Post-Quantum Primtives For Constrained Environments

Abhishek Gautam*, Armaan Goyal†, Riaz Munshi‡ and Vyom Chhabra§
Applied Cryptography and Computer Security
Department of Computer Science and Engineering
The State University of New York at Buffalo, Buffalo, NY 14260-1660
* Email: agautam2@buffalo.edu
† Email: armaango@buffalo.edu
‡ Email: riazmuns@buffalo.edu
§ Email: vyomchha@buffalo.edu

***Abstract*—Quantum Computers have the power to use algorithms that break classical public-key algorithms whose secrecy depends on some hard mathematical problems, unsolvable by today's computers. Existence of quantum computers will threaten the entire security system that exists and leads to major work being done on post-quantum cryptographic algorithms which refers to using conventional non-quantum cryptographic algorithms that remain secure even after practical quantum computing is a reality in few years. While several methods have been derived over the years in developing post-quantum cryptographic algorithms but at a cost of large key-size and algorithm primitives that are not feasible given the time consumed in key generation and encryption. We propose to analyze security characteristics in 1. Code-based 2. Hash-based 3. Multi-Variate Quadratic 4. Lattice Based Post-Quantum Cryptographic Algorithms. We further deal with lattice based cryptographic signature schemes and work on mitigating Post-Quantum primitives to resource-constrained devices and analyze security characteristics in great details.**

I. Introduction

Imagine that its fifteen years from now and someone announces the successful construction of a large quantum computer. Which would necessary mean that all of the public-key algorithms used to protect the Internet have been broken. But, could this be an end to the word cryptography, the science behind the secrecy of communication over public networks like the Internet.Perhaps, after seeing quantum computers destroy RSA and DSA and ECDSA, Internet users will leap to the conclusion that cryptography is dead; that there is no hope of scrambling information to make it incomprehensible to, and unforgettable by, attackers; that securely storing and communicating information. A closer look reveals, however, that there is no justification for the leap from quantum computers to destroy RSA and DSA and ECDSA.

Post-Quantum Cryptographic algorithms that can be optimized to work on highly constrained hardware is an active area of research and we propose to do a comparative study on the various post-quantum security schemes that have been studies over the years, their security proof and optimizations that could be made to make these algorithms lightweight yet post-quantum safe.

Next, we discuss Lattice-based cryptography and Hash-based cryptography and describe our motivation to work towards various Lattice-based models.

A. Lattice-based cryptography

This approach includes cryptographic systems such as Learning with Errors, Ring-Learning with Errors (Ring-LWE), the older NTRU or GGH encryption schemes, and the newer NTRU signature and BLISS signatures. Some of these schemes like NTRU encryption have been studied for many years without anyone finding a feasible attack. Others like the Ring-LWE algorithms have proofs that their security reduces to a worst-case problem.

B. Hash-based cryptography

This includes cryptographic systems such as Lamport signatures and the Merkle signature scheme. Hash based digital signatures were invented in the late 1970s by Ralph Merkle and have been studied ever since as an interesting alternative to number theoretic digital signatures like RSA and DSA. Their primary drawback is that for any Hash based public key, there is a limit on the number of signatures that can be signed using the corresponding set of private keys. This fact had reduced interest in these signatures until interest was revived due to the desire for cryptography that was resistant to attack by quantum computers. There appear to be no patents on the Merkle signature scheme and there exist many non-patented hash functions that could be used with these schemes.

Our motivation is, doing comparative implementations of lattice based cryptography techniques focusing on their versatile average case problem. We have focused on the comparative study of the implementation of Ring-LWE and bi-modal lattice signature schemes (BLISS) which appears to be promising lattice based security schemes that could work on constrained hardware. Since the number theoretic transform (NTT) is one of the core components in implementations of lattice-based cryptosystems, we will be using Raspberry Pi B+ Version and some basic Networking Hardware and reviewing techniques to apply approximations to get similar results at a shorter time, by attempting to reducing time of the polynomial multiplication. We also touch upon the security proofs of lattice based cryptography, bounding the closeness of two probability distributions (e.g., zero centered and non-zero centered discrete Gaussian distributions)

The document is divided into 6 sections. Section I covered the introduction. Section II discusses some preliminaries. Section III has related work followed by the implementation details in Section IV. Section V discusses evaluations and results and lastly we have our conclusions and findings.

II. Preliminaries

A. The Post Quantum Era

Quantum computing studies theoretical computation systems which make direct use of quantum-mechanical phenomena, such as superposition and entanglement, to perform operations on data. Quantum computers are different from digital electronic computers based on transistors. Whereas digital computers require data to be encoded into binary digits (bits), each of which is always in one of two definite states (0 or 1), quantum computation uses quantum bits (qubits), which can be in superpositions of states Post-quantum cryptography refers to cryptographic algorithms (usually public-key algorithms) that are thought to be secure against an attack by a quantum computer.

B. Lattice

A lattice L in real analysis is a set of points in the n-dimensional Euclidean space R_n with a strong periodicity property.

