A Novel Approach to Communicate Secret Message between Users Using Sponge Function Technique on NTRU

S.Varaprasad[(1)#], K.Venkata Rao[(2)*], and P.S.Avadhani[(1)@]

[(1)]*Dept.of Computer Science & Systems Engineering Andhra University,Visakhapatnam, INDIA*
[(2)]*Dept.of Computer Science Vignan Instuite of Information Technology,Visakhapatnam,INDIA*

e-mail: #Vara996@gmail.com; *vrkoduganti@gmail.com; @psavadhani@yahoo.com

Abstract - This paper presents a novel approach for a (key distribution) for secret message communication among a group (G). In order to increase security to distribute secret message (key), we introduce sponge functions using these at a specific permutation. We generate a key and distribute this key using (PKCS)(public key crypto systems), the absorbing, squeezing functions are used. In this paper an introduction part which briefs regarding sponge functions, key distribution centre, group communication and NTRU, key generation authentication, in literature review we describe about the research states of sponge functions, lightweight hash functions-KDC – NTRU. In proposed work we propose how the group communication establishes registration of users, entry and exit of a user. The encryption and decryption algorithm are used between sender and receiver. The entire proposed work is verified in VHDL and 'MATLABS'.

[Keywords : *Sponge function; NTRU; encryption; decryption; Keydistributioncenter (KDC) Absorbing; Squeezing functions]*

I. INTRODUCTION

Designers of lightweight cryptographic algorithms or protocols have to trade off between two opposite design phi mlosophies. The first consists in creating new schemes from scratch, whereas the second consists in reusing available schemes and adapting them to system constraints. They are more in line with the latter approach—as illustrated by their DM-PRESENT proposal—we ten more towards the former. Although QUARK borrows components from previous works, it integrates a number of innovations that make it unique and that optimize its light weightness. As explained in this section, QUARK combines a sponge construction with a capacity *c* equal to the digest length *n*, a core permutation inspired by previous primitives, optimized for reduced resources consumption. This design strategy as an attempt to optimize its security-performance ratio. Subsequent proposals of lightweight hash functions followed a similar strategy, with PHOTON and SPONGENT respectively building their core permutations on AES- and SERPENT-like algorithms.

II. MATERIALS AND METHODS

I. Separating digest length and security level

We observe that the digest length of a hash function has generally been identified with its security level, with (say) n-bit digests being equivalent to n-bit security against preimage attacks. However, this rule restricts the variety of designs, as it forces designers to exclude design paradigms that may otherwise increase usability or performance the notation introduced in the context of sponge finctions[13] was first step towards a separation of digest length and security level, and thus towards more inventive designs. In particular, the necessity of n-bit (second) preimage resistance is questionable from a pragmatic standpoint, when one needs to assume that $2^{n}/^{2}$ is an infeasible effort, to avoid birthday collision search. Designers may thus relax the security requirements against (second) preimages—as informally suggested by several researchers in the context of the SHA-3 Competition—so as to propose more efficient algorithms[13].

III Working with shift registers

In cryptography, linear or non-linear feedback shift registers have been widely used as a building block of stream ciphers, thanks to their simplicity and efficiency of implementation (be it in terms of area or power consumption). In the design of QUARK, we opt for an algorithm based on bit shift registers combined with(non-linear) Boolean functions, rather than for a design based on S-boxes combined with a linear layer (as PHOTON and SPONGENT). This is motivated by the simplicity of description and of implementation, and by the close-to-optimal area requirements it induces. Indeed, the register serves both to store the internal state (mandatory in any construction) and to perform the operations bringing confusion and diffusion .

IV Description of the QUARK hash family

This section gives a complete specification of QUARK and of its three proposed instances: U-QUARK, D-QUARK, and S-QUARK. In particle physics, the u-quark is lighter than the d-quark, which itself is lighter than the s-quark; our eponym hash functions compare similarly.

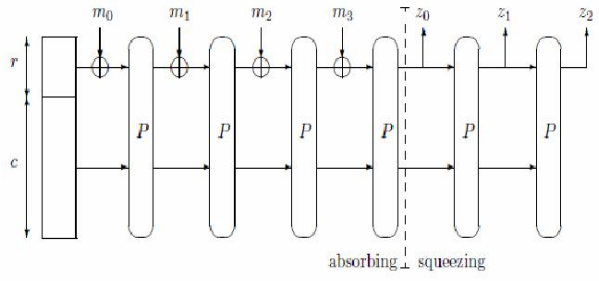


Fig.1 Sponge Construction of a 4-block padded Message

2..Sponge construction

QUARK uses the sponge construction, depicted in Fig. 1, and a 6-bit permutation *P* (that is, a bijective function over $\{0,1\}^b$). Following the notations introduced a QUARK instance is parameterized by a *rate* (or block length) r, a *capacity* c, and an *output length n.* The *width b = r+c of a* sponge construction is the size of its internal state. We denote this internal state $s = (s_0 \ldots, s_{b-1})$, where s_0 is referred to as the *first* bit of the state. Given a predefined initial state of b bits (specified for each instance of Quark) the sponge construction processes a message m in three steps [4].

