Investigating Lattice-based Cryptography

BY

MICHAELA MOLINA

ADVISOR: DR. CHRIS POLLETT

COMMITTEE: DR. THOMAS AUSTIN

DR. ROBERT CHUN

Table of Contents

Purpose

Purpose

- Symmetric keys
- Public keys
- Quantum computers
- Quantum-resistant cryptography
- Rust programming language
- What we did in this project
- Goal of this project

Purpose
Symmetric Keys

- Used since ancient times
- Believed to be secure “forever”
- What is the problem?

Purpose
Public Keys

- Invented in 1976 by Diffie and Hellman
- Solve the problem with symmetric keys
- Diffie-Hellman key exchange and RSA in use today

Purpose
Quantum Computers

- Have capabilities beyond being faster
- Threaten public key systems but not symmetric key systems
- Shor's algorithm
- Why are public key systems still safe?

Purpose
Post-quantum Cryptography

- 6 approaches
- Security reductions
- Cost of replacing current public-key systems

Purpose
Rust Programming Language

- Fast and memory efficient
- Memory safe
- Useful error messages and package manager

Purpose
What we did in this project

- Implement the McEliece public key system using Rust
  - Code-based
  - Security reduction to SDP
- Implement the Regev public key system using Rust
  - Lattice-based
  - Security reduction to GAPSVP and SIVP
- Run experiments to show the cost of repacing current public-key systems with post-quantum systems

Purpose
Goal of this Project

- Understand and implement two candidates for post-quantum algorithms
  - McEliece public key system
  - Regev public key system
- Analyze the cost of replacing pre-quantum algorithms with post-quantum algorithms
- Use a new programming language (Rust) to provide some novelty

Background

Background

- Finite fields
- Linear codes
- Learning with errors

Background
Finite fields

- Set of elements together with two binary operations such that for all elements of the set the field axioms are obeyed
- Field axioms:
  - closure under addition
  - associativity under addition
  - additive identity
  - additive inverse
  - commutativity of addition
  - closure under multiplication
  - associativity of multiplication
  - distributive laws
  - commutativity of multiplication
  - multiplicative identity
  - no zero divisors
  - multiplicative inverse

Background
Finite fields

- Set of elements together with two binary operations such that for all elements of the set the field axioms are obeyed
- Field axioms:
  - closure under addition
  - associativity under addition
  - additive identity
  - additive inverse
  - commutativity of addition
  - closure under multiplication
  - associativity of multiplication
  - distributive laws
  - commutativity of multiplication
  - multiplicative identity
  - no zero divisors
  - multiplicative inverse

GF(2^m)

irreducible polynomial: $f(x) = x^3 + x + 1$

elements: $0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1$

$= 000, 001, 011, 100, 101, 110, 111$

GF(2^{mt})

irreducible polynomial: $g(x) = x^2 + x + 110$

elements: $0x + 0, 0x + 1, 0x + 10, 0x + 11,$

$0x + 100, 0x + 101, 0x + 110, 0x + 111,$

$1x + 0, 1x + 1, 1x + 10, 1x + 11, 1x + 100,$

$1x + 101, 1x + 110, 1x + 111,$

...

Background
Linear Codes

- Coding theory invented in 1978
- Wants to select a message at one point and reproduce it exactly or approximately at another point
- Binary symmetric channel
- Binary symmetric source, sender
- (n, k) linear code
- Encoding algorithm – generator matrix
- Decoding algorithm – parity check matrix, syndrome

Background
Learning With Errors

- For an integer $n \geq 1$ and a real number $\epsilon \geq 0$ the "learning from parity with error" problem is as follows.
  - Given a list of "equations with errors"

$$\langle \vec{s}, \vec{a}_1 \rangle \approx_\epsilon b_1 (mod\ 2)$$
$$\langle \vec{s}, \vec{a}_2 \rangle \approx_\epsilon b_2 (mod\ 2)$$
$$\ldots$$
$$\langle \vec{s}, \vec{a}_m \rangle \approx_\epsilon b_m (mod\ 2)$$

  - Where each $\vec{a}_i$ is a vector chosen uniformly from Z_2^n
  - Where $\langle \vec{s}, \vec{a}_i \rangle$ is the dot product of $\vec{s}$ and $\vec{a}_i$ modulo 2
  - And where $\approx_\epsilon$ means congruent with probability ϵ

- Find the unknown $\vec{s} \in Z_2^n$

Steps

Steps

- Key generation
- Encryption
- Decryption

Steps
McEliece: Key Generation

- Step 1:
  - Generate an irreducible polynomial of degree m over GF(2), call it f(x)
  - Generate an irreducible polynomial of degree t over GF(2^m=n), call it g(x)
- Step 2:
  - Find the parity check matrix for the linear code corresponding to g(x)
- Step 3:
  - Private key: find the generator matrix for the (n,k) linear code
- Step 4:
  - Public key: scramble the generator matrix

Steps
McEliece: Encryption

- Step 1:
  - choose message m and compute y = mG'
- Step 2:
  - choose error vector e with t errors and compute y '= mG'+e
  - Send y'

Steps
McEliece: Decryption

- Step 1:
  - Compute $y'P^{-1} = mSGP^{-1} + eP^{-1}$
- Step 2:
  - Use Patterson's algorithm to find eP^{-1}
  - Add eP^{-1} to $y'P^{-1}$ to get $mSGPP^{-1} = mSG$
- Step 3:
  - Find mS by row reducing $[G^T|(mSG)^T]$
- Step 4:
  - Find m by computing mSS^{-1}

Steps
Regev: Key Generation

all arithmetic done modulo p

- Step 1
  - Choose input parameters n and ϵ, n and integer and ϵ real
  - Set p such that $n^2 \leq p \leq 2n^2$ and set $m = (1+\epsilon)(1+n)\log p$
- Step 2
  - Private key: choose $\vec{s} \in Z_p^n$ randomly
- Step 3
  - Public key:
    - Create the error list $e_1 \ldots e_m$ by choosing e_i from the probability distribution $\chi = \overline{\Psi}_{\alpha(n)}$
    - Create the list $(\vec{a}_i, b_i)$ for $i = 1, \ldots, m$ where each $\vec{a}_i$ is chosen randomly from Z_p^n and $b_i = \langle \vec{a}_i, \vec{s} \rangle + e_i$

Steps
Regev: Key Generation

- Choosing e_i from the probability distribution $\chi = \overline{\Psi}_{\alpha(n)}$
  - Sample from a normal distribution with mean 0 and standard deviation $\frac{\alpha(n)}{\sqrt{2\pi}}$
    - Where $\alpha(n) = \frac{1}{\sqrt{n}\log^2 n}$
  - Reduce the result modulo 1
  - Multiply by p
  - Round to the closest integer modulo p

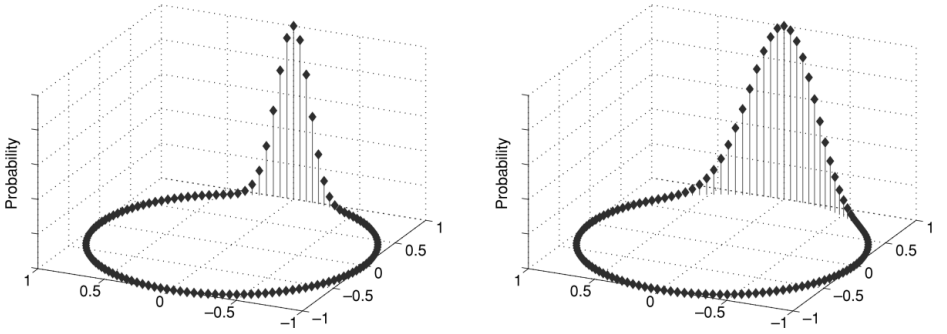


FIG. 1. $\bar{\Psi}_\alpha$ for $p = 127$ with $\alpha = 0.05$ (left) and $\alpha = 0.1$ (right). The elements of $\mathbb{Z}_p$ are arranged on a circle.

Steps
Regev: Encryption

all arithmetic done modulo p

- Step 1
  - Choose a subset S of the public key
- Step 2
  - Encrypt a 0 as $(\sum_{i \in S} \boldsymbol{a_i}, \sum_{i \in S} b_i)$
  - Encrypt a 1 as $(\sum_{i \in S} \boldsymbol{a_i}, \left\lfloor \frac{p}{2} \right\rfloor + \sum_{i \in S} b_i)$
  - Send the encrypted bit $(\vec{a}, b)$

Steps
Regev: Decryption

all arithmetic done modulo p

- Step 1
  - Compute $\vec{b} - \langle \vec{a}, \vec{s} \rangle$ from the bit $(\vec{a}, b)$
- Step 2
  - Compute the circular distance of $\vec{b} - \langle \vec{a}, \vec{s} \rangle$ to 0
  - Compute the circular distance of $\vec{b} - \langle \vec{a}, \vec{s} \rangle$ to $\left\lfloor \frac{p}{2} \right\rfloor$
  - If the first distance is smaller, decrypt as a 0
  - Otherwise, decrypt as 1

$p-1$ 0 1 2

$p-2$ $\dots$

$\dots$

$\left\lfloor \frac{p}{2} \right\rfloor + 1$ $\left\lfloor \frac{p}{2} \right\rfloor$ $\left\lfloor \frac{p}{2} \right\rfloor - 1$

Example

Example
McEliece: Key Generation

- Step 1:
  - Choose n and t with n a power of 2
  - Generate an irreducible polynomial of degree m over GF(2), call it f(x)
  - Generate an irreducible polynomial of degree t over GF(2^m=n), call it g(x)
- Example:

$(n, t) = (8,2)$

$f(x) = x^3 + x + 1$

$g(x) = x^2 + x + 111$

Example McEliece: Key Generation

$$1^{-1}\ mod\ x^3+x+1=001$$
$$x^{-1} mod\ x^3+x+1=x^2+1=101$$
$$(x+1)^{-1} mod\ x^3+x+1=x^2+x=110$$
$$(x^2)^{-1} mod\ x^3+x+1=x^2+x+1=111$$
$$(x^2+1)^{-1} mod\ x^3+x+1=x=010$$
$$(x^2+x)^{-1} mod\ x^3+x+1=x+1=011$$
$$(x^2+x+1)^{-1} mod\ x^3+x+1=x^2=100$$

