SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks. (2008)
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/sec.39

NTRU-based sensor network security: a low-power hardware implementation perspective

Fei Hu[1], Kyle Wilhelm[2], Michael Schab[2], Marcin Lukowiak[2], Stanislaw Radziszowski[3] and Yang Xiao[4]*,†

[1]*Department of Electrical and Computer Engineering, The University of Alabama, U.S.A.*
[2]*Department of Computer Engineering, RIT, Rochester, New York, U.S.A.*
[3]*Department of Computer Science, Rochester Institute of Technology, Rochester, New York, U.S.A.*
[4]*Department of Computer Science, University of Alabama, U.S.A.*

Summary

Wireless sensor network security requires the cryptography software extremely low complex and energy efficient due to the limited memory and CPU capacity in a sensor. The NTRU (*N*th degree truncated polynomial ring) encrypt algorithm has been shown to provide certain advantages when designing low power and resource constrained systems, while still providing comparable security levels to higher complexity algorithms. Unlike the current works that build NTRU software in a chip, this research focuses on the hardware implementation of NTRU algorithms because hardware implementation has much higher execution speed than software implementation. In contrast to previous research, the focus is shifted away from specific optimizations but rather provides a study of many of the recommended practices and suggested optimizations with particular emphasis on polynomial arithmetic and parameter selection. Recommendations for algorithm and parameter selection are made regarding implementation in hardware with respect to the resources available. Copyright © 2008 John Wiley & Sons, Ltd.

KEY WORDS: sensor network security; NTRU; hardware design; cryptography algorithms

1. Introduction

Public-key cryptography can be used in both wired and wireless networks. A general assumption that can be made for wired network security is that the amount of security desired is the driving factor for the cryptosystem. In this case, the cryptosystem is allowed to use any required resources for any required amount of time, within practicality, to accomplish the desired security level. The increasing number of applications which transmit confidential data over insecure channels require that authentication is achieved through public/private key pair. Typically the sender's private key is used to generate a message authentication code (MAC) for the message and then the receiver can use the sender's public key to verify the source.

However, in wireless sensor networks, we have serious restriction on the amount of computational power, memory storage, gate area, and power that are allowed to be consumed by tiny sensors [1]. Existing public-key schemes have been found to be challenging in terms of resource consumption (e.g., RSA [2]) or in terms of power scalability (e.g., ECC [3]).

This research aims to use the NTRU (*N*th degree truncated polynomial ring) algorithm in sensor networks since NTRU has been claimed to be able to

*Correspondence to: Yang Xiao, Department of Computer Science, University of Alabama, U.S.A.
†E-mail: yangxiao@ieee.org

deliver security level similar to RSA or ECC at less computational effort and lower power consumption [4]. Once the session key is established with NTRU, AES [2] can be used for encryption and decryption of subsequent data [5].

Although NTRU encryption/decryption algorithms could be implemented in pure software such as in C/C++/Java [6], many sensor networks need to achieve real-time sensor data authentication and intrusion detection (typically the stream decryption time cannot go beyond 100 ms [2]). In order to speed up the security algorithms, hardware approach is necessary. This research will propose a series of optimizations in the NTRU circuit design to achieve required operation speed with ultra-low-power dissipation. To the best of our knowledge, there is no or very little research on NTRU-based security hardware for sensor networks.

The rest of this paper is organized as follows: Section 2 will briefly introduce the NTRU algorithms. The NTRU optimization strategies for sensor networks will be explained in Section 3. We then provide the RTL-level design (*via* VHL) in Section 4. Finally, Section 5 concludes this paper.

2. NTRU Algorithms

For the convenience of future discussions, we first briefly give the basic principle of lattice-based NTRU ciphers. For details and security analysis of NTRU, please refer to Reference [7]. The NTRU public key cryptosystem is based on ring theory and relies for its security on the difficulty of solving certain lattice problems. It uses a ring R and two (relatively prime) ideals p and q in R. Suppose f, g, r, e, and a are all ring polynomials. A standard implementation of NTRU uses the ring of convolution polynomials:

$$R = \frac{Z[X]}{(X^N - 1)}$$

and all polynomials have integer coefficients.

In most cases p and q are primes, with p much smaller than q. Most of subsequent computations are done mod p or mod q, and all polynomials are taken modulo $(X^N - 1)$. Multiplication in the ring R is sometimes referred to as *star multiplication* (see Reference [8] for details).

(1) *Key creation*: Suppose Bob creates a public key h by choosing elements f; $g \in R$, computing the mod q inverse f_q^{-1} of f, and setting:

$$h \equiv f_q^{-1} * g \pmod{q} \tag{1}$$

Bob's private key is the element f. Bob also precomputes and stores the mod p inverse f_q^{-1} of f.

(2) *Encryption*: In order to encrypt a plaintext message $m \in R$ using the public key h, Alice selects a random element $r \in R$ and forms the ciphertext:

$$e \equiv r * h + m \pmod{q} \tag{2}$$

(3) *Decryption*: In order to decrypt the ciphertext e using the private key f, Bob first computes:

$$a \equiv f * e \pmod{q} \tag{3}$$

Bob then chooses $a \in R$ to satisfy this congruence and to lie in a certain prespecified subset of R. He next does the mod p calculation $f_q^{-1} * a \pmod{p}$ and the value he computes is equal to m modulo p.