1. **Initialization:** the message is padded by appending a '1' bit followed by the minimal (possibly zero) number of '0' bits to reach a length that is a multiple of *r*.
2. **Absorbing phase:** the r-bit message blocks are XOR's with the last *r* bits of the state (that is $S_{b-r}, \ldots S_{b-2}, S_{b-1}$) interleaved with applications of the permutation *P*. The absorbing phase starts with an XOR between the first block and the state, and it finishes with a call to the permutation *P*.
3. **Squeezing phase:** the last *r* bits of the state are returned as output, interleaved with applications of the permutation *P*, until *n* bits are returned. The squeezing phase starts with the extraction of *r* bits, and also finishes with the extraction of *r* bits.

2.1.Permutation

As depicted in Fig. the internal state of *P* is viewed as three feedback shift registers (FSRs) two non-linear ones (NFSRs) of b/2 bits each, and a linear one (LFSR) of [log 4b] bits. The state at epoch $t \geq 0$ is thus composed of

a. AnNFSRX ofb/2bits, denoted $X^t = (X_0^t, \ldots X^t_{b/2-1})$.
b. An NFSR*Y* of b/2 bits, denoted $Y^t=(Y_0^t, \ldots Y^t_{b/2-1})$.
c. An LFSR *L* of [log 4b] bits,denoted $L^t= (L_0^t \ldots\ldots L^t_{|\log 4b|-1})$
d. Given a b-bit input, *P* proceeds in three stages, as described below.
e. Initialization. Upon input of the b-bit internal state of the sponge construction $s = (s_0, \ldots, s_{b-1})$, *P* initializes its internal state as follows:

X is initialized with the first b/2 input bits: $(X_0^0, \ldots, X^0_{b/2-1}) := (s_0, \ldots\ldots, S_{b/2-1})$.
Y is initialized with the last b/2 input bits: $(Y_0^0, \ldots\ldots Y^0_{b/2-1}) := (s_{b/2}, \ldots\ldots s_{b-1})$.
L is initialized to the all-one string: $(L_0^0, \ldots\ldots, L^0_{|\log 4b|-1}) := (1, \ldots\ldots 1)$.

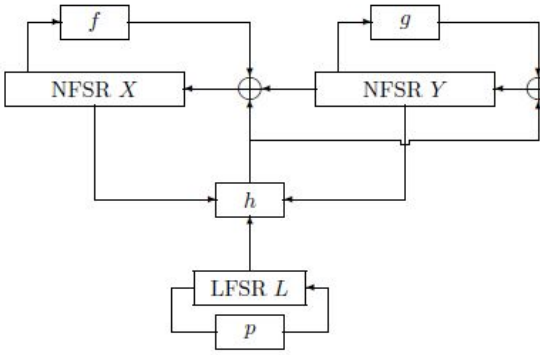
4. **State update**

From an internal state (X^t, Y^t, L^t), the next state $(X^{t+1}, Y^{t+1}, L^{t+1})$ is determined by clocking the internal mechanism as follows

a. The function *h* is evaluated upon input bits from X^t, Y^t, and L^t, and the result is written $h^t ::= h(x^t, Y^t, L^t)$.
b. *X* is clocked using Y_0^T, the function *f*, and h^t:
c. $(X_0^{t+1}, \ldots\ldots X^{t+1}_{b/2-1}) := (X_1^t, \ldots\ldots X_{b/2-1}, Y_0^t + f(X^t) + h^t)$.
d. *Y* is clocked using the function *g* and *H*= $(Y_0^{t+1}, \ldots\ldots Y^{t+1}_{b/2-1}) := (Y_1^t, \ldots\ldots Y_{b/2-1}, g(Y^t) + h^t)$.
e. *L* is clocked using the function *p:* $(L_0^{t+1}, \ldots\ldots L^{t+1}_{|\log 4b|-1}) := (L_1^t, \ldots\ldots L^t_{|\log 4b|-1}, p(L^t))$.

Table 1 summarizes the parameters of the three instances proposed.

U-QUARK is the lightest flavor of QUARK. It was designed to provide 128-bit preimage resistance and at least 64-bit security against all other attacks, and to admit a parallelization degree of 8. It has parameters r=8,c=128, b=136,n=136.