C. Lattice Based Cryptography

Lattice-based cryptography is the generic term for asymmetric cryptographic primitives based on lattices. While lattice-based cryptography has been studied for several decades, there has been renewed interest in lattice-based cryptography as prospects for a real quantum computer improve. A basis of L is a set of vectors such that any element of L is uniquely represented as their linear combination with integer coefficients.

D. Constrained Device

Small devices with limited CPU, memory, and power resources, so called constrained devices (also known as sensor, smart object, or smart device).

E. Shortest Vector Problem (SVP)

In SVP, a basis of a vector space V and a norm N (often L^2) are given for a lattice L and one must find the shortest non-zero vector in V, as measured by N, in L. In other words, the algorithm should output a non-zero vector v such that $N(v) = \lambda(L)$.

In the γ-approximation version SVP_γ, one must find a non-zero lattice vector of length at most $\gamma\lambda(L)$.

F. Closest Vector Problem (CVP)

In CVP, a basis of a vector space V and a metric M (often L^2) are given for a lattice L, as well as a vector v in V but not necessarily in L. It is desired to find the vector in L closest to v (as measured by M). In the γ-approximation version CVP_γ, one must find a lattice vector at distance at most γ.
The CVP is a generalization of the SVP. It is easy to show that given an oracle for CVP_γ, one can solve SVP_γ by making some queries to the oracle.

G. Learning with Errors (LWE)

Learning with errors (LWE) [2] is a generalization of the parity learning problem, in machine learning that is conjectured to be hard to solve. An algorithm is said to solve the LWE problem if, when given access to samples (x,y) where $x \in \mathbb{Z}_q^n$ and $y \in \mathbb{Z}_q$, with the assurance, for some fixed linear function $f : \mathbb{Z}_q^n \to \mathbb{Z}_q$, that $y = f(x)$ with high probability and deviates from it according to some known noise model, the algorithm can recreate f or some close approximation of it with high probability.

III. Related Work

Since the seminal work of Ajtai , lattice-based cryptography has attracted much attention from the cryptography community. Provably-secure lattice-based schemes have a remarkable feature that their securities are proved assuming a basic lattice problem is hard in the worst-case. This type of assumption is more reliable than average-case hardness assumptions in number-theoretic schemes supported by a security proof. Moreover, lattice-based cryptography is one of the main candidates for post-quantum cryptography. No efficient quantum algorithm has been found yet to break lattice-based schemes. In contrast, widely used schemes in practice, such as RSA or elliptic curve (EC) based constructions which are based on the hardness of factoring or discrete logarithm, will be broken upon appearance of large-scale quantum computers.

Early proposed lattice-based schemes with a heuristic security analysis were more efficient than RSA and EC-based ones. However, initial provably-secure lattice-based schemes were suffering from heavy computation and large key sizes. A major event in the development of lattice-based cryptography is the introduction of ideal lattices. An ideal lattice has extra algebraic structure which reduces the key size and computation time to a sub-quadratic order. Moreover, cryptographic schemes based on ideal lattices enjoy a security proof assuming worst-case hardness of basic problems on ideal lattices. It is in contrast to the NTRU encryption, which is a well-known and efficient lattice-based scheme. The security of NTRU is not proven and is preserved heuristically.

Recent noticeable improvements on the efficiency of provably-secure lattice-based constructions have made it possible to port these schemes to constrained devices such as smart cards and micro-controllers.

The performance results on the smart card as per the previous work [1], show that the decryption of LP-LWE, as a provably secure and patent-free lattice-based encryption, is

4.4 times faster than the NTRU decryption, and its encryption performance is comparable to NTRU. Moreover, the key generation of NTRU is much slower. Results in Section 5.3 indicate that the LP-LWE decryption is also 1.8 times faster than a simplified CPA-secure version of NTRU. This shows that recent progress on improving efficiency of provably-secure lattice-based cryptography has made it comparable to and even better than the famous NTRU crypto-system.

IV. Implementation

A. NTRU

NTRU is a post quantum safe cryptosystem which works basically on the truncated ring polynomial algorithm. The Encryption process uses a mixing system based on polynomial algebra and reduction modulo two numbers p and q, while the decryption procedure uses an unmixing system whose validity depends on probability theory. The security of the NTRU public key cryptosystem comes from the interaction of the polynomial mixing system with the independence of the reduction modulo p and q. Security also relies on the fact that for most lattices, it is very difficult to find extremely short vectors. NTRU fits into the general framework of a probabilistic cryptosystem. This means that encryption includes a random element, so each message has many possible encryptions. Encryption and decryption with NTRU are extremely fast and key creation is fast and easy. NTRU takes O(N^2) operations to encrypt or decrypt a message block of length N, making it considerably faster than the O(N^3) operations required by RSA. Further, NTRU key lengths are O(N) which compares well with the O(N^2) key lengths required by other fast public key systems.

Next we explain in brief the basic Key Gen, Enc and Dec algorithms for NTRU.

