Analysis of BCNS and NewHope Key-Exchange Protocols

Seyedamirhossein Hesamian
Supervised by Dr. Guangwu Xu

Department of Computer Science
University of Wisconsin Milwaukee

Spring 2017

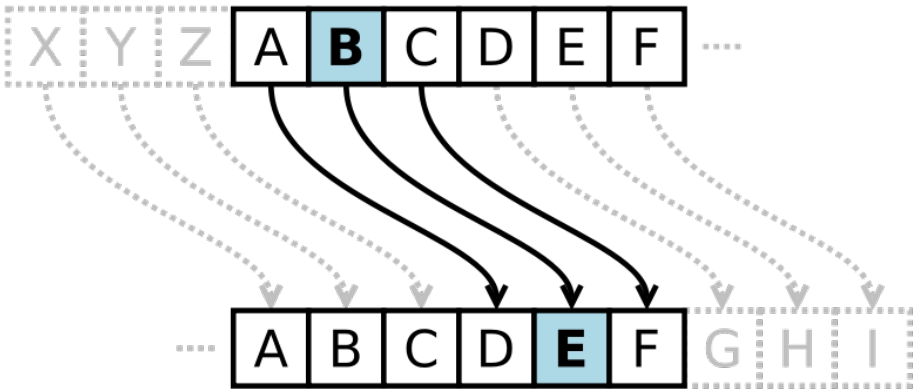
Outline

Symmetric cryptography dates back to 2000 years ago

Caesar cipher ($\approx$ 2000 years ago!)

Encrypt : $x \mapsto x + 3 \bmod 26$

Decrypt : $x \mapsto x - 3 \bmod 26$



$*$ Symmetric because the same key is used for both encryption and decryption.

Importance of key-exchange

Only symmetric cryptography is not enough

Problem with symmetric only cryptography:

- assumes two parties exchanging secret messages share a common secret key
- However, secure distribution of such a key is a major issue
- Asymmetric cryptography two parties that have no prior knowledge of each other to jointly establish a shared key over an insecure channel

Using public-key cryptosystem to replace key-exchange

Two parties should create session key together instead of sharing it

Definition

Session key is a single-use symmetric key used for encrypting all messages in one communication session.

If we simply encrypt random key and encrypt it using server's public key then when server's private key gets leaked then all past communication with that server can be decrypted

This is an unintended consequence.

Importance of forward secrecy

Key-exchange protocol enables forward secrecy

But if an key-exchange method was used, a private key leak would not compromise past communications, since the keys used for the key exchange are never transferred and never saved.

Definition

Forward secrecy is a property in which compromise of long-term keys does not compromise past session keys. Forward secrecy protects past sessions against future compromises of secret keys or passwords.

DDH problem and relation to discrete logarithm

Discrete logarithm can solve decision problem below and it's computational variant

Given cyclic group G with generator g and order q

Definition

Given an element $h \in G$, discrete logarithm problem is to find the smallest positive integer x such that $h = g^x$

DDH: Tuples below are computationally indistinguishable:

- (g^a, g^b, g^{ab}), where a and b are randomly chosen from $\mathbb{Z}_q$
- (g^a, g^b, g^c), where a, b, c are randomly chosen from $\mathbb{Z}_q$

Solve above by computing discrete logarithm of g^a of base g and then computing g^{ab} by exponentiation and then test if $g^{ab} \stackrel{?}{=} g^c$

Diffie-Hellman protocol

Simplest form of key-exchange

If group is a multiplicative group of a finite field then:

Alice		**Bob**
$a \xleftarrow{\$} \mathbb{Z}_p$		$b \xleftarrow{\$} \mathbb{Z}_p$
	$\xrightarrow{g^a \bmod p}$	
	$\xleftarrow{g^b \bmod p}$	
$K_{AB} = (g^b)^a \equiv g^{ab} \bmod p$		$K_{AB} = (g^a)^b \equiv g^{ab} \bmod p$

Figure: Diagram of Diffie-Hellman key-exchange protocol

Note that g is a primitive root mod p and $\xleftarrow{\$}$ means chosen uniformly random

Shor's algorithm

Shor's algorithm factors integers in polynomial time

Shor's algorithm [Shor, 1994], named after mathematician Peter Shor, is a quantum algorithm (an algorithm that runs on a quantum computer) for integer factorization. Informally it solves the following problem:

- given an integer, find its prime factors in polynomial time $\approx \mathcal{O}(n^2)$ where n is number of bits of input
- Substantially faster than the most efficient known classical factoring algorithm, the general number field sieve, that factors large integers (e.g. more than 140 digits) which works in sub-exponential time $e^{(1.923+\mathcal{O}(1))(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}}}$

Shor's algorithm

Shor's algorithm beaks Diffie-Hellman key-exchange protocol

Shor's algorithm was initially designed to factor integers but later it was shown that it can be modified to solve discrete logarithm problem in polynomial time [Boneh and Lipton, 1995]
Therefore,

- RSA $\rightarrow$ **broken!**
  - security based on factorization
- Diffie-Hellman $\rightarrow$ **broken!**
  - security based on discrete logarithm

Lattice-based cryptography

Alternative approach to find a quantum safe cryptosystem

Lattice-based cryptography is a *quantum safe* alternative.

- Unlike more widely used and known public key cryptography which are attacked by a quantum computer, some lattice-based cryptosystems appear to be resistant to attack by both classical and quantum computers
- Learning with Errors (LWE) variants of lattice-based cryptography comes with security proofs which demonstrate that breaking the cryptography is equivalent to solving known hard problems in lattices

Worst-case hardness vs. average-case hardness

Lattices problems are hard in worst case but factorization is hard on average case

If we solve 1% of lattice based cryptographic function, then we can solve all instances of lattice problems.

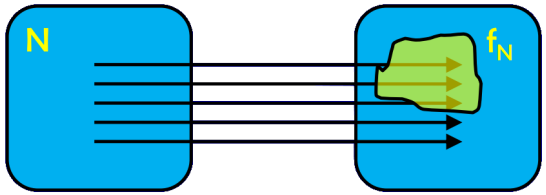


Figure: Average-case problem (e.g. factorization)

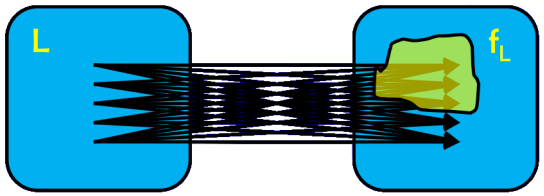


Figure: Worst-case problem (e.g. lattice problems)

Lattice-based key-exchange

Diffie-Hellman like key-exchange protocol using (Ring-)LWE

Intention is to create an alternative Diffie-Hellman like key-exchange protocol.

- (Ring-)LWE key-exchange is similar in nature to Diffie-Hellman (i.e. having a public and private components to derive a shared key)
- **But** the underlying mathematics behind Diffie-Hellman and (Ring-)LWE have no connection with each other.

Lattice definition

Basis is a set of linearly independent vectors, their linear combination generates lattice

Given set of n-linearly independent vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ known as a basis of the lattice, all linear combinations of the basis vectors forms a lattice.

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\}$$

Linear independence means that no vector can be written as linear combination of the other vectors.

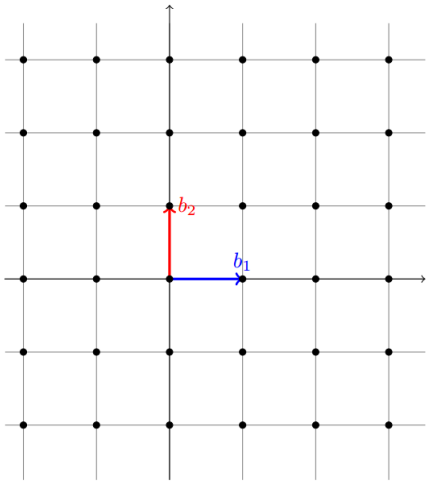
Examples of lattices

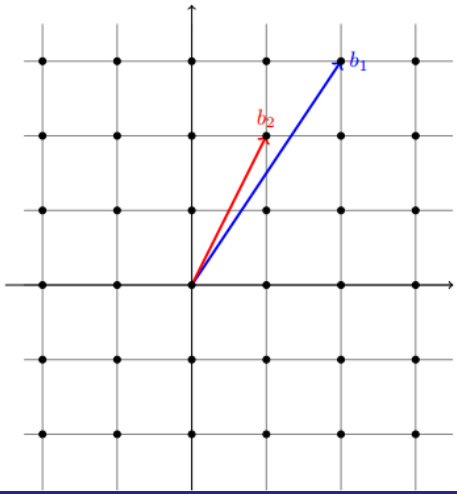
Different lattice bases can generate the same lattice

* Lattice can be generated from infinitely many bases

$\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B'})$ because $\mathbf{B} = \mathbf{B'} \times \mathbf{U}$ where $|\det(\mathbf{U})| = 1$

