アルゴリズム 32－9
（1 9 9 3. 3. 1 8）

総和計算の二分割による多倍長剰余乗算アルゴリズム

高木 直史
京都大学工学部

高速で必要記憶領域の少ない多倍長剰余乗算アルゴリズムを提案する。アルゴリズムでは、多倍長乗算における総和計算を二分割する。まず、すべての部分積の上位部の総和を求め、その剰余を計算し、次に、この剰余に部分積の下位部の総和を加え、この和の剰余を計算する。剰余の計算には、新しく提案する効率良い計算法を用いる。ｎワードの剰余乗算おいて、中間結果を記憶するためにｎ＋１ワードしか必要としない。また、計算量は、ｎワードの乗算と２ｎワード／ｎワードの除算の合計と同程度である。アルゴリズムは、カードコンピュータ等、主記憶量の少ない計算機上での実現に適しており、これらの計算機の公開鍵暗号によるセキュリティ向上に有効である。

A Multiple-Precision Modular Multiplication Algorithm with a Two-Way Split Summation Process

Naofumi TAKAGI
Department of Information Science, Kyoto University
Kyoto 606-01, Japan
takagi@kuis.kyoto-u.ac.jp

A new algorithm for multiple-precision modular multiplication is proposed. In the algorithm, we split the summation process of the ordinary multiple-precision multiplication into two. We first add up the upper half part of the whole partial products, and then calculate the residue of the sum. Next, we add the sum of the lower half part of the whole partial products to the residue, and then calculate the residue of the total amount. We use a new efficient procedure for residue calculation. For an n-word modular multiplication, the algorithm requires only (n+1)-word memory space for storing the intermediate result and requires about the same number of operations as for an n-word multiplication and a 2n-word by n-word division. The algorithm is fast and efficient for implementation on a small computer such as a card computer, and is useful for adoption of public-key cryptosystem in such computer.

1 Introduction

With the spread use of card computers so-called smart cards which are the basis of prepaid cards, banking cards, credit cards and so on, security on these computers increases its importance. Adopting public key cryptosystems, such as RSA [1] and ElGamal [2], to card computers for the sake of security is attractive. In encryption and decryption of such public key cryptosystems, modular multiplication with a large modulus (longer than 500-bit) is the main operation. Therefore, developing a fast algorithm for multiple-precision modular multiplication which requires small amount of memory space is the key to adopting a public key cryptosystem to card computers which have small amount of main memory.

Various algorithms for multiple-precision modular multiplication have been proposed. Most of them are classified into two methods, i.e., "division-after-multiplication" and "division-during-multiplication". In an n-word modular multiplication by the former method, an ordinary n-word multiplication is carried out first and then a 2n-word by n-word division for residue calculation is performed. In the latter method, each subtraction step for the division for residue calculation is embedded in the repeated multiply-addition [3]. The former requires more amount of memory space than the latter does, because it treats a 2n-word number [4]. On the other hand, in general, the latter requires more addition/subtractions for residue calculation than the former does [5].

In this paper, we propose a new multiple-precision modular multiplication algorithm which is efficient for implementation on a very small computer such as a card computer. It is a completely new algorithm and belongs to neither the division-after-multiplication method nor the division-during-multiplication method. It requires about the same amount of memory space that the division-during-multiplication method does. At the same time, it requires about the same number of addition/subtractions that the division-after-multiplication method does.

In the algorithm, we split the summation process of the ordinary multiple-precision multiplication into two. we first add up the upper half part of the whole partial products, and then calculate the residue of the sum. Next, we add the sum of the lower half part of the whole partial products to the residue, and then calculate the residue of the total amount. We use a new efficient procedure for residue calculation. For an n-word modular multiplication, our algorithm requires only (n+1)-word memory space for storing the intermediate result and requires about the same number of operations as for an n-word multiplication and a 2n-word by n-word division.

In the next section, we give assumptions on our computation model and describe the notations to be used through the paper. We show a new efficient procedure for residue calculation in Section 3, and propose a new multiple-precision modular multiplication algorithm using it in Section 4. Section 5 is a conclusion.

