A Novel and Efficient Design for an RSA Cryptosystem With a Very Large Key Size

Xinming Huang, *Senior Member, IEEE*, and Wei Wang, *Member, IEEE*

***Abstract*—This brief presents a novel and efficient design for a Rivest–Shamir–Adleman (RSA) cryptosystem with a very large key size. A new modular multiplier architecture is proposed by combining the fast Fourier transform-based Strassen multiplication algorithm and Montgomery reduction, which is different from the interleaved version of Montgomery multiplications used in traditional RSA designs. A new modular exponentiation algorithm is also proposed for the RSA design. Applying this method, we have implemented 8K/12K-bit and 48K-bit RSA on application-specific integrated circuit designs. The results show that the proposed method gains more advantage as the key size increases, which matches the complexity analysis. Performance comparisons show that the 48K-bit design, which is applicable for both RSA and fully homomorphic encryption, outperforms the previous works with respect to throughput and efficiency.**

***Index Terms*—Application-specific integrated circuit (ASIC), fully homomorphic encryption (FHE), Montgomery algorithm, Rivest–Shamir–Adleman (RSA), Strassen's multiplication.**

I. Introduction

THE Rivest–Shamir–Adleman (RSA) [1] system is one of the most widely used public key cryptography systems. The RSA operation is essentially a modular exponentiation. Its security level relies on that there are no effective procedures or algorithms that can factorize a large integer within a short time period using a regular computer. Nowadays, the size of modulus is typically 2048 bits, which can provide a good level of security. As Moore's law continues its scaling and more powerful computers emerge, the key size of 2048 bits may not be sufficient. It becomes necessary to increase the key size to 4096 bits or even 8192 bits to provide information security. RSA cryptosystem recursively performs modular multiplications to complete an operation of modular exponentiation. As a result, the performance of RSA systems relies on the throughput rate of the modular multiplication. In addition to the RSA cryptosystem, fully homomorphic encryption (FHE) also requires a large amount of modular multiplications. The first implementation of lattice-based FHE, which was proposed by Gentry and Halevi [2], utilizes modular multiplications with its public key size around 780 000 bits in the small setting. Later, a leveled FHE scheme is constructed with asymptotically linear efficiency. The public key size is from 9326 to 93 623 bits for different circuit depths in the level FHE implementation reported in [3].

When the RSA key size becomes large, hardware accelerators are often required to achieve high throughput rate. There are two popular methods for modular multiplication. One is the interleaved Montgomery's multiplication algorithm [4], and the other is multiplications followed by modular reduction. Traditionally, the interleaved Montgomery's multiplication algorithm with complexity $O(N^2)$ is employed for modular multiplication. For RSA with small key size, the interleaved Montgomery modular multiplication algorithm is a good choice. It achieves high throughput at a low cost of hardware resource [5]–[9].

The fast Fourier transform (FFT)-based Strassen algorithm [10] is an efficient algorithm for large-number multiplication. It has the complexity of $O(N \cdot \log N \cdot \log\log N)$. Although the algorithm was originally created for large-number arithmetic in scientific computing, it can also be used for large-number modular multiplications. Therefore, we suggest it as a promising solution for RSA and FHE implementations when the key size becomes very large. In this brief, we extend our previous work on large-number multiplier design for field-programmable gate array (FPGA) [11] and propose a novel approach of modular multiplication by combining the FFT-based multiplications and Montgomery reduction [4]. Hardware architecture and application-specified integrated circuit (ASIC) implementation results are presented.

The rest of this brief is organized as follows: Section II describes Strassen's algorithm and also the Montgomery multiplication and exponentiation algorithms; Section III presents the hardware architecture and the detail design for 8 K/12 K-bit RSA; Section IV discusses about 48 K-bit RSA design; Section V compares the performance of the proposed ASIC implementations with previous works; Section VI provides the conclusion.