As we can see from above descriptions, the basic parameters of the NTRU cryptosystem are N, p, and q. The parameter N is used to define the degree of polynomials used in the convolution polynomial ring. The modulus p is defined as the small modulus and the modulus q is the large modulus, where p is much less than q. Most operations in the convolution ring will occur modulo q whereas the modulus p is used to reduce the random generation components used in encryption and to constrain the message space. Modular reduction of a convolution polynomial is performed by reducing each coefficient.

Although the NTRU algorithm has been scrutinized and peer reviewed by numerous parties, there have not been many widely published implementations in hardware. NTRU Cryptosystems Inc. has made public some of their own hardware design works [9,10], while other significant works include References [4,6] from the Worcester Polytechnic Institute. The most recent source of interest found was a software implementation that provided a possible increase in efficiency for polynomial multiplication [11].

Although hardware aspects have been considered in previously published works [4], there has been no known publication of an in-depth study of the NTRU system regarding implementation in sensor hardware. With the goal of introducing many of the hardware-related design issues, the general system will be examined, followed by specific considerations for design and modeling based on the IEEE 1363.1 draft standard. While only the aspects for design are

DOI:10.1002/sec

presented here, the following chapter will provide in-depth discussion and analysis.

For implementation of almost any system, it is often advisable to consider the primary operands and operations that are involved in the system. For the NTRU system, the primary operands are convolution polynomials or their integer coefficients. Operations to consider are addition, multiplication, and inversion of convolution polynomials and modular reduction and inversion of coefficients. Although these factors can be evaluated independently of the specific parameters used in the system, often it is more valuable to assess particular groups of parameters.

Depending on the choice of parameters, the length of a polynomial used in the NTRU system can range from around three thousand bits to well into the tens of thousands of bits. Although one of these polynomials may seem to fit trivially in the amount of memory that is commonly available to hardware systems, several polynomials are used in the system and the total amount of memory needed may be unachievable for some systems. In addition, access patterns should be taken into account when deciding on the method of operating on the polynomials, which is often dependent on the type of storage used. At times it may be better to pack coefficients in the minimum space required but there are also situations where it may be better to have padding between coefficients. Some hardware configurations may not support arithmetic on operands above a certain bit length or may be less efficient when operating on bit lengths that are between standard operand boundaries. For example, consider a configurable piece of hardware which has embedded multiplier units accepting 16-bit operands. If the system parameters allow for a minimum coefficients size of 10 bits, then an implementation using only 10 bits might create an implementation using configurable logic or lookup table version of a 10-bit multiplier in order to keep the more efficient embedded resources available. If the option is available, it is possible to force the operation onto the embedded resource anyway, but it would also be possible to pad the coefficients out to 16 bits to cause a migration into the embedded resource. Additional details concerning storage methods and memory requirements can be found in the following chapter.

Following almost every operation in the NTRU system, the result is reduced either modulo p or q. The modular reduction of a polynomial being defined by reducing each coefficient, there are N reductions per operation. Assuming that a general modulus is used, a general reduction algorithm would have to be used, but often the form of the chosen modulo allows for better performance. In hardware, power of two reductions is performed at no cost by truncating the result of an operation or can be performed at minimal cost using a masking operation.

NTRU Cryptosystems Inc. has set out to define standard interfaces to provide definitions on secure and efficient ways to implement an NTRU system. Although NTRU Cryptosystems is active in numerous bodies, the two most frequently referenced are the Institute of Electrical and Electronics Engineers (IEEE) and the Consortium for Efficient Embedded Security (CEES). Collaboration with the CEES has generated the Efficient Embedded Security Standard #1 (EESS) [12]. Current work with the IEEE has generated the 1363.1 draft standard [13], a work that is still in progress. The scope differs between the two. The EESS document seeks to provide a standard implementation interface for the NTRU system in wired and wireless applications. The EESS is also targeted more toward applications using a microcontroller and so limits the scope of the parameters recommended. The IEEE 1363.1 document is more of a reference for techniques, theoretic background, and security considerations. Due to the difference, the EESS document contains much more material on interface definitions but references the IEEE 1363.1 document for detailed discussion of security. Perhaps due to still being a draft revision, the IEEE 1363.1 document lacks the detailed information on NTRU-Sign, a signature scheme based on the NTRU operations, which can be found in the EESS document. Despite these differences, the similarities between the two standards are apparent when analyzing the recommended primitives and procedures.

3. Hardware Design—Optimization Strategies

3.1. Design Plan of NTRU Polynomial Multiplier

This project has designed and integrated the following major hardware components to achieve polynomial multiplications operations in the NTRU data path. The top-level structure of our design is presented in Figure 1.

(1) *Public/private key memory* which maps the indices of the public and private key polynomial coefficients to their values; additional information, whether the value of the coefficient is 0 or