KEY Creation: To create an NTRU key, we choose 2 random polynomials f,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 and the actual computation of these inverses is easy using a modification of the Euclidean algorithm. We will denote these inverses by F_q and F_p, that is,

$$F_q * f \equiv 1 \pmod{q} \text{ and } F_p * f \equiv 1 \pmod{p}$$

Next we compute the quantity

h$\equiv F_q * g \pmod{q}$
The public key is the polynomial h and private key is the polynomial f.

Encryption: Now when a user needs to send a message, he begins by selecting a message m from the set of plaintexts L_m. Next he randomly chooses a polynomial $\phi \in L_\phi$ and uses the public key generated in the previous step (h) to compute

$$e \equiv p\phi * h + m \pmod{q}.$$

This is the encrypted message transmitted by the sender to the receiver.

Decryption: Now when the receiever has received the message e from the sender and he wants to decrypt it using his private key f as derived earlier in the key generation process. To do this efficiently, he should have pre computed the polynomial F_p.
In order to decrypt e he first computes,
$a \equiv f * e \pmod{q}$,
where he chooses the coefficients of a in the interval from $-q/2$ to $q/2$. Now treating a as a polynomial with integer co efficients, he recovers the message by computing,

$$F_p * a \pmod{p}.$$

Practical Implementations of NTRU: Here we present three distinct sets of parameters which yield different levels of security. the norms of f and g have been chosen so that decryption failure occurs with probablilty less than 5.10^{-5}
Case A: Moderate Security: The moderate security parameters are suitable for situations in which the intrinsic value of any individual message is small and in which keys will be changed with reasonable frequency. Examples might include encrypting of television pager and cellular telephone transmissions.
(N,p,q)= (107,3,64)
$L_f = L(15,14), L_g = L(12,12), L_\phi = L(5,5)$

These give key sizes:
Private key= 340 bits and Public key = 642 bits,
and (meet in the middle) security levels
Key security = 2^{50} and Message security = $2^{26.5}$

Case B: High Security
(N,p,q)= (167,3,128)
$L_f = L(61,60), L_g = L(20,20), L_\phi = L(18,18)$
Private key = 530 bits and Public key = 1169 bits
Key security = $2^{82.9}$ and Message security = $2^{77.5}$

Case C: Highest Security
(N,p,q)= (503,2,256)
$L_f = L(216,215), L_g = L(72,72), L_\phi = L(55,55)$
Private key = 1595 bits and Public key = 4024 bits
Key security = 2^{285} and Message security = 2^{170}

B. U-LP

As previously stated, lattice-based cryptography is the generic term for asymmetric cryptographic primitives based on lattices.
LP relies on the hardness of the learning with errors (LWE) problem and uses discrete Gaussian sampling for noise and secret generation. Lindner and Peikert proposed their provably secure LP encryption scheme, which is based on the LWE problem and samples error from a discrete Gaussian distribution $D_{z\sigma}$ with standard deviation σ.

We performed analyses on the U-LP cryptosystem code [3] repository and that creates a generic library for using this cryptosystem in practice. Post analysis we have implemented the key generation encryption, decryption in Python (for the normal variant as well as for the ring-LWE variant).

The public and private keys are generated by sampling random data and error rates e_1, e_2, e_3 are randomly sampled from Gaussian Distribution.

U-LP, gathers noise (error), and secret from a uniform distribution, instead of a discrete Gaussian. This allows a 1. Simpler implementation, 2. Precludes decryption failures and 3. Gives hope for more efficient operations due to the simpler sampling. U-LP will require a bigger data set so it conforms to the security standards. U-LP is worst-case secure regarding standard lattice problems. In U-LP an equivalent ring based analogue was proposed that lead to notable smaller key sizes and performance increases. Below are the mathematical notations U-LP Key Generation, Encryption and Decryption and for its Ring Key analogue:

- **Key Generation:** Sample $A \leftarrow U_q^{nXn}$, $E \leftarrow U_{s_k}^{lXn}$ Compute $P = E - SA \in Z_q^{lXn}$. The public key is (A, P), the private key is S.
- **Encryption:** To encrypt an l bit message m, sample $e_1 \leftarrow U_{s_e}^n$, $e_1 \leftarrow U_{s_e}^n$ $e_3 \leftarrow U_{s_e}^l$, and set $m' = encode(m) \in Z_q^l$. The ciphertext $c = (c_1, c_2)$ is computed as follows: $c_1 = Ae_1 + e_2$ and $c_2 = Pe_1 + e_3 + m'$
- **Decryption:** For decryption, compute and return $decode(Sc_1 + c_2) \in Z_2^l$.