- Step 2:
  - Find the parity check matrix for the linear code corresponding to g(x)
- Example:

choose L: $L=\{\alpha_1,\dots,\alpha_n\}=\{011,100,110,001,000,101,010,111\}$

create x:

$$x=\begin{bmatrix}001 & 000\\ 001 & 001\end{bmatrix}$$

$$x=\begin{bmatrix}g_t & 0 & 0 & \dots & 0\\ g_{t-1} & g_t & 0 & 0 & 0\\ \dots & \dots & \dots & 0 & 0\\ g_1 & g_2 & g_3 & \dots & g_t\end{bmatrix}$$

$$y=\begin{bmatrix}001 & 001 & 001 & 001 & 001 & 001 & 001 & 001\\ 011 & 100 & 110 & 001 & 000 & 101 & 010 & 111\end{bmatrix}$$

$$y=\begin{bmatrix}\alpha_1^0 & \alpha_2^0 & 0 & \dots & \alpha_n^0\\ \alpha_1^1 & \alpha_2^1 & 0 & \dots & \alpha_n^1\\ \dots & \dots & \dots & \dots & \dots\\ \alpha_1^{t-1} & \alpha_2^{t-1} & 0 & \dots & \alpha_n^{t-1}\end{bmatrix}$$

$$z=\begin{bmatrix}001 & 000 & 000 & 000 & 000 & 000 & 000 & 000\\ 000 & 010 & 000 & 000 & 000 & 000 & 000 & 000\\ 000 & 000 & 110 & 000 & 000 & 000 & 000 & 000\\ 000 & 000 & 000 & 100 & 000 & 000 & 000 & 000\\ 000 & 000 & 000 & 000 & 100 & 000 & 000 & 000\\ 000 & 000 & 000 & 000 & 000 & 010 & 000 & 000\\ 000 & 000 & 000 & 000 & 000 & 000 & 001 & 000\\ 000 & 000 & 000 & 000 & 000 & 000 & 000 & 110\end{bmatrix}$$

$$z=\begin{bmatrix}\frac{1}{g(\alpha_1)} & 0 & 0\\ 0 & \frac{1}{g(\alpha_2)} & 0\\ \dots & \dots & \dots\\ 0 & \dots & \frac{1}{g(\alpha_n)}\end{bmatrix}$$

compute H=xyz:

$$H=xyz=\begin{bmatrix}001 & 010 & 110 & 100 & 100 & 010 & 001 & 110\\ 010 & 001 & 100 & 000 & 100 & 011 & 011 & 010\end{bmatrix}$$

convert H to binary:

$$H=\begin{bmatrix}0 & 0 & 1 & 1 & 1 & 0 & 0 & 1\\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1\\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1\\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0\end{bmatrix}$$

Example
McEliece: Key Generation

- Step 3:
  - Find the generator matrix for the linear code

$$nullspace(H) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$G = nullspace^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Example
McEliece: Key Generation

- Step 4:
  - Scramble the generator matrix

$$S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$G' = SGP = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Example
McEliece: Encryption

- Step 1:
  - choose message m and compute mG'

$$m = [0 \quad 1]$$

$$G' = SGP = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$mG' = [1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1]$$

Example
McEliece: Encryption

- Step 2:
  - choose error vector e and compute y' = mG'+e
  - Send y

$$e = [1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0]$$

$$mG' = [1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1]$$

$$y' = mG' + e = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1]$$

Example
McEliece: Decryption

- Step 1:
  - Compute $y'P^{-1} = mSGP^{-1} + eP^{-1}$

$$y' = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1]$$

$$y'P^{-1} = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1]$$

$$P^{-1} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Example
McEliece: Decryption

- Step 2:
  - Use Patterson's algorithm to find eP^{-1}
    - a) Compute the syndrome s(x)

$y'P^{-1} = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1]$

$L = \{011,100,110,001,000,101,010,111\}$

$s(x) = \sum_{i=1}^{n} \frac{y_i}{x-\alpha_i} mod\ g(x) = \left(\frac{1}{x+110}\right) modg(x) + \left(\frac{1}{x}\right) modg(x) + \left(\frac{1}{x+101}\right) modg(x) +$

$\left(\frac{1}{010x+011}\right) modg(x) =$

$= (x+110)^{-1}\ modg(x) + (x)^{-1} modg(x) + (x+101)^{-1} modg(x) + (010x+011)^{-1} modg(x)$

$= 110x + 1$

Example
McEliece: Decryption

- Step 2:
  - Use Patterson's algorithm to find eP^{-1}
    - b) Compute $v(x) \equiv \sqrt{s(x)^{-1} - x}\ mod\ g(x) = 110x + 111$
    - c) Find $a(x)$ and $b(x)$ so that $a(x) \equiv b(x)v(x)\ mod g(x)$ and $\deg(a) \leq \left\lfloor \frac{t}{2} \right\rfloor$ and $\deg(b) \leq \left\lfloor \frac{t-1}{2} \right\rfloor$: $a(x) = 110x + 111,\ \ b(x) = 1$
    - d) Set $\sigma(x) = a(x)^2 + x \cdot b(x)^2 = 10x^2 + x + 11$
    - e) Locate the errors by plugging in the values of L: they are at indices 1 and 3
      - $\sigma(011) = 001$
      - $\sigma(100) = 000$
      - $\sigma(110) = 001$
      - $\sigma(001) = 000$
      - $\sigma(000) = 011$
      - $\sigma(101) = 011$
      - $\sigma(010) = 010$
      - $\sigma(111) = 010$

Example
McEliece: Decryption

- Step 2:
  - Use Patterson's algorithm to find eP^{-1}

$$eP^{-1} = [0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

  - Add eP^{-1} to $y'P^{-1}$ to get $mSGPP^{-1} = mSG$

$$y'P^{-1} = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1]$$
$$eP^{-1} = [0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$mSG = eP^{-1} = [0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1]$$

Example McEliece: Decryption

- Step 3:
  - Find mS by row reducing $[G^T|(mSG)^T]$

$$[G^T|(mSG)^T] = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$[G^T|(mSG)^T]\ reduced = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Example
McEliece: Decryption

- Step 4:
  - Find m by computing mSS^{-1}

$$mS = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$S^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$m = mSS = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

Example
Regev: Key Generation

- Step 1
  - Choose n = 3, ϵ = 0.5. Set p to 10, then m = 20
- Step 2
  - Private key: choose s = [2,7,3]
- Step 3
  - list of a:

$a_1 = [1,0,2]$
$a_2 = [8,2,3]$
$a_3 = [1,6,3]$
$a_4 = [5,5,7]$
$a_5 = [2,0,4]$
$a_6 = [2,3,7]$
$a_7 = [8,5,2]$
...

errors:
$e_1 = 9$
$e_2 = 1$
$e_3 = 0$
$e_4 = 0$
$e_5 = 2$
$e_6 = 0$
$e_7 = 0$
...

list of b:
$b_1 = (1)(2) + (0)(7) + (2)(3) + 9$ = 2+6+9=17 mod 10 = 7
...
$b_5 = (2)(2) + (0)(7) + (4)(3) + 2$ = 4+12+2 = 18 mod 10 = 8
...
$b_7 = (8)(2) + (5)(7) + (2)(3) + 0 =$ 16+35+6 = 57 mod 10 = 7

public key:
$([1,0,2], 7)$
...
$([2,0,4], 8)$
...
$([8,5,2], 7)$
...

Example
Regev: Encryption

all arithmetic done modulo p

- Step 1

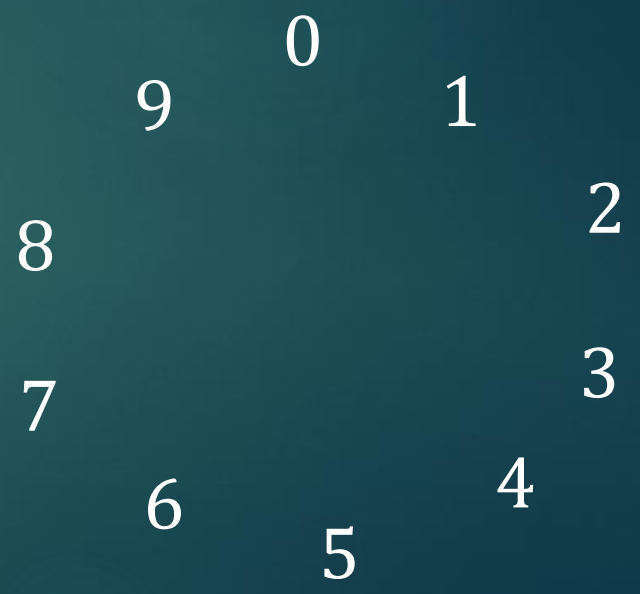
  Choose $([1,0,2],7)$
  $([2,0,4],8)$
  $([8,5,2],7)$

- Step 2
  - Encrypt a 0: as $(\sum_{i\in S} \boldsymbol{a_i}, \sum_{i\in S} b_i) = ([1,5,8],2)$
  - Encrypt a 1 as $(\sum_{i\in S} \boldsymbol{a_i}, \left\lfloor \frac{p}{2} \right\rfloor + \sum_{i\in S} b_i) = ([1,5,8],2+5) = ([1,5,8],7)$

Example
Regev: Decryption

all arithmetic done modulo p

- Step 1 private key s = [2,7,3]
  - Decrypting the 0 where the ciphertext is $([1,5,8], 2)$:
    - Compute $\vec{b} - \langle \vec{a}, \vec{s} \rangle = 2 - (1)(2)+(5)(3)+(8)(7) = 2 - 61 = 2 - 1 = 1$
  - Decrypting the 1 where the ciphertext is $= ([1,5,8], 7)$:
    - Compute $\vec{b} - \langle \vec{a}, \vec{s} \rangle = 7 - (1)(2)+(5)(3)+(8)(7) = 7 - 61 = 7 - 1 = 6$
- Step 2
  - Decrypting the 0:
    - Is 1 closer to 0 or 5 on the circle? It is closer to 0 -> decrypt as 0
  - Decrypting the 1:
    - Is 6 closer to 0 or 5 on the circle? It is closer to 5 -> decrypt as 1



Implementation

Implementation McEliece

- Finite field GF(2^m) arithmetic
- Finite field GF(2^mt) arithmetic
- Matrix operations
- Encrypting and decrypting operations
- System functionality

Implementation McEliece

- Finite field GF(2^m) arithmetic
  - An element of GF(2^m) is represented as a u64 in Rust
  - ```
    0b0000000000000000000000000000000000000000000000000000000000000111
    ```
  - Multiplication of the element by x is implemented by a left bit shift
  - Division of the element by x is implemented by a right bit shift
    ```

Implementation McEliece

- Finite field GF(2^m) arithmetic
  - Primary methods
    - add()
    - multiply()
    - reduce()
    - gcd()
    - inverse()
    - pow()
    - is_irreducible()
  - Helper methods
    - vec()
    - swap()
    - bit()
    - rz()
    - deg()
    - string()
    - shift_right()
    - shift_left
    - shift_right_i()
    - shift_left_i()
    - get_random_polynomial()
    - get_irreducible_polynomial()
    - create_L()

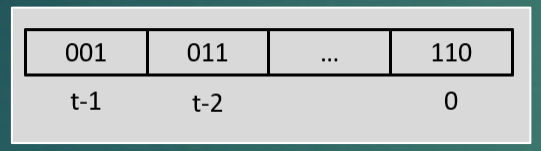
Implementation McEliece

- Finite field GF(2^m) arithmetic
  - Primary methods
    - add()
    - multiply()
    - reduce()
    - gcd()
    - inverse()
    - pow()
    - **is_irreducible()**

