A Brief Intro to Lattice-based Cryptography

Jackson Walters

January 13, 2025

History of cryptography

Shor's algorithm

Hard Problems on Lattices

- Connection to Ring-LWE

ring-LWE

Implementation of ring-LWE in Rust

Outline

- A brief tour of cryptography from a tourist

Outline

- A brief tour of cryptography from a tourist
- An overview of Shor's algorithm

Outline

- A brief tour of cryptography from a tourist
- An overview of Shor's algorithm
- Relevant hard problems on lattices

Outline

- A brief tour of cryptography from a tourist
- An overview of Shor's algorithm
- Relevant hard problems on lattices
- Implementation of ring-LWE in Rust

Outline

- A brief tour of cryptography from a tourist
- An overview of Shor's algorithm
- Relevant hard problems on lattices
- Implementation of ring-LWE in Rust
- Shout out to module-LWE, FIPS 203

History of cryptography

- People have been trying to conceal transmitted information for thousands of years

History of cryptography

- People have been trying to conceal transmitted information for thousands of years
- Three phases: manual (pre-computer), mechanical (switches, tapes, relays), digital/electronic

History of cryptography

- People have been trying to conceal transmitted information for thousands of years
- Three phases: manual (pre-computer), mechanical (switches, tapes, relays), digital/electronic
- First recorded use was a scytale formed by wrapping a leather strap around a baton upon which words were written

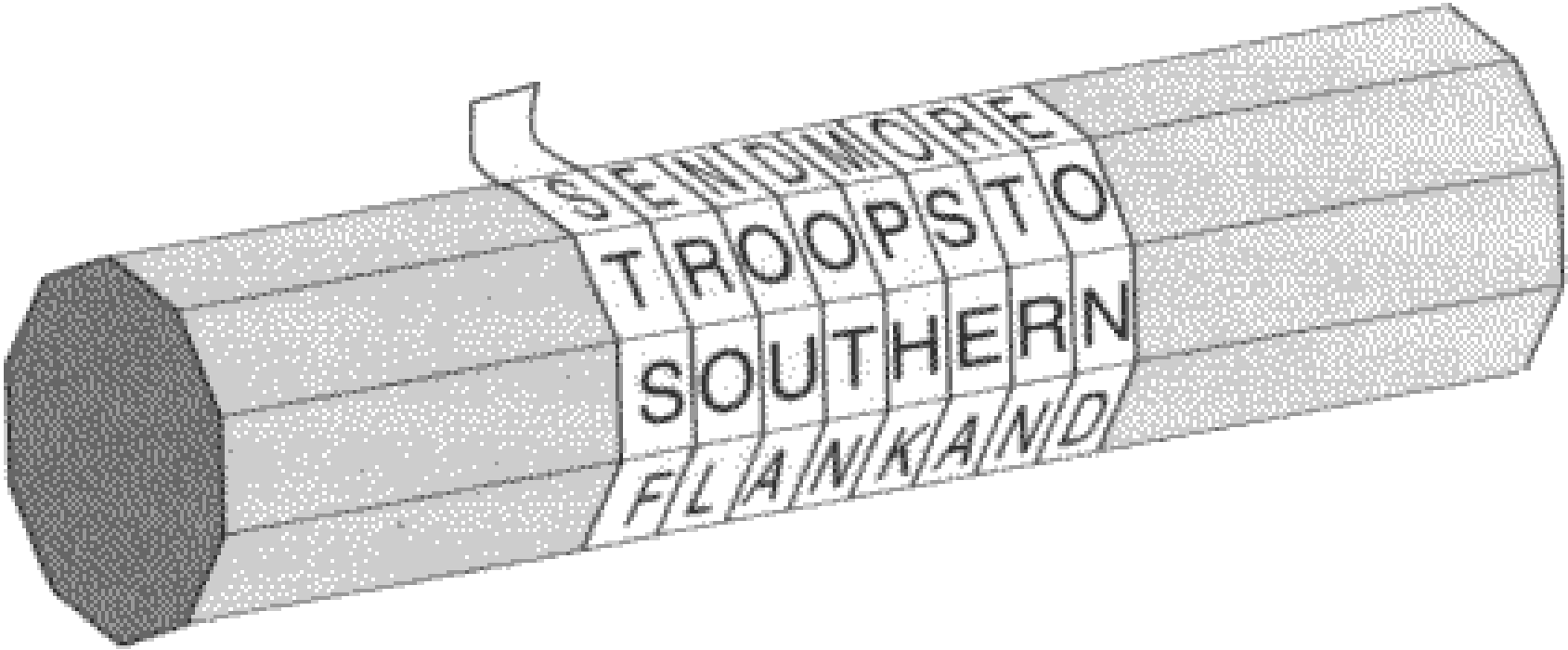
History of cryptography

- People have been trying to conceal transmitted information for thousands of years
- Three phases: manual (pre-computer), mechanical (switches, tapes, relays), digital/electronic
- First recorded use was a scytale formed by wrapping a leather strap around a baton upon which words were written
- When unwrapped, the letters appeared scrambled

History of cryptography

- People have been trying to conceal transmitted information for thousands of years
- Three phases: manual (pre-computer), mechanical (switches, tapes, relays), digital/electronic
- First recorded use was a scytale formed by wrapping a leather strap around a baton upon which words were written
- When unwrapped, the letters appeared scrambled
- This is an example of a transposition cipher. Letters are moved a fixed distance from their original position to form the "ciphertext"

History of cryptography: scytale



History of cryptography

- A shift by three units is known as a Caesar cipher.

History of cryptography

- A shift by three units is known as a Caesar cipher.
- Could index letters X by elements of $\mathbb{Z}/N\mathbb{Z}$, so then $X_i \mapsto X_{i+3}$.

History of cryptography

- A shift by three units is known as a Caesar cipher.
- Could index letters X by elements of $\mathbb{Z}/N\mathbb{Z}$, so then $X_i \mapsto X_{i+3}$.
- Could also do $X_i \mapsto X_{\sigma(i)}$ for permutation σ (permutation cipher)

History of cryptography

- A shift by three units is known as a Caesar cipher.
- Could index letters X by elements of $\mathbb{Z}/N\mathbb{Z}$, so then $X_i \mapsto X_{i+3}$.
- Could also do $X_i \mapsto X_{\sigma(i)}$ for permutation σ (permutation cipher)
- A variation is the Vigenère cipher, in which a key is repeated many times, and the letter in the key determines how many positions to shift the letter.

History of cryptography: Vigenère cipher

```
Plaintext:    attackatdawn
Key:          LEMONLEMONLE
Ciphertext:   LXFOPVEFRNHR
```

History of cryptography

- $C_i = X_i + K_i \mod 26$, where we view K_i (key letters) and X_i (plaintext) as integers representing one of twenty-six letters. These are known as polyalphabetic ciphers.

