An FPGA Implementation of the NTRUEncrypt Cryptosystem

Abdel Alim Kamal and Amr M. Youssef
Concordia Institute for Information Systems Engineering
Concordia University, Montreal, Canada
{*a_kamala,youssef*}@ciise.concordia.ca

***Abstract*—The NTRU encryption algorithm, also known as NTRUEncrypt, is a parameterized family of lattice-based public key cryptosystems. Both the encryption and decryption operations in NTRU are based on simple polynomial multiplication which makes it very fast compared to other alternatives such as RSA, and elliptic-curve-based systems. Recently, the NTRU system has been accepted to the IEEE P1363 standards under the specifications for lattice-based public-key cryptography (IEEE P1363.1).**

In this paper, we investigate several hardware implementation options for the NTRU encryption algorithm. In particular, by utilizing the statistical properties of the distance between the non-zero elements in the polynomials involved in the encryption and decryption operations, we present an architecture that offers different area-speed trade-off and analyze its performance. A prototype for the proposed design is implemented using the virtex-E xcv1600e-8-fg860 FPGA chip.

I. Introduction

Traditional public key systems [8] can be categorized, based on the underlying hard problem, into discrete logarithm-based, such as Diffie-Hellman, DSA and their elliptic curve variants, or integer factorization-based such as the RSA algorithm. Despite the significant progress that has been made in implementing these systems, the computational cost associated with them remain some what expensive, especially in resource constrained environments. For example, a typical public key algorithm might involve about 500,000 32-bit multiplications which can take up to 5 seconds on low-end chips [6].

To address this problem, a new generation of public key systems based on second generation of hard problems, such as lattice reduction [5], is now being given further consideration. The NTRU public key cryptosystem [3] is one of these new generation systems that offer large speed improvement compared to RSA and elliptic curve counterparts. Recently, the NTRU system has been accepted to the IEEE P1363 standards under the specifications for lattice-based public-key cryptography (IEEE P1363.1) [2].

Previous works on the hardware implementations of the NTRUEncrypt algorithm include [1], published by NTRU cryptosystems Inc., in which an FPGA implementation of the NTRU encryption engine was described. Also, in [10], O'Rourke presented a scalable architecture to perform the NTRU multiplication and a unified architecture that uses Montgomery multiplication to provide support for NTRU and other cryptographic schemes. A scalable low power design for the NTRU polynomial multiplications was described in [7].

In this paper, we investigate several hardware implementation options for the NTRUEncrypt algorithm. In particular, by utilizing the statistical properties of the distance between the non-zero elements in the polynomials involved in the encryption and decryption operations, we present an architecture that offers different area-speed trade-off and analyze its performance.

The rest of the paper is organized as follows. The relevant details of the NTRU encryption algorithm is described in the next section. The proposed hardware architecture is analyzed in section III and compared to the naive implementation approach. Finally, our implementation results are given in section IV.

II. Description of the NTRUEncrypt algorithm

The NTRU encryption algorithm is a lattice-based public key cryptosystems that is parameterized by three integers: (N, p, q), where N is prime, gcd(p, q) = 1 and $p << q$. Let R, R_p, and R_q be the polynomial rings

$$R = \frac{\mathbb{Z}[x]}{x^N - 1}, R_p = \frac{\mathbb{Z}/p\mathbb{Z}[x]}{x^N - 1}, R_q = \frac{\mathbb{Z}/q\mathbb{Z}[x]}{x^N - 1}.$$

The product of two polynomials $a(x), b(x) \in R$ is given by

$$a(x) \star b(x) = c(x)$$

where

$$c_k = \sum_{i+j=k \pmod N} a_i b_{k-i}$$

For any positive integers d_1 and d_2, let $\tau(d_1, d_2)$ denote the set of ternary polynomials given by

$$\left\{ a(x) \in R \middle| \begin{array}{l} \text{a(x) has } d_1 \text{ coefficients equal to 1,} \\ \text{a(x) has } d_2 \text{ coefficients equal to -1,} \\ \text{a(x) has all other coefficients equal to 0} \end{array} \right\}$$

In what follows, we briefly describe the key generation, encryption and decryption operations in the NTRU cryptosystem [4] [5]:

A. Key Generation

- Choose private $f(x) \in \tau(d+1, d)$ that is invertible in R_q and R_p.
- Choose a private $g(x) \in \tau(d, d)$.