```
// Ben-Or irreducibility test pseudocode
for i:=1 to n/2 do
    g := gcd(f, x^(q^i) - x mod f);
    if g != 1 return false
return true
```

```
pub fn is_irreducible(f:u64) -> bool {
    let mut m = deg(f);
    let mut i = 0;
    let mut rhs = 2 as u64; // rhs = x
    rhs = multiply(rhs, rhs, f); // rhs = rhs^2 mod f
    i += 1;
    while i <= (m as f64/2 as f64).floor() as u64 {
        let mut rhs_plus_x = add(rhs,2);
        let mut gcd = gcd(f, rhs_plus_x);
        if gcd != 1 {
            return false;
        }
        rhs = multiply(rhs, rhs, f); // rhs = rhs^2 mod f
        i += 1;
    }
    return true;
}
```

Implementation McEliece

- Finite field GF(2^mt) arithmetic
  - An element of GF(2^mt) is represented as a vector of u64s
  - 

  - Multiplication of the element by x is implemented by a left bit shift
    - Where the left bit shift is implemented as inserting a 0 at index 0
  - Division of the element by x is implemented by a right bit shift
    - Where the right bit shift is implemented as removing the element at index 0

Implementation McEliece

- Finite field GF(2^mt) arithmetic
  - Primary methods
    - add()
    - gcd()
    - inverse()
    - reduce()
    - square()
    - sqrt()
    - is_irreducible()
  - Helper methods
    - get_bx_ax()
    - multiply_by_constant()
    - is_one()
    - is_constant()
    - is_zero()
    - shift_left()
    - shift_right()
    - shift_left_i()
    - shift_right_i()
    - deg()
    - string()
    - compute()
    - get_random_polynomial()
    - get_irreducible_polynomial()

Implementation McEliece

- Finite field GF(2^mt) arithmetic
  - Primary methods
    - add()
    - gcd()
    - inverse()
    - reduce()
    - square()
    - sqrt()
    - **is_irreducible()**

```
// Ben-Or irreducibility test pseudocode
for i:=1 to n/2 do
     g := gcd(f, x^(q^i) - x mod f);
     if g != 1 return false
return true
```

```
pub fn is_irreducible(g:&Vec<u64>, f:u64) -> bool {
     let mut t = deg(&g);
     let mut m = f1::deg(f);
     let mut rhs = vec![0,1]; // rhs = x
     let t_over_2 = (t as f64/2 as f64).floor() as u64;
     let mut j = 0;
     for i in 0..m { // square rhs m times
          rhs = square(&rhs, &g, f);
          reduce(&mut rhs, &g, f);
          exp *= 2;
     }
     j += 1;
     while j <= t_over_2 {
          let mut rhs_plus_x = add(&mut rhs, &vec![0 as u64, 1 as u64]);
          let mut gcd = gcd(&g, &rhs_plus_x, f);
          if !is_one(&gcd) {
               return false;
          }
          for i in 0..m { // square rhs m times
               rhs = square(&rhs, &g, f);
               reduce(&mut rhs, &g, f);
          }
          j += 1;
     }
     return true;
}
```

Implementation McEliece

- Matrix operations
  - Primary methods
    - generator()
    - dense_nonsingular_matrix()
    - permutation_matrix()
    - multiply()
    - x()
    - y()
    - z()
    - nullspace_t()
- Rust library `peroxide`
  - peroxide Matrix holds u64s
- Helper methods
  - rref_for_decrypt()
  - rref_mod_2()
  - mod2()
  - mod2_matrix()
  - row_is_zero()
  - is_col()
  - find_col()
  - append()
  - binary_matrix()

Implementation McEliece

- Encrypting and decrypting operations
  - Primary methods
    - encrypt()
    - decrypt()
    - error_vector()
    - sigma()
    - syndrome()
    - choose_error_vector()

Implementation McEliece

- System functionality
  - Primary methods
    - create_system()
    - initialization_vector()
    - encrypt()
    - decrypt()
- Structs
  - public_key
  - private_key
- Helper methods
  - matrix()
  - matrix_blocks_into_vec()
  - vec_into_matrix_blocks()
  - xor_matrix()

Implementation McEliece

- System functionality
  - Primary methods
    - create_system()
    - initialization_vector()
    - encrypt()
    - decrypt()
- Structs
  - **public_key**
  - **private_key**
- Helper methods
  - matrix()
  - matrix_blocks_into_vec()
  - vec_into_matrix_blocks()
  - xor_matrix()

```
pub struct public_key {
    generator: Matrix,
    length: u64,
    dimension: u64,
    t: u64
}

pub struct private_key {
    S: Matrix,
    G: Matrix,
    P: Matrix,
    L: Vec<u64>,
    g: Vec<u64>,
    f: u64,
    length: u64,
    dimension: u64
}
```

Implementation Regev

- One file
  - Primary methods
    - create_system()
    - draw_vector_from_Znp()
    - get_list()
    - inner_product()
    - draw_element_from_X()
    - alpha()
    - encrypt_bit()
    - choose_subset()
    - add_vector()
    - decrypt_bit()
    - circular_distance()
  - Structs
    - public_key
    - private_key
    - pair
  - Helper Methods
    - all_possible_messages()
    - vec()
    - is_equal()
    - encrypt_vector()
    - decrypt_vector()
    - encrypt()
    - decrypt()

Implementation Regev

- Structs
  - pair
  - public_key
  - private_key

```
pub struct pair {
    pub a: Vec<u64>,
    pub b: u64
}

pub struct public_key {
    pub m: u64,
    pub n: u64,
    pub p: u64,
    pub list: Vec<pair>
}

pub struct private_key {
    pub s: Vec<u64>,
    pub p: u64
}
```

Implementation Regev

- Primary methods
  - create_system()
  - draw_vector_from_Znp()
  - get_list()
  - inner_product()
  - draw_element_from_X()
  - alpha()
  - encrypt_bit()
  - choose_subset()
  - add_vector()
  - decrypt_bit()
  - **circular_distance()**

```
fn circular_distance(point:u64, target:u64, modulus:u64) -> u64
    let diff = (point as i64 - target as i64).abs() as u64;
    return std::cmp::min(diff, modulus-diff);
}
```

Tests

Tests
McEliece

- Finding an irreducible polynomial
- For degree n should take n tries

Tests
McEliece

- Finding an irreducible polynomial
- For degree n should take n tries
- For polynomial over GF(2), idea holds for up to n = 50

```
average tries for polynomial of degree 1: 0
average tries for polynomial of degree 2: 3
average tries for polynomial of degree 3: 3
average tries for polynomial of degree 4: 6
average tries for polynomial of degree 5: 4
average tries for polynomial of degree 6: 7
average tries for polynomial of degree 7: 7
average tries for polynomial of degree 8: 8
average tries for polynomial of degree 9: 9
average tries for polynomial of degree 10: 8
$
```

Tests

McEliece

McEliece suggested m = 10, t = 50

- Finding an irreducible polynomial
- For degree n should take n tries
- For polynomial over GF(2^3):
  - t = 2: 10 tries on average
  - t = 3: 20 tries on average
  - t = 4: 52 tries on average
  - t = 5: 216 tries on average
- Found maximum parameters
- Standalone test:
  - t = 5, m = 10
  - Tested 500,000 polynomials and never found one

	m	max t	t	max m
	1	2	2	14
	2	3	3	14
	3	9	4	--
	4	9		
	5	7		
	6	--		

-- took more than 30 seconds

Tests McEliece

1	(2,1)	11	(8,6)	21	(16,7)	31	(2,2)	41	(2048,2)	51	(128,3)
2	(2,2)	12	(8,7)	22	(16,8)	32	(4,2)	42	(4096,2)	52	(256,3)
3	(4,1)	13	(8,8)	23	(16,9)	33	(8,2)	43	(8192,2)	53	(512,3)
4	(4,2)	14	(8,9)	24	(32,1)	34	(16,2)	44	(16384,2)	54	(1024,3)
5	(4,3)	15	(16,1)	25	(32,2)	35	(32,2)	45	(2,3)	55	(2048,3)
6	(8,1)	16	(16,2)	26	(32,3)	36	(64,2)	46	(4,3)	56	(4096,3)
7	(8,2)	17	(16,3)	27	(32,4)	37	(128,2)	47	(8,3)	57	(8192,3)
8	(8,3)	18	(16,4)	28	(32,5)	38	(256,2)	48	(16,3)	58	(16384,3)
9	(8,4)	19	(16,5)	29	(32,6)	39	(512,2)	49	(32,3)	59	
10	(8,5)	20	(16,6)	30	(32,7)	40	(1024,2)	50	(64,3)	60	

All possible values of (n,t).

Tests McEliece

1	(2,1)	11	(8,6)	21	(16,7)	31	(2,2)	41	(2048,2)	51	(128,3)
2	(2,2)	12	(8,7)	22	(16,8)	32	(4,2)	42	(4096,2)	52	(256,3)
3	(4,1)	13	(8,8)	23	(16,9)	33	(8,2)	43	(8192,2)	53	(512,3)
4	(4,2)	14	(8,9)	24	(32,1)	34	(16,2)	44	(16384,2)	54	(1024,3)
5	(4,3)	15	(16,1)	25	(32,2)	35	(32,2)	45	(2,3)	55	(2048,3)
6	(8,1)	16	(16,2)	26	(32,3)	36	(64,2)	46	(4,3)	56	(4096,3)
7	(8,2)	17	(16,3)	27	(32,4)	37	(128,2)	47	(8,3)	57	(8192,3)
8	(8,3)	18	(16,4)	28	(32,5)	38	(256,2)	48	(16,3)	58	(16384,3)
9	(8,4)	19	(16,5)	29	(32,6)	39	(512,2)	49	(32,3)	59	
10	(8,5)	20	(16,6)	30	(32,7)	40	(1024,2)	50	(64,3)	60	

constraint: $2^m > mt$
where $n = 2^m$

Tests McEliece

dnw = did not wait

set #	(n,t)	# msgs	# errs	total	set #	(n,t)	# msgs	# errs	total	set #	(n,t)	# msgs	# errs	total
7	(8,2)	100	10	1000	35	(32,2)	100	10	1000	48	(16,3)	100	10	1000
16	(16,2)	100	10	1000	36	(64,2)	100	10	1000	49	(32,3)	100	10	1000
17	(16,3)	100	10	1000	37	(128,2)	10	1	10	50	(64,3)	100	10	1000
25	(32,2)	100	10	1000	38	(256,2)	10	1	10	51	(128,3)	1	1	1
26	(32,3)	100	10	1000	39	(512,2)	1	1	1	52	(256,3)	1	1	1
27	(32,4)	100	10	1000	40	(1024,2)	1	1	1	53	(512,3)	1	1	1
28	(32,5)	100	10	1000	41	(2048,2)	dnw			54	(1024,3)	dnw		
29	(32,6)	100	10	1000	42	(4096,2)	dnw			55	(2048,3)	dnw		
33	(8,2)	100	10	1000	43	(8192,2)	dnw			56	(4096,3)	dnw		
34	(16,2)	100	10	1000	44	(16384,2)	dnw			57	(8192,3)	dnw		

All possible values of (n,t).

Tests
McEliece