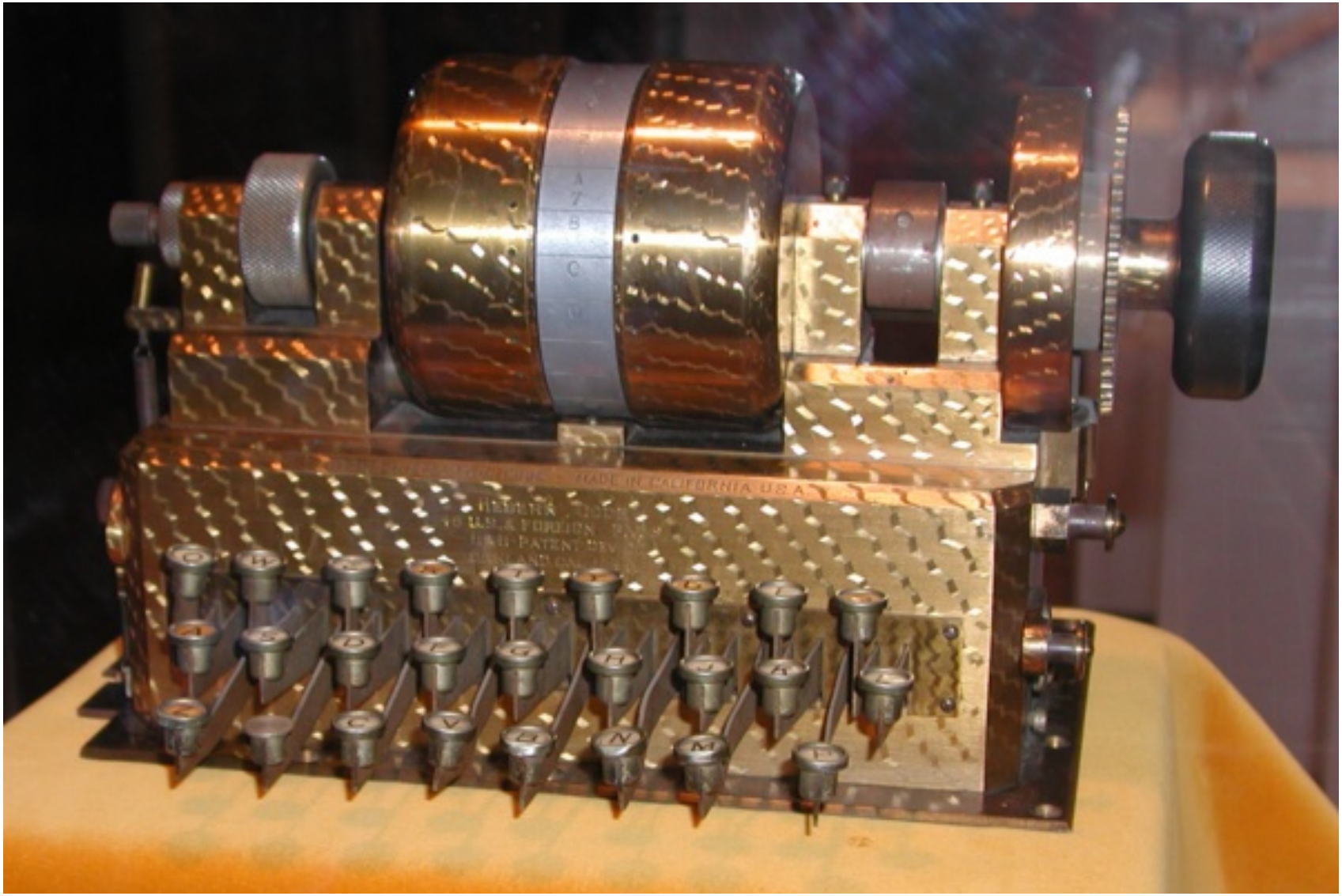
History of cryptography

- $C_i = X_i + K_i \mod 26$, where we view K_i (key letters) and X_i (plaintext) as integers representing one of twenty-six letters. These are known as polyalphabetic ciphers.
- The Vigenère cipher was considered to be unbreakable by many through the early twentieth century. This was not the case. The Confederate army used this cipher and it was often broken. They relied on keyphrases such as "Manchester Bluff", "Complete Victory" and, as the war came to a close, "Come Retribution".

History of cryptography

- $C_i = X_i + K_i \mod 26$, where we view K_i (key letters) and X_i (plaintext) as integers representing one of twenty-six letters. These are known as polyalphabetic ciphers.
- The Vigenère cipher was considered to be unbreakable by many through the early twentieth century. This was not the case. The Confederate army used this cipher and it was often broken. They relied on keyphrases such as "Manchester Bluff", "Complete Victory" and, as the war came to a close, "Come Retribution".
- Around 1917, it was realized by Edward H. Hebern that these monoalphabetic ciphers could be chained together using a system of rotors by hardwiring one to the next. A set or stack of these rotors was put together, and as one rotates the other rotates by one tenth or so of the previous one. There is some dispute as to whether the Dutch Navy first invented this system a couple years before.

History of cryptography: Hebern rotor machine



History of cryptography: Hagelin design M-209

- These poly-alphabetic rotor ciphers would form the precursor to machines like the Enigma machine used in World War II.

History of cryptography: Hagelin design M-209

- These poly-alphabetic rotor ciphers would form the precursor to machines like the Enigma machine used in World War II.
- The Hagelin design M-209 U.S. cipher machine was used for tactical communications during World War II.

History of cryptography: Hagelin design M-209

- These poly-alphabetic rotor ciphers would form the precursor to machines like the Enigma machine used in World War II.
- The Hagelin design M-209 U.S. cipher machine was used for tactical communications during World War II.
- Such devices are pin-and-lug machines, and they typically consist of a collection of rotors having a prime number of labeled positions on each rotor.

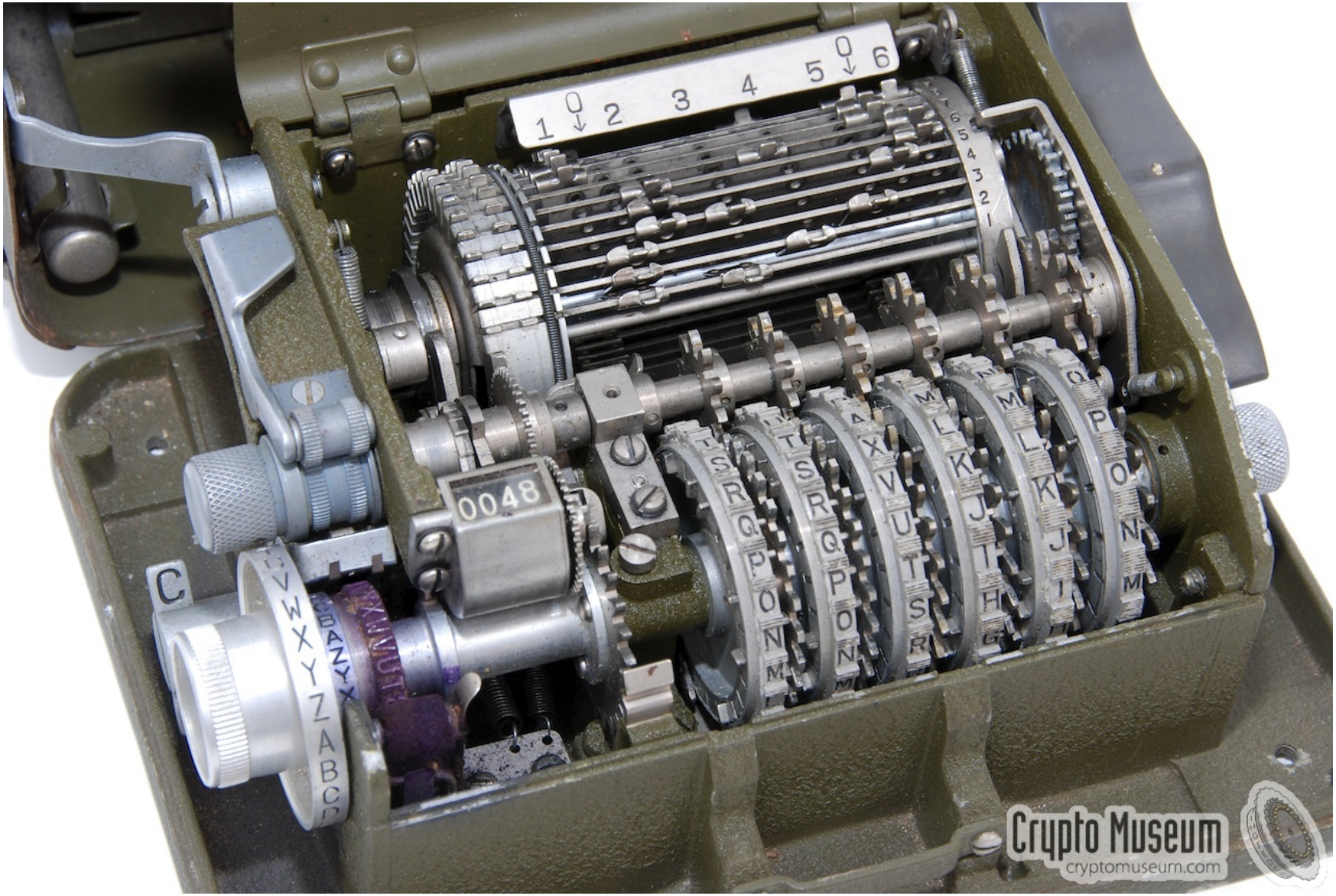
History of cryptography: Hagelin design M-209