2 Assumptions and Notations

We consider a modular multiplication of $A \times B \bmod N$. We assume that the modulus N is an n-word number and satisfies $r^n/2 \leq N < r^n$ where r is the radix of each word. (For example, when each word consists of 8 bits, $r = 2^8 = 256$.) We assume $r \geq 2n$. (This assumption is very practical. For example, when the modulus is 512-bit and $r = 256$, $n = 64$ and this assumption is satisfied.) The multiplicand A and the multiplier B are also n-word numbers and satisfy $0 \leq A, B < N$.

The i-th word of N where $i = 0, 1, ..., n-1$ is denoted by N_i. Namely, $N = \sum_{i=0}^{n-1} N_i \cdot r^i$. Similarly, $A = \sum_{i=0}^{n-1} A_i \cdot r^i$ and $B = \sum_{i=0}^{n-1} B_i \cdot r^i$.

In the proposed algorithm, we represent intermediate result P which satisfies $-N \cdot r \leq P < N \cdot r$ as an (n+1)-word number with a sign ps. When ps is 0, P is positive and otherwise (i.e., ps is 1), P is negative. P is represented in two's complement form. Namely, $P = -ps \cdot r^{n+1} + \sum_{i=0}^{n} P_i \cdot r^i$.

We assume that our computer has the following operations.

1. Addition/Subtraction

   Adding two unsigned single-word operands with the content of the carry flag as carry-in, and obtaining the unsigned single-word sum and the carry. The carry (overflow) is automatically set to the carry flag. Subtraction is performed by addition with complementation of the subtrahend.

2. Complementation

   Given a single-word operand, and obtaining its one's complement. $\bar{x}$ denotes the one's complement of x. $\bar{x} = r - x - 1$.

3. Multiplication

   Multiplying two unsigned single-word operands, and obtaining the unsigned double-word product. The upper word and the lower word of the product of x and y are denoted by $(x \cdot y)_{high}$ and $(x \cdot y)_{low}$, respectively.

4. Division

   Dividing an unsigned double-word dividend by an unsigned single-word divisor, and obtaining the unsigned single-word quotient and the unsigned single-word remainder. The dividend is assumed to be smaller than r times the divisor. We refer to the quotient and the remainder of x/y as $DIV(x, y)$ and $REM(x, y)$, respectively.

5. Comparison

   Comparing a single-word number with another single-word number.

Note that an n-word addition/subtraction can be performed by n "Addition/Subtraction" operations.
When x and y are single-word numbers, $x\|y$ is the concatenation of x and y, i.e., the double-word number $x \cdot r + y$.

3 A Residue Calculation Procedure

In our modular multiplication algorithm, we use a new efficient procedure, Function $MOD(P, N)$, for residue calculation where N is the modulus satisfying $r^n/2 \leq N < r^n$ and P satisfies $-N \cdot r \leq P < N \cdot r$. P is represented by $n+1$ words with sign. $MOD(P, N)$ satisfies $MOD(P, N) \equiv P \pmod N$ and $-N \leq MOD(P, N) < N$.

The function is as follows.

```
Function $MOD(P, N)$
(Arguments)
  $N \ (= \sum_{i=0}^{n-1} N_i \cdot r^i)$    (an n-word number, $r^n/2 \leq N < r^n$)
  $P \ (= -ps \cdot r^{n+1} + \sum_{i=0}^{n} P_i \cdot r^i)$    (an (n+1)-word number with sign, $-N \cdot r \leq P < N \cdot r$)
(Result)
  $MOD(P, N)$    (an n-word number with sign, $MOD(P, N) \equiv P \pmod N$, $-N \leq MOD(P, N) < N$)
(Procedure)
  if $ps = 0$ (/* $P \geq 0$ */) then do
  begin
    if $P_n = N_{n-1}$ then $q := r - 1$
    else do
    begin
      $\hat{q} := DIV(P_n \| P_{n-1}, N_{n-1})$
      $rem := REM(P_n \| P_{n-1}, N_{n-1})$
```

```
        if $(\hat{q} \cdot N_{n-2})_{high} - rem > \lfloor N_{n-1}/2 \rfloor$ then $q := \hat{q} - 1$ else $q := \hat{q}$
      end
    end
    else (/* $P < 0$ */) do
    begin
      if $\bar{P}_n = N_{n-1}$ then $q := r - 1$
      else do
      begin
        $\hat{q} := DIV(\bar{P}_n \| \bar{P}_{n-1}, N_{n-1})$
        $rem := REM(\bar{P}_n \| \bar{P}_{n-1}, N_{n-1})$
        if $(\hat{q} \cdot N_{n-2})_{high} - rem > \lfloor N_{n-1}/2 \rfloor$ then $q := \hat{q} - 1$ else $q := \hat{q}$
      end
    end
    return $P - (-1)^{ps} \cdot q \cdot N$                                              □
```

The function returns an n-word number with sign. The n-word is the lower n-word of the result of the final addition/subtraction. We let the sign be 0 if the n-th word (the most significant word) of the result of the final addition/subtraction is 0 and be 1 otherwise (i.e., the n-th word is $r - 1$).