DOI:10.1002/sec

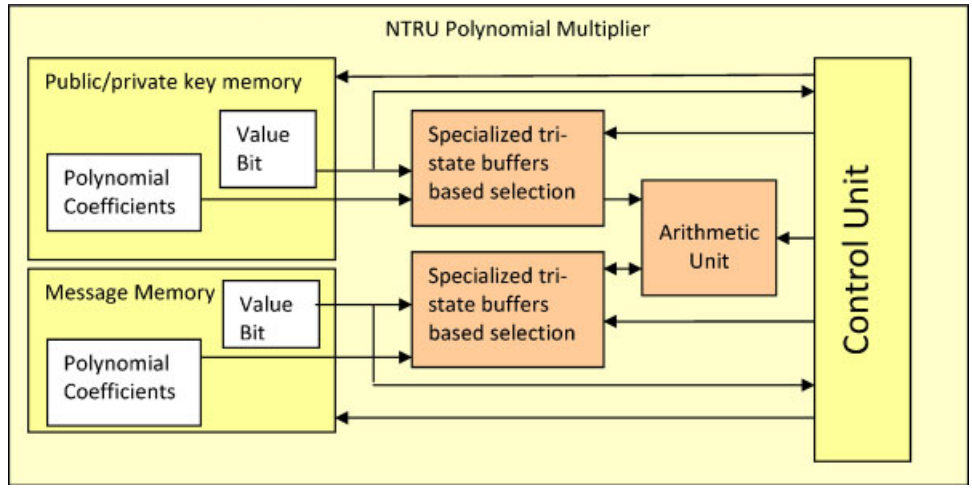

Fig. 1. Top-level structure of the NTRU polynomial multiplier.

not, is stored in the value bit portion of the memory (it will be later used to optimize the arithmetic operations and logical activity on interconnections).

(2) *Message memory* which maps the indices of the message polynomial coefficients to their values.
(3) *Specialized tri-state buffers-based selection unit* which implement low-power selection of the polynomial coefficient for processing.
(4) *Processing unit* which is a low-level optimized glitch free adder.
(5) *Control unit* which is a state machine that controls the process of polynomial multiplication.

3.2. NTRU *Throughput* Optimizations—Algorithm Level

Obviously, if we could shape the circuit design based on more time-efficient NTRU algorithms, we would improve throughput and thus speed up the encryption and decryption process. This project has investigated the optimization of the corresponding NTRU circuits based on the following NTRU algorithmic optimizations:

- Since the most time consuming part is polynomial multiplications, (this includes the calculation of the product $(r * h \bmod q)$ during encryption process, the two products during decryption: $(f * e \bmod q)$ and $(f_p^{-1} a \bmod p)$, and the inverses f_p^{-1} and f_q^{-1} during key creation), we have enhanced the control unit circuit so it is possible to store and access the coefficients of a polynomial $f(x)$.
- We have also investigated another way to speed up the encryption and decryption processes, that is, adopting products of *low* Hamming weight polynomials to decompose $r(x)$ [14]. In other words, we can significantly reduce the *star multiplication* time by taking $r(x)$ to be a product of polynomials with fewer *ones*. Thus, suppose that $r(x) = r_1(x) \cdot r_2(x)$, where r_1 and r_2 are binary polynomials with d_1 and d_2 *ones*, respectively. Then $r(x)$ will have approximately $d_1 \cdot d_2$ ones. Notice that the computation of the product $r(x) * h(x) = r_1(x) * (r_2(x) * h(x))$ requires only $(d_1 + d_2) \cdot N$ operations, so the computational time is largely reduced. From hardware implementation viewpoint, this project has designed a path-equalized multiplier with two sets of coefficients from low Hamming weight polynomials.

During the final step of the decryption process, the inverse of the private key modulo the small modulus p is multiplied by the candidate value to obtain the candidate plain text. One of the optimizations used is to choose the form of the private key to be $f = 1 + p * F$, where F is a random polynomial with d_F non-zero coefficients. By choosing the private key in this form, the inverse modulo p of the private key is simply one which eliminates the need for the final convolution multiplication during decryption. Additional savings are achieved in terms of storage and key generation computations, as the inverse of the private key modulo p does not need to be stored or computed.

To obtain greater efficiency during multiplication operations, an alternative form is suggested that takes

DOI:10.1002/sec

advantage of sparse polynomials. The suggested form, $f = f1 * f2 + f3$, is constructed from polynomials $f1, f2$, and $f3$ having d_{f1}, d_{f2}, and d_{f3} non-zero coefficients respectively. By multiplying against these separate vectors, the entire multiplication by f would require $(d_{f1} + d_{f2} + d_{f3})N$ operations instead of $(d_f)N$ operations per coefficient.

For example, with $N = 251$ and $d_f = 90$, one might choose $d_{f1} = d_{f2} = d_{f3} = 9$. A multiplication of a polynomial a by f would require $(d_f)N = 22590$ operations per coefficient. Using the alternative representation, the result could be calculated in three steps by $a * f1$, $(a * f1) * f2$, and $a * f3$, leading to $(d_{f1} + d_{f2} + d_{f3})N = 6777$ operations per coefficient.

Operating under the assumption that construction of message using a binary small modulus would be appealing, an algorithm is provided for efficient multiplication of a binary polynomial with a large modulus reduced polynomial. The algorithm takes advantage of representing the binary polynomial by a vector of positions of the non-zero elements. While such an approach might be applicable to a larger modulus, the efficiency of storing the positions of non-zero elements would be reduced with the need to store what value the non-zero elements were.

Optimizations can also be made to perform faster reduction of certain classes of moduli, such as Mersenne primes [9] which are of the form $p = 2^x - 1$. An example of a fast algorithm for reduction of Mersenne primes is shown in Figure 2. Modular inversion of integers is used during key generation in the NTRU system depending on the choice of inversion used for polynomials. Polynomial inversion using the Extended Euclidean Algorithm (EEA) requires an integer modular inversion per iteration and one final inversion to calculate the result. For smaller moduli, the inverse can often be easily stored in a lookup table, but the EEA can also be used for large moduli. In order to calculate the inverse modulo the power of a prime, an algorithm based on Newton's iteration is presented in Reference [15] and is repeated here, for convenience, in Figure 3. Although the algorithm is presented for convolution polynomials, the same method is applicable to integers as well.