- Square integer matrix **U** is called a unimodular matrix
- Below is an "Integer lattice" or $\mathbb{Z}^2$





Lattice properties

The smaller lattice determinant, the denser the lattice

- $\det(\mathcal{L}_{\mathbf{B}})$ is the n-dimensional volume of the fundamental parallelepiped defined by the lattice basis $\mathbf{B}$
- smaller the $\det(\mathcal{L}_{\mathbf{B}})$, the denser the lattice

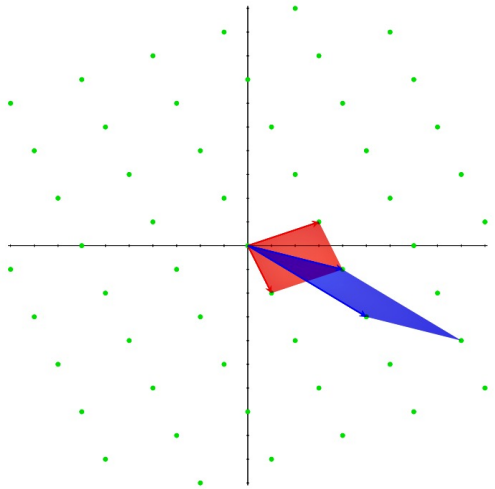


Figure: Two fundamental parallelepiped of the same lattice

Lattice properties

Shortest lattice vector λ_1 and successive minima λ_i

- Length of the shortest nonzero vector in the lattice or λ_1. It is the smallest r such that the lattice points inside a ball of radius r span a space of dimension 1.

This leads to the generalization of λ_1, known as successive minima.
$*$ We have to ask for a nonzero vector since the zero vector is always contained in a lattice and its norm is zero.

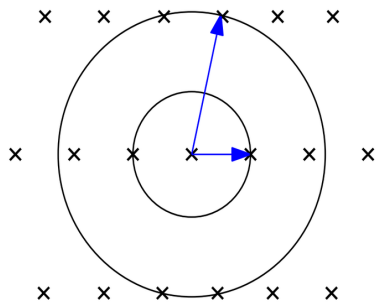


Figure: Visualization of first and second lattice minima

Lattice reduction

Goals is to find short, not necessarily shortest vectors

The goal of lattice basis reduction is given an integer lattice basis as input, to find a basis with short, nearly orthogonal vectors

- Lenstra-Lenstra-Lovsz (LLL) lattice basis reduction algorithm is a polynomial time lattice reduction invented by A. Lenstra, H. Lenstra and L. Lovsz in 1982 [Lenstra et al., 1982]

Given basis of a lattice $\mathcal{L}$, the LLL algorithm calculates an LLL-reduced lattice basis in polynomial time (for $\delta \in (0.25, 1)$)

$||b|| \leq 2^{\frac{n-1}{2}} \lambda_1$ (LLL reduced basis is far from shortest at n gets larger)

Example of lattice reduction and shortest vector in lattice

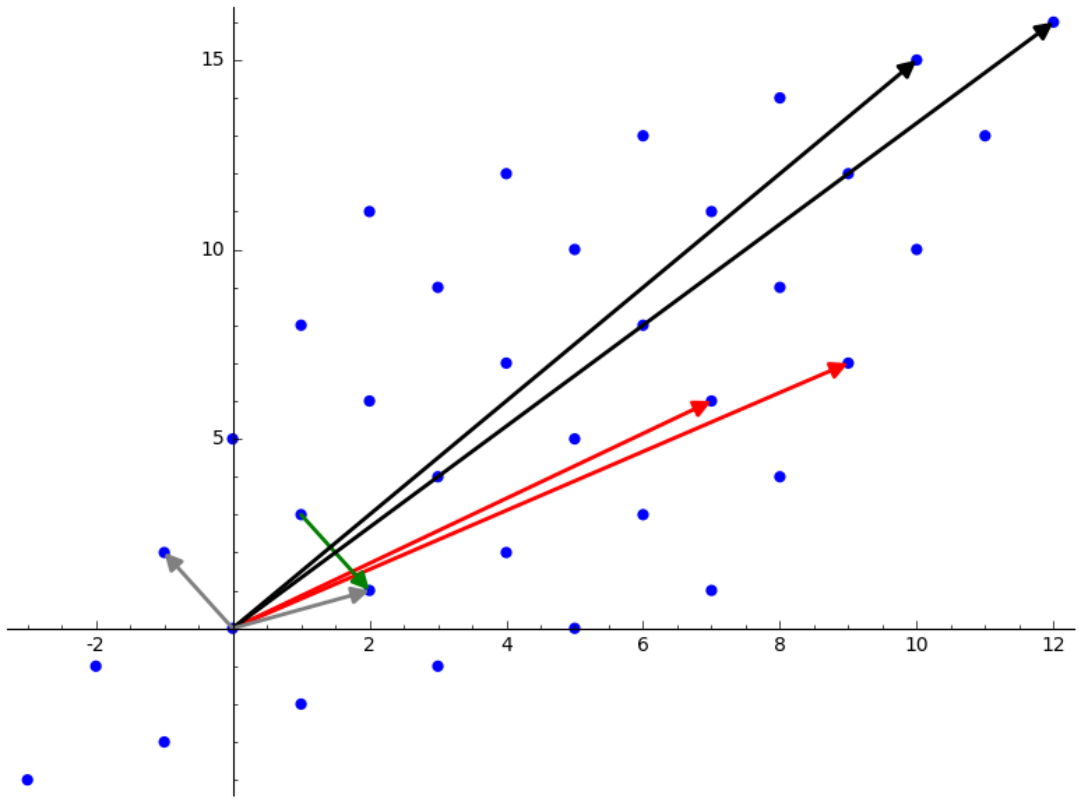


Figure: Grey vectors are LLL-reduced basis and green vector is a shortest vector in lattice

Ideal lattices

Representing lattices as polynomials instead of matrices

Definition

Let I be an ideal of $\mathbb{Z}[x]/(f(x))$ where $\deg(f) = n$ and $\mathcal{L}_I = \{(a_0, a_1, \ldots, a_{n-1}) | a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in I\}$. We call $\mathcal{L}_I$ an ideal lattice.

* coefficients of polynomial represent lattice vector
* polynomial f should be irreducible

Theorem

Let $\mathcal{L}$ be a lattice that corresponds to an ideal in the ring $\mathbb{Z}[x]/(x^n + 1)$ and let $\mathbf{u} \in \mathcal{L}$ be a vector in lattice. Then the vectors $\mathbf{u}, x\mathbf{u}, x^2\mathbf{u}, \ldots, x^{n-1}\mathbf{u}$ are linearly independent

Ideal lattices

Re-create lattice basis matrix from quotient polynomial ring

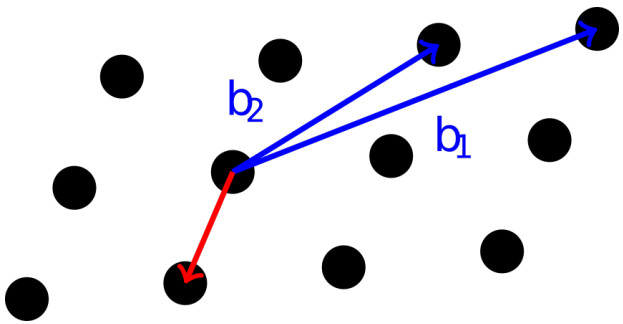
Simple irreducible polynomial can be: $x^n + 1$ when n is a power of 2. We can use a wrapping rule to represent a polynomial in $\mathbb{Z}[x]/(x^n + 1)$. The wrapping rule can be: $x \mapsto -x \bmod 13$ applied to the shifted value.
For example: $\begin{bmatrix} 4 & 1 & 11 & 10 \end{bmatrix}$ and $\mathbb{Z}_{13}[x]/(x^4 + 1)$. Using first row is enough to re-create the matrix and after $2 \times 4 = 8$ rows, rows are repeating.

$$\begin{bmatrix} 4 & 1 & 11 & 10 \\ 3 & 4 & 1 & 11 \\ 2 & 3 & 4 & 1 \\ 12 & 2 & 3 & 4 \\ 9 & 12 & 2 & 3 \\ 10 & 9 & 12 & 2 \\ 11 & 10 & 9 & 12 \\ 1 & 11 & 10 & 9 \end{bmatrix} \leftrightarrow \begin{array}{c} x^0 \times (4x^0 + 1x^1 + 11x^2 + 10x^3) = 4x^0 + 1x^1 + 11x^2 + 10x^3 \rightarrow [4, 1, 11, 10] \\ x^1 \times (4x^0 + 1x^1 + 11x^2 + 10x^3) = 3x^0 + 4x^1 + 1x^2 + 11x^3 \rightarrow [3, 4, 1, 11] \\ x^2 \times (4x^0 + 1x^1 + 11x^2 + 10x^3) = 2x^0 + 3x^1 + 4x^2 + 1x^3 \rightarrow [2, 3, 4, 1] \\ x^3 \times (4x^0 + 1x^1 + 11x^2 + 10x^3) = 12x^0 + 2x^1 + 3x^2 + 4x^3 \rightarrow [12, 2, 3, 4] \\ \cdots \\ x^7 \times (4x^0 + 1x^1 + 11x^2 + 10x^3) = 1x^0 + 11x^1 + 10x^2 + 9x^3 \rightarrow [1, 11, 10, 9] \end{array}$$