In the function, we determine q from only the most significant two words of P, i.e., P_n and P_{n-1}, and the most significant two words of N, i.e., N_{n-1} and N_{n-2}.

The function can be performed by a couple of operations for determining q and an (n+1)-word addition/subtraction. The operations for determining q at most includes two single-word complementations, two single-word comparisons, one double-word by single-word division, one single-word multiplication, and two single-word subtractions. An (n+1)-word addition/subtraction is performed by n+1 single-word addition/subtractions.

Restricting the range of P in $[-N \cdot r, N \cdot r)$ and keeping the residue in the range $[-N, N)$ make the determination of q very simple and reduce the number of required operations.

We can prove the following lemma.

[Lemma 1]

$MOD(P, N) \equiv P \pmod{N}$ and $-N \leq MOD(P, N) < N$ hold.

(Proof)

Since $MOD(P, N) = P - (-1)^{ps} \cdot q \cdot N$ for a certain integer q, $MOD(P, N) \equiv P \pmod{N}$ holds.

To prove $-N \leq MOD(P, N) < N$, we have to consider the following six cases. Hereafter, P' denotes the part of P less than r^{n-1} and N' denotes the part of N less than r^{n-2}. Namely, $P = -ps \cdot r^{n+1} + P_n \cdot r^n + P_{n-1} \cdot r^{n-1} + P'$ and $N = N_{n-1} \cdot r^{n-1} + N_{n-2} \cdot r^{n-2} + N'$.

Case 1: $P \geq 0$

Case 1-1: $P_n = N_{n-1}$

$q = r - 1$

$MOD(P, N) = P - (r - 1) \cdot N$

$$MOD(P, N) \geq N_{n-1} \cdot r^n - (r - 1) \cdot N > r^{n-1} \cdot (N_{n-1} - r + 1) > -N \quad (1)$$

$$MOD(P, N) < N \cdot r - (r - 1) \cdot N = N \quad (2)$$

From (1) and (2), $-N < MOD(P, N) < N$ holds.

Case 1-2: $P_n < N_{n-1}$

Note that $P_n \cdot r + P_{n-1} - \hat{q} \cdot N_{n-1} = rem$ and $0 \leq rem < N_{n-1}$.

Case 1-2a: $(\hat{q}\cdot N_{n-2})_{high} - rem \leq \lfloor N_{n-1}/2 \rfloor$

$q = \hat{q}$

$MOD(P,N) = P - \hat{q}\cdot N = rem \cdot r^{n-1} + P' - \hat{q}\cdot(N_{n-2}\cdot r^{n-2} + N')$

$$MOD(P,N) > (rem - (\hat{q}\cdot N_{n-2})_{high} - 2)\cdot r^{n-1} \geq -(\lfloor N_{n-1}/2 \rfloor + 2)\cdot r^{n-1} > -N \tag{3}$$

$$MOD(P,N) \leq rem \cdot r^{n-1} + P' < N \tag{4}$$

From (3) and (4), $-N < MOD(P,N) < N$ holds.

Case 1-2b: $(\hat{q}\cdot N_{n-2})_{high} - rem > \lfloor N_{n-1}/2 \rfloor$

$q = \hat{q} - 1$

$MOD(P,N) = P - (\hat{q}-1)\cdot N = rem\cdot r^{n-1} + P' - \hat{q}\cdot(N_{n-2}\cdot r^{n-2} + N') + N$

$$MOD(P,N) \geq -\hat{q}\cdot(N_{n-2}\cdot r^{n-2} + N') + N > -(r-1)\cdot r^{n-1} + N > -N \tag{5}$$

$$MOD(P,N) \leq (rem - (\hat{q}\cdot N_{n-2})_{high})\cdot r^{n-1} + P' + N < (-\lfloor N_{n-1}/2 \rfloor + 1)\cdot r^{n-1} + N < N \tag{6}$$

From (5) and (6), $-N < MOD(P,N) < N$ holds.