Mathematical notations related to Ring U-LP:

- **Ring Key Generation:** Sample $a \leftarrow U_q^n$, $e \leftarrow U_{s_k}^n$, and $s \leftarrow U_{s_k}^n$. Calculate $p = e - sa \in R_q$ The public key is (a, p), the private key is s
- **Ring Encryption:** To encrypt n bit message m, sample $e_1 \leftarrow U_{s_e}^n$, $e_2 \leftarrow U_{s_e}^n$ and $e_3 \leftarrow U_{s_e}^n$ Set $m' = encode(m)$ The ciphertext $c = (c1, c2)$, is computed as follows: $c1 = ae1 + e2$ and $c2 = pe1 + e3 + m'$.
- **Ring Decryption** For decryption, compute and return $decode(sc1 + c2) \in Z_2^n$.

C. LP

Lindner and Peikert proposed their provably secure LP encryption scheme, which is based on the LWE problem and samples error from a discrete Gaussian distribution $D_{Z,\sigma}$ with standard deviation σ. In the following, n denotes the security parameter, l the message length, and q the modulus. Additionally, a pair of error-tolerant encoding/decoding functions are part of the implementation.

Cryptosystem

The cryptosystem involves a few parameters: an integer modulus $q \geq 2$ and integer dimensions $n1, n2 \geq 1$, which relate to the underlying LWE problems; Gaussian parameters $s_k ands_e$ for key generation and encryption, respectively; and a message alphabet $\sum$ (for example, $\sum$ = 0, 1) and message length $l \geq 1$. We also require a simple error-tolerant encoder and decoder, given by functions encode : $\sum \rightarrow Z_q$ and decode : $Z_q \rightarrow \sum$, such that for some large enough threshold $t \geq 1$, decode(encode(m) + e mod q) = m for any integer $e \in [t, t)$. For example, if $\sum$ = 0, 1, then we can define encode(m) := m.$\lfloor \frac{q}{2} \rfloor$, and decode($\overline{m}$) := 0 if $\overline{m} \in [\frac{q}{4} \subset Z_q$, and 1 otherwise. This method has error tolerance $t = \lfloor \frac{q}{4} \rfloor$. We also extend encode and decode to vectors, component-wise.

To get the smallest public keys, our system makes use of a uniformly random public matrix $\overline{A} \in Z_q^{n1n2}$ that is generated by a trusted source, and is used by all parties in the system. If there is no trusted source, then $\overline{A}$ may be chosen by the users themselves as part of key generation, and included in the public key. Next, we will describe the mathematical notations and the algorithm related to U-LP cyptosystem:

GEN($\overline{A}, 1^l$): choose $R_1 \rightarrow D_{Z,s_k}^{n_1 \times l}$ and $R_2 \rightarrow D_{Z,s_k}^{n_2 \times l}$. and let P = $R_1 - \overline{A}.R_2 \in Z_q^{n_1 \times l}$ The public key is P (and $\overline{A}$, if needed), and the secret key is R_2. In matrix form, the relationship between the public and secret keys is:

$$\begin{bmatrix} \overline{A} & P \end{bmatrix} . \begin{bmatrix} R_2 \\ I \end{bmatrix} = R_1 \text{ mod q.}$$

ENC($\overline{A}$,P,m $\in \sum^l$): choose $e = (e_1, e_2, e_3) \in Z^n1 \times Z^n2 \times Z^l$ with each entry drawn independently from D_{Z,s_e}. Let $\overline{m}$ = encode(m) $\in Z_q^l$, and compute the ciphertext

$$c^t = \begin{bmatrix} c_1^t & c_2^t \end{bmatrix} = \begin{bmatrix} e_1^t & e_2^t & e_3^t + \overline{m}^t \end{bmatrix} . \begin{bmatrix} \overline{A} & P \\ I & \\ & I \end{bmatrix} \in Z_q^{I \times (n_2 + l)}$$

DEC($c^t = \begin{bmatrix} c_1^t & c_2^t \end{bmatrix}, R_2$): output decode $(c_1^t.R_2 + c_2^t)^t \in \sum^l$. Using Equation in ENC followed by Equation in GEN, we are applying decode to

$$\begin{bmatrix} c_1^t & c_2^t \end{bmatrix} . \begin{bmatrix} R_2 \\ I \end{bmatrix} = (e^t + \begin{bmatrix} 0 & 0 & \overline{m}^t \end{bmatrix}). \begin{bmatrix} R_1 \\ R_2 \\ I \end{bmatrix} = e^t.R + \overline{m}^t$$

where R = $\begin{bmatrix} R_1 \\ R_2 \\ I \end{bmatrix}$. Therefore, decryption will be correct as long as each —$\langle e, r_j \rangle$— ¡ t, the error threshold of decode.

D. BLISS

In this work we only consider the efficient ring-based instantiation of BLISS. Key generation requires uniform sampling of sparse and small polynomials f, g, rejection sampling (N(S)), and computation of an inverse. To sign a message, two masking polynomials y1, y2 DZn,sigma are sampled from a discrete Gaussian distribution using the SampleGauss function. The computation of ay1 is performed using the NTT and the compressed u is then hashed together with the message by Hash. The binary string c 0 is used by GenerateC to generate a sparse polynomial c. Polynomials y1, y2 then hide the secret key which is multiplied with the sparse polynomials using the SparseMulfunction. This function exploits that only coefficients in c are set and only d1 + d2 coefficients in s1 and s2.

