SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

June 2015, Vol. 58 062401:1–062401:16
doi: 10.1007/s11432-014-5215-4

Fast RSA decryption through high-radix scalable Montgomery modular multipliers

WU Tao[1,3]*, LI ShuGuo[2]* & LIU LiTian[2]*

[1]*Department of Microelectronics and Nanoelectronics, Tsinghua University, Beijing* 100084*, China;*
[2]*Institute of Microelectronics, Tsinghua University, Beijing* 100084*, China;*
[3]*Shanghai Fudan Microelectronics Group Company, Shanghai* 200433*, China*

Received July 23, 2014; accepted September 3, 2014; published online March 17, 2015

Abstract This paper improves the quotient-pipelined high radix scalable Montgomery modular multiplier by processing w-bit and k-bit words in carry save form instead of some $(w + k)$-bit length operands. It directly reduces both the critical path and the area overhead of the original processing elements. Then based on this improved high-radix scalable Montgomery modular multiplier, we propose an efficient hardware architecture for RSA decryption with Chinese Remainder Theorem. With simple configuration logics, the hardware unit works in three modes: (1) scalable modular reduction for precomputation, (2) scalable Montgomery modular multiplication for modular exponentiation, where an approximation method is developed to reduce the expanded result below the modulus, and (3) scalable multiplication for post-processing. Hardware implementation shows that the proposed architecture is optimal with reference to the literature in terms of speed, area, and frequency. A 4096-bit RSA decryption in XC2V6000-6 FPGA can be completed in 11.05 ms with 14041 slices/17409 LUTs, 128 16 × 16 multipliers, and 70 kbits of block RAMs. Finally, by the use of Montogmery powering ladder the modular exponentiation unit based on the improved high radix scalable Montgomery modular multiplier can be built resistant to fault and simple power attacks. A 1024-bit modular exponentiation unit with such resistances costs about 255K NAND2 gates in .18 μm CMOS process, and one full modular exponentiation takes about 1.44 ms at 250 MHz.

Keywords RSA, high radix, scalable, Montgomery modular multiplication, CRT

Citation WU T, Li S G, Liu L T. Fast RSA decryption through high-radix scalable Montgomery modular multipliers. Sci China Inf Sci, 2015, 58: 062401(16), doi: 10.1007/s11432-014-5215-4

1 Introduction

Public-key cryptography (PKC) can be used to exchange the secret keys for symmetric cryptography and data signatures for e-commerce, both of which are quite important in network security. The usual PKCs include RSA cryptography [1], elliptic curve cryptography [2], and the scheme based on coding theory [3], among which RSA is the most widely used.

RSA computation is mainly composed of exponentiations or multiplications in a quite large prime field. The encryption and decryption of RSA means applying RSA in the background of symmetric cryptography, where a small key can be chosen for encryption and a much larger key is used for decryption. This work targets the acceleration of RSA decryption, or any RSA computation with a large key.

* Corresponding author (email: twu03ster@gmail.com, lisg@tsinghua.edu.cn, liulitian@tsinghua.edu.cn)

In order to obtain high performance for multiprecision modular multiplications with limited hardware resources, a tradeoff between area overhead and time complexity can be reached by scalable architecture [4,5]. To continuously generate results with long-precision the processing elements (PEs) within a scalable Montgomery modular multiplier can be reused or replicated, independent of the data path precision for which the unit was originally designed [4]. Unlike the interleaved Montgomery algorithm [6,7], the parallel words are allocated in a pipeline. The low-latency scalable architecture is efficient with limited hardware and fast if sufficient resources are available [8–10]. Meanwhile, high-radix scalable architecture cuts down the number of clock cycles for one scalable Montgomery modular multiplications, although it has a longer critical path and latency between PEs [11, 12]. It was also noticed that if the scalable hardware units did not work by the largest round of the algorithm, then the low latency between PEs will not play a significant role in the whole multiprecision modular multiplication.

Meanwhile, the RSA implementation can be simplified by the Chinese Remainder Theorem (CRT) [13–15]. In fact, RSA by CRT (CRT-RSA) cuts down about 3/4 computational complexity with formal RSA decryption. Nevertheless, CRT-RSA is then vulnerable to the fault attacks and simple power analysis (FA-SPA), which can be avoided by Montgomery powering ladder [16–18]. and parallel hardware units for hardware implementation. If the cryptosystem works in a safe background, then the power analysis attacks will not be a problem.

In this work, the quotient pipelined high-radix scalable Montgomery modular multiplier (HSCM) in [12] is further optimized by us. First, the carry save addition is moved from the critical path, which reduces the area overhead at the same time; second, the HSCM is reconfigured so as to perform scalable multiplications and scalable modular reductions. These optimizations result in an efficient and relatively compact hardware architecture for CRT-RSA. This FA-SPA-Resistant characteristic can also be implemented by HSCM for modular exponentiation.

The rest of this paper is organized as follows. Section 2 briefly reviews the quotient-pipelined Montgomery modular multiplication. Then in Section 3 the optimized HSCM is proposed for modular exponentiation, with a reconfiguration for scalable multiplications and modular reductions. And in Section 4 the modular reduction after modular exponentiation is presented. Section 5 then gives hardware implementation results of our proposals. The last section concludes this paper.

2 Scalable Montgomery modular multiplication with quotient pipeline

Montgomery algorithm [6] is efficient for modular multiplications, since it replaces the original modular reductions over M by that over 2^n. Suppose A, B, M, R, n are all integers with $R = 2^n$, $0 \leqslant A < R$, $0 \leqslant B < R$, $R/2 < M < R$, $A \cdot B < R \cdot M$ and $\mathrm{GCD}(M, 2) = 1$. Precompute $\mu = (-M^{-1}) \mod R$, then Montgomery shows that [6]

$$S = (A \cdot B + (A \cdot B \cdot \mu \mod R) \cdot M)/R,$$

with $S \equiv A \cdot B \cdot R^{-1}(\mod M) \in [0, 2M)$.

Usually, Montgomery modular multiplication is interleaved to reduce area overhead [7]. By scaling one operand A as $2^k \cdot A$ and precomputing $N = (\mu \cdot -M^{-1} \mod (2^{k(d+1)}) + 1)/2^{k(d+1)}$, the quotient pipelined Montgomery modular multiplication algorithm [19] is further improved in [7], as is shown in Algorithm 1.

Apparently, the determination of the quotient digit q_i gets simplified, while another iteration is required to offset the scaling $2^k \cdot A$.

The basic architecture of scalable Montgomery modular multiplier [4] is shown in Figure 1. The multiplier digit b_i is directly loaded in PEs from the shift-register, the quotient digit q_i is generated from the previous PE, and the multiplicand word A_j and N_j are transported through the PEs. It can be noticed that the quotient digit q_i can be stored in a small register rather than their entry into the first-in-first-out registers (FIFOs). The redundant representation $\{S'_j, S''_j\}$ denotes temporary results.

Algorithm 1 has been applied in a scalable architecture in [12] with $d = 1$ in Algorithm 2, where $(S_i)_{j,u}$ denotes the i-th loop, the j-th word, the u-th bit of S.

Algorithm 1 Improved quotient pipelined Montgomery modular multiplication [7]

Require: Integers $A, B, M, \mu, \tilde{M}, N, R, n, k, m, d$, with $n = k \cdot m$, $R = 2^n = 2^{k \cdot m}$, $\mu = -M^{-1} \mod (2^{k(d+1)})$, $\tilde{M} = \mu \cdot M$, $N = (\tilde{M} + 1)/2^{k(d+1)}$. In addition, $\mathrm{GCD}(M, 2) = 1$, $0 \leqslant A < 2\tilde{M}$, $0 \leqslant B < 2\tilde{M}$, $B = \sum_{i=0}^{m+d} b_i \cdot 2^{ki}$, with $b_i = 0$ for $i \geqslant m$.

Ensure: $S = A \cdot B \cdot R^{-1} \mod M$, $0 \leqslant S < 2\tilde{M}$.

1: $S_0 = 0$, $q_{-d} = 0$, $q_{-d+1} = 0$, …, $q_{-1} = 0$;
2: **for** $i = 0$ to $m + d$ **do**
3: $\quad q_i = S_i \mod 2^k$;
4: $\quad S_{i+1} = \lfloor S_i / 2^k \rfloor + q_{i-d} \cdot N + b_i A$;
5: **end for**
6: $S = 2^{kd} S_{m+d+1} + \sum_{i=0}^{d-1} q_{m+i+1} 2^{ki}$;
7: **return** S.