Case 2: $P < 0$

Case 2-1: $\bar{P}_n = N_{n-1}$

$q = r - 1$

$MOD(P,N) = P + (r-1)\cdot N$

$$MOD(P,N) \geq -N\cdot r + (r-1)\cdot N = -N \tag{7}$$

$$MOD(P,N) < -N_{n-1}\cdot r^n + (r-1)\cdot N < r^{n-1}\cdot(r - 1 - N_{n-1}) < N \tag{8}$$

From (7) and (8), $-N \leq MOD(P,N) < N$ holds.

Case 2-2: $\bar{P}_n < N_{n-1}$

Note that $(r^2 - P_n\cdot r - P_{n-1} - 1) - \hat{q}\cdot N_{n-1} = rem$ and $0 \leq rem < N_{n-1}$.

Case 2-2a: $(\hat{q}\cdot N_{n-2})_{high} - rem \leq \lfloor N_{n-1}/2 \rfloor$

$q = \hat{q}$

$$\begin{aligned} MOD(P,N) &= P + \hat{q}\cdot N \\ &= (-r^{n+1} + P_n\cdot r^n + P_{n-1}\cdot r^{n-1} + P') + \hat{q}\cdot(N_{n-1}\cdot r^{n-1} + N_{n-2}\cdot r^{n-2} + N') \\ &= (-r^2 + P_n\cdot r + P_{n-1} + \hat{q}\cdot N_{n-1})\cdot r^{n-1} + P' + \hat{q}\cdot(N_{n-2}\cdot r^{n-2} + N') \\ &= -(rem+1)\cdot r^{n-1} + P' + \hat{q}\cdot(N_{n-2}\cdot r^{n-2} + N') \end{aligned}$$

$$MOD(P,N) \geq -(rem+1)\cdot r^{n-1} \geq -N_{n-1}\cdot r^{n-1} \geq -N \tag{9}$$

$$\begin{aligned} MOD(P,N) &= (-rem - 1 + (\hat{q}\cdot N_{n-2})_{high})\cdot r^{n-1} + (\hat{q}\cdot N_{n-2})_{low}\cdot r^{n-2} + P' + \hat{q}\cdot N' \\ &< (\lfloor N_{n-1}/2 \rfloor + 2)\cdot r^{n-1} < N \end{aligned} \tag{10}$$

From (9) and (10), $-N \leq MOD(P,N) < N$ holds.

Case 2-2b: $(\hat{q}\cdot N_{n-2})_{high} - rem > \lfloor N_{n-1}/2 \rfloor$

$q = \hat{q} - 1$

$MOD(P,N) = P + (\hat{q}-1)\cdot N = -(rem+1)\cdot r^{n-1} + P' + \hat{q}\cdot(N_{n-2}\cdot r^{n-2} + N') - N$

$$MOD(P,N) \geq (-rem - 1 + (\hat{q}\cdot N_{n-2})_{high})\cdot r^{n-1} - N > (\lfloor N_{n-1}/2 \rfloor - 1)\cdot r^{n-1} - N > -N \tag{11}$$

$$MOD(P,N) \leq -r^{n-1} + P' + \hat{q}\cdot(N_{n-2}\cdot r^{n-2} + N') - N < r^n - N \leq N \tag{12}$$

From (11) and (12), $-N < MOD(P,N) < N$ holds.

Thus, in any case, $-N \leq MOD(P,N) < N$ holds, and the lemma has been proven. □

4 A Multiple-Precision Modular Multiplication Algorithm

In this section, we propose a novel efficient multiple-precision modular multiplication algorithm. In the algorithm, we use the function stated in the previous section iteratively.

We first add up the upper half part of the whole partial products, and then calculate the residue of the sum using Function $MOD(P, N)$ iteratively. Next, we add the sum of the lower half part of the whole partial products to the residue, and then calculate the residue of the total amount using Function $MOD(P, N)$ again.

The algorithm is as follows.

