Radix-4 Modular Multiplication and Exponentiation Algorithms for the RSA Public-Key Cryptosystem

Jin-Hua Hong and Cheng-Wen Wu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013
R.O.C.

Abstract

We propose a radix-4 modular multiplication algorithm based on Montgomery's algorithm, and a radix-4 cellular-array modular multiplier based on Booth's multiplication algorithm. The radix-4 modular multiplier can be used to implement fast RSA cryptosystem. Due to reduced number of iterations and pipelining, our modular multiplier is four times faster than the cellular-array modular multiplier based on the original Montgomery's algorithm. The time to calculate a modular exponentiation is about n^2 clock cycles, where n is the word length, and the clock cycle is roughly equal to the delay time of a full adder. The utilization of the multiplier is 100% by interleaving consecutive exponentiations. Locality, regularity, and modularity make the proposed architecture suitable for VLSI implementation.

Keywords: cellular array, Montgomery algorithm, modular multiplication, high radix modular multiplier, public-key cryptography, RSA.

1 Introduction

With the increasing popularity of electronic communication, data security is becoming more and more important. In 1976, Diffie and Hellman invented a new concept of public-key cryptography [1]. Later, in 1978, Rivest, Shamir, and Adleman proposed the RSA public-key cryptosystem which is relatively secure and easy to implement [2]. In the RSA cryptosystem, both the encryption and decryption are modular exponentiation, which can be done by a sequence of modular multiplications. Therefore, fast modular multiplication becomes the key to real-time encryption and decryption in such a scheme.

Various algorithms for modular multiplication have been proposed in the past, and some of them have been realized [3–6]. One of the most attractive modular multiplication algorithms was proposed by Montgomery [7]. Montgomery's algorithm needs n iterations in each modular multiplication and two additions per iteration, where n is the word length. Cellular arrays based on Montgomery's algorithm can be found in [8–10].

A modified Montgomery's algorithm was first reported in [11], where the multiplication and modular reduction steps in Montgomery's algorithm are separated such that only one addition is required in each iteration. However, the number of iterations in the modified algorithm is two times that of Montgomery's, hence the overall computation time is not reduced. In [12, 13], the algorithm was further modified to reduce the number of iterations, doubling the speed of modular multiplication.

In this paper, we reduce the number of partial products by using radix-4 Booth's algorithm, and propose a radix-4 modular exponentiation algorithm. The proposed algorithm can be implemented using a linear cellular array. For a word length of n, the array has about n cells, and each cell contains two full adders and some logic gates. The two full adders in the cell are pipelined. The number of iterations in the multiplication process is $\lceil \frac{n+3}{2} \rceil$ as compared with $2n$ in the original multiplier. Therefore, the speed of the proposed modular multiplier is quadrupled. The time to calculate a modular exponentiation is $n^2\tau$, where τ is the delay time of a full adder.

To improve the utilization for radix-r modular multiplication without interleaving, we can merge $\log r$ cells into one processing element (PE), resulting in a digit-level array.

2 Modular Multiplication Algorithm

In RSA, to encrypt a message using the encryption key (E, N), we first partition the message (a string of bits) into a sequence of blocks and consider each block M as an integer between 0 and $N-1$. Then, we encrypt the message by raising each block to the Eth power modulo N, i.e., $C = M^E \pmod N$, for each message block M. Similarly, to decrypt the ciphertext using the decryption key (D, N), we raise each ciphertext block to the power D modulo N, i.e., $M = C^E \pmod N$, for each ciphertext block C.

Exponentiation is performed by repeated (iterated) squaring and multiplication operations. Let the binary representation of the exponent E be $e_{n-1}e_{n-2}\cdots e_1e_0$, then $M^E = M^{2^{n-1}\cdot e_{n-1}}\cdots M^{2^2\cdot e_2}\cdot M^{2\cdot e_1}\cdot M^{e_0}$. A simple way to perform modular exponentiation is to repeat the modular squaring (M_i^2) and modular multiplication $(M_i \times P_i)$ operations from the least-significant bit (LSB) of E. This is called the L-algorithm. In the L-algorithm, n iterations are needed and each iteration needs two modular multiplications. However, the modular multiplications M_i^2 and $M_i \times P_i$ can be done in parallel. To reduce the time complexity, we have developed a radix-4 modular multiplication algorithm and designed a radix-4 modular multiplier based on Montgomery's modular arithmetic.

