How to Improve an Exponentiation Black-Box

[Published in K. Nyberg, Ed., *Advances in Cryptology – EUROCRYPT '98*, vol. 1403 of *Lecture Notes in Computer Science*, pp. 211–220, Springer-Verlag, 1998.]

Gérard Cohen[1], Antoine Lobstein[1], David Naccache[2], and Gilles Zémor[1]

[1] ENST, Informatique et Réseaux
46 rue Barrault, 75634 Paris, France
{cohen, lobstein}@inf.enst.fr, zemor@res.enst.fr
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
naccache@compuserve.com

Abstract. In this paper we present a method for improving the performance of RSA-type exponentiations. The scheme is based on the observation that replacing the exponent d by $d' = d + k\phi(n)$ has no arithmetic impact but results in significant speed-ups when k is properly chosen. Statistical analysis, verified by extensive simulations, confirms a performance improvement of 9.3% for the square-and-multiply scheme and 4.3% for the signed binary digit algorithm. However, the most attractive feature of our method seems to be the fact that in most cases, *existing* exponentiation black-boxes can be accelerated by simple *external* one-time pre-computations without any internal code or hardware modifications.

1 Introduction

RSA-type cryptosystems use two functions:

$$m \mapsto m^e \bmod n$$
$$m \mapsto m^d \bmod n$$

where $n = pq$ is generally the product of two primes, $ed \equiv 1 \bmod \phi(n)$ and $\phi(n)$ is the Euler totient function. The public exponent can be chosen short (typically $e = 3$) but the secret exponent d must not have any particular structure.

The computation of $m^d \bmod n$ is cumbersome and any of its speed-up tricks is potentially interesting for actual implementations. The simplest and most popular way to compute $m^d \bmod n$ is the square-and-multiply method which consists of repeated squarings and multiplications by m. It can be summarily described by the following algorithm:

```
$x := 1$
for $i := 1$ to $\ell$ do
    $x := x^2 \bmod n$
    if $a_{\ell-i} = 1$ then $x := xm \bmod n$
```

where d is an ℓ-bit integer with binary representation $d = \sum_{i=0}^{\ell-1} a_i 2^i$.

The complexity of this scheme is:

$$c(d) = \ell(d) + \alpha w(d)$$

where $w(d)$ denotes the Hamming weight of the binary vector $[a_{\ell-1}, \cdots, a_1, a_0]$ representing d (the number of a_i's equal to 1) and α represents the cost of a modular multiplication compared to a modular squaring. For large n, using standard techniques it is asymptotically considered [8] that $\alpha \approx 2$. The cost $c(d)$ therefore represents the squaring-equivalents needed to complete the exponentiation. Note that in general, $\phi(n)$ is of the same order of magnitude as n, so that when d ranges over the integers $1, 2, \ldots, \phi(n) - 1$, the average Hamming weight of the binary representation of d is approximately $\frac{1}{2}\log_2 n$; when $\alpha = 2$ the average cost is therefore:

$$\overline{c}(d) \approx 2\log_2 n.$$

For the sake of completeness, let us mention that exponentiations are frequently done separately modulo p and q and re-combined modulo n using the Chinese remainder theorem [11].

There are several strategies and time-memory trade-offs for lowering the complexity of the computation of $m^d \bmod n$ in different *scenarii*: one line of research has been to look for short additions chains [14, 12] which prove to be suited to settings where squarings are not significantly faster than multiplications. Most methods adapted to the situation when squarings are faster than multiplications involve redundant binary representations (RBRs) of the exponent. An RBR of d is a vector $[b_{\ell-1}, \cdots, b_1, b_0]$ where $d = \sum_{i=0}^{\ell-1} b_i 2^i$, and where the b_i's belong to some enlarged set of integers $B \supset \{0, 1\}$. Given an RBR of d, the square-and-multiply algorithm generalises naturally to:

```
pre-compute the set $\{m^b \bmod n, b \in B\}$.
$x := 1$
for $i = 1$ to $\ell$ do
    $x := x^2 \bmod n$
    if $b_{\ell-i} \neq 0$ then $x := xm^{b_{\ell-i}} \bmod n$
```

The time complexity of this algorithm is easily shown to be

$$c_B(d) = \ell(d) + \alpha w_B(d) + p(B) \qquad (1)$$

where $w_B(d)$ is the Hamming weight of the vector $[b_{\ell-1}, \cdots, b_1, b_0]$ and $p(B)$ denotes the number of squaring-equivalents necessary to pre-compute the set $\{m^b \bmod n, b \in B\}$.