II. Algorithms

A. Strassen's Algorithm for Large-Number Multiplication

Large-number modular multiplication is a crucial part of RSA computation. Traditional binary multiplication algorithms are not efficient when both multiplicands become very large spanning thousands of bits. Strassen's algorithm was developed from convolution theory [10]. Briefly, each multiplicand is broken into N samples, which are also called digits, in base β. It then employs FFT and inverse FFT (IFFT) operations to

Manuscript received January 12, 2015; revised March 19, 2015; accepted May 19, 2015. Date of publication July 17, 2015; date of current version September 25, 2015. This brief was recommended by Associate Editor G. Masera. *(Corresponding author: Wei Wang.)*

The authors are with the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609 USA (e-mail: xhuang@wpi.edu; wei.wang217@gmail.com).

Color versions of one or more of the figures in this brief are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSII.2015.2458033

TABLE I
ROOT OF UNITY r_K FOR THE CHOSEN PRIME p

FFT size K	the root of unity r_K
2	2^{96}
4	2^{48}
8	2^{24}
16	2^{12}
32	2^{6}
64	2^{3}

compute the product of two multiplicands. The detail procedures are listed in Algorithm 1 [10].

Algorithm 1 Strassen's FFT-Based Multiplication Algorithm

Input: Multiplicands a and b, base β, a and b are two non-negative integers, each having at most N digits in base β.
Outputs: $c = a * b$
1. Initialize: Zero-pad both of a and b until each has length $2N$ for cyclic convolution
2. Transforms and component-wise multiplication: $C[i] =$ FFT$(a)[i] *$ FFT$(b)[i]$.
3. Inverse transform: $c =$ IFFT(C)
4. Re-assembly:
carry $= 0$;
for $(0 \le i \le 2N)\{$
$y = c[i] +$ carry;
$c[i] = y \bmod \beta$; carry $= \lfloor y/\beta \rfloor$;
$\}$
return c

Typically, FFT operations are performed using floating-point arithmetic, which consumes large amount of hardware resources and is often slower than fixed-point designs. Alternatively, we choose to perform the computation using *Number Theoretical Transform* in the finite field $\boldsymbol{Z}/p\boldsymbol{Z}$, with prime p. By choosing p as a special prime $2^{64} - 2^{32} + 1$ [12], it has certain special identities, such as: $(2^{192} \mod p = 1)$; $(2^{96} \mod p = -1)$; $(2^{64} \mod p = 2^{32} - 1)$. These special identities can reduce the computations considerably.

The finite field K-point FFT can be represented as

$$X_i = \sum_{j=0}^{K-1} x_j (r_K)^{ij} \pmod p \tag{1}$$

where $K = 2N$, and r_K is the Kth root of unit for prime p, such that $r_K^K \bmod p = 1$ while $r_K^i \bmod p \neq 1$ $(i = 1, 2, \ldots, K-1)$.

Similarly, the IFFT can be represented as

$$x_i = K^{-1} \sum_{j=0}^{K-1} X_j (r_K)^{-ij} \pmod p. \tag{2}$$

Table I lists the unity roots for small-size finite-field FFTs $(K \le 64)$. They are all powers of 2. From (1) and (2), it is obvious that all these small-size FFTs can be computed using only shift and modulo addition operations rather than costly multiplications. Subsequently, we can apply the well-known Cooley–Tukey algorithm to compute a large FFT using small FFTs in the form of the "butterfly" structure.

TABLE II
DIFFERENT SIZES OF FFT-BASED MULTIPLICATIONS

Multiplicand size $(\frac{K}{2}\log_2\beta)$	FFT size (K)	Butterfly elements	Base unit β
3K bits	2^8	Radix-2/4	2^{24} (24-bit)
12K bits	2^{10}	Radix-2/4/32	2^{24} (24-bit)
48K bits	2^{12}	Radix-2/4/8/16/64	2^{24} (24-bit)

Note that the condition $(1/2)K(\beta - 1)^2 < p$ must be satisfied to prevent the digitwise overflow in convolution. For prime $p = 2^{64} - 2^{32} + 1$, Table II shows three multiplications in different sizes using the same base unit of 24 bits.

B. Montgomery Multiplication and Exponentiation

Popular algorithms for modular reduction include the Montgomery reduction [4] and the Barrett reduction [13]. Both have similar computational complexity. However, the control logic of Barrett algorithm is more complicated if implemented in hardware. Thus, we choose Montgomery reduction in this work.