```
(n,t) = (16,3)
1 tries to find irreducible f
26 tries to find irreducible g
f = 0b00000000000000000000000000000000000000000000000000000000000010011
g = x^3: 0b00000000000000000000000000000000000000000000000000000000000000001
x^2: 0b00000000000000000000000000000000000000000000000000000000000001111
x^1: 0b00000000000000000000000000000000000000000000000000000000000001101
x^0: 0b00000000000000000000000000000000000000000000000000000000000000110
length = 16, dimension = 4
test 48 results:
  1000/1000 correct, 0/1000 incorrect
```

```
(n,t) = (32,3)
5 tries to find irreducible f
64 tries to find irreducible g
f = 0b00000000000000000000000000000000000000000000000000000000000100101
g = x^3: 0b00000000000000000000000000000000000000000000000000000000000000001
x^2: 0b00000000000000000000000000000000000000000000000000000000000011011
x^1: 0b00000000000000000000000000000000000000000000000000000000000011111
x^0: 0b00000000000000000000000000000000000000000000000000000000000000110
length = 32, dimension = 17
test 49 results:
  1000/1000 correct, 0/1000 incorrect
```

Tests
McEliece

```
(n,t) = (64,3)
0 tries to find irreducible f
502 tries to find irreducible g
f = 0b0000000000000000000000000000000000000000000000000000000001100001
g = x^3: 0b0000000000000000000000000000000000000000000000000000000000000001
x^2: 0b0000000000000000000000000000000000000000000000000000000000100110
x^1: 0b0000000000000000000000000000000000000000000000000000000000101010
x^0: 0b0000000000000000000000000000000000000000000000000000000000101000
length = 64, dimension = 46
test 50 results:
  1000/1000 correct, 0/1000 incorrect
```

```
(n,t) = (512,3)
1 tries to find irreducible f
1563 tries to find irreducible g
f = 0b0000000000000000000000000000000000000000000000000000001010111101
g = x^3: 0b0000000000000000000000000000000000000000000000000000000000000001
x^2: 0b0000000000000000000000000000000000000000000000000000000100101101
x^1: 0b0000000000000000000000000000000000000000000000000000000111000110
x^0: 0b0000000000000000000000000000000000000000000000000000000101100
length = 512, dimension = 485
test 53 results:
  1/1 correct, 0/1 incorrect
```

Tests McEliece

- (8,2)
- (16,2)
- (16,3)
- (32,2)
- (32,3)

- 2^2 characters
- 2^4 characters
- 2^6 characters
- 2^8 characters
- 2^{10} characters
- 2^{12} characters
- 2^{14} characters

1801. - I have just returned from a visit to my landlord - the solitary neighbour that I shall be troubled with. This is certainly a beautiful country! In all England, I do not believe that I could have fixed on a situation so completely removed from the stir of society. A perfect misanthropists heaven: and Mr. Heathcliff and I are such a suitable pair

Tests

McEliece

```
0 tries to find irreducible f
0 tries to find irreducible g
f = 0b00000000000000000000000000000000000000000000000000000000000001101
g =
x^2: 0b00000000000000000000000000000000000000000000000000000000000000001
x^1: 0b00000000000000000000000000000000000000000000000000000000000000001
x^0: 0b00000000000000000000000000000000000000000000000000000000000000111
(8,2) System
Decrypted 4 characters correctly: True
Decrypted 16 characters correctly: True
Decrypted 64 characters correctly: True
Decrypted 256 characters correctly: True
Decrypted 1024 characters correctly: True
Decrypted 4096 characters correctly: True
Decrypted 16384 characters correctly: True
```

```
2 tries to find irreducible f
33 tries to find irreducible g
f = 0b00000000000000000000000000000000000000000000000000000000000011111
g =
x^3: 0b00000000000000000000000000000000000000000000000000000000000000001
x^2: 0b00000000000000000000000000000000000000000000000000000000000000101
x^1: 0b00000000000000000000000000000000000000000000000000000000000001100
x^0: 0b00000000000000000000000000000000000000000000000000000000000000101
(16,3) System
Decrypted 4 characters correctly: True
Decrypted 16 characters correctly: True
Decrypted 64 characters correctly: True
Decrypted 256 characters correctly: True
Decrypted 1024 characters correctly: True
Decrypted 4096 characters correctly: True
Decrypted 16384 characters correctly: True
```

Tests Regev

Tests
Regev

- Arbitrary initial tests: (n, E) = (5, 0.5), $\chi = \overline{\Psi}_{\alpha(n)}$
  - case 1: p = 4, m = 6
  - case 2: p = 5, m = 3
  - case 3: p = 6, m = 4

- Initial results on 2^{10} messages each:
  - Average correct over 100 tests:
    - case 1: 1022/1024
    - case 2: 1020/1024
    - case 3: 1023/1024
  - Average incorrect over 100 tests:
    - case 1: 1/1024
    - case 2: 3/1024
    - case 3: 0/1024

Tests

Regev

