FFT-Based McLaughlin's Montgomery Exponentiation without Conditional Selections

Wangchen Dai, *Student Member, IEEE*, Donglong Chen,
Ray C. C. Cheung, *Member, IEEE*, and Çetin Kaya Koç, *Fellow, IEEE*

Abstract—Modular multiplication forms the basis of many cryptographic functions such as RSA, Diffie-Hellman key exchange, and ElGamal encryption. For large RSA moduli, combining the fast Fourier transform (FFT) with McLaughlin's Montgomery modular multiplication (MLM) has been validated to offer cost-effective implementation results. However, the conditional selections in McLaughlin's algorithm are considered to be inefficient and vulnerable to timing attacks, since extra long additions or subtractions may take place and the running time of MLM varies. In this work, we restrict the parameters of MLM by a set of new bounds and present a modified MLM algorithm involving no conditional selection. Compared to the original MLM algorithm, we inhibit extra operations caused by the conditional selections and accomplish constant running time for modular multiplications with different inputs. As a result, we improve both area-time efficiency and security against timing attacks. Based on the proposed algorithm, efficient FFT-based modular multiplication and exponentiation are derived. Exponentiation architectures with dual FFT-based multipliers are designed obtaining area-latency efficient solutions. The results show that our work offers a better efficiency compared to the state-of-the-art works from and above 2048-bit operand sizes. For single FFT-based modular multiplication, we have achieved constant running time and obtained area-latency efficiency improvements up to 24.3 percent for 1,024-bit and 35.5 percent for 4,096-bit operands, respectively.

Index Terms—Montgomery modular multiplication, modular exponentiation, RSA encryption, number-theoretic weighted transform, field-programmable gate array (FPGA)

1 Introduction

Modular multiplication is the core operation of modular exponentiation based cryptosystems. These include the RSA algorithm [1], the Diffie-Hellman key exchange scheme [2], and many other cryptographic functions. The security of RSA relies on the difficulty of finding the two prime factors p and q of modulus n. Due to the advances in factorization methods, the commonly used modulus size is at least 1,024-bit for a secure RSA. Moreover, as recommended by the NIST [3], 3,072-bit or even larger modulus would be used in the near future for long-term protection. While larger modulus renders higher security, it also increases the processing time and consumes more hardware resources. Therefore, high-performance modular exponentiation processor for large operands is in demand for RSA or other cryptosystems.

Modular exponentiation is usually performed by the binary (square-and-multiply) method [4], which scans one exponent bit every iteration either from left to right (L2R) or from right to left (R2L). For a 1,000-bit exponent, around 1,500 multiplications would be performed on average. Several techniques have been studied to reduce the number of multiplications by scanning multiple bits each time, such as the m-ary method and the constant length sliding-windows (CLSW) method [5]. However, the cost of such savings is high [6]. Specifically, compared to the 1,024-bit binary method, m-ary saves 19 percent of the multiplications but needs 5 times of the hardware resources on FPGA; meanwhile, CLSW saves 22 percent of the multiplications but needs 4 times of the hardware resources. For larger operands, the asymptotic value of savings offered by m-ary is 33 percent, and CLSW is 35 percent [4]. As a result, choosing the binary method would be better for a cost-effective exponentiation architecture, as both m-ary and CLSW are costly in terms of hardware resources.

The efficiency of modular multiplication has a direct impact on the performance of modular exponentiation. For the modular multiplications by using the regular methods, such as the schoolbook method, the Karatsuba method [7], and the Toom-Cook method [8], either area or latency will become unacceptable when the operand size reaches thousands of bits [6], [9]. On the other hand, combining the Montgomery modular multiplication (MMM) [10] with the fast Fourier transform (FFT) method [11], [12], [13], [14] offers an efficient solution for large operand sizes (3,072-bit and above) because of its low complexity [15], [16], [17].

The benefits of applying FFT method to McLaughlin's Montgomery modular multiplication [18] have been explored in [17]. Compared to other FFT-based approaches [15], [16], the work in [17] reduces the FFT length by half

- *W. Dai and R.C.C. Cheung are with the Department of Electronic Engineering, City University of Hong Kong, Kowloon Tong, Hong Kong, China. E-mail: w.dai@my.cityu.edu.hk, ccheung@ieee.org.*
- *D. Chen is with the Tencent Technology (Shenzhen) Co., Ltd., Shenzhen 518057, China. E-mail: donglongchen@hotmail.com.*
- *Ç.K. Koç is with İstinye University, Nanjing University of Aeronautics and Astronautics, and University of California Santa Barbara, Santa Barbara, CA 93106. E-mail: cetinkoc@ucsb.edu.*

Manuscript received 10 Aug. 2017; revised 24 Feb. 2018; accepted 27 Feb. 2018. Date of publication 5 Mar. 2018; date of current version 15 Aug. 2018. (Corresponding author: Donglong Chen.)
Recommended for acceptance by W. Liu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2018.2811466

and improves area-time efficiency due to the avoidance of zero-padding [19], [20]. However, the conditional selections in MLM are considered to be inefficient. First, they must be performed in sequence due to the data dependency, and thus parallel computation for higher throughputs is unavailable. Second, extra hardware resources are needed as complex control logic is employed. Besides, every conditional selection involves an extra large size addition or subtraction, performing such operation is costly due to the long carry propagation. In addition to affecting the efficiency, the conditional selections may also raise security concerns. Studies in [21] and [22] reveal that the conditional selection in MMM [10] may be vulnerable against timing attacks. Since the conditional selection occurs with different possibilities for squares and multiplications [23], the secret key of RSA can be deduced after collecting enough timing observations [23], [24]. Similar timing attacks may be mounted against MLM as it also involves conditional selections. Therefore, for both efficiency and security concerns, the MLM algorithm needs to be further improved.

This paper studies and investigates the conditional selection issue in MLM. We propose an improved algorithm, named as McLaughlin's multiplication without conditional selections (MLWS), which eliminates all the conditional selections by enlarging the input or output (I/O) bound. As it performs no extra addition or subtraction and runs at a constant time, both area-time efficiency and security against timing attacks [23], [25] are improved. A secondary objective of this paper is to create implementations of modular exponentiation with high area-time efficiency, rather than to create very low area or ultra high-speed implementations at the high cost of the other. The contributions of this paper are listed as follows:

- MLM without conditional selections (MLWS) is proposed, which eliminates all conditional selections of MLM and achieves improvements on both area-time efficiency and security against timing attacks.
- Combining MLWS with the FFT method, we proposed the FFT-base MLWS (FMLM). With carry-save technique and computing data-flow optimization, FMLM reduces the number of long additions and subtractions from 10 to 3.
- An efficient FFT-based modular exponentiation (FMLE) algorithm is derived by combining FMLM with the binary exponentiation method.
- Pipelined architecture with dual FFT-based multipliers is designed for FMLE. We also equip the architecture with protection against simple power analysis. The FPGA implementation results show that better area-time efficiency is achieved for 2,048-bit and above operand sizes.
- At last, we show that the number of long additions and subtractions of MLWS can be reduced to only 1 if an even larger I/O bound is allowed.

The rest part of this paper is organized as follows. Section 2 provides the mathematical backgrounds. Section 3 proposes the MLWS algorithm. Section 4 presents the FMLM algorithm and its algorithm level improvements. Section 5 provides the FMLE algorithm and its parameter specifications. Section 6 describes the pipelined architectures of FMLE and FMLM in detail. Section 7 provides the FPGA implementation results and comparisons with other works. Section 8 discusses a method to further reduce the number of long additions in MLWS. Section 9 summarizes the remarks.

TABLE 1
Abbreviation and Notation List

Abbr. and Notation	Description
FFT	fast Fourier transform
NWT	number-theoretic weighted transform
CT	cyclic transform
NCT	nega-cyclic transform
MMM	Montgomery modular multiplication
MLM	McLaughlin's Montgomery modular multiplication
MLWS	McLaughlin's multiplication without conditional selections
FMLM	FFT-based McLaughlin's multiplication
FMLE	FFT-based McLaughlin's exponentiation
I/O	input or output
R2L	right to left
PEA/B	processing element A / B
SPA	simple power analysis
n	modulus of RSA
r, h	moduli of McLaughlin's MMM
l	bit length of r
b	radix for long integer decomposition
s	transform length / number of digits
u	bit length of b
v	number of FFT stages
ω	primitive sth root of unity
ϕ	primitive $2s$th root of unity
q	ring size of NWT
c	integer to generate q
$\boldsymbol{d}$	$\{d_{s-1}, \ldots, d_1, d_0\}$ weight digit sequence of NWT
e	exponent
τ	bit length of exponent e

2 Preliminaries

We use lower case letters $x, y \ldots$ to denote time domain variables, upper case letters $X, Y \ldots$ for spectral domain variables, and boldface letters $\boldsymbol{x}, \boldsymbol{X} \ldots$ for integer sequences in time or spectral domain. For the ease of reference, notations and abbreviations in this paper, as well as their descriptions, are listed in Table 1.

2.1 Number-Theoretic Weighted Transform

For a non-negative integer x, its base-b representation can be defined as: $x = \sum_{i=0}^{s-1} x_i b^i$, where $0 \leqslant x_i \leqslant b-1$. The collection of digits x_i is denoted as $\boldsymbol{x} = \{x_i : 0 \leqslant i \leqslant s-1\}$. Evaluating the polynomial $x(t) = x_{s-1}t^{s-1} + x_{s-2}t^{s-2} + \cdots + x_1 t + x_0$ at $t = b$ yields the standard representation of $\boldsymbol{x}$. We usually choose b as a power of 2, i.e., $b = 2^u$, for the ease of computation.

Assume $x, y < b^s - 1$ are in base-b representation, then $z = xy \mod (b^s - 1)$ is equivalent to a length-s *cyclic convolution* of $\boldsymbol{x}$ and $\boldsymbol{y}$, denoted as $(\boldsymbol{x} *_s \boldsymbol{y})$ [26]:

$$\begin{aligned} z &= \sum_{k=0}^{2s-1} \left(\sum_{i+j=k} x_i y_j \right) b^k \bmod (b^s - 1) \\ &= \sum_{k=0}^{s-1} \left(\sum_{i+j=k} x_i y_j \right) b^k + \sum_{k=0}^{s-1} \left(\sum_{i+j=k+s} x_i y_j \right) b^k \qquad (1) \\ &:= \sum_{k=0}^{s-1} (\boldsymbol{x} *_s \boldsymbol{y})_k b^k. \end{aligned}$$

In terms of modulus b^s+1, $z = xy \bmod (b^s+1)$ is equivalent to a length-s *nega-cyclic convolution* of $\boldsymbol{x}$ and $\boldsymbol{y}$, denoted as $(\boldsymbol{x} \bullet_s \boldsymbol{y})$ [26]:

$$\begin{aligned} z &= \sum_{k=0}^{2s-1} \left(\sum_{i+j=k} x_i y_j \right) b^k \bmod (b^s+1) \\ &= \sum_{k=0}^{s-1} \left(\sum_{i+j=k} x_i y_j \right) b^k - \sum_{k=0}^{s-1} \left(\sum_{i+j=k+s} x_i y_j \right) b^k \\ &:= \sum_{k=0}^{s-1} (\boldsymbol{x} \bullet_s \boldsymbol{y})_k b^k. \end{aligned} \tag{2}$$

Similar to the frequency domain in signal processing literature, the number-theoretic transform (NTT) provides the *spectral domain* where cyclic convolution is computed point-wisely [27]. Crandall and Fagin [26] modified this transform to support nega-cyclic convolution by introducing an extra weight digit sequence comprised of s non-zero integers:

$$\boldsymbol{d} = \{d_i : i = 0, 1, 2, \ldots, s-1\}. \tag{3}$$

This modified NTT is named as number-theoretic weighted transform (NWT). Let $\boldsymbol{X}$ be the NWT of $\boldsymbol{x}$, the forward and inverse NWTs over ring $\mathbb{Z}_q$ are defined as:

$$\begin{aligned} X_k &= \mathrm{NWT}(\boldsymbol{x})_k := \sum_{i=0}^{s-1} d_i x_i \omega^{ik} \pmod q, \\ x_k &= \mathrm{INWT}(\boldsymbol{X})_k := (d_k s)^{-1} \sum_{i=0}^{s-1} X_i \omega^{-ik} \pmod q, \end{aligned} \tag{4}$$

where $k = 0, 1, \ldots, s-1$, ω is the primitive sth root of unity in $\mathbb{Z}_q$ ($\omega^s \equiv 1 \bmod q$).