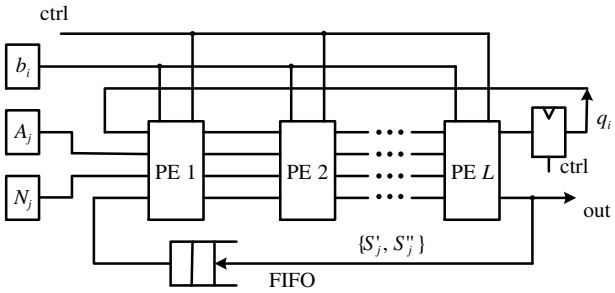

Figure 1 Scalable architecture of a Montgomery modular multiplier.

Algorithm 2 HSCM in [12]

Require: Integers A, B, M are between $[0, 2^n)$, with $2^{n-1} < M < 2^n$, $\mathrm{GCD}(M, 2) = 1$. The multiplicand A is divided into e words, with $A = (A_{e-1} \cdots A_1 A_0)_{2^w}$, $e = \lceil n/w \rceil + 1$. The multiplier B is divided into f digits, with $B = (b_f \cdots b_1 b_0)_{2^k}$, $f = \lceil n/k \rceil + 2$, $b_f = 0$. Integer M' satisfies $(-MM') \mod 2^{2k} = 1$. Set $\tilde{M} = M(M' \mod 2^{2k})$, $\hat{M} = (\tilde{M} + 1)/2^{2k}$, $R = 2^{n+2 \cdot k}$.

Ensure: $S_f \equiv A \cdot B \cdot R^{-1} (\mod M)$, $0 \leqslant S_f < 2\tilde{M}$.

1: $S_0 := 0$
2: $Q_{-1} := 0$
3: **for** $i = 0$ to f **do**
4: $\quad C := 0$
5: $\quad Q_i := (S_i)_0$;
6: $\quad$ **for** $j = 0$ to e **do**
7: $\quad\quad (C, (S_i)_j) := \big((S_i)_{j,k-1..0}, (S_i)_{j-1,w-1..k}\big) + Q_{i-1} M_j + b_i \cdot A_j + C$.
8: $\quad$ **end for**
9: **end for**
10: $S := (S_f \ll k) + Q_f$;
11: **return** S.

3 Optimized HSCM

A scalable Montgomery modular multiplier consists of a group of PEs, RAMs, a FIFO, and the control logics. The whole computation is interleaved and then broken into word-based operations, so that the calculation of S_{i+1} starts only 2 or 1 clock cycles after computing the first word of S_i.

High-radix scalable Montgomery modular multiplications indicates a large value of k in Algorithm 1. Usually, when $k \geqslant 2$ it is called high radix, and typical high-radix scalable designs with $k \geqslant 3$ have been reported in [11,12,20,21]. In scalable Montgomery modular multiplications, one usually divides the multiplicand and the multiplier into a few words and digits, with the word size w and digit size k chosen separately. In most cases, with high radix architectures they are usually chosen as the same [11, 12].

The optimized HSCM divides the original $(w + k)$ serial bits into parallel w-bit operands. Also, the quotient pipeline depth $d = 1$. The high-radix scalable Montgomery modular multiplication is shown in Algorithm 3, in which the carries are denoted by $(C_i)_j$ collectively.

The PE with Algorithm 3 is shown in Figure 2, which consists of 3 full adders; 1 carry save adder

Algorithm 3 Optimized HSCM

Require: Integers $A, B, M, \mu, \tilde{M}, D, n, k, m, f, e$, with $f = \lfloor n/k \rfloor + 3$, $R = 2^{k \cdot f}$, $e = \lfloor n/w \rfloor + 4$, $\mu = -M^{-1} \mod (2^{2k})$, $\tilde{M} = \mu \cdot M$, $N = (\tilde{M} + 1)/2^{2k} = \sum_{j=0}^{f-2} N_j \cdot 2^{kj}$. In addition, $\mathrm{GCD}(M, 2) = 1$, $0 \leqslant A < 2^{n+2k+1}$, $0 \leqslant B < 2^{n+2k+1}$, $B = \sum_{i=0}^{f+1} b_i \cdot 2^{ki}$, with $b_i = 0$ for $i = f$ and $i = f + 1$.

Ensure: $S = A \cdot B \cdot R^{-1} \mod M$, $0 \leqslant S < 2\tilde{M}$.

1: $S'_0 = 0$, $S''_0 = 0$, $T_0 = 0$, $q_{-1} = 0$, $(C_0)_{-1} = 0$;
2: **for** $i = 0$ to $f + 1$ **do**
3: $\quad q_i = \left|(S'_i)_0 + (S''_i)_0\right|_{2^k}$;
4: $\quad 2^k \cdot (C_i)_0 + (S'_{i+1})_0 + (S''_{i+1})_0 := \left(\lfloor S'_i/2^w \rfloor\right)_0 + \left(\lfloor S''_i/2^w \rfloor\right)_0 + |q_{i-1}N_0|_{2^w} + |b_i A_0|_{2^w} + (C_i)_{-1}$;
5: $\quad$ **for** $j = 1$ to $e - 1$ **do**
6: $\quad\quad 2^k (C_i)_j + (S'_{i+1})_j + (S''_{i+1})_j := \left(\lfloor S'_i/2^w \rfloor\right)_j + \left(\lfloor S''_i/2^w \rfloor\right)_j + \lfloor q_{i-1}N_{j-1}/2^w \rfloor + \lfloor b_i A_{j-1}/2^w \rfloor + |q_{i-1}N_j|_{2^w} + |b_i \cdot A_j|_{2^w} + (C_i)_{j-1}$;
7: $\quad$ **end for**
8: **end for**
9: $S = 2^k (S'_{f+2} + S''_{f+2}) + q_{f+1}$;
10: **return** S.

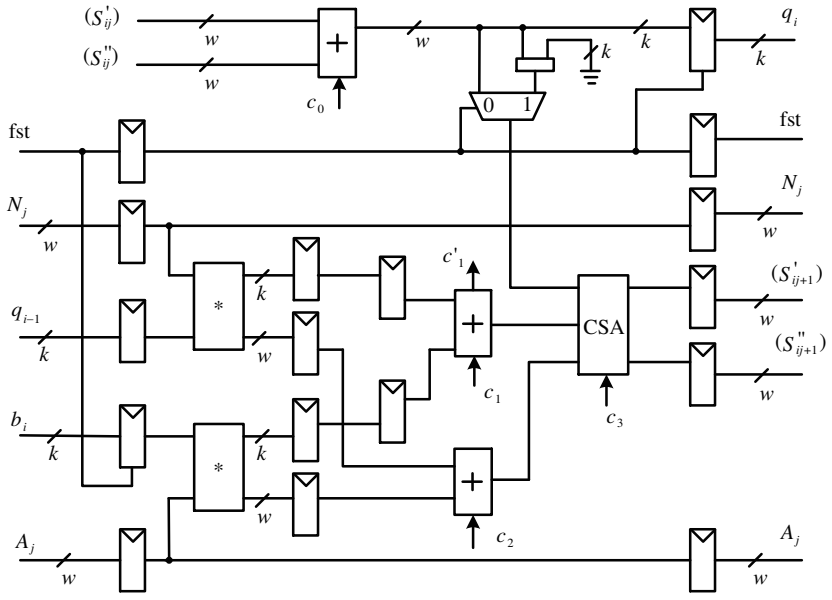

Figure 2 PE with optimized HSCM.

(CSA), 2 multipliers, 1 multiplexor, and a few registers. The carry out c_1 is of weight 2^k and comes from the previous PE. Since the latency between PEs are 2 clock cycles, the carry out c'_1 should be buffered by 2 clock cycles and then fed into the next PE as c_1. The other carries are buffered for 1 clock cycle and then used in the current PE. The carries $2^w \cdot (C_i)_j$ include c'_1 for all the words in the loops, which makes the carry $(C_i)_{-1}$ into the first word non-ignorable.

An n-bit modular exponentiation consists of continuous modular multiplications, and the input and output range of Montgomery modular multiplications are in the same range [22]. This condition can be met by selecting proper value of f, just referring to the choice of $m + d$ in Algorithm 1. In this work, it is satisfied by setting $f = \lfloor n/k \rfloor + 3$, $e = \lceil n/k \rceil + 4$, as is explained below.

