Maejo Int. J. Sci. Technol. **2010**, *4*(03), 428-434

Maejo International
Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Communication

A random number generator based on NTRU cryptosystem

He Debiao*, Chen Jianhua and Hu Jin

School of Mathematics and Statistics, Wuhan University, Wuhan, China

* Corresponding author, e-mail: hedebiao@163.com

Received: 23 February 2010 / Accepted: 11 October 2010 / Published: 16 October 2010

Abstract: A random number generator based on the operation of the NTRU cryptosystem is proposed. By using the proposed generator together with the NTRU cryptosystem, we can save hardware and software components. Theoretical analyses show that periods of the proposed random number generator are sufficiently long. Moreover, the generated sequences have passed the U.S. NIST statistical test.

Keywords: random number generator, public-key cryptosystem, NTRU cryptosystem, polynomial ring

INTRODUCTION

The NTRU (Number Theory Research Unit) cryptosystem [1-3], patented by the company NTRU, is one of the fastest public-key encryption schemes known. Although this may not be a decisive advantage compared to the hybrid encryption with RSA for example, NTRU has attracted considerable interest and is being considered by the efficient embedded security standards [4] and the IEEE P1363 study group for future public-key cryptography standards [5].

On the other hand, the security of most cryptographic systems depends upon the generation of unpredictable quantities that must be of sufficient size and randomness. Taking NTRU cryptosystem as example, we need to generate random bits in order to create random polynomial. This implies that we usually need to implement a random number generator in a cryptographic system. A number of random number generators have been proposed [6-11]. However, they are usually not designed together with the cryptographic system and so extra design and implementation effort are required. If both the tasks of random number generation and encryption can be done by using the same software or hardware module, we can save hardware cost, memory space and design time. This is especially important in developing applications in an environment with limited resources such as smart cards.

With this goal, we propose a random number generator that makes use of the basic operations required in NTRU.

The organisation of the rest of the paper is as follows. The background of NTRU cryptosystem is first introduced. The proposed random number generator is then described. Periods of the proposed generator are subsequently analysed and the test results reported.

NTRU PUBLIC KEY CRYTOSYSTEM

NTRU cryptosystem is a polynomial ring-based public-key cryptosystem that was fully introduced in 1998 [1]. The scheme is set up by three integers, viz. N, p, q, such that:

- N is prime;
- p and q are relatively prime, gcd(p, q)=1;
- q is much larger than p.

NTRU cryptosystem is based on polynomial additions and multiplications in the ring $R = Z[x]/(x^N - 1)$. We use * to denote a polynomial multiplication in R, which is the cyclic convolution of two polynomials. After completion of a polynomial multiplication or addition, the coefficients of the resulting polynomial need to be reduced either modulo q or p. As a side note, the key creation process also requires two polynomial inversions, which can be computed using the extended Euclidean algorithm. NTRU cryptosystem requires approximately $o(N^2)$ operations and a key length of $o(N)$.

More information on NTRU cryptosystem has been described [1-3]. We briefly outline the procedures below.

Key Generation. To generate the public key, the user must:

- Choose a secret key, a random polynomial $f \in R$, where coefficients are in $(-\frac{p}{2}, \frac{p}{2})$;
- Choose a random polynomial, $g \in R$, where coefficients are in $(-\frac{p}{2}, \frac{p}{2})$;
- Compute the inverse polynomial F_q of the secret key f modulo q.

Once the above has been completed, the public key, h, is found as

$$h = F_q * g (\mathrm{mod}\, q) \tag{1}$$

Encryption. The encrypted message is computed as

$$e = pr * h + m (\mathrm{mod}\, q) \tag{2}$$

where the message, $m \in R$, and the random polynomial, $r \in R$, have coefficients reduced modulo p.

Decryption. The decryption procedure requires three steps:

- $a = f * e (\mathrm{mod}\, q)$;
- Shift coefficients of a to the range $(-\frac{q}{2}, \frac{q}{2})$;
- $d = F_p * a (\mathrm{mod}\, p)$.