Function *f*. Given a 68-bit register,*x*,*f* returns:-

$X_0+X_9+X_{14}+X_{21}+X_{28}+X_{33}+X_{37}+X_{45}+X_{50}+X_{5+}X_{55}X_{59}$
$+X_{33}X_{37+}X_9X_{15+}X_{45}X_{52}X_{55+}X_{21+}X_{28+}X_{33}$
$X_9X_{28}X_{45}X_{59+}X_{33}X_{37}X_{52}X_{55+}X_{21}X_{28}X_{33+}X_9X_{28}X_{45}X_{59+}$
$X_{33}X_{37}X_{52}X_{55}+X_{15}X_{21}X_{55}X_{59+}X_{37}X_{45}X_{52}X_{55}X_{59+}$
$X_{37}X_{45}X_{52}X_{55}X_{59+}X_9X_{15}X_{21}X_{28}X_{33+}X_{21}X_{28}X_{33}X_{37}X_{45}X_{52}$.

D-QUARK is the second-lightest flavor of QUARK. It was designed to provide 160-bit preimage resistance and at least 80-bit security against all other attacks, and to admit a parallelization degree of 8. It has parameters $r = 16$, $c = 160$, $b = 176$, $n = 176$.

Function *f*. D-QUARK uses the same function / as U-QUARK, but with taps 0, 11, 18, 19, 27, 36, 42, 47, 58, 64, 67, 71, 79 instead of 0, 9, 14, 15, 21, 28, 33, 37, 45, 50, 52, 55, 59, respectively.

S-QUARK is the heaviest flavor of QUARK. It was designed to provide 224-bit preimage resistance and at least 112-bit security against all other attacks, and to admit a parallelization

degree of 16. It has parameters r = 32, c = 224, b = 256, n = 256.

Function *f*. S-QUARK uses the same function / as U-QUARK, but with taps 0, 16, 26, 28, 39, 52, 61, 69, 84, 94, 97,103,111 instead of 0, 9, 14, 15, 21, 28, 33, 37, 45, 50, 52, 55, 59, respectively.

3.Keying QUARK

As a sponge function, all results known on the sponge construction apply to QUARK. This includes proofs of security for keyed modes of operation, as described in . A keyed sponge function processes its input by simply hashing the string composed of the key followed by the said input. The following primitives can then be realized:

Message authentication code (MAC);
Pseudorandom generator;
Stream cipher;
Random-access stream cipher;
Key derivation function.

Furthermore, the QUARK instances can easily be modified to operate in the *duplex construction* (a variant of the sponge construction), to allow the realization of functionalities as authenticated encryption or

3.1.A Brief Description of the Present Block Cipher

Present is a 31-round SPN structure block cipher with block size of 64 bits, the cipher is described in fighre-. It supports 80 and 128 –bit secret key. Firstly, the plaintext Xored subkey K^1 as the input of the 1st round alter 31 rounds iterations, the 31st round output Xored with the subkey K^{32} is the cipher text.

Encryption Procedure. Each encryption round consists of the following 3 steps :-

(1). Add RoundKey –AK : At the beginning of each round 64 bits output of the last round function is Xored with the subkey.

(2) SBoxlayer–SL: The SL function $\{0,1\}$ maps input (x_0,x_1,x_2,x_3) to output (y_0,y_1,y_2,y_3) , 16 identical 4-bit S-boxes are used in parallel. The Boolean function of S-box is $y_0 = x_0 + x_2 + x_3 + x_1x_2$.

(3) Player PL : the i^{th} bit is moved to bit position P(i) by a constant permutation table Figure 1.Overview of Present Encryption Algorithm

4. *KEY DISTRIBUTION*

Cryptography has for a long time conformed to the idea that the techniques used to protect sensitive data had themselves to be kept secret. Such principle, known as "cryptography by obscurity" has however become inadequate in our modern era. Cryptography, that has developed as a science in the 1970s and 1980s allowed to move away from this historical picture and most of the modern cryptographic systems are now based on publicly announced algorithms while their security lies in the use of secret keys

Distributing keys among a set of legitimate users while guaranteeing the secrecy of these keys with respect to any potential opponent is thus a central issue in cryptography, known as the *Key Establishment Problem.*

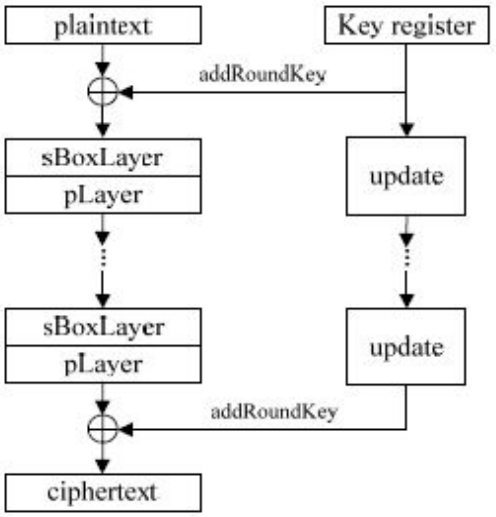


Fig 3. Over view of Present Encryption Algorithm

There are currently five families of cryptographic methods that can be used to solve the Key Establishment Problem between distant users:

1. Classical Information-theoretic schemes
2. Classical public-key cryptography
3. Classical computationally secure symmetric-key cryptographic schemes
4. Quantum Key Distribution
5. Trusted couriers

We will present how each of those cryptographic families can provide solutions to the Key Establishment problem and discuss, in each case, the type of security that can be provided. We will also consider a sixth type of Key Establishment schemes: hybrid schemes built by combining some of the methods listed above.