After a rejection sampling and compression step the signature (z1, z 2 , c) is returned. The verification procedure BLISSver in Algorithm 6 just checks norms of signature components and compares the hash output with c in the signature. In this work we focus on the 128-bit prequantum secure BLISS-I parameter set which uses n = 512 and q = 12289 (same base parameters as RLWEencIIa). The density of the secret key is 1 = 0.3 and 2 = 0, the standard deviation of the coefficients of y1 and y2 is = 215.73 and the repetition rate is 1.6. The number of dropped bits in z2 is d = 10, = 23, and p = b2q/2 d c. The final size of the signature is 5,600 bits with Huffman encoding and approx. 7,680 bits without Huffman encoding.

We did a comparison-based study of Hash and Sign signature schemes and Fiat Shamir signature schemes and analysed the performance of both the schemes. Our findings are: Hash and Sign Scheme is Less efficient and its implementation is cumbersome where as Fiat Shamir Signature Schemes are widely understood and they are easy to implement both in software and hardware. Also There are no known serious theoretical attacks on these schemes. They offer features like - high security levels and short signatures/keys, linear impact on performance when scaling parameters, low-cost implementation on ASICs/RFIDs, vulnerability against physical attacks countermeasures. Bliss is also being early adopted. The 3rd party VPN solution, strongswan already has support for Bliss. So this is clearly an interesting candidate for standardization.

To get a better understanding of the security proof of Bliss, we did some research on the security analysis of Bliss by going through the available material. We studied some combinatorial attacks such as the Brute Force and Meet in the Middle Attack. Here, internal structure is unimportant. An attacker requires the ability to encrypt and decrypt, and the possession of pairs of plaintexts and corresponding ciphertexts. We also studied some lattice reduction attacks and the theory of the hermite constant. We ended our study with some hybrid attacks.

We have further studied the various building blocks of the bliss implementation to study the performance characteristics in detail. One of the most fundamental building block of the bliss algorithm is computation of the Numerical Theoretical Transform (NTT) which central role lies in computing the polynomial multiplication. One of the straightforward implementation of NTT that we have studies is Cooley-Tuckey radix-2 decimation-in-time (DIT) approach , which involves a bit reversal step whereby the we bit reverse the input in ordered to produce naturally ordered output.To compute the NTT as defined in Section 2.1 the NTTCT bo $\leftarrow$no algorithm applies the Cooley-Tukey (CT) butterfly, which computes a 0 $\rightarrow$ a + φb and b 0 $\rightarrow$ a φb for some values of φ, a, b Zq, overall n log2 (n) 2 times. The biggest disadvantage of relying solely on the NTTCT bo$\rightarrow$no algorithm is the need for bit-reversal, multiplication by constants, and that it is impossible to merge the final multiplication by powers of $\rightarrow$ 1 into the twiddle factors of the inverse NTT. With the assumption that twiddle factors (powers of $\rightarrow$) are stored in a table and thus not computed on-the-fly it is possible to further simplify the computation and to remove bit-reversal and to merge certain steps.

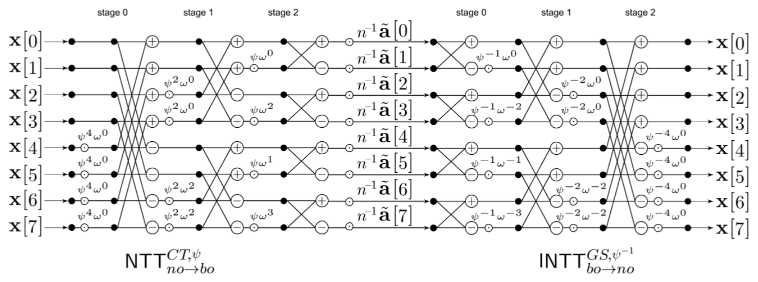
stage 0
stage 1
stage 2
$n^{-1}\tilde{a}[0]$
$\mathbf{x}[0]$
$\mathbf{x}[7]$
$\mathsf{NTT}^{CT,\psi}_{no\rightarrow bo}$
$\mathsf{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$

Fig. 1. NTT Implementation

Polynomial multiplication is crucial for overall performance.Cooley-Tukey decimation-in-time NTT algorithm requires bit-reversa n/2log$_2$(n) multiplication in Z$_q$.Other ingredients that affect the implementation of this scheme are Discrete Gaussian and Discrete Uniform sampling.
The rate of Gaussian Sampling can be improved by avoiding large tables and costly evaluation of exponential functions.Some of the popular sampling techniques being:

1) Rejection Sampling (straight and expensive)
2) Bernoulli (quite efficient and fast)
3) Discrete Ziggurat (moderately fast)
4) Knuth-Yao (moderately large tables)

Use of Cumulative Distribution Tables , which make use of Convolution theorem to combine values from smaller tables and also implements guide table to accelerate sampling process is the start-of-the-art practice to generate effective Gaussian sampling.