An NWT becomes precisely the NTT in the case when $\boldsymbol{d} = \boldsymbol{1}$ [26]. We rename such NWT as *cyclic transform* (CT). Let $\boldsymbol{X} = \mathrm{CT}(\boldsymbol{x})$, $\boldsymbol{x} = \mathrm{ICT}(\boldsymbol{X})$, and $\odot$ be the point-wise multiplication, then cyclic convolution can be computed as:

$$(\boldsymbol{x} *_s \boldsymbol{y}) = \mathrm{ICT}[\mathrm{CT}(\boldsymbol{x}) \odot \mathrm{CT}(\boldsymbol{y})] \pmod q. \tag{5}$$

In terms of nega-cyclic convolution, we define another special case of NWT as *nega-cyclic transform* (NCT) by setting $\boldsymbol{d} = \{d_i = \phi^i\}$, where ϕ is the primitive $2s$th root of unity ($\phi^{2s} \equiv 1 \bmod q$) [26]. Let $\hat{\boldsymbol{X}} = \mathrm{NCT}(\boldsymbol{x})$ and $\boldsymbol{x} = \mathrm{INCT}(\hat{\boldsymbol{X}})$, then nega-cyclic convolution can be computed as:

$$\begin{aligned} (\boldsymbol{x} \bullet_s \boldsymbol{y}) &= \boldsymbol{d}^{-1} \odot [(\boldsymbol{d} \odot \boldsymbol{x}) *_s (\boldsymbol{d} \odot \boldsymbol{y})] \\ &= \mathrm{INCT}[\mathrm{NCT}(\boldsymbol{x}) \odot \mathrm{NCT}(\boldsymbol{y})] \pmod q. \end{aligned} \tag{6}$$

Note that $\mathbb{Z}_q$ supports an length-s convolution if and only if $s|(q_i - 1)$ for every prime factor q_i of q [26], [27]. In addition, to avoid data overflow, the ring size q must be greater than the largest possible component of the convolutions, i.e., $q > s(b-1)^2$.

When s is a power of 2, i.e., $s = 2^v$, the radix-2 fast Fourier transform [28] can be applied to both CT and NCT [26], which has the advantage of reducing the digit-level complexity from $\mathcal{O}(s^2)$ to $\mathcal{O}(s \log s)$. Therefore, by invoking the radix-2 FFT algorithm to Equations (5) and (6), cyclic convolution and nega-cyclic convolution could be further accelerated as a lower computational complexity is achieved.

2.2 Right-to-Left Binary Method

In this paper, we implement the right-to-left binary method for modular exponentiation. The R2L binary method is described in Algorithm 1, where a squaring is performed in each iteration, and depending on the scanned bit, a multiplication is performed. The maximum number of modular multiplications required by the algorithm is found to be 2τ in the worst-case when $e = (111\ldots11)_2$. While in the average-case, the number is found to be $3\tau/2$. For fast modular reduction, MMM or MLM algorithm can be applied to perform the modular multiplication steps, i.e., Steps 4 and 5 in Algorithm 1.

Algorithm 1. Right-to-Left Binary Method for Modular Exponentiation [4]

Input: x, n, $e = (e[\tau-1]e[\tau-2]\cdots e[1]e[0])_2$
Output: $t = x^e \pmod n$
1: $y \leftarrow x$
2: $t \leftarrow 1$
3: **for** $i = 0$ to $\tau - 1$ **do**
4: **if** $e[i] = 1$ **then** $t \leftarrow ty \pmod n$
5: $y \leftarrow y^2 \pmod n$
6: **end for**
7: **return** t

2.3 McLaughlin's Montgomery Modular Multiplication

Montgomery modular multiplication [10] is particularly efficient for modular exponentiation. But it has the zero-padding issue when adopting the FFT method [16]. Different from the MMM algorithm, McLaughlin's Montgomery modular multiplication (MLM) [18] redefines $r = 2^l - 1$ with an additional modulus $h = 2^l + 1$, cf. Algorithm 2. Besides, by carefully selecting the parameters, e.g., let $2^l = b^s$, the modular multiplication steps in MLM can be computed by the FFT method without zero-padding [17]. Thus, a lower computational complexity is achieved compared to the FFT-based MMM [16]. As a trade-off, MLM involves more l-bit additions and subtractions than MMM because of the conditional selections and the special forms of r and h. When l is sufficiently large, these l-bit operations are non-trivial due to their long carry propagations.

Algorithm 2. McLaughlin's Montgomery Modular Multiplication (MLM) [18]

Input: Let $x \equiv x'r \bmod n$, $y \equiv y'r \bmod n$, both $x, y < n$. Choose $r = 2^l - 1 > n$ and $h = 2^l + 1$, verify $\gcd(r, n) = 1$ and $\gcd(r, 2h) = 1$, $n' = -n^{-1} \bmod r$
Output: $t = xyr^{-1} \pmod n$
1: $m \leftarrow xyn' \pmod r$
2: $g \leftarrow (xy + mn) \pmod h$
3: $f \leftarrow -g \pmod h$
4: **if** $2|f$ **then** $w \leftarrow f/2$, **else** $w \leftarrow (f+h)/2$
5: **if** $xy + mn \equiv w \pmod 2$ **then** $t \leftarrow w$, **else** $t \leftarrow w + h$
6: **if** $t < n$ **then return** t, **else return** $t - n$

2.4 Timing Analysis on McLaughlin's Multiplication

Algorithm 2 consists of three extra operations, namely, $f + h$ in Step 4, $w + h$ in Step 5, and $t - n$ in Step 6. Depending on

the input values, the need for each extra operation is different, which results in a variable running time of MLM.

Meanwhile, there exist dependencies among the three extra operations. Given $0 \leqslant f < h$ in Step 4 of Algorithm 2, we would have $0 \leqslant w < h/2$ when $2|f$, otherwise $h/2 \leqslant w < h$. It is a common case to use a large modulus n for modular multiplication, so we assume $h/2 < n < r$. Thus, $t = w < h/2 < n$ is ensured ($t - n$ is not needed) when both additions, $f + h$ in Step 4 and $w + h$ in Step 5, are not computed. Besides, if $w + h$ is performed, $t - n$ is needed since $t = w + h > n$. Base on the above observations, let T_{con} be the time spent on each extra operation, the timing measurements of MLM can be divided into two subsets depending on the relationship between t and n:

- If $t \geqslant n$ and $t - n$ is needed, at least one of the two extra operations $f + h$ and $w + h$ must be computed. The timing difference is $2T_{con}$ or $3T_{con}$ compared to the case when no extra operation is computed.
- If $t < n$ and $t - n$ is not needed, $w + h$ is unnecessary. The timing difference is 0 or T_{con}.

Therefore, similar to MMM, timing attack on MLM also relies on the relationship between t and n.

Though different methods are employed, the basic concepts of MMM and MLM are the same: both algorithms are based on the equality of $rt = xy + mn$ [10], [18]. Thus, the need for $t - n$ in MLM is also different for square and multiplication, and it can be evaluated by the probability estimation methods of MMM [23], [24]. As a result, when using MLM for modular exponentiation, similar timing attack methods proposed in [22], [23], [24] can be mounted for recovering the exponent bits.

There are simple countermeasures to avoid timing attacks. One of these is to perform every extra operations within each MLM. Thus dummy operations might be produced. However, as MLM involves more conditional selections than MMM, implementing this approach may reduce the performance significantly. In this paper, we proposed a modified MLM algorithm which prevents all the conditional selections, so that each modular multiplication can be performed within constant running time. Due to this fact, the modified algorithm is considered to be secure against timing attacks [21], [23], [24]. Moreover, as fewer long additions or subtractions are required, the area-time efficiency is also improved.

3 McLaughlin's Montgomery Modular Multiplication without Conditional Selections

In MLM algorithm, performing the conditional selections is costly as extra computational efforts are required. Besides, since the running time varies from different inputs, using MLM for RSA computation is considered to be vulnerable to timing attacks. For both efficiency and security concerns, an improved MLM is proposed in this section, which eliminates all the conditional selections.

The original MMM [10] computes

$$t = \frac{xy + mn}{r}. \tag{7}$$

Given $x < 2n$, $y < 2n$, $r > 4n$ and $m = xyn' \bmod r < r$, t is bounded by

$$t < \frac{4n^2 + rn}{r} = \frac{4n^2}{r} + n < 2n. \tag{8}$$

Let $h > r > 4n$ and $\gcd(r, h) = 1$, the equality of Equation (7) still holds when both sides are reduced by h:

$$t = \frac{xy + mn}{r} \pmod{h}. \tag{9}$$

Next, we demonstrate that t in Equation (9) can be solved more efficiently than MLM when x, y and t are bounded by $2n$, and $h = 2^l + 1 > r = 2^l - 1 > 4n$ where $\gcd(r, h) = 1$.

Merging Steps 3 and 4 in Algorithm 2: As can be derived from Equation (9), g in Step 2 of Algorithm 2 equals to $g = rt \pmod{h}$. Because $r \equiv -2 \pmod{h}$, we would obtain

$$2t \equiv -g \pmod{h} = f. \tag{10}$$

Since $h > 4n > 2t$, if $f = g = 0$, the only possible solution is $t = 0$.

Now we prove that $t = 0$ if and only if $xy = 0$. Assume $xy = kn$ where k is a positive integer. Then $n|xy$ if $xy \neq 0$, and $m = xyn' \pmod{r} = r - k$. After substituting xy and m into Equation (9), we have:

$$t = \frac{kn + rn - kn}{r} \pmod{h} = n. \tag{11}$$

Since $t = n < 2n$ is a valid output, only $xy = 0$ results in $t = 0$. In practice, encrypting a zero message using RSA is meaningless as the output is still zero. Therefore, Steps 3 and 4 can be merged by ensuring $xy \neq 0$, and Step 3 is revised to:

if $2|g$ **then** $w \leftarrow h - g/2$, **else** $w \leftarrow (h - g)/2$.

Eliminating the Parity Check of g: Since $xy + mn = rt < 2rn < 2^{2l}$, the bit length of $xy + mn$ is less than $2l$. Then, given $h = 2^l + 1$, computing $g = xy + mn = rt \pmod{h}$ requires two steps. First, we compute

$$g_h = (rt \bmod 2^l) - \left\lfloor \frac{rt}{2^l} \right\rfloor, \tag{12}$$

then add h if $g_h < 0$. From $r = 2^l - 1$, we derive

$$rt = 2^l t - t = (t - 1)2^l + (2^l - t). \tag{13}$$

As we assume $xy \neq 0$, $t > 0$ is ensured, and thus $0 \leqslant t - 1 < 2^l$ and $0 < 2^l - t < 2^l$. Finally, combining Equation (12) with Equation (13), g_h is solved by

$$g_h = 2^l - 2t + 1 = h - 2t. \tag{14}$$

Equation (14) reveals two observations: first, $g = g_h$ as $0 \leqslant g_h < h$ is always true; second, g_h (or g) is always odd. The first observation indicates that g can be obtained by only one subtraction, and the latter indicates the parity check of g can be removed. Consequently, the first conditional selection of MLM is eliminated, and Step 3 is further revised to:

$w \leftarrow (h - g)/2$.