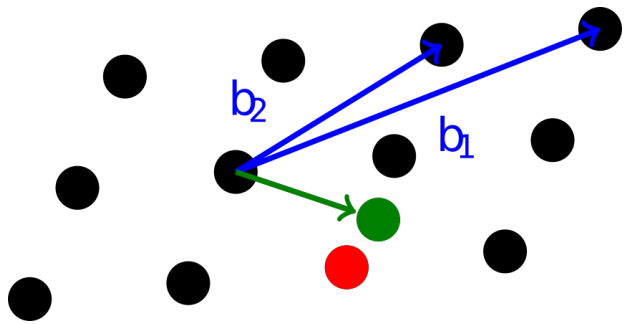
Lattice problems

Two main problems in lattices are: SVP and CVP

- Shortest Vector Problem (SVP): find a shortest non-zero vector in $\mathcal{L}$.
- Closest Vector Problem (CVP): given a vector $t \in \mathbb{R}^n$ not in $\mathcal{L}$, find a vector in $\mathcal{L}$ that is closest to t.



(a) SVP



(b) CVP

Lattice problems (approximate variant)

Create approximate variant of lattice problems thanks to Minkowski theorem

Theorem

(Minkowski): Let $\mathcal{L}$ be lattice of $\mathbb{R}^n$ with rank n, the length of shortest lattice vector of $\mathcal{L}$, $\lambda_1 \leq \sqrt{n} \times \det(\mathcal{L})^{\frac{1}{n}}$

Minkowski theorem gives us an upper-bound to λ_1

Definition

Approximate Shortest Vector Problem (SVP_γ): given a lattice $\mathcal{L}$, task is to find a non-zero vector $\mathbf{v} \in \mathcal{L}$, such that $||\mathbf{v}|| \leq \gamma\lambda_1(\mathcal{L})$.

Ajtai's one-way function

Introduced by Ajtai in 1996

Ajtai's one-way hash function ($m \geq n \log q$ to get compression):

- parameters: $m, n, q \in \mathbb{Z}$
- key: $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$
- input: $\mathbf{x} \in \{0,1\}^m$
- output: $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q$

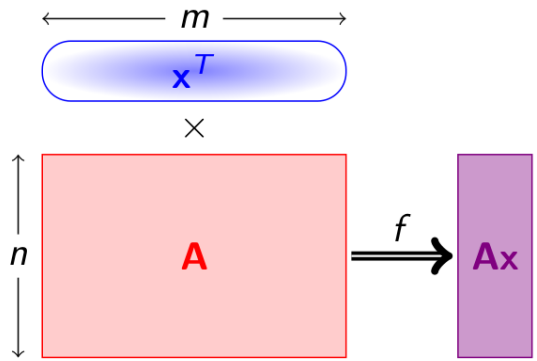


Figure: Visualization of Ajtai's one-way function

Specifize Ajtai's one-way function to SIS problem

SIS is a average-case problem

Definition

$\text{SIS}_{n,m,q,\beta}$: let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be an $n \times m$ matrix with entries in $\mathbb{Z}_q$ that consists of m uniformly random vectors $\mathbf{a}_i \in \mathbb{Z}_q^n : \mathbf{A} = (\mathbf{a}_1, \cdots, \mathbf{a}_m)$. Find a nonzero vector $\mathbf{x} \in \mathbb{Z}^m$ such that:

- $\|\mathbf{x}\| \leq \beta$
- $f_{\mathbf{A}}(\mathbf{x}) : \mathbf{A}\mathbf{x} = 0 \in \mathbb{Z}_q^n$

We require $\beta < q$, otherwise $x = (q, 0, \ldots, 0) \in \mathbb{Z}^m$ is a trivial solution.

SIS problem as hash function

SIS forms a collision resistant hash function

SIS problem can also form a collision-Resistant Hash function:
Given: $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_m) \in \mathbb{Z}_q^n$, define $\text{Hash}_{\mathbf{A}} : \{0,1\}^m \mapsto \mathbb{Z}_q^n$ where $\text{Hash}_{\mathbf{A}}(z_1, \ldots, z_m) = a_1 z_1 + \cdots + a_m z_m$. Set $m > n \log q$ to get compression. Collision would occur when:

$$\mathbf{a}_1 z_1 + \cdots + \mathbf{a}_m z_m = \mathbf{a}_1 y_1 + \cdots + \mathbf{a}_m y_m$$

By adjusting the above we get:
$\mathbf{a}_1(z_1 - y_1) + \cdots + \mathbf{a}_m(z_m - y_m) = 0$ and $z_i - y_i$ are in $\{-1, 0, 1\}$.
Trivial solution is not a valid solution to SIS problem, further because the only way we could find a collision is when $z_i - y_i = 0 \bmod q$. SIS is collision resistant

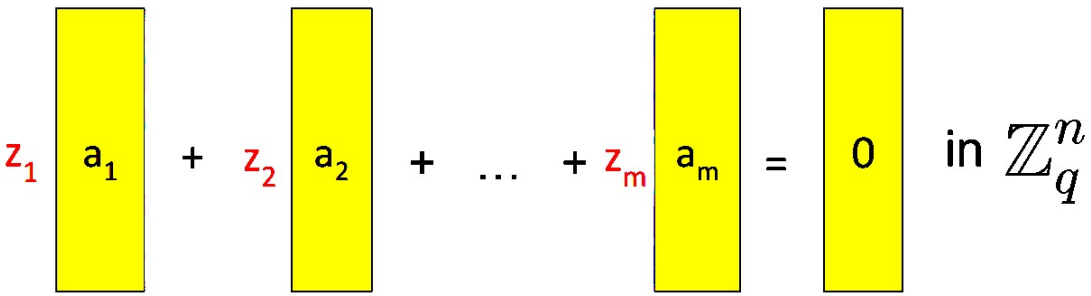
Relation between SIS and SVP

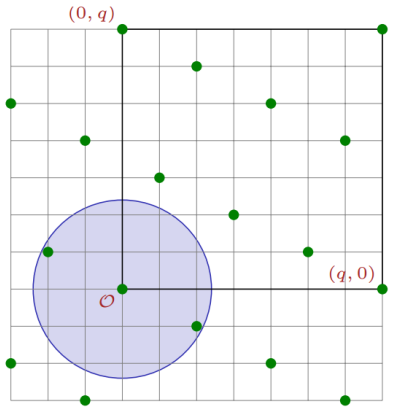
SIS and SVP are very similar!

Ajtai showed that SIS is secure in average-case if SVP is hard in worse-case.

$$\text{(q-ary lattice) } \mathcal{L}_q^{\perp}(\mathbf{A}) = \{\mathbf{e} \in \mathbb{Z}^m \mid \mathbf{Ae} \equiv \mathbf{0} \bmod q\}$$

SIS is SVP on "q-ary" lattices $\mathcal{L}_q^{\perp}$. (right) looking for smallest radius to contain enough lattice points to reconstruct the lattice.





Learning with errors problem

A, b are public but s, e are kept private

LWE problem is to efficiently distinguish vectors created from a "noisy" set of linear equations and uniformly random vectors. Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{Z}_q^m$, the goal is to determine whether $\mathbf{b}$ has been sampled uniformly at random from $\mathbb{Z}_q^m$ or whether $\mathbf{b} = \mathbf{As} + \mathbf{e}$ for some random $\mathbf{s} \in \mathbb{Z}_q^m$ and $\mathbf{e} \in \chi_m$, where χ is a small "noise" distribution over $\mathbb{Z}_q$.