2.1 Review of Montgomery's Algorithm

Suppose $N = (n_{n-1},\ldots,n_1,n_0)$ is an n-bit odd integer. Let $A = (a_{n-1},\ldots,a_1,a_0)$ and $B = (b_{n-1},\ldots,b_1,b_0)$ be two n-bit integers, where $0 \le A,B < N$. Montgomery's modular multiplication is shown below, which generates a series of numbers, $S[0], S[1], \ldots, S[n]$ as outputs.

```
MG(A,B,N)
{
    S[0] = 0;
    for(i = 0; i < n; i++) {
        q_i = (S[i] + a_i B) (mod 2);
        S[i+1] = (S[i] + a_i B + q_i N)/2;
    }
    return S[n];
}
```

In each iteration of the above procedure we need to accumulate three n-bit integers and divide the result by 2. This can be done by using two n-bit adders and right shifting by one bit the adder output. Therefore, each cell is composed of two full adders. Futhermore, by induction, the value of $S[n]$ falls in the range $(0, 2N)$ instead of the initial range $(0, N)$. Therefore, post adjustment is required before the next modular multiplication is performed.

2.2 Radix-4 Modular Multiplication

We present the radix-4 Montgomery algorithm. By using radix-4 numbers for multiplication and modular reduction, the number of iterations can be reduced by half. The number is n in Montgomery's algorithm, which is equivalent to the number of partial products. To reduce the number of iterations (and the number of partial products to be accumulated) we propose a radix-4 Montgomery algorithm which requires only $\lceil \frac{n+3}{2} \rceil$ iterations. The radix-4 Montgomery algorithm can be implemented by Booth's multiplier.

Let A be $(n+3)$-bit and B be $(n+1)$-bit 2's-complement numbers and $N = (n_{n-1},\ldots,n_1,n_0)$ be an n-bit odd integer, where $-N \le A,B < N$. Also, let $PP_i = (pp_{i(n+1)},\ldots,pp_{i1},pp_{i0})$ represent the radix-4 Booth partial product of iteration i. Since the radix-4 Booth recoding is considered, we have $PP_{\lceil \frac{n+1}{2} \rceil} = 0$ and $-2N \le PP_i < 2N$, where $0 \le i < \lceil \frac{n+1}{2} \rceil$. Suppose $T_i = S[i] + PP_i$, then the two LSBs of T_i (i.e., t_{i1} and t_{i0}) can be used to determine the modular reduction value (i.e., N_i, where $N_i = 2N$, $\pm N$, or 0). The proposed radix-4 Montgomery algorithm is shown below.

```
R4MG(A,B,N)
{
    S[0] = 0;
    for (i = 0; i < ⌈(n+3)/2⌉; i++) {
        (t_i1, t_i0) = (S[i] + PP_i) (mod 4);
        if (t_i0 = 0) {
            if (t_i1 = 0)
                S[i+1] = (S[i] + PP_i)/4;
            else
                S[i+1] = (S[i] + PP_i + 2N)/4;
        }
        else {
            if (t_i1 = n_1)
                S[i+1] = (S[i] + PP_i - N)/4;
            else
                S[i+1] = (S[i] + PP_i + N)/4;
        }
    }
    return S[⌈(n+3)/2⌉];
}
```

Procedure R4MG() has about $\frac{n}{2}$ iterations, while previous algorithms require n or $2n$ iterations [7, 11]. In each iteration, two additions are required, which is the same as MG(). Therefore, it is faster than Procedures MG(). By induction, the value of $S[\lceil \frac{n+3}{2} \rceil]$ falls in the range $(-N, N)$, which is the same as the initial range. Therefore, no post adjustment is required during the entire exponentiation process.

2.3 Radix-4 Modular Exponentiation

Our modular exponentiation algorithm is based on the L-algorithm. Let $R = 4^{\lceil \frac{n+3}{2} \rceil}$ and $C = R^2 \pmod N$, then the proposed exponentiation procedure is as shown below, where the final value P_n is equal to $M^E \pmod N$. Also, as discussed above, the range for the intermediate numbers (M_i's and P_i's) will not grow since we use R4MG().

```
R4ME(M, E, N)
{
    M_0 = R4MG(M, C, N);
    P_0 = 1;
    for (i = 0; i < n; i++) {
        M_{i+1} = R4MG(M_i, M_i, N);
        if (e_j = 1)
            P_{i+1} = R4MG(M_i, P_i, N);
        else
            P_{i+1} = P_i;
    }
    if (P_n < 0)
        P_n = P_n + N;
    return P_n;
}
```