We further look into the efficient implementation of gaussian sampling for constrained devices, which is crucial for running in pace with the hardware constraints.Since its introduction, and with the noticeable exception of NTRU, lattice-based cryptosystems operating at a standard security level have remained out of reach of constrained devices by several orders of magnitude. A first step towards a practical lattice-based signature scheme was achieved by [BLISS] with an implementation on a low-cost FPGA, by avoiding Gaussians, at the cost of some compactness and security compared to [NTRU]. At this time, all known algorithms to sample according to a distribution statistically close to a discrete Gaussian distribution on a lattice require either long-integer arithmetic at some point or large memory storage. Some progress was made in [LCP], showing that lazy techniques can limit the need for high precision; one can use floating-point numbers at double precision (53 bits) most of the time, native on high-end architectures but costly on embedded devices. Section Outline. The main goal of this section is to show how to efficiently sample discrete Gaussian without resorting to large precomputed tables, nor evaluations of transcendental function. The first step is being able to sample according to a Bernoulli distribution with bias of the form exp(x/f) (and 1/ cosh(x/f)) without actually computing transcendental functions. The second step is to build an appropriate and efficient distribution as input of rejection sampling to reduce its rejection rate. Our new algorithm still requires precomputed tables, but of much smaller size; precisely of size logarithmic in rather than linear.

Now we describe the final algorithms needed to realize BLISS.

```
Key pair (A,S) such that AS = q mod 2q
Choose f,g as uniform polynomials with $d_1$ entries in $\{\pm 1\}$ and $d_2$ entries in $\{\pm 2\}$
$S = (s1, s2)^t \leftarrow (f, 2g+1)^t$
if $N_k(S) \geq C^2.5.(\lceil \delta_1 n \rceil + 4\lceil \delta_2 n \rceil).K$ then
    restart
end if
$a_q$ = (2g+1)/f mod q (restart if f is not feasible)
return (A,S) where A = (2$a_q$,q-2) mod 2q
```

Algorithm 1: BLISS Key Generation

```
Require: Message $\mu, publickeyA = (a_1, q-2) \in R_{2q}^{1x2}, secretkeyS = (s_1, s_2)^t \in R_{2q}^{2x1}$
Ensure: A signature $(z_1, z_2, c) of the message \mu$
    $y_1, y+2 \leftarrow D_z,$
    $u = a_1.y_1 + y_2$ mod 2q
    $c \leftarrow H(\lceil u \rceil modp, \mu)$
    Choose a random bit b
    $z_1 \leftarrow y_1 + (-1)^b.s_1.c$
    $z_2 \leftarrow y_2 + (-1)^b.s_2.c$
    Continue with probability $1/(Mexp\left(\frac{-\|Sc\|^2}{2\sigma^2}\right)^2)$ otherwise restart
    $z_2 \leftarrow (\lceil u \rceil - \lceil u - z_2 \rceil)$ mod p
```

Algorithm 2: BLISS Signature Algorithm

```
Require: $Message \mu, publickeyA = (a, q-2) \in R_{2q}^{1x2}, signature(z_1, z_2, c)$
Ensure: Accept or Reject the signature
    if $\| (z_1 | 2^d.z_2) \| > B_2$ then
        Reject
    end if
    if $\| (z_1 | 2^d.z_2) \| > B_\infty$ then
        Reject
    end if
    Accept if c = $H(\lceil a_1.z_1 + q.c \rceil + z_2 modp, \mu)$
```

Algorithm 3: BLISS verification algorithm

V. Evaluation and Discussion

We have conducted extensive analysis and testing of running of post-quantum safe as well as non post-quantum algorithms on modern computers as well as constrained devices. We have run classical cryptography algorithms like RSA, ECDSA and measured the performance characterestics and benchmarking on key parameters like time consumed for (1) Key Generation (2) Time taken for Encryption (3) Time taken for decryption (4) Time taken to verify a signature and so on. Following are some of the results that were obtained running the following algorithms in two different types of hardware platforms (un-constrained and constrained). We also plot the characteristics of the study , following the results.

1) Post-Quantum Unsafe methods

RSA:

a) Below are the experimental results for RSA on a 64 bit i5 system

Str Size	SignKeygen/sec	Sign	Verify
512	19003	0.000015s	0.000009s
1024	6098	0.000124s	0.000080s
2048	1145	0.001529s	0.000910s
4096	185	0.125789s	0.004777s
8192	38	0.824619s	0.010001s

b) Below are the experimental results for RSA on a ARMv7 cortex CPU (700Mhz) with 1 GB of RAM.