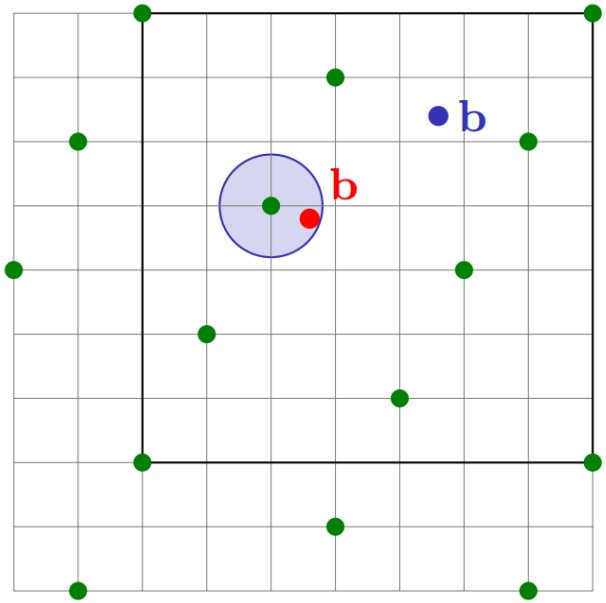
$$
\begin{aligned}
14s_1 + 15s_2 + 5s_3 + 2s_4 &\approx 8 \bmod 17 \\
13s_1 + 14s_2 + 14s_3 + 6s_4 &\approx 16 \bmod 17 \\
6s_1 + 10s_2 + 13s_3 + 1s_4 &\approx 3 \bmod 17 \\
10s_1 + 4s_2 + 12s_3 + 16s_4 &\approx 12 \bmod 17 \\
9s_1 + 5s_2 + 9s_3 + 6s_4 &\approx 9 \bmod 17 \\
3s_1 + 6s_2 + 4s_3 + 5s_4 &\approx 16 \bmod 17 \\
&\dots \\
6s_1 + 7s_2 + 16s_3 + 2s_4 &\approx 3 \bmod 17
\end{aligned}
$$

LWE properties

Validate potential solution and random self-reduction

Visualization of LWE reduced to Bounded distance decoding (BDD) problem which is a variant of CVP problem

$$\mathbf{b}^t \text{ (in red)} = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \text{ vs. } \mathbf{b} \leftarrow \mathbb{Z}_q^m$$



LWE properties

Validate potential solution and random self-reduction property

Simple properties of LWE:

- Check a candidate solution $\mathbf{s}' \in \mathbb{Z}_q^n$, we test if $\mathbf{b} - \langle \mathbf{s}', \mathbf{a} \rangle$ is small. If $\mathbf{s}' \neq \mathbf{s}$, then $\mathbf{b} - \langle \mathbf{s}', \mathbf{a} \rangle = \langle \mathbf{s} - \mathbf{s}', \mathbf{a} \rangle + \mathbf{e}$ is well spread in $\mathbb{Z}_q$
- Shift the secret by any $\mathbf{t} \in \mathbb{Z}_q^n$ given $(\mathbf{a}, \mathbf{b} = \langle \mathbf{s}, \mathbf{a} \rangle + \mathbf{e})$ output:

$$(\mathbf{a}, \mathbf{b}' = \mathbf{b} + \langle \mathbf{t}, \mathbf{a} \rangle = \langle \mathbf{s} + \mathbf{t}, \mathbf{a} \rangle + \mathbf{e})$$

LWE properties

SIS solver can solve LWE; solving LWE comes down to solving SIS problem

Difference / relation between SIS and LWE:

- If we have a SIS oracle, then we can ask the oracle to find short vector $\mathbf{z}$ such that $\mathbf{Az} \equiv 0 \bmod q$ and as $\mathbf{b} = \mathbf{As} + \mathbf{e}$, then $\mathbf{bz} = (\mathbf{As} + \mathbf{e}) \cdot \mathbf{z} = 0 + \mathbf{ez} = \mathbf{ez}$. Note that $\mathbf{ez}$ is small because $\mathbf{z}$ is short but $\mathbf{bz}$ is well spread. Therefore, we just solved LWE

Ring variant of LWE problem

Quotient polynomial ring over finite field instead of matrices

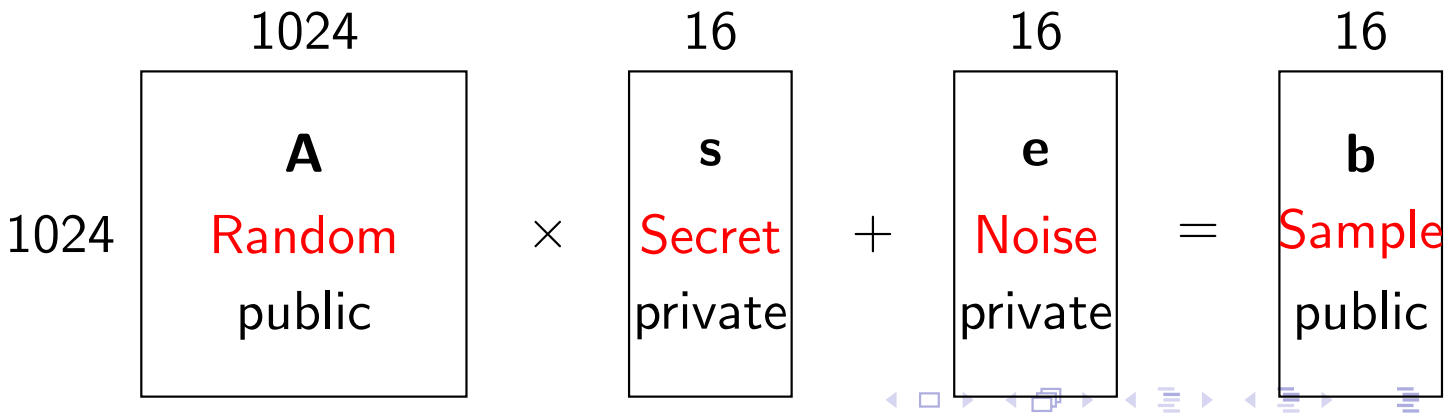
Adding more structure by considering ideal lattices instead of random ones, we obtain the Ring-LWE problem, also called the RLWE problem. "search" version and a "decision" version both begin with the same construction.

- $a_i(x)$ be a random element but known polynomials from $\mathbb{Z}_q[x]/(\Phi(x))$ with coefficients from all of $\mathbb{Z}_q$
- $e_i(x)$ be small random element and unknown polynomials relative to a bound b in the ring $\mathbb{Z}_q[x]/(\Phi(x))$
- $s(x)$ be a small unknown polynomial relative to a bound b in the ring $\mathbb{Z}_q[x]/(\Phi(x))$.
- $b_i(x) = (a_i(x) \cdot s(x)) + e_i(x)$

Basics of LWE as key-exchange

Multiplying our secret with LWE samples of other party and use it as a shared key

Alice		**Bob**
$\mathbf{e} \leftarrow \chi$, $\mathbf{A}, \mathbf{s} \leftarrow$ uniformly random		$\mathbf{e'} \leftarrow \chi$, $\mathbf{s'} \leftarrow$ uniformly random
	$\xrightarrow{\mathbf{A}, \mathbf{b}=\mathbf{As}+\mathbf{e}}$	
	$\xleftarrow{\mathbf{b'}=\mathbf{As'}+\mathbf{e'}}$	
$Key = \mathbf{s} \times \mathbf{b'} = \mathbf{Ass'} + \mathbf{se'}$		$Key = \mathbf{s'} \times \mathbf{b} = \mathbf{Ass'} + \mathbf{s'e}$



Issue with using LWE instead of ring variant of LWE

Achieving cumulative property by shuffling

Matrix multiplication is not commutative, we need to shuffle and transpose key and noise to achieve cumulative property. In the following example we see the procedure to achieve 16×16 shared key (i.e. matrix).

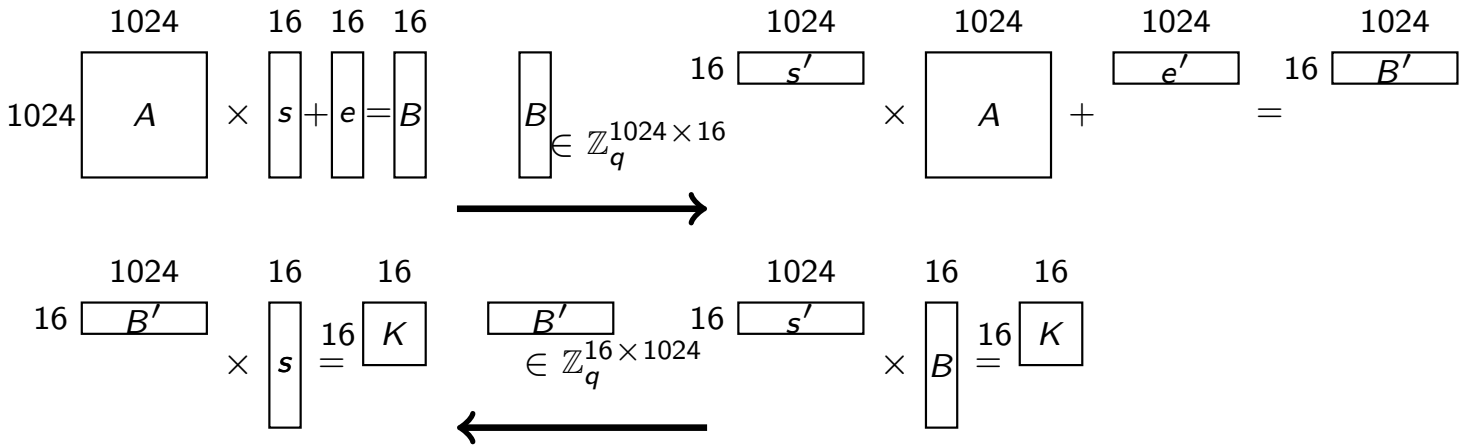


Figure: Visualization of LWE based key-exchange

Need for reconciliation

Notice that calculated keys are close but not equal.