```
iteration 92: correct = 1018/1024
iteration 92: incorrect = 6/1024

iteration 93: correct = 1018/1024
iteration 93: incorrect = 6/1024

iteration 94: correct = 1019/1024
iteration 94: incorrect = 5/1024

iteration 95: correct = 1022/1024
iteration 95: incorrect = 2/1024

iteration 96: correct = 1020/1024
iteration 96: incorrect = 4/1024

iteration 97: correct = 1021/1024
iteration 97: incorrect = 3/1024

iteration 98: correct = 1022/1024
iteration 98: incorrect = 2/1024

iteration 99: correct = 1018/1024
iteration 99: incorrect = 6/1024

average correct over 100 tests = 1020/1024
average incorrect over 100 tests = 3/1024
$
```

Tests
Regev

- Parameter Constraints, for input (n, ϵ):
  - $n^2 \leq p \leq 2n^2$
  - $m = (1+\epsilon)(n+1)\log_2 p$
- Arbitrary initial tests: E=0.5, $\chi = \overline{\Psi}_{\alpha(n)}$
  - case 1: n=10, p =164, m = 121
  - case 2: n=12, p = 248, m = 155
  - case 3: n=15, p = 345, m = 202
- Initial results on 2^{10} messages each:
  - Average correct over 10 tests:
    - case 1: 637/1024
    - case 2: 186/1024
    - case 3: 358/1024
  - Average incorrect over 10 tests:
    - case 1: 386/1024
    - case 2: 837/1024
    - case 3: 665/2024

Tests
Regev

Suppose m = 10

public key:

$$[\overrightarrow{a_1}, b_1] = [\overrightarrow{a_1}, \langle \overrightarrow{a_1}, \vec{s} \rangle + e_1]$$
$$[\overrightarrow{a_2}, b_2] = [\overrightarrow{a_2}, \langle \overrightarrow{a_2}, \vec{s} \rangle + e_2]$$
$$[\overrightarrow{a_3}, b_3] = [\overrightarrow{a_3}, \langle \overrightarrow{a_3}, \vec{s} \rangle + e_3]$$
$$[\overrightarrow{a_4}, b_4] = [\overrightarrow{a_4}, \langle \overrightarrow{a_4}, \vec{s} \rangle + e_4]$$
$$[\overrightarrow{a_5}, b_5] = [\overrightarrow{a_5}, \langle \overrightarrow{a_5}, \vec{s} \rangle + e_5]$$
$$[\overrightarrow{a_6}, b_6] = [\overrightarrow{a_6}, \langle \overrightarrow{a_6}, \vec{s} \rangle + e_6]$$
$$[\overrightarrow{a_7}, b_7] = [\overrightarrow{a_7}, \langle \overrightarrow{a_7}, \vec{s} \rangle + e_7]$$
$$[\overrightarrow{a_8}, b_8] = [\overrightarrow{a_8}, \langle \overrightarrow{a_8}, \vec{s} \rangle + e_8]$$
$$[\overrightarrow{a_9}, b_9] = [\overrightarrow{a_9}, \langle \overrightarrow{a_9}, \vec{s} \rangle + e_9]$$
$$[\overrightarrow{a_{10}}, b_{10}] = [\overrightarrow{a_{10}}, \langle \overrightarrow{a_{10}}, \vec{s} \rangle + e_{10}]$$

encrypting a 0:

$$[\overrightarrow{a_2}, b_2] = [\overrightarrow{a_2}, \langle \overrightarrow{a_2}, \vec{s} \rangle + e_2]$$
$$[\overrightarrow{a_5}, b_5] = [\overrightarrow{a_5}, \langle \overrightarrow{a_5}, \vec{s} \rangle + e_5]$$
$$+ \quad [\overrightarrow{a_6}, b_6] = [\overrightarrow{a_6}, \langle \overrightarrow{a_6}, \vec{s} \rangle + e_6] \qquad +e_2 + e_5 + e_6$$

$$\overrightarrow{(a, b)} = [\vec{a}_{2+5+6}, b_{2+5+6}] = [\vec{a}_{2+5+6}, (\langle \overrightarrow{a_2}, \vec{s} \rangle + \langle \overrightarrow{a_5}, \vec{s} \rangle + \langle \overrightarrow{a_6}, \vec{s} \rangle)]$$

encrypting a 1:

$$[\overrightarrow{a_2}, b_2] = [\overrightarrow{a_2}, \langle \overrightarrow{a_2}, \vec{s} \rangle + e_2]$$
$$[\overrightarrow{a_5}, b_5] = [\overrightarrow{a_5}, \langle \overrightarrow{a_5}, \vec{s} \rangle + e_5]$$
$$+ \quad [\overrightarrow{a_6}, b_6] = [\overrightarrow{a_6}, \langle \overrightarrow{a_6}, \vec{s} \rangle + e_6] \qquad +e_2 + e_5 + e_6$$

$$(\vec{a}, b) = [\vec{a}_{2+5+6}, b_{2+5+6} + \left\lfloor \frac{p}{2} \right\rfloor] = [\vec{a}_{2+5+6}, (\langle \overrightarrow{a_2}, \vec{s} \rangle + \langle \overrightarrow{a_5}, \vec{s} \rangle + \langle \overrightarrow{a_6}, \vec{s} \rangle) + \left\lfloor \frac{p}{2} \right\rfloor]$$

Tests Regev

- Using the equality $\langle \overrightarrow{a_2}, \vec{s} \rangle + \langle \overrightarrow{a_5}, \vec{s} \rangle + \langle \overrightarrow{a_6}, \vec{s} \rangle = \langle \vec{a}_{2+5+6}, \vec{s} \rangle$

- decrypting the 0 with no error:

  $b - \langle \vec{a}, \vec{s} \rangle = \langle \vec{a}_{2+5+6}, \vec{s} \rangle - \langle \vec{a}_{2+5+6} \vec{s} \rangle = 0$

- decrypting the 1 with no error:

  $b - \langle \vec{a}, \vec{s} \rangle = \langle \vec{a}_{2+5+6}, \vec{s} \rangle - \langle \vec{a}_{2+5+6} \vec{s} \rangle + \left\lfloor \frac{p}{2} \right\rfloor = \left\lfloor \frac{p}{2} \right\rfloor$

Tests
Regev

- Using the equality $\langle \overrightarrow{a_2}, \vec{s} \rangle + \langle \overrightarrow{a_5}, \vec{s} \rangle + \langle \overrightarrow{a_6}, \vec{s} \rangle = \langle \vec{a}_{2+5+6}, \vec{s} \rangle$

- decrypting the 0 with some error:

  $b - \langle \vec{a}, \vec{s} \rangle = \langle \vec{a}_{2+5+6}, \vec{s} \rangle - (\langle \vec{a}_{2+5+6} \vec{s} \rangle + e_2 + e_5 + e_6) = e_2 + e_5 + e_6$

- decrypting the 1 with some error:

  $b - \langle \vec{a}, \vec{s} \rangle = \langle \vec{a}_{2+5+6}, \vec{s} \rangle - (\langle \vec{a}_{2+5+6} \vec{s} \rangle + \left\lfloor \frac{p}{2} \right\rfloor + e_2 + e_5 + e_6) = \left\lfloor \frac{p}{2} \right\rfloor e_2 + e_5 + e_6$

Tests
Regev

Recall: what "closer to 0 than to floor(p/2)" modulo p means

Suppose p = 10, $\left\lfloor \frac{p}{2} \right\rfloor$=5

Tests
Regev

Decrypting a 0:

$b - \langle \vec{a}, \vec{s} \rangle = \langle \vec{a}_{2+5+6}, \vec{s} \rangle - (\langle \vec{a}_{2+5+6} \vec{s} \rangle + +e_2 + e_5 + e_6) = +e_2 + e_5 + e_6$

Decrypting a 1:

$b - \langle \vec{a}, \vec{s} \rangle = \langle \vec{a}_{2+5+6}, \vec{s} \rangle - (\langle \vec{a}_{2+5+6} \vec{s} \rangle + \left\lfloor \frac{p}{2} \right\rfloor + +e_2 + e_5 + e_6) = \left\lfloor \frac{p}{2} \right\rfloor + e_2 + e_5 + e_6$

0 1 2 3 4 5 6 7 8 9

If $x = e_2 + e_5 + e_6$ then we can see that in order for a bit to decrypt correctly,

$x \ mod \ p = 0, 1 \ or \ 2$ so that the distance will not be closer to the other number

Or

$x \ mod \ p = 8 \ or \ 9$ so that it will loop back around and still be closer to the right number

Tests
Regev

- The acceptable error range becomes:
  - $0 \leq \sum_{i \in S} e_i \, mod \, p \leq \left\lfloor \frac{\lfloor \frac{p}{2} \rfloor}{2} \right\rfloor$
  - Or
  - $p - \left\lfloor \frac{\lfloor \frac{p}{2} \rfloor}{2} \right\rfloor \leq \sum_{i \in S} e_i \, mod \, p \leq p - 1$

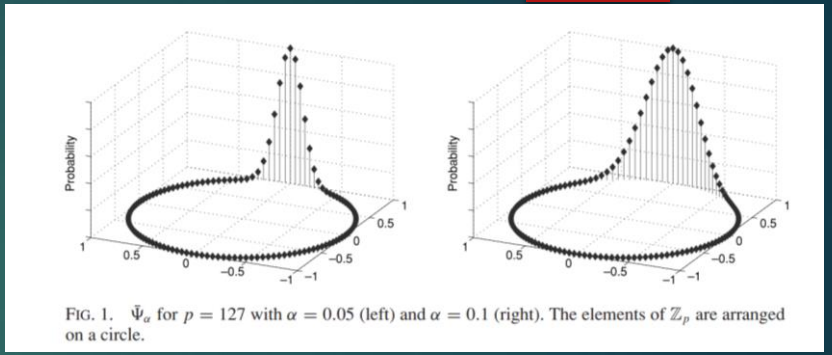


FIG. 1. $\bar{\Psi}_\alpha$ for $p = 127$ with $\alpha = 0.05$ (left) and $\alpha = 0.1$ (right). The elements of $\mathbb{Z}_p$ are arranged on a circle.

Tests
Regev

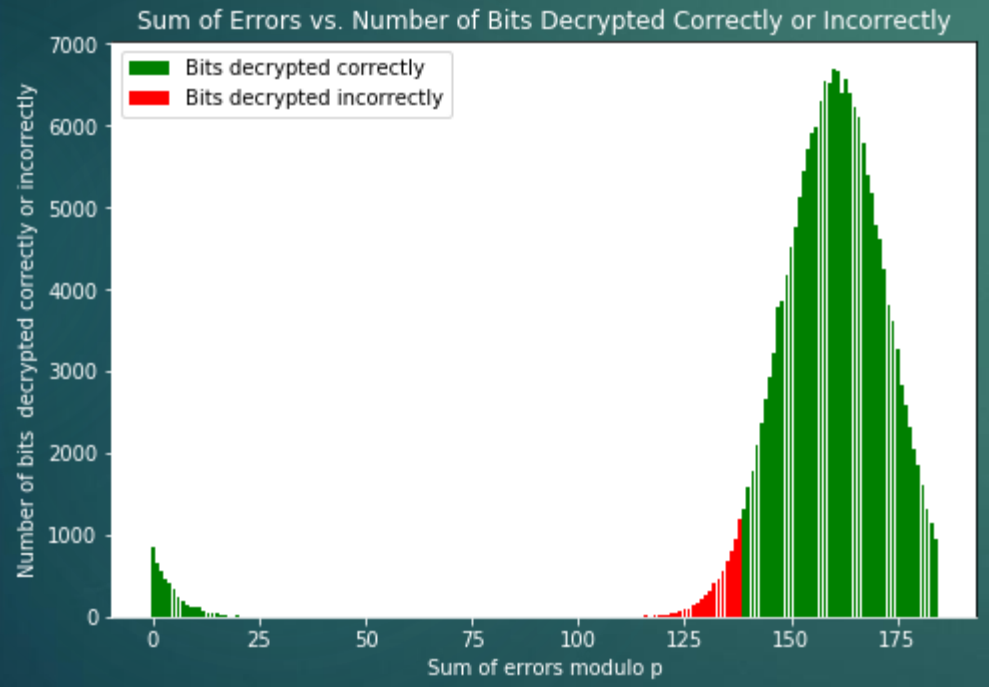
- Informal testing:

```
n = 10
p = 199
m = 126
acceptable error_sum range 1: [0, 49]
acceptable error_sum range 2: [150, 198]
error_sum % p = 5: bit '0' decrypted correctly
error_sum % p = 29: bit '1' decrypted correctly
error_sum % p = 193: bit '0' decrypted correctly
error_sum % p = 13: bit '1' decrypted correctly
error_sum % p = 59: bit '0' decrypted incorrectly
error_sum % p = 20: bit '1' decrypted correctly
error_sum % p = 29: bit '0' decrypted correctly
error_sum % p = 20: bit '1' decrypted correctly
error_sum % p = 15: bit '0' decrypted correctly
error_sum % p = 11: bit '1' decrypted correctly
$ _
```

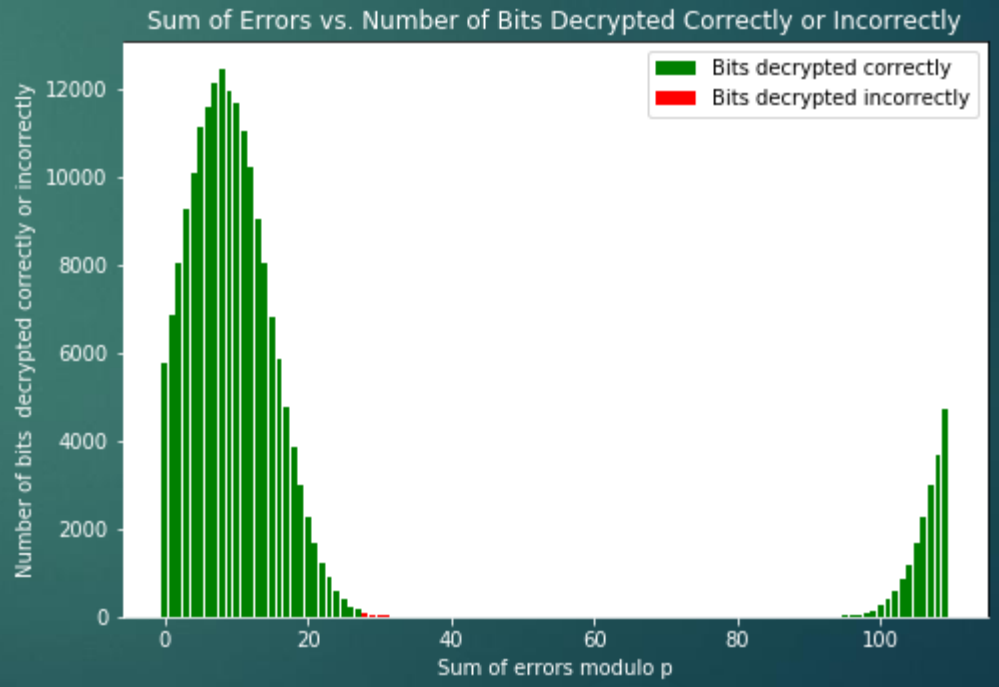
Tests
Regev

- More thorough testing:

n = 10, p = 185, m = 124
acceptable error_sum range 1: [0, 46]
acceptable error_sum range 2: [139, 184]



n = 10, p = 110, m = 112
acceptable error_sum range 1: [0, 27]
acceptable error_sum range 2: [83, 109]



Tests

Regev

- Testing per bit: correct = 99.14%, incorrect = 0.86%