- These poly-alphabetic rotor ciphers would form the precursor to machines like the Enigma machine used in World War II.
- The Hagelin design M-209 U.S. cipher machine was used for tactical communications during World War II.
- Such devices are pin-and-lug machines, and they typically consist of a collection of rotors having a prime number of labeled positions on each rotor.
- At each position a small pin can be set to an active or inactive position. In operation, all of the rotors advance one position at each step.

History of cryptography: Hagelin design M-209

- These poly-alphabetic rotor ciphers would form the precursor to machines like the Enigma machine used in World War II.
- The Hagelin design M-209 U.S. cipher machine was used for tactical communications during World War II.
- Such devices are pin-and-lug machines, and they typically consist of a collection of rotors having a prime number of labeled positions on each rotor.
- At each position a small pin can be set to an active or inactive position. In operation, all of the rotors advance one position at each step.
- Therefore, if the active pin settings are chosen appropriately, the machine will not recycle to its initial pin configuration until it has been advanced a number of steps equal to the product of the number of positions in each one of the rotors.

History of cryptography: Hagelin design M-209



History of cryptography: DES $\rightarrow$ AES

- After World War II, the digital computer became operational and was applied to crypto-machines. They were much faster than their mechanical counterparts.

History of cryptography: DES $\rightarrow$ AES

- After World War II, the digital computer became operational and was applied to crypto-machines. They were much faster than their mechanical counterparts.
- In 1973, NBS (National Bureau of Standards, now NIST) issued a request for proposals for a new cyptographic standard. No viable subfissions were received. A second request was made, which led to the DES (data encryption standard).

History of cryptography: DES $\rightarrow$ AES

- After World War II, the digital computer became operational and was applied to crypto-machines. They were much faster than their mechanical counterparts.
- In 1973, NBS (National Bureau of Standards, now NIST) issued a request for proposals for a new cyptographic standard. No viable subfissions were received. A second request was made, which led to the DES (data encryption standard).
- The DES is a product block cipher in which 16 iterations, or rounds, of substitution and transposition (permutation) process are cascaded. The block size is 64 bits. The key, which controls the transformation, also consists of 64 bits; however, only 56 of these can be chosen by the user and are actually key bits.

History of cryptography: DES $\rightarrow$ AES

- After World War II, the digital computer became operational and was applied to crypto-machines. They were much faster than their mechanical counterparts.
- In 1973, NBS (National Bureau of Standards, now NIST) issued a request for proposals for a new cyptographic standard. No viable subfissions were received. A second request was made, which led to the DES (data encryption standard).
- The DES is a product block cipher in which 16 iterations, or rounds, of substitution and transposition (permutation) process are cascaded. The block size is 64 bits. The key, which controls the transformation, also consists of 64 bits; however, only 56 of these can be chosen by the user and are actually key bits.
- The security of the DES is no greater than its work factor—the brute-force effort required to search 256 keys. That is a search for a needle in a haystack of 72 quadrillion straws. In 1977 that was considered an impossible computational task. In 1999 a special-purpose DES search

History of cryptography: DES $\rightarrow$ AES

- After World War II, the digital computer became operational and was applied to crypto-machines. They were much faster than their mechanical counterparts.
- In 1973, NBS (National Bureau of Standards, now NIST) issued a request for proposals for a new cyptographic standard. No viable subfissions were received. A second request was made, which led to the DES (data encryption standard).
- The DES is a product block cipher in which 16 iterations, or rounds, of substitution and transposition (permutation) process are cascaded. The block size is 64 bits. The key, which controls the transformation, also consists of 64 bits; however, only 56 of these can be chosen by the user and are actually key bits.
- The security of the DES is no greater than its work factor—the brute-force effort required to search 256 keys. That is a search for a needle in a haystack of 72 quadrillion straws. In 1977 that was considered an impossible computational task. In 1999 a special-purpose DES search

History of cryptography: AES

- AES (Advanced Encryption Standard) is a block cipher and a type of symmetric encryption algorithm, meaning the same key is used for both encryption and decryption. It transforms fixed-size blocks of plaintext (128 bits) into ciphertext using a secret key, which can be 128, 192, or 256 bits. AES works by applying a series of transformations called rounds to the data. Each round involves substituting bytes, shuffling rows, mixing columns, and combining the data with a version of the key using XOR operations. The number of rounds depends on the key size: 10 for 128-bit keys, 12 for 192-bit keys, and 14 for 256-bit keys.

History of cryptography: AES

- AES (Advanced Encryption Standard) is a block cipher and a type of symmetric encryption algorithm, meaning the same key is used for both encryption and decryption. It transforms fixed-size blocks of plaintext (128 bits) into ciphertext using a secret key, which can be 128, 192, or 256 bits. AES works by applying a series of transformations called rounds to the data. Each round involves substituting bytes, shuffling rows, mixing columns, and combining the data with a version of the key using XOR operations. The number of rounds depends on the key size: 10 for 128-bit keys, 12 for 192-bit keys, and 14 for 256-bit keys.
- The strength of AES comes from its use of multiple rounds that make the ciphertext appear random and secure against attacks. Even a small change in the plaintext or key results in completely different ciphertext due to the scrambling process. Decryption reverses these steps using the same key. AES is fast, efficient, and widely adopted for securing data in applications like VPNs, disk encryption, and secure

RSA basics

- RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman

RSA basics

- RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman
- RSA is a public-key cryptographic algorithm that relies on the mathematical properties of large prime numbers
- It is asymmetric, i.e. it generates two keys: a public key for encryption and a private key for decryption

RSA basics

- RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman
- RSA is a public-key cryptographic algorithm that relies on the mathematical properties of large prime numbers
- It is asymmetric, i.e. it generates two keys: a public key for encryption and a private key for decryption
- The core idea of RSA is based on the difficulty of factoring the product of two large prime numbers, a problem that is computationally infeasible for sufficiently large primes

RSA basics

- RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman
- RSA is a public-key cryptographic algorithm that relies on the mathematical properties of large prime numbers
- It is asymmetric, i.e. it generates two keys: a public key for encryption and a private key for decryption
- The core idea of RSA is based on the difficulty of factoring the product of two large prime numbers, a problem that is computationally infeasible for sufficiently large primes
- RSA encrypts data by raising it to a power (the public key exponent) modulo a large number (the product of the two primes)

RSA basics

- RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman
- RSA is a public-key cryptographic algorithm that relies on the mathematical properties of large prime numbers
- It is asymmetric, i.e. it generates two keys: a public key for encryption and a private key for decryption
- The core idea of RSA is based on the difficulty of factoring the product of two large prime numbers, a problem that is computationally infeasible for sufficiently large primes
- RSA encrypts data by raising it to a power (the public key exponent) modulo a large number (the product of the two primes)
- decryption reverses this operation using the private key exponent.