Algorithm [MODMUL]

```
(Inputs)
  Modulus: $N\ (= \sum_{i=0}^{n-1} N_i \cdot r^i)$   (an n-word number, $r^n/2 \le N < r^n$)
  Multiplicand: $A\ (= \sum_{i=0}^{n-1} A_i \cdot r^i)$   (an n-word number, $0 \le A < N$)
  Multiplier: $B\ (= \sum_{i=0}^{n-1} B_i \cdot r^i)$   (an n-word number, $0 \le B < N$)
(Output)
  Product: $P$   (an n-word number, $0 \le P < N$, $P \equiv A \times B \pmod{N}$)
(Algorithm)
  Step 1: $P := \sum_{i+j \ge n-1} A_i \cdot B_j \cdot r^{i+j-n+1}$
  Step 2: $P := MOD(P, N)$
  Step 3: for $k := n-2$ down to 0 do $P := MOD(P \cdot r, N)$
  Step 4: $P := P + \sum_{i+j<n-1} A_i \cdot B_j \cdot r^{i+j}$
  Step 5: $P := MOD(P, N)$
  Step 6: if $P < 0$ then $P := P + N$                                   □
```

Fig. 1 illustrates the outline of Algorithm [MODMUL].

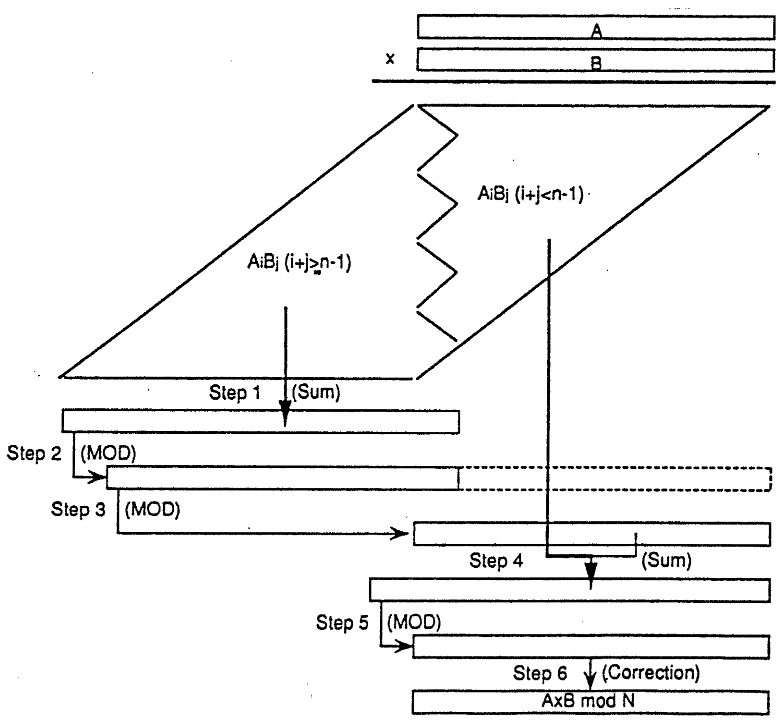

Fig. 1 Outline of Algorithm [MODMUL]

In Step 1, we add up $(A_i \cdot r^i) \cdot (B_j \cdot r^j)$'s such that $i + j \geq n - 1$. Namely, we add up the upper half part of the whole partial products. Since $0 \leq \sum_{i+j \geq n-1} A_i \cdot B_j \cdot r^{i+j-n+1} < N \cdot r^{n-n+1} = N \cdot r$, the obtained P satisfies $0 \leq P < N \cdot r$.

In Step 2, we calculate $P \bmod N$ for P obtained in the previous step. The obtained P satisfies $-N \leq P < N$. (Recall [Lemma 1].)

In Step 3, we calculate $P \cdot r^{n-1} \bmod N$ for P obtained in the previous step, by iterative calculation of $P \cdot r \bmod N$. The obtained P satisfies $-N \leq P < N$. (Again, recall [Lemma 1].)

Through Steps 2 and 3, the residue of the sum of the upper half part is calculated. P obtained in Step 3 satisfies $P \equiv \sum_{i+j \geq n-1} A_i \cdot B_j \cdot r^{i+j} \pmod N$ and $-N \leq P < N$.