Several choices of B have been put forward and extensively analysed. The set $B = \{0, 1, -1\}$ yields the *signed digit binary representation* of d and appears also useful in many (non-cryptographic) arithmetic contexts [2, 13]. The sets $B' = \{0, 1, 2, 3, \cdots, 2^r - 1\}$ and $B = \{-(2^r - 1), \cdots, -2, -1, \} \bigcup B'$ yield essentially the q-ary and signed q-ary representations of d [9]. An improved choice of B

consists of the set $B = \{0, 1, 3, \cdots, 2i+1, \cdots, 2^r - 1\}$ which yields [7]. The set $B = \{0, 1, 3, 7, \cdots, 2^i - 1, \cdots, 2^r - 1\}$ was considered in [6] and the set B obtained after a Lempel-Ziv parsing of the binary representation of d was also considered in the literature [1].

In this paper we decrease the exponentiation cost by replacing d by $d + k\phi(n)$. This approach, suggested in a sentence[1] but never taken-up for study since, will increase the number ℓ of squarings but, for properly chosen k, will diminish the number w of multiplications to do more than compensate. Finding the proper k may require a few thousands of additions but, for RSA-type applications where d is fixed, this needs to be performed only once. In the next sections, we first apply this idea to the square-and-multiply method. We then adapt it to its various improvements involving RBRs and discuss its practical aspects.

2 The Binary Case

From now on we write ϕ for short instead of $\phi(n)$. Suppose that we replace d by $d + k\phi$. The number of squarings increases from $\ell = \ell(d)$ to $\ell(d + k\phi)$ which we can consider approximately equal to $\ell(k\phi) = \ell(k) + \ell(\phi)$. The size of d being most of the time very close to that of ϕ, the number of squarings can be considered to be approximately $(1+t)\ell$ where $t\ell = \ell(k)$. The idea is to compensate the growth in the number of squarings by decreasing the number of multiplications, *i.e.* $w(d + k\phi)$. In theory, an extensive computing effort may be necessary to find the proper k. However this pre-computation needs to be performed only once per d and, as will appear from the equations to come, happens to be moderate for nearly-optimal exponents.

We need to study the minimum of $w(d + k\phi)$ when k ranges over the set of integers of length $t\ell$. Let us set $\ell' = (1+t)\ell$ and $d' = d + k\phi$ of minimum binary weight when k ranges over the integers of length $t\ell$.

Let us make the further reasonable assumption (confirmed by field experiments) that the set of the $2^{t\ell}$ binary $(1+t)\ell$-tuples behaves as a set of vectors chosen randomly and independently among the $2^{\ell'}$ binary vectors of length ℓ'. In this case, the expectation of the number of vectors of weight u in the set is:

$$E_u = \binom{\ell'}{u} \times 2^{t\ell - \ell'}$$

and is greater than 1 as long as

$$\binom{\ell'}{u} > 2^{\ell}. \tag{2}$$

Letting $w' = \inf_{E_u \geq 1} u$, the average cost of a raising to the power $d' = d + k\phi$ is therefore $c' = \ell' + \alpha w'$. Setting $w' = y\ell'$, we get from (2):

$$\ell' H(y) = \ell,$$

[1] "[11]: let us remark that the exponents d_1 and d_2 may be chosen to be greater than $p - 1$ and $q - 1$."

in other words

$$H(y) = \frac{1}{1+t}$$

where $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$ is the binary entropy function [10].

Consequently,

$$c'/\ell = (1+t)\left(1 + \alpha H^{-1}\left(\frac{1}{1+t}\right)\right),$$

the evolution of which as a function of t for $\alpha = 2$ is depicted in figure 1.

Note that we have:

$$c'/\ell = \frac{1}{H(y)}(1+\alpha y)$$

whence

$$\frac{H(y)}{\ell}\frac{\partial}{\partial y}c' = \alpha - (1+\alpha y)\frac{H'(y)}{H(y)},$$

from which we deduce that the minimum of c' is obtained when y satisfies

$$\alpha H(y) - (1+\alpha y)H'(y) = 0$$

which (since $H'(y) = \log_2((1-y)/y)$) boils down to

$$(1-y)^{1+\alpha} - y = 0. \tag{3}$$

Summarising, the minimum of c'/ℓ is obtained when

$$t = \frac{1}{H(\zeta)} - 1$$

where ζ is the root belonging to $[0, 1/2]$ of equation (3), which yields, in the asymptotic case $\alpha = 2$:

$$\zeta = \sqrt[3]{\frac{\sqrt{31}\sqrt{3}}{18} - \frac{1}{2}} - \sqrt[3]{\frac{\sqrt{31}\sqrt{3}}{18} + \frac{1}{2}} + 1.$$

We obtain

$$t \approx 0.109.$$

For this t, the average number of squaring-equivalents diminishes from 2ℓ to 1.813ℓ and represents a non-negligible speed-up of 9.3%, confirmed by extensive simulations.