Using polynomials instead of matrix would address the high cost of matrix multiplication and inversion. So, we use Ring variant of LWE, known as Ring-LWE in key-exchange.
However, shared keys do not match because $s \times e' \neq s' \times e$.
Although errors are supposed be relatively small compared to shared polynomial A and their product with secret is small (i.e. $||s \times e'||, ||s' \times e|| \ll ||A||$), but still resulting shared key do not match and it is a problem!

Shared secret Alice calculates: $s \times b' = s \times (A \times s' + e') = A \times s \times s' + \boldsymbol{s \times e'}$
Shared secret Bob calculates: $s' \times b = s' \times (A \times s + e) = A \times s \times s' + \boldsymbol{s' \times e}$

Sampling secret from noise is beneficial for reconciliation

If secret is small then it's product with error would also be small

Lemma

([Applebaum et al., 2009], Lemma 2) LWE is no easier if the secret is drawn from the error distribution

The advantage to sample s from noise distribution is that it's product with error would be small. Because in key-exchange we receive as a shared key:

$$s \times (As' + e') = Ass' + se' \leftrightarrow s' \times (As + e) = Ass' + s'e$$

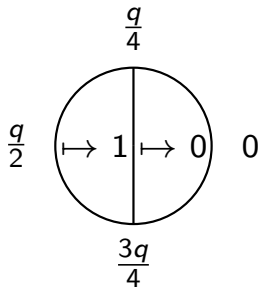
So if both parties choose small s and s' hence their inner product would be small then chance of reconciliation fail to extract the same key would be reduced substantially

Regev's basic rounding approach

Simple rounding approach that works surprising well

As each coefficient of polynomial is an integer modulo q, round either to 1 when coefficient is between $(\frac{q}{4}, \frac{3q}{4})$ and round to 0 when coefficient is between $(\frac{q}{4}, \frac{-q}{4})$.

$*$ probability of failure is $\frac{1}{1024}$ which is not good enough considering that we have 1024 coefficients in our shared polynomial to achieve a reasonable security.



$$4079331x^0 + 1894732x^1 + \cdots + 472608x^{1022} + 516748x^{1023} = 01\ldots00$$

$$4079332x^0 + 1894733x^1 + \cdots + 472607x^{1022} + 516748x^{1023} = 01\ldots00$$

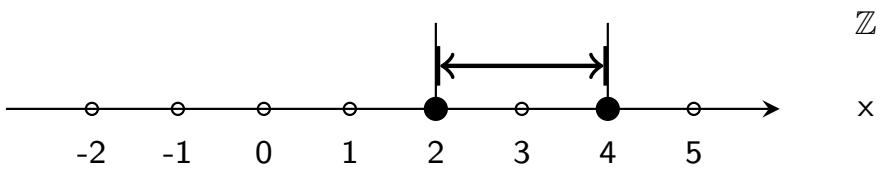
Key idea behind Ding's method

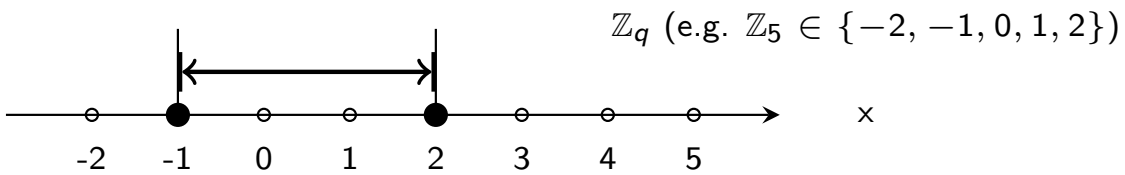
More than just multiplying error terms by 2 is needed.

In $\mathbb{Z}$ difference between two points is even but in $\mathbb{Z}_q$ (e.g. $\mathbb{Z}_5$) difference is odd, hence no benefit in just multiplying error by 2

Alice $\rightarrow$ Bob: $b = A \times s + 2 \times e$
Bob $\rightarrow$ Alice: $b' = A \times s' + 2 \times e'$
Shared secret Alice calculates: $s \times b' = s \times (A \times s' + 2 \times e') = A \times s \times s' + 2 \times s \times e'$

Shared secret Bob calculates: $s' \times b = s' \times (A \times s + 2 \times e) = A \times s \times s' + 2 \times s' \times e$

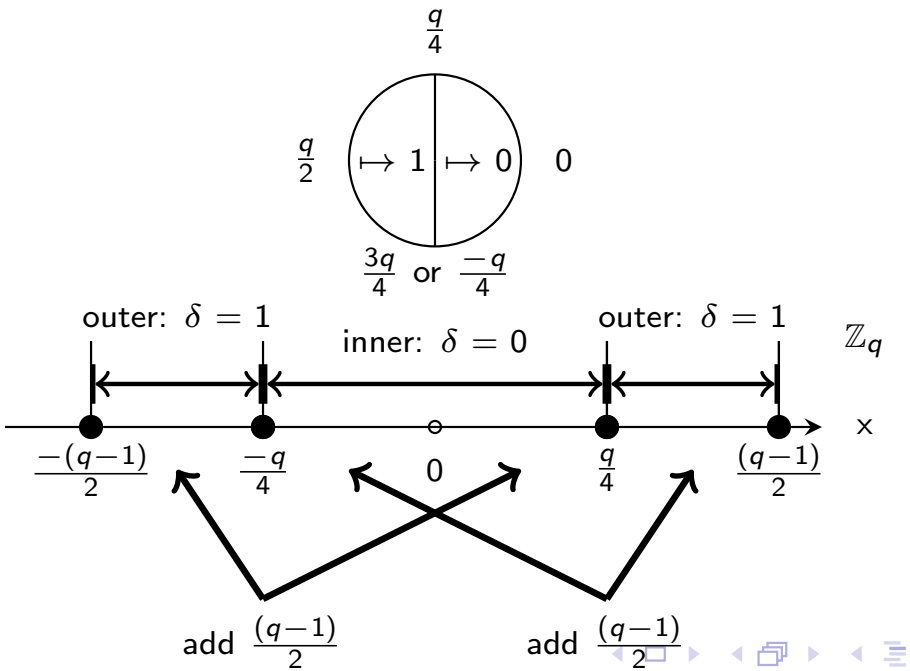




Overview of Ding's reconciliation method

Trying to prevent the case when difference between two points is odd

If $\delta = 1$ or we are in outer region, then we add $\frac{(q-1)}{2}$ to respective coefficient, hence we will be in inner region and then mod2; if $\delta = 0$ then we just mod2.



Concrete definition of Ding's method

Straightforward procedure to reconcile

To describe Ding's reconciliation method, we define the following functions: $\delta : \mathbb{Z}_q \mapsto \{0,1\}$, $Signal : \mathbb{Z}_q \mapsto \{0,1\}$, and $Encode : \mathbb{Z}_q \times \{0,1\} \mapsto \{0,1\}$ via:

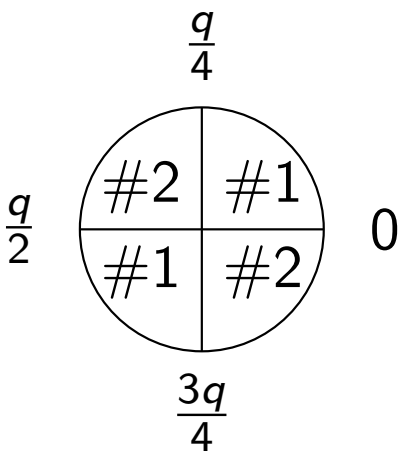
$$Signal(x) = \delta(x) \mapsto \begin{cases} 0 & \text{if } x \in [-\lfloor \frac{q}{4} \rfloor, \lfloor \frac{q}{4} \rceil] \\ 1 & \text{otherwise} \end{cases}$$

$$Encode(x, \delta) = (x + \delta \times (\frac{q-1}{2}) \bmod q) (\bmod 2)$$

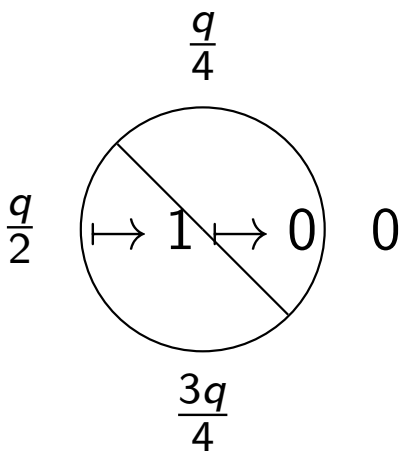
Peikert's improved rounding method

Simpler than Ding's method

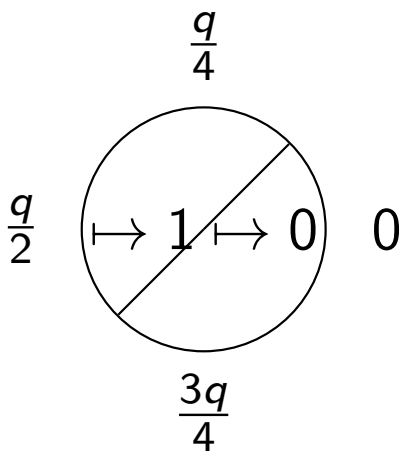
Notice if difference of coefficients is $\leq \frac{q}{8}$ then this method always works.