RSA basics: multiplicative homomorphism

- One interesting property of RSA is that it is multiplicatively homomorphic.

RSA basics: multiplicative homomorphism

- One interesting property of RSA is that it is multiplicatively homomorphic.
- This means that if you multiply two ciphertexts, the result corresponds to the encryption of the product of the two plaintexts.

RSA basics: multiplicative homomorphism

- One interesting property of RSA is that it is multiplicatively homomorphic.
- This means that if you multiply two ciphertexts, the result corresponds to the encryption of the product of the two plaintexts.
- Mathematically,

$$C_1 = M_1^e \mod N$$

$$C_2 = M_2^e \mod N$$

$$C_1 \cdot C_2 \mod N = M_1^e \cdot M_2^e = (M_1 \cdot M_2)^e \mod N$$

RSA basics: multiplicative homomorphism

- One interesting property of RSA is that it is multiplicatively homomorphic.
- This means that if you multiply two ciphertexts, the result corresponds to the encryption of the product of the two plaintexts.
- Mathematically,

$$C_1 = M_1^e \mod N$$

$$C_2 = M_2^e \mod N$$

$$C_1 \cdot C_2 \mod N = M_1^e \cdot M_2^e = (M_1 \cdot M_2)^e \mod N$$

- It's homomorphic, so we're done with the talk.

RSA basics: multiplicative homomorphism

- One interesting property of RSA is that it is multiplicatively homomorphic.
- This means that if you multiply two ciphertexts, the result corresponds to the encryption of the product of the two plaintexts.
- Mathematically,

$$C_1 = M_1^e \mod N$$

$$C_2 = M_2^e \mod N$$

$$C_1 \cdot C_2 \mod N = M_1^e \cdot M_2^e = (M_1 \cdot M_2)^e \mod N$$

- It's homomorphic, so we're done with the talk.
- jk. it's not additively homomorphic!

Elliptic curve cryptography (ECC)

- Elliptic Curve Cryptography (ECC) is a type of public-key cryptography based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), which is considered much harder to solve than the integer factorization problem RSA relies on.

Elliptic curve cryptography (ECC)

- Elliptic Curve Cryptography (ECC) is a type of public-key cryptography based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), which is considered much harder to solve than the integer factorization problem RSA relies on.
- The ECDLP involves finding an integer k, given two points P and Q on an elliptic curve such that $Q = kP$.

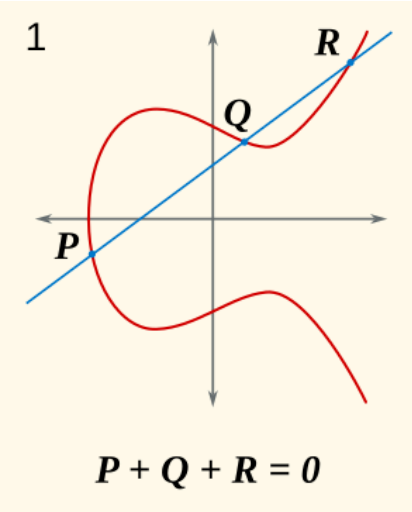
Elliptic curve cryptography (ECC)

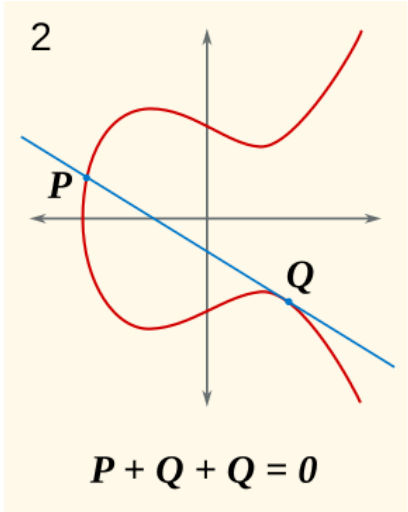
- Elliptic Curve Cryptography (ECC) is a type of public-key cryptography based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), which is considered much harder to solve than the integer factorization problem RSA relies on.
- The ECDLP involves finding an integer k, given two points P and Q on an elliptic curve such that $Q = kP$.
- The operations on elliptic curves, like point addition and scalar multiplication, are defined over finite fields and exhibit complex algebraic properties that make reversing Q back to k computationally infeasible for large curves.

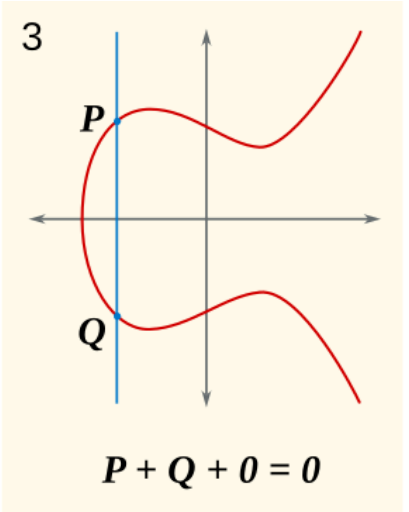
Elliptic curve cryptography (ECC)

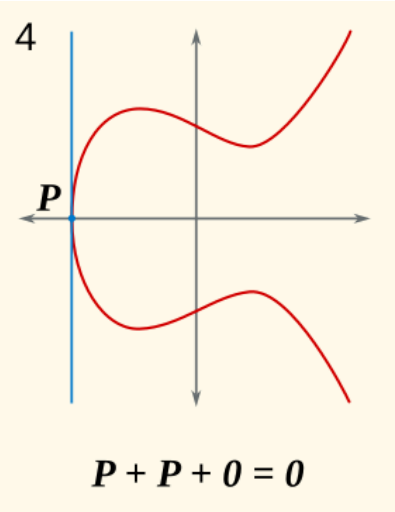
- Elliptic Curve Cryptography (ECC) is a type of public-key cryptography based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), which is considered much harder to solve than the integer factorization problem RSA relies on.
- The ECDLP involves finding an integer k, given two points P and Q on an elliptic curve such that $Q = kP$.
- The operations on elliptic curves, like point addition and scalar multiplication, are defined over finite fields and exhibit complex algebraic properties that make reversing Q back to k computationally infeasible for large curves.
- This difficulty provides the foundation for ECC's security.

ECC: how to add points on an elliptic curve









Discrete logarithm: connection between RSA and ECC

- Interestingly, RSA can also be described in terms of a discrete logarithm problem.

Discrete logarithm: connection between RSA and ECC

- Interestingly, RSA can also be described in terms of a discrete logarithm problem.
- In RSA, the encryption of a plaintext M is performed by computing $C = M^e \mod N$ where e is the public exponent and N is the modulus.