3 The Signed Digit Binary Case

A particular redundant binary representation is obtained when $B = \{-1, 0, 1\}$. In this case, the square-and-multiply requires the storing of $m^{-1} \bmod n$.

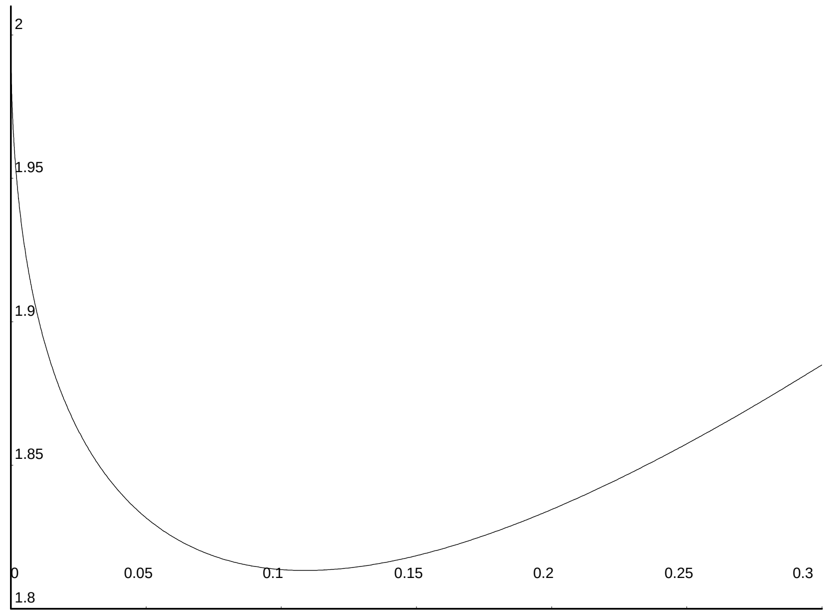

Fig. 1. Evolution of c'/ℓ as a function of t, when k ranges over the integers of size $t\ell$.

If d is an integer, a *signed digit binary representation* of d is of the form

$$d = \sum_i b_i 2^i \tag{4}$$

with $b_i \in B = \{-1, 0, 1\}$. Such a representation is not unique. Any form (4) with a minimal number $w_a(d)$ of nonzero coefficients b_i is called *minimal* and $w_a(d)$ is called the *arithmetic weight* of d. A minimal representation is generally not unique. However, the representation:

$$d = \sum_{i=0}^{\ell-1} b_i 2^i \tag{5}$$

with $b_i \cdot b_{i+1} = 0$ for $i = 0, 1, \ldots, \ell - 2$, (called *nonadjacent form* (NAF) of d) is unique, minimal, exists for all integers, and is easy to compute. If d is ℓ-bit long, then its NAF is at most $(\ell + 1)$-bit long and its average arithmetic weight is $\ell/3$ (see [3,4]), whereas the average Hamming weight of a binary ℓ-tuple is $\ell/2$. The cost (1) of computing $m^d \bmod n$ now becomes

$$c_a(d) = \ell(d) + \alpha w_a(d)$$

plus an asymptotically negligible extra squaring and the amount of work necessary to pre-compute m^{-1}. Consequently, the average cost of the scheme using the signed digit binary representation is essentially $\ell(d)+\alpha\ell(d)/3 \approx (1+\alpha/3)\log_2 n$, instead of $\ell(d)+\alpha\ell(d)/2 \approx (1+\alpha/2)\log_2 n$ for the binary representation (for $\alpha = 2$, we get $\frac{5}{3}\log_2 n$ instead of $2\log_2 n$).

Now suppose that we replace d by $d + k\phi$. We need to study the minimum c'_a of $c_a(d + k\phi)$ when k ranges over the set of integers of length $t\ell$. As before, set $\ell' = (1+t)\ell$ and $d' = d + k\phi$ of minimum arithmetic weight when k ranges over the integers of length $t\ell$.

