Towards ideal lattice-based cryptography on ASIC : a custom implementation of number theoretic transform

Ho, Truong Phu Truan; Chang, Chip-Hong

2018

Ho, T. P. T., & Chang, C.-H. (2018). Towards ideal lattice-based cryptography on ASIC : a custom implementation of number theoretic transform. Proceedings of the 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP).
doi:10.1109/ICDSP.2018.8631681

<https://hdl.handle.net/10356/145855>

<https://doi.org/10.1109/ICDSP.2018.8631681>

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The published version is available at:
<https://doi.org/10.1109/ICDSP.2018.8631681>

Downloaded on 26 Apr 2023 16:16:06 SGT

Towards Ideal Lattice-Based Cryptography on ASIC: A Custom Implementation of Number Theoretic Transform

Truong Phu Truan Ho and Chip-Hong Chang

School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

Abstract—Ideal lattice-based cryptography has received considerable attention recently due to its proven quantum resistant property and its application in homomorphic encryption. Although efficient implementation of ideal lattice-based cryptography on various software and reconfigurable hardware platforms has been widely studied, its ASIC design remains unexplored. This paper presents the first custom implementation of Number Theoretic Transform towards accelerating ideal lattice-based cryptography. Compared with previous architectures on reconfigurable hardware, for large parameter sets, our custom architecture has substantially reduced the latency and increases the throughput of lattice-based cryptography by 28.38% to 54.85%.

I. INTRODUCTION

Nowadays, a large majority of digital devices and applications are being secured under the hardness of classical hard problems, such as integer factorization (RSA) and discrete logarithm (ECC). However, recent development in quantum technology has posed serious threats to systems whose security are guaranteed by RSA and ECC, since they can be solved in polynomial time by using Shor algorithm [1]. Though it is not clear whether a feasible quantum computer can be built in the near future, it is necessary to invest effort into the research of quantum-resistant cryptography to cope with the unpredictable advancement of quantum technology, rapid improvement of cryptanalysis and major strides in computing power.

Ideal lattice-based cryptography, proven to be immune from quantum attacks [2], is being considered a promising candidate to replace the vulnerable cryptosystems. Since its introduction, many cryptosystems have been proposed and experimented for commercial products, such as key exchange protocol [3] and digital signature [4]. Besides, ideal lattice is also popular for its usage in building efficient homomorphic encryption schemes [5]. Unlike conventional encryption schemes where encrypted data is required to be decrypted for further processing, homomorphic encryption allows arbitrary computation directly on encrypted data, which is a significant breakthrough in cryptography. In cloud computing, this important feature can be used advantageously to safeguard the secret of the users from the cloud service providers, thus preserving the confidentiality of cloud computing.

Several implementations of ideal lattice-based cryptography have been proposed, targeting various software and hardware platforms [6-9]. However, hardware architectures targeting ideal lattice-based cryptography have

been reported only on reconfigurable devices, leaving ASIC-based designs unexplored. This is because ideal lattice-based cryptography is in its nascent stage, and its parameter selection is still an open problem. With recent governmental warnings of quantum technology and calling for the standardization of post-quantum cryptography [10], it is crucial to start exploring and evaluating the designs of ideal lattice-based cryptography on ASIC platform.

In this work, we take a step forward towards the ASIC design of ideal-lattice based cryptography with the first custom implementation of the Number Theoretic Transform (NTT) for polynomial multiplication - the most critical and computationally intensive operation in ideal lattice-based cryptography. In our design, we make use of the algorithmic properties of either Barrett or Montgomery multiplication algorithm and present two simplified modulo multipliers. Synthesis results show that the butterfly circuit designed with the simplified Montgomery multiplier has lower latency than the one designed with the simplified Barrett multiplier but costs more area. To increase the throughput, two parallel half-sized dual-port memories are employed in our NTT design and pipeline flip-flops are inserted into the butterfly circuits according to the access time of the memories. When compared with previous NTT designs on FPGAs, for large parameter sets, our custom design has not only the lower latency but also significantly higher throughput.

II. NUMBER THEORETIC TRANSFORM FOR IDEAL LATTICE-BASED CRYPTOGRAPHY

Ideal lattice is defined as a general lattice with special algebraic structure. It corresponds to ideals in the ring $R = \mathbb{Z}_M[x]/f(x)$, where M is a prime number and $f(x)$ is an irreducible polynomial of degree N . Simply speaking, R can be viewed as the set of polynomials whose degrees are at most $N - 1$ and their coefficients can take values in $[0, M)$. From security perspective, the special algebraic structure reduces the key size growth from quadratic complexity $O(n^2)$ to linear complexity $O(n)$ [2]. Consequently, cryptographic schemes based on ideal lattice are more efficient than those schemes that are based on general lattice [6].