We employ the FFT-based Strassen's algorithm described earlier to perform three large-number multiplications in the Montgomery procedures, as shown in Algorithm 2. Multiplying two numbers X and Y is expressed as IFFT(FFT$(X) \odot$ FFT(Y)), where $\odot$ is the componentwise product. In Steps 3 and 5, we can precompute the FFTs of M and M'. In addition to three operations of large-number multiplication, there are also two operations of large-number addition as in Steps 6 and 8.

Algorithm 2 Montgomery Algorithm Using FFT-Based Multiplications

Procedure Montgomery(X, Y, M): $c = XYR^{-1} \pmod M$
Pre-computation: $n = \lceil \log_2^M \rceil$, $R = 2^n$, $M' = -M^{-1} \pmod R$
1. $T \leftarrow$ IFFT(FFT$(X) \odot$ FFT(Y));
2. $t \leftarrow T \bmod R$;
3. $U \leftarrow$ IFFT(FFT$(t) \odot$ FFT(M'));
4. $u \leftarrow U \bmod R$;
5. $W \leftarrow$ IFFT(FFT$(u) \odot$ FFT(M));
6. $C \leftarrow T + W$;
7. $c \leftarrow C/R$;
8. If $c \ge M$ then $c \leftarrow c - M$, end if
end procedure.

To compare the arithmetic cost of the interleaved Montgomery algorithm and the FFT-based algorithm, we implement both algorithms in carefully tuned MIPS64 assembly and count the number of arithmetic logic unit (ALU) operations for each, as shown in Fig. 1. It shows that the FFT-based algorithm has significantly lower computational complexity when the key size becomes larger.

We use the algorithm shown in Algorithm 3 for modular exponentiation similar to the most significant bit-first algorithm in [14]. In this algorithm, P is a k-bit message with value less than the modulus M. E is denoted as an m-bit exponent or key. Similarly, we can employ the FFT-based large-number multiplications. The square operation in Step 3 of Algorithm 3 requires

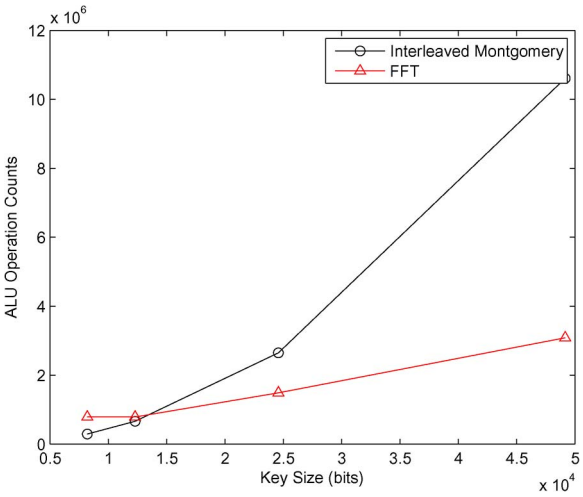

Fig. 1. FFT-based algorithm has less computation when key size becomes very large.

one FFT operation and one IFFT operation. In Step 5, the FFT results of P' are computed only once and then stored in the memory. Hence, the multiplication in Step 5 takes one FFT operation and IFFT operation in each iteration. Effectively, the total number of FFT and IFFT operations is reduced by 1/3 in Algorithm 3.

Algorithm 3 Modular Exponentiation Using FFT-Based Multiplications

Procedure Exponentiation$(P, E, C,\ M)$: $C = P^E (\mathrm{mod}\ M)$
Inputs: P = plain text; E = exponent = $[e_{k-1}e_{k-2}\ldots e_0]$, $e_i \in [0,1]$;M = module of m bits.
Pre-computation: $n = \lceil \log_2^M \rceil$, $R = 2^n$,$R' = R^{-1} \mathrm{mod}\, M$, $\mathrm{FFT}(cur) = \mathrm{FFT}(1 \times R \,\mathrm{mod}\, \mathrm{M})$
1. $P' = \mathrm{IFFT}(\,\mathrm{FFT}(P) \odot \mathrm{FFT}(cur)\,) \bmod M$;
2. for i in $k-1$ to 0 do
3. $\quad cur \leftarrow \mathrm{IFFT}(\,\mathrm{FFT}(cur) \odot \mathrm{FFT}(cur)\,) \bmod M$;
4. $\quad$ if $e_i = 1$ then
5. $\quad\quad cur \leftarrow \mathrm{IFFT}(\,\mathrm{FFT}(cur) \odot \mathrm{FFT}(P')\,) \bmod M$;
6. end for;
Post-computation: $C \leftarrow cur \times R' \mathrm{mod}\, M$;
end procedure.