Eliminating Steps 5 and 6 in Algorithm 2: Step 4 in Algorithm 2 removes the scalar 2 of Equation (10), so $t = w \pmod{h}$. As mentioned previously, we have $0 < w = (h - g)/2 < h$ and $0 < t < 2n < h$, so that both w and t are within the range of $[0, h)$. Therefore, $t = w$ and the parity

checks of w and $xy + mn$ can be saved. This conclusion is able to be verified by substituting Equation (14) into $w = (h - g)/2$, and we get $w = (h - g)/2 = (h - h + 2t)/2 = t$. As a result, the second conditional of MLM is eliminated. In addition, as x, y and t are bounded by $2n$, the last conditional selection (Step 6) is also eliminated.

The algorithm of McLaughlin's multiplication without conditional selections (MLWS) is provided as shown in Algorithm 3. Compared to the original MLM algorithm, the efficiency of MLWS is improved as no extra addition or subtraction is ever performed. Besides, MLWS computes modular multiplication with constant running time, which is considered to be one of the countermeasures against timing attacks [23]. Therefore, the security of MLWS is also improved.

Algorithm 3. McLaughlin's Multiplication without Conditional Selections (MLWS)

Input: Let $x \equiv x'r \bmod n$, $0 < x < 2n$, and $y \equiv y'r \bmod n$, $0 < y < 2n$. Choose $r = 2^l - 1 > 4n$ and $h = 2^l + 1 > 4n$. Verify $\gcd(r, n) = 1$ and $\gcd(r, h) = 1$. Compute $n' = -n^{-1} \bmod r$
Output: $t \equiv xyr^{-1} \pmod n$, $t < 2n$
1: $m \leftarrow xyn' \pmod r$
2: $g \leftarrow (xy + mn) \pmod h$
3: $w \leftarrow (h - g)/2$
4: **return** $t \leftarrow w < 2n$

4 FFT-Based Moduluar Multiplication under McLaughlin's Framework

In order to compute the modular multiplication with large operand size efficiently, the FFT method is employed in the modular multiplication steps of MLWS.

4.1 FFT-Based MLWS Algorithm

Let $r = 2^{us} - 1$, $h = 2^{us} + 1$, where $l = us$, u is the digit length, and s is the transform length, the FFT method can be used to compute m and g in Algorithm 3. Depending on the moduli, CT is used to compute m while NCT is used for g. Algorithm 4 presents the FFT-based MLWS (FMLM), which involves 9 length-s FFTs and $4s$ point-wise multiplications. Compared to FFT-based MMM [15] and [16], the transform length in FMLM is reduced by half due to the avoidance of zero-padding. Besides, compared to [17], FMLM involves fewer long additions because the conditional selections in MLM are prevented.

When modular multiplications are required repeatedly, such as modular exponentiation, n and n' would appear more than once. Therefore, we precompute their transformed results $\hat{\boldsymbol{N}}$ and $\boldsymbol{N'}$ in Algorithm 4, and save them for later use.

Observing that modulo-r or h reduction is required after every inverse transform in Steps 1, 3 and 5 of Algorithm 4. Taking Step 1 as an example, since $\boldsymbol{a'} = (\boldsymbol{x} *_s \boldsymbol{n'})$, $a'_k < sb^2 = 2^{2u+v}$, and thus, $\sum_{k=0}^{s-1} a'_k b^k < 2^{l+u+v+1}$. This means the accumulation result may be greater than r, therefore, a modulo-r reduction is performed in Step 2 to adjust the result to be less than r. For the same reason, modular reductions are employed after Steps 3 and 5.

Computing each modulo-r reduction in FMLM requires one l-bit addition and one l-bit subtraction. For instance, we let $a' = \sum_{k=0}^{s-1} a'_k b^k$ in Step 2, so that $a' < 2^{l+u+v+1} < 2^{2l}$. Thus, $a = a' \bmod r$ can be computed within two steps. The first step computes

$$a_r = \left\lfloor \frac{a'}{2^l} \right\rfloor + (a' \bmod 2^l), \tag{15}$$

where a_r equals to either a or $a + r$. Then, the second step obtains a by removing the extra r. Similarly, m in Step 4 can be computed by the same operations. In terms of modulo-h reduction in Step 7, g can be obtained by only one l-bit subtraction according to the discussion in Section 3. As a result, each FMLM involves 6 long additions (subtraction is counted as addition).

Algorithm 4. FFT-Based McLaughlin's Multiplication without Conditional Selections (FMLM)

Input: Let $x \equiv x'r \pmod n$, $0 < x < 2n$, and $y \equiv y'r \pmod n$, $0 < y < 2n$. Choose $r = 2^{us} - 1 > 4n$ and $h = 2^{us} + 1 > 4n$. Verify $\gcd(r, n) = 1$ and $\gcd(r, h) = 1$. Precompute $\boldsymbol{N'} = \mathrm{CT}(\boldsymbol{n'})$, $\hat{\boldsymbol{N}} = \mathrm{NCT}(\boldsymbol{n})$ where $n' = -n^{-1} \pmod r$.
Output: $t = \mathrm{FMLM}(x, y) = xyr^{-1} \pmod n < 2n$
1: $\boldsymbol{a'} \leftarrow \mathrm{ICT}[\mathrm{CT}(\boldsymbol{x}) \odot \boldsymbol{N'}] \pmod q$
2: $a \leftarrow \left(\sum_{k=0}^{s-1} a'_k b^k\right) \pmod r$
3: $\boldsymbol{m'} \leftarrow \mathrm{ICT}[\mathrm{CT}(\boldsymbol{y}) \odot \mathrm{CT}(\boldsymbol{a})] \pmod q$
4: $m \leftarrow \left(\sum_{k=0}^{s-1} m'_k b^k\right) \pmod r$
5: $\boldsymbol{g'} \leftarrow \mathrm{INCT}[\mathrm{NCT}(\boldsymbol{x}) \odot \mathrm{NCT}(\boldsymbol{y}) + \mathrm{NCT}(\boldsymbol{m}) \odot \hat{\boldsymbol{N}}] \pmod q$
6: restrict each g'_k by applying (18)
7: $g \leftarrow \left(\sum_{k=0}^{s-1} g'_k b^k\right) \pmod h$
8: $w \leftarrow (h - g)/2$
9: **return** $t \leftarrow w$

In Step 5, we add xy and mn in spectral domain pointwisely. Therefore, the largest possible value is decided by the sum of two nega-cyclic convolutions. Due to this fact, the ring size of $\mathbb{Z}_q$ should be extended by one more bit to avoid data overflow:

$$q > 2s(b-1)^2. \tag{16}$$

Besides, the purpose of Step 5 is to compute $\boldsymbol{g'} = (\boldsymbol{x} \bullet_s \boldsymbol{y}) + (\boldsymbol{m} \bullet_s \boldsymbol{n})$. Thus, based on Equation (2), g'_k may be a negative integer since it is bounded by

$$-2(s-1-k)(b-1)^2 \leqslant g'_k \leqslant 2(k+1)(b-1)^2. \tag{17}$$

However, for a negative g'_k, $g'_k + q \geqslant 0$ would be obtained instead since the INCT in Step 5 is defined over $\mathbb{Z}_q$. Therefore, a restriction step (Step 6) is performed immediately after Step 5 to recover the bounded value of g'_k:

$$g'_k \leftarrow \begin{cases} g'_k, & \text{if } g'_k \text{ satisfies (17)}, \\ g'_k - q, & \text{otherwise.} \end{cases} \tag{18}$$

FMLM can be simplified when performing squaring or common-multiplicand multiplication. Feeding both inputs of Algorithm 4 with the same value yields the squaring variation of FMLM, where 2 FFTs are saved. For common-multiplicand multiplication, assume x is the common-multiplicand which appears in more than one modular multiplication, we can compute $\boldsymbol{A}$ ($a = xn' \bmod r$) and $\hat{\boldsymbol{X}}$ and save them for subsequent operations. Thus, in common-multiplicand multiplication variation of FMLM, Steps 1 and 2 in Algorithm 4 are removed, Step 3 is revised to

TABLE 2
Number of Operations in Different FFT-Based Algorithms

Algorithm	FFT (length)	Point-wise multiplication	Point-wise addition	Long addition
[16]	$7\,(2s)$	$3s$	$2s$	0
[17]	$9\,(s)$	$4s$	s	10
FMLM	$9\,(s)$	$4s$	s	3
FMLM (Squaring)	$7\,(s)$	$4s$	s	3
FMLM (Com.-Multi.)	$5\,(s)$	$3s$	s	3
FMLM*	$7\,(s)$	$4s$	s	3
[15]	$9\tau+9\,(2s)$	$9\tau s+6s$	0	$3\tau+2$
FMLE	$9.5\tau+10\,(s)$	$5.5\tau s+6s$	$1.5\tau s+2s$	$4.5\tau+6$
FMLE*	$7.5\tau+10\,(s)$	$5.5\tau s+6s$	$1.5\tau s+2s$	$4.5\tau+6$

* *The all-at-once technique is applied to reduce the number of FFTs; Com.-Multi. refers to common-multiplicand multiplication.*

$$\boldsymbol{m'} \leftarrow \mathrm{ICT}[\mathrm{CT}(\boldsymbol{y}) \odot \boldsymbol{A}] \pmod{q}, \tag{19}$$

and Step 5 is revised to

$$\boldsymbol{g'} \leftarrow \mathrm{INCT}[\hat{\boldsymbol{X}} \odot \mathrm{NCT}(\boldsymbol{y}) + \mathrm{NCT}(\boldsymbol{m}) \odot \hat{\boldsymbol{N}}] \pmod{q}. \tag{20}$$

As a result, 4 FFTs and s point-wise multiplications are saved for common-multiplicand multiplication.

4.2 Further Optimizations for FMLM

Employing the carry-save technique in the modulo-r reduction (Step 2 of Algorithm 4) can save one l-bit addition and one l-bit subtraction. The two operations are replaced by one $(u+v+2)$-bit addition, which contains a much shorter carry chain compared to the previous ones, cf. Section 5.1 in [16] or Section 4.1 in [17]. However, the correct value of $m = m' \pmod{r}$ must be obtained during the modulo-r reduction in Step 4, since a different modulus h is involved in the subsequent steps.

Optimizing the computing data-flow of Steps 7 and 8 can save one more l-bit subtraction. Combining the computation of Step 7 with Step 8 we have

$$\frac{h-g}{2} = \frac{1}{2}\left[h + \left\lfloor \frac{g'}{2^l} \right\rfloor - (g' \bmod 2^l)\right]. \tag{21}$$

According to Equation (17), g' is bounded by $2^{l+u+v+2}$. Thus, $\lfloor g'/2^l \rfloor < 2^{u+v+2} \ll 2^l$. Given that $h = 2^l + 1 = (10000\ldots000001)_2$, the computation of Steps 7 and 8 is then simplified to one $(u+v+3)$-bit addition $(1 + \lfloor g'/2^l \rfloor)$, one l-bit subtraction and one bitwise shift. Based on the above optimizations, the number of long additions is further reduced from 6 to 3.

In addition, the number of FFTs can also be reduced. Computing $m = xyn' \bmod r$ sequentially requires 5 FFTs, cf. Steps 1-4 in Algorithm 4. While multiplying the three operands "all-at-once" requires only 3 FFTs:

$$\boldsymbol{m'} \leftarrow \mathrm{ICT}[\mathrm{CT}(\boldsymbol{x}) \odot \mathrm{CT}(\boldsymbol{y}) \odot \boldsymbol{N'}] \pmod{q}. \tag{22}$$

As a trade-off, a larger q is required in order to maintain the dynamic bound:

$$q > s^2(b-1)^3. \tag{23}$$

The number of operations in FMLM and other FFT-based approaches are compared in Table 2, where the long subtractions are counted as long additions. Note that a length-$2s$ FFT involves $s\log(2s)$ modulo-q multiplications. This number is reduced by more than 50 percent in a length-s FFT, which involves $\frac{s}{2}\log s$ modulo-q multiplications.