Let us make again the assumption that the $2^{t\ell}$ vectors representing $d + k\phi$ behave like a set of $2^{t\ell}$ vectors chosen randomly and independently amongst ternary nonadjacent vectors of length $\ell' = (1+t)\ell$.

A random ternary nonadjacent vector of length ℓ' and of Hamming weight u can be looked upon as a string of $\ell' - u$ symbols of the form 0, 10, and -10. Any such vector can therefore be obtained by first choosing a binary vector of length $\ell' - u$ and weight u and then replacing each 1 symbol by either 10 or -10. Their number equals $2^u\binom{\ell'-u}{u}$. The expectation of the number of ternary nonadjacent vectors of weight u in the set of ternary nonadjacent vectors representing $d+k\phi$ is therefore:

$$E_u = \binom{\ell' - u}{u} \times 2^{t\ell+u-\ell'}$$

which is greater than 1 as long as

$$\binom{\ell' - u}{u} > 2^{\ell-u}. \tag{6}$$

As before, set $w' = \inf_{E_u \geq 1} u$. The average cost of a raising to the power $d' = d + k\phi$ is therefore

$$c'_a = \ell' + \alpha w'.$$

Setting $w' = y\ell'$, this time (6) yields:

$$\ell'\left(y + (1-y)H\left(\frac{y}{1-y}\right)\right) = \ell,$$

in other words

$$f(y) = \frac{1}{1+t}$$

where

$$f(y) = y + (1-y)H\left(\frac{y}{1-y}\right).$$

We have therefore

$$c'_a = \ell(1+t)\left(1 + \alpha f^{-1}\left(\frac{1}{1+t}\right)\right).$$

The evolution of c'_a/ℓ as a function of t is represented in figure 2 for $\alpha = 2$. The minimum of c'_a/ℓ is obtained for $t = 0.0497$ and the corresponding average number of squaring-equivalents drops from $1.667 \log_2 n$ to $1.595 \log_2 n$ which represents a 4.3% time improvement. Although this appears small, one should keep in mind that there is *already* a 5/6 performance ratio between the standard and the signed binary exponentiation algorithms.

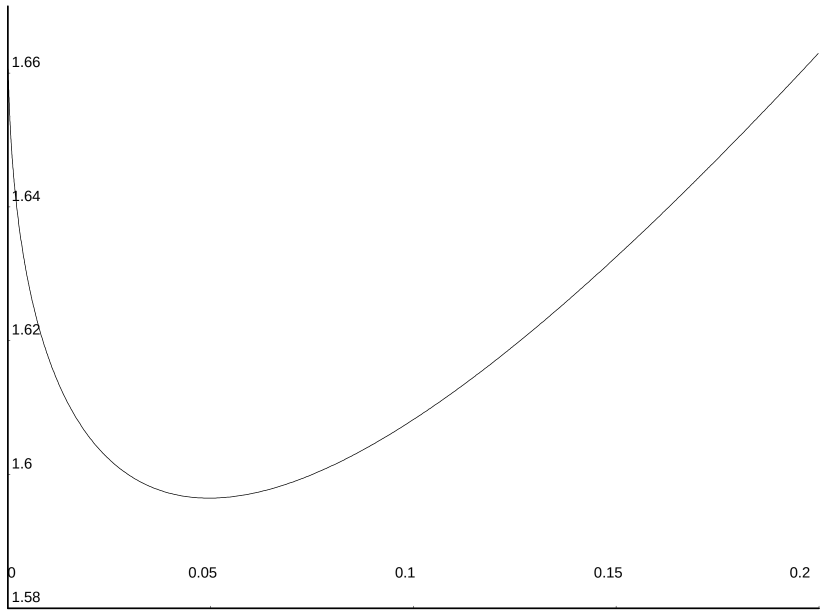

Fig. 2. Evolution of c'_a/ℓ as a function of t, when k ranges over the integers of size $t\ell$.

4 The Odd-Set Case

A potential drawback of the signed digit binary representation is that its pre-computation involves a modular division ($m^{-1} \bmod n$). Alternative algorithms avoid this problem by pre-computing and storing $m^3 \bmod n$ or some other odd powers of m. In other words, the set B is chosen to be $B = \{0, 1, 3\}$. Let us describe the idea by first observing that the square-and-multiply method computes at step i the number $m^{[a_{\ell-1} \dots a_{\ell-i}]}$ where $d = \sum_{i=1}^{\ell} a_{\ell-i} 2^{\ell-i}$ and $[a_{\ell-1} \dots a_{\ell-i}]$ stands for the binary representation of $\sum_{j=1}^{i} a_{\ell-j} 2^{\ell-j}$.