For practical reasons, ideal lattice-based cryptosystems are configured to operate on the ring $R = \mathbb{Z}_M[x]/(x^N + 1)$, where N is a power of 2. When $M \equiv 1 \pmod{2N}$, there exists

a $2N$ -th root of unity $\psi_{2N} \in \mathbb{Z}_M$ such that $\psi_{2N}^{2N} \equiv 1 \pmod{M}$. Such root of unity reduces the complexity of polynomial multiplication, which is the most crucial and computationally intensive operation in R , by using NTT. The application of NTT in polynomial multiplication is similar to the well-known fast Fourier transform, except that the integer root of unity ψ_{2N} replaces the complex root of unity, and the modulo M arithmetic replaces the floating-point arithmetic. To the best of our knowledge, the most optimized NTT is described in [6], which is presented in the pseudo code of Algorithm 1 below. The essential optimizations include the removal of bit reversal step and halving the number of required NTT points. These optimizations are detailed in [6].

Algorithm 1 Optimized NTT Algorithm [6]

Precondition: Store N powers of ψ_{2N} in bit-reversed order vector S

Input: Coefficients $a[0], a[1], a[2], \dots, a[N-1]$ of polynomial $a(x)$

Procedure:

```

    m = 1;
    k = N/2;
    while m < N do
        for i = 0 to m - 1 do
            j_first = 2 * i * k;
            j_last = j_first + k - 1;
             $\Psi_i = S[m + i]$ ;
            for j = j_first to j_last do
                l = j + k;
                 $a[j] = a[j] + a[l] \times \Psi_i \pmod{M}$ ;
                 $a[l] = a[l] - a[j] \times \Psi_i \pmod{M}$ ;
            end for
        end for
        m = 2 * m;
        k = k/2;
    end while
    return  $a(x)$ ;

```

III. SIMPLIFIED BARRETT'S AND MONTGOMERY'S MULTIPLIERS

The modulo M multiplication required in NTT algorithm is an expensive operation. For a general modulus M , two modulo multiplication algorithms, Barrett [11] and Montgomery [12], are commonly used. Barrett's method (see Algorithm 2) uses a precomputed constant to approximate the reciprocal of M . Using this constant, costly division by M can be replaced by a single multiplication and some right shifts. In Montgomery's method (see Algorithm 3), division and modular reduction by a power of two are free and thus, there is no expensive operation in this algorithm. Since the result of Montgomery's method is not $AB \pmod{M}$ but $AB2^{-n} \pmod{M}$, forward transformation into the Montgomery domain at the start of NTT and backward transformation at the end of NTT are necessary [7]. In our proposed architecture, this transformation is eliminated by modifying the N powers of ψ_{2N} stored in the bit-reversed order vector S .

The integer multiplications (Steps 1, 3, and 5 of Algorithms 2 and 3) of Barrett and Montgomery algorithms

are usually implemented using the fast DSP multipliers on FPGAs. In most cases, the bit-width of the required multiplication is not the same as the bit-width of the DSP multipliers, which leads to inefficient computations. The algorithmic properties of either Barrett's or Montgomery's method can be more efficiently exploited to optimize the design in ASIC implementation. Fig. 2 and Fig. 3 illustrate our proposed Barrett and Montgomery modulo M multipliers, respectively.

Algorithm 2 Barrett Modulo Multiplication

Input: A, B, M where $A, B < M$, $2^{n-1} \leq M < 2^n$

Precomputation: $C = \lfloor 2^{2n+3}/M \rfloor$

Output: $R = AB \pmod{M}$

Procedure:

1. $P = AB$;
  2. $T_0 = \lfloor P / 2^{n-2} \rfloor$;
  3. $T_1 = T_0 \times C$;
  4. $Q = \lfloor T_1 / 2^{n+5} \rfloor$;
  5. $T_2 = Q \times M$;
  6. $R = P - T_2$;
  7. if $R \geq M$ then $R = R - M$;
  8. return R ;
-

Algorithm 3 Montgomery Modulo Multiplication

Input: A, B, M where $A, B < M$, $2^{n-1} \leq M < 2^n$

Precomputation: $C = -M^{-1} \pmod{2^n}$

Output: $R = AB2^{-n} \pmod{M}$

Procedure:

