Cryptography Lecture 12
Post-quantum cryptography

LINKÖPING
UNIVERSITY

Public key cryptography rests on hardness of a mathematical problem

- In RSA, the mathematical problem is factoring. Alice creates $N = pq$ where p and q are prime numbers, and publishes N (and the encryption exponent e)

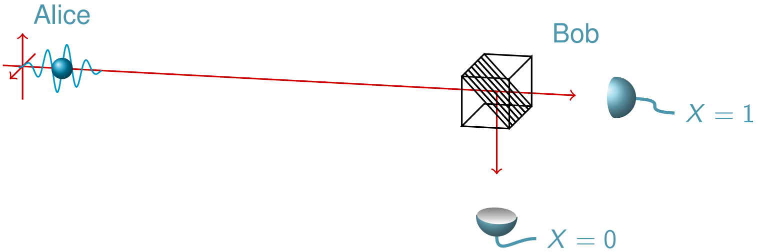
- If an eavesdropper can factor N, it is simple to calculate the decryption exponent d

- The best classical factoring algorithms we have are $O(e^{\sqrt[3]{n}})$

- Quantum computers are said to solve the problem much faster, complexity is $O(n^3)$

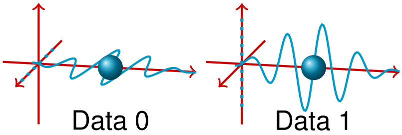
Quantum computers

- Quantum computers use the same information-encoding technique as quantum cryptography
- But the similarities end quickly
- Quantum computers use quantum gates to perform calculations

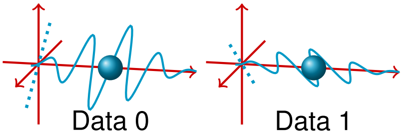


Encoding of information into quantum systems

Coding HV (Horizontal-Vertical), $+$, encoding 0

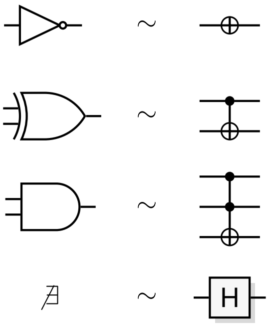


Coding PM (Plus-Minus 45°), $\times$, encoding 1



Quantum Algorithms use quantum bits and quantum gates

- The qubit is a spin-$\frac{1}{2}$-system
- $|\downarrow\rangle = |0\rangle$ and $|\uparrow\rangle = |1\rangle$
- $x = 0$ or 1 becomes
  $|\psi\rangle = a\,|0\rangle + be^{i\phi}\,|1\rangle$
- Gates are unitary maps, or reversible
- Hadamard gate
  $$H : \begin{cases} |0\rangle \to |+\rangle = |0\rangle + |1\rangle \\ |1\rangle \to |-\rangle = |0\rangle - |1\rangle \end{cases}$$

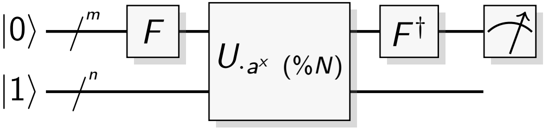


Shor's algorithm finds the period of a function f

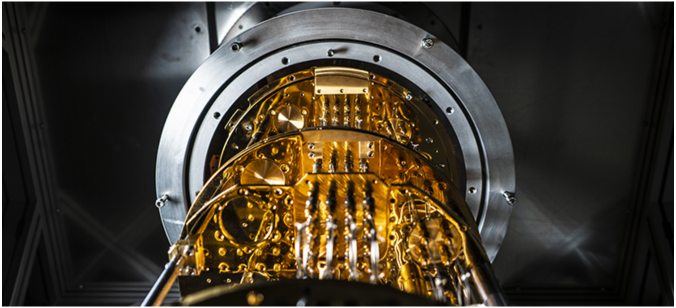
- Remember that $a^{\phi(N)} = 1 \bmod N$ (where ϕ is the totient function), so the function $f(x) = a^x \bmod N$ is periodic

$$1,\ a,\ a^2, \dots, a^{\phi(N)-1},\ a^{\phi(N)} = 1,\ a,\ a^2, \dots \qquad (\bmod N)$$

- Shor's algorithm finds a period r (such that $f(x + r) = f(x)$)
- The period can be used to find factors in N (how?)
- Shor's algorithm needs $O(n^3)$ *quantum* operations



There aren't any good quantum computers, . . . , yet



- Several giant projects are under way
- The above picture is from Chalmers, one of the participants of Wallenberg Center for Quantum Technology

We will probably need to replace RSA with Post-quantum(-computer) cryptography

- Don't rely on factorization or discrete log
- Use a properly hard problem
- Candidates are NP-complete or even NP-hard problems (at least as hard as NP-complete problems)
- But remember the failure of Knapsack crypto

