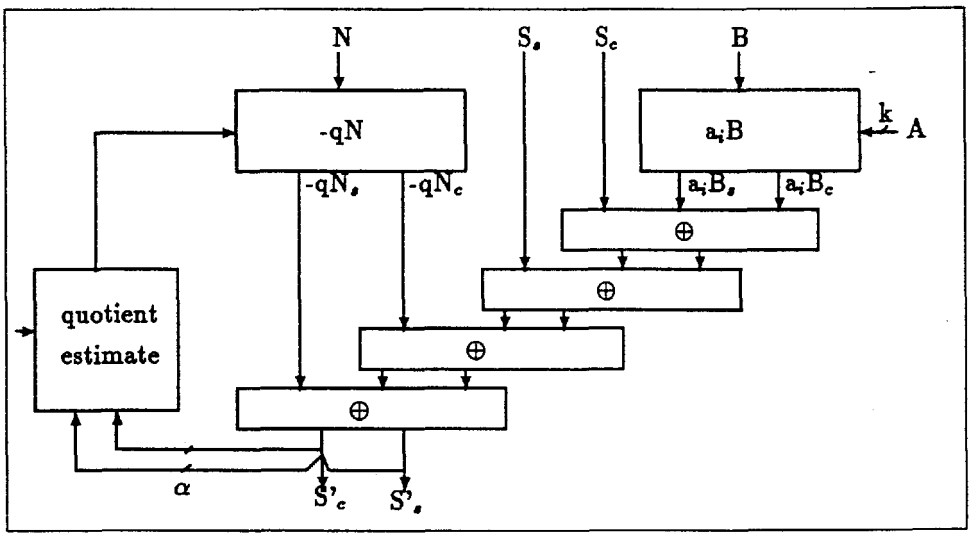

Figure 5: Hardware architecture of the multiplication unit

The multiplication unit consists of circuits for generating the values $-qN$ and $a_i B$. Each circuit returns the result represented in two words, i.e. the value $a_i B$ is represented as $a_i B_s$ and $a_i B_c$, where $a_i B_s + a_i B_c = a_i B$. Similarly the accumulator S is represented in two words S_s and S_c. The main task of the loop is now to add six words together and represent the sum as two words S'_s and S'_c. Using the carry save addition technique this is easily done with four rows of fulladders as shown in figure 5. The critical path of the multiplication unit is calculating the quotient estimate, generating $-qN$ followed by the delay of two fulladders. To be able to use the parallel version of the exponentiation algorithm we have to perform two multiplications i parallel. This can be achieved by pipelining the circuit and adding an extra A register. It is not necessary to duplicate the B register since the two multiplications always have a common operand. See algorithm 1.

3.2 Generating $a_i B$ and $-qN$

To compute $-qN$ we again use the carry save technique. Observe that all numbers in $[0; 42]$ can be expressed as a sum of three powers of two. Table 2 shows which values, α,

β and γ, are needed to compute $-qN = \alpha N + \beta N + \gamma N$. The values $\alpha N, \beta N$ and γN are generated through a selection network, and added through a row of fulladders as shown in figure 6. Here again the result is represented in two words $-qN_c$ and $-qN_s$.

q	α	β	γ	q	α	β	γ	q	α	β	γ	q	α	β	γ	q	α	β	γ
0	0	0	0	10	-16	4	2	20	-16	-4	0	30	-32	0	2	40	-32	-8	0
1	0	0	-1	11	-16	4	1	21	-16	-4	-1	31	-32	0	1	41	-32	-8	-1
2	0	-4	2	12	-16	4	0	22	-32	8	2	32	-32	0	0	42	-32	-8	-2
3	0	-4	1	13	-16	4	-1	23	-32	8	1	33	-32	0	-1				
4	0	-4	0	14	-16	0	2	24	-32	8	0	34	-32	0	-2				
5	0	-4	-1	15	-16	0	1	25	-32	8	-1	35	-32	-4	1				
6	-8	0	2	16	-16	0	0	26	-32	4	2	36	-32	-4	0				
7	-8	0	1	17	-16	0	-1	27	-32	4	1	37	-32	-4	-1				
8	-8	0	0	18	-16	-4	2	28	-32	4	0	38	-32	-4	-2				
9	-8	0	-1	19	-16	-4	1	29	-32	4	-1	39	-32	-8	1				

Table 2: q expressed as the sum of α, β and γ

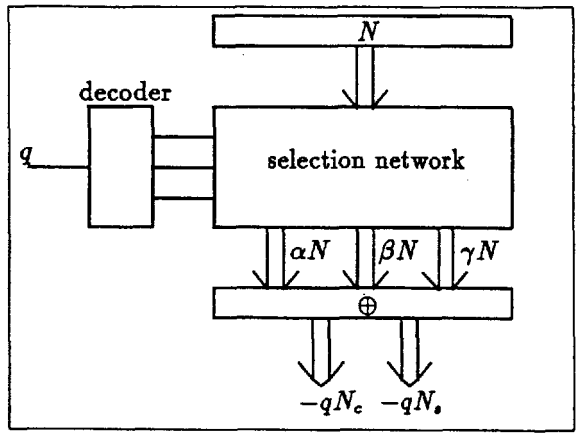

Figure 6: Unit for generating $-qN$

The computation of a_iB is performed following the same principle.