1. $P = AB$;
  2. $T_0 = P \pmod{2^n}$;
  3. $T_1 = T_0 \times C$;
  4. $Q = T_1 \pmod{2^n}$;
  5. $T_2 = Q \times M$;
  6. $R = (P + T_2) / 2^n$;
  7. if $R \geq M$ then $R = R - M$;
  8. return R ;
-

Considering the computation of T_0 and Q in Barrett's method (see Algorithm 2). Since both T_0 and Q are obtained from a rounded down division by a power of 2, a truncated multiplier design can be used instead of a slower, larger and more power hungry full width multipliers. However, as they introduce errors into T_0 and Q , more number of subtractions by M will be required at Step 7 to bring R into the range $[0, M)$. Therefore, in our design, T_0 and Q are implemented by discarding $n-2$ and $n+5$ least significant bits (LSBs), respectively from the output of a full-width multiplier. The arithmetic units for Steps 5 and 6 can be simplified by considering the fact that after Step 6, R is at most $2M - 1$, which is an $(n+1)$ -bit number. For Step 6, instead of a $2n$ -bit subtractor, an $(n+1)$ -bit subtractor is implemented. The inputs to the $(n+1)$ -bit subtractor are the $n+1$ LSBs of P and T_2 . Thus, for Step 5, the logic required for the computation of the $n-1$ most significant bits (MSBs) of T_2 can be saved since they are not needed. This results in the proposed

“reduced” Barrett’s multiplier shown in Fig. 1 without incurring additional subtractions of M in Step 7.

Considering the computation of T_0 in Montgomery’s method (see Algorithm 3). It is not possible to reduce the multiplier design by discarding the n MSBs of P since those MSBs are required at Step 6. On the other hand, Q can be computed by a reduced multiplier since the n MSBs of T_1 are not required for the subsequent steps. Therefore, Step 3 and 4 can both be realized in the reduced multiplier as shown in Fig. 2 without a T_1 bus. Step 6 is normally implemented by a $2n$ -bit adder. However, by taking advantage of the fact that the division in Step 6 is exact [12], this step can be sped up as follows. Since the n LSBs of the sum of P and T_2 are all zeros, R is either equal to $P/2^n + T_2/2^n + 1$ or $P/2^n + T_2/2^n$, depending on the carry generated by the sum $P \bmod 2^n + T_2 \bmod 2^n$. In our proposed simplified design, both sums $P/2^n + T_2/2^n + 1$ and $P/2^n + T_2/2^n$ are evaluated in parallel and the correct value of R in Step 6 is determined by the output from the carry generation logic (CGL) block. The delay for this step is reduced from the delay of a $2n$ -bit adder to the delay of an n -bit adder and a multiplexer.

Fig. 1: Hardware architecture of simplified Barrett multiplier.

Fig. 2: Hardware architecture of simplified Montgomery multiplier.

Fig. 3: Hardware architecture of (a) modulo adder and (b) modulo subtractor.

Figs. 3(a) and (b) show the implementation of the modulo M adder and modulo M subtractor, respectively. The inputs of the adder are three n -bit numbers A , B and M such that $A, B < M$. Since $0 \leq A + B \leq 2M - 2$, only a single subtraction by M is sufficient to keep the result in the range $[0, M)$. The inputs of the subtractor are three n -bit numbers A , B and M such that $A, B < M$. Since $-M + 1 \leq A - B \leq M - 1$, only a single addition by M is sufficient to keep the result in the range $[0, M)$.

IV. PROPOSED NTT ARCHITECTURE

Fig. 4 shows our proposed hardware architecture for NTT, which is based on Algorithm 1. The core components include a butterfly circuit built from the modulo arithmetic units presented in Section III, a ROM (S) to hold N precomputed powers of ψ_{2N} , a controller, and two RAMs, RAM_0 and RAM_1 . Each RAM holds $N/2$ coefficients. The butterfly circuit receives two coefficients and generates two new values per clock cycle. Storing all N coefficients in a single RAM will lead to idle cycles in the butterfly circuit. Modern RAMs usually support dual-port access, which means that two values can be read/written in one clock cycle. Since the indices of two coefficients entering the butterfly always have different parities [13], by having two dual-port RAMs working in parallel, the butterfly circuit can be executed continuously with no idle cycle. Moreover, two parallel RAMs, each of half the size of a single RAM, can operate at a higher operating frequency than a single RAM.

Fig. 4: Hardware architecture for the computation of NTT.