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

Fig. 2. Reduction of mersenne primes [9].

Input:	$a(X)$, p (a prime), r (the exponent for p)
Output:	$b(X) \equiv a(X)^{-1} \pmod p$
Step 1:	$b(X) \equiv a(X)^{-1} \pmod{p^r}$
Step 2:	$q = p$
Step 3:	do while $q < p^r$
	$q = q^2$
Step 4:	$b := b(X)(2 - a(X)b(X)) \pmod q$

Fig. 3. Newton's iteration [15].

3.3. NTRU *Throughput* Optimizations—Circuit Level

To improve NTRU throughput at circuit level, we have investigated two possible approaches—parallelization and pipelining of the Arithmetic Units (AUs) (see Figure 4). Adding memory elements (registers) to the inputs and outputs of the AUs will allow the use of multiple instances of AU component (between 2 and K $(K < 2N)$) to implement *quasi-parallel* structure. The AUs' inputs and outputs will be controlled by a modified FSM. One advantage of the *quasi-parallel* circuit is that it does not sacrifice throughput/power ratio. The only additional components are registers in the AUs and a few states in the *control unit*. Additional registers can be implemented inside the AUs to allow further processing speed-up through pipelining.

This research has targeted the following issues: designing the data path to allow quasi-parallel accesses by tapping the polynomial coefficients into different positions; designing the FSM to accept multiple AU outputs without idle waiting between transition states; avoiding the access conflict (to reduce latency) between the NTRU inner and outer nested loops.

3.4. NTRU Circuitry *Power* Optimizations

Power dissipation P in digital circuit is usually divided into *dynamic* power (consumed by switching and short-circuit) and *static* power (leakage), as shown in Equation (4) (where N is the number of gate output transitions per clock cycle, and f is the operating frequency). The *switching* power is due to charge and discharge of the capacitors driven by the circuit; the *short-circuit* power is caused by the simultaneous conducting of nMOS and pMOS transistors; the *static* power is due to the leakage currents.

DOI:10.1002/sec

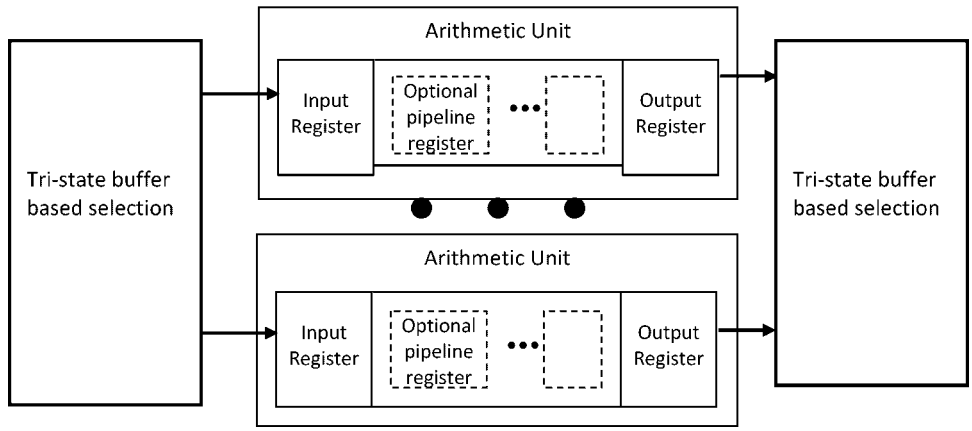

Fig. 4. Parallel/pipelined architecture of the NTRU datapath to improve throughput.

The *dynamic* part of the total power is still the dominating part in technologies of 90 nm and larger; *static* power is gaining importance at sub-90 nm technologies.

$$P = P_{\text{Dynamuc}} + P_{\text{Static}} = f \cdot N \cdot \left(\frac{C_{\text{Load}} \cdot V_{\text{DD}}^2}{2} + I_{\text{SCavrage}} \cdot V_{\text{DD}} \right) + I_{\text{Leakage}} \cdot V_{\text{DD}} \quad (4)$$

It is possible to analyze and optimize power dissipation at several levels such as algorithm, architecture, circuit, and device. The choice of target technology determines which of the levels are available. If FPGA or standard cell ASIC are the target technologies, power consumption can be optimized at algorithm, architecture, and circuit levels. If the target technology is full custom ASIC (each primitive logic cell or transistor is manually designed and optimized) then power consumption can be additionally optimized at the device level, however, in this case the initial design cost and design time are much higher. Very effective technique, that can be used in conjunction with those already mentioned, is voltage scaling. Here, the digital system is designed in the way so it will accept different supply voltages to allow controlling the speed and power consumption requirements.

In this part of our project, we have used a hierarchical approach to the power optimization of hardware implementation of a NTRU processor. First, we have analyzed the key components of NTRU algorithm and their activation depending on input transitions, scheduling, allocation and binding. This allows us to develop power-aware high-level architecture by applying the usual steps of high-level synthesis (however, with emphasis on the minimization of the power consumption). The NTRU architecture obtained in this way can be further optimized at circuit and device levels depending on the chosen target technology.