Str size	SignKeygen/sec	Sign	Verify
512	4364.5	0.002215s	0.000229s
1024	1540.3	0.011254s	0.000649s
2048	439.5	0.073529s	0.002276s
4096	115.7	0.535789s	0.008645s
8192	33.4	0.824619s	0.010645s

ECDSA:

a) Below are the experimental results for ECDSA after running on a 64 bit i5 system

String size	SignKeygen	Sign	Verify
48	0.160	0.058	0.116
56	0.230	0.086	0.165
64	.305	0.112	0.220
96	.801	0.289	0.558s
132	1.582	0.584	1.152

b) Below are the experimental results for ECDSA after running on a ARMv7 cortex CPU (700Mhz) with 1 GB of RAM.

String size	SignKeygen	Sign	Verify
48	0.520	0.201	0.574
56	0.890	0.620	0.782
64	.1205	0.512	0.926
96	3.206	0.899	2.001
132	4.782	2.012	4.789

2) Post-Quantum Safe methods
NTRU:

a) Below are the experimental results for NTRU after running a java based implementation on a 64 bit i5 system

String size	SignKeygen/sec	Sign/sec	Verify/sec
256	.49	947.87	950.5
256	.45	765.56	740.8
128	.57	879.4	869.8
128	.52	734.5	768.7
64	.67	1102	1105
64	.64	1056.7	1033.5

b) Below are the experimental results for NTRU after running a java based implementation on a ARMv7 cortex CPU (700Mhz) with 1 GB of RAM.

String size	SignKeygen/sec	Sign/sec	Verify/sec
256	.01	212.12	200.5
256	.01	120.56	156.8
128	.02	140.4	123.8
128	.02	230.5	189.7
64	.02	456	250
64	.02	500.7	256.5

U-LP:

a) Below are the experimental results for U-LP, running on a 64bit i7 system

Sec Param	String Size	Enc/sec	Dec/sec
256	256	219.338	2037.971
256	1000	231.893	2085.035
328	256	171.537	1629.108
328	1000	183.184	1625.030
412	256	136.768	1297.041
412	1000	148.146	1298.702

b) Below are the experimental results for U-LP , running on a ARMv7 cortex CPU (700Mhz) with 1 GB of RAM.

Sec Param	String Size	Enc/sec	Dec/sec
256	256	37.21	211.71
256	1000	39.119	204.54
328	256	29.89	152.81
328	1000	27.90	149.30
412	256	19.1	145.0
412	1000	20.1	143.8

LP:

a) Below are the experimental results for LP, after running on a 64bit i7 system.

Sec Param	String Size	Enc/sec	Dec/sec
80	256	786.040	6732.430
80	1000	807.5927	6886.294
128	256	478.147	4183.339
128	1000	505.622	4378.187
160	256	382.037	3425.254
160	1000	394.996	3525.049

b) Below are the experimental results for LP , running on a ARMv7 cortex CPU (700Mhz) with 1 GB of RAM.

Sec Param	String Size	Enc/sec	Dec/sec
80	256	106.040	980.450
80	1000	98.3	910.123
128	256	63.2	612.89
128	1000	75.78	654.89
160	256	53.03	511.89
160	1000	49.26	3525.049

BLISS:

a) Below are the experimental results for BLISS, after running on a 64bit i7 system.

Sec Param	String Size	Enc/sec	Dec/sec
128	9.6kb	99814.1	20334.12
128	5.6kb	9823.2	19233.12
160	5kb	12965.89	20152.99
196	6kb	11008.10	18922.11

b) Below are the experimental results for BLISS , running on a ARMv7 cortex CPU (700Mhz) with 1 GB of RAM.

Sec Param	String Size	Sign/sec	Ver/sec
128	9.6kb	4234.1	8232.23
128	5.6kb	2123.2	8122.12
160	5kb	4765.89	7212.67
196	6kb	2908.10	7200.15