III. 8K/12K-bit RSA Hardware Design

Here, we propose the hardware architecture that implements 8 K- or 12 K-bit RSA cryptosystem using the same design. The key component is the FFT-based large-number multiplier that is frequently used during the operations of modular multiplication and modular exponentiation. In the following, we present the design of the 12 K-bit Montgomery multiplier that can be configured to perform 8 K-bit multiplications as well.

A. Overall System Architecture

Given a large-number multiplicand with 12 288 bits, we can decompose it into a series of 512 samples, and each sample is 24-bit, which is also referred to as base β. For the same design,

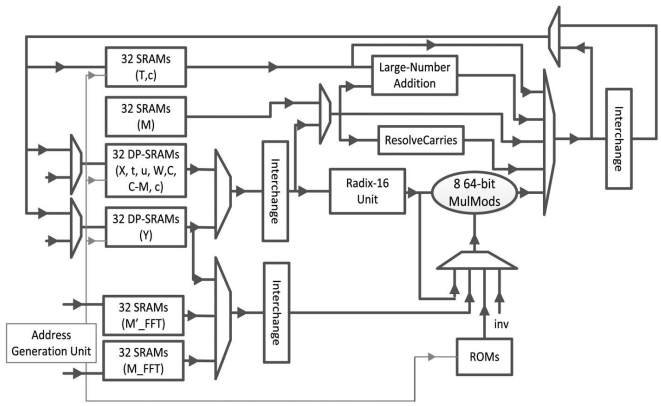

Fig. 2. Architecture design of the 8K/12K-bit Montgomery multiplier.

if we change the base β from 24 to 16 bits, then it performs 8192-bit multiplications. Therefore, the proposed 12 K-bit multiplier design can also be used for 8 K-bit RSA simply by padding each 16-bit sample with 8 bits of zeros to the left. Fig. 2 shows the overall architecture of the 12 K-bit Montgomery multiplier.

Applying Strassen's algorithm, multiplication of two numbers is to obtain cyclic convolution of two data series, each with 512 samples. Typically, cyclic convolution involves "zero padding," and the result contains approximately twice many samples as that of the input signal. Hereby, a 1024-point finite-field FFT processor is constructed in this design. Based on the Cooley–Tukey algorithm, a single-stage 1024-point FFT can be decomposed into two stages of 32-point FFTs. Using in-place memory-based FFT architecture, as described in Section III-B, these two stages are computed iteratively on the same hardware unit and memory space. In fact, we only deploy one highly optimized radix-16 butterfly unit that is used four times recursively to complete a 32-point FFT operation. A radix-16 butterfly unit is much smaller than a radix-32 butterfly unit. The decision is based on the tradeoff between throughput and hardware resource usage. The detail design of the finite-field FFT will be discussed in Section III-B.

The process can be described as follows. At each clock cycle, 32 input samples are read from memory, permuted into a proper order by the *Interchange* unit, fed to the radix-16 unit to process recursively four times, modular multiplied by the twiddle factors supplied by ROMs, permuted again by the *Interchange* unit, and written back to the memory. The memory is partitioned into 32 banks with 32 words in each bank. An in-place memory addressing scheme can be derived to ensure there is no memory access conflict in reference to [15] and [16]. The memory needs to perform data read and data write concurrently; hence, dual-port SRAMs are employed in Fig. 2.

At the first stage of FFT or IFFT operations, 8 units of 64-bit *ModMuls* are used to multiply the output from radix-16 unit with the twiddle factors. At the second stage, the same 8 units of 64-bit *ModMuls* can be reused to compute the componentwise product of $\mathrm{FFT}(X)$ and $\mathrm{FFT}(Y)$. Upon completion of the IFFT operation, the data are fed into the resolve carries unit that produces the 24 576-bit results in a series of 24-bit samples.