	N	p	q
Moderate Security	167	3	128
Standard Security	251	3	128
High Security	347	3	128
Highest Security	503	3	256

TABLE I
TYPICAL PARAMETER SETS FOR NTRU [9]

- Computes $F_q(x) = f^{-1}(x)$ in R_q and $F_p(x) = f^{-1}(x)$ in R_p.
- Compute $h(x) = f_q(x) \star g(x)$ in R_q.

The polynomial $h(x)$ is the user's public key. The corresponding private key is the pair $(f(x), F_p(x))$. Alternatively, the user can store only $f(x)$ and recompute $F_p(x)$ from it. The following steps denote the encryption operations for plaintext $m(x) \in R_p$.

B. Encryption

- Choose a random ephemeral key $r(x) \in \tau(d, d)$.
- Compute the ciphertext $e(x) = pr(x) \star h(x) + m(x) \bmod q$.

C. Decryption

- Compute $a(x) = f(x) \star e(x) \bmod q$.
- Centerlift $a(x)$ to $a(x) \in R$
- Compute $m = F_p(x) \star a(x) \bmod p$.

Note that by choosing $f(x) = 1 + pf_1(x)$, where $f_1 \in R$ in the key generation step, the polynomial multiplication in the last decryption step is eliminated since we will have $F_p = 1 \bmod p$.

The IEEE P1363.1 standard presents a few typical parameter sets for NTRU. In this paper, for efficiency reasons, we focus on the set of parameters where q is in the form of 2^n because of the simplicity of the modular operations for such choices of q , where the effect of the modular operation can be achieved by simply truncating the results to n bits. Table I shows some typical choices for (N, p, q) for different security levels.

III. HARDWARE IMPLEMENTATION

In what follows, we assume that at the beginning of the encryption operation, the public key $h(x) \in R_q$ is loaded into the chip through N parallel I/O PINS in $T_{Ld(h)} = log_2(q)$ clock cycles. Similarly, for each plaintext $m(x) \in R_p$, an ephemeral key $r(x) \in \tau(d, d)$ will be loaded into the chip in $T_{ld(r)} = \lceil log_2(3) \rceil$ clock cycles and $m(x)$ will be loaded into the chip in $T_{Ld(m)} = \lceil log_2(p) \rceil$ clock cycles. Furthermore, $T_{op(e)} = log_2(q)$ clock cycles will be required to output the ciphertext $e(x) \in R_q$ using N parallel I/O PINS.

From the description of the encryption and decryption operations above, it is clear that the most time consuming part in both operations is the convolution product, T_{conv}, required to compute $r(x) \star h(x)$ and $f(x) \star e(x)$ during the encryption and decryption operations, respectively.

In general, the convolution product $a(x) \star b(x)$ requires N^2 multiplications and additions where each of the c_k terms, $k = 0 \cdots N-1$, can be evaluated in one clock cycle using a shift, multiply and add operation. However, because of the ternary nature of both $f(x) \in \tau(d+1, d)$ and $r(x) \in \tau(d, d)$, these convolution can be evaluated without any multiplications. Furthermore, the time required to load h can be ignored when encrypting a large number of blocks since h is loaded only once into the chip and remains unchanged throughout the encryption process. Hence, and by ignoring the few extra clock cycles required by the control circuity, a straightforward implementation of the above encryption algorithm would require approximately

$$\begin{array}{ccccccccc} T_{Ld(r)} & + & T_{Ld(m)} & + & T_{conv} & + & T_{op(e)} = \\ \lceil log_2(3) \rceil & + & \lceil log_2(p) \rceil & + & N & + & log_2(q) \end{array}$$

clock cycles to encrypt each plaintext block.

On the other hand, d is usually much smaller than N. For example, for NTRU with parameters $(N, p, q) = (251, 3, 128)$, we have $d = 36$, i.e., the number of non-zero coefficients in $r(x)$ is 72 [9] [11]. Thus if the locations of these non-zero elements in $r(x)$ are loaded into the chip,together with one bit indicating the value of each coefficient (i.e., 1 or -1), then T_{conv} can be significantly reduced from N to $2d$. To simplify the control circuity, we assume that the j^{th}-bits of the locations of the non-zero elements in r, $j = 1, \cdots \lceil log_2(N) \rceil$ are loaded in one clock cycle. Thus we have $T_{ld(r)} = \lceil log_2(N) \rceil + 1$ and hence the number of clock cycles required to implement the encryption operation will be reduced to

$$\begin{array}{ccccccccc} T_{Ld(r)} & + & T_{Ld(m)} & + & T_{conv} & + & T_{op(e)} = \\ (\lceil log_2(N) \rceil + 1) & + & \lceil log_2(p) \rceil & + & (2d) & + & log_2(q). \end{array}$$

This reduction in T_{conv}, however, requires the availability of a hardware circuity that can be used to perform an arbitrary number of shifts, up to $N - 2d$, for the $h(x)$ coefficients in one clock cycle, e.g., a Barrel shifter. However, implementing $\lceil log_2(q) \rceil$ N-bit Barrel shifters would require a prohibitively large hardware.