Based on the above power source analysis, we have summarized the following six important principles we have followed during the design of power-saving NTRU circuits:

(1) Minimize the number of gates' output transitions to reduce the *dynamic* power dissipation.
(2) Minimize the size of the circuit (number and sizes of transistors) to reduce both the dynamic and static power dissipation.
(3) Minimize the number of glitches in combinational part to reduce dynamic power dissipation.
(4) If target technology allows, implement non-critical paths using transistors with higher threshold voltage to reduce *static* power dissipation.
(5) Make non-critical paths operate at lower power supply to reduce both the dynamic and static power consumption.
(6) Apply additional techniques (e.g., substrate biasing) to reduce static power consumption if target technology is 90 nm or less.

3.5. Implementation of *Power* Optimized NTRU Circuitry

First, based on *Principles* 1, 2, and 3 (see above), less circuit units could reduce power consumption. Thus this research looks for all possible strategies that can reduce the circuit size with emphasis on power con-

DOI:10.1002/sec

sumption. For instance, the control unit implemented as *finite state machine* (FSM) should use minimum number of states but encoded in a way that will also minimize the size of the next state and output logic. It is very important that state transitions do not create any unnecessary glitches, so states will be encoded using Gray (and decomposed) codes in as many places as possible. As another example, we have noticed that we could use smaller word size mod p for coefficients of the random polynomial $r(x)$ (see Equation (2)). This can directly translate into fewer storage elements such as flip-flops.

Second, the NTRU star multiplications could consume significant power among all NTRU arithmetic operations. Suppose $c(X) = a(X) * b(X)$, we can calculate the coefficients of polynomial $c(X)$ as follows:

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_{N-1} b_{K+1} = \sum_{i+j \equiv k \bmod N} a_i b_j \tag{5}$$

We have designed NTRU circuits with the following three features to save power during *star multiplication*:

(1) Based on the observation that the sparse nature of $f_i(X)$ causes most of the inner product terms to be 0, we designed a value bit memory to be used in the selection of the operands for only those inner product terms which may be non-zero. Thus we largely save the power consumed on computing zero terms. Using additional memory seems to be better approach then comparing the value of each coefficient every time it is read from the memory—especially for the coefficients of the key which will not change during computations.
(2) We carefully designed the counters layout for the two nested loops that are used to implement cyclic convolutions. Linear feedback shift registers (LFSR) is a good choice for those counters in order to reduce gate-switching activities. Clock gating is used whenever possible to allow further power reduction.
(3) Specialized selection is necessary for repeated coefficient additions. However, a traditional simple register structure could cause many power-intensive gates transitions. This research conducts the design of a specialized tri-state buffers-based selection structure as presented in Figure 5. The value bit of each coefficient together with the selection control signal will select only non-zero coefficient for the processing.

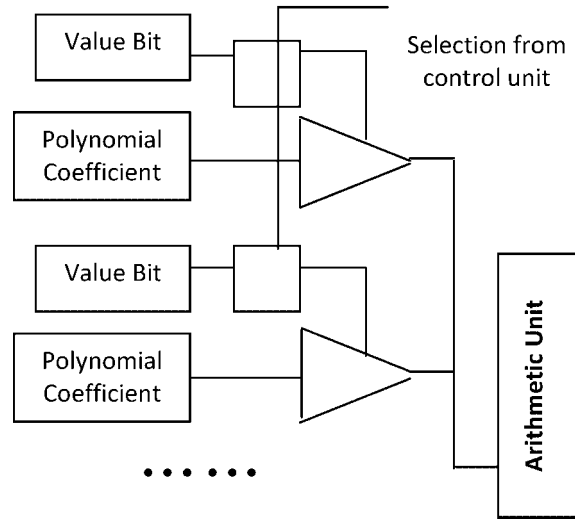

Fig. 5. Tri-state buffers based selection.

4. RTL-level Hardware Design *via* VHDL

4.1. RTL Design Goal

To investigate the aspects of the NTRU system, a hybrid behavioral and structural *very large integrated circuit hardware description language* (VHDL) model was designed. Components that were easily translatable to hardware were implemented using structural models, while some of the more complex components were written using behavioral style code. Two main goals were used during the design and creation of the IEEE 1363.1 system model (see Figure 6).

The first goal was to make the system as modular as possible to facilitate the changing of individual modules without need for recreation of the entire model. Due to the draft status of the IEEE 1363.1 standard, it is likely that future edits could change the recommended practices, which is reflected in the separation of the system into fairly small functional blocks. Each of the functional blocks making a sensible division, the modules are mainly separated around the boundaries of the algorithms presented in the standard.

The second goal was to make the system flexible with regard to changing of the top-level system parameters. Although the system could have been created in a manner similar to the algorithm blocks defined in the IEEE 1363.1 draft standard, the choice was instead made to use generic declarations to control each module. Use of generics in each module allowed the system testbench to control the parameters used in testing without modification of each individual