3.3 Quotient estimation

The quotient estimate q is calculated by adding the δ most significant bits of S'_c and S'_s and then multiplying the sum with the ε most significant bits of $\frac{1}{N}$. The quotient can then be found by discarding the $\delta + \varepsilon - (k+2)$ least significant bits of the product, where k is the scan factor.

Earlier we have given an upper bound of $q = 2^{k+1}$, and this restricted $k \leq 4$. In [OS90] we have investigated the interdependency of $q, \delta, \varepsilon, k$ and found an expression that gives a smaller upper bound for q:

$$q_{max} = \frac{2^k(2^{k+3-\delta} + 1 + 2^{1-k})}{1 - 2^{k-\varepsilon}}$$

Table 3 shows that we can achieve an upper bound equal to 42 with $k = 5$ by selecting $\delta = 10$ and $\varepsilon = 11$. This scan factor is optimal for the presented hardware architecture because the maximal value of a_i will exceed 42 if k is greater than 5. Simulations indicate

k	δ	ε	$\lfloor q_{max} \rfloor$
4	8	8	27
4	8	9	26
4	9	9	22
4	9	10	22
5	9	9	51
5	9	10	50
5	10	10	43
5	10	11	42

Table 3: Upper bounds for the quotient estimate

that an even lesser upper bound for q can be found, which means that δ and ε can be reduced, giving a simpler circuit.

4 Performance

The critical path of the multiplication unit has been designed in a 2μ process. Simulations show that a loop in the multiplication unit takes less than 85 ns. In a pipelined version each loop takes two clock cycles, thereby giving a clock period less than 50 ns., corresponding to a clock frequency of more than 20 MHz. The layout shows high regularity and the area is estimated at approximately 100 mm^2.

The efficiency of the algorithms is:

$$\frac{n \cdot 1000\text{bits}}{2 \cdot n(\frac{n+1}{k} + 2)\text{cycles}}$$

For $n = 512$ we achieve an efficiency of 3.8 for $k = 4$ and 4.8 for $k = 5$. The bit rates for a clock frequency of 20 MHz are 78 Kbit/sec and 97 Kbit/sec respectively.

5 Conclusion

We have presented a way to speed up a well known exponentiation algorithm by performing two multiplications in parallel, and we have shown how these multiplications can be performed efficiently using multiple bit scan. Further more we have developed a highly regular hardware architecture, based on the carry save addition technique, implementing the multiplication algorithm with a scan factor of up to 5.

Currently a prototype is being developed at the Computer Science Department, Aarhus University.

References

[Bar86] Paul Barrett. Implementing the Riverst Shamir and Adleman public key encryption system on a standard digital signal processor. In *Advances in Cryptology - CRYPTO '86*, pages 311–323, 1986.

[Bla83] G.R. Blakely. A Computer Algorithm for Calculating the Product AB Modulo M. *IEEE Trans. Computers*, C-32:497–500, 1983.

[Bri89] Ernest F. Brickell. A Survey of Hardware Implementations of RSA. In *CRYPTO '89*, 1989.

[EMI88] THORN EMI. RSA Evaluation Board. Technical Report 10, Thorn EMI Central Reasearch Laboratories, 1988.

[HDVG88] Frank Hoornaert, Marc Decroos, Joos Vandewalle, and René Govaerts. Fast RSA-Hardware: Dream or Reality ? In *Advances in Cryptology - EUROCRYPT '88*, pages 257–264, 1988.

[Knu69] Donald E. Knuth. *The Art of Computer Programming - Seminumerical Algorithms*, volume 2. Addison-Westley, 1969.

[ORSP86] G.A. Orton, M.P. Roy, P.A. Scott, and L.E. Peppard. VLSI implementation of public-key encryption algorithms. In *Advances in Cryptology - CRYPTO '86*, pages 277–301, 1986.

[OS90] Holger Orup and Erik Svendsen. VICTOR. Forbedringer og videreudviklinger af VICTOR - en integreret kreds til understøttelse af RSA-kryptosystemer. Computer Science Department of Aarhus University - Internal report, 1990.

[OSA90] Holger Orup, Erik Svendsen, and Erik Andreasen. VICTOR - Teoretiske og eksperimentelle undersøgelser af algoritmer til understøttelse af RSA-kryptosystemer med henblik på VLSI design. Master's thesis, Computer Science Department of Aarhus University, 1990.