A considerable reduction in this hardware complexity can be achieved if we only require the shifter to be able to shift the coefficients in $h(x)$ by a relatively small number of locations, $s << N$, in each clock cycles.

Figure 1 shows a typical implementation of such a circuit that can circularly shift its N bit inputs by up to s bits in one clock cycle.

Let d_i, $1 \leq i \leq 2d$, denote the location of the i^{th} non-zero element in $r(x)$. For convenience of notation, we set $d_0 = 0$. Then the number of clock cycles required to evaluate the convolution operation during the encryption operation is given by

$$T_{conv(s)} = \sum_{i=1}^{2d} \lceil \frac{d_i - d_{i-1}}{s} \rceil$$

Since these non-zero coefficients are distributed uniformly across the N terms of the corresponding polynomials, then, on average, we have

$$d_i - d_{i-1} \approx \frac{N}{2d} << N.$$

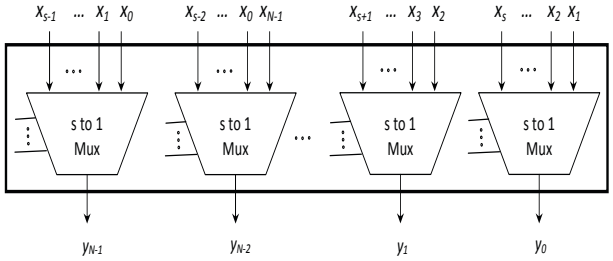

Fig. 1. Example of (N,s)-shifter

Throughout the rest of the paper we focus on the $(251, 3, 128)$-version of NTRU. Figure 2 shows how the average value of $T_{conv(s)}$ decreases with s, for $1000,000$ randomly generated polynomials $r(x) \in \tau(36, 36)$. Figure 3 shows the number of slices required to implement this (251,s)-shifter. From both figures, it is clear that choosing $s > 8$ offers a small marginal increase in throughput at the expense of a relatively large area overhead.

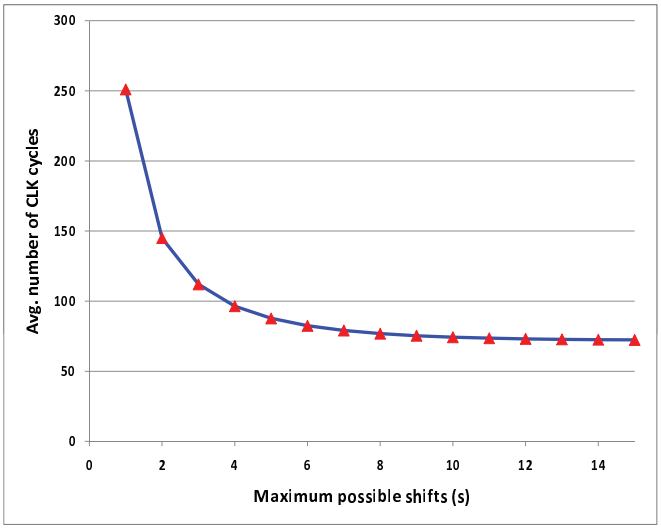

Fig. 2. Average value of $T_{conv(s)}$

Figures 4 and 5 show the histogram of $T_{conv(s)}$ for $1000,000$ randomly generated $r(x) \in \tau(36, 36)$ for $s = 4, 8$ respectively. The average, minimum and maximum values for $T_{conv(4)}$ are 96.4, 85 and 106, respectively. The corresponding values for $T_{conv(8)}$ are 76.85, 72, and 85.

Similar argument applies in optimizing the decryption algorithm except that calculating mod p in the final decryption step is non trivial. In this work, we investigated two methods to calculate the mod p operation. The first method is using the Mersenne primes algorithm [12] to calculate $a(x)$ mod p, in this method $a(x)$ can be split into sections each of length $log_2(p+1)$ bits and the addition of these sections is the $a(x)$ mod p. The second method is to use look-up tables (LUTs). This LUT approach is relatively efficient in case of small p.