4.1.Key Establishment based on public-key cryptography:-

As shown by Whitfield Diffie and Martin Hellman in 1976 , public-key cryptography can be used to establish a shared secret key over an unprotected classical communication channel, without using a prior shared secret. It thus provides a practical way to implement key distribution over open networks.

4.1.1.Security of public-key cryptography Current asymmetric classical cryptographic schemes, such as RSA, are based on the difficulty to compute logarithms within a finite field. Today's implementations of RSA require to use private and public keys of at least 1024 bits, in order to offer a reasonable security margin against the computational efforts of an eavesdropper [1], and asymmetric keys of 2048 bits are preferable. It is also important to note that most of the currently used public-key cryptographic schemes (for example

RSA) could be cracked in polynomial time with a quantum computer: this results from Shor's algorithm for discrete log and factoring, that has a complexity of $O(n^3)$ [13].

4.1.2. Performance of public-key cryptography. Making the computations relative to the asymmetric cryptographic protocols (over keys longer than 1024 bits) is a rather computational intensive and time-consuming task. The performance of RSA-based key distribution implementations depend heavily on hardware : for RSA 2048 implemented on a recent PC (Pentium IV with a 2.1 GHz processor running under Windows XP), the computations needed for one key exchange (essentially one RSA encryption and one decryption) take roughly 30 ms . The same key exchange would be approximately 10 times faster (thus in the ms range) on dedicated coprocessors and 10 times slower (in the time range of a few tens of a second) on smart card coprocessors , Because of those relatively low exchange rates, public-key cryptography is most commonly used solely for initial session key distribution (in network protocols like SSL for example), and classical symmetric-key cryptography is then generally used for symmetric encryption and/or authentication of data.

4.1.3.ClassicalComputationally Secure Symmetric key Cryptography and key Establishment

Symmetric-key cryptography refers to cryptography methods in which both the sender and receiver share the same key. Symmetric-key encryption was the only kind of encryption publicly known until the discovery of public-key cryptography in 1976. Symmetric-key ciphers are used to guarantee the secrecy of the encrypted messages. The modern study of symmetric-key ciphers relates mainly to the study of block ciphers and stream ciphers and to their applications. AES is a block cipher that had been designed by a team of Belgium cryptographers (Joan Daemen et Vincent Rijmen) and has been adopted as an encryption standard by the US government (in replacement of DES). Block ciphers can be used to compute Message Authentication Codes (MACs) and can thus also be used to guarantee integrity and authenticity of messages. Stream ciphers, in contrast to the block ciphers, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the One-Time-Pad. We will not consider stream ciphers in the remaining part of this sub-section, since, unlike block ciphers, they cannot be easily used to perform Key Establishment.

4.1.4..Key Establishment based **on Classical Computationally.** Secure Symmetric-Key Cryptography Key Establishment can be realised by making use of only symmetric-key cryptographic primitives. Indeed, the combination of a symmetric-key encryption scheme with a symmetric-key authentication scheme allows one to build a Key Establishment primitive. Provided that a secret key is previously shared, symmetrically, by Alice and Bob, one can use a symmetric-key cipher to encrypt a message that will constitute the secret key for the key distribution protocol (this message can be random or not). Part of the previously shared symmetric key material can also be used to symmetrically compute (on Alice's side) and check (on Bob's side) a message authentication tag. Key Establishment based on symmetric-key cryptographic primitives are always based on a pre-established symmetric secret, needed for authentication. In this sense, they only allow *Key Expansion* more than *Key Establishment.*

4.1.5. Security of classical computationally secure symmetric-key cryptography. The security of key distribution based on classical symmetric-key cryptography depends on the security of the cryptographic primitives that are used, and on the composability of those crypto primitives. Shannon has proven that there is no unconditionally secure encryption scheme which requires less key than a One-Time Pad, i.e., the number of key bits is at least as large as the length of the message . Hence, if we consider the possibility of building an unconditionally secure symmetric key expansion scheme, i.e$_v$ a method to symmetrically generate secret key out of a short initial symmetric shared secret key, the former results from Shannon tell us that such a scheme is impossible to achieve in the framework of classical cryptography. This is a fundamental limitation of any communication scheme relying solely on the exchange of classical messages since, in contrast to quantum messages, classical messages can be copied without errors. It is however possible to use classical symmetric-key encryption and authentication schemes, that are not unconditionally secure, to build a Key Establishment scheme. AES can for example be used for symmetric-key encryption and can be also used to compute message authentication codes (using AES-MAC). Note that the security model that applies to such symmetric-key classical encryption schemes (symmetric-key block ciphers and stream ciphers) is not unconditional security (the entropy of the key is smaller than the entropy of the message) and not even "provable computational security" (based on some proven upper bounds or on some equivalence between the complexity of the crypt-analysis of a given cipher and another well-studied problem[2]). The security model that applies to classical symmetric-key cryptography can be called "practical computational security": a cryptographic scheme is considered "practically computationally secure" if the best-known attacks require too much resource (such as computation power, time, memory) by an acceptable margin The main problem with such a security model is that it is unable to guarantee anything about yet unknown attacks . There are no publicly known efficient quantum attacks on classical symmetric-key cryptographic schemes (but no proof that efficient attacks cannot be found), and the crypt-analysis of symmetric-key classical cryptography on a quantum computer reduces to exhaustive search. Here a quantum computer would thus still give an advantage: the complexity of exhaustive search in a unsorted database of N elements is of $O(N)$ on a classical computer but only of $O(VN)$ on a quantum computer.