The last step of decryption requires the user to compute the inverse polynomial F_p of the secret key f modulo p. The decryption process outlined above will recover the original message.

THE PROPOSED RANDOM NUMBER GENERATOR

In this paper we use the encryption progress of NTRU cryptosystem to generate the random sequence, the parameters being as follows:

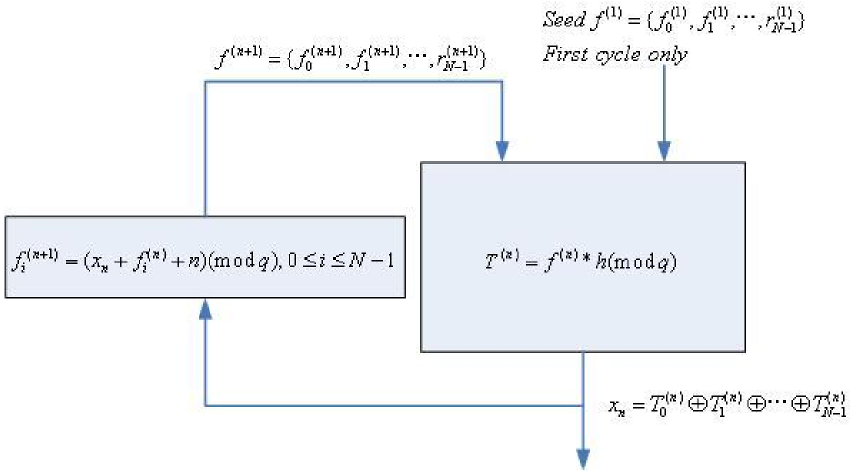
- N is prime;
- q is prime and is large enough;
- h is an element of the ring $R = Z[x]/(x^N - 1)$ and is randomly selected.

Let f be an element of the ring $R = Z[x]/(x^N - 1)$, then f can be represented as

$$f = \sum_{i=0}^{N-1} f_i x^i, F_i \in Z, i = 0,1,\ldots,N-1. \qquad (3)$$

The representation is denoted by $f = [f_0, f_1, \ldots, f_{N-1}]$.

A block diagram of the proposed random number generator is shown in Figure 1, where $\oplus$ denotes the operation of XOR. It is easy to see that when these operations are done recursively, a sequence of bits can be obtained by collecting the x_n.



$Seed\ f^{(1)} = (f_0^{(1)}, f_1^{(1)}, \cdots, r_{N-1}^{(1)})$
First cycle only
$f^{(n+1)} = (f_0^{(n+1)}, f_1^{(n+1)}, \cdots, r_{N-1}^{(n+1)})$
$f_i^{(n+1)} = (x_n + f_i^{(n)} + n)(\mathrm{mod}\, q), 0 \le i \le N-1$
$T^{(n)} = f^{(n)} * h(\mathrm{mod}\, q)$
$x_n = T_0^{(n)} \oplus T_1^{(n)} \oplus \cdots \oplus T_{N-1}^{(n)}$

Figure 1. Block diagram of the proposed random number generator

PERIOD ANALYSIS

The purpose of setting $f_i^{(n+1)} = (x_n + f_i^{(n)} + n)(\mathrm{mod}\, q), 0 \le i \le N-1$ is to increase the period of the generator. If n is not added, the bit sequence depends solely on the output of the $T^{(n)} = f^{(n)} * h(\mathrm{mod}\, q)$ operation and the period may be very small.

In the s^{th} cycle,

$$f_i^{(s+1)} = (x_s + f_i^{(s)} + s)(\mathrm{mod}\, q), 0 \le i \le N-1 \qquad (4)$$

where x_s denotes the output of the proposed random number generator, shown in Figure 1.

In the t^{th} cycle,

$$f_i^{(t+1)} = (x_t + f_i^{(t)} + t)(\mathrm{mod}\, q), 0 \le i \le N-1 \qquad (5)$$

where x_t denotes the output of the proposed random number generator, shown in Figure 1.