For Montgomery multiplication, a large-number addition unit is introduced to perform the operation of Step 6 in Algorithm 2. For the comparator in Step 8, the 2's complement

of M is precomputed and stored in the memory; hence, the large-number addition unit is reused as a comparator.

For modular exponentiation, it is implemented on the same architecture by performing modular multiplications recursively. In this design, the 12 288-bit exponent is stored in registers and fed to a state machine to control the modular exponentiation process. As described in Algorithm 3, Step 2 is a square operation, and Step 4 is a multiplication. The same process repeats for k times until the modular exponentiation completes.

B. Design of the FFT Unit

The finite-field FFT and IFFT units are the main components of the large-number multiplier. For high-throughput applications, a pipeline FFT architecture is often used [17]. However, the pipeline design requires a local memory buffer at every stage [17], which often results in larger chip area and more power consumption. In contrast, a memory-based in-place FFT architecture stores the intermediate results at the same memory where the input data are read from. As a result, it minimizes the memory usage but still produces high throughput [15]. Thus, the memory-based strategy is adopted in this design.

The memory-based FFT architecture mainly consists of a butterfly unit, a data memory, a ROM storing the twiddle factors, and a control logic unit. The butterfly unit needs to read γ input data from the memory and then write back γ output results back to the memory. Hereby, we adopt a conflict memory access approach [15], [16] that partitions the memory into γ banks for concurrent conflict-free read and write access. More specifically, the input data with indexes $D = [d_{n-1}, d_{n-1}, \ldots, d_0]_\gamma$, where $n = \log_\gamma^K$, are stored at address $= [d_{n-1}, d_{n-2}, \ldots, d_1]_r$ and bank $= (\sum_{i=0}^{n-1} d_i) \bmod r$. In this design, γ is 32, and K is 1024.

In this architecture, we need to design a 1024-point finite-field FFT processor. For hardware efficiency, we choose to use two stages of 32-point FFTs to implement the 1024-point FFT. A radix-16 butterfly unit can be used recursively for four times to compute one radix-32 FFT. For 16-point FFT, the root of unity is 2^{12}. From (1) and (2), it is obvious that only shift operations and modulo additions are needed to compute the FFT. The detail design of the fast and efficient radix-16 butterfly unit can be referred to our previous work [11].

C. Design of the Resolve Carries Unit

The memory consists of γ banks; hence, γ data can be read from memory banks concurrently to resolve carries. The sequential algorithm in Algorithm 1 is the bottleneck of the system throughput. A parallel scheme is needed to resolve the carries rapidly.

The base-β representation of a nonnegative integer x is $x = \sum_{i=0}^{N-1} x_i \beta^i$, where $0 \le x_i < \beta$. Each datum in the memory bank is a 64-bit integer. The integer x can be viewed as a vector $\overrightarrow{x}$ in the base-β representation. The resolve carries procedure can be viewed as a vector–vector multiplication if computed in parallel. Usually, β is set to the powers of 2; hence, the computation in the resolve carries procedure is converted to the large-number additions, as shown in (3), which we can turn to the fast carry-look-ahead scheme. We use the pipeline carry-look-ahead scheme-based carry-resolve unit in our previous work [11] for the hardware implementation. The carry-resolve unit can resolve 32 data samples per clock cycle to meet the system throughput requirement. Thus

$$\text{ResolveCarries} = (1, \beta, \beta^2, \ldots, \beta^{\gamma-1}) \cdot \begin{pmatrix} \overrightarrow{x_0} \\ \overrightarrow{x_1} \\ \overrightarrow{x_2} \\ \vdots \\ \overrightarrow{x_{\gamma-1}} \end{pmatrix}. \quad (3)$$

TABLE III
SYNTHESIS RESULTS USING 90-nm CMOS TECHNOLOGY (CORE VOLTAGE OF 1.32 V)

	Core Area	Radix-16	SRAMs	Logic Gates G_{cal} (SRAM excluded)
12K-bit RSA	11.3 mm^2	2.13 mm^2	6.8 mm^2	2,045 K
48K-bit RSA	23.6 mm^2	2.13 mm^2	18.7 mm^2	2,200 K