```
p = 123
m = 115
correct = 19260/20000
incorrect = 740/20000
n = 10
p = 173
m = 123
correct = 19999/20000
incorrect = 1/20000
n = 10
p = 187
m = 125
correct = 19998/20000
incorrect = 2/20000
n = 10
p = 190
m = 125
correct = 19975/20000
incorrect = 25/20000
n = 10
p = 193
m = 125
correct = 19908/20000
incorrect = 92/20000
average correct bits: 19828/20000
average incorrect bits: 172/20000
$
```

Experiments

Experiments

- What we hope to accomplish in the experiments
- Choosing an x-axis
- Experiment 1: Work Factor vs. Public Key Size
- Experiment 2: Work Factor vs. Key Generation Speed
- Experiment 3: Work Factor vs. Encryption Speed
- Experiment 4: Work Factor vs. Decryption Speed
- Experiment 5: Work Factor vs. Encrypted Message Size

Experiments

- What we hope to accomplish

Experiments

- Choosing an x-axis

Experiments

- Choosing an x-axis

	Work factor	McEliece $O(n!)$ Attack	McEliece $O(2^n)$ Attack	Regev $O(2^n)$ Attack	Regev $O(2^{nlogn})$ Attack	RSA Attack $O(2^n)$
security parameters:		(n,k)	(n,k)	(n,E)	(n,E)	n
	2^1	(2,2) = (2,2)	(1,2)	(1,0.5)	(2,0.5)	1
	2^2	(2,2) = (2,2)	(2,2)	(2,0.5)	(2,0.5)	2
	2^4	(3,2) = (4,2)	(4,2)	(4,0.5)	(3,0.5)	4
	2^8	(5,2) = (8,2)	(8,2)	(8,0.5)	(4,0.5)	8
	2^{16}	(8,2)	(16,2)	(16,0.5)	(7,0.5)	16
	2^{32}	(13,2) = (16,2)	(32,2)	(32,0.5)	(10,0.5)	32
	2^{64}	(20,2) = (32,2)	(64,2)	(64,0.5)	(16,0.5)	64
	2^{128}	(34,2) = (64,2)	(128,2)	(128,0.5)	(27,0.5)	128
	2^{256}	(57,2) = (64,2)	(256,2)	(256,0.5)	(47,0.5)	256
	2^{512}	(98,2) = (128,2)	(512,2)	(512,0.5)	(81,0.5)	512
	2^{1024}	(171,2) = (256,2)	(1024,2)	(1024,0.5)	(160,0.5)	1024
	2^{2048}	(301,2) = (512, 2)	(2048,2)	(2048,0.5)	(318,0.5)	2048

Experiments
1. Work Factor vs. Public Key Size

- Hypothesis:

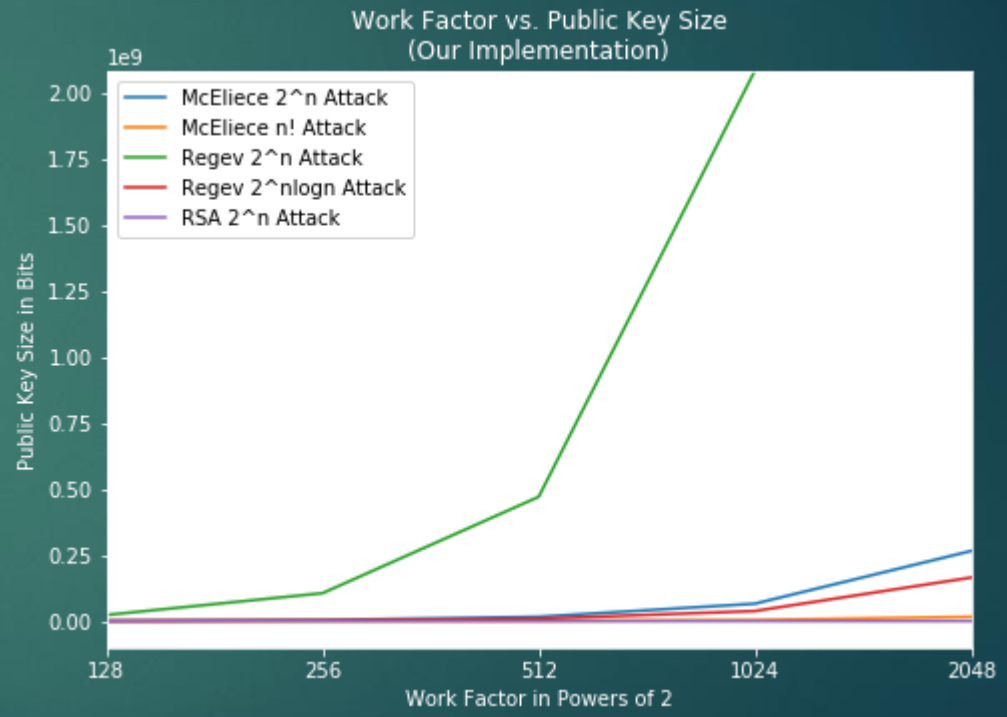
	McEliece	Regev	RSA
public key	$O(n^2)$	$O(mn) = O(n^2 log n^2)$	$O(n)$

Experiments
1. Work Factor vs. Public Key Size

- Results (our implementation):

< 1 Kb

	Work factor	McEliece $O(n!)$ Attack	McEliece $O(2^n)$ Attack	Regev $O(2^n)$ Attack	Regev $O(2^{nlogn})$ Attack	RSA $O(2^n)$ Attack
public key size		bits	bits	bits	bits	bits
	2^1				1728	
	2^2			1472	1728	
	2^4			8128	4800	
	2^8	1216	1216	48320	7872	
	2^{16}	1216	8384	234688	35136	
	2^{32}	8384	45248	1034432	79552	
	2^{64}	45248	213184	4968640	222400	
	2^{128}	213184	934080	23019712	732864	192
	2^{256}	213184	3932352	105545920	2614144	320
	2^{512}	934080	16187584	470614208	8719680	576
	2^{1024}	3932352	65798336	2084700352	37970112	1088
	2^{2048}	16187584	265552064	8989442240	164810688	2112

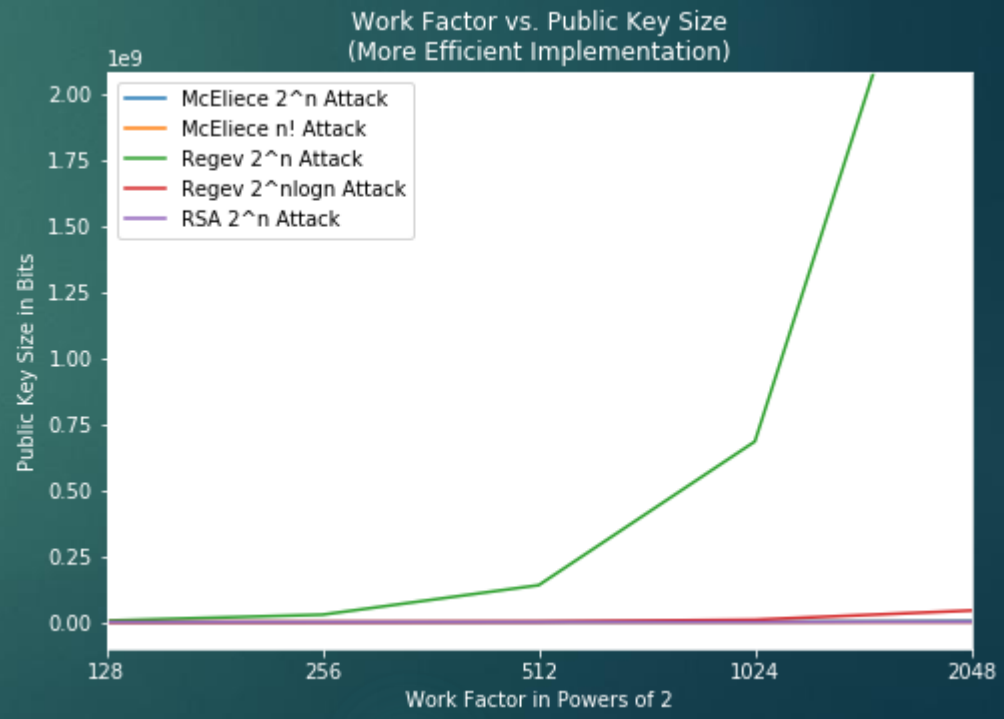


Experiments
1. Work Factor vs. Public Key Size

- Results (more efficient implementation): < 1 Kb

number of bits to store [0,p) = $(\lfloor \log_2(p) \rfloor + 1)$

	Work factor	McEliece $O(n!)$ Attack	McEliece $O(2^n)$ Attack	Regev $O(2^n)$ Attack	Regev $O(2^{nlogn})$ Attack	RSA $O(2^n)$ Attack
public key size		bits	bits	bits	bits	bits
	2^1				24*3+192=264	
	2^2			20 * 3 + 192=252	24*3+192=264	
	2^4			124 * 5 + 192=812	72*4+192=480	
	2^8	208	208	752 *7 + 192=5456	120*5+192=792	
	2^{16}	208	320	3664 * 9 + 192=33168	546*6+192=3468	
	2^{32}	320	896	16160 * 11 + 192=177952	1240*7+192=8872	
	2^{64}	896	3520	77632 * 13 + 192=1009408	3472*9+192=31440	