DOI:10.1002/sec

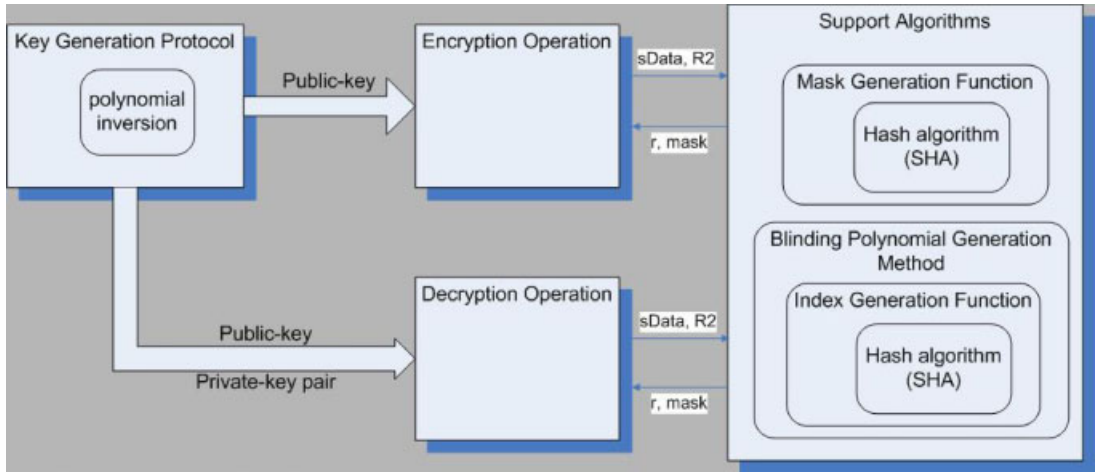

Fig. 6. Our design is based on IEEE 1363.1 standard.

module but also allowed for a reduced number of inputs and outputs compared to that needed for a dynamically changing system. The full modules for key generation, encryption, and decryption were wrapped in a testbench which provided inputs based on available testing data and checked for the expected outputs. The final model became a testing platform for all of the pieces of the IEEE 1363.1 draft standard with the ability to be piece-wise adapted for further study and optimization.

Figure 7 shows our VHDL design procedure which includes NTRU components implementation and test. In order to facilitate the convolution multiplications of a general system, a model was created to handle all of the cases that might need to be handled (see Figure 8). The model accepts two polynomials modulo p, two polynomials modulo q or a combination of the both. In addition, the output polynomial can be reduced modulo an integer input, mod n, to allow for special cases and also allows a scaling of the output polynomial, through use of the integer input *scale*, before reduction for calculations such as the public key where the output needs to be scaled by p. The type of input used and whether or not the output is scaled is controlled by the input *datyp*. The calculation that performs the convolution multiplication is the basic algorithm, taking N^2 operations, where each operation is a multiplication and an addition. The calculation is initiated by a rising edge on the *daclk* and is calculated immediately due to the behavioral nature of the model. In addition to the modulo p and q outputs, there is a modulo two output used for directly inputting into the mask generation function after the calculation of $R = r * h \pmod{q}$. The generic values

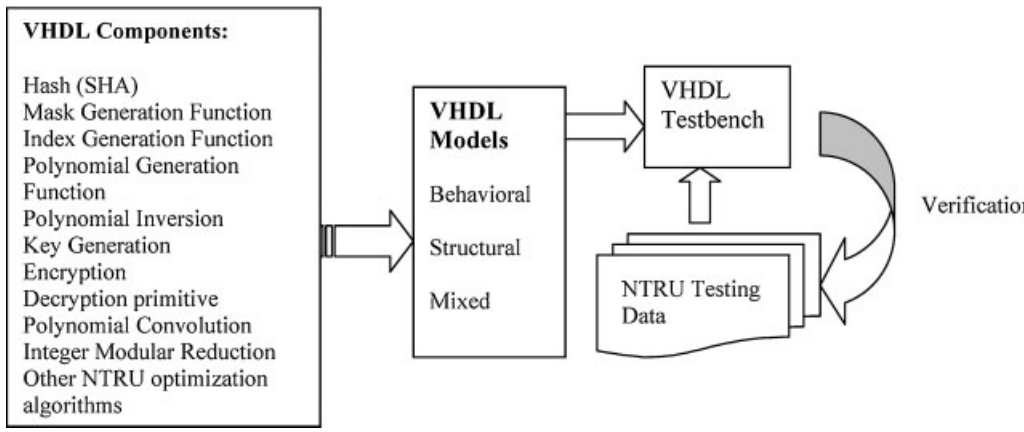

Fig. 7. NTRU hardware design procedure.

DOI:10.1002/sec

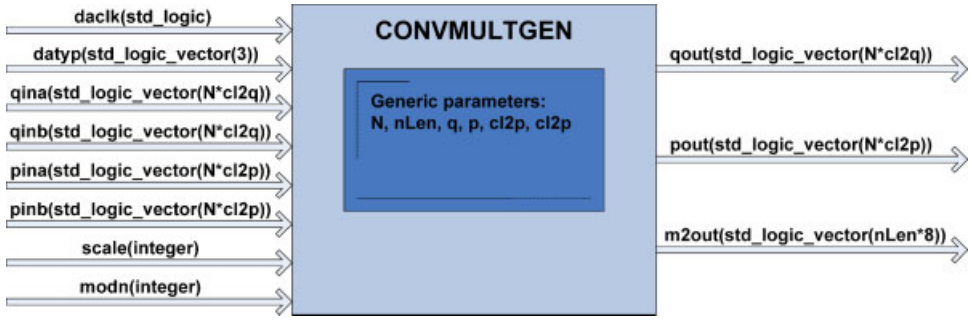

Fig. 8. NTRU VHDL design modules.

cl2q and *cl2p* are used throughout the system and represent the number of bits that should be used to store each coefficient in q and p, respectively.

4.2. System Testing Results

The behavioral VHDL model was mainly tested using data provided for the IEEE 1363.1 draft standard. The four-parameter sets, that data were available for, were ees347ep2, ees397ep1, ees587ep1, and ees787ep1. Since testing data were not yet available for product forms, the focus of the testing was conducted using the less optimized full polynomial algorithms. The testbench used a series of constant assignments applied through the generics for each individual component model to control the parameters for testing.