Discrete logarithm: connection between RSA and ECC

- Interestingly, RSA can also be described in terms of a discrete logarithm problem.
- In RSA, the encryption of a plaintext M is performed by computing $C = M^e \mod N$ where e is the public exponent and N is the modulus.
- Recovering M from C and e involves solving for the discrete logarithm of C with base M modulo N, given by $C = M^e \mod N$.

Discrete logarithm: connection between RSA and ECC

- Interestingly, RSA can also be described in terms of a discrete logarithm problem.
- In RSA, the encryption of a plaintext M is performed by computing $C = M^e \mod N$ where e is the public exponent and N is the modulus.
- Recovering M from C and e involves solving for the discrete logarithm of C with base M modulo N, given by $C = M^e \mod N$.
- However, RSA avoids framing its security directly in terms of this discrete log problem because its underlying strength is tied to the difficulty of factoring N into its prime components, not directly to solving discrete logs.

Discrete logarithm: connection between RSA and ECC

- Interestingly, RSA can also be described in terms of a discrete logarithm problem.
- In RSA, the encryption of a plaintext M is performed by computing $C = M^e \mod N$ where e is the public exponent and N is the modulus.
- Recovering M from C and e involves solving for the discrete logarithm of C with base M modulo N, given by $C = M^e \mod N$.
- However, RSA avoids framing its security directly in terms of this discrete log problem because its underlying strength is tied to the difficulty of factoring N into its prime components, not directly to solving discrete logs.
- In contrast, ECC's security is inherently tied to the hardness of solving discrete logarithms, making it a more direct analog to cryptographic schemes based on discrete log problems.

Discrete logarithm: connection between RSA and ECC

- Interestingly, RSA can also be described in terms of a discrete logarithm problem.
- In RSA, the encryption of a plaintext M is performed by computing $C = M^e \mod N$ where e is the public exponent and N is the modulus.
- Recovering M from C and e involves solving for the discrete logarithm of C with base M modulo N, given by $C = M^e \mod N$.
- However, RSA avoids framing its security directly in terms of this discrete log problem because its underlying strength is tied to the difficulty of factoring N into its prime components, not directly to solving discrete logs.
- In contrast, ECC's security is inherently tied to the hardness of solving discrete logarithms, making it a more direct analog to cryptographic schemes based on discrete log problems.
- This results in smaller key sizes for ECC as compared to RSA

Shor's algorithm to break RSA

- In order to decrypt ciphertext $C = M^e \mod N$ quickly, we need to find the modular inverse of e, where $N = p \cdot q$ is a large semiprime.

Shor's algorithm to break RSA

- In order to decrypt ciphertext $C = M^e \mod N$ quickly, we need to find the modular inverse of e, where $N = p \cdot q$ is a large semiprime.
- We are working in the group $(\mathbb{Z}/N\mathbb{Z})^\times$, the multiplicative group of integers modulo N.

Shor's algorithm to break RSA

- In order to decrypt ciphertext $C = M^e \mod N$ quickly, we need to find the modular inverse of e, where $N = p \cdot q$ is a large semiprime.
- We are working in the group $(\mathbb{Z}/N\mathbb{Z})^\times$, the multiplicative group of integers modulo N.
- Note that the order of this group is given by the number of elements coprime to N, which is just $\varphi(N) = (q-1)(p-1)$.

Shor's algorithm to break RSA

- In order to decrypt ciphertext $C = M^e \mod N$ quickly, we need to find the modular inverse of e, where $N = p \cdot q$ is a large semiprime.
- We are working in the group $(\mathbb{Z}/N\mathbb{Z})^\times$, the multiplicative group of integers modulo N.
- Note that the order of this group is given by the number of elements coprime to N, which is just $\varphi(N) = (q-1)(p-1)$.
- Once $\varphi(N)$ is known, one can use the extended Euclidean algorithm to compute the modular inverse d of e, i.e. $d \cdot e \equiv 1 \mod N$. Then $C^d \equiv M^{de} \equiv M^1 \equiv M \mod N$.

Shor's algorithm: period to integer factors

- Shor's algorithm is a quantum algorithm which can be used to factor integers, and hence break RSA given enough qubits.
- The essential component of Shor's algorithm is finding the multiplicative order of an integer modulo N.

Shor's algorithm: period to integer factors

- Shor's algorithm is a quantum algorithm which can be used to factor integers, and hence break RSA given enough qubits.
- The essential component of Shor's algorithm is finding the multiplicative order of an integer modulo N.
- For if we have such an a such that $a^r \equiv 1 \mod N$, then we can write $a^r - 1 \equiv 0 \mod N$.

Shor's algorithm: period to integer factors

- Shor's algorithm is a quantum algorithm which can be used to factor integers, and hence break RSA given enough qubits.
- The essential component of Shor's algorithm is finding the multiplicative order of an integer modulo N.
- For if we have such an a such that $a^r \equiv 1 \mod N$, then we can write $a^r - 1 \equiv 0 \mod N$.
- As long as r is even (and it is with enough probability, so if not we just try again), we can write $(a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \mod N$.

Shor's algorithm: period to integer factors

- Shor's algorithm is a quantum algorithm which can be used to factor integers, and hence break RSA given enough qubits.
- The essential component of Shor's algorithm is finding the multiplicative order of an integer modulo N.
- For if we have such an a such that $a^r \equiv 1 \mod N$, then we can write $a^r - 1 \equiv 0 \mod N$.
- As long as r is even (and it is with enough probability, so if not we just try again), we can write $(a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \mod N$.
- This means we can extract a factor via $\gcd(a^{r/2} - 1, N)$ or $\gcd(a^{r/2} + 1, N)$ since we know $N|(a^{r/2} - 1)(a^{r/2} + 1)$.

Shor's algorithm in a nutshell

Shor's algorithm proceeds essentially in four steps:

- create a uniform superposition $2^{-N/2} \sum_{k=0}^{N-1} |k\rangle$ using Hadamard gates
- apply modular exponential gates $U|k\rangle = |a^k \mod N\rangle$

Shor's algorithm in a nutshell

Shor's algorithm proceeds essentially in four steps:

- create a uniform superposition $2^{-N/2} \sum_{k=0}^{N-1} |k\rangle$ using Hadamard gates
- apply modular exponential gates $U|k\rangle = |a^k \mod N\rangle$
- use the quantum Fourier transform (QFT) to perform phase estimation and extract period r

Shor's algorithm in a nutshell

Shor's algorithm proceeds essentially in four steps:

- create a uniform superposition $2^{-N/2} \sum_{k=0}^{N-1} |k\rangle$ using Hadamard gates
- apply modular exponential gates $U|k\rangle = |a^k \mod N\rangle$
- use the quantum Fourier transform (QFT) to perform phase estimation and extract period r
- use classical continued fractions to extract the actual period

Shor's algorithm in a nutshell

- We construct the superposition using Hadamard gates which map

$$|0\rangle \mapsto |0\rangle + |1\rangle$$

$$|1\rangle \mapsto |0\rangle - |1\rangle$$

  mixing each qubit state. Taking tensor products, we get the full superposition.
- Use modular exponentiation and repeated doubling gates U_{2^j}, where $U_{2^j}|k\rangle = |a^{2^j}\rangle$ and we do this for each bit of k. Very hard-coded.