	2^{128}	3520	14784	359680 * 15 + 192=5395392	11448*10+192=114672	192
	2^{256}	3520	61632	1649152 * 17 + 192=28035317	40843*12+192=490308	320
	2^{512}	14784	253120	7353344 * 19 + 192=139713728	136242*13+192=1771338	576
	2^{1024}	61632	1028288	32573443 * 21 + 192=684042495	593283*15+192=8899437	1088
	2^{2048}	253120	4149440	140460035 * 23 +192=3230580997	2575167*17+192=43778031	2112



Experiments
2. Work Factor vs. Key Generation Speed

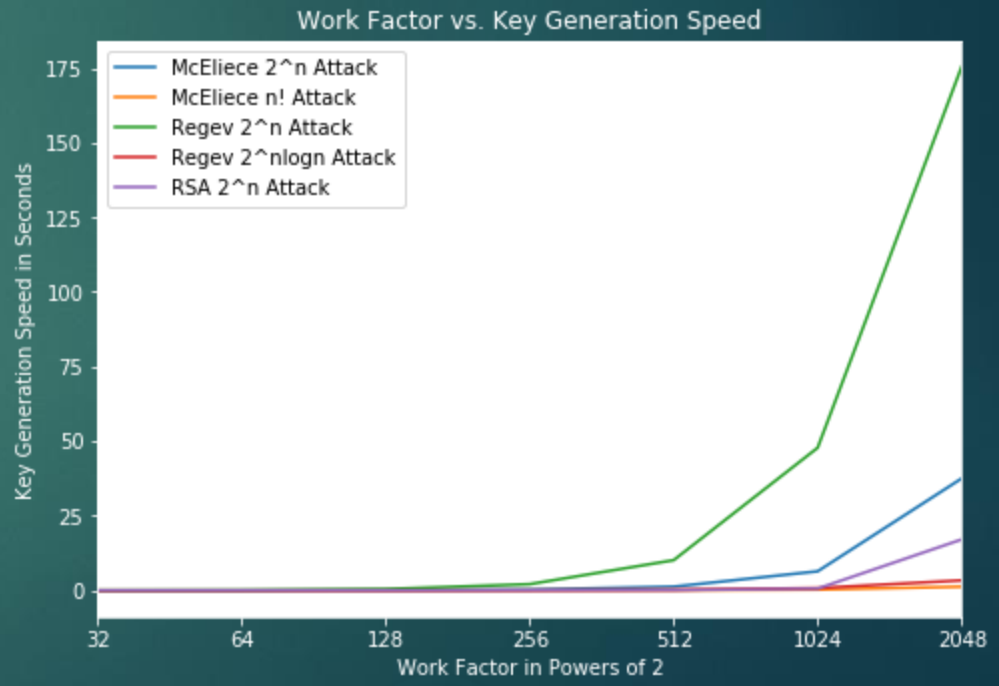
- Hypothesis

	McEliece	Regev	RSA
key generation speed	$O(n^2)$	$O(mn) = O(n^2 log n^2)$	$O(n\text{^}2)$

Experiments
2. Work Factor vs. Key Generation Speed

- Results

	Work factor	McEliece $O(n!)$ Attack	McEliece $O(2^n)$ Attack	Regev $O(2^n)$ Attack	Regev $O(2^{nlogn})$ Attack	RSA Attack $O(2^n)$
key generation time		seconds	seconds	seconds	seconds	seconds
	2^1				.000123	
	2^2			.000112392	.000055	
	2^4			.000212368	.000130	
	2^8	.000784	.000918	.001173041	.000220	
	2^{16}	.000512	.001768	.004583284	.000790	.010810
	2^{32}	.001943	.006540	.020853274	.001781	.003524
	2^{64}	.005974	.020723	.102617203	.004651	.003550
	2^{128}	.020653	.074750	.44960736	.015713	.004828
	2^{256}	.019606	.273176	2.0868353	.054341	.031310
	2^{512}	.075963	1.24842	10.126917	.177354	.103344
	2^{1024}	.274670	6.39750	47.790256	.780439	.602274
	2^{2048}	1.239622	37.5098	175.428124	3.354826	17.069227



Experiments
3. Work Factor vs. Encryption Speed

- Hypothesis

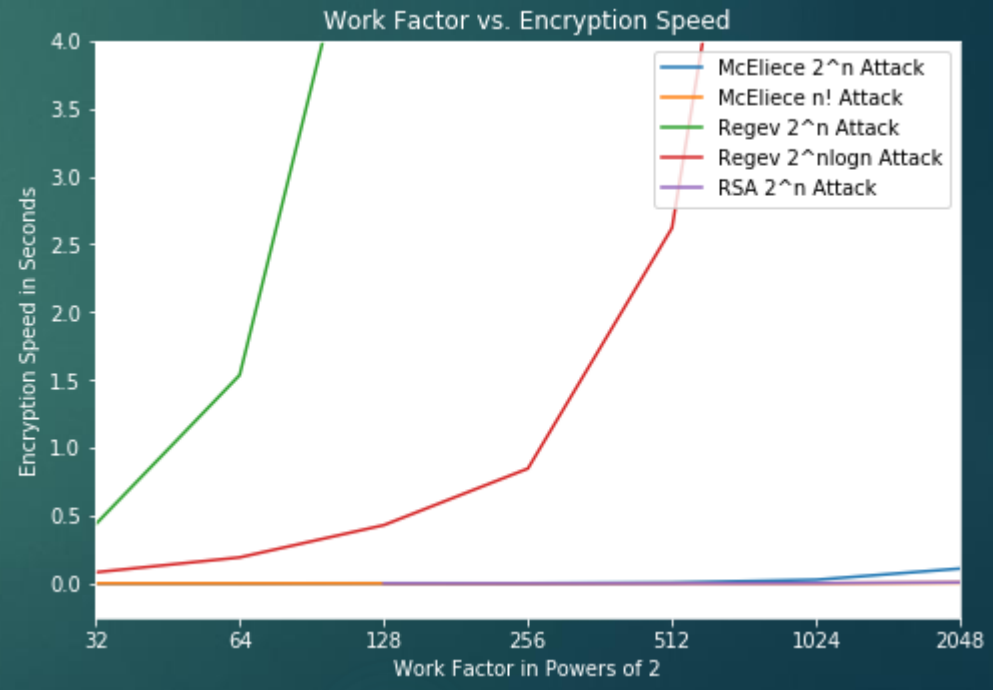
	McEliece	Regev	RSA
encryption speed	$O(n^2)$	$O(mn) = O(n^2 log n^2)$	O(n^2)

Experiments
3. Work Factor vs. Encryption Speed

- Results

inf = greater than 5 minutes

	Work factor	McEliece $O(n!)$ Attack	McEliece $O(2^n)$ Attack	Regev $O(2^n)$ Attack	Regev $O(2^{nlogn})$ Attack	RSA Attack $O(2^n)$
encryption time in seconds		seconds	seconds	seconds	seconds	seconds
	2^1				.003851	
	2^2			.004972	.003950	
	2^4			.012093	.007741	
	2^8	.002640	.002525	.041004	.014580	
	2^{16}	.002546	.001736	.133774	.030920	
	2^{32}	.001144	.001083	.435323	.082704	
	2^{64}	.000904	.001130	1.535871	.192512	
	2^{128}	.001087	.002128	5.8007805	.430259	.001512
	2^{256}	.001027	.001821	24.071204	.847360	.000834
	2^{512}	.002385	.006608	98.704114	2.620565	.000984
	2^{1024}	.001593	.027680	inf	9.082701	.002692
	2^{2048}	.006030	.110428	inf	37.743688	.008710



Experiments
4. Work Factor vs. Decryption Speed

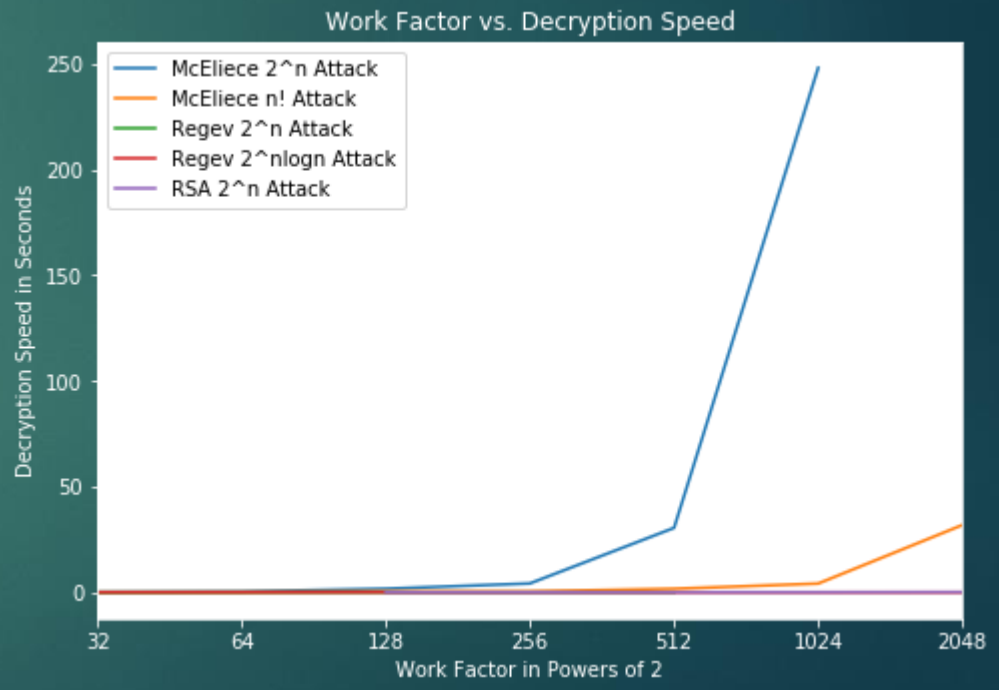
- Hypothesis

	McEliece	Regev	RSA
decryption speed	$O(n^3)$	$O(n)$ per bit	$O(n^3)$

Experiments
4. Work Factor vs. Decryption Speed

- Results

	Work factor	McEliece $O(n!)$ Attack	McEliece $O(2^n)$ Attack	Regev $O(2^n)$ Attack	Regev $O(2^{nlogn})$ Attack	RSA Attack $O(2^n)$
decryption time in seconds		seconds	seconds	seconds	seconds	seconds
	2^1				.000217	
	2^2			.000231	.000280	
	2^4			.000255	.000242	
	2^8	.057649	.058582	.000326	.000256	
	2^{16}	.062007	.072849	.000500	.000324	
	2^{32}	.068922	.125848	.000780	.000395	
	2^{64}	.141895	.387596	.001371	.000515	
	2^{128}	.388524	1.656945	.002543	.000643	.002256
	2^{256}	.388930	4.169249	.004916	.001085	.002033
	2^{512}	1.643109	30.439314	.009317	.001760	.003246
	2^{1024}	4.112714	247.936048	?	.003120	.014033