Performances In terms of performance, symmetric-key classical cryptography is much faster and less computational intensive than asymmetric cryptography[3]. In terms of speed, there are now 128-bit AES encryptors able to encrypt data at rates in the Gbit/s range , This is the reason why it is widely preferred to use symmetric-key schemes for encryption and/or authentication over currently deployed communication networks. AES is currently the chosen standard for symmetric-key classical block ciphers. Under the assumption that the best way to break a symmetric-key cryptographic scheme is exhaustive search within the key space[4], then, a symmetric key modulus of 77 bits is roughly comparable, in terms of computational requirements, to an asymmetric key modulus of 2048 bits . Note that doubling the length of a symmetric key implies squaring the computational efforts needed for exhaustive search; on the other hand, the computational efforts scale not as fast with key length in the case of asymmetric cryptography.

5.NTRU

Description of NTRU :- NTRU is based on the algebraic structures of certain polynimal rings.The 'hard problem" on which NTRU is based is the Short Vector Problem (finding a short vector in a lattice).

a. Notation. Before we proceed, we set some notation. The following are all part of the domain parameters for an implementation of NTRU.

n The dimension of the polynomial ring used in NTRU. (The polynominals will have degree n-1.)

p A positive integer specifying a ring $Z/_pZ$ over which the coefficients of a certain product of polynomials will be reduced during the encryption and decryption processes.

q A positive integer specifying a ring $Z/_qZ$ over which the coefficients of a certain product of polynomials will be reduced during the encryption and decryption processes, also used in the construction of the public key.

k A security parameter which controls resistance to certain types of attack including plain text awareness.

d_f The distribution of the coefficients of the polynomial *f*, below (*f* is part of the private key).

d_g The distribution of the coefficients of the polynomial *g*, *below(**g** is used to construct the public key).

d_r The number of 1s and -1s used in a certain random polynomial r, below , in the encryption process.

We will also use the following notation.

f A polynomial in $Z^{[X]}$ (X^n-1)

f_p A polynomial in $Z^{[X]}/(p,X^n-1)$ this is part of the private key). This polynomial is obtained by reducing the coefficients of *f* modulo p.

f_q A polynomial in $Z^{[X]}/(q,X^n-1)$. This polynomial is obained by reducing the coefficients of *f* modulo *q.*

L_f The set of polynomials in $Z^{[X]}/(x^n-1)$ whose coefficients satisfy d_f.

A polynomial in $Z^{[X]}/(q,X^n-1)$ (used with f_q to construct the public key).

L_q The set of polynomials in $Z^{[X]}/(x^n-1)$ whose cofficients satisfy d_g.

L_r The set of polynomials in $Z^{[X]}/(x^n-1)$ whose coefficients satisfy d_r.

f_p^{-1}. The inverse of f_q in $Z^{[X]}/(q, X^n-1)$.

h The public key, a polynomial in $Z^{[X]}/(q,X^n-1)$.

r A polynomial in $Z^{[X]}$ (q,X^n-1 (used with *h* to encode a message).

m The plaintext message, a polynomial in $Z^{[X]}/(p,X^n-1)$.

e The encrypted message, a polynomial in $Z^{[X]}/(q,X^n-1)$.

G A *generating function* (defined below).

H A *hashing function (*defined below).

Case 2 and 3 show that none of the operations used in the key generation process is effective if the cipher key is made of all 0's or all 1's. These types of cipher keys need to be avoided, as discussed in Chapter 6.

S-DES is very vulnerable to brute-force attack because of its key size (10bits)

Throughout this paper, we work in the ring $R = \mathbb{Z}[X]/(x^n-1)$. An element $f \in R$ will be written as a polynormal or a vector.

$$f = \sum_{i=0}^{n-1} f_i x^i = [f_0, f_1, \ldots\ldots, f_{n-1}]$$

We write * to denote multiplication in R. This star multiplication is given explicitly as a cyclic convolution product,

$$f * g = h \text{ with}$$

$$h_k = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{n-1} f_i g_{n+k-i} = \sum_{i+j=k \bmod n} f_i g_j .$$

When we do a multiplication modulo (say) q, we mean to reduce the coefficients modulo q, so the result lies in $\mathbb{Z}[X]/(q, X^n-1)$.