If $m^3 = m^{[11]}$ is pre-computed, then computing $m^{[a_{\ell-1} \dots a_{\ell-i-1} a_{\ell-i-2}]}$ from $m^{[a_{\ell-1} \dots a_{\ell-i}]}$ requires two squarings and a multiplication if $[a_{\ell-i-1} a_{\ell-i-2}]$ equals [11] or [10].

We therefore observe that the number of multiplications necessary in the square-and-multiply method is the number of nonzero symbols obtained when

$[a_{\ell-1}\dots a_0]$ is parsed and represented as a string of characters belonging to the alphabet $0, 10, 11$. In other words the number of multiplications equals the Hamming weight of the ternary vector obtained from $[a_{\ell-1}\dots a_0]$ by the above parsing. We see easily that the analysis of the behaviour of the representation of $d + k\phi$ obtained in this fashion is exactly the same as that of the previous sections.

More generally, the odd-set algorithm [7] uses $B = \{0, 1, 3, 5, \cdots, 2^r - 1\}$, and requires 2^{r-1} pre-computations. Now if $[a]$ is the binary representation of an integer, $[b]$ the binary representation of an integer of length r and $[a][b]$ their concatenation, then it is easy to check that computing $m^{[a][b]}$ from $m^{[a]}$ requires r squarings and one multiplication. Therefore, if we parse the binary representation of an integer as a string of symbols belonging to the alphabet $\mathcal{A}$ made up of 0 and the binary vectors of length r starting with 1, we see that the number of necessary multiplications is exactly the weight of the $|\mathcal{A}|$-vector thus obtained.

Now replace d by $d+k\phi$ for $0 \le k \le 2^{t\ell}$ and choose d' of minimum Hamming weight when represented as a string of elements belonging to $\mathcal{A}$. To evaluate the average weight of d' we proceed as in the previous sections. First evaluate the expectation of the number of $|\mathcal{A}|$-strings of weight u: this is easily seen to be

$$E_u = 2^{t\ell} \times \frac{2^{(r-1)u}\binom{\ell'-(r-1)u}{u}}{2^{\ell'}}.$$

Calculations proceed as before: this time we obtain that the average cost of raising to the power d' equals

$$c'_r = \ell(1+t)\left(1 + \alpha f_r^{-1}\left(\frac{1}{1+t}\right)\right)$$

with

$$f_r(y) = (r-1)y + (1-(r-1)y)H\left(\frac{y}{1-(r-1)y}\right).$$

For $r = 3$ and $\alpha = 2$, the evolution of c'_r/ℓ as a function of t is represented in figure 3. Since the original average weight is $\ell/(r+1)$, the game begins with $(r+3)\ell/(r+1)$ squaring-equivalents for $\alpha = 2$ and becomes 1.5ℓ for $r = 3$; whereas the minimal cost 1.467ℓ results in a 2.2% speed-up[2].

5 Applications and Further Research

In this paper we investigated the impact of replacing an exponent d by a functionally equivalent $d' = d + k\phi(n)$. This surprisingly simple optimisation, to the best of our knowledge never treated in the literature, appears to offer rather significant performance improvements and does not present any real disadvantage (at worst, the exponent size will increase by a few bits). Moreover, this strategy

[2] when r gets bigger, the exponentiation engine's performances improve but the speed-up due to our optimisation strategy decreases.