	2^{2048}	31.787255	inf	?	.005721	.083484



Experiment
5. Work Factor vs. Encrypted Message Size

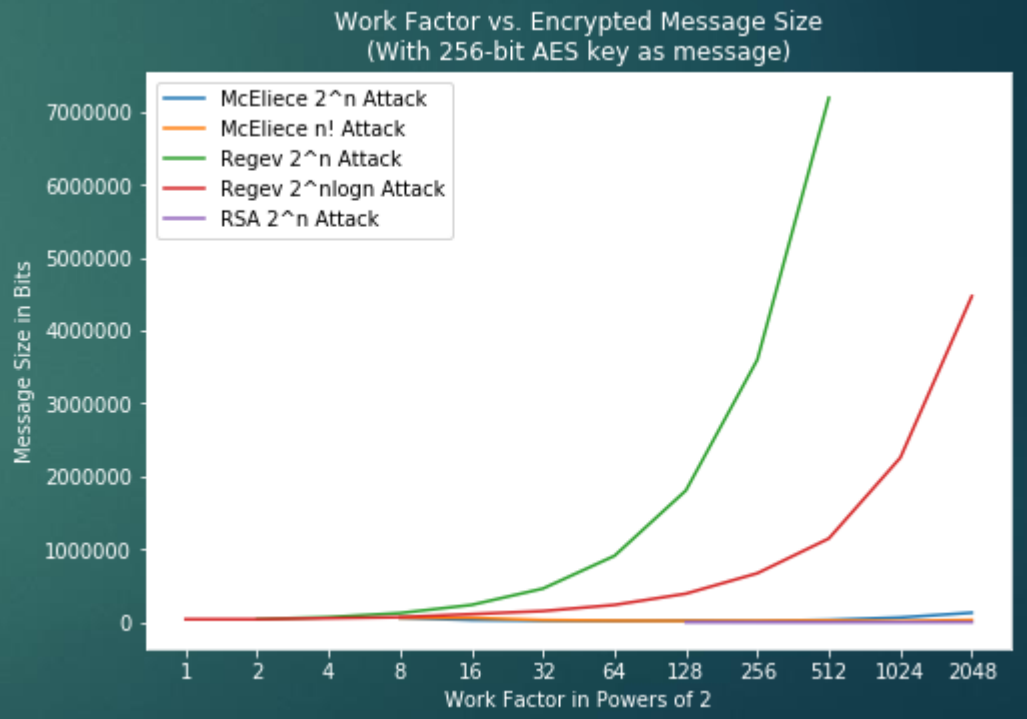
- Hypothesis

	McEliece	Regev	RSA
encrypted message length	(n-k)/k +1 bits per bit = O(n)	n+1 bits per bit = O(n)	O(n) bits total

Experiment
5. Work Factor vs. Encrypted Message Size

- Results: our implementation

	Work factor	McEliece $O(n!)$ Attack	McEliece $O(2^n)$ Attack	Regev $O(2^n)$ Attack	Regev $O(2^{nlogn})$ Attack	RSA Attack $O(2^n)$
# bits		bits	bits	bits	bits	bits
	2^1				42048	
	2^2			42048	42048	
	2^4			70080	56064	
	2^8	61440	61440	126144	70080	
	2^{16}	61440	30720	238272	112128	
	2^{32}	30720	22528	462528	154176	
	2^{64}	22528	20480	911040	238272	
	2^{128}	20480	24576	1808064	392448	768
	2^{256}	20480	16384	3602112	672768	512
	2^{512}	24576	32768	7190208	1149312	512
	2^{1024}	16384	65536	?	2256576	1024
	2^{2048}	32768	131072	?	4471104	2048

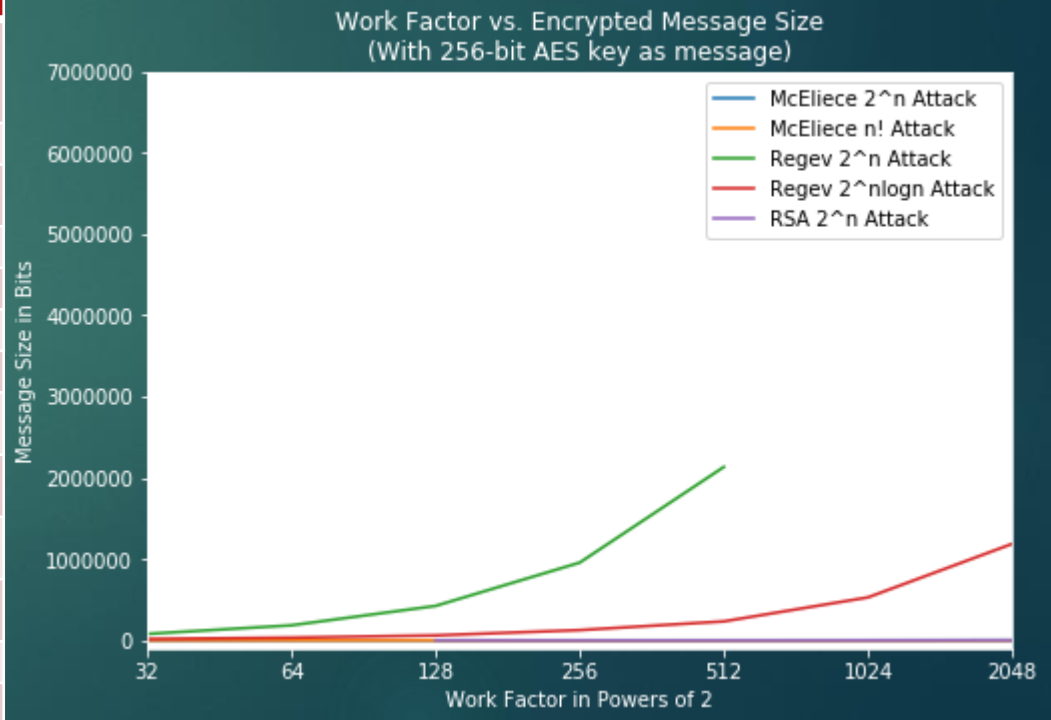


Experiment
5. Work Factor vs. Encrypted Message Size

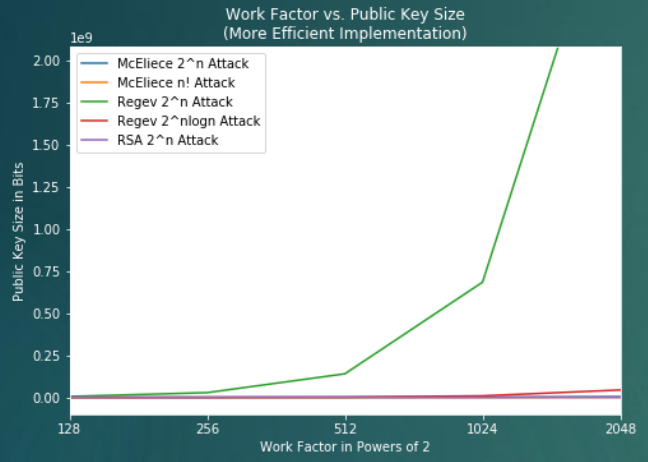
- Results: more efficient implementation

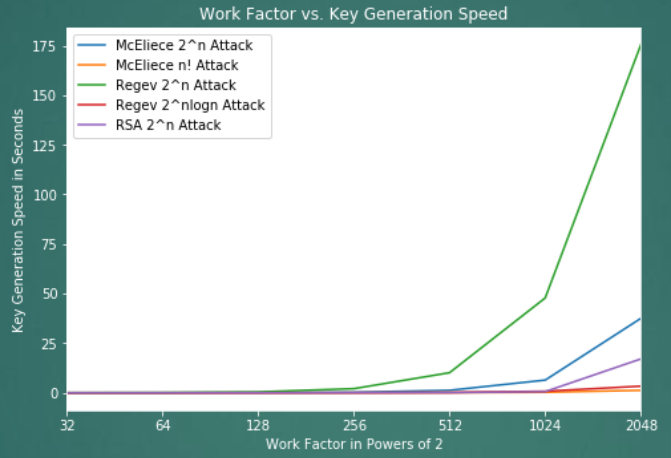
number of bits to store [0,p) = $(\lfloor \log_2(p) \rfloor + 1)$

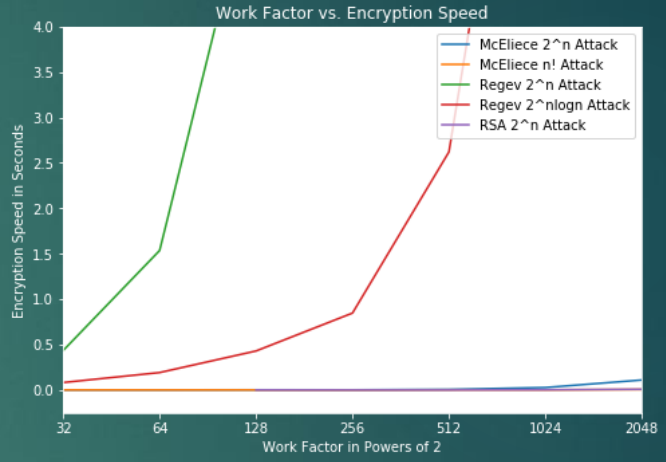
	Work factor	McEliece $O(n!)$ Attack	McEliece $O(2^n)$ Attack	Regev $O(2^n)$ Attack	Regev $O(2^{nlogn})$ Attack	RSA Attack $O(2^n)$
		bits	bits	bits	bits	bits
	2^1				657*3=1971	
	2^2			657 * 3=1971	657*3=1971	
	2^4			1095 * 5=5475	876*4=3504	
	2^8	960	960	1971 * 7=13797	1095*5=5475	
	2^{16}	960	480	3723 * 9=33507	1752*6=10512	
	2^{32}	480	352	7227 * 11=79497	2409*7=16863	
	2^{64}	352	320	14235 * 13=185055	3723*9=33507	
	2^{128}	320	384	28251 * 15=423765	6132*10=61320	768
	2^{256}	320	256	56283 * 17=956811	10512*12=126144	512
	2^{512}	384	512	112347 * 19=2134593	17958*13=233454	512
	2^{1024}	256	1024	?	35259*15=528885	1024
	2^{2048}	512	2048	?	69861*17=1187637	2048

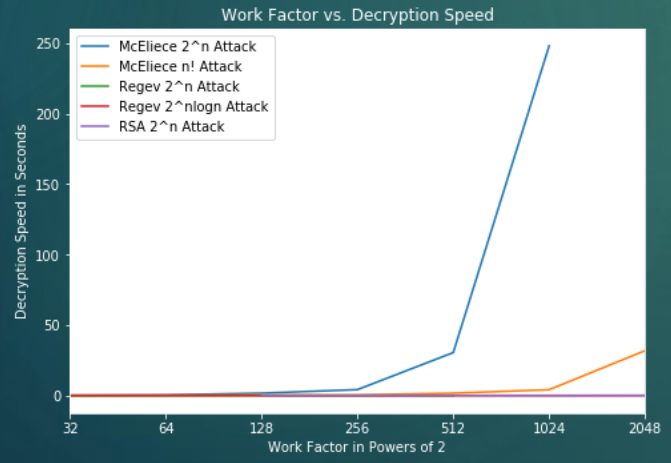


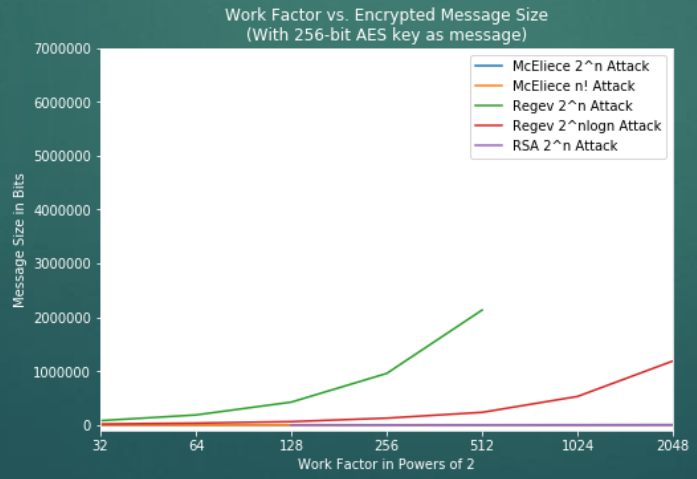
Experiments Results











Challenges

- McEliece and Regev both difficult to understand
- McEliece difficult to implement
- Did not finish in time to put everything in report

Conclusion

- In this project we used a unique programming language to implement two candidate public key systems for post-quantum cryptography
- We gained an understanding of lattice-based systems and code-based systems
- We showed the cost of replacing pre-quantum systems with post-quantum systems

References

[1] McEliece, R. J. "A public-key system based on algebraic coding theory, 114-116. deep sace network progress report, 44." *Jet Propulsion Laboratory, California Institute of Technology* (1978).

[2] Regev, Oded. "On lattices, learning with errors, random linear codes, and cryptography." *Journal of the ACM (JACM)* 56, no. 6 (2009): 1-40.

[3] Hankerson, Darrel, Alfred J. Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[4] Gao, Shuhong, and Daniel Panario. "Tests and constructions of irreducible polynomials over finite fields." In *Foundations of computational mathematics*, pp. 346-361. Springer, Berlin, Heidelberg, 1997.

[5] https://stackoverflow.com/questions/6192825/c-calculating-the-distance-between-2-floats-modulo-12