5 FFT-Based Moduluar Exponentiation under McLaughlin's Framework

Combining the right-to-left binary method with FMLM provides an efficient computation of modular exponentiation with large operand size.

5.1 FFT-Based Modular Exponentiation

Algorithm 5 presents the FFT-based McLaughlin's exponentiation (FMLE), which is based on R2L binary modular exponentiation, cf. Algorithm 1. In Steps 10 and 11 of Algorithm 5, the two FMLMs are performed simultaneously. Specifically, Step 10 computes the common-multiplicand multiplication variation of FMLM, which shares a common-multiplicand y with Step 11, while Step 11 performs the squaring variation of FMLM.

Algorithm 5. FFT-Based McLaughlin's Exponentiation (FMLE)

Input: $e = (e[\tau-1]e[\tau-2]\cdots e[1]e[0])_2$, $0 < x < n$, $r = 2^{us}-1 > 4n$ and $h = 2^{us}+1 > 4n$. Ensure $\gcd(r,n) = 1$ and $\gcd(r,h) = 1$. Find integer $n' = -n^{-1} \bmod r$.
Output: $t = \mathrm{FMLE}(x,e) = x^e \pmod{n}$
/* *Precomputed variables* */
1: $r_0 \leftarrow r \bmod n$
2: $r_1 \leftarrow r^2 \bmod n$
3: $r_2 \leftarrow r_1 n' \bmod r$
4: $\hat{\boldsymbol{R}}_{\boldsymbol{1}} \leftarrow \mathrm{NCT}(\boldsymbol{r_1})$, $\boldsymbol{R_2} \leftarrow \mathrm{CT}(\boldsymbol{r_2})$
5: $\hat{\boldsymbol{N}} \leftarrow \mathrm{NCT}(\boldsymbol{n})$, $\boldsymbol{N'} \leftarrow \mathrm{CT}(\boldsymbol{n'})$
/* *Main steps* */
6: $y \leftarrow x$
7: $t \leftarrow r_0$
8: $y \leftarrow \mathrm{FMLM}(y, r_1)$
9: **for** $i = 0$ to $\tau - 1$ **do**
10: **if** $e[i] = 1$ **then** $t \leftarrow \mathrm{FMLM}(t, y)$
11: $y \leftarrow \mathrm{FMLM}(y, y)$
12: **end for**
13: $t \leftarrow \mathrm{FMLM}(t, 1)$
14: **return** t

Since the essence of FMLM is the Montgomery modular multiplication [10], the input x of Algorithm 5 should be converted into Montgomery form before the iterations. Step 8 performs the conversion $y \equiv xr \equiv x \cdot (r^2 \bmod n) \cdot r^{-1} \pmod{n}$, where $y < 2n$. Then in Step 13, after the iterations, the result is converted out of Montgomery form by performing $t \cdot 1 \cdot r^{-1} \pmod{n}$. Note that $\mathrm{FMLM}(t,1) = [t + (tn' \bmod r)n]/r \leqslant (2n-1+rn-n)/r \leqslant n$. The case when $\mathrm{FMLM}(t,1) = n$ is excluded due to the parameter restrictions of modular exponentiation in RSA [29], therefore, FMLE needs no final conditional subtraction. Besides, the CT and NCT of 1 cost no extra computational efforts, since $1 = (000\ldots001)_b$ and $\mathrm{CT}(1) = \mathrm{NCT}(1) = \mathbf{1}$.

We also include the number of operations required by different FFT-based exponentiation algorithms in Table 2. Since the average-case of exponentiation is considered, the Hamming weight of the exponent is assumed to be $\tau/2$.

TABLE 3
Eligible Parameter Sets for Different Key Sizes

$\lceil \log_2 n \rceil$	l	u	$s = 2^v$	c	q	ω	ϕ
1,024	1,056	33	$32 = 2^5$	3	$2^{96}+1$	64	8
1,024*	1,088	17	$64 = 2^6$	1	$2^{64}+1$	4	2
2,048*	2,112	33	$64 = 2^6$	2	$2^{128}+1$	16	4
2,048	2,176	17	$128 = 2^7$	0.5	$2^{64}+1$	2	$\sqrt{2}$
3,072	3,136	49	$64 = 2^6$	2	$2^{128}+1$	16	4
3,072	3,200	25	$128 = 2^7$	0.5	$2^{64}+1$	2	$\sqrt{2}$
4,096	4,160	65	$64 = 2^6$	3	$2^{192}+1$	64	8
4,096*	4,224	33	$128 = 2^7$	1	$2^{128}+1$	4	2
7,680	7,744	121	$64 = 2^6$	4	$2^{256}+1$	256	16

* *The same (u, v, c) also support the all-at-once technique.*

Therefore, every FMLE involves $\tau/2 + 2$ common-multiplicand multiplications and τ squares.

5.2 Parameter Specification

Selecting appropriate parameters would accelerate the computation of FMLM, and therefore, improve the performance of FMLE. For example, we choose $b = 2^u$ for fast base-b decomposition, choose $s = 2^v$ for applying the radix-2 FFT algorithm, and define $r = 2^{us} - 1$ and $h = 2^{us} + 1$ with $l = us$ for invoking the FFT method to compute modulo-r and h multiplications in MLWS. To further accelerate the computation, other parameters also need to be selected carefully.

Table 2 indicates that FFT is the most compute-intensive operation in FMLE. Since FFT is defined over $\mathbb{Z}_q$, every operation during the transform is reduced by q. Besides, the point-wise multiplications performed in spectral domain are also reduced by q. Therefore, a well-selected q would benefit the performance of FMLE a lot. In this work, we selected q in the form of

$$q = 2^{cs} + 1, \tag{24}$$

where $c > 0$ is an integer [30], and q is either a Fermat number or a pseudo-Fermat number depending on c.

The selection of q has following advantages:

- Fast modular reduction can be employed during FFT computation, where each modulo-q is computed by one subtraction [16];
- We can define $\omega = 2^{2c}$ and $\phi = 2^c$ for length-s FFT, since 2 is the $2cs$th primitive root of unity ($2^{2cs} \equiv 1 \mod q$). Thus, multiplying a power of ω or ϕ is simply a bitwise shift [27];
- The supported transform length of each q is flexible, i.e., choosing a small c results in large transform length, and vice versa.

Combining Equation (24) with Equation (16) derives $c \geqslant (v + 2u + 1)/s$, when employing the all-at-once technique, $c \geqslant (2v + 3u)/s$. In practice, we usually choose the smallest possible value of c in order to minimize ring size q. In addition, for the special case when $c = 1/2$, there exists an eligible expression of $\phi = \sqrt{2}$ in $\mathbb{Z}_q$ [27]:

$$\sqrt{2} \equiv 2^{3 \cdot 2^{v-3}} - 2^{2^{v-3}} \pmod{2^{2^{v-1}} + 1}. \tag{25}$$

Thus, $c = 1/2$ is also considered when selecting the parameters because a smaller $q = 2^{s/2} + 1$ may be obtained.

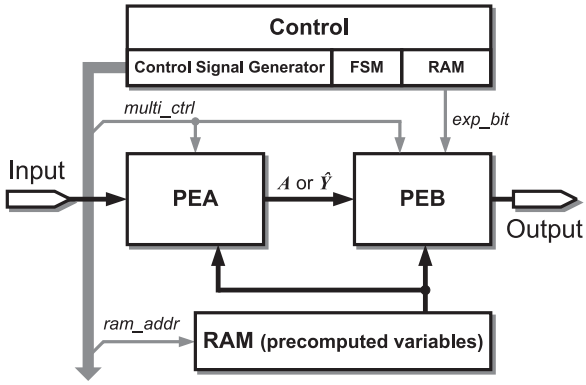


Fig. 1. Top-level architecture of FMLE, A denotes the CT of $a = yn' \pmod r$ and $\hat{Y}$ denotes the NCT of y.

Based on the above discussion, a parameter set selection method for FMLE is summarized as follows:

1) For a given modulus n, its bit-length $\lceil \log_2 n \rceil$ is one of the recommended RSA key size (i.e., 1,024, 2,048, 3,072, ...), set $l = \lceil \log_2 n \rceil + 2$ and $v = 2$;
2) Compute $s = 2^v$, $u = \lceil l/s \rceil$ and $b = 2^u$;
3) Generate $r = 2^{us} - 1$ and $h = 2^{us} + 1$;
4) If $(v + 2u + 1)/s \leqslant 1/2$ then $c = 1/2$, otherwise $c = \lceil (v + 2u + 1)/s \rceil$;
5) If $c \geqslant (2v + 3u)/s$ then apply the all-at-once technique, otherwise apply Algorithm 4 for modular multiplications;
6) Compute $q = 2^{cs} + 1$, $\omega = 2^{2c}$ and $\phi = 2^c$;
7) Record $(l = us, r, h, u, v, q, \omega, \phi)$ as an eligible parameter set for the given n;
8) Increase v by 1 and repeat 2-7 until c first reaches $1/2$.

Table 3 presents some eligible parameter sets generated by the parameter selection method. The marked sets in the table indicate the values of (u, v, s) support both FMLMs with and without the all-at-once technique. For these parameter sets, the number of FFTs is reduced at no cost. Another observation is that the use of pseudo-Fermat numbers slacks off the growth rate of the ring size q. Taking the first parameter set in the table as an example, if q can only be a regular Fermat number, $q = 2^{128} + 1$ instead of $q = 2^{96} + 1$ would be the choice for 1,024-bit moduli, which results in a higher computational complexity.

6 Architecture Design of FMLE

Fig. 1 depicts the top-level architecture of FMLE, which consists of two processing elements (*PEA* and *PEB*), one *Control* unit, and one *RAM* unit. In general, the FMLE computation can be divided into three stages. Stage 1 converts x into Montgomery form, cf. Step 8 in Algorithm 5, stage 2 computes the iterations (Steps 9-12), and stage 3 converts the result out of Montgomery form (Steps 13). As the size of the exponent for RSA computation is usually more than 1,000 bits, the time spent on stage 1 or 3 is trivial when compared to stage 2. To minimize the running time of stage 2, we employ two processing elements (PE) for the FFT-based squaring and FFT-based common-multiplicand multiplication, respectively. Fig. 2 provides the workloads of two PEs during the exponentiation, where *PEA* is responsible for the multiplication in stage 1 and all squares in stage 2, while

Processing Element A	Processing Element B
$y \leftarrow x$	$t \leftarrow r \bmod n$
$y \leftarrow \mathrm{FMLM}(y, r^2 \bmod n)$	/* *IDLE* */
for $i = 0$ **to** $\tau - 1$	
if $(e[i] = 1)$	
$y \leftarrow \mathrm{FMLM}(y, y)$	$t \leftarrow \mathrm{FMLM}(t, y)$
else	
$y \leftarrow \mathrm{FMLM}(y, y)$	/* *IDLE* */
end for	
/* *IDLE* */	$t \leftarrow \mathrm{FMLM}(t, 1)$
return t	

Fig. 2. Parallel computation of FMLE $t \leftarrow x^e \pmod n$.

PEB is responsible for the rest of the multiplications. As a result, the running time T_{exp} of FMLE depends only on the bit length of e, which can be evaluated by

$$T_{exp} = \tau T_{squaring} + 2T_{common-multi}. \tag{26}$$

6.1 Architecture of Processing Element (PE)

Fig. 3 presents the architecture of *PEA*, and Fig. 4 shows the workloads of two PEs within the iteration when $e[i] = 1$. Because the underlying operations involved in FFT-based squaring and FFT-based common-multiplicand multiplication are the same, *PEB* has the same building blocks as *PEA*. On the other hand, different from *PEA*, the RAM module in *PEB* pre-loads r instead of x. Besides, the *Multiply-adder* in *PEB* involves one more input to forward $\boldsymbol{A}$ and $\hat{\boldsymbol{Y}}$ from *PEA* as the precomputed variables of common-multiplicand multiplication, cf. Fig. 4. After some minor modifications, it is possible to switch both PEs between squaring mode and common-multiplicand mode.