Remark. The naive computation of a product $f * S$ requires n^2 multiplications. However, in a typical product used by NTRU, one of f or g has small coefficients that are all 0's and ± 1's, so $f * g$ may be computed extremely rapidly. Further, for large values of n one may choose n to be highly divisible by 2, in which case the convolution product can be computed in $O(n \log n)$ operations by using Fast Fourier Transforms.

In addition to this convolution product, there are two other operations we need to define on rings of polynomials. These are a generating function and a hashing function. They are

required in order to build a digital envelope into the NTRU protocol. We first let

$P_p(n)$ = {polynomials of degree at most $n-1$ with mode p coefficients}, and we will write.

$[g]_p$ { g with its coefficients reduced modulo p into the range $(-p/2, p/2)$ }.

We may now describe more precisely what we mean by a generating function G and a hashing function H,

$$G: P_p(N) \to P_p(N) \text{ and}$$

$$H: P_p(N) \times P_p(N) \to P_p(K)$$

These should be easy to compute, highly non-linear and unpredictable. There are numerous examples of such functions, constructed out of shits and other primitive operations, in the literature.

The NTRU PKC digital, envelope depends on the choice of the functions G and H, and on an integer k. The probability of forging a valid ciphertext will be p^{-k}.

Remark : The original presentation of NTRU [HPS] did not suggest the use of a digital envelope (i.e., in the present discussion, both G and H would be functions which, no matter what the input, produce an output of 0). This provides an insecure digital envelope as described in [NT7] (cf.[BKS]).

Key Creation. To create an NTRU key, Bob randomly choose 2 polynomials $f \in L_f$ and $g \in L_g$. The polynomial f must satisfy the additional requirement that it have inverses modulo q and modulo p. For suitable parameter choices, this will be true for most choices of f (see [NT9]), and the actual computation of these inverses is easy using a modification of the Euclidean algorithm (see [NT1, NT14] for details). As noted above, we will denote these inverses by f_q^{-1} and f_p^{-1}, that is

1) $f_q^{-1} * f \equiv 1 \bmod q$ and $f_p^{-1} * f \equiv 1 \bmod p$

Bob next computes the quantity

2) $h \equiv p f_q^{-1} * g \bmod q$.

Bob's public key is the polynomial h. Bob's private key is the polynomial f, although in practice he will also want to store f_p^{-1}. For an extremely efficient algorithm to compute f_p^{-1} and f_q^{-1}, please see [NT14]; for an efficient algorithm for multiplication, please see [NT10]

Encryption. We now describe how Alice wraps and sends a message to Bob using Bo's NTRU public key h. Alice chooses her plaintext m from the set

$$m \in P_p(n-k)$$

She also choosen a random polynomial $r \in L_r$. She computes

$$e \equiv r * h + \left[m + H\left(m, [r*h]_p\right) X^{n-k} + G\left(\left[r*h\right]_p\right) \right]_p \pmod q$$

Sender then sends e to Receiver.

6.1. Decryption. Suppose that Bob has received the message e from Alice and wants to decrypt it using his private key f. To do this efficiently, Bob should have precomputed the polynomial f_p^{-1} described in Section 1.1.

In order to decrypt e, Bob first computes the temporary polynomial a by

$a \equiv f * e \bmod q$, Throughout this paper , we work in the ring R = $Z^{[X]}/(X^n-1)$. An element $f \in R$ Will be written as a polynomial or a vector.

$f = \sum_{i=0}^{n-1} f_i x^i = [f_0, f_1, \ldots\ldots, f_{n-1}]$ .where he chooses the coefficients of a in the interval from $-q/2$ to $q/2$. Now treating a as polynomial with integer coefficients, Bob computes the temporary polynomial $t \in \square [X]/(p, X^n - 1)$

$t = f_p^{-1} \otimes a \pmod p$,

Further computes the two temporary quantities $b \equiv e - t \pmod p$ and $c \equiv t - G(b) \pmod p$,

And then writes c in the form $c = c'' + X^{n-k}$ with $\deg(c') < n-k$ and $\deg(c'') < k$.

(Note that the quantity b is supposed to play the role of $[r*h]_p$. Finally, he compares the quantities.

c'' and $H(c', b)$.

If they are the same, he accepts c' as a valid decryption. Otherwise he rejects the message as invalid.