In Step 4, we add $(A_i \cdot r^i) \cdot (B_j \cdot r^j)$'s such that $i + j < n - 1$ to the result of the previous step. Namely, we add the sum of the lower half part of the whole partial products to the result of the previous step. The obtained P satisfies $P \equiv \sum A_i \cdot B_j \cdot r^{i+j} \pmod N$. Namely, P is equivalent to the product of A and B with respect to modulo N. Since $0 \leq \sum_{i+j<n-1} A_i \cdot B_j \cdot r^{i+j} < (n-1) \cdot r^n$ and we have assumed $N \geq r^n/2$ and $r \geq 2n$, P satisfies $-N \leq P < N \cdot r$.

In Step 5, we calculate $P \bmod N$ for P obtained in the previous step. The obtained P satisfies $P \equiv \sum A_i \cdot B_j \cdot r^{i+j} \pmod N$ and $-N \leq P < N$. (Again, recall [Lemma 1].)

In Step 6, we add N to the result of the previous step if it is negative. The obtained P satisfies $P \equiv \sum A_i \cdot B_j \cdot r^{i+j} \pmod N$ and $0 \leq P < N$.

Thus, the following theorem holds.

[Theorem 1]

Algorithm [MODMUL] performs n-word modular multiplication. Namely, the obtained P satisfies $P \equiv A \times B \pmod N$ and $0 \leq P < N$. $\square$

Through Algorithm [MODMUL], we require only (n+1)-word memory space for storing the intermediate result. This is about the same as that required by the division-during-multiplication method and is much smaller than that required by the division-after-multiplication method which is about 2n-word.

In Step 1, we require $n(n+1)/2$ single-word multiplications for generating double-word partial products, $A_i \cdot B_j$'s for $i + j \geq n - 1$, and $n^2 + 2n - 2$ single-word additions for adding up them in an adequate order. (Note that we may use only (n+1)-words for the intermediate result.) In Step 4, we require $(n-1)n/2$ single-word multiplications for generating double-word partial products, $A_i \cdot B_j$'s for $i + j < n - 1$, and $n^2 + 2n - 5$ single-word additions for adding them to the result of Step 3 in an adequate order. The number of operations required in Steps 1 and 4 in total is about the same as that required for an n-word multiplication.

In Steps 2, 3 and 5, Function $MOD(P, N)$ is evaluated $n + 1$ times. (Note that we need no computation for obtaining $P \cdot r$.) The number of operations in these steps in total is about the same as that required for a 2n-word by n-word division.

Thus, the number of operations required by Algorithm [MODMUL] is about the same as that required for the division-after-multiplication method, and is much fewer than that required for the division-during-multiplication method.

5 Conclusion

We have proposed a new multiple-precision modular multiplication algorithm. It is efficient for implementation on a very small computer such as a card computer.

In the algorithm, we first add up the upper half part of the whole partial products, and then calculate the residue of the sum. Next, we add the sum of the lower half part of the whole partial products to the residue, and then calculate the residue of the total amount. We use a new efficient procedure for residue calculation. For an n-word modular multiplication, our algorithm requires only (n+1)-word memory space for storing the intermediate result and requires about the same number of operations as that required for an n-word multiplication and a 2n-word by n-word division. The proposed algorithm achieves high-speed computation and small amount of memory requirement at the same time. The procedure for residue calculation used in the algorithm itself is also very interesting.

References

[1] R.L.Rivest, A.Shamir and L.Adleman: "A method for obtaining digital signatures and public-key cryptosystems", Commun. ACM, vol.21, no.2, pp.120-126, Feb. 1978.

[2] T.ElGamal: "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Trans. Information Theory, vol.IT-31, no.4, pp.469-472, July 1985.

[3] E.F.Brickell: "A fast modular multiplication algorithm with application to two key cryptography", D.Chaum et al Eds., 'Advances in Cryptology, Proceedings of CRYPTO 82', pp.51-60, Plenum Press, New York, 1983.

[4] H.Morita: "A fast modular-multiplication algorithm based on a higher radix", Lecture Notes in Computer Science, vol.435, G.Brassard Ed., 'Advances in Cryptology - CRYPTO'89 Proceedings', pp.387-399, Springer-Verlag, 1990.

[5] A.Vandemeulebroecke, E.Vanzieleghem, T.Denayer and P.G.A.Jespers: "A new carry-free division algorithm and its application to a single-chip 1024-b RSA processor", IEEE J. Solid-State Circuits, vol.25, no.3, June 1990.