- With e. For $i \leqslant f - 1$ there is

$$S_i \leqslant 2^{n+3k+1} + 2^{n+k}, \tag{1}$$

where $k \geqslant 2$. By recursion, one gets

$$\begin{aligned} S_{i+1} &\leqslant (2^{n+2k+1} + 2^n) + (2^k - 1) \cdot (2^n - 1) + (2^k - 1)(2^{n+2k+1} - 1) \\ &< 2^{n+3k} + 2^{n+k}. \end{aligned} \tag{2}$$

Table 1 Optimized HSCM VS. the architecture in [12]

Reference	Length (bits)	Area	Critical path	Clock cycles
This work	n	$2L \cdot A_{\mathrm{MULT}(w\times w)}$ +$15wL \cdot A_{\mathrm{REG}}$+$4wL \cdot A_{\mathrm{FA}}$	$T_{\mathrm{MULT}(w\times w)}$	$2n^2/(Lw^2)$
[12]	n	$2L \cdot A_{\mathrm{MULT}(w\times w)}$ +$15wL \cdot A_{\mathrm{REG}}$+$8wL \cdot A_{\mathrm{FA}}$	$T_{\mathrm{MULT}(w\times w)}$+$T_{\mathrm{CSA}}$	$2n^2/(Lw^2)$

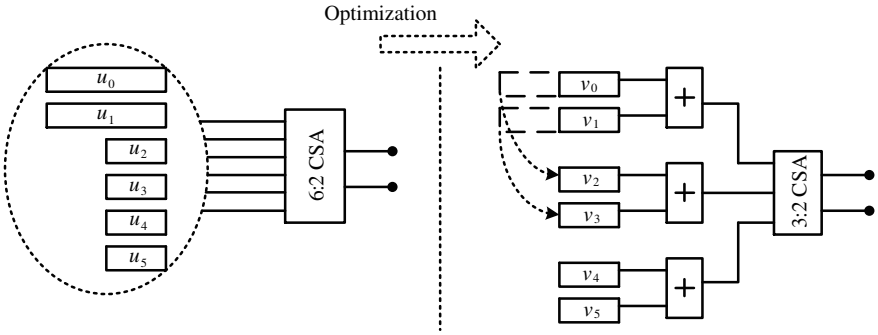

Figure 3 Optimization of PE.

Thus, the length $e = \lceil n/k \rceil + 4$ is enough to store the temporary results of high-radix scalable Montgomery modular multiplications.

• With f. The last two loops with Algorithm 3 should reduce the temporary results from e words to f words, i.e.,

$$\begin{aligned} S_{f+1} &< (2^{n+3k} + 2^{n+k})/2^k + (2^k - 1)(2^n - 1) \\ &= 2^{n+2k} + 2^{n+k} - 2^k + 1, \\ S_{f+2} &\leqslant 2^{n+k} + 2^n - 1. \end{aligned}$$

The final result reads

$$\begin{aligned} S &\leqslant 2^{n+2k} + 2^{n+k} - 2^k + (2^k - 1) \\ &= 2^{n+2k} + 2^{n+k} - 1 < 2^{n+2k+1}. \end{aligned} \tag{3}$$

Obviously, S can be denoted by $n/k + 3$ words, which leads to $f = \lfloor n/k \rfloor + 3$.

In Table 1, our optimized HSCM is compared with that in [12] in case of $w = k$, where L marks the number of PEs. It can be found that the area overhead and the critical path are both reduced, while the time complexity remains constant. The symbol $A_{\mathrm{MULT}}(w \times w)$ denotes the area of a $w \times w$-bit multiplier, while $T_{\mathrm{MULT}}(w \times w)$ denotes its critical path delay. The symbol 'REG' denotes a register, 'FA' denotes a full adder, and 'CSA' denotes a carry save adder. The critical path is measured by ASIC gates, which varies between $T_{w-\text{bit FA}}$+T_{CSA} in FPGA devices.

The difference between the PE of this work and that in [12] is illustrated in Figure 3, where the original 6:2 CSA is replaced by a 3:2 CSA and 3 full adders. Its optimization goal is to reduce critical path, by separating the CSA from the multiplier. It will be shown in Section 5 the maximum frequency for FPGA implementation has been greatly increased. While the FA-SPA resistant architecture increases area overhead, and enlarges the room for further speedup when the hardware resources are abundant.

3.1 Modular exponentiation

Modular exponentiation can be performed by a left-to-right or right-to-left binary method, counting the exponent bit one by one from left to right or vice versa. It breaks the exponentiation into serial modular squaring and modular multiplications [23].

The sliding window method accelerates the modular exponentiation by reducing the number of modular multiplications [24, 25], while requiring more memories to store precomputation results. The sliding-window method is described in Appendix A.

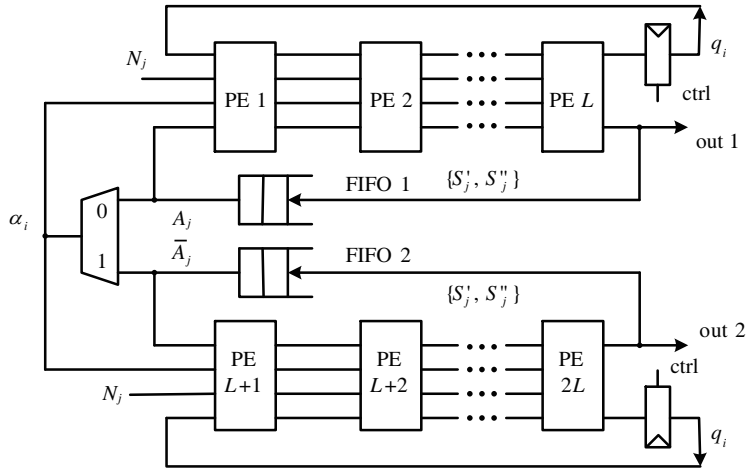

Figure 4 Modular exponentiation by Montgomery ladder and two HSCMs.

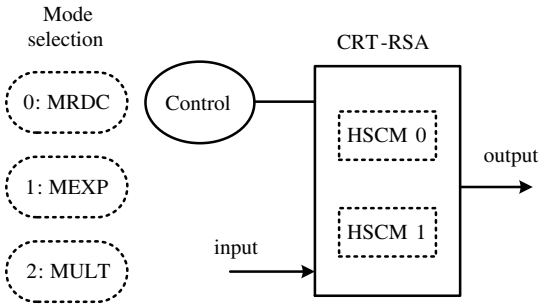

Figure 5 Modes of CRT-RSA.

The sliding-window method applies many RAMs to accelerate modular exponentiations, which is suitable for the background with rich memory units.

Another idea is to apply the FA-SPA resistant modular exponentiation architecture, which performs all n-bit modular exponentiations in the same time. For this case, one can employ two scalable Montgomery modular multipliers in parallel, as is shown in Figure 4. The original multiplicand words A_j is still represented as A_j on the top HSCM, or as $\overline{A_j}$ at bottom HSCM. Meanwhile, the multiplier digits b_i is replaced by α_i, which is shared between two HSCMs in Montgomery ladder algorithm [18]. There are two results out from the HSCMs in coherence, i.e., out2 = (out1 $\cdot$ C) mod M, where C is the base and M is the modulus.

3.2 CRT-RSA

The critical point in CRT-RSA is to separate an n-bit modular exponentiation into 2 $(n/2)$-bit modular exponentiations, which greatly decreases the computations [13]. Suppose p and q are the two secret RSA moduli, ω is the message in binary form, $d_p = d \mod (p-1)$, $d_q = d \mod (q-1)$, $\omega_p = \omega \mod p$, $m_q = \omega \mod q$, then one can compute $S_p = \omega_p^{d_p} \mod p$, $S_q = \omega_q^{d_q} \mod q$, and obtain [13]

$$S = S_p \cdot v \cdot q + S_q \cdot u \cdot p - t \cdot N, \tag{4}$$

where $u = q^{-1} \mod p$, $v = p^{-1} \mod q$, $t = 0, 1, 2, 3$. Obviously, the original modular exponentiation $S = \omega^d$ is not directly implemented in CRT-RSA.

In this work, the CRT-RSA is based on two HSCMs and the external controller. As is shown in Figure 5, the operations of CRT can be divided into three sequential stages or modes, i.e., 0: modular reduction (MRDC), 1: modular exponentiation (MEXP), 2: multiplication (MULT).

The modes except 'MEXP' are discussed below.

3.2.1 *Scalable modular reduction*

During RSA decryption, ω is received as a whole, so one has to compute $\omega_p = \omega \mod p$ and $\omega_q = \omega \mod q$ before applying CRT-RSA. Each of them is an $n-bit$ modular reduction over a $(n/2)$-bit modulus, which is cubersome due to the large value of n. In the literature, they are assumed to be available from software environment, which is separate from the direct RSA decryption process in a hardware unit. The separation not only increases the efforts in the high level of a computing system, but also raises the risk of leaking the p and q values.