Shor's algorithm in a nutshell

- We construct the superposition using Hadamard gates which map

$$|0\rangle \mapsto |0\rangle + |1\rangle$$
$$|1\rangle \mapsto |0\rangle - |1\rangle$$

  mixing each qubit state. Taking tensor products, we get the full superposition.
- Use modular exponentiation and repeated doubling gates U_{2^j}, where $U_{2^j}|k\rangle = |a^{2^j}\rangle$ and we do this for each bit of k. Very hard-coded.
- Peaked for the frequencies that are present in the function on $(\mathbb{Z}/N\mathbb{Z})^\times$. Namely, the function $|k\rangle \mapsto |a^k\rangle$, which is periodic of period r, so again $a^r \equiv 1 \mod N$.

Shor's algorithm in a nutshell

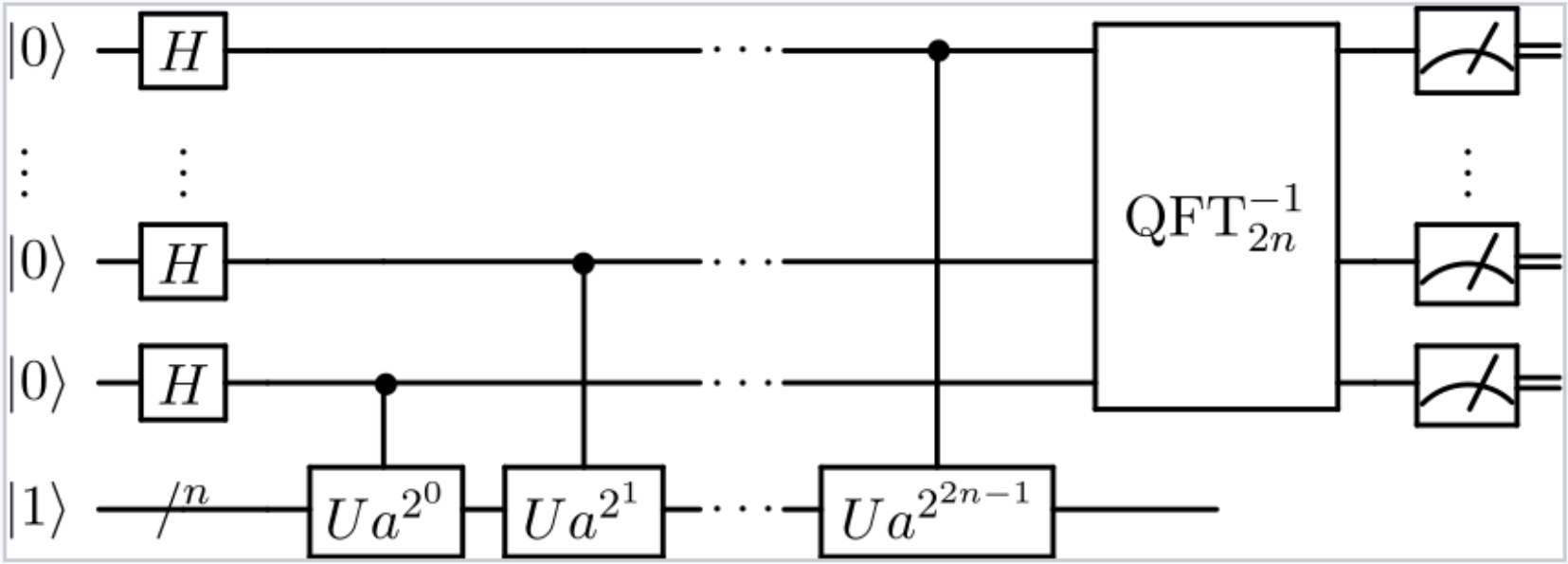
- We construct the superposition using Hadamard gates which map

$$|0\rangle \mapsto |0\rangle + |1\rangle$$

$$|1\rangle \mapsto |0\rangle - |1\rangle$$

  mixing each qubit state. Taking tensor products, we get the full superposition.
- Use modular exponentiation and repeated doubling gates U_{2^j}, where $U_{2^j}|k\rangle = |a^{2^j}\rangle$ and we do this for each bit of k. Very hard-coded.
- Peaked for the frequencies that are present in the function on $(\mathbb{Z}/N\mathbb{Z})^\times$. Namely, the function $|k\rangle \mapsto |a^k\rangle$, which is periodic of period r, so again $a^r \equiv 1 \mod N$.
- obtain phase ϕ related to r, can be expressed as k/r. Use continued fractions algorithm to approximate ϕ as k/r, then deduce r

Shor's algorithm: phase estimation circuit



Quantum subroutine in Shor's algorithm

Motivation for Modern Cryptography

For modern cryptography, we aim to find problems that are impractical or ideally impossible to solve, even on a quantum computer.

The "learning with errors" (LWE) problem, along with its ring-LWE variant, is an example of such problems. It involves distinguishing between two distributions:

- A set of random linear equations perturbed by a small error (noise)
- A truly uniform random distribution

The Learning with Errors Problem

Given $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_{\mathbf{q}}^{\mathbf{n} \times \mathbf{m}} \times \mathbb{Z}_{\mathbf{q}}^{\mathbf{m}}$ where $\mathbf{b} = \mathbf{As} + \mathbf{e} \mod \mathbf{q}$:

- $\mathbf{A}$ is a known random matrix
- $\mathbf{s}$ is a secret vector
- $\mathbf{e}$ is a noise vector sampled from a narrow error distribution
- q is a large modulus

The goal is to recover the secret $\mathbf{s}$ or distinguish $(\mathbf{A}, \mathbf{b})$ from uniformly random samples. We'd like to know that instances of this problem are hard in the average case. One way to do this is by proving a reduction, that solving implies you've solved a "hard" problem in computer science. The two relevant reductions are:

Hardness of LWE

We would like to ensure that instances of LWE are hard in the average case. This is done by proving reductions from well-known hard problems in computer science:

- **GapSVP**: Shortest vector problem with a gap
- **SIVP**: Shortest independent vector problem

These lattice problems are:

- NP-hard in their exact versions
- Computationally hard in their approximate versions

The Ring-LWE Problem

The variant we focus on is ring-LWE, where the lattices are number-theoretic and derived from ideals in certain polynomial rings:

$$R_q = \mathbb{Z}[x]/(f(x))$$

where $f(x)$ is typically a polynomial like $x^n - 1$. However, this choice is insecure. Instead, we use:

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

where n is typically a power of two. This is the "anti-cyclotomic" ring of integers. The hard problem becomes:

- **Ideal** $-$ **SVP**: Shortest vector problem for ideal lattices

Lattice Structure of the Ring

Consider the ring $R = \mathbb{Z}[x]/(x^n + 1)$:

- This is a $\mathbb{Z}$-module of rank n.
- Any element $a(x) \in R$ can be written as:

$$a(x) = \sum_{i=0}^{n-1} c_i x^i$$

- The vector $(c_i)_{i=0}^{n-1}$ is the coefficient vector, making $R \cong \mathbb{Z}^n$.

Connection Between Ring-LWE and Ideal Lattices

- $(x^n + 1)$ is an ideal, and elements $e(x) \in R$ map to elements in this lattice.
- The error $e(x)$ corresponds to a short vector (in the Euclidean norm) in the lattice, representing a small perturbation.

Ideal lattices inherit the ring's structure, such as multiplication by polynomials.

Reduction from Ring-LWE to Ideal-SVP

Solving ring-LWE allows efficient recovery of short vectors in ideal lattices:

- Ring-LWE enables recovery of the secret $s(x)$, which corresponds to information about the underlying lattice structure.
- Decoding the noisy lattice point perturbed by $e(x)$ reveals a short vector.