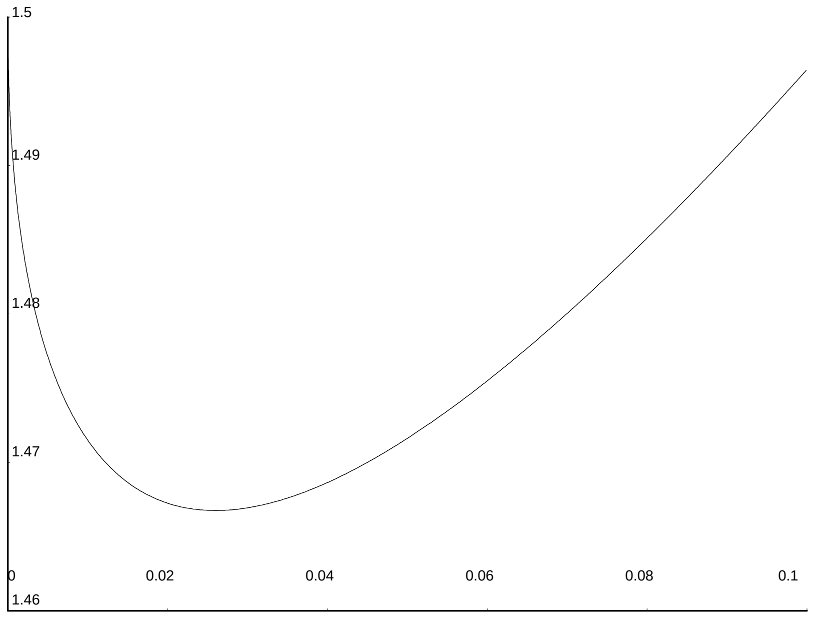

Fig. 3. Evolution of c'_3/ℓ as a function of t, when k ranges over the integers of size $t\ell$.

can be applied to *existing* black-boxes (such as compiled arithmetic libraries or cryptographic co-processors) without any modification. We performed extensive practical tests on three existing platforms: Mathematica's `PowerMod[,,]` function, BSAFE and the Miracl big number library. In each case we did not modify the source code and compared the performances of random exponentiations to those obtained with their optimal equivalents, generated by adding an appropriate multiple of ϕ. Mathematica's `PowerMod[,,]` became 7.1% faster while Miracl and BSAFE's performances improved by 5.4% and 6.9%. Elliptic-curves should feature even better: projective doubling over GF(2^m) requires 5 field squarings and 5 multiplications (4 temporary variables) and projective addition requires 5 squarings and 15 multiplications (9 temporary variables); wherefrom an $\alpha \approx 2.33$.

An interesting open question consists in optimising the time complexity of random exponentiation oracles (black-boxes that compute $m^d \bmod n$ in a time complexity which does not depend on any regular function of d). In this setting, the optimiser only knows the oracle's expectation distribution and is allowed to make a polynomial number of queries in order to find a d' better than d.

References

1. I. Bocharova, B. Kudryashov, *Fast exponentiation in cryptography*, AAECC-11, Lecture Notes in Computer Science 948, Springer Verlag, pp. 146–157, 1995.

2. A. Booth, *A signed binary multiplication technique*, Quarterly Journal of Mechanics and Applied Mathematics vol. 4, pp. 236–240, 1951.
3. A. Chiang, I. Reed, *Arithmetic norms and bounds of the arithmetic AN codes*, IEEE Trans. on Information Theory, vol. IT-16, pp. 470–476, 1970.
4. W. Clark, J. Liang, *On arithmetic weight for a general radix representation of integers*, IEEE Trans. on Information Theory, vol. IT-19, pp. 823–826, 1973.
5. C. Frougny, *Linear numeration systems of order two*, Information and Computation, vol. 77, pp. 233–259, 1988.
6. D. Gollmann, Y. Han, C. Mitchell, *Redundant integer representations and fast exponentiation*, Designs, Codes and Cryptography, vol. 7, pp. 135–151, 1996.
7. L. Hui, K. Lam, *Fast square-and-multiply exponentiation for RSA*, Electronic Letters, vol. 30, pp. 1396–1397, 1994.
8. D. Knuth, *The Art of Computer Programming*, Volume 2: Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1981.
9. Ç. Koç, *High-radix and bit re-coding techniques for modular exponentiation*, Intern. J. Computer Math., vol. 40, pp. 139–156, 1991.
10. F. MacWilliams, N. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, p. 309, 1977.
11. J. Quisquater, C. Couvreur, *Fast decipherment algorithm for RSA public-key cryptosystem*, Electronic Letters, vol. 18, pp. 905–907, 1982.
12. J. Sauerbrey, A. Dietel, *Resource requirements for the application of addition chains in modulo exponentiation*, EUROCRYPT'92, Lecture Notes in Computer Science 658, Springer Verlag, pp. 174–182, 1992.
13. N. Takagi, S. Yajima, *Modular multiplication hardware algorithms with a redundant representation and their application to RSA cryptosystem*, IEEE Trans. on Computers, vol. 41, 1992.
14. Y. Yacobi, *Exponentiating faster with addition chains*, EUROCRYPT'90, Lecture Notes in Computer Science 473, Springer Verlag, pp. 222–229, 1991.