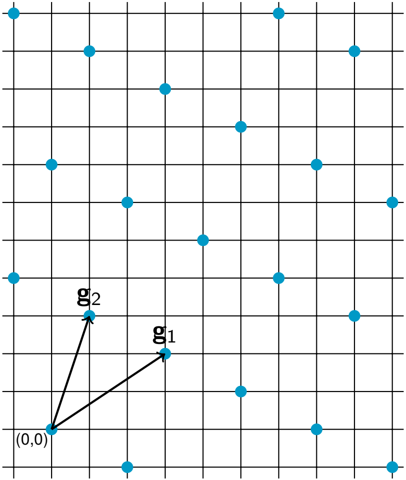
We will probably need to replace RSA with Post-quantum(-computer) cryptography

- Don’t rely on factorization or discrete log
- Use a properly hard problem
- Candidates are NP-complete or even NP-hard problems (at least as hard as NP-complete problems)
- But remember the failure of Knapsack crypto

- The terms "NP-complete" and "NP-hard" only refers to the hardest instance of the problem
- In cryptography, average hardness is the important property

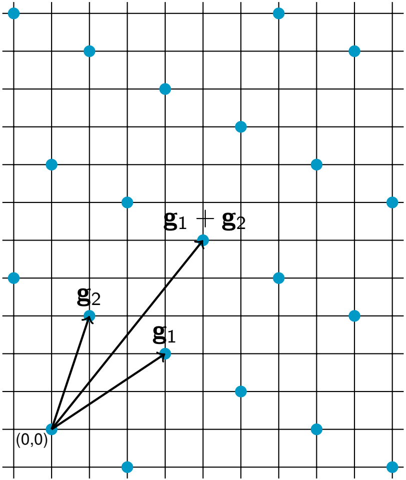
McEliece or code-based cryptography (1978)

- Use Error-Correcting Codes
- More precisely, use a general linear ECC
- Code words are vectors in $\mathbb{R}^n$ (simplest example is binary vectors)
- A linear code is such that adding two code words gives a third code word



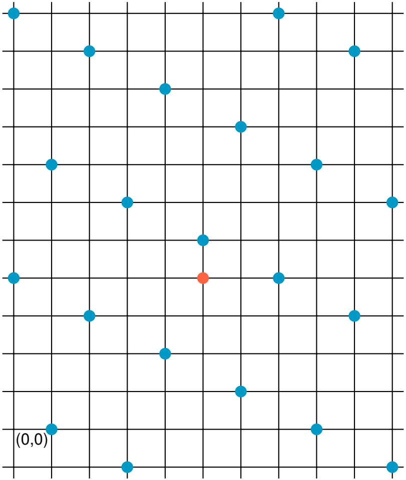
McEliece or code-based cryptography (1978)

- Use Error-Correcting Codes
- More precisely, use a general linear ECC
- Code words are vectors in $\mathbb{R}^n$ (simplest example is binary vectors)
- A linear code is such that adding two code words gives a third code word



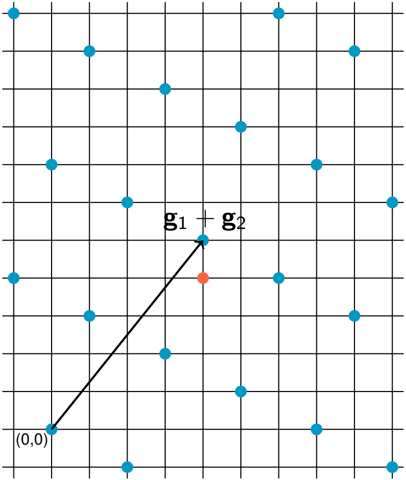
McEliece or code-based cryptography (1978)

- Errors in transmission gives random shifts
- "Decoding" or "Error correction" is the same as finding the closest code word



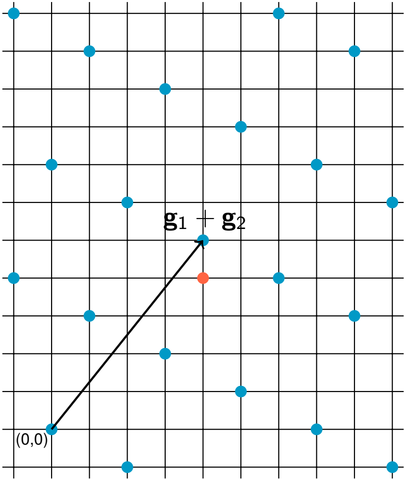
McEliece or code-based cryptography (1978)

- Errors in transmission gives random shifts
- "Decoding" or "Error correction" is the same as finding the closest code word



McEliece or code-based cryptography (1978)

- Errors in transmission gives random shifts
- "Decoding" or "Error correction" is the same as finding the closest code word
- Efficient decoding exists for known families of ECC
- But decoding a general linear ECC is NP-hard



Mathematical notation

- Our code has 2^k code words, each code word is n bits long, and can correct t one-bit errors (t is given by the construction)