6.1.1 FFT Operator and Multiply-Adder

The FFT Operator computes the forward and inverse FFTs, where decimation-in-time FFT is employed. The jth stage equations of FFT for CT are expressed as follows:

$$\begin{cases} x_i^{(j+1)} = x_{2i}^{(j)} + x_{2i+1}^{(j)} 2^{2cJ\cdot\lfloor i/J \rfloor} & (\bmod q) \\ x_{i+s/2}^{(j+1)} = x_{2i}^{(j)} - x_{2i+1}^{(j)} 2^{2cJ\cdot\lfloor i/J \rfloor} & (\bmod q), \end{cases} \tag{27}$$

where $J = 2^{v-j-1}$, $i = 0, 1, \ldots, \frac{s}{2} - 1$ and $j = 0, 1, \ldots, v - 1$. The jth stage equations of FFT for NCT are expressed as

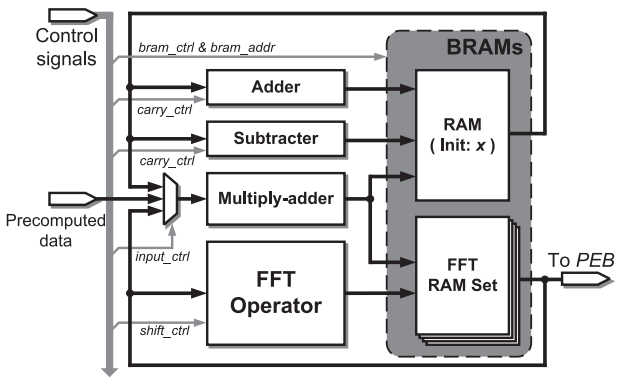


Fig. 3. Architecture of *PEA* module. Different from *PEA*, the RAM in *PEB* pre-loads r, and one more input port is connected to the *Multiply-adder* for the precomputed variables of common-multiplicand multiplication.

Processing Element A	Processing Element B
$\boldsymbol{Y} \leftarrow \mathrm{CT}(\boldsymbol{y})$	$\boldsymbol{T} \leftarrow \mathrm{CT}(\boldsymbol{t})$
$\boldsymbol{A'} \leftarrow \boldsymbol{Y} \odot \boldsymbol{N'}$	/* *IDLE* */
$\boldsymbol{a'} \leftarrow \mathrm{ICT}(\boldsymbol{A'})$	/* *IDLE* */
$a \leftarrow \sum_{k=0}^{s-1} a'_k b^k \bmod r$	/* *IDLE* */
$\boldsymbol{A} \leftarrow \mathrm{CT}(\boldsymbol{a})$	/* *IDLE* */
/* *Forward* $\boldsymbol{A}$ *from PEA to PEB* */	
$\boldsymbol{M'} \leftarrow \boldsymbol{Y} \odot \boldsymbol{A}$	$\boldsymbol{M'} \leftarrow \boldsymbol{T} \odot \boldsymbol{A}$
$\boldsymbol{m'} \leftarrow \mathrm{ICT}(\boldsymbol{M'})$	$\boldsymbol{m'} \leftarrow \mathrm{ICT}(\boldsymbol{M'})$
$m \leftarrow \sum_{k=0}^{s-1} m'_k b^k \bmod r$	$m \leftarrow \sum_{k=0}^{s-1} m'_k b^k \bmod r$
$\hat{\boldsymbol{Y}} \leftarrow \mathrm{NCT}(\boldsymbol{y})$	$\hat{\boldsymbol{T}} \leftarrow \mathrm{NCT}(\boldsymbol{t})$
/* *Forward* $\hat{\boldsymbol{Y}}$ *from PEA to PEB* */	
$\hat{\boldsymbol{Z}}_{\boldsymbol{0}} \leftarrow \hat{\boldsymbol{Y}} \odot \hat{\boldsymbol{Y}}$	$\hat{\boldsymbol{Z}}_{\boldsymbol{0}} \leftarrow \hat{\boldsymbol{T}} \odot \hat{\boldsymbol{Y}}$
$\hat{\boldsymbol{M}} \leftarrow \mathrm{NCT}(\boldsymbol{m})$	$\hat{\boldsymbol{M}} \leftarrow \mathrm{NCT}(\boldsymbol{m})$
$\hat{\boldsymbol{Z}}_{\boldsymbol{1}} \leftarrow \hat{\boldsymbol{M}} \odot \hat{\boldsymbol{N}}$	$\hat{\boldsymbol{Z}}_{\boldsymbol{1}} \leftarrow \hat{\boldsymbol{M}} \odot \hat{\boldsymbol{N}}$
$\boldsymbol{g'} \leftarrow \mathrm{INCT}(\hat{\boldsymbol{Z}}_{\boldsymbol{0}} + \hat{\boldsymbol{Z}}_{\boldsymbol{1}})$	$\boldsymbol{g'} \leftarrow \mathrm{INCT}(\hat{\boldsymbol{Z}}_{\boldsymbol{0}} + \hat{\boldsymbol{Z}}_{\boldsymbol{1}})$
restrict $\boldsymbol{g'}$ by (18)	restrict $\boldsymbol{g'}$ by (18)
$g \leftarrow \sum_{k=0}^{s-1} g'_k b^k \bmod h$	$g \leftarrow \sum_{k=0}^{s-1} g'_k b^k \bmod h$
$y \leftarrow (h - g)/2$	$t \leftarrow (h - g)/2$
return y	**return** t

Fig. 4. Parallel computations of $y \leftarrow \mathrm{FMLM}(y, y)$ and $t \leftarrow \mathrm{FMLM}(t, y)$.

follows:

$$\begin{cases} x_i^{(j+1)} = x_{2i}^{(j)} + x_{2i+1}^{(j)} 2^{2cJ\cdot\lfloor i/J \rfloor + cJ} & (\bmod q) \\ x_{i+s/2}^{(j+1)} = x_{2i}^{(j)} - x_{2i+1}^{(j)} 2^{2cJ\cdot\lfloor i/J \rfloor + cJ} & (\bmod q). \end{cases} \tag{28}$$

Changing $2^{2cJ\cdot\lfloor i/J \rfloor}$ in Equation (27) to $2^{2cs-2cJ\cdot\lfloor i/J \rfloor}$ yields the jth stage equations of IFFT for both ICT and INCT.

The Multiply-adder computes the point-wise multiplication and addition of FMLM. In our case, the operand size of point-wise multiplication is usually a small value (less than 300 bits). Thus, applying the FFT method recursively to computed the point-wise multiplication is inefficient since the time spent on data transfer is non-trivial. On the contrary, choosing one of the regular methods, such as the schoolbook method or the Karatsuba method [7], would be faster and more straightforward [31].

As a result, our PE modules apply the pipelined FFT operator and Karatsuba multiplier provided in [17] for an area-time efficient design. In particular, the pipelined FFT Operator involves two butterfly structures, so that four inputs are operated in parallel every clock cycle. Meanwhile, the Multiply-adder consists of two parts. The first part is a pipelined modulo-q Karatsuba multiplier, which performs the recursive Karatsuba method for the point-wise multiplications, cf. Steps 1, 3, 5 in Algorithm 4. The second part is a subsequent conditional adder, when selected, the point-wise addition in Step 5 would be performed.

6.1.2 The Design of Adder

Performing a long addition either in a single clock cycle or digit-by-digit in s cycles is considered to be inefficient. The first approach reduces the clock frequency due to the long carry chain, while the latter approach requires too many cycles and no other operations can be performed during this period due to the data dependency. Note that the bit length of q is $cs + 1$, where $cs + 1 \geqslant 2u + v + 1 > 2u$. Thus, at least $2u$ bits can be performed every cycle without

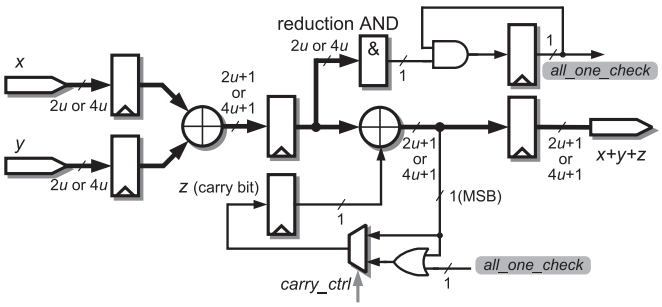


Fig. 5. Architecture of Adder module, MSB refers to the most significant Bit.

affecting the clock frequency of the FFT Operator or the Multiply-adder. As a result, the clock cycle required by each long addition reduces from s to $s/2$. For some parameter sets, the long addition can be performed even faster. Taking the second parameter set in Table 3 as an example, when $u = 17$ and $q = 2^{64} + 1$, the long addition can be added every $4u$-bit (68-bit) per cycle, and thus only $s/4$ cycles are required. In this work, we call this $2u$ or $4u$-bit digit as a *segment*.

The Adder module is designed for the modulo-r reductions of FMLM, cf. Fig. 5. It consists of two cascade adders, the first adder computes $x + y$, and their sum adds the carry bit z in the subsequent adder. As discussed in Section 4.2, performing the modulo-r reduction in Step 2 of Algorithm 4 requires only one clock cycle, since it can be simplified to a $(u + v + 2)$-bit addition and $u + v + 2 < 2u$. The modulo-r reduction in Step 4 requires two operations, we first compute

$$m_r = \left\lfloor \frac{m'}{2^l} \right\rfloor + (m' \bmod 2^l), \tag{29}$$

then remove the extra r by computing

$$m = (m_r \bmod 2^l) + m_r[l], \tag{30}$$

where $l = us$ and $m_r[l]$ is the lth bit of m_r. However, when $m_r = r = (111\ldots1111)_2$ is obtained in Equation (29), because r has l bits and $m_r[l] = 0$, an incorrect result $m = r$ would be delivered by Equation (30). To avoid this incorrect result, an all-one check is employed during the first addition

$$\epsilon = m_r[0] \wedge m_r[1] \wedge m_r[2] \wedge \cdots \wedge m_r[l-1]. \tag{31}$$

Then the correct m is obtained by computing

$$m = [(m_r \bmod 2^l) + (m_r[l] \vee \epsilon)] \bmod 2^l. \tag{32}$$

Equations (29), (31) and (32) are computed segment-by-segment. Depending on the segment size, computing m requires s or $s/2$ clock cycles.

6.1.3 The Design of Subtracter

The Subtracter module is designed for the modulo-h reduction and $(h - g)/2$ in Algorithm 4. The architecture of Subtracter is provided in Fig. 6, which consists of 2 cascade subtracters. Similar with long addition, the long subtraction is also computed segment-by-segment. According to Equation (21), the two operations, $g' \bmod h$ and $(h - g)/2$, can be merged and computed by involving only one long subtraction. Moreover, since $g = h - 2t > h - r = 2$ and

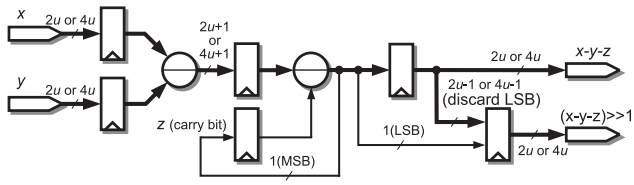


Fig. 6. Architecture of Subtracter module, LSB refers to the least significant bit.

$h - g < 2^l$, computing $h - g$ in binary representation is equivalent to first obtaining

$$1 - g = 1 + \left\lfloor \frac{g'}{2^l} \right\rfloor - (g' \bmod 2^l), \tag{33}$$

then ignoring the lth (or signed) bit. Finally, dropping the least significant bit of $h - g$ yields $t = (h - g)/2$. Note that $1 + \lfloor g'/2^l \rfloor$ can be computed in one clock cycle since $u + v + 3 < 2u$.