(a) 4 regions



(b) case #1



(c) case #2

Calculated key using Ding's method has bias

Peikert's method removes the bias using randomized doubling function

Any deterministic map from $\mathbb{Z}_q$ to $0, 1$ must be biased when q is odd. To address the issue Peikert suggested, if the modulus q is odd (which it is in Ring-LWE key-exchange), it requires to work in $\mathbb{Z}_{2q}$ instead of $\mathbb{Z}_q$ to avoid bias in the derived bits.
* randomized doubling function as suggested by Peikert.

$$\text{dbl} : \mathbb{Z}_q \to \mathbb{Z}_{2q}, x \mapsto 2x + e, \text{ where } e = \begin{cases} -1 & \text{with probability } \frac{1}{4} \\ 0 & \text{with probability } \frac{1}{2} \\ 1 & \text{with probability } \frac{1}{4} \end{cases}$$

Visualization of Alkim reconciliation method

Also known as NewHope reconciliation method

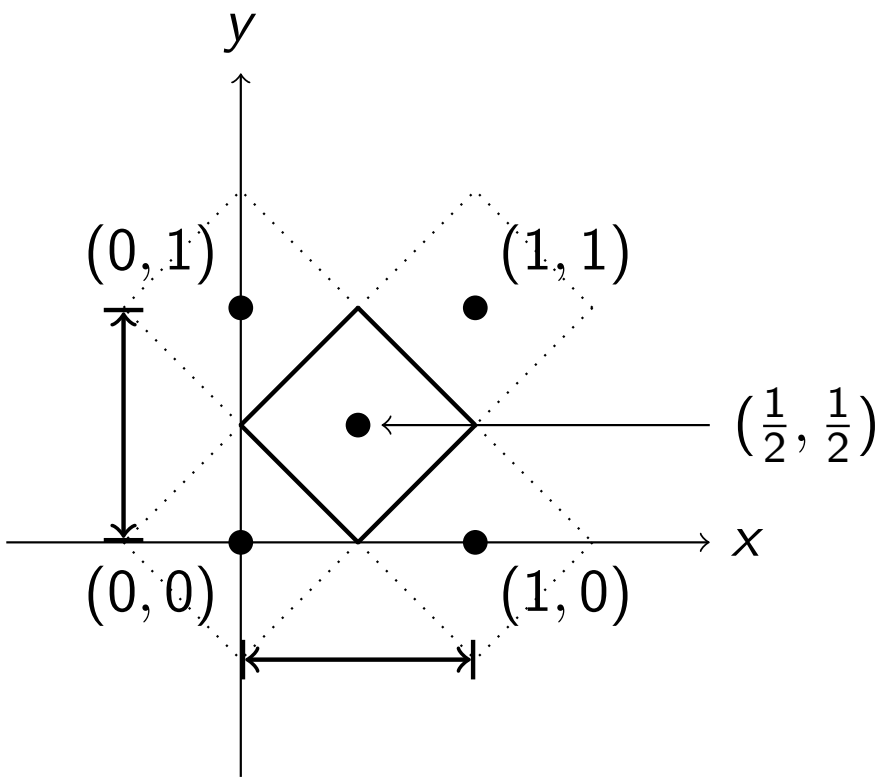


Figure: 2-dimension Voronoi cell centered at $(\frac{1}{2}, \frac{1}{2})$

Valid Voronoi cell

Volume of polyhedron should be $\frac{1}{2}$ to achieve uniform key

Given: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$

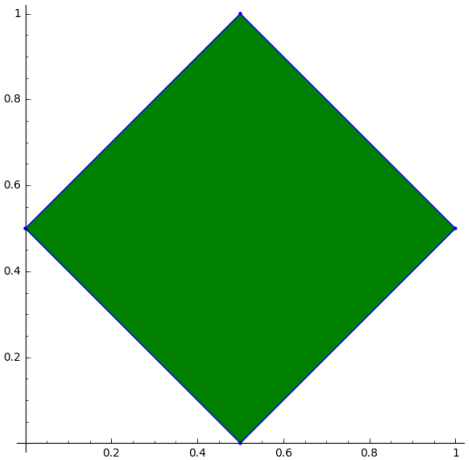
Remark

The valid Voronoi cell (or polyhedron generated by Diamond cutting algorithm) should have a volume (or area in 2-dimensions) of $\frac{1}{2}$ [Viterbo and Biglieri, 1996].

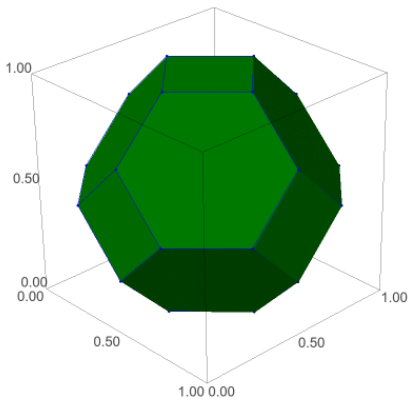
Above implies that probability of $0, 1$ bit $= \frac{1}{2}$. Hence, there is no bias in generated key and it is uniformly distributed.

Polyhedron generated by diamond cutting algorithm

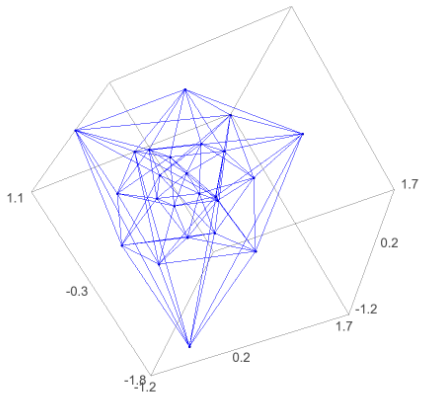
Visualization of result of diamond cutting algorithm



(a) 2 dimension



(b) 3 dimension



(c) 4 dimension

Splitting Voronoi cell into sub-cells

Just send the sub-cell number instead of difference

Sending array of floating point numbers is not efficient. Solve issue by splitting each Voronoi cells into sub-cells. Then find the closest center of Voronoi sub-cell and send the sub-cell number to other party as a reconciliation information.

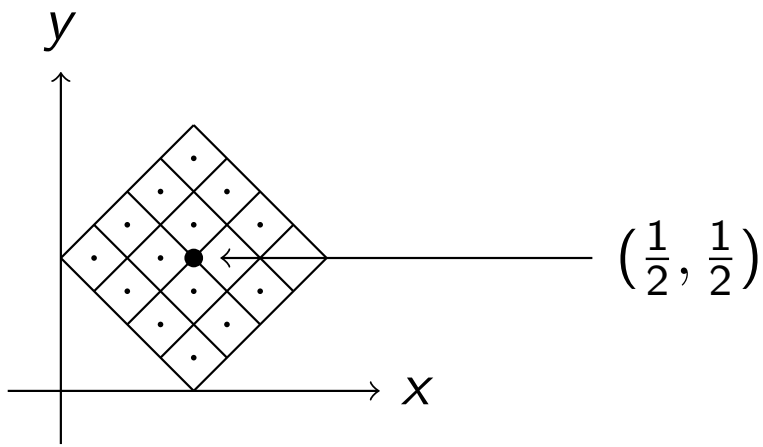


Figure: 2-dimension Voronoi cell centered at $(\frac{1}{2}, \frac{1}{2})$ split into 16 sub-cells

Generalized form of randomized doubly function

Adding $\frac{1}{2q}$ with probability $\frac{1}{2}$

Probability of **x** being in a center Voronoi cell is the same as for **x** being in other Voronoi cells. This would be the case if **x** actually followed a continuous uniform distribution.

- **However**, coefficients of **x** are discrete values in $\{0, \frac{1}{q}, \ldots, \frac{q-1}{q}\}$ and with the protocol described so far, the bits of **v** would have a small bias.
- **Solution** is to add to **x** with probability $\frac{1}{2}$ the vector $(\frac{1}{2q}, \ldots, \frac{1}{2q})$ before running error reconciliation.
- This has close to no effect for most values of **x**, but, with probability $\frac{1}{2}$ moves **x** to another Voronoi cell if it is very close to one side of a border.