Therefore, it is beneficial to conduct the modular reduction together with RSA decryption in the same hardware unit. Usually, modular reductions can be performed by 'shift and subtract', which may deteriorate the critical path due to large moduli p and q. As what we did with modular multiplications, a scalable modular reduction unit may avoid the path deterioration.

In fact, scalable modular reduction here can be embedded in the existing HSCM unit. The basic idea is inspired by the Montgomery modular multiplications in [26,27], which compute $Z = A \cdot B \cdot R^{-1} \mod M$ with $R = 2^n$ as follows:

- $T = A \cdot B$, $T_1 = \lfloor T/R \rfloor$, $T_0 = T \mod R$.
- $Z = T_1 + T_0 \cdot |R^2|_M \cdot R^{-1}$.

The above procedures keep the Montgomery algorithm unchanged, but process the product T as two separate parts. While the lower part needs further processing, the higher part only requires an addition.

The procedures in this work are shown in Eq. (5),

$$\begin{aligned} &x = \omega \cdot R^{-1} \bmod p, \\ &(\omega \cdot R) \bmod p = x \cdot \left|R^3\right|_p \cdot R^{-1} \bmod p. \end{aligned} \tag{5}$$

Eq. (5) just obtains ω_p in the Montgomery domain with $R = 2^{n+3k}$, where the first step can be assumed as an Montgomery modular product as follows:

$$x = \omega \cdot R^{-1} \bmod p = \lfloor \omega/R \rfloor + |\omega|_R \cdot 1 \cdot R^{-1} \bmod p.$$

Notice that the two steps in Eq. (5) cannot be combined, otherwise the intermediate results may get out of the definition ranges. So the modular reduction can be performed by two sequential scalable Montgomery modular multiplications.

3.2.2 *Scalable multiplication in CRT-RSA*

Second, the combination of 2 modular exponentiation results by CRT will require large multipliers to perform multiplications. Because modular exponentiations or multiplications are often carried out by interleaved multiplications and reductions (or Horner's rule), it seems impossible to carry out multiplications in the existing hardware. In fact, large multipliers may consume larger overhead area or lead to long critical path.

At the end of CRT-RSA, there are two multiprecision multiplications, which is hard for interleaved Montgomery modular multiplications, where no separate large multipliers exist. In this work, we propose a scalable multiplication method to deal with large multiplications. It computes the large multiplication by small-size multiplications and accumulates results in a pipelined way. The proposed method is shown in Algorithm 4, and another form of pipelined multipliers can be found in [28].

In Algorithm 4, the subscript j with (S', S'') denotes the j-th word, while the subscript i is used to denote the i-th outer loop. Its dataflow is similar to scalable Montgomery modular multiplications. By setting $q_{i-1} \equiv 0$ or $N_j \equiv 0$ in Figure 2, scalable multiplication in Algorithm 4 can be implemented in HSCM with the same PEs.

The dataflow of the scalable multiplier can be considered similar to the scalable Montgomery modular multiplier only for the higher half of the products. In order to collect the words with the lower half of the products, a multiplexor should be used. As is shown in Figure 6, the words of the lower half (about $m + 3$ words) are selected from PEs in sequence, while the higher words can be piped out from a fixed PE.

Algorithm 4 Scalable multiplication

Require: A, B are both n-bit numbers. $n = m \cdot k$, $A = \sum_{i=0}^{m-1} A_i \cdot 2^{k \cdot i}$, $B = \sum_{j=0}^{m-1} b_j \cdot 2^{k \cdot j}$.
Ensure: $S = A \cdot B$.
1: $S_{-1} = 0$;
2: **for** $i = 0$ to $m - 1$ **do**
3: $\quad 2^w \cdot (C_i)_0 + (S_i)_0 = (S_{i-1}/2^k)_0 + b_i \cdot A_0$;
4: $\quad$ **for** $j = 1$ to $m - 1$ **do**
5: $\quad\quad 2^w \cdot (C_i)_j + (S_i)_j = (S_{i-1}/2^k)_j + b_i \cdot A_j + (C_i)_{j-1}$;
6: $\quad$ **end for**
7: **end for**
8: **return** $S = ((S_{m-1})_m, \ldots, (S_{m-1})_1, (S_{m-1})_0, (S_{m-2})_0, \ldots, (S_0)_0)$.

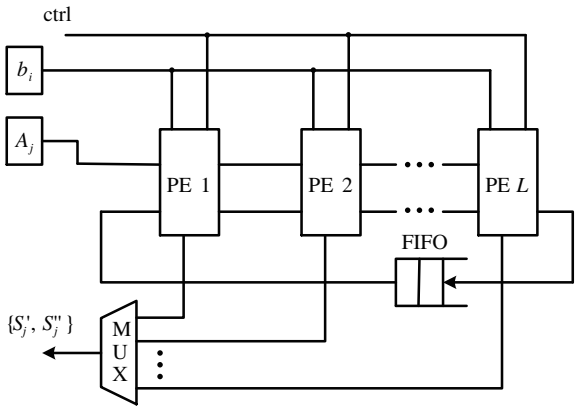

Figure 6 Outputs from the scalable multiplier.

4 Reduction after modular exponentiation

The result obtained from high-radix scalable Montgomery modular multiplications is a $(n + 2k + 1)$-bit number, requiring final modular reductions. In order to perform this modular reduction, we present an approximation method in Appendix B. A similar bounding technique can be found in [29] for modular multiplication.

In the modular reduction of modular exponentiation, we can reduce the final result Z by $R_1 = Z - \tilde{q} \cdot M$, with $\tilde{q} = \left\lfloor \frac{Y-1}{M+1} \right\rfloor$. Meanwhile, let us suppose that the accurate result read $R = Y \mod M = Y - q \cdot M$. For modular exponentiation, the upper bound of q at output can be considered in detail as follows.

In modular exponentiation, the last Montgomery modular multiplication is HSCM$(1, B)$, with $B = S$. In the inner loop of HSCM, for i there are:

$$\begin{aligned}
S_1 &= S_0 + q_0 \cdot N + b_0 \cdot 1 \\
&= 0 + 0 \cdot N + b_i \leqslant (2^k - 1), \\
S_2 &= \lfloor S_1/2^k \rfloor + q_0 \cdot N + b_1 \cdot 1 \\
&\leqslant 0 + (2^k - 1) \cdot (2^n - 1) + (2^k - 1) \\
&\leqslant (2^k - 1) \cdot 2^n, \\
S_3 &= \lfloor S_2/2^k \rfloor + q_1 \cdot N + b_2 \cdot 1 \\
&\leqslant (2^n - 2^{n-k}) + (2^k - 1) \cdot (2^n - 1) + (2^k - 1) \\
&\leqslant 2^{n+k} - 2^{n-k}, \\
S_4 &= \lfloor S_3/2^k \rfloor + q_2 \cdot N + b_3 \cdot 1 \\
&\leqslant (2^n - 2^{n-2k}) + (2^k - 1) \cdot (2^n - 1) + (2^k - 1) \\
&\leqslant 2^{n+k} - 2^{n-2k}, \quad \ldots, \\
&\ldots.
\end{aligned}$$

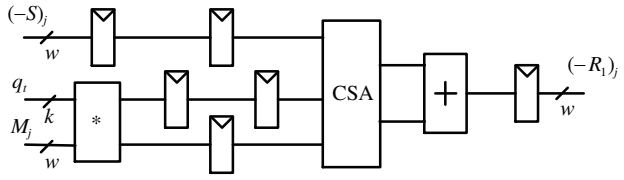

Figure 7 Word-based operation for modular reduction: $R_1' = -S + q_t M$, with $q_t = q_h$ or $q_t = q_l$.

The above equations can be sequentially written until S_{f-1}, i.e.,

$$\begin{aligned} S_{f-1} = S_{m+2} &= \lfloor S_{m+1}/2^k \rfloor + q_m \cdot N + b_{m+1} \cdot 1 \\ &\leqslant (2^n - 2^{n-m\cdot k}) + (2^k - 1) \cdot (2^n - 1) + (2^k - 1) \\ &\leqslant 2^{n+k} - 1, \end{aligned} \tag{6}$$

where $f = m + 3$, $n = m \cdot k$ is used. Next,

$$\begin{aligned} S_{m+3} &\leqslant 2^n + (2^k - 1) \cdot N + 0 \cdot 1 \\ &\leqslant 2^n + (2^k - 1) \cdot ((2^{2k} - 1)M + 1)/2^{2k} \\ &= 2^n + (2^k - 1)M - \frac{M-1}{2^{2k}}(2^k - 1) \\ &\leqslant 2^n + (2^k - 1)M - 2^{n-1-2k}(2^k - 1) \\ &= 2^n + (2^k - 1) \cdot (M - 2^{n-2k-1}), \end{aligned} \tag{7}$$

where $M \geqslant 2^{n-1} + 1$ is assumed.