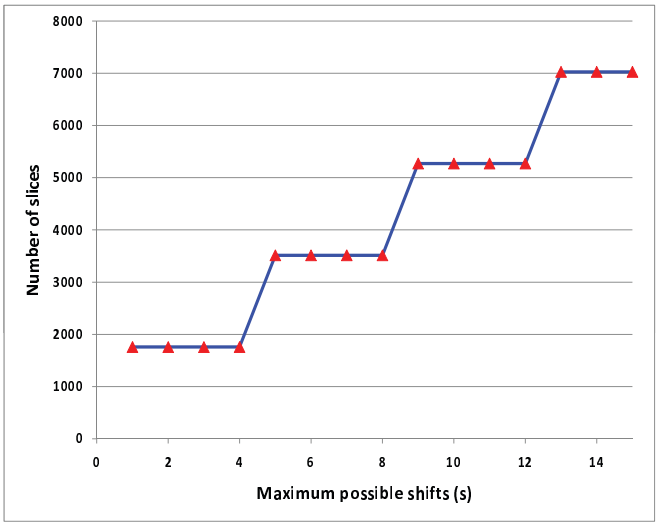

Fig. 3. The number of slices for (251,s)-shifter

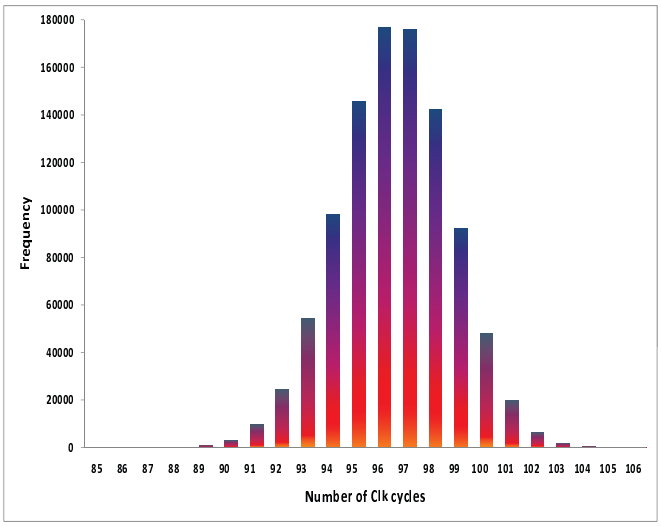

Fig. 4. Histogram of $T_{conv(4)}$

Note that choosing q in the form of 2^n eliminates the need for such circuits during the encryption operation.

IV. Implementation Results

The targeted hardware platform for our implementation is the xcv1600e-8-fg860 FPGA using Xilinx ISE 9.1i as the Syn-

	Modular reduction using Mersenne primes algorithm
Input:	an integer a, a Mersenne prime p
Output:	$b = a$ mod p
Step 1:	$b = a$
Step 2:	do while $b > p$
Step 3:	split b into sections $c_i \vert c_{i-1} \vert ... \vert c_1 \vert c_0$ each of length $log_2(p+1)$ bits
Step 4:	$b = c_i + c_{i-1} + ... + c_1 + c_0$

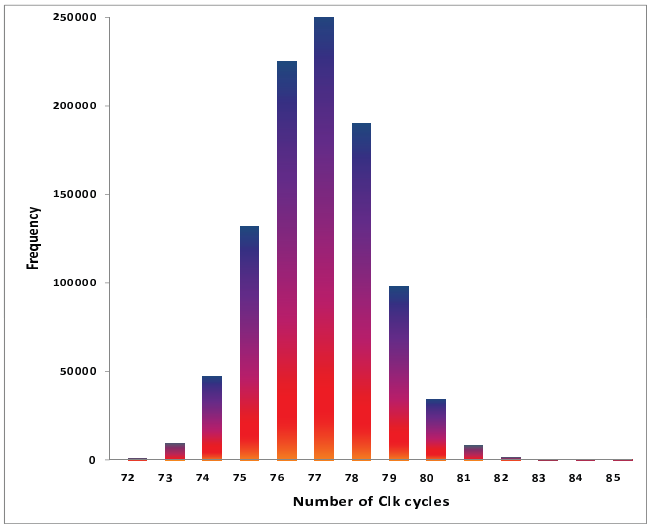

Fig. 5. Histogram of $T_{conv(8)}$

thesis, Translation, Mapping, Place & Route File Generation tool. The control unit for our encryption/decryption engine is implemented using a finite state machine data-path (FSMD) model. Simulations were conducted in ISE simulation model generated from the VHDL code to ensure performance was representative of the actual FPGA.

Table II shows the implementation results for the encryption-decryption engine using the naive approach for polynomial multiplication.