Suppose that the output of the proposed random number generator in the t^{th} cycle is the same as that in the s^{th} cycle, i.e.

$$x_t = x_s \qquad (6)$$

If the output of the module of the $(t+1)^{th}$ cycle is also equal to that of the $(s+1)^{th}$ cycle, then

$$f_i^{(t+1)} \equiv f_i^{(s+1)} \bmod q, f_i^{(t)} \equiv f_i^{(s)} \bmod q, 0 \le i \le N-1. \tag{7}$$

By equations (4), (5) and (7), we have

$$x_t + t = x_s + s \tag{8}$$

By equations (6) and (8), we can get that

$$s \equiv t \bmod q \tag{9}$$

Since $t > s$, we have $t = s + m\ q$, where m is a nonzero positive integer. Hence, the output pattern will repeat only after q cycles.

TEST RESULTS

The U.S. NIST statistical test suite [12] is used to test the randomness of the generated bits. It includes 15 statistical tests and each of them is formulated to test a null hypothesis that the sequence being tested is random. There is also an alternative hypothesis which states that the sequence is not random. For each test, there is an associated reference distribution (typically normal distribution or χ^2 distribution), based on which a P_value is computed from the binary sequence. If this value is greater than a pre-defined threshold α (0.01 in default), the sequence passes the test. The two approaches that NIST has adopted are the examination of: (1) proportion of sequences that pass a statistical test, and (2) uniformity of the distribution of those P_values.

According to NIST [12], if m sequences were tested, the proportion of sequences that passed a specific statistical test should lie above p_α:

$$p_\alpha = (1-\alpha) - 3\sqrt{\frac{\alpha(1-\alpha)}{m}} \tag{10}$$

In our experiment, $m = 1000$, $\alpha = 0.01$ and $p_\alpha = 98.05\%$.

To check the distribution of P_values, the interval between 0 and 1 is divided into 10 sub-intervals. The number of P_values in each sub-interval is counted, based on which a $P\_valuesT$ is calculated. If $P\_valuesT > 0.0001$ holds, the sequences are considered to be uniformly distributed.

The NIST statistical test suite contains 15 tests (the Lempel-Ziv complexity test having been removed from the test suite since Version 1.7) [12]. Some tests such as FT, FBT, RT, ST, AET and CST require only 100 bits for each sequence. Other tests, however, require more bits. Specially, the PTMT, LZCT, RET and REVT tests need about $1M$ for each sequence.

In our experiment, a 100-bit sequence is generated for each two parameters (N, q) by using 100 pairs of the randomly-selected initial $h, f^{(1)}$. Because the expression of the element of the ring $R = Z[x]/(x^N - 1)$ is very complicated, we just list one case here for reference. The parameters are

- N=107
- q=0xCC137071797457130CFCF2CC51406B442E15EEA463F80F3C085C3ECC381BF01B
- $h = x^{106} + x^{105} + \ldots + x + 1$
- $f^{(1)} = x^{106} - 1$

In the test of P_values uniformity for FT and CST, each sequence is set to 1024 bits by concatenating the output of the proposed random generator. Otherwise, the number of different P_values will not be sufficient to carry out the uniformity test. As for other tests which require more than 100 bits for each sequence, we collect the first 256 bits for different initial $h, f^{(1)}$. The test results can be found in Table 1.

It can be observed from Table 1 that the proposed random number generator passes all the statistical tests, i.e. the passing proportions are greater than 98.05% and $P\_valuesT$ greater than 0.0001. According to NIST [12], we can conclude that the data generated by these two approaches are random.

Table 1. Test results for the random number generator

Test name	Proportion	*P_valueT*
FT	0.9901	0.1624
FBT	0.9852	0.3781
CST*	0.9863	0.1639
RT	0.9942	0.1498
LROBT	0.9875	0.1397
AET	0.9846	0.4153
ST*	0.9811	0.5741
RBMRT	0.9973	0.1542
DFTT	0.9918	0.3681
ATMT*	0.9857	0.2456
PTMT	0.9961	0.1572
MUST	0.9921	0.3660
RET*	0.9836	0.2718
REVT*	0.9918	0.2249
LCT	0.9826	0.1152

DISCUSSION

Choice of Parameters

By simply changing the seed f and the initial polynomial h, a different bit sequence can be generated. These two parameters should be kept secret for security. Basically, h can be any polynomial in the finite field F_q.