Figure 9 shows our NTRU encryption/decryption results in VHDL models.

The method of storage and the amount of storage required to implement the NTRU system can be of particular concern for those using resource constrained hardware or for those seeking optimize performance. The trivial method of storing polynomials is to store each coefficient in a linear array, taking $N * \lceil \log_2(q) \rceil$ or $N * \lceil \log_2(p) \rceil$ bits of storage for a polynomial modulo q or p, respectively. The appeal of this method is the simplicity involved, however, this is clearly not as appealing as the number of bits needed to store each coefficient or the degree of the polynomials increase. Using the idea that many of the coefficients involved in a polynomial will be zero, an alternative method is to store each non-zero coefficient and the degree that coefficient represents. Assuming there were num_{nz} percentage of non-zero coefficients, this method would require:

$$N(\lceil \log_2(q) \rceil + \lceil \log_2(N) \rceil)\text{num}_{nz} \text{ or}$$
$$N(\lceil \log_2(p) \rceil + \lceil \log_2(N) \rceil)\text{num}_{nz}$$

bits of storage per polynomial. To evaluate which method is better in a general manner is difficult to imagine because the equations are based on N, p, or q

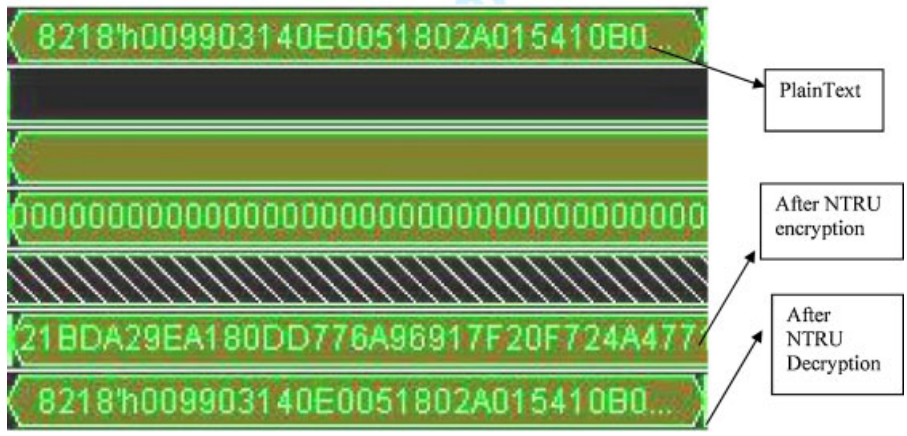

Fig. 9. VHDL simulation results.

DOI:10.1002/sec

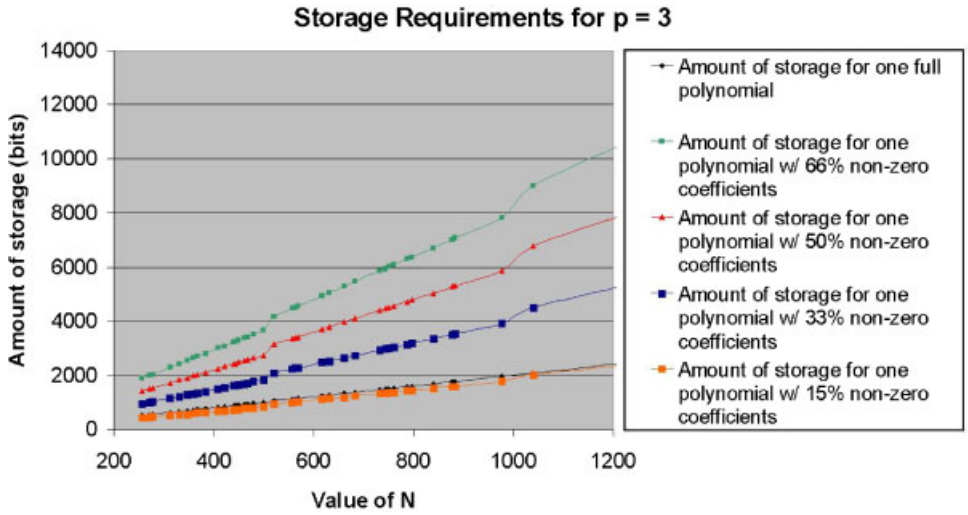

Fig. 10. Memory overhead analysis.

and num_{nz}, which requires varying of three input variables and assessment of the output amount of storage. Instead, to avoid over complication during examination of the results, one of the three input variables can be constant, a second can be varied and a graph can be made for each of a series of values for the third value. The choice was made to hold a value for either p or q, vary N and graph the results for a series of different values for num_{nz}. To examine one of the extremes, the results for $p = 3$ are shown in Figure 10.

Inversion is an expensive operation in many cryptosystems, and such is the case for the NTRU system. For key generation in NTRU, the inverse of the private key polynomial must be taken modulo the large modulus, q. For security purposes, it is also checked that the inverse of g exists modulo q. For now, two suggested algorithms exist for calculation of the inverse, the EEA and the almost inverse algorithm (AIA) [10,16,17], of which only the EEA is recommended for use in the IEEE 1363.1 draft standard. Although an implementation of each algorithm was tested during the course of this research, they were not developed to the point where a reasonable comparison of hardware results could be made. Instead, a comparison can be made by examining one iteration of each algorithm in terms of the operations required. To start with, the general EEA presented can be compared with the general AIA, as shown in Table I. From the comparison in Table I, it might be thought that the AIA contains more work per iteration than the EEA. The AIA has two polynomial rotations and one polynomial degree calculation more than the EEA, but one fewer polynomial addition. While this could be considered a fair comparison of the main loops of each algorithm, what it ignores is the inner loop inside the

Table I. Comparison of operations per iteration between inversion methods.