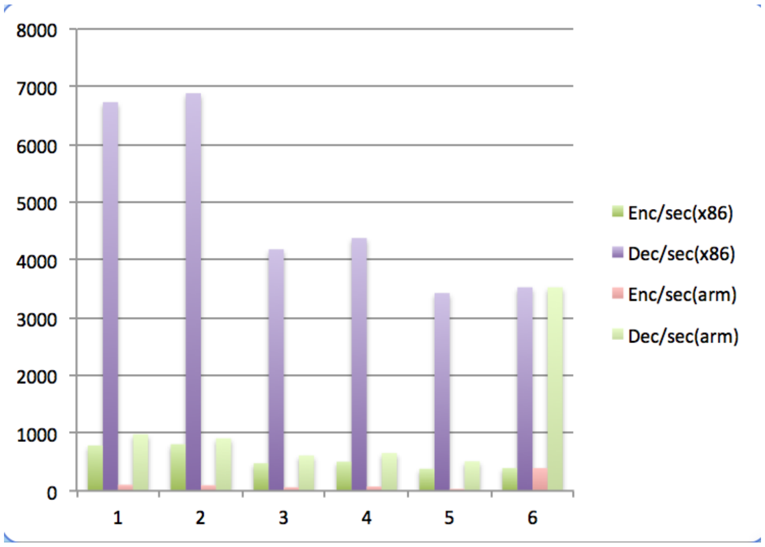

Fig. 2. LP on x86 vs LP on arm V7

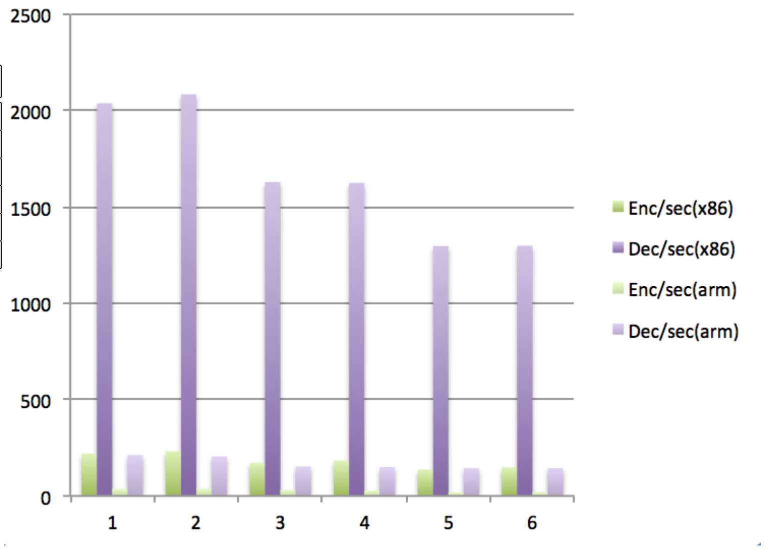

Fig. 3. ULP on x86 vs ULP on arm V7

VI. Conclusions

Over the course of this project, our team has researched on a variety of algorithms available and have analysed which of them are quantum computer safe. We selected a couple of algorithms like NTRU, LP, ULP and BLISS.
We analysed the security proofs of these algorithms and then we moved on to the implementation part of the algorithms. We found a couple of different implementations for all the selected algorithms. We implemented the codes for some of

those algorithms ourselves and then moved on to the analysis phase.

We ran the implementations first on our normal machines, i5 or i7 processors and then implemented them on our constrained device, the Raspberry pi. Based on our analysis for the comparisons with our results for these algorithms with the results for traditional algorithms like RSA and AES, we found that since these algorithms provide security against Quantum Computers, they have great efficiency.
Out of the selected algorithms we found that based on the ease of implementations both in software and hardware and also the performance, we found that BLISS is the most effective algorithm and the best choice among all four selected algorithms. We feel that there can be a lot of more future work in improving the implementation of the BLISS algorithm.

Acknowledgment

The authors would like to thank Aziz Mohaisen, Assistant Professor, Department of Computer Science and Engineering, at The State University of New York at Buffalo, for his help, guidance and motivation to make this experimental research project a success.

References

[1] NTRU- A ring based public key Crypto Jeffery Hoffstein, Jill Pipher, Joseph H. Silverman
https : //www.securityinnovation.com/uploads/Crypto/ANTS97.pdf

[2] Lattice Signatures and Bimodal Gaussians
http : //eprint.iacr.org/2013/383.pdf

[3] A quantum-safe circuit-extension handshake for Tor
https : //eprint.iacr.org/2015/287.pdf

[4] Better Key Sizes (and Attacks) for LWE-Based Encryption. Richard Lindner and Chris Peikert.
http : //web.eecs.umich.edu/ cpeikert/pubs/lwe − analysis.pdf

[5] The Learning with Errors Problem. Oded Regev.
http : //www.cims.nyu.edu/ regev/papers/lwesurvey.pdf

[6] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model.
http : //research.microsoft.com/en − us/um/people/adum/publications/2003 − Noise − $Tolerant_Learning.pdf$

[7] An Improved BKW Algorithm for LWE with Applications to Cryptography and Lattices. Paul Kirchner and Pierre-Alain Fouque.
https : //eprint.iacr.org/2015/552.pdf

[8] Accelerating Bliss: the geometry of ternary polynomials.
https : //eprint.iacr.org/2014/874.pdf

[9] Practical Lattice-based Digital Signature Schemes
http : //csrc.nist.gov/groups/ST/post − quantum − 2015/papers/session9 − oneill − paper.pdf

[10] On Constrained Implementation of Lattice-based Cryptographic Primitives and Schemes on Smart Cards.
https : //eprint.iacr.org/2014/514.pdf

[11] Precomputation Methods for Faster and Greener Post-Quantum Cryptography on Emerging Embedded Platforms
https : //eprint.iacr.org/2015/288.pdf

[12] An Efficient Lattice-Based Signature Scheme with Provably Secure Instantiation
https : //eprint.iacr.org/2016/030.pdf

[13] High-Performance Ideal Lattice-Based Cryptography on 8-bit ATxmega Microcontrollers
https : //pdfs.semanticscholar.org/c6e3/46084b7f06929d67f1dac2ab7a3dde02912b.pdf

[14] Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems *https : //eprint.iacr.org/2015/1132.pdf*

[15] U-LP git repo
https : //github.com/gautamgitspace/ulpcrypt