Table III shows the corresponding results when using $(251, s)$ shifters with $s = 4, 8$. For s=4, the average encryption throughput has increased from 51.22 to 134 Mbps, i.e., by about 161.62%, when using the Mersenne primes algorithm and from 51.22 to 129 Mbps, i.e., by 151.85% when using the LUT method. Similarly, for decryption, the average throughput has increased from 51.81 to 143.99 Mbps, i.e., by 177.92% when using the Mersenne primes algorithm and from 51.81 to 138.6 Mbps, with percentage 167.52%, when using the LUT method. On the other hand, the area cost has increased from 8973 to 13028 slices, i.e., by 45.19% when using the Mersenne primes algorithm and from 8648 to 12472, i.e., by about 44.22% when using the LUT method.

Similarly, for s=8, the average encryption throughput has increased by about 216.83% and the average decryption throughput has increased by about 241.75%. This occurs at the expense of about 60%-66% increase in the number of slices.

V. Conclusions

We have presented a hardware architecture that offers different area-speed trade-offs for the NTRUEncrypt algorithm and analyzed its performance. It would be interesting to explore this architecture for other choices of parameters, e.g., when q is not in the form of 2^n, as well as for the NTRUSign digital signature algorithm. It is also interesting to explore side-channel resistant implementation options for both the NTRUEncrypt and NTRUSign algorithms.

	Mersenne algorithm	LUTs
Device components report		
# of Slices	8973	8648
# of Slice FFs	4922	4922
# of 4-input LUTs	16273	15913
# of Roms (128x2-bits)	-	251
# of IOBs	506	506
Timing report		
Clk Freq.(MHz)	54.08	54.08
Throughput(Mbps)	51.22 Enc., 51.81 Dec.	

TABLE II
Implementation results using the naive polynomial multiplication algorithm

		s=4		s=8	
		Mersenne algorithm	LUTs	Mersenne algorithm	LUTs
Device components report					
# of Slices		13028	12472	14406	14352
# of Slice FFs		5424	4838	5469	5160
# of 4-input LUTs		24360	21654	27216	27292
# of Roms (128x2-bits)		-	251	-	251
# of IOBs		579	579	579	579
Timing report					
Clk Freq.(MHz)		61.61	59.31	61.61	62.33
Avg. Throughput (Mbps)	Enc.	134.00	129.00	161.34	163.22
	Dec.	143.99	138.6	176.03	178.09

TABLE III
Implementation results using the proposed approach for $s = 4$ and $s = 8$

References

[1] D. V. Bailey, D. Coffin, A. Elbirt, J. H. Silverman, and A. D. Woodbury, *NTRU in constrained devices,* in Workshop on Cryptographic Hardware and Embedded Systems - CHES, pp. 266-277, 2001.

[2] IEEE P1363.1: *Public-Key Cryptographic Techniques Based on Hard Problems over Lattices*, version D12, October 2008.

[3] J. Hoffstein, J. Pipher and J. Silverman, *NTRU: a ring based public key cryptosystem,* In Proc. of ANTS III, LNCS 1423, pp. 267288, Springer-Verlag, june 1998.

[4] J. Hoffstein and J. H. Silverman, *Optimizations for NTRU,* in Proc. of Public Key Cryptography and Computational Number Theory, de Gruyter, Warsaw, September 2000.

[5] J. Hoffstein, J. Pipher and J. H. Silverman, *An Introduction to Mathematical Cryptography*: Undergraduate Texts in Mathematics, Springer, 2008.

[6] B. Kaliski, *Considerations for New Public-key Algorithms*, Network Security Volume 2000, Issue 9, pp. 9-10, September 2000.

[7] J. P. Kaps, *Cryptography for ultra-low power devices,* Ph.D. dissertation, Worcester Polytechnic Institute, 2006.

[8] A J. Menezes, P. C. van Oorschot and S A. Vanstone *Handbook of Applied Cryptographic Research*, CRC Press, 1996.

[9] *The NTRU Public Key Cryptosystem - A Tutorial*, Available from $http://www.ntru.com/cryptolab/index.htm$.

[10] C. M. O'Rourke, *Efficient NTRU implementations,* Masters thesis, Worcester Polytechnic Institute, 2002.

[11] J. Pipher, *Lectures on the NTRU encryption algorithm and digital signature scheme,* Brown University, Grenoble, June 2002.

[12] K. Wilhelm, *Aspects of Hardware Methodologies for the NTRU Public-Key Cryptosystem*, M.Sc in computer engineering, Rochester Institute of Technology, Rochester, New York, February 2008.