6.2 The Control Unit

The *Control* unit consists of three sub-modules: a finite-state machine (FSM), a block RAM, and a control signal generator. FSM consists of four states as shown in Fig. 7. States S0, S1 and S2 are corresponded to the three computing stages of FMLE, respectively. The state transitions are described as follows:

- If signals *rst_n* and *en* equal to 1, IDLE→S0, otherwise IDLE→IDLE;
- If $y = \mathrm{FMLM}(y, r_1)$ is obtained, S0→S1;
- If $i = \tau - 1$ and $t = \mathrm{FMLM}(t, y)$ is obtained, S1→S2, otherwise S1→S1;
- If $t = \mathrm{FMLM}(t, 1)$ is obtained, S2→IDLE.

The block RAM pre-loads the exponent e, and sequentially outputs $e[0], e[1], \ldots e[\tau - 1]$ each time when FSM transits to state S1. The control signal generator controls the two PEs to perform the FMLMs in FMLE by following the computing data-flows as shown in Fig. 4. Moreover, since the computing data-flows of $\mathrm{FMLM}(y, y)$ and $\mathrm{FMLM}(t, y)$ are similar, we can use the same set of control signals to control the two PEs simultaneously. Also, a same control signal generation mechanism is repeatedly used for all the FMLMs in FMLE.

6.3 The RAM Unit in the Top-Level Architecture

Since the squaring and common-multiplicand multiplication use the same parameter set, a *RAM* unit with $(cs + 1)$-bit data width and $5s$ depth is designed to store all the precomputed variables $\boldsymbol{N'}$, $\hat{\boldsymbol{N}}$, $\boldsymbol{B_u}$, $\hat{\boldsymbol{R}}_{\boldsymbol{2}}$ and $\hat{\boldsymbol{R}}_{\boldsymbol{1}}$. In particular, $\boldsymbol{B_u}$ represents the upper bounds of g'_k in Equation (17):

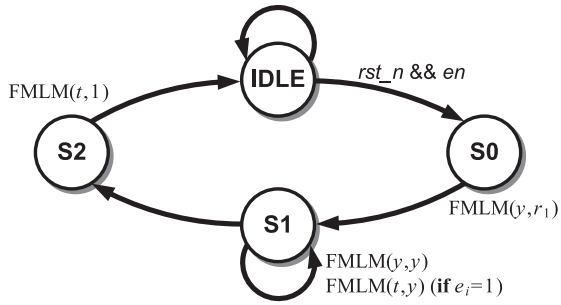


Fig. 7. State transition diagram of the FSM in *Control* module.

Processing Element A	Processing Element B
$y \leftarrow x$	$t \leftarrow r \bmod n$
$y \leftarrow \mathrm{FMLM}(y, r^2 \bmod n)$	$t \leftarrow \mathrm{FMLM}(t, t)$
for $i = \tau - 1$ **downto** 0	
if $(e[i] = 1)$	
$y \leftarrow \mathrm{FMLM}(y, y)$	$t \leftarrow \mathrm{FMLM}(t, y)$
else	
$y \leftarrow \mathrm{FMLM}(y, t)$	$t \leftarrow \mathrm{FMLM}(t, t)$
end for	
$y \leftarrow \mathrm{FMLM}(y, y)$	$t \leftarrow \mathrm{FMLM}(t, 1)$
return t	

Fig. 8. Simple power analysis protected Montgomery powering ladder.

Processing Element A	Processing Element B
$\boldsymbol{Y} \leftarrow \mathrm{CT}(\boldsymbol{y})$	$\boldsymbol{T} \leftarrow \mathrm{CT}(\boldsymbol{t})$
$\boldsymbol{A'} \leftarrow \boldsymbol{Y} \odot \boldsymbol{N'}$	$\boldsymbol{A'} \leftarrow \boldsymbol{T} \odot \boldsymbol{N'}$
$\boldsymbol{a'} \leftarrow \mathrm{ICT}(\boldsymbol{A'})$	$\boldsymbol{a'} \leftarrow \mathrm{ICT}(\boldsymbol{A'})$
$a \leftarrow \sum_{k=0}^{s-1} a'_k b^k \bmod r$	$a \leftarrow \sum_{k=0}^{s-1} a'_k b^k \bmod r$
$\boldsymbol{A} \leftarrow \mathrm{CT}(\boldsymbol{a})$	$\boldsymbol{A} \leftarrow \mathrm{CT}(\boldsymbol{a})$
/ Forward $\boldsymbol{Y}$ from PEA to PEB */*	
$\boldsymbol{M'} \leftarrow \boldsymbol{Y} \odot \boldsymbol{A}$	$\boldsymbol{M'} \leftarrow \boldsymbol{Y} \odot \boldsymbol{A}$
.....	.....
$y \leftarrow (h - g)/2$	$t \leftarrow (h - g)/2$
return y	**return** t

Fig. 9. Simple power analysis protected parallel computation of $y \leftarrow \mathrm{FMLM}(y, y)$ and $t \leftarrow \mathrm{FMLM}(t, y)$.

$$\mathbf{B_u} = \{2(k+1)(b-1)^2 : k = 0, 1, 2 \ldots s-1\}. \quad (34)$$

The result of INCT is a collection of non-negative integers, but the lower bounds in Equation (17) satisfy $-2(s-1-k)(b-1)^2 \leqslant 0$. Therefore, it is unnecessary to precompute and store the lower bounds.

6.4 FMLE Architecture with Simple Power Analysis Protection

The two PEs behave differently during FMLE computation. In particular, Fig. 2 indicates that *PEA* works all the time while *PEB* only works when $e[i] = 1$. Moreover, Fig. 4 shows that after *PEB* obtains $\boldsymbol{T}$, it must wait until *PEA* obtains $\boldsymbol{A}$ in each iteration when $e[i] = 1$. Due to these two facts, simple power analysis (SPA) can be mounted to retrieve the secret key. To this end, we should make the two PEs behave more regularly by either employing dummy operations into the R2L binary method or implementing the Montgomery powering ladder algorithm [32]. Also, the common-multiplicand multiplication method should be modified in order to remove all the idle states in Fig. 4.

In this paper, we implement Montgomery powering ladder for SPA protection. Fig. 8 presents the computing dataflow of Montgomery powering ladder [32]. The first FMLM in *PEB* and the last FMLM in *PEA* are dummy operations, note that $\mathrm{FMLM}(r, r) = r$. Thus, one squaring and one multiplication are preformed constantly which is highly regular. When $e[i] = 1$, we forward $\boldsymbol{Y}$ instead of $\boldsymbol{A}$ from *PEA* to *PEB* so that the operations performed in both PEs are synchronized, cf. Fig. 9. This minor change may increase the computational complexity as more FFTs are introduced. However, from the hardware point of view, this change costs no extra resource. In addition, when $e[i] = 0$, $\boldsymbol{T}$ and $\hat{\boldsymbol{T}}$ are forwarded from *PEB* to *PEA*.

The proposed architecture is capable of performing Montgomery powering ladder with some modifications. First, the exponent bit should be read from left to right. Second, the input ports of Multiply-adder modules in *PEA* and *PEB* should be rearranged in order to perform both squaring and common-multiplicand multiplication. Finally, one input port and one output port should be added to *PEA* and *PEB*, respectively, so that $\boldsymbol{T}$ and $\hat{\boldsymbol{T}}$ can be transferred when $e[i] = 0$. These modifications require no extra operator, but need more wiring and multiplexers. Besides, the critical path of exponentiation remains the same when implementing the Montgomery powering ladder. Therefore the running time still can be evaluated by Equation (26).

7 Implementation Results and Comparisons

The architectures of FMLE and its SPA protected version are implemented on the Xilinx Virtex-6 (xc6vlx240t-3) FPGA device. Parameter sets with 1,024, 2,048, 3,072, 4,096 and 7,680-bit operand sizes are described in Verilog-HDL and synthesized by ISE 14.7. We use the block RAMs with certain look-up tables (LUTs) to build all the RAM modules, and use the DSP48E1 slices to build the pipelined Karatsuba multiplier in the PEs. Both block RAM and DSP48E1 are synthesized with maximum operating frequency.

Table 4 reports the post place-and-route results of FMLE with different parameter sets. The RSA public key is assumed to be $2^{16} + 1$ [33], [34], [35], while the private key is

TABLE 4
Virtex-6 Implementation Results of the FMLE Architecture and Its SPA Protected Version

Key-length $\lceil \log_2 n \rceil$	l	u	v	FFT length s	Ring size q	LUTs	Slices	RAM blocks 36Kb/18Kb	DSP48E1 Slices	Cycles per FMLM	Period (ns)	Encryption time (ms)	Decryption time (ms)
1,024*	1,088	17	6	64	$2^{64}+1$	11,834	3,470	32/2	18	822	2.99	0.047	2.52
1,024* (SPA)	1,088	17	6	64	$2^{64}+1$	11,884	3,433	32/2	18	822	3.01	0.047	2.54
2,048*	2,112	33	6	64	$2^{128}+1$	27,363	7,644	62/2	54	849	4.33	0.070	7.54
2,048* (SPA)	2,112	33	6	64	$2^{128}+1$	27,386	7,719	62/2	54	849	4.24	0.068	7.38
3,072	3,136	49	6	64	$2^{128}+1$	27,069	7,539	65/1	54	1,266	4.33	0.104	16.85
3,072 (SPA)	3,136	49	6	64	$2^{128}+1$	27,050	7,649	65/1	54	1,266	4.25	0.102	16.54
4,096*	4,224	33	7	128	$2^{128}+1$	27,799	7,832	66/1	54	1,632	4.29	0.133	28.69
4,096* (SPA)	4,224	33	7	128	$2^{128}+1$	27,914	7,994	66/1	54	1,632	4.22	0.131	28.24
7,680	7,744	121	6	64	$2^{256}+1$	67,950	18,598	122/4	162	1,274	6.91	0.167	67.63
7,680 (SPA)	7,744	121	6	64	$2^{256}+1$	67,950	18,827	122/4	162	1,274	6.95	0.168	68.02

* *The all-at-once technique is adopted.*

TABLE 5
Performance Speculation and Comparison of Modular Exponentiations from 1,024-Bit to 7,680-Bit

Designs	Devices	LUTs	Slices	Latency (ms)	Area-latency product (LUTs×s)	Area-latency product (Slices×s)
$\lceil \log_2 n \rceil = \tau = 1{,}024$ (bit)						
[36]	ASIC	715,621 (μm^2)		2.23 (μs)	—	—
[9]	Stratix-III	4,127	—	11.06	46	—
[34]	Virtex-II	—	12,537	10.35	—	130
[35] (5 to 2)	Virtex-II	—	26,136	2.73	—	71
[6] (R2L, $d = 2$)	Virtex-5-3†	27,750	7,303	1.38	38	10
[37] (R2L)	Virtex-5-3	—	12,716	0.92	—	12
FMLE	Virtex-6-3	11,834	3,470	2.52	30	9
$\lceil \log_2 n \rceil = \tau = 2{,}048$ (bit)						
[9]	Stratix-III	8,150	—	53.98	440	—
[6] (R2L, $d = 1$)	Virtex-5-3	39,012	—	10.53	411	—
[6] (R2L, $d = 2$)	Virtex-5-3	55,309★	—	5.80★	321	—
[37] (R2L)	Virtex-5-3	—	25,432★	3.68★	—	94
FMLE	Virtex-6-3	27,363	7,644	7.54	206	58
$\lceil \log_2 n \rceil = \tau = 4{,}096$ (bit)						
[9]	Stratix-III	16,193	—	231.21	3,744	—
[6] (R2L, $d = 2$)	Virtex-5-3	110,236★	—	24.30★	2,679	—
[37] (R2L)	Virtex-5-3	—	50,864★	14.72★	—	749
FMLE	Virtex-6-3	27,799	7,832	28.69	798	225
$\lceil \log_2 n \rceil = \tau = 7{,}680$ (bit)						
[9] (8,192-bit)	Stratix-III	32,262	—	1062.19	34,268	—
[38] (8,192-bit)	Stratix-V	214,321	—	119‡	25,504	—
[6] (R2L, $d = 2$)	Virtex-5-3	192,248★	—	88.92★	17,095	—
[37] (R2L)	Virtex-5-3	—	89,012★	51.52★	—	4,586
FMLE	Virtex-6-3	67,950	18,598	67.63	4,595	1,258

★ *Speculated results obtained by estimating the growth tendency of area or latency;*
† *Virtex-5-3 denotes Virtex-5 FPGA with speed grade 3;*
‡ *Estimated by* $9.7\ \mu s \times 1.5 \times 8{,}192$, *each MMM require 9.7* μs.

with the same bit length of n. It can be found that the area and latency differences between the FMLE architecture and its SPA protected version are less than 3 percent, which matches our analysis in Section 6.4. The cycle requirement of FMLE is determined by the transform length s and whether the all-at-once technique is applied. Parameter sets performed by the same technique with the same s always result in similar numbers of clock cycles (e.g., the 1,024-bit and 2,048-bit cases in Table 4). On the other hand, the FPGA resource usage and clock frequency are determined by the ring size q. Parameter sets with the same q result in similar areas (number of LUTs or slices) and clock periods (e.g., the 2,048-bit and 3,072-bit cases in Table 4).