Then,

$$\begin{aligned} S_{m+4} &\leqslant 2^{n-k} + (2^k - 1) \cdot (M - 2^{n-2k-1}) \cdot \frac{1 + 2^k}{2^k}, \\ S_{m+5} &\leqslant 2^{n-2k} + (2^k - 1) \cdot (M - 2^{n-2k-1}) \cdot \frac{1 + 2^k + 2^{2k}}{2^{2k}}. \end{aligned}$$

According to Algorithm 3, the final result reads

$$\begin{aligned} S = 2^k \cdot S_{f+2} + q_{f+1} &= 2^k \cdot S_{m+5} + q_{m+4} \\ &< 2^{n-k} + (2^k - 1)(2^k + 1 + 2^{-k})(M - 2^{n-2k-1}) + 2^k - 1 \\ &= 2^{n-k} + (2^{2k} - 2^{-k})(M - 2^{n-2k-1}) + 2^k - 1 \\ &= (2^{2k} - 2^{-k})M - 2^{n-1} + 2^{n-k} + 2^{n-3k-1} + 2^k - 1 \\ &< (2^{2k} - 2^{-k})M - 2^{n-2}, \end{aligned} \tag{8}$$

where the final inequality is satisfied by considering $k \geqslant 3$ for high-radix Montgomery modular multiplication. Now, dividing both sides of Eq. (8) by M yields

$$\frac{S}{M} < 2^{2k} - 2^{-k} - \frac{2^{n-2}}{M}. \tag{9}$$

As a result,

$$q = \left\lfloor \frac{S}{M} \right\rfloor \leqslant \frac{S}{M} < 2^{2k} - 2^{-k} - \frac{2^{n-2}}{M} < 2^{2k}. \tag{10}$$

Therefore, $\tilde{q} \leqslant q \leqslant 2^{2k} - 1$, and both of them can be represented by $2k$ binary bits.

To avoid the signed multiplication $-\tilde{q} \cdot M$, we compute $R_1' = -R_1 = \tilde{q} \cdot M + (-S)$. For the sake of hardware implementation, we set $\tilde{q} = 2^k q_h + q_l$, with $q_h < 2^k$, $q_l < 2^k$. Then, $R_1' = 2^k q_h \cdot M + q_l \cdot M + (-S)$. These operations can be processed in a separate unit as is shown in Figure 7. The input to the multiplier are the quotient digits and the digits with the original modulus M_j (which is different from the transformed

Table 2 Hardware implementationx of HSCM

Reference	Length (bits)	PEs	Digit (bits)	Word (bits)	Technology	Area (slices/Luts/Dsps)	Max freq. (MHz)	Cycles (Clock)	Time (μs)
This work	1024	16	16	16	0.18 μm CMOS	131K gates	263.17	354	1.35
This work	1024	16	16	16	0.13 μm CMOS	140K gates	312.5	354	1.13
This work	1024	16	16	16	90 nm CMOS	130K gates	526.32	354	0.62
This work	1024	32	16	16	0.18 μm CMOS	232K gates	263.17	218	0.72
This work	1024	32	16	16	0.13 μm CMOS	236K gates	303.03	218	1.06
This work	1024	32	16	16	90 nm CMOS	207K gates	526.32	218	0.41
This work	1024	16	16	16	XC2V-6	2096/1799/32	202	357	1.77
This work	1024	32	16	16	XC2V-6	3846/3126/64	202	218	1.09
This work	1024	16	16	16	XC2V-4	2096/1800/32	151	357	2.37
This work	2048	32	16	16	90 nm CMOS	234K gates	526.32	674	1.28
This work	2048	32	16	16	XC2V-6	3948/3375/64	202	677	3.35
This work	2048	32	16	16	XC2V-4	3948/3375/64	117	677	5.79
[30]	1024	65	1	16	0.13 μm CMOS	82K gates	675.68	1107	1.64
[10]	1024	65	2	16	0.13 μm CMOS	139K gates	584	590	1.01
[30]	1024	257	1	4	XC2V-6	5158/5430/0	254.55	1287	4.05
[30]	1024	257	1	4	XC2V-4	4647/4918/0	195.98	1287	5.26
[10]	1024	65	2	16	XC2V-6	11516 LUTs	217	–	2.7
[10]	1024	65	2	16	XC2V-4	11516 LUTs	174	–	3.37
[31]	1024	65	1	16	XC2V-4	9319 LUTs	116.4	1088	9.349
[8]	1024	64	2	16	XC2V-4	8310/11416/0	165	602	3.65
[8]	1024	64	2	16	XC2V-6	8268/11361	208	602	2.89
[32]	1024	–	–	–	XC2V	11520 slices	111.32	1002	9.0
[31]	2048	129	1	16	XC2V-4	18535 LUTs	116.4	2176	18.698
[32]	2048	–	–	–	XC2V	23108 slices	90.73	1002	11.03

modulus N_j). The product is then separated into two parts Z_H and Z_L, with $2^k \cdot Z_H + Z_L = q_t \cdot M_j$, and both of them are delayed by 2 clock cycles before computation. The carry outs are fed back to the adders as carry ins in the next clock cycle. In addition, to save one multiplier in some FPGA devices, such a unit can also be implemented by revising the logic in some PE.

At last, we compute $R'_2 = R'_1 + M$, and the final result can be covered in two cases:

- If $R'_2 < 0$, then $M - R_1 < 0$, i.e., $R_1 > M$. In this case, $R = R_1 - M = -R'_2$.
- if $R'_2 \geqslant 0$, then $M - R_1 \geqslant 0$, i.e., $R_1 \leqslant M$. In this case, $R = R_1 = -R'_1$.

Since $R'_1 = -R_1 > -2M > -2^{n+1}$, R'_1 can be represented by an $(n+2)$-bit signed number. Also, since $R'_2 = M - R_1 \in (-M, M]$, it can be represented by an $(n+1)$-bit signed number.

Finally, the output range will fall between $(0, M]$, and it is a little different from expected bound of $[0, M)$. However, in cryptographic applications the modulus M is a prime (e.g., ElGamal Cryptography), or a product of two primes (e.g., RSA cryptography), which makes the disparity negligible. Suppose it appears that $C^E \mod M = 0$, then $M|C$, however that never occurs if $C < M$.

5 Hardware implementation

The HSCM, the modular exponentiation units based on HSCM and the CRT-RSA units based on HSCM have all been described by Verilog HDL, simulated in Modelsim SE 6.2, and synthesized by Synopsys Synplify Pro 9.6.2 targeting FPGA devices or Design Compiler targeting ASIC gates. All results for FPGA devices are obtained after placing and routing to Xilinx ISE 10.1. The word 'Counted' in the table denotes the corresponding units those have been counted as ASIC gates in the tatal area.

The hardware implementation of the HSCM is shown in Table 2, where the final result are kept in $[0, 2^{2k+1}M)$.