Operation	# in EEA	# in AIA	Steps found in
Integer invasion	1	1	EEA: step 8 AIA: step 10
Polynomial rotation	0	2	AIA: step 4
Polynomial degree	1	2	EEA: step 9 AIA: steps 5 and 8
Polynomial addition/subtraction	1/2	0/2	EEA: step 13/steps 12 and 14 AIA: steps 11 and 12
Polynomial convolution multiplication	2	2	EEA: steps 12 and 14 AIA: steps 11 and 12

DOI:10.1002/sec

EEA. The inner loop of the EEA contains a polynomial degree calculation, a polynomial addition, a polynomial subtraction and a polynomial convolution multiplication. Assuming the inner loop executes at least twice on a given iteration of the EEA, the AIA achieves inversion through fewer computations.

5. Conclusions

In this paper, we have presented a parameter and component flexible testing model for the NTRU public-key cryptosystem in conformance to the IEEE 1363.1 draft standard. The model was successfully tested using provided and generated test datasets and is now adaptable for further software and hardware research. Research conducted during creation and testing of the model was used to analyze the NTRU system with respect to both general underlying mathematical operations and specific qualities relating to the IEEE 1363.1 draft standard. The results of the research suggest that the system is highly adaptable to many conditions based on choices in the system parameters. Representation of the polynomial operands can be chosen to maximize storage efficiency. Hardware implementations benefit by using the maximum value of q which fits the bit width allowable in the hardware, followed by adjustment in N to achieve the desired security level. Parallelism in the operations of the system can be exploited to achieve better efficiency but must be carefully considered to avoid complications when adapting to new parameter sets. Overall, the work presented here can be used as underlying research for further investigation of the NTRU system and IEEE 1363.1 draft standard.

Acknowledgement

This project has been supported in part by U.S.A. National Scientific Foundation (NSF) under the grants of CNS-0716211 and #CNS-0716455. Any ideas from this paper do not necessarily reflect the opinions of NSF.

References

1. Perrig A, Szewczyk R, Tygar JD, Wen V, Culler DE. Spins: security protocols for sensor networks. In *Wireless Networks*, V8, N5, Springer, Berlin, Germany, 2002; 521–534.
2. Stinson DR. *Cryptography: Theory and Practice* (3rd edn). Chapman & Hall/CRC, New York, 2006.
3. Koblitz N. Elliptic curve cryptosystems. In *Mathematics of Computation*, Vol. 48, 1987; 203–209.
4. Kaps J-P. Cryptography for ultra-low power devices. *Ph.D. dissertation*, Worcester Polytechnic Institute, 2006.
5. Rivest R, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM*, Vol. 21, 1978; 120–126.
6. O'Rourke CM. Efficient NTRU implementations. *Master's thesis*, Worcester Polytechnic Institute, 2002.
7. Hoffstein J, Pipher J, Silverman JH. NTRU: a ring-based public key cryptosystem. In *Algorithmic Number Theory* (*ANTS III*), Vol. 1423, Buhler JP (ed.). Lecture Notes in Computer Science (LNCS) Springer, Berlin, 1998; 267288.
8. Howgrave-Graham N. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. *NTRU Cryptosystems Inc*., 2007.
9. Hoffstein J, Silverman JH, Whyte W. NTRU cryptosystems technical report #12, version 2: estimated breaking times for NTRU lattices. *NTRU Cryptosystems Inc*., 2003. [Online]. Available: http://ntru.com/
10. Silverman JH, Whyte W. NTRU cryptosystems technical report #21, version 1: timing attacks on NTRUEncrypt via variation in the number of hash calls. *NTRU Cryptosystems Inc*., 2006. [Online]. Available: http://ntru.com/
11. Buchmann J, Döring M, Lindner R. Efficiency improvement for NTRU. Technische Universität Darmstadt, 2007.
12. Bailey DV, Coffin D, Elbirt A, Silverman JH, Woodbury AD. NTRU in constrained devices. In *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, 2001; 266–277.
13. IEEE P1363 working group for standards. In Public Key Cryptography, IEEE 1363-2000 Standard Specifications for Public-Key Cryptography. *Institute of Electrical and Electronics Engineers Inc*., 2000. [Online]. Available: http://grouper.ieee.org/groups/1363/P1363/
14. Hoffstein J, Silverman JH. Random small hamming weight products with applications to cryptography. *NTRU Cryptosystems Inc*., 2000. [Online]. Available: http://ntru.com/
15. Howgrave-Graham N, Silverman JH, Whyte W. Choosing parameter sets for NTRUEncrypt with NAEP and SVES-3. *NTRU Cryptosystems Inc*., 2005. [Online]. Available: http://ntru.com/
16. Hoffstein J, Howgrave-Graham N, Pipher J, Silverman JH, Whyte W. Hybrid lattice reduction and meet in the middle resistant parameter selection for NTRUEncrypt. *NTRU Cryptosystems Inc*., 2007.
17. Efficient embedded security standard (EESS) #1. *Consortium for Efficient Embedded Security*, 2003. [Online]. Available: http://www.ceesstandards.org

DOI:10.1002/sec