The performance comparison between FMLE and other modular exponentiation architectures is provided in Table 5. Since [38] only provides the MMM latency, we estimate its exponentiation latency by computing $9.7\ \mu s \times 1.5 \times 8{,}192$, where $9.7\ \mu s$ is the reported multiplication time and $1.5 \times 8{,}192$ is the number of MMMs for the average-case. Besides, we speculate the area and latency of [6] ($d = 2$) and [37] for 2,048-bit, 4,096-bit and 7,680-bit operand sizes and list the results in Table 5. Since the two designs are based on regular modular multiplication methods, their area and latency growth rates would be constant. Thus, both their time and area could be speculated once the growth rates are estimated.

Taking the R2L ($d = 2$) design in Table VIII of [6] as an example, the reported latencies are 0.33 ms for 512-bit and 1.38 ms for 1,024-bit, the growth rate is approximately 4.2. This indicates for each time when the operand size is doubled, the latency may be increased by roughly 4.2 times. Thus, we may speculate the latencies as 5.80 ms for 2,048-bit, 24.30 ms for 4,096-bit and 88.92 ms for 7,680-bit. Speculation results of [37] can be obtained in the same way.

For 1,024-bit in Table 5, the area of FMLE is larger than the design of [9], but smaller than the other works. Meanwhile, the latency of FMLE is lower than the designs of [9], [34] and [35], but higher than [6] and [37]. Thus, the area-latency efficiency of FMLE is not sufficiently superior to the others: 9 for FMLE, 10 for [6], and 12 for [37]. This indicates the FFT method is not guaranteed to be the optimal solution for 1,024-bit modular exponentiation. Appropriate computation method should be considered based on different design targets.

In terms of 2,048-bit and above key sizes in Table 5, though [6] and [37] have speed advantages, they consume much more LUTs or Slices than FMLE. As a result, FMLE achieves better area-time efficiency than the other compared works. Moreover, studies in [16] and [17] reveal that it is possible to speedup FMLE by either choosing a smaller transform length s or involving more butterfly structures, however, at the expense of consuming more hardware resources. Due to the area-time efficiency superiority of FMLE, such trade-off could be more cost-effective than the other approaches. Therefore, choosing FMLE would be more suitable for cost-effective modular exponentiation with 2,048-bit and above operand sizes.

We also compared the performance of a single FMLM with the state-of-the-art MMM architectures in Table 6, and the area-latency product growth tendency of different FFT-based multipliers are provided in Fig. 10. We only select the most area-latency efficient design of each operand size for a fair comparison. Meanwhile, both best-case and worst-case cycle requirements of [17] are included in the table, the best-case skips all the extra operations in MLM, while the worst-case performs them all. In addition, since different designs may involve different DSP slices and RAM blocks, their costs are also included for evaluation.

When compared with [16], FMLM obtains a smaller area-latency product and the efficiency is improved around 60 percent on average. Such improvement mainly comes from the avoidance of zero-padding, i.e., for the same u, the transform length s of FMLM is always half of [16]. The design in [16] implements Walter's variation of MMM [25], which eliminates the conditional selection of MMM. Thus, both multipliers have constant running time, and they are considered to be secure against timing attacks.

Both FMLM and [17] are based on McLaughlin's framework. Due to the elimination of the conditional selections in MLM, fewer clock cycles are required by FMLM for every multiplication. Besides, the clock period of FMLM is also reduced due to the simplified control logic. Thus, FMLM achieves better efficiency for 1,024-bit, 3,072-bit and 4,096-bit. On the other hand, the efficiency of [17] is better for 2,048-bit. This is because $s = 128$ and $q = 2^{64} + 1$ are no longer available for FMLM when using the all-at-once technique $c = (3u + 2v)/s = 65/128 > 1/2$ and a larger q must

TABLE 6
Performance Comparison Between FMLM and the State-of-the-Art Modular Multiplication Architectures

l	Platform	Design	DSP	RAMB36[1]	LUTs	Slices	Cycles	Period (ns)	Latency (μs)	Area-latency product (LUTs$\times\mu$s)	Area-latency efficiency improvement (%)
1,024	Virtex-II-4[2]	[39] (radix-2, $\omega = 32$)	—	—	5,310	—	1,056	9.55	10.09	53,577	—
1,024	Virtex-5-2	[40] (LZD = 2)	—	—	—	6,091	340[3]	2.5	0.851	—	—
1,084	Virtex-6-1	[16] ($u = 17$, $s = 128$)	9	11	4,818	1,483	1,440	5.29	7.57	36,472	87.5
1,024	Virtex-6-1	[17] ($u = 16$, $s = 64$)	9	16.5	6,047	1,757	1,052 (992)	3.80	4.00	24,173	24.3
1,088	Virtex-6-1	FMLM ($u = 17$, $s = 64$)	9	14.5	5,794	1,915	917	3.67	3.37	19,449	—
2,048	Virtex-II-4	[39] (radix-2, $\omega = 32$)	—	—	10,587	—	2,112	9.79	20.68	218,939	—
2,076	Virtex-6-1	[16] ($u = 65$, $s = 64$)	54	27.5	14,895	5,534	837	6.6	5.52	82,220	39.6
2,048	Virtex-6-1	[17] ($u = 16$, $s = 128$)	9	17.5	7,337	2,083	2,036 (1,928)	3.88	7.90	57,960	-1.6
2,112	Virtex-6-1	FMLM ($u = 33$, $s = 64$)	27	26.5	12,771	4,074	924	4.99	4.61	58,884	—
3,072	Virtex-II-4	[39] (radix-2, $\omega = 32$)	—	—	15,197	—	3,168	9.77	30.94	470,195	—
3,196	Virtex-6-1	[16] ($u = 25$, $s = 256$)	9	11	5,835	1,977	3,633	5.09	18.49	107,889	59.6
3,072	Virtex-6-1	[17] ($u = 24$, $s = 128$)	9	22.5	7,351	2,103	2,770 (2,566)	3.84	10.64	78,191	15.6
3,200	Virtex-6-1	FMLM ($u = 25$, $s = 128$)	9	21.5	7,561	2,436	2,391	3.74	8.94	67,613	—
4,096	Virtex-II-4	[39] (radix-2, $\omega = 32$)	—	—	19,621	—	4,224	9.91	41.85	821,138	—
4,124	Virtex-6-1	[16] ($u = 129$, $s = 64$)	135	49.5	27,839	9,530	846	9.2	7.78	216,587	86.7
4,096	Virtex-6-1	[17] ($u = 32$, $s = 128$)	27	29	14,243	3,898	2,040 (1,932)	5.41	11.04	157,191	35.5
4,224	Virtex-6-1	FMLM ($u = 33$, $s = 128$)	27	29	13,083	4,194	1,784	4.97	8.87	116,000	—

[1] *RAMB36 refers to 36 Kb RAM blocks, each 18 Kb block is counted as 0.5 RAMB36;*
[2] *Virtex-II-4 denotes Virtex-II FPGA with speed grade 4;*
[3] *Estimated value.*

be selected. When compared to [17], the security of FMLM is also improved, since its running time is constant while [17] varies.

In summary, the efficiency of FMLM is improved compared to [16], and both efficiency and security are improved compared to [17].

8 Discussion on Further Reducing the Number of Long Additions

It is possible to save two more l-bit long additions by further extending the I/O bounds from $2n$ to $3n$, and redefining $r > 9n$. As a result, the number of long additions and subtractions is reduced from 10 to 1. This improvement is described as follows.

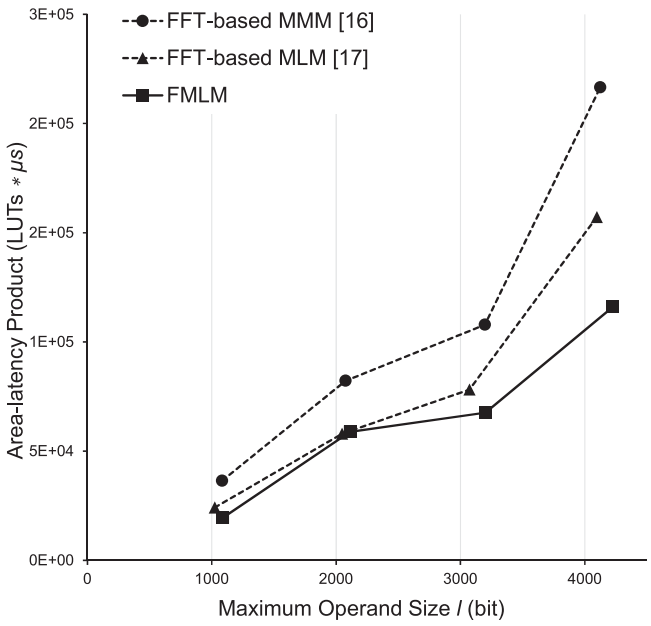


Fig. 10. Area-latency product growth tendency of FMLM, [16] and [17].

Recalled in Section 4.2 that either a or $a + r$ is tolerable in Step 2 of Algorithm 4. Similarly, if obtaining $m + r$ in Step 4 is permitted, we could use one $(u + v + 2)$-bit addition for the modulo-r reduction, and finally compute $(xy + mn + rn)/r = t + n$. Since $t + n \equiv t \pmod{n}$, $t + n$ is still a valid output as long as the input bounds are consistent with the output. The output is bounded by $t + n < 3n$, therefore, the input bounds are extended to $x < 3n$ and $y < 3n$. Besides, as $xy < rn$ should be satisfied in MMM [10], we have $h > r > 9n$.

With the $3n$ I/O bound, q must be restricted to $q > 3(b-1)^2 s$ to avoid data overflow during the computation of $(\boldsymbol{x} \bullet_s \boldsymbol{y}) + (\boldsymbol{m} \bullet_s \boldsymbol{n}) + (\boldsymbol{r} \bullet_s \boldsymbol{n})$. In addition, since $r > 9n$, the specification of l would be revised to $l = \lceil \log_2 n \rceil + 4$. Though the expressions of q and l change, it can be found that the parameter sets in Table 3 still support the $3n$ I/O bound.

Consequently, the long additions and subtractions of FMLM can be reduced from 10 to 1 when applying the $3n$ I/O bound. However, an extra conditional subtraction would be involved during modular exponentiation, since $\mathrm{FMLM}(t, 1) = [t + (tn' \bmod r)n]/r \leqslant (3n - 1 + 2rn - n)/r \leqslant 2n$. As the subject of this paper is to avoid conditional selections, we did not apply such improvement to the architecture of FMLE.