Table 3 Modular exponentiation by HSCM without sliding-window method

Reference	Length	Technology	PEs	Digit (bits)	Word (bits)	Area (slices/Luts/Dsps)	RAM (Kbits)	Max freq. (MHz)	Time (ms)
This work	1024	0.18 μm CMOS	16	16	16	130K gates	Counted	238.10	2.27
This work	1024	0.18 μm CMOS	32	16	16	222K gates	Counted	232.56	1.42
This work*	1024	0.18 μm CMOS	32	16	16	255K gates	Counted	250.0	1.44
This work*	1024	XC2V6000-6	32	16	16	6992/8862/64	~ 5	185	1.95
This work	1024	XC3S1400A-5	16	16	16	2397/2574/32	~ 5	158	3.45
This work	1024	XC2V1000-6	16	16	16	2635/2551/32	~ 5	195	2.80
This work	1024	XC2V2000-6	32	16	16	4404/3900/64	~ 5	196	1.70
This work	1024	XC4VSX25-10	32	16	16	3833/3865/64	~ 5	222	1.50
This work	1024	XC4VSX25-12	32	16	16	3837/3871/64	~ 5	280	1.19
This work	2048	0.18 μm CMOS	32	16	16	251K gates	Counted	232.56	8.92
This work	2048	XC2V3000-6	32	16	16	5550/6198/64	~ 9	190	10.92
This work	2048	XC4VSX25-12	32	16	16	5012/6202/64	~ 9	260	7.98
This work	2048	XC4VSX25-12	64	16	16	8913/10714/128	~ 9	256.0	4.91
[33]	1024	90 nm CMOS	–	32		11.2K gates	–	471.70	7.27
[33]	1024	90 nm CMOS	–	128	128	150K gates	–	421.94	0.67
[12]	1024	XC2V2000-6	16	16	16	–/3825/32	~ 5	135.7	5.1
[9]	1024	XC2V6000-6	65	2	16	11611/18789/0	0	161	3.72
[35]	1024	XC4000-9	–	4	–	6633 CLBs/-/0	0	45.6	11.95
[36]	1024	XC2V3000-6	62	17	17	14334/-/62	–	90.11	2.33

Refs. [10, 30] present a radix-2 and radix-4 low-latency designs with similar algorithms and quotient pipelines. It can be found that for ASIC implementations this work has about the same area overhead and performance to that in [10], and higher performance than that in [30]. As expected, for FPGA implementation this work then gets much faster than [10, 30] owing to dedicated multipliers.

The scalable Montgomery modular multipliers in [31] and [8] achieves similar performances to that in [10, 30] in another low-latency architecture. Also due to dedicated multipliers, this work is faster than [8, 31] on FPGA platforms.

The Montgomery modular multiplier with [32] does not possess scalable architecture. It uses long carry save additions to perform interleaved Montgomery algorithm. As is seen from Table 2, this work is faster than that for both 1024-bit and 2048-bit architectures.

Modular exponentiation based on HSCM without sliding-window method is shown in Table 3. Only a left-to-right binary method is used, and the exponent is chosen as $E = (2/3) \times (2^n - 1)$, with $n = 1024$ or 2048, which enjoys a Hamming weight of 0.5. The modular reduction at last step has been included.

This work with a superscript (*) is with the FA-SPA-Resistant modular exponentiation, which has two HSCM located parallelly. As a result, its time for modular exponentiation is always about $(n+2)\cdot T_{\mathrm{HSCM}}$, where T_{HSCM} is the time to complete one n-bit Montgomery modular multiplication by HSCM. As was expected if whole area overhead is considered, then this architecture will be less efficient in terms of Area×Time than that by sliding-window method. This is a usual tradeoff between performance and security in cryptosystems.

The modular exponentiation in [33] is performed by a high-radix Montgomery modular multiplier [34], in which only one central arithmetic unit is employed to calculate the word-based modular multiplication. While it obtains similar performance and scalability like this work, the combinational area overhead will soon get much larger in case of very high radix multipliers. As is well-known, a large combinational circuit will lead to a great increase in power consumption. By contrast, this work does not rely on much large multipliers, and the combinational and non-combinational circuits keep a more average ratio. In addition, the area overhead of [33] with Table 3 is only composed of the data path, with register files and other memory units not taken into account. Finally, as long as FPGA implementations are concerned, the design in [33] will get inconvenient due to the fixed sizes of DSPs.

In [12], the quotient pipeline is firstly applied in HSCM. As is shown in Table 3, this work completes 1024-bit modular exponentiation in 2.80 ms, which is 82.1% faster than 5.1 ms in [12], where both designs

Table 4 Modular exponentiation by HSCM and sliding-window method

Reference	Length	Technology	PEs	Digit (bits)	Word (bits)	Area (Slices/Luts/Dsps)	RAM (Kbits)	Max freq. (MHz)	Cycles (Clock)	Time (ms)
This work	1024	0.18 μm CMOS	16	16	16	159k gates	Counted	232.56	454,648	1.95
This work	1024	0.18 μm CMOS	32	16	16	246k gates	Counted	232.56	277,140	1.19
This work	2048	0.18 μm CMOS	32	16	16	303k gates	Counted	232.56	1,714,940	7.37
This work	1024	XC4VLX15-10	16	16	16	2736/2987/32	12	222	454,648	2.05
This work	1024	XC2V1000-6	16	16	16	3025/2991/32	12	196	454,648	2.32
This work	1024	XC2V3000-6	32	16	16	4777/4317/64	12	192	277,140	1.45
This work	2048	XC2V1000-6	16	16	16	5020/6982/32	24	165	3,066,220	18.58
This work	2048	XC4VSX25-12	16	16	16	4737/6992/32	24	262	3,066,220	11.70
This work	2048	XC2V3000-6	32	16	16	6782/8324/64	24	177	1,714,940	9.69
This work	2048	XC4VSX25-12	32	16	16	6245/8322/64	24	271	1,714,940	6.33
[25]	1024	XC4VFX12-10	17	17	17	3937/−/17	18	400	684,000	1.71
[25]	2048	XC4VFX12-10	17	17	17	3937/−/17	18	400	5,040,000	12.6

employ 16 PEs .

In [9], a centrosymmetric modular exponentiation architecture based on common-multiplicand Montgomery modular multiplication is described. It achieves high performance in terms of time, area and security. Nevertheless, this work still obtains a significant speedup.

In [35], the high-radix Montgomery modular multiplier is performed in FPGA by precomputation rather than multiplications. After computing and storing multiple of $\delta \cdot A$, the multiplication of n-bit operand by 4-bit digits can be implemented by multiplexors in a series of PEs. Owing to much higher radix of 2^{16} with dedicated multipliers, this work is much faster.

In [36], the modular exponentiations are also performed by quotient pipelined high-radix Montgomery modular. The architecture still performs serial computation of two partial products, with the multiplications of $q_i \cdot N_j$ and $b_i \cdot A_j$ are not interleaved. Compared with this work, its performance is greatly reduced due to a long critical path.

In Table 4, [25] presents a high-performance modular exponentiation unit by the use of FPGA DSPs, which is faster than this work in the same platform. It is most likely because the design in [25] just targets the FPGA devices and makes the best use of DSPs. For example, a 16-bit full addition and CSA in FPGA will be always slower than the dedicated 16-bit multipliers with DSPs. In fact, this problem directly constrains the circuit frequency in FPGA devices. By contrast, [25] employs a double-edge triggered circuit to counter this problem, so the highest frequency keeps up with the DSP slice, which reaches about 400 MHz. As a result, although our proposed circuit should function at a higher frequency than that in [25] for ASIC implementation, its maximum frequency is more than 30% below that in [25] in Xilinx IV devices. Other factors like a larger word size and sliding-window size also helps the unit in [25] faster. It could be considered that the aforementioned problems causes a slowdown of about 40.5%. Nevertheless, the advantages with [25] will disappear for ASIC implementation.

The CRT-RSA in Table 5 is based on modular exponentiation with sliding-window method. However, it can be easily extended from the previous modular exponentiation by two parallel HSCMs, if the FA-SPA-Resistant RSA is required. In this case, one may use sequential CRT-RSA to reduce area overhead.

In Table 5, the CRT-RSA in [35] is performed by two parallel 512-bit Montgomery modular multipliers, both of which are implemented as radix-16 architectures. The result of CRT-RSA in [32] is obtained by a few long carry save adders. If it is able to perform a 2048-bit RSA decryption with the same hardware, then the time delay may be multiplied by a factor of 2^3, i.e., $2.73 \times 8 \approx 21.8$ (ms), which is much slower than this work with the same FPGA device (below 3 ms).

The CRT-RSA in [37] is marked by a large number of memory units, which are not accounted in area overhead. This work employs 35–70 Kbits of memory units, while the design in [37] instantiates about 1224 kbits of memory units, where Montgomery algorithm in residue number system is applied with parallel multipliers. Since one bit of memory usually accounts for several ASIC gates, the area overhead in [37] will be several times larger than this work. As the time delay for 2048-bit CRT-RSA is concerned,

Table 5 CRT-RSA decryption by different hardware architectures

Reference	Length (bits)	PEs	Digit (bits)	Word (bits)	Technology	Area (Slices/Luts/Dsps)	RAM (Kbits)	Max freq. (MHz)	Cycles (Clock)	Time (ms)
This work	2048	32	16	16	0.18 μm CMOS	395K gates	Counted	222.22	455,274	2.05
This work	4096	64	16	16	0.18 μm CMOS	699K gates	Counted	222.22	1,724,369	7.76
This work	2048	32	16	16	XC2V6000-6	6645/7098/64	35	177	455,274	2.57
This work	2048	64	16	16	XC2V6000-6	10360/10246/128	35	160	278,496	1.74
This work	4096	32	16	16	XC2V3000-6	10674/15206/64	70	156	3,082,565	19.76
This work	4096	32	16	16	XC4VSX25-12	10162/15184/64	70	260	3,082,565	11.86
This work	4096	64	16	16	XC2V6000-6	14041/17409/128	70	156	1,724,369	11.05
This work	4096	64	16	16	XC4VSX35-12	11727/15013/128	70	252	1,724,369	6.84
This work	4096	128	16	16	XC4VSX55-12	18960/22854/256	70	238	1,045,271	4.69
[35]	1024		4		XC40250XV	6826 slices	–	45.6	141,360	3.10
[32]	1024				XC2V6000	26136 slices	–	97.08	265,028	2.73
[37]	2048				0.25 μm CMOS	333K gates	1244	80	712,000	8.9

this work is about 4.3 times faster, while the technology may contribute to about 1.4 times (.25/.18). Therefore, this work is probably more efficient than [37] with respect to the same technology used.