As mentioned in Section II, if fast Montgomery multiplier is used to build the butterfly circuit, forward transformation into the Montgomery domain is required at the start of the NTT and backward transformation is required at the end of the NTT. It is observed that the inputs of every modulo multiplication within the NTT are a coefficient and a

precomputed value loaded from S . For the butterfly circuit built with Montgomery multiplier, we can change every constant stored in S to its value scaled by $2^n \bmod M$ to eliminate these transformations. Let b be a coefficient and $\Psi' = \Psi 2^n \bmod M$ be a value loaded from the scaled S . Applying Montgomery multiplication on b and Ψ' gives us the desired result of $b\Psi \bmod M$ because

$$b\Psi 2^{-n} \bmod M = b\Psi 2^n 2^{-n} \bmod M = b\Psi \bmod M \quad (1)$$

Hence, no transformation is needed.

V. EVALUATION AND COMPARISON

From Fig. 2 and Fig. 3, the simplified Barrett multiplier costs approximately two full n -bit by n -bit multipliers, one reduced n -bit by n -bit multiplier, two n -bit subtractors, and one multiplexer while the area cost of the simplified Montgomery multiplier is approximately two full n -bit by n -bit multipliers, one reduced n -bit by n -bit multiplier, three n -bit subtractors (the cost of an adder and the cost of a subtractor are approximately equal), two multiplexers and a CGL block. The delay of the simplified Barrett multiplier can be approximated by the sum of the delays of two full n -bit by n -bit multipliers, one reduced n -bit by n -bit multiplier, two n -bit subtractors, and one multiplexer. According to Algorithm 3, since T_0 is obtained from the n LSBs of P , the total delay of the Montgomery multiplier can be estimated by the sum of the delays of one full n -bit by n -bit multiplier, two reduced n -bit by n -bit multipliers, two n -bit subtractors and two multiplexers. Based on gate-level component estimation, the butterfly circuit built from the simplified Barrett multiplier is smaller and slower than the butterfly circuit built from the simplified Montgomery multiplier.

The butterfly circuits based on the simplified Barrett and Montgomery multipliers are modeled by Verilog codes, synthesized and mapped to the TSMC 40nm CMOS process technology using Synopsys Design Compiler. The silicon area and critical path delay are shown in Table I. The synthesis results validate their relative cost and performance estimated by the above technology-independent complexity analysis at component-level.

Table I: Delay (D) and area (A) of the butterfly circuit constructed with either the simplified Barrett or Montgomery multiplier.

N	M	Butterfly Barrett		Butterfly Montgomery	
		D (ns)	A (μm^2)	D (ns)	A (μm^2)
256	7681	3.81	9864.88	3.27	12506.89
256	1049089	4.68	20952.29	4.00	24359.75
512	12289	4.00	10981.44	3.43	13262.76
512	5941249	4.87	22098.86	4.28	23912.87
1024	1061093377	5.38	32578.19	4.71	34969.64
2048	$2^{57}+25 \times 2^{13}+1$	6.79	108821.03	6.05	112465.01

Since the parameter selection of ideal-lattice based cryptography is still an open problem, we decide to compare our design with those that can support a wide range of parameters. To the best of our knowledge, only architectures in [8] and [9] feature a generic modular reduction which is based on an iterative shift-and-subtract division algorithm and can be used to implement a large variety of ideal-lattice

based schemes. The performances of the architectures in [8] and [9] on FPGA, the performances of their reimplementation on ASIC, and the performances of our design under different parametric combinations of N and M are shown in Figs. 5, 6, 7 and 8, respectively. The RAMs and ROM in Fig. 4 of our proposed implementations are generated by the ARM memory compiler with TSMC 40nm technology. Pipelined flip-flops are inserted into the synthesized butterfly circuit according to the maximum operating frequency reported by the memory compiler to maximize the throughput with an optimal number of pipeline stages.

Fig. 5: Comparison of normalized synthesized area of ASIC reimplementation of FPGA designs in [8] and [9] and our designs with either Barrett or Montgomery multiplier under different combination of (N, M) .

Fig. 6: Comparison of frequency of design [8] and [9] on FPGA, their ASIC reimplementation, and our designs with either Barrett or Montgomery multiplier under different combination of (N, M) .

Fig. 7: Comparisons of required cycles per NTT of designs in [8] and [9] and our designs with either Barrett or Montgomery multiplier under different combination of (N, M) .

Fig. 8: Comparisons of normalized NTT operation per second of design [8] and [9] on FPGA, their ASIC reimplementations, and our designs with either Barrett or Montgomery multiplier under different combination of (N, M) .