9 Conclusions

In this work, an improved variation of McLaughlin's Montgomery modular multiplication (MLWS) is proposed. MLWS eliminates all the conditional selections in the original McLaughlin's multiplication (MLM) algorithm by accepting a larger input or output bound and redefining larger moduli (r and h). Compared to the original MLM algorithm, the area-time efficiency of MLWS is improved as no extra addition or subtraction is required during the modular multiplication. Besides, its security against timing attacks is also improved as the processing time of MLWS is constant.

Then, we apply the FFT method to the modular multiplication steps of MLWS and derive the FFT-based MLWS

(FMLM) algorithm. Based on McLaughlin's framework, the transforms involved in FMLM can be performed efficiently without zero-padding. Besides, by applying the carry-save technique and optimizing the computing data-flow, the number of long additions and subtractions required by the modulo-r and h reductions is further reduced to 3. Combining the right-to-left binary method with FMLM, we derive an FFT-based modular exponentiation (FMLE) algorithm. Due to the constant running time of FMLM, FMLE is considered to be secure against timing attacks and suitable for RSA computation.

Finally, an area-latency efficient hardware architecture of FMLE is created with dual pipelined FFT-based multipliers. We also equip our FMLE architecture with protection against simple power analysis (SPA) by adopting the Montgomery powering ladder. Because of the dual-multiplier design, the SPA protected FMLE only increases less than 3 percent of the area and requires the same clock cycles when compared to its unprotected version. The implementation results of FMLM validate that it provides constant running time for different inputs as the cycle requirement remains the same. Therefore, FMLM achieves both efficiency and security improvements when compared to the state-of-the-art FFT-based approaches. In addition, the comparison results show that our FMLE exhibits the superiority of area-time efficiency over the state-of-arts from and above 2,048-bit operand sizes.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. They are also grateful to Mr. Jingwei Hu who contributed the key ideas on side-channel analysis. This work was supported by the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. CityU 111913) and Croucher Statup Allowance, 9500015.

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, 1978.

[2] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[3] E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, P. D. Gallagher, "Recommendation for key management–part 1: General (revision 3)," *NIST special publication*, vol. 800, no. 57, pp. 1–147, 2012.

[4] Ç. K. Koç, "High-speed RSA implementation," RSA Data Security, Inc., RSA Laboratories, Tech. Rep. TR 201, 1994.

[5] D. E. Knuth, *The Art of Computer Programming, Seminumerical Algorithms*, 2nd ed., vol. 2. Reading, MA, USA: Addition-Wesley, 1981.

[6] G. D. Sutter, J.-P. Deschamps, and J. L. Imaña, "Modular multiplication and exponentiation architectures for fast RSA cryptosystem based on digit serial computation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 7, pp. 3101–3109, Jul. 2011.

[7] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *Doklady Akad. Nauk SSSR*, vol. 145, no. 293–294, 1962, Art. no. 85.

[8] S. A. Cook and S. O. Aanderaa, "On the minimum computation time of functions," *Trans. Amer. Math. Soc.*, vol. 142, pp. 291–314, Aug. 1969.

[9] C. P. Rentería-Mejía, V. Trujillo-Olaya, and J. Velasco-Medina, "Design of an 8,192-bit RSA cryptoprocessor based on systolic architecture," in *Proc. VIII Southern Conf. Programmable Logic*, Mar. 2012, pp. 1–6.

[10] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, 1985.

[11] A. Schönhage and V. Strassen, "Schnelle multiplikation großer zahlen," *IEEE Comput.*, vol. 7, no. 3–4, pp. 281–292, 1971.

[12] M. Fürer, "Faster integer multiplication," *SIAM J. Comput.*, vol. 39, no. 3, pp. 979–1005, 2009.

[13] D. Harvey, J. van der Hoeven, and G. Lecerf, "Even faster integer multiplication," *J. Complexity*, vol. 36, pp. 1–30, 2016.

[14] S. Covanov and E. Thomé, "Fast arithmetic for faster integer multiplication," *CoRR*, vol. abs/1502.02800, 2015. [Online]. Available: http://arxiv.org/abs/1502.02800

[15] J. P. David, K. Kalach, and N. Tittley, "Hardware complexity of modular multiplication and exponentiation," *IEEE Trans. Comput.*, vol. 56, no. 10, pp. 1308–1319, Oct. 2007.

[16] D. D. Chen, G. X. Yao, R. C. C. Cheung, D. Pao, and Ç. K. Koç, "Parameter space for the architecture of FFT-based Montgomery modular multiplication," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 147–160, Jan. 2016.

[17] W. Dai, D. D. Chen, R. C. C. Cheung, and Ç. K. Koç, "Area-time efficient architecture of FFT-based montgomery multiplication," *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 375–388, Mar. 2017.

[18] P. McLaughlin Jr, "New frameworks for montgomerys modular multiplication method," *Math. Comput.*, vol. 73, no. 246, pp. 899–906, 2004.

[19] D. S. Phatak and T. Goff, "Fast modular reduction for large wordlengths via one linear and one cyclic convolution," in *Proc. 17th IEEE Symp. Comput. Arithmetic*, Jun. 2005, pp. 179–186.

[20] G. Saldamlı and Ç. K. Koç, "Spectral modular exponentiation," in *Proc. 18th IEEE Symp. Comput. Arithmetic*, Jun. 2007, pp. 123–132.

[21] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proc. 16th Annu. Int. Cryptology Conf. Adv. Cryptology*, 1996, pp. 104–113.

[22] J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestré, J. J. Quisquater, and J. L. Willems, "A practical implementation of the timing attack," in *Proc. Int. Conf. Smart Card Res. Appl.*, 1998, pp. 167–182.

[23] W. Schindler, "A timing attack against RSA with the chinese remainder theorem," in *Proc. 2nd Int. Workshop Cryptographic Hardware Embedded Syst.*, 2000, pp. 109–124.

[24] C. D. Walter and S. Thompson, "Distinguishing exponent digits by observing modular subtractions," in *Proc. Conf. Topics Cryptology*, 2001, pp. 192–207.

[25] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electron. Lett.*, vol. 35, no. 21, pp. 1831–1832, 1999.

[26] R. Crandall and B. Fagin, "Discrete weighted transforms and large-integer arithmetic," *Math. Comput.*, vol. 62, no. 205, pp. 305–324, 1994.

[27] H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, 2nd ed. Berlin, Germany: Springer, 1982.

[28] J. M. Pollard, "The fast Fourier transform in a finite field," *Math. Comput.*, vol. 25, no. 114, pp. 365–374, 1971.

[29] C. D. Walter, "Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli," in *Proc. Cryptographer's Track RSA Conf. Topics Cryptology*, 2002, pp. 30–39.

[30] R. Creutzburg and M. Tasche, "Number-theoretic transforms of prescribed length," *Math. Comput.*, vol. 47, no. 176, pp. 693–701, 1986.

[31] D. J. Bernstein, "Multidigit multiplication for mathematicians," [Online]. Available: http://cr.yp.to/papers.html#m3

[32] M. Joye and S.-M. Yen, "The Montgomery powering ladder," in *Proc. Cryptographic Hardware Embedded Syst.*, 2003, pp. 291–302.

[33] E. Barker, L. Chen, and D. Moody, "Recommendation for pairwise key-establishment schemes using integer factorization cryptography," NIST, Gaithersburg, MD, USA, Tech. Rep. 800-56B, Rev 1, 2014.

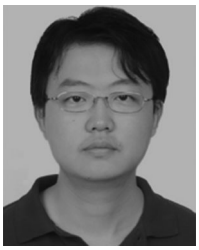
[34] M. D. Shieh, J. H. Chen, H. H. Wu, and W. C. Lin, "A new modular exponentiation architecture for efficient design of RSA cryptosystem," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 9, pp. 1151–1161, Sep. 2008.

[35] C. McIvor, M. McLoone, and J. V. McCanny, "Modified montgomery modular multiplication and RSA exponentiation techniques," *IEEE Proc.—Comput. Digit. Techn.*, vol. 151, no. 6, pp. 402–408, Nov. 2004.

[36] S. Vollala, V. Varadhan, K. Geetha, and N. Ramasubramanian, "Design of RSA processor for concurrent cryptographic transformations," *Microelectron. J.*, vol. 63, pp. 112–122, 2017.

[37] A. Rezai and P. Keshavarzi, "High-throughput modular multiplication and exponentiation algorithms using multibit-scan–multibit-shift technique," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 9, pp. 1710–1719, Sep. 2015.

[38] W. Wang and X. Huang, "A novel fast modular multiplier architecture for 8,192-bit RSA cryposystem," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, Sep. 2013, pp. 1–5.
[39] M. Huang, K. Gaj, and T. El-Ghazawi, "New hardware architectures for Montgomery modular multiplication algorithm," *IEEE Trans. Comput.*, vol. 60, no. 7, pp. 923–936, Jul. 2011.
[40] A. Rezai and P. Keshavarzi, "Compact SD: A new encoding algorithm and its application in multiplication," *Int. J. Comput. Math.*, vol. 94, no. 3, pp. 554–569, 2017.



Wangchen Dai received the BEng degree in electrical engineering and automation from Beijing Institute of Technology, China, in 2010, and the MASc degree from the University of Windsor, Canada, in 2013. He is currently working toward the PhD degree in the Department of Electronic Engineering, City University of Hong Kong. His research interests include high-performance hardware implementation of computer arithmetic, architecture design and optimization of cryptographic algorithms, and reconfigurable computing. He is a student member of the IEEE.

Donglong Chen received the BEng degree in electronic and information engineering from Wuhan University of Technology, Wuhan, China, in 2011, and the PhD degree in electronic engineering from the City University of Hong Kong, Hong Kong, China, in 2015. He served as an engineer with Huawei Technology Co., Ltd., from 2015 to 2017. He is currently working as a senior engineer with Tencent Technology (Shenzhen) Co., Ltd., China. His research interests include cryptographic algorithm optimization in hardware and high-speed digital system design, especially in field programmable gate array (FPGA).

Ray C. C. Cheung received the BEng and MPhil degrees in computer engineering and computer science and engineering from the Chinese University of Hong Kong (CUHK), Hong Kong, in 1999 and 2001, respectively, and the PhD degree and DIC in computing from Imperial College London, London, United Kingdom, in 2007. After completing the PhD work, he received the Hong Kong Croucher Foundation Fellowship for his postdoctoral study in the Electrical Engineering Department, University of California, Los Angeles (UCLA). In 2009, he worked as a visiting research fellow in the Department of Electrical Engineering, Princeton University, Princeton, NJ. Currently, he is an associate professor in the Department of Electronic Engineering, City University of Hong Kong (CityU). He is the author of more than 100 journal and conference papers. His research team, CityU Architecture Lab for Arithmetic and Security (CALAS), focuses on the following research topics: reconfigurable trusted computing, applied cryptography, and high-performance biomedical VLSI designs. He is a member of the IEEE.

Çetin Kaya Koç received the PhD degree in electrical & computer engineering from the University of California, Santa Barbara, in 1988. His research interests include electronic voting, cyber-physical security, cryptographic hardware and embedded systems, elliptic curve cryptography and finite fields, and deterministic, hybrid and true random number generators. He is the co-founder of the Cryptographic Hardware and Embedded Systems Conference, and the founding editor-in-chief of the Journal of Cryptographic Engineering. He has also been in the editorial boards of the *IEEE Transactions on Computers* (2003-now) and the *IEEE Transactions on Mobile Computing* (2003-2007). He is the author and co-author of the about 200 articles 3 books in computer science and cryptography. He was elected as an IEEE fellow for his contributions to cryptographic engineering in 2007. Currently, Koç has appointments at Istinye University (İstanbul, Turkey), Nanjing University of Aeronautics and Astronautics (Nanjing, China), and University of California Santa Barbara. He is a fellow of the IEEE.