The NTRU system just requires that p and q be relatively prime. However, if q in our system is not prime, the period of the output of the generator may be a divisor of q. Then our system requires that q is prime. In general, the random number generator is secure if its period is about 2^{256}.

Implementation

Various types of implementation of NTRU cryptosystem have been proposed [13-16] and our proposed random number generator is based on the core operations of NTRU cryptosystem, so the proposed random generator can be designed and implemented efficiently using the existing components, and thus the cost of the implementation can be reduced.

The equation of our random number generator looks different from that of the NTRU cryptosystem. Actually, we just replace the polynomial $p*h$ with another polynomial h'. In the process of generating random numbers using existing components, we just let p be one.

CONCLUSIONS

In this paper, a new approach for constructing a random number generator using the operation of NTRU cryptosystem is presented. Periods of the generator are analysed theoretically. The test results indicate that all the random number sequences pass the U.S. NIST statistical test. Therefore, the proposed generator can be accepted as a reliable random number generator for integrating with the NTRU system to generate the dynamic private keys.

ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers and Dr. Duang Buddhasukh for their valuable comments. This research was supported by the Fundamental Research Funds for the Central Universities under Grants 201275786.

REFERENCES

1. J. Hoffstein, J. Pipher and J. H. Silverman. "NTRU: a ring based public key cryptosystem", Proceedings of Algorithmic Number Theory: Third International Symposium, **1998**, Portland, USA, pp. 267-288.
2. J. Hoffstein and J. H. Silverman, "Optimizations for NTRU" , http://www.sisecure.com/cryptolab/pdf/TECH_ARTICLE_OPT.pdf. (Accessed: 1 December **2009**)
3. C. O'Rourke and B. Sunar, "Achieving NTRU with Montgomery multiplication", *IEEE Trans. Comput.*, **2003**, *52*, 440-448.
4. EESS: Consortium for Efficient Embedded Security, "Efficient embedded security standards #1: Implementation aspects of NTRU and NSS", Draft Version 3.0, available from http://www.ceesstandards.org, July **2001**.
5. IEEE Standard 1363, "Standard specifications for public key cryptography", available from http://grouper.ieee.org/groups/1363, August **2000**.
6. D. Knuth, "The Art of Computer Programming, Vol. II: Seminumerical Algorithms", 3rd Edn., Addison-Wesley, New York, **1998**, pp. 95-100.
7. M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits", *SIAM J. Comput.,* **1984**, *13*, 850-863.
8. O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions", *J. Assoc. Comput. Mach. ,* **1986**, *33*, 792-807.
9. J. A. Gonzalez and R. Pino, "A random number generator based on unpredictable chaotic functions", *Comput. Phys. Commun.* **1999**, *120*, 109-144.
10. M. Orlov, "Optimized random number generation in an interval", *Inf. Process. Lett.*, **2009**, *109*, 722-725.

11. H. Shi, S. Jiang and Z. Qin, "More efficient DDH pseudorandom generators", *Designs Codes Cryptogr.*, **2010**, *55*, 45–64.
12. National Institute of Standards and Technology (NIST), "A statistical test suite for random and pseudo-random number generators for cryptographic applications", http://csrc.nist.gov/rng/rng2.html, 2001. (Accessed: 1 November **2009**).
13. C. O'Rourke and B. Sunar, "Achieving NTRU with Montgomery multiplication", *IEEE Trans. Comput.*, **2003**, *52*, 440-448.
14. J. H. Silverman, "Commutative NTRU: Pseudo-code implementation", http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.3682 (Accessed: 1 November **2009**).
15. D. V. Bailey , "Daniel coffin, NTRU in constrained services" , http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.9174 (Accessed: 1 November **2009**).
16. M.-K. Lee, J. W. Kim, J. E. Song and K. Park, "Sliding window method for NTRU", Proceedings of the 5th International Conference on Applied Cryptography and Network Security, **2007**, Zhuhai, China, pp. 432-442.