Remark. For appropriate parameter values, there is an extremely high probability that the decryption procedure will recover the original message. However, some parameter choices may cause occasional decryption failure, so one should probably include a few check bits in each message block. The usual cause of decryption failure will be that the message is improperly centered. In this case Bob will be able to recover the message by choosing the coefficients of $a \equiv f \otimes e \bmod q$ in a slightly different interval, for example from $-q/2 + x$ to $q/2 + x$ for some small (positive or negative) value of x. If not value or x works, then we say that we have *gap failure* and the message cannot

be decrypted as easily. For well-chosen parameter values, this will occur so rarely that it can be ignored in practice.

6.Why Decryption Works. The polynomial a that Bob computes satisfies

$a \equiv f \otimes e$

mod q)

$= f \otimes pr \otimes f_q^{-1} \otimes g +$

$f \otimes \left[m + H\left(m, [r \otimes h]_p\right) X^{n-k} + G([r \otimes h]_p \right]_p$

mod q

From (1),

$= pr \otimes g + f \otimes$

$\left[m + H\left(m[r \otimes h]_p\right) X^{n-k} + G\left([r \otimes h]_p\right)\right]_p$ From (2).

Consider this last polynomial. For appropriate parameter choices, we can ensure that (almost always) all of its coefficients lie between $-q/2$ and $q/2$, so that it doesn't change if its coefficients are reduced modulo q. This means that when Bob reduces the coefficients of $f \otimes e$ modulo q into the interval from $-q/2$ and $q/2$, he recovers *exactly* the polynomial.

$a = pr \otimes g + f \otimes$

$\left[m + H\left(m[r \otimes h]_p\right) X^{ni-k} + G\left([r \otimes h]_p\right)\right]_p$ in R.

Reducing a modulo p then gives him the polynomial

$f \otimes \left[m + H\left(m, [r \otimes h]_p X^{n-k} + G([r \otimes h]_p\right)\right]_p$ in R,

And then multiplying by f_p^{-1} produces

$t = m + H(m, [r \otimes h]_p) X^{n-k} + G([r \otimes h]_p)$ in

$[X]/\left(p, X^n - 1\right)$.

Thus when Bob computes $b = e - t$ above, he is really recovering $b = r \otimes h$.

Therefore his computation of c yields

$c = m + H\left(m, [r \otimes h]_p\right) X^{n-k}$.

Accordingly, c' is the original message m, and c^n should match up with the hash $H(m, [r \otimes h]_p) = H(m, b)$, as noted above.

Parameter choices – notation and a norm estimate. We define the *width* of an element $f \in R$ to be

$|f|_\infty = \max_{0 \le i \le n-1} \{f_i\} - \max_{0 \le i \le n-1} \{f_i\}$.

As our notation suggests, this is a sort of L^∞ norm on R. Similarly, we define a *centered* L^2 *norm on* R by

$|f|_2 = \left(\sum_{i=0}^{n-1} \left(f_i - \bar{f} \right)^2 \right)^{1/2}$, where $\bar{f} = \frac{1}{n} \sum_{i=0}^{n-1} fi.$

(Equivalently, $|f|_2 / \sqrt{n}$ is the standard deviation of the coefficients of $F.$) The following proposition was suggested to us by Don Coppersmith.

Proposition. *For any* $\in > 0$ *there are constants* $\gamma 1, \gamma 2 > 0$, *depending on* $\in$ *and N, such that for randomly chosen polynomials* $f, g \in R$, *the probability is greater than* $1 - \in$ *that they satisfy*

$\gamma_1 |f|_2 |g|_2 \le |f \otimes g|_\infty \le \gamma_2 |f|_2 |g|_2$.

Of course, this proposition would be useless form a practical viewpoint if the ratio γ_2 / γ_1 were very large fro small $\in's$. However, it turns out that even for moderately large values on N and very small values of $\in$, the constants γ_1, γ_2 are not at all extreme. We have verified this experimentally for a large number of parameter values.

Sample spaces. The space of messages L_m consists of all polynomials modulo p. Assuming p is odd, it is most convenient to take

$$L_m = \left\{ \begin{array}{c} m \in R : m \text{ has coefficients lying between} \\ -\frac{1}{2}(p-1) \text{ and } \frac{1}{2}(p-1) \text{ and has degree at most } n-k-1 \end{array} \right\}$$

To describe the other sample spaces, we will use sets of the form

$$L(d_1, d_2) = \left\{ \begin{array}{c} f \in R: \ f \text{ has } d_1 \text{ coefficients equal } 1, \\ d_2 \text{ coefficients equal } -1, \text{ the rest } 0 \end{array} \right\}.$$

With this notation, we choose three positive integers d_f, d_g, d_r and set

$L_f = L\left(d_f, d_f - 1\right)$, $L_g = L\left(d_g, d_g\right)$, and $L_r = L\left(d_r, d_r\right)$.

(The reason we don't set $L_f = L\left(d_f, d_f\right)$ is because we want f to be invertible, and a polynomial satisfying $f(1) = 0$ can never be invertible) Notice that $f \in L_f$, $g \in L_g$, and $r \in L_r$ have L^2 norms

$$|f|_2 = \sqrt{2d_f - 1 - n^{-1}}, \qquad |g|_2 = \sqrt{2d_g},$$
$$|r|_2 = \sqrt{2d_r}.$$

Later we will give values for d_f, d_g, d_r that allow decryption while maintaining various security levels.