Table 1: Intermediate values of Procedure R4ME().

i	0	1	2	3	$\cdots$	n
M_i	$M \cdot R$	$M^2 \cdot R$	$M^4 \cdot R$	$M^8 \cdot R$	$\cdots$	–
P_i	1	M^{e_0}	$M^{2e_1+e_0}$	$M^{4e_2+2e_1+e_0}$	$\cdots$	M^E

Note that we keep intermediate results M_i and P_i in the range $(-N, N)$ during the entire exponentiation process and convert only the final result P_n to the range $(0, N)$ by adding N to P_n if $P_n < 0$. The intermediate values are shown in Table 1. We can perform the exponentiation for one message block concurrently when we read the next message block and write the previously processed cipher block (with post adjustment).

3 Modular Multipliers

3.1 Montgomery's Modular Multiplier

We implement Montgomery's algorithm using a cellular array circuit. The dependence graph (DG) and signal flow graph (SFG) [14] of the modular multiplication algorithm are shown in Fig. 1. In the figure, the projection direction vector $\vec{d} = (0, 1)$, and the schedule vector $\vec{s} = (1, 2)^T$. The utilization ratio of the cells is 50%. The resulting modular multiplier is shown in Fig. 2, where the signal q_i is generated by XORing the LSB of $S[i]$ and $a_i B$. We use two n-bit adders to add $S[i]$, $a_i B$, and $q_i N$. The pipeline technique is applied to the design to reduce the clock period. We divide a cell into three subcells which belong to different pipeline stages. As shown in Fig. 2, the D flip-flops (FFs) are inserted where the dashed lines cross the signals. The clock period is about the delay time of a full adder.

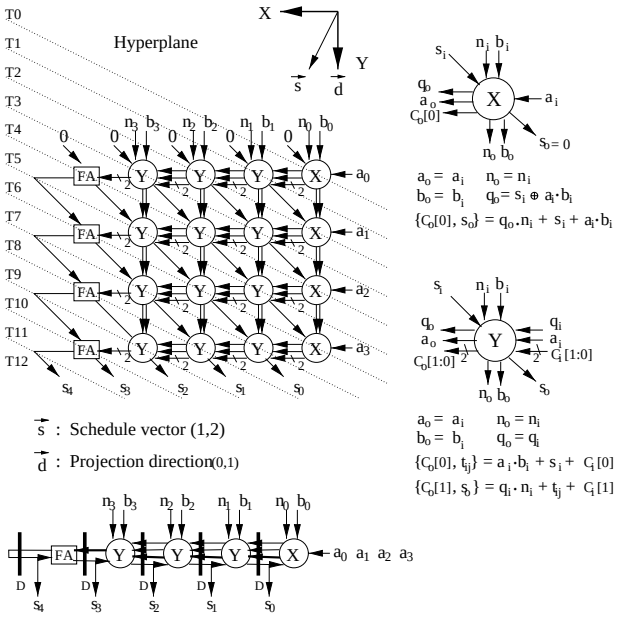

Figure 1: The DG and SFG of a 4-bit modular multiplier.

3.2 Radix-4 Modular Multiplier

The recoding rules for the radix-4 Booth algorithm are shown in Table 2 [15]. The signals Code[2:0] are used to select $\pm B$, $\pm 2B$, or 0 as the partial product.

The DG and SFG of the radix-4 modular multiplication algorithm are shown in Fig. 3. We let the projection direction vector $\vec{d} = (0, 1)$ and the schedule vector $\vec{s} = (1, 3)^T$. Note that the utilization ratio of the cells is 33%. The

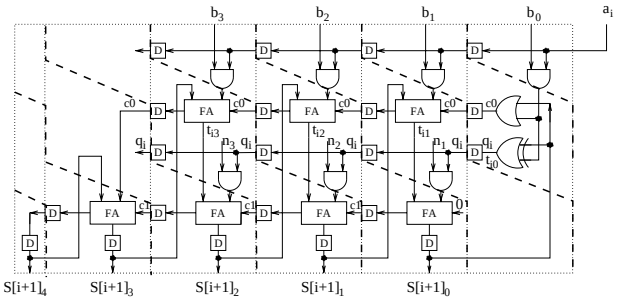

Figure 2: A 4-bit Montgomery modular multiplier.