6 Conclusion

This paper gives a brief introduction of the quotient pipelined high-radix Montgomery modular multiplier in the background of public key cryptography. Then it shows a simple way of reducing the area overhead and critical path to obtain higher performance. For the reduction after modular exponentiation that is not considered in the literature before, an easy approximation method is also presented to tackle it. In addition, the proposed high-radix Montgomery modular multipliers can be applied for FA-SPA-Resistant modular exponentiation by the Montgomery powering ladder algorithm.

Then following the discussion of CRT-RSA, this paper for the first time demonstrates that the high-radix scalable Montgomery modular multiplier can be configured for scalable modular reduction and multiplication, and therefore gets quite suitable for CRT-RSA. In this way, the whole CRT-RSA can divided into three macro-stages and implemented in just the same hardware. Such a configurable architecture is very efficient compared with state-of-the-art work in terms of area and time and works at much lower frequency with about the same Time×Area.

Acknowledgements

This work was partly supported by the National High Technology Research and Development Program of China (Grant No. 2012AA012402), the Tsinghua National Laboratory for Information Science and Technology (to be granted in 2015), and the Independent Research and Development Program of Tsinghua University (Grant No. 2011Z05116). The authors would also like to thank the editor and reviewers for their comments. The first author also appreciates the discussion with Jicheng Lu, Zhimin Zhang and Qing Li, and the technical support from Dong Zhang, Yulan Fan, Juan Ling and Yeyang Zheng from Shanghai Fudan Microelectronics Group Company.

References

1 Rivest R, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Commun ACM1978, 21: 120–126

2 Koblitz N. Elliptic curve cryptosystems. Math Comp, 1987, 48: 203–209

3 Eisenbarth T, Güneysu T, Heyse S, et al. Microeliece: Mceliece for embedded devices. In: Clavier C, Gaj K, eds. CHES. 2009, 5747: 49–64

4 Tenca A, Koç Ç K. A scalable architecture for Montgomery multiplication. In: First International Workshop on Cryptographic Hardware and Embedded Systems, Worcester, USA 1999. 94–108

5 Tenca A, Koç Ç K. A scalable architecture for modular multiplication based on montgomery's algorithm. IEEE Trans Comp, 2003, 52: 1215–1221

6 Montgomery P. Modular multiplication without trial division. Math Comp, 1985, 44: 519–521
7 Orup H. Simplifying quotient determination in high-radix modular multiplication. In: The 12th IEEE Symposium on Computer Arithmetic, 1995. 193–199
8 Wu T. Improving radix-4 feedforward scalable Montgomery modular multiplier by precomputation and double Booth-encodings. In: IEEE International Conference on Computer Science and Network Technology, Dalian, 2013. 596–600
9 Wu T, Li S, Liu L. Fast, compact and symmetric modular exponentiation architecture by common-multiplicand Montgomery modular multiplications. Integ VLSI J, 2013, 46: 323–332
10 Wang S H, Lin W C, Ye J H, et al. Fast scalable radix-4 Montgomery modular multiplier. In: IEEE Symposium on Circuits and Systems, Seoul, 2012. 3049–3052
11 Kelley K, Harris D. Parallelized very high radix scalable Montgomery multipliers. In: Proc. IEEE 39th Asilomar Conference on Signals, Systems, and Computers, Asilomar, 2005. 1196–1200
12 Jiang N, Harris D. Quotient pipelined very high radix scalable Montgomery multipliers. In: The 40th Asilomar Conference on Signals, Systems and Computers, Asilomar, 2006. 1673–1677
13 Quisquater J J, Couvreur C. Fast decipherment algorithm for RSA public-key cryptosystem. Electr Lett, 1982, 18: 905–907
14 Taylor F. Residue arithmetic: A tutorial with examples. Computer, 1984, 17: 50–62
15 Han L, Wang X, Xu G. On an attack on rsa with small crt-exponents. Sci China Inf Sci, 2010, 53: 1511–1518
16 Fournaris A. Fault and simple power attack resistant RSA using Montgomery modular multiplication. In: IEEE Symposium on Circuits and Systems, Paris, 2010. 1875–1878
17 Giraud C. An RSA implementation resistant to fault attacks and to simple power analysis. IEEE Trans Comp, 2006, 55: 1116–1120
18 Joye M, Yen S. The montgomery powering ladder. In: International Workshop on Cryptographic Hardware and Embedded Systems. Lect Notes Comp Sci, 2002. 2523: 291–302
19 Shand M, Vuillemin J. Fast implementations of RSA cryptography. In: The 11th IEEE Symposium on Computer Arithmetic, Windsor, 1993. 252–259
20 Tenca A, Todorov G, Koç Ç K. High-radix design of a scalable modular multiplier. In: Koc C, Naccache D, Paar C, eds. Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001). Lect Notes Comp Sci, 2001, 2162: 185–201
21 Amberg P, Pinckney N, Harris D. Parallel high-radix Montgomery multipliers. In: 42nd Asilomar Conference on Signals, Systems and Computers, Asilomar, 2008. 772–776
22 Walter C. Montgomery exponentiation needs no final subtractions. Electr Lett, 1999, 35: 1831–1832
23 Wu T, Li S, Liu L. A two-stage pipelined architecture for parallel modular exponentiation. In: International Conference on Information Science and Technology, Wuhan, 2012. 215–218
24 Koç Ç K. Analysis of sliding window techniques for exponentiation. Comp Math Appl, 1995, 30: 17–24
25 Suzuki D. How to maximize the potential of FPGA resources for modular exponentiation. In: International Workshop on Cryptographic Hardware and Embedded Systems (CHES). Lect Notes Comp Sci, 2007, 4727: 272–288
26 Oh J, Moon S. Modular multiplication method. IEEE Proc Comp Digital Tech, 1998, 145: 317–318
27 Su C, Hwang S, Chen P, et al. An improved Montgomery's algorithm for high-speed rsa public-key cryptosystem. IEEE Trans VLSI Syst, 1999, 7: 280–284
28 Senturk A, Gok M. Pipelined large multiplier designs on FPGAs. In: 15th Euromicro Conference on Digital System Design (DSD), 2012. 809–814
29 Dhem J, Joye M, Quisquater J. Normalisation in diminished-radix modulus transformation. Electr Lett, 1997, 33: 1931
30 Shieh M D, Lin W C. Word-based Montgomery modular multiplication algorithm for low-latency scalable architectures. IEEE Trans Comp, 2010, 59: 1145–1151
31 Huang M, Gaj K, El-Ghazawi T. New hardware architecture for Montgomery modular multiplication algorithm. IEEE Trans Comp, 2011, 60: 923–936
32 McIvor C, McLoone M, McCanny J. Modified Montgomery modular multiplication and RSA exponentiation techniques. IEEE Proc Comp Digital Tech, 2004, 151: 402–408
33 Miyamoto A, Homma N, Aoki T, et al. Systematic design of RSA processors based on high-radix Montgomery multipliers. IEEE Trans VLSI Syst, 2011, 19: 1136–1146
34 Koç Ç K, Acar T, Kaliski B S. Analyzing and comparing Montgomery multiplication algorithms. IEEE Micro, 1996, 16: 26–33
35 Blum T, Paar C. High-radix Montgomery modular exponentiation on reconfigurable hardware. IEEE Trans Comp, 2001, 50: 759–764
36 Tang S, Tsui K, Leong P. Modular exponentiation using parallel multipliers. In: Proc. IEEE International Conference on Field-Programmable Technology, Tokyo, 2003. 52–59
37 Nozaki H, Motoyama M, Shimbo A, et al. Implementation of RSA algorithm based on RNS Montgomery modular multiplication. In: Third International Workshop on Cryptographic Hardware and Embedded Systems. Lect Notes Comp Sci, 2001, 2162: 364–376

Appendix A Modular exponentiation algorithm with sliding-window method

Algorithm A1 Sliding window method for modular exponentiation

Require: C, M, E are all n-bit number, $E = \sum_{i=0}^{m-1} E_i$, $E_i \in [0, 2^k)$, $E_{m-1} \neq 0$, $n = k \cdot m$. Precompute $C_{2j+1} = C^{2j+1}$ for $j = 0, 1, \ldots, 2^{k-1} - 1$.