The reduction shows that solving ring-LWE allows decoding perturbed lattice points for any ideal lattice in R, solving Ideal-SVP in the process.

ring-LWE Overview

Our goal is to introduce the simplest possible implementation of the ring-LWE encryption scheme.

- Choose a moderately large prime p and large n
- n should be a power of two and 512 or above
- Example: $p = 3329$

Let

$$R_p := \mathbb{F}_p[x]/(x^n + 1)$$

This is a finite ring with p^n elements. It is not a finite field, as $x^n + 1$ factors modulo p (though it is irreducible over $\mathbb{Z}$).
The elements are:

$$R_p = \{a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0 : a_i \in \mathbb{F}_p\}$$

Public Setup

The public setup involves the following:

1. A prime p and dimension n resulting in R_p
2. A moderately large integer $k \in \mathbb{Z}$
3. A notion of "small", as applied to elements of R_p.
   - These will be "ternary polynomials" with coefficients in $\{-1, 0, +1\}$

Key Generation: Private/Public Keypair

Bob creates a private/public keypair as follows:

1. Bob selects a small random element s of R_p
2. Bob selects a small random element e_1 of R_p
3. Bob defines $(a, b = as + e_1) \in R_p \times R_p$

- The element e_1 can be discarded
- Bob keeps s as his secret key
- Bob makes (a, b) public as his public key

Encryption by Alice

Alice encrypts a message m as follows:

1. Select a small random $r \in R_p$ (ephemeral key)
2. Select small random $e_2, e_3 \in R_p$
3. Define $v = ar + e_2$, $w = br + e_3 + km$

- Alice may discard k, e_2, and e_3
- The ciphertext is (v, w), which is sent to Bob

Decryption by Bob

Bob decrypts the message:

1. Compute $x = w - vs$
2. Round x to the nearest multiple of k
3. The result should be an integer; divide it by k to reveal the message m

Example: Parameters

- $n = 4$
- $p = 101$
- $k = 20$

Example: Key Generation

- Private key: $s = x^3 + 100$
- Error for public key: $e = x^3 + x^2 + 100x$

Public key:

$$a = 83x^3 + 23x^2 + 51x + 77$$
$$b = as + e = 96x^3 + 97x^2 + 26x + 74$$

Example: Encryption

- Message $m \in \{0, 1, 2, 3, 4\}$, e.g., $m = 3$
- Ephemeral key: $r = 100x^2 + 100x$
- Errors for ciphertext:

$$e_1 = x^2$$
$$e_2 = 100x^2 + x$$

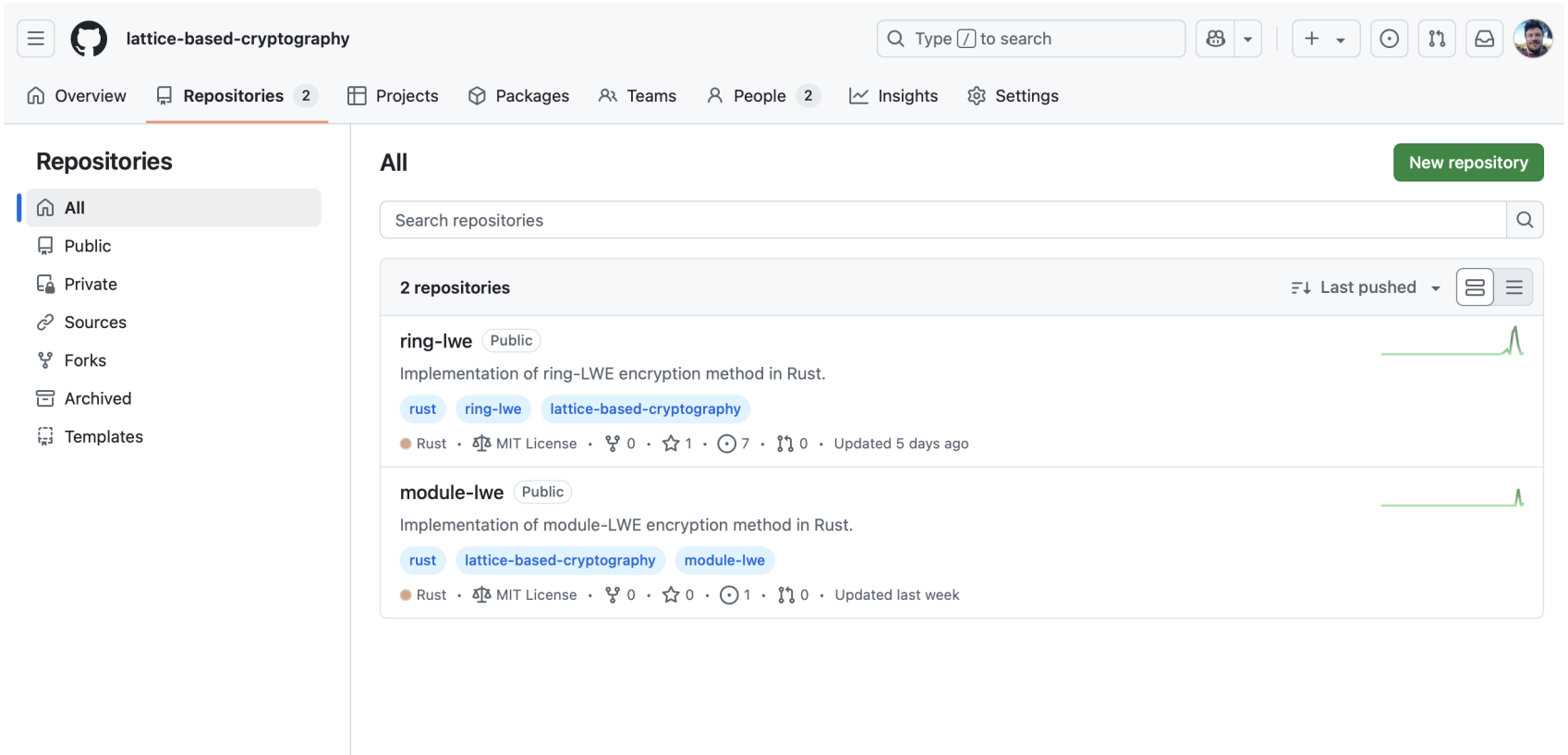
Ciphertext:

$$v = ar + e_1 = 27x^3 + 75x^2 + 6x + 5$$
$$w = br + e_2 + km = 79x^3 + 23x + 51$$

Example: Decryption

- Decryption formula: $w - vs = x^2 + 3x + 62$
- Round to nearest 20: $60 = 3k$
- The message is $m = 3$

https://github.com/lattice-based-cryptography



ring-LWE: Library functions

```
use polynomial_ring::Polynomial;
use rand_distr::{Uniform, Normal, Distribution};
use rand::SeedableRng;
use rand::rngs::StdRng;

...

impl Default for Parameters {
    fn default() -> Self {
        let n = 16;
        let q = 65536;
        let t = 512;
        let mut poly_vec = vec![0i64;n+1];
        poly_vec[0] = 1;
        poly_vec[n] = 1;
        let f = Polynomial::new(poly_vec);
        Parameters { n, q, t, f }
    }
}
```