Table 2: Radix-4 Booth's recoding rules.

a_i	a_{i-1}	a_{i-2}	Booth code	Action	Code[2:0]
0	0	0	0 0	$+0$	000
0	1	0	0 1	$+B$	001
1	0	0	$\overline{1}$ 0	$-2B$	110
1	1	0	0 $\overline{1}$	$-B$	101
0	0	1	0 1	$+B$	001
0	1	1	1 0	$+2B$	010
1	0	1	0 $\overline{1}$	$-B$	101
1	1	1	0 0	$+0$	000

utilization can be improved by interleaving as shown in Fig. 4, where inputs (a,b) are interleaved with the inputs (a',b'). Interleaving two modular multiplications increases the utilization to 67%. As described in R4MG(), we must determine the modular reduction value N_i, where $N_i = (n_{i(n+1)}, \ldots, n_{i1}, n_{i0})$. Three control signals q[2:0] are used to select $2N$, $\pm N$, or 0 as the reduction value, where $q[0] = t_{i0}$, $q[1] = t_{i1}$, and $q[2] = \overline{t_{i1} \bigoplus n_{i1}}$. Note that subtracting N is easier than adding $3N$, so we subtract N when $t_{i0} = 1$ and $t_{i1} = n_1$. Therefore, a $2's$-complement modular multiplier is required. In $2's$-complement multiplication, sign extension is needed when we accumulate the partial products. This can be done by using signals $ctr1$ and $ctr2$ and the pp_{i4} XNOR gate as shown in Fig. 4.

As shown in Fig. 5, we need to add at most three D FFs in each cell. However, the utilization ratio is reduced to 25% due to pipelining. Fortunately, in RSA, a message to be encrypted is divided into a sequence of blocks, and each block is raised to the Eth power modulo N independently, so the modular exponentiations of the successive message blocks can be done in parallel by interleaving. Furthermore, in a modular exponentiation, the computation of M_i^2 and $M_i \times P_i$ can also be interleaved. Our design thus can execute four modular multiplications at the same time by this *double interleaving*. The utilization is then increased to 100%. Since the number of partial products

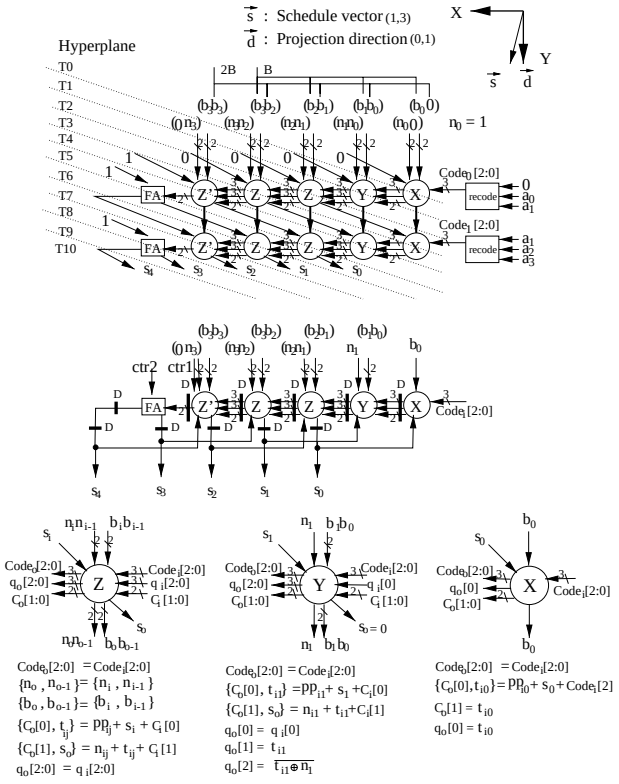

Figure 3: The DG and SFG of a radix-4 modular multiplier.