Ensure: $Z = C^E (\mod M)$.

$S := C$;
$h = \lfloor \log_2 E_{m-1} \rfloor + 1$
if $\mathrm{GCD}(E_{m-1}, 2^k) = 2^t$ **then**
  $S := S^{2^{h-t}} \mod M$;
  $S := S \cdot C_{E_{m-1}/2^t} \mod M$;
  if $t \geqslant 1$ **then**
    $S := S^{2^t} \mod M$;
  end if
end if
for $i = m - 2$ downto 0 **do**
  if $E_i = 0$ **then**
    $S := S^{2^k} \mod M$;
  else if $\mathrm{GCD}(E_i, 2^k) = 2^t$ **then**
    $S := S^{2^{k-t}} \mod M$;
    $S := S \cdot C_{E_i/2^t} \mod M$;
    if $t \geqslant 1$ **then**
      $S := S^{2^t} \mod M$;
    end if
  end if
end for
return $Z = S$.

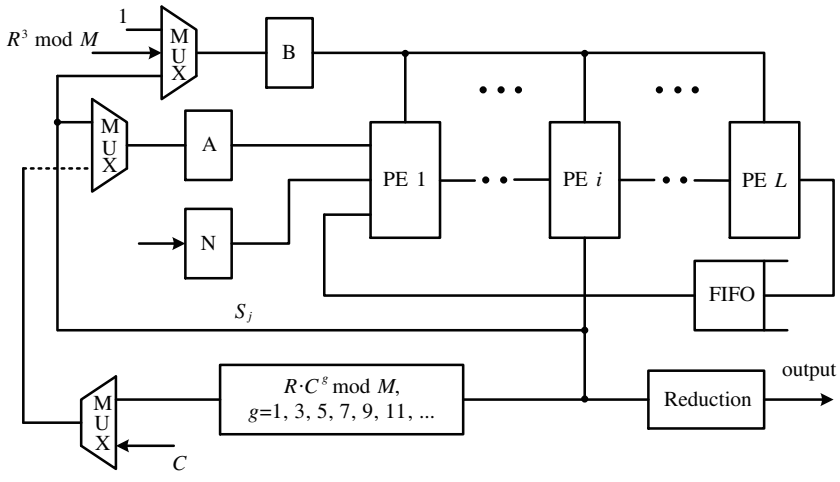

Figure A1 Modular exponentiation by sliding window method and Montgomery modular multiplications.

The system for modular exponentiation is shown in Algorithm A1 and Figure A1, which computes $C^k \mod M$ by sliding window method and HSCMs.

Appendix B Modular reduction of multiprecision integers by approximation

Suppose X is an n_1-bit number, and Y is an n_2-bit number, with $n_1 > n_2$. In this work, $n_1 = 1057$, $n_2 = 1024$. Firstly, we substitute floating numbers for X and Y, i.e., $x = 2^{-n_0} X = x_1 + \delta_1 < 2^{n_1 - n_0}$, $y = 2^{-n_0} Y = y_1 + \delta_2 < 2^{n_2 - n_0}$, where $x_1 = \lfloor x \rfloor$, $y_1 = \lfloor y \rfloor$, $\delta_1 \in [0, 1)$, and $\delta_2 \in [0, 1)$. Furthermore, set $t = n_2 - n_0$, and $n_1 - n_0 = (2t - h)$, with $h \geqslant 3$. For example, we have set $t = 36$, $h = 3$ and $n_0 = 988$ for this work. Figure B1 illustrates the substitution of $x_1 : y_1$ for $x : y$.

Then, set

$$q = \left\lfloor \frac{X}{Y} \right\rfloor = \left\lfloor \frac{x}{y} \right\rfloor = \left\lfloor \frac{x_1 + \delta_1}{y_1 + \delta_2} \right\rfloor, \tag{B1}$$

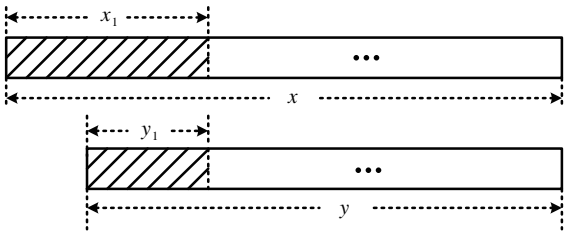

Figure B1 Approximate x/y by truncation.

Algorithm B1 Simple division

Require: $P := x_1 - 1 < 2^t$, $U := y_1 + 1 < 2^{2t-h}$.
Ensure: $Q = \lfloor U/P \rfloor$.
1: $Q = (q_{t-h} \cdots q_1 q_0)_2 = q_{t-h..0} := (0 \cdots 00)_2$, $T := u_{2t-h-1..t-h+1}$;
2: **for** $i = t - h$ downto 0 **do**
3: $C = 2T + u_i$, $C' = C - P$;
4: **if** $C' < 0$ **then**
5: $q_i := 0$, $T := C$;
6: **else**
7: $q_i = 1$, $T := C'$;
8: **end if**
9: **end for**
10: **return** $q_{t-h..0}$.

and

$$q_1 = \left\lfloor \frac{x_1 - 1}{y_1 + 1} \right\rfloor. \tag{B2}$$

According to Eq. (B2), q_1 can be calculated by simple division, as is shown in Algorithm B1. In Algorithm B1, q_i denotes the i-th bit of Q, and they are obtained bit by bit through t-bit subtractions. The expression of $2T + u_i$ can be seemed as a concatenation of u_i after T.

On the one hand, since

$$\left\lfloor \frac{x}{y} \right\rfloor = \left\lfloor \frac{x_1 + \delta_1}{y_1 + \delta_2} \right\rfloor \geqslant \left\lfloor \frac{x_1 - 1}{y_1 + 1} \right\rfloor,$$

there is

$$q \geqslant q_1. \tag{B3}$$

On the other hand, there are $2^{2t-h-1} \leqslant x_1 < 2^{2t-h}$, $2^{t-1} \leqslant y_1 < 2^t$, $h \geqslant 3$, so

$$y_1^2 \geqslant 2^{2t-2} = 2 \cdot 2^{2t-3} \geqslant 2 \cdot 2^{2t-h} > 2x_1. \tag{B4}$$

Meanwhile, there is

$$\begin{aligned} 0 < \Delta &= \frac{x_1 + \delta_1}{y_1 + \delta_2} - \frac{x_1 - 1}{y_1 + 1} \\ &< \frac{x_1 + 1}{y_1} - \frac{x_1 - 1}{y_1 + 1} = \frac{x_1 + 2y_1 + 1}{y_1(y_1 + 1)} \\ &< \frac{y_1^2 + 4y_1 + 2}{2y_1(y_1 + 1)} < \frac{y_1^2 + 4y_1 + 3}{2y_1(y_1 + 1)} = \frac{y_1 + 3}{2y_1} \\ &< 2^{-1} + 2^{2-t} = 2^{-1} + 2^{-34} < 1. \end{aligned} \tag{B5}$$

In the above relation, Eq. (B4) has already been used. Now, we have

$$\begin{aligned} q &= \left\lfloor \frac{x_1 + \delta_1}{y_1 + \delta_2} \right\rfloor = \left\lfloor \frac{x_1 - 1}{y_1 + 1} + \Delta \right\rfloor \\ &\leqslant \left\lfloor \frac{x_1 - 1}{y_1 + 1} + 1 \right\rfloor \leqslant \left\lfloor \frac{x_1 - 1}{y_1 + 1} \right\rfloor + 1 \\ &= q_1 + 1. \end{aligned} \tag{B6}$$

Based on Eq. (B3) and (B6), there is

$$q - 1 \leqslant q_1 \leqslant q. \tag{B7}$$

As a result, $X \mod Y = r \equiv r_1 = X - q_1 \cdot Y(\mod Y)$, with $r_1 = r$ or $r_1 = r + Y$. In order to determine the accurate residue, one should further judge whether r_1 is larger than Y.