IV. 48 K-BIT RSA HARDWARE DESIGN

The 48 K-bit RSA design has the same hardware architecture as the 12 K-bit RSA design. Each 48k-bit multiplicand is split into 2048 data, each with 24 bits. It requires a 4096-point FFT that is accomplished by applying the same radix-16 butterfly operation iteratively in three stages. However, the memory usage is much larger, which is partitioned into 16 banks with 256 words in each bank. For the resolve carry and large-number adder units, 16 data samples are needed to be processed in each cycle. Similar to the 12 K-bit RSA design, the 49 152-bit exponent are stored in a single-port SRAM and fed into a state machine to control the modular exponentiation process. In addition to RSA applications, the 48K-bit modular multiplier can also be used for the leveled FHE scheme with a circuit depth $L = 40$ in [3].

V. ASIC IMPLEMENTATIONS AND PERFORMANCE EVALUATION

The proposed Montgomery multiplier is synthesized for ASIC implementation using Synopsys Design Compiler, DesignWare building block libraries, and IBM 90-nm and 0.13-μm CMOS 9FLP standard-cell library. The SRAMs in the 8 K/12 K-bit design are synthesized using Synopsys DesignWare library. The SRAMs in the 48 K-bit design are generated by the memory compiler. The 8 K/12 K-bit design requires 320 K-bit SRAMs. The 48 K-bit design utilizes 1.46 M-bit SRAMs that is about 4.6 times more. Owing to the optimization by the memory compiler, the area of SRAMs in 48 K-bit design is only about 2.8 times larger than the 8 K/12 K-bit design, as shown in Table III.

Currently, there are few, if any, ASIC reports on RSA designs with key size of 12 K or larger. Most of the existing ASIC implementations are designed for RSA with key size of 1024 or 2048 bits, which are scalable to 4 K or 8 K bits. In order to compare the *throughput* among these designs, we consider the throughput per area-delay product called bit multiplications

TABLE IV
THROUGHPUT AND EFFICIENCY COMPARISON WITH EXISTING RSA DESIGNS

Ref	Technology	Area (gates)	Period (ns)	MulMod/s	Key Size	$MPGPS$	$MPGPC$
[7]	0.5 μm CMOS	156K	20.0	94.2K	1,024	1.27 M	0.025
[8]	0.18 μm CMOS	148K	2.2	438.6K	1,024	6.2 M	0.014
[5]	0.13 μm CMOS	139K	2.0	648.6K	1,024	9.8 M	0.020
[9]	0.13 μm CMOS	110K	2.21	866K	1,024	16.5 M	0.036
12,288-bit RSA	90 nm CMOS	5,100K	2.1	231.4K	12,288	13.7 M	0.028
12,288-bit RSA	0.13 μm CMOS	5,100K	3	162.6K	12,288	9.6 M	0.028
49,152-bit RSA	90 nm CMOS	10,700K	2.1	81K	49,152	36.6 M	0.077
49,152-bit RSA	0.13 μm CMOS	10,700K	3	56.9K	49,152	25.7 M	0.077

per gate per second (MPGPS), which measures the number of bit multiplications that each gate of hardware can complete within a second. Although the designs in [5] and [7]–[9] use different optimization strategies to improve efficiency, they all complete the same computations as in the original interleaved Montgomery algorithm. Hence, we use the original interleaved Montgomery algorithm as a standard to estimate the number of bit multiplications. For bit size M, that is $2M^2 + M$ bit multiplications, and N_{mul} is the number of modular multiplications processed. MPGPS is expressed as

$$\text{MPGPS} \approx \frac{2M^2 * N_{\text{mul}}}{\text{Gates} \cdot \text{Seconds}}. \tag{4}$$

Different designs in Table IV use different CMOS process technologies. For the evaluation of *efficiency* among different designs, we need to compare them at the same clock frequency in terms of bit multiplications per gate per cycle (MPGPC). That is

$$\text{MPGPC} \approx \frac{2M^2 * N_{\text{mul}}}{\text{Gates} \cdot \text{Seconds}} * \frac{1}{f}. \tag{5}$$