A Decryption Criterion. To ease notation, we let $m' = \left[m + H\left(m, [r \otimes h]_p\right) X^{n-k} + G\left([r \otimes h]_p\right) \right]_p$ Be the polynomial used by Alice for encryption. (That is, $e \equiv r \otimes h + m' \mod q.$) In order for the decryption process to work, it is necessary that

$|f \otimes m' + pr \otimes g|_\infty < q.$

We have found that this will virtually always be true if we choose parameters so that

$|f \otimes m'|_\infty \le q/4$ and $|pr \otimes g|_\infty \le q/4;$

In view of the above Proposition, this suggests that we take

(3) $|f|_2 |m|_2 \approx q/4\gamma_2$ and $|r|_2 |g|_2 \approx q/4p\gamma_2$

for a γ_2 corresponding to a small value for $\in$. For example, experimental evidence suggests that for $N = 167$ and $N = 503.$ appropriate values for γ_2 are 0.27 and 0.17 respectively.

Table : Table shows three cases of key generation-

Steps	**Illustration**		
Cipher Key	1011100110	0000000000	1111111111
Absorbing	1100101110	0000000000	1111111111
Squeezing	L: 11001	L : 00000	L : 11111
Squeezing	R : 01110	R : 00000	R : 11111
Round 1 :			
Shifted Keys :	L : 10011	L : 00000	L : 11111
Combined Key :	R : 11100	R : 00000	R : 11111
	1001111100	0000000000	1111111111
Round Key 1 :	101111100	0000000000	1111111111
Round 2 :			
Shifted Keys :	L ; 01110	L : 00000	L : 11111
	R : 10011	R : 00000	R : 11111
Combined Keys :	0111010011	0000000000	1111111111
Round Key : 2	11010011	0000000000	1111111111

CONCLUSION

In this paper we propose the secret message transmission between group using sponge function technique, every user has to register in order to communicate in the group. This entire process will be examined in KDC(Key Distribution Center) based on NTRU Technique. The Encryption and Decryption will be taken place.

REFERENCES

[1]. Gilles Van Assche. Errata for Keccak presentation. Email sent to the NIST SHA-3 mailing list on Feb 7 2011, on behalf of the Keccak team.

[2]. Itai Dinur, Tim G uneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An experimentally verified_attack on full Grain-128 using dedicated recon_gurable hardware. Cryptology ePrint Archive, Report 2011/282, 2011. to appear at Asiacrypt 2011.

[3] .Guang Gong and Kishan Chand Gupta, editors. Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings,volume 6498 of LNCS. Springer, 2010.

[4]. Simon Knellwolf, Willi Meier, and Mar__a Naya-Plasencia. Conditional di_erential cryptanalysis of NLFSR-based cryptosystems. In Masayuki Abe, editor, ASIACRYPT, volume 6477 of LNCS, pages 130{145. Springer, 2010.

[5]. Jean-Philippe Aumasson, Itai Dinur, Luca Henzen, Willi Meier, and Adi Shamir. E_cient FPGA implementations of highly-dimensional cube testers on the stream cipher Grain-128. In SHARCS, 2009.

[6]. Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the EMD transform. In Xuejia Lai and Kefei Chen, editors, ASIACRYPT, volume 4284 of LNCS, pages 299{314. Springer, 2006.

[7]. Martin _Agren, Martin Hell, Thomas Johansson, and Willi Meier. A new version of Grain-128 with authentication. In ECRYPT Symmetric Key Encryption Workshop 2011. Available at http://skew2011.mat.dtu.dk/.

[8]. Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In Phillip Rogaway, editor, CRYPTO, volume 6841 of LNCS, pages 222{239. Springer, 2011.

[9]. Simon Knellwolf, Willi Meier, and Mar__a Naya-Plasencia. Conditional di_erential cryptanalysis of Trivium and KATAN. In Selected Areas in Cryptography, 2011. to appear.

[10]. Guido Bertoni, Joan Daemen, Micha el Peeters, and Gilles Van Assche. Keccak sponge function family main document (version 2.1). Submission to NIST (Round 2), 2010.

[11]. Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. Cryptology ePrint Archive, Report 2010/532, 2010. Full version of [25].

[12]. Daniel J. Bernstein. CubeHash appendix: complexity of generic attacks. Submission to NIST, 2008. http://cubehash.cr.yp.to/submission/generic.pdf

[13]. Guido Bertoni, Joan Daemen, Micha el Peeters, and Gilles Van Assche. On the indi_erentiability of the sponge construction. In Nigel P. Smart, editor, EUROCRYPT, volume 4965 of LNCS, pages 181{197. Springer, 2008.