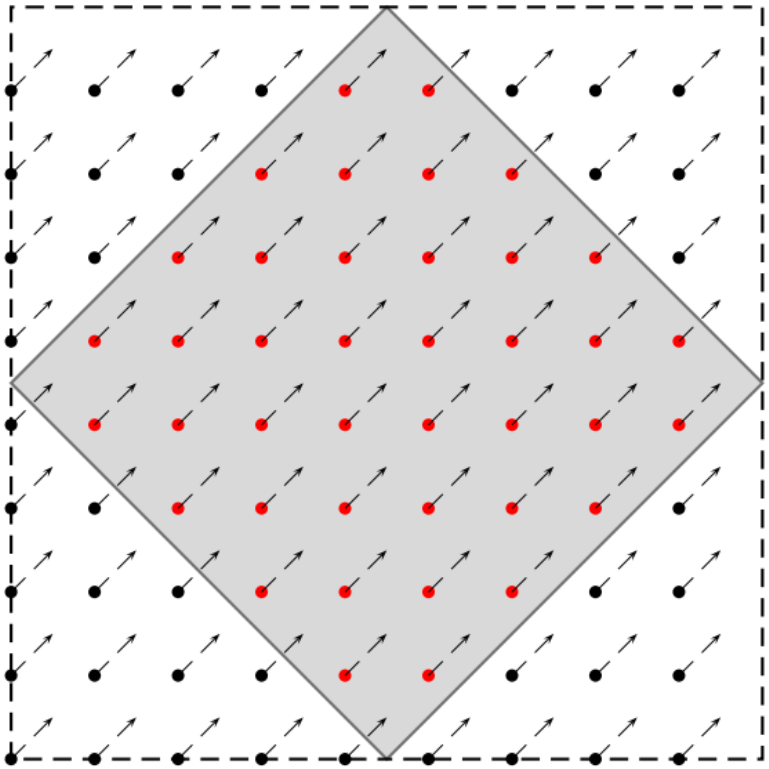


Figure: Effect of generalized form of randomized doubling function on vectors

BCNS key-exchange protocol diagram

$\langle . \rangle$ is cross-rounding function

$a \leftarrow$ Uniformly random		
Alice		**Bob**
$s, e \xleftarrow{\$} \chi$		$s', e' \xleftarrow{\$} \chi$
$b \leftarrow as + e$	$\xrightarrow{a,b}$	$b' \leftarrow as' + e'$
		$e'' \xleftarrow{\$} \chi$
		$v \leftarrow bs' + e''$
	$\xleftarrow{b',c}$	$\bar{v} \xleftarrow{\$} \mathrm{dbl}(v)$
		$c \leftarrow \langle \bar{v} \rangle_{2q,2} \in \{0,1\}^n$
$k_A \leftarrow \mathrm{rec}(2b's, c) \in \{0,1\}^n$		$k_B \leftarrow \mathrm{rec}(2bs', c) \in \{0,1\}^n$

NewHope key-exchange protocol diagram

Ψ_{16} is a 16 bit binomial sampler

$q = 12289 < 2^{14}, n = 1024, \Psi_{16}$		
Alice		**Bob**
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$a \leftarrow \text{parse}(\text{SHAKE-128}(seed))$		
$s, e \xleftarrow{\$} \Psi_{16}^n$		$s', e, e'' \xleftarrow{\$} \Psi_{16}^n$
$b \leftarrow as + e$	$\xrightarrow{(b, seed)}$	$a \leftarrow \text{parse}(\text{SHAKE-128}(seed))$
		$u \leftarrow as' + e'$
		$v \leftarrow bs' + e''$
$v' \leftarrow us$	$\xleftarrow{(u, r)}$	$r \xleftarrow{\$} \text{HelpRec}(v)$
$v \leftarrow \text{Rec}(v', r)$		$v \leftarrow \text{Rec}(v, r)$
$\mu \leftarrow \text{SHA3-256}(v)$		$\mu \leftarrow \text{SHA3-256}(v)$

Parameter choices for Ring-LWE key-exchange

$x^{1024}+1$, Gaussian distribution $\sigma = 8/\sqrt{2\pi}$

Singh in [Singh, 2015] suggested two sets of parameters;
$\sigma = \frac{8}{\sqrt{2\pi}} = 3.192$: $n = 1024$ $q = 40961$ $\Phi(x) = x^{1024}+1$

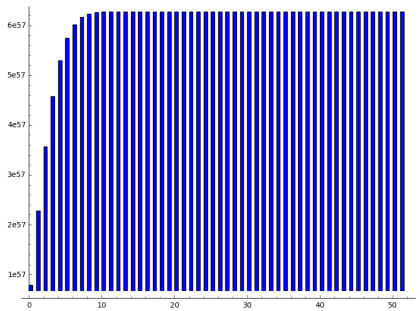
* BCNS protocol kept the σ and Φ as suggested by Singh, but changed modulus of finite field to $2^{32}-1$. This resulted in failure probability of $2^{-131072}$
* NewHope kept the Φ but changed modulus of finite field to $12289 < 2^{14}$ and used binomial distribution sampler instead resulted in failure probability of 2^{-60}

Remark

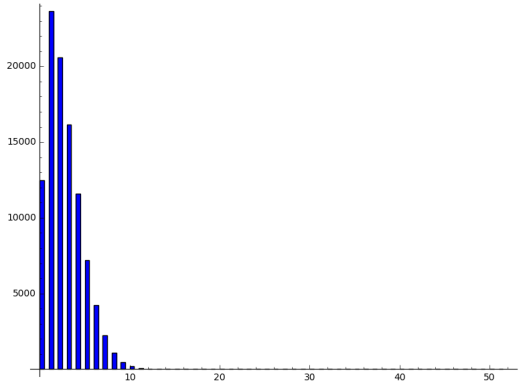
The smaller q results in faster key-exchange (by reducing over-head size) and increase in security, but yields higher failure probability.

Gaussian sampler as suggested in BCNS

BCNS protocol uses inversion sampling with pre-computed look-up table of CDF. To samples, independently generate a 192-bit integer s uniformly at random, and compute unique smallest integer index $\in [0, 50]$ such that $s <$ table[*index*] and one additional random bit to decide the sign.



(a) CDF table



(b) Samples from pre-computed CDF table

Binomial distribution sampler suggested in NewHope

Difference of two independently uniformly sampled random variables is a binomial distribution sample when bits are uniformly sampled. This distribution has standard deviation $\sigma = 2.828$

Algorithm 1 Binomial distribution sampler with Ψ_{16} as written in NewHope

▷ sample two 16-bits uniformly random and return their difference

1: **function** SAMPLE(big_integers)
2: $x \leftarrow$ generate a 16 bits number uniformly random
3: $y \leftarrow$ generate a 16 bits number uniformly random
4: **return:** $x - y$

Visualization of NewHope's binomial sampler

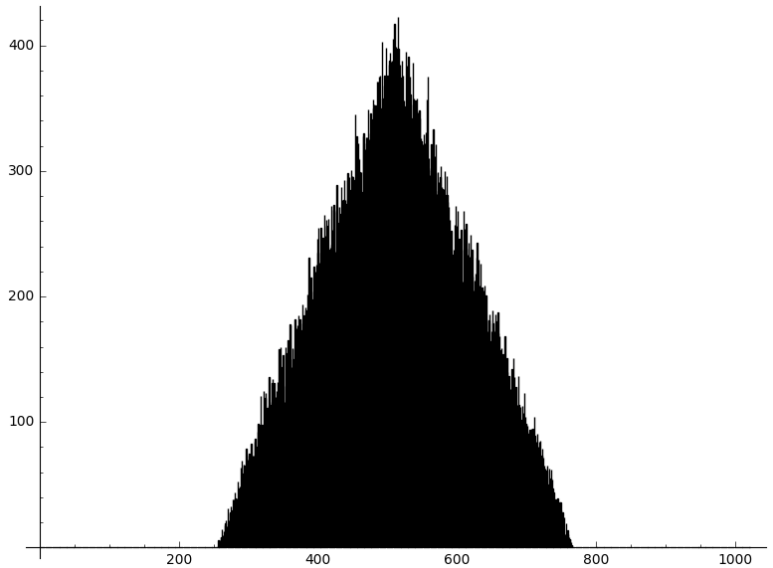


Figure: Plot of Binomial distribution Ψ_8, *shifted* for visualization purposes

Sharing random polynomial with other party

Two fundamentally different approaches

- BCNS during install of Open-SSL will generate a uniformly random shared polynomial and will use it for all the key-exchanges. Downside is key-exchange initiator needs to send the shared polynomial to other party as well and sending 4KB of data is not an ideal solution
- NewHope protocol addresses the issue by sending a seed so that other party can recreate the shared polynomial by themselves instead

In details, NewHope uses 256-bit seed through SHAKE-128 which is a hash function with arbitrary length output. Then slicing the resulting 16384 bits into 1024×16 bit numbers