Table IV is a summary for previous works and the proposed implementations. In reference to the complexity analysis in Fig. 1, the FFT-based method has slightly higher computational complexity for key size of 8 K bits and about the same for 12 K bits. The hardware implementation for 8 K/12 K-bit RSA shows similar performance results when compared to previous RSA designs in the literature. However, our 48 K-bit RSA design outperforms all others in both measurements of MPGPS and MPGPC. In fact, results in Table IV matches well with the complexity analysis in Fig. 1. As the public key size increases, the FFT-based method has more advantages. For very large key size RSA or leveled FHE scheme using 48 K-bit modular multiplication, the FFT-based method outperforms the Montgomery multiplication significantly. In Table III, a large amount of SRAMs are used to store the precomputed and intermediate results in our design. For instance, SRAMs occupy about 80% of the chip area in 48K-bit design. Future work will be focused on reducing the usage of SRAMs.

VI. CONCLUSION

In this brief, an FFT-based modular multiplication and exponentiation algorithm and its hardware architecture have been proposed for RSA cryptosystem with very large key size. ASIC implementations of the 8 K/12 K-bit and 48 K-bit RSA cryptosystem are presented, and their performances are compared with the previous works. Complexity analysis shows that the proposed method has more advantages when the key size becomes larger. The 48 K-bit RSA design outperforms others with respect to throughput and efficiency.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
[2] C. Gentry and S. Halevi, "Implementing Gentry's fully-homomorphic encryption scheme," in *Advances in Cryptology-EUROCRYPT*. Berlin, Germany: Springer-Verlag, 2011, pp. 129–148.
[3] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *Advances in Cryptology-CRYPTO*. Berlin, Germany: Springer-Verlag, 2012, pp. 850–867.
[4] P. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, 1985.
[5] 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. (VLSI) Syst.*, vol. 16, no. 9, pp. 1151–1161, Sep. 2008.
[6] C. P. Rentería-Mejía, V. Trujillo-Olaya, and J. Velasco-Medina, "Design of an 8192-bit RSA cryptoprocessor based on systolic architecture," in *Proc. 8th SPL*, 2012, pp. 1–6.
[7] T.-W. Kwon, C.-S. You, W.-S. Heo, Y.-K. Kang, and J.-R. Choi, "Two implementation methods of a 1024-bit RSA cryptoprocessor based on modified Montgomery algorithm," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2001, vol. 4, pp. 650–653.
[8] Q. Liu, F. Ma, D. Tong, and X. Cheng, "A regular parallel RSA processor," in *Proc. 47th MWSCAS*, 2004, vol. 3, pp. iii-467–iii-470.
[9] S.-R. Kuang, J.-P. Wang, K.-C. Chang, and H.-W. Hsu, "Energy-efficient high-throughput Montgomery modular multipliers for RSA cryptosystems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 11, pp. 1999–2009, Nov. 2013.
[10] A. Schönhage and V. Strassen, "Schnelle multiplikation grosser zahlen," *Computing*, vol. 7, no. 3, pp. 281–292, 1971.
[11] W. Wang and X. Huang, "A novel fast modular multiplier architecture for 8,192-bit RSA cryposystem," in *Proc. IEEE HPEC Conf.*, 2013, pp. 1–5.
[12] N. Emmart and C. C. Weems, "High precision integer multiplication with a GPU using Strassen's algorithm with multiple FFT sizes," *Parall. Process. Lett.*, vol. 21, no. 3, pp. 359–375, Sep. 2011.
[13] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology (CRYPTO 86)*. Berlin, Germany: Springer-Verlag, 1987, pp. 311–323.
[14] 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.
[15] L. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 39, no. 5, pp. 312–316, 1992.
[16] H. Lo, M. Shieh, and C. Wu, "Design of an efficient FFT processor for DAB system," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2001, vol. 4, pp. 654–657.
[17] L. Jia, Y. Gao, and H. Tenhunen, "A pipelined shared-memory architecture for FFT processors," in *Proc. 42nd IEEE Midwest Symp. Circuits Syst.*, 1999, vol. 2, pp. 804–807.