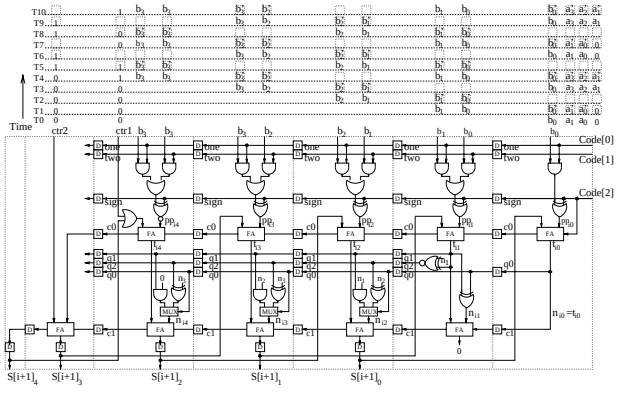

Figure 4: The radix-4 modular multiplier (n=3).

is reduced in half, the number of iterations is so reduced. Compared with [12, 13], we have two times the speed and 1.5 times the hardware cost. The extra 50% hardware cost is due to the interleaving control circuit and the pipeline registers.

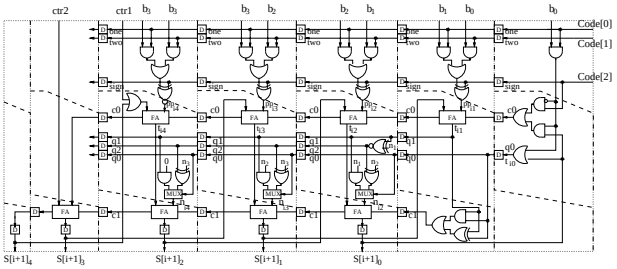

Figure 5: The pipeline radix-4 modular multiplier.

The time to compute four interleaved modular multiplications is about $2n$ clock cycles, and the time to compute a modular exponentiation is about n^2 clock cycles. Table 3 compares the hardware and time complexity of several linear-array RSA systems. In the table, α and τ are roughly the area and delay of a full adder cell, respectively. We normalize the clock period to the delay time τ. From the table, our architecture has the lowest computation time as compared with other works. When the size of the message block and encryption key are 512 bits, the encryption throughput rate is about 300K bps using a 150MHz clock.

Table 3: Comparison of linear-array RSA systems.

Approach	Time	Area
[8]	$4n^2\tau$	$4n\alpha$
[9]	$4n^2\tau$	$2n\alpha$
[11]	$4n^2\tau$	$2n\alpha$
[12]	$2n^2\tau$	$2n\alpha$
[13]	$2n^2\tau$	$2n\alpha$
Ours	$n^2\tau$	$3n\alpha$

4 Digit-Level Modular Multiplier

For a radix-r system, the number of iterations is $\lceil \frac{n+(\log r+1)}{\log r} \rceil$. However, the utilization is reduced to $\frac{1}{\log r+1}$. To increase the utilization of a radix-r modular multiplier without interleaving, we can merge $\log r$ cells into one processing element (PE) and form a digit-level array multiplier. For example, in Fig. 6 two cells are merged into one PE, so the utilization is increased from $1/3$ to $1/2$. In a radix-4 digit-level cellular array, each PE contains two 2-bit adders. The critical path is equal to the delay of three full adders. Therefore, the clock period is roughly 3τ.

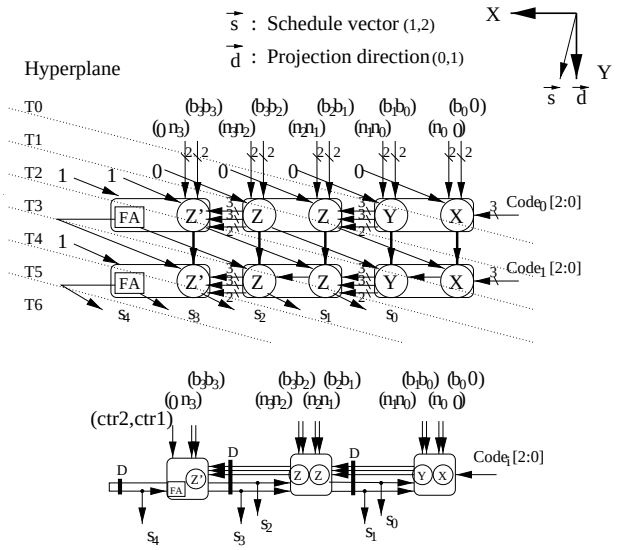

Figure 6: The DG and SFG of a radix-4 digit-level multiplier.