As can be seen from Table I, the butterfly circuit with Montgomery multiplier is faster than that with Barrett multiplier. However, this difference does not make a noticeable difference in the cycle counts in Fig. 7 since the rounded ratio between the delay of the butterfly circuit and the minimum clock period of the memories is the same regardless of the type of multiplier. Thus, it can be concluded that NTT with Barrett multiplier is more efficient while NTT with Montgomery multiplier may offer better performance if the minimum of clock period of memories is small enough to reflect the difference between two types of multiplier. From Fig. 6, compared to FPGA implementation, ASIC reimplementations of the designs of [8] and [9] have increased the operating frequency and leads to significant improvement in throughput, as shown in Fig. 8. From Figs. 5, 6 and 7, our custom architectures, in general, achieve lower latency and higher operating frequency at the cost of higher silicon area compared to the ASIC reimplementations of the designs of [8] and [9]. For small (N, M) such as $(256, 7681)$, compared to ASIC reimplementations of FPGA design, our custom designs achieve 1.26% throughput improvement with at least 7% silicon area overhead. For large (N, M) such as the last three pairs, our custom designs are clearly preferable over ASIC reimplementations of FPGA design: throughput is increased from 28.38% to 54.85% at the cost of at most 7.5% hardware overhead. Therefore, the designs of [8] and [9] are more suitable for small parameter sets while our custom designs are more efficient for large parameter sets which are often found in time-critical homomorphic encryption.

VI. CONCLUSION

A custom implementation of NTT by optimizing the modulo multiplier designs of Barrett and Montgomery multiplication algorithms has been presented. The simplified Barrett multiplier exploits truncation without incurring extra modulo reduction operation while the simplified Montgomery multiplier eliminates the conversion overheads. Experimental results show that our custom designs are more suitable for high-performance application of ideal-lattice based cryptography. The throughput is further increased by

designing the butterfly circuit around two parallel half-sized dual-port RAMs with no idle cycle.

ACKNOWLEDGMENT

This work was supported by the Singapore Ministry of Education MOE AcRF Tier I grant no. MOE 2018-T1-001-131.

REFERENCES

- [1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc. 35th Annu. IEEE Symp. Foundations Comput. Sci.*, Santa Fe, NM, USA, pp.124-134, Nov. 1994.
- [2] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proc. EUROCRYPT 2010*, French Riviera, France, pp. 1-23, May 2010.
- [3] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila, "Post-quantum key exchange for the TLS protocol from the ring learning with errors problem," in *Proc. 2015 IEEE Symp. Security and Privacy*, San Jose, CA, USA, pp. 553-570, May 2015.
- [4] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *Proc. CRYPTO 2013*, Santa Barbara, CA, USA, pp. 40-56, Aug. 2013.
- [5] C. Gentry, A. Sahai, B. Waters, "Homomorphic encryption from learning with errors: conceptually simple, asymptotically faster, attribute based," in *Proc. CRYPTO 2013*, Santa Barbara, CA, USA, pp. 75-92, Aug. 2013.
- [6] T. Poppelmann, T. Oder, and T. Guneyssu, "High performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers," in *Proc. LATINCRYPT 2015*, Guadalajara, Mexico, pp. 346-365, Aug. 2015.
- [7] Z. Liu, T. Poppelmann, T. Oder, H. Seo, S. S. Roy, T. Guneyssu, J. Grobischadl, H. Kim, and I. Verbauwhede, "High performance ideal lattice-based cryptography on 8-bit AVR microcontrollers," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 4, Article 117, pp. 1-24, Sept. 2017.
- [8] T. Poppelmann and T. Guneyssu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Proc. LATINCRYPT 2012*, Santiago, Chile, pp. 139-158, Oct. 2012.
- [9] T. Poppelmann and T. Guneyssu, "Towards practical lattice-based public key encryption on reconfigurable hardware," in *Proc. SAC 2013*, Burnaby, BC, Canada, pp. 68-85, Aug. 2013.
- [10] T. Oder, T. Guneyssu, F. Valencia, A. Khalid, M. O'Neill, and F. Regazzoni, "Lattice-based cryptography: From reconfigurable hardware to ASIC," in *Proc. ISIC 2016*, Singapore, pp. 1-4, Dec. 2016.
- [11] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Proc. Adv. Cryptolog.*, California, USA, pp. 311-323, 1987.
- [12] P. Montgomery, "Modular multiplication without trial division," *Math. Computation*, vol. 44, no. 170, pp. 519-521, 1985.
- [13] J. Baek and K. Choi, "New address generation scheme for memory-based FFT processor using multiple radix-2 butterflies," in *Proc. ISOC 2008*, Busan, South Korea, pp. 273-276, Nov. 2008.