Listing 1: lib.rs

ring-LWE: polymul

```
pub fn polymul(x : &Polynomial<i64>, y : &Polynomial<i64>, modulus : i64, f : &
     Polynomial<i64>) -> Polynomial<i64> {
    //Multiply two polynoms
    //Args:
    //  x, y: two polynoms to be multiplied.
    //  modulus: coefficient modulus.
    //  f: polynomial modulus.
    //Returns:
    //  polynomial in Z_modulus[X]/(f).
        let mut r = x*y;
        r = mod_coeffs(r, modulus);
        r.division(f);
        mod_coeffs(r, modulus)
}
```

Listing 2: polymul function

ring-LWE: Key generation

```
use polynomial_ring::Polynomial;
use ring_lwe::{Parameters, polymul, polyadd, polyinv, gen_binary_poly,
     gen_uniform_poly, gen_normal_poly};
use std::collections::HashMap;

pub fn keygen(params: &Parameters, seed: Option<u64>) -> ([Polynomial<i64>; 2],
     Polynomial<i64>) {

    //rename parameters
    let (n, q, f) = (params.n, params.q, &params.f);

    // Generate a public and secret key
    let sk = gen_binary_poly(n,seed);
    let a = gen_uniform_poly(n, q, seed);
    let e = gen_normal_poly(n, seed);
    let b = polyadd(&polymul(&polyinv(&a,q*q), &sk, q*q, &f), &polyinv(&e,q*q),
         q*q, &f);

    // Return public key (b, a) as an array and secret key (sk)
    ([b, a], sk)
}
```

Listing 3: keygen

ring-LWE: Encryption

```
use polynomial_ring::Polynomial;
use ring_lwe::{Parameters, mod_coeffs, polymul, polyadd, gen_binary_poly,
     gen_normal_poly};

pub fn encrypt(
    pk: &[Polynomial<i64>; 2],     // Public key (b, a)
    m: &Polynomial<i64>,          // Plaintext polynomial
    params: &Parameters,         //parameters (n,q,t,f)
    seed: Option<u64>             // Seed for random number generator
) -> (Polynomial<i64>, Polynomial<i64>) {
    let (n,q,t,f) = (params.n, params.q, params.t, &params.f);
    // Scale the plaintext polynomial. use floor(m*q/t) rather than floor (q/t)*
         m
    let scaled_m = mod_coeffs(m * q / t, q);

    // Generate random polynomials
    let e1 = gen_normal_poly(n, seed);
    let e2 = gen_normal_poly(n, seed);
    let u = gen_binary_poly(n, seed);

    // Compute ciphertext components
    let ct0 = polyadd(&polyadd(&polymul(&pk[0], &u, q*q, f), &e1, q*q, f),&
         scaled_m,q*q,f);
    let ct1 = polyadd(&polymul(&pk[1], &u, q*q, f), &e2, q*q, f);

    (ct0, ct1)
}
```

Listing 4: encrypt

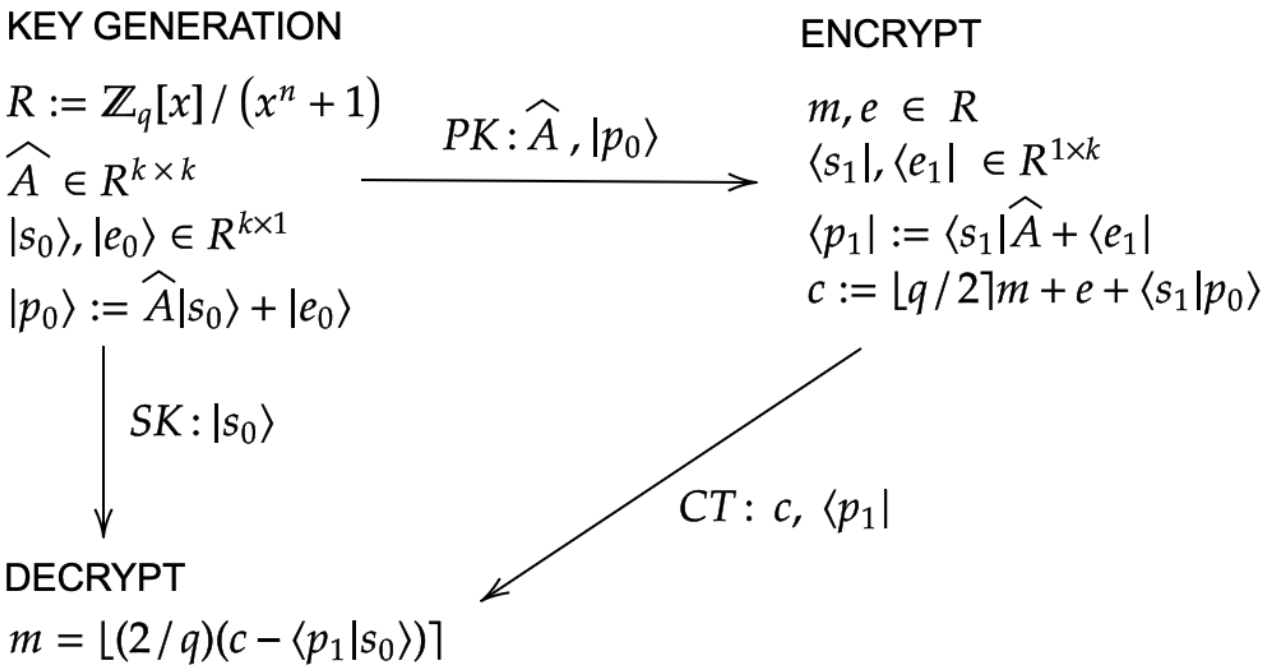
ring-LWE: decryption

```
use polynomial_ring::Polynomial;
use ring_lwe::{Parameters, polymul, polyadd, nearest_int};

pub fn decrypt(
    sk: &Polynomial<i64>,     // Secret key
    ct: &[Polynomial<i64>; 2],        // Array of ciphertext polynomials
    params: &Parameters
) -> Polynomial<i64> {
    let (_n,q,t,f) = (params.n, params.q, params.t, &params.f);
        let scaled_pt = polyadd(&polymul(&ct[1], sk, q, f),&ct[0], q, f);
        let mut decrypted_coeffs = vec![];
        let mut s;
        for c in scaled_pt.coeffs().iter() {
                s = nearest_int(c*t,q);
                decrypted_coeffs.push(s.rem_euclid(t));
        }
    Polynomial::new(decrypted_coeffs)
}
```

Listing 5: decrypt

module-LWE diagram



FIPS 203

- NIST recently (Aug.) released their post-quantum cryptography standards
- CRYSTALS Kyber was selected for the KEM, forming the FIPS 203 standard

FIPS 203

- NIST recently (Aug.) released their post-quantum cryptography standards
- CRYSTALS Kyber was selected for the KEM, forming the FIPS 203 standard
- The core of this algorithm is module-LWE

FIPS 203

- NIST recently (Aug.) released their post-quantum cryptography standards
- CRYSTALS Kyber was selected for the KEM, forming the FIPS 203 standard
- The core of this algorithm is module-LWE
- See: `https://pq-crystals.org/kyber/` and `https://csrc.nist.gov/pubs/fips/203/final`

References I


"Cryptography," *Encyclopedia Britannica*, Encyclopedia Britannica, Inc., 2025. Available: https://www.britannica.com/topic/cryptology/History-of-


Jackson Walters. *a brief introduction to quantum algorithms*. https://jacksonwalters.com/blog/?p=1


Katherine Stange. *Ring-LWE notes* https://math.colorado.edu/~kstange/teaching-resources/c


Jackson Walters, Thomas Silverman https://github.com/lattice-based-cryptography