Though the number of iterations is reduced, the hardware complexity and the clock period increase. As Fig. 7 shows, each PE of a radix-16 multiplier contains an N-Selector circuit, a B-Selector circuit, and two 4-bit adders. For n-bit modular multiplication, the number of iterations is $\lceil \frac{n+5}{4} \rceil$, but the clock period is roughly 5τ (i.e., $\log r+1$). Because the utilization is 50%, we need roughly $\frac{n}{2}$ clock cycles to compute two interleaved modular multiplications. We can reduce the clock period by using a fast 4-bit adder (such as the carry look-ahead adder), but this is basically a tradeoff between computation time and silicon area. Another possibility is that the signed digit number system can be used to avoid carry propagation in the PE and hence increase the speed of the high-radix modular multiplier.

5 Conclusions

We have presented radix-4 modular multiplication and modular exponentiation algorithms. In the algorithms, we are able to keep the intermediate results in the range $(-N, N)$, so no post adjustment is required for each multiplication during the modular exponentiation process. A cellular-array modular multiplier based on the algorithm and Booth's multiplication has been presented, which is

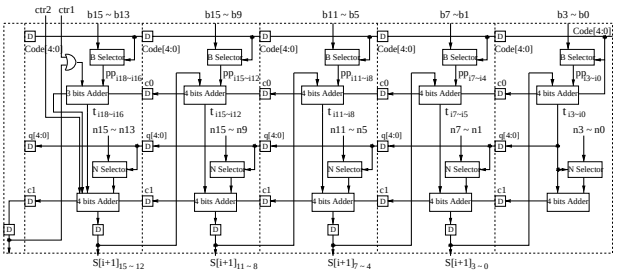

Figure 7: A radix-16 digit-level modular multiplier.

efficient especially for RSA cryptosystem where modular multiplications are performed iteratively. Compared with previous designs, the number of clock cycles of our pipelined radix-4 modular multiplier for a modular exponentiation is about $n^2\tau$, where τ is roughly the delay of a full-adder. The proposed multiplier is four times faster than those based on the original Montgomery algorithm, and is two times faster than those reported in [12, 13]. A high-radix digit-level modular multiplier was also discussed. Extending the design for a larger moduli is straightforward.

References

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Information Theory*, vol. 22, pp. 644–654, Nov. 1976.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

[3] Ç. K. Koç and C. Y. Hung, "Bit-level systolic arrays for modular multiplication," *J. VLSI Signal Processing*, vol. 3, pp. 215–223, 1991.

[4] Ç. K. Koç, "RSA hardware implementation," technical report, RSA Laboratories, RSA Data Security, Inc., Redwood City, CA, 1995.

[5] N. Takagi and S. Yajima, "Modular multiplication hardware algorithms with a redundant representation and their application to rsa cryptosystem," *IEEE Trans. Computers*, vol. 40, pp. 887–891, July 1992.

[6] N. Takagi, "A radix-4 modular multiplication hardware algorithm for modular exponentiation," *IEEE Trans. Computers*, vol. 41, pp. 949–956, Aug. 1992.

[7] P. L. Montgomery, "Modular multiplication without trial division," *Math. Computation*, vol. 44, pp. 519–521, 1985.

[8] P. Kornerup, "A systolic, linear-array multiplier for a class of right-shift algorithms," *IEEE Trans. Computers*, vol. 43, pp. 892–898, Aug. 1994.

[9] C. D. Walter, "Systolic modular multiplication," *IEEE Trans. Computers*, vol. 42, pp. 376–378, Mar. 1993.

[10] M. Shand and J. Vuillemin, "Fast implementation of RSA cryptography," in *Proc. 11th IEEE Symp. Computer Arithmetic*, (Windsor, Ontario), pp. 252–259, June 1993.

[11] P.-S. Chen, S.-A. Hwang, and C.-W. Wu, "A systolic RSA public key cryptosystem," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, (Atlanta), pp. 408–411, May 1996.

[12] C.-C. Yang, T.-S. Chang, and C.-W. Jen, "A new RSA cryptosystem hardware design based on Montgomery's algorithm," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, pp. 908–913, July 1998.

[13] C.-Y. Su, S.-A. Hwang, P.-S. Chen, and C.-W. Wu, "An improved Montgomery algorithm for high-speed RSA public-key cryptosystem," *IEEE Trans. VLSI Systems*, vol. 7, pp. 280–284, June 1999.

[14] S.-Y. Kung, *VLSI Array Processors*. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1988.

[15] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, New Jersey, 07632: Prentice-Hall Inc., 1993.