Deterministically create a shared polynomials from a seed

Algorithm 2 Extracting coefficients of shared Polynomial as described in NewHope

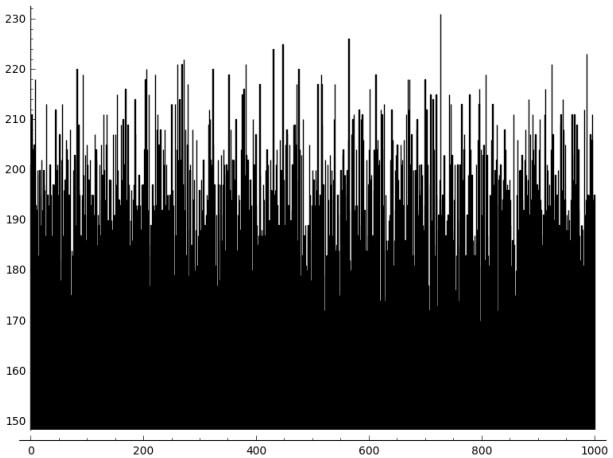
modulus ← 12289
number_of_coefficients ← 1024

function CREATE_COEFFICIENT_ARRAY(seed)
▷ set the seed and length of output in bits
    raw ← SHAKE-128(seed, number_of_coefficients ×16)
    chunks ← convert hex digest of 'raw' to chunks (or sub-strings) of bit length = 16
    for i ← 0 to 1024 **do**
▷ reduce chunk to modulo 2^{14} by setting 2 significant bits to zero
        chunks[i] ← chunk[i] XOR 0xC000
▷ if coefficient is greater than modulus then set to 0
        chunks[i] ← 0 if chunk[i] ≥ modulus else chunk
    return chunks
▷ set the seed and then extract coefficients
seed ← 256 bit sampled uniformly random
coefficients ← create_coefficient_array(seed)

Not all coefficients should be non-zero

Shared polynomials should be large relative to noise, but not necessary too large

The advantage is some of the coefficients of shared polynomial would be set to 0 hence shared polynomial would not be too large and not too small. This results in a increased speed.
We can see that number of non-zero coefficients of shared polynomial is on-average 200 out of all 1024 coefficients.



Speed comparison of RLWE key-exchange

NewHope's AVX2 implementation outperforms fastest elliptic-curve based key-exchange

NewHope's AVX2 is faster than fastest Diffie-Hellman implementation (X25519), that is Sandy2x implemented using AVX2:

Protocol	BCNS	NewHope's C ref.	NewHope's AVX2 ref.	X25519
Key generation (server)	2477958	258246	88920	52169
Key generation (client)	3995977	384994	110986	52169
Shared key computation	481937	86280	19422	159128
Sum of clock cyles	$\approx$ 6955K	$\approx$ 729K	$\approx$**219K**	$\approx$**263K**

Table: Clock cycle comparisons of BCNS, NewHope and fastest X25519 implementation (i.e. Sandy2x)

NewHope vs. post quantum key-exchange competitors

Key-exchange with digital signature either RSA or ECDSA

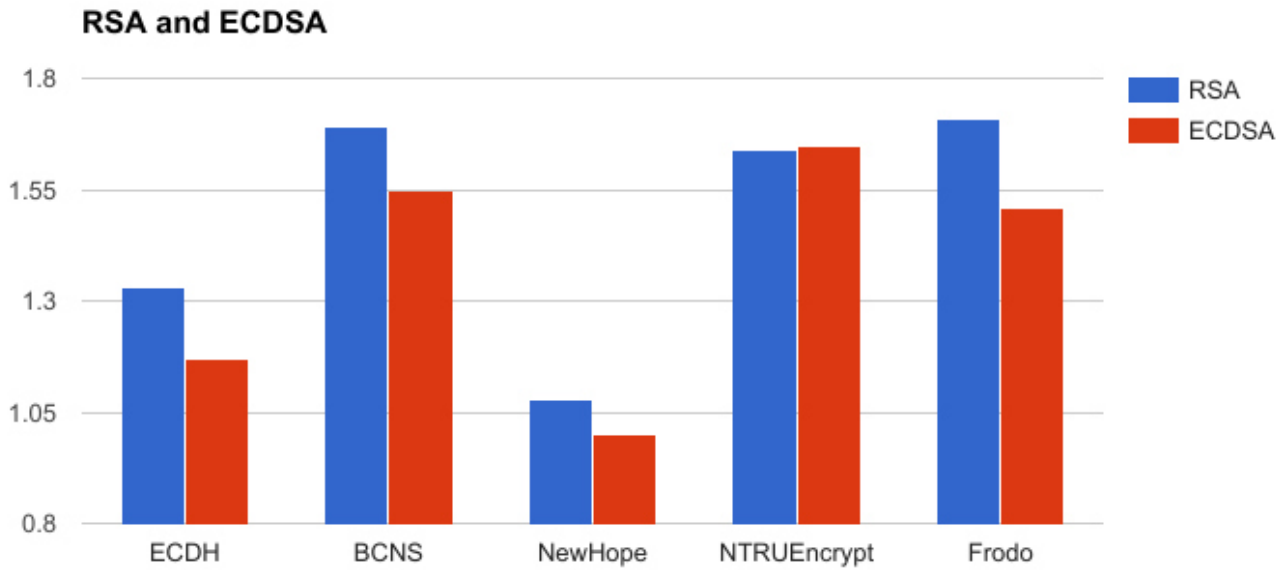


Figure: Performance comparison of KEX protocols in conjunction with RSA and ECDSA

The shorter the bar chart, the shorter TLS handshake would be.

Addition to popular cryptographic libraries

Two new ciphersuites

Ring-LWE-based ciphersuite described in the previous subsection are both implemented into Open-SSL and Boring-SSL. Specifically, two new ciphersuites, all designed to achieve a 128-bit security level. The two ciphersuites are: `RLWE-ECDSA-AES128-GCM-SHA256` and `RLWE-RSA-AES128-GCM-SHA256`, and they consist of:

- Key-exchange based on Ring-LWE key exchange
- Authentication based on ECDSA (elliptic curve Digital Signature Algorithm) or RSA digital signatures
- Authenticated encryption based on AES-128 in GCM (Galois Counter Mode)
- Key derivation and hashing based on SHA-256

Basic implementations of all reconciliations methods

SageMath links various math libraries together under one interface

BCNS and NewHope implementations are highly efficient but hard to read and difficult to modify

- One goal of this thesis was to implement a simple to understand, easy to modify and straightforward implementation of all four reconciliation mechanisms (i.e. Regev, Ding, Peikert and NewHope)
- We used SageMath (extension of Python programming language) to implement all reconciliation methods

`https://github.com/amir734jj/LWE-KEX`

Summary of thesis

This thesis provides:

1. brief survey on lattice based cryptography how it relates to LWE problem
2. reason behind creation of Ring-LWE (speed!)
3. basics of Ring-LWE key-exchange and need for reconciliation
4. reviewed parameter choices and compare the specifics of different reconciliation algorithms
5. analysis of two real-world and highly optimized implementations of Ring-LWE key-exchange protocols
6. noise sampler, ways to generate shared polynomial and speed comparison

Conclusion and the main take away

Ring-LWE offers quantum resistant + speed

LWE and Ring-LWE promise a quantum safe passive key-exchange and recent implementations not only offer a drop-in replacement for cryptography software libraries but they also outperform the existing key-exchange algorithm as well.

So, switching to NewHope results in post quantum safe key-exchange and faster speed!

Future works

Faster reconciliation + exploring other candidates

- Improving reconciliation algorithms with the goal of achieving higher success probability
- Studying claim of security against quantum computers and even classical computers needs
- Standardizing key-exchange protocols and their efficient implementations in various applications and programming languages
- Exploring other candidates for post-quantum cryptography and attempts to make them more efficient in terms of both space and time complexity

Reference

Applebaum, B., Cash, D., Peikert, C., and Sahai, A. (2009).

Fast cryptographic primitives and circular-secure encryption based on hard learning problems.
In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '09, pages 595–618, Berlin, Heidelberg. Springer-Verlag.

Boneh, D. and Lipton, R. J. (1995).

Quantum Cryptanalysis of Hidden Linear Functions, pages 424–437.
Springer Berlin Heidelberg, Berlin, Heidelberg.

Lenstra, A. K., Lenstra, H. W., and Lovász, L. (1982).

Factoring polynomials with rational coefficients.
Mathematische Annalen, 261(4):515–534.

Shor, P. W. (1994).

Algorithms for quantum computation: Discrete logarithms and factoring.
In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, pages 124–134, Washington, DC, USA. IEEE Computer Society.

Singh, V. (2015).

A practical key exchange for the internet using lattice cryptography.
Cryptology ePrint Archive, Report 2015/138.

Viterbo, E. and Biglieri, E. (1996).

Computing the voronoi cell of a lattice: the diamond-cutting algorithm.
IEEE Transactions on Information Theory, 42(1):161–171.

Acknowledgement

Thank you!

Advisor: Dr. Xu (gxu4uwm@uwm.edu)

Thesis committee members:

- Dr. Dumitrescu (dumitres@uwm.edu)
- Dr. Wang (wang289@uwm.edu)