- Our code maps from k-bit strings to n-bit strings using a bit matrix called generator matrix G

$$\mathbf{y}^t = \mathbf{x}^t G \pmod 2$$

- To each G, there is a decoding procedure (a function) we denote D

$$D(\mathbf{y}^t) = D(\mathbf{x}^t G) = \mathbf{x}^t$$

Mathematical notation

- Our code has 2^k code words, each code word is n bits long, and can correct t one-bit errors (t is given by the construction)

- Our code maps from k-bit strings to n-bit strings using a bit matrix called generator matrix G

$$\mathbf{y}^t = \mathbf{x}^t G \pmod 2$$

- To each G, there is a decoding procedure (a function) we denote D

$$D(\mathbf{y}^t) = D(\mathbf{x}^t G) = \mathbf{x}^t$$

- The decoding procedure can correct errors, so if $\mathbf{z}$ has less than k ones,

$$D(\mathbf{y}^t + \mathbf{z}^t) = D(\mathbf{y}^t) = \mathbf{x}^t$$

McEliece or code-based cryptography (1978)

- Use Error-Correcting Codes
- Efficient decoding exists for known families of ECC
- Decoding a general linear ECC is NP-hard
- Use a code from a known family, but randomize the code so that the code can't be identified
- Then only general-linear-ECC decoding is available

McEliece or code-based cryptography (1978)

- Use Error-Correcting Codes
- Efficient decoding exists for known families of ECC
- Decoding a general linear ECC is NP-hard
- Use a code from a known family, but randomize the code so that the code can't be identified
- Then only general-linear-ECC decoding is available

- But remember the failure of Knapsack crypto

McEliece or code-based cryptography (1978)

- Bob chooses a "binary Goppa code" C (length n with 2^k code words, that corrects t errors), a generator matrix G and the corresponding decoding algorithm D
- Bob also selects a random $k \times k$ binary invertible matrix S and a random $n \times n$ permutation matrix P, and calculates $\widehat{G} = SGP$
- Bob makes $\widehat{G}$ and t public, and keeps S, P, D (and G) secret
- Alice encrypts $\mathbf{m}$ as $\mathbf{c}^t = \mathbf{m}^t\widehat{G} + \mathbf{z}^t$ where $\mathbf{z}$ is a random n-bit string with t bits set
- Bob decrypts $\mathbf{c}$ as

$$\begin{aligned}
D(\mathbf{c}^t P^{-1})S^{-1} &= D\big((\mathbf{m}^t SGP + \mathbf{z}^t)P^{-1}\big)S^{-1} \\
&= D(\mathbf{m}^t SG + \mathbf{z}^t P^{-1})S^{-1} \\
&= D(\mathbf{m}^t SG)S^{-1} \\
&= (\mathbf{m}^t S)S^{-1} = \mathbf{m}^t,
\end{aligned}$$

Trapdoor one-way function candidate: randomized Goppa code + errors

A trapdoor one-way function is a function that is easy to compute but computationally hard to reverse

- Easy to calculate $(\mathbf{x}^t \widehat{G} + \mathbf{z}^t)$ from $\mathbf{x}$
- Hard to invert: to calculate $\mathbf{x}$ from $(\mathbf{x}^t \widehat{G} + \mathbf{z}^t)$

The trapdoor is that with knowledge of S, P, and D it is easy to invert, to calculate $\mathbf{x}^t = D\big((\mathbf{x}^t \widehat{G} + \mathbf{z}^t)P^{-1}\big)S^{-1}$

Decoding a general linear code $(\widehat{G})$ is NP-hard

McEliece or code-based cryptography (1978)

- The family of codes used turns out to be very important
- The binary Goppa codes used in the initial proposal are basically the only family that works
- For example, Reed-Solomon codes enables a “structural attack,” an efficient algorithm for randomized RS codes

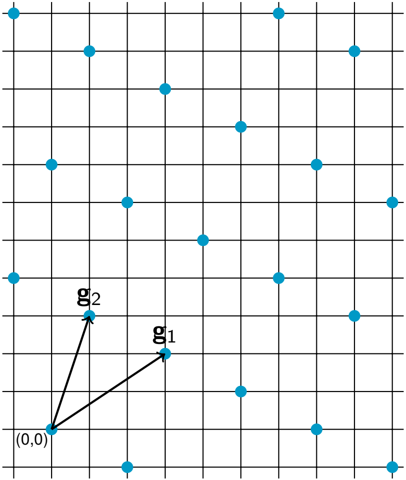
- The best known general-linear-ECC decoder is “information set decoding,” the initial key size 262 kbit gives $\sim$60 bits of security
- Current recommendation is 8.4 Mbit keys (!)
- On the positive side, system is faster than RSA

More recent candidate: Lattice problems

- A lattice in $\mathbb{R}^n$ is defined by a basis $\mathbf{g}_i$ with k elements, $k \leq n$
- It consists of the points

$$\sum_{i=1}^{k} x_i \mathbf{g}_i = \mathbf{x}^t G, \quad \forall x_i \in \mathbb{Z}$$

- The Shortest Vector Problem (SVP) is to find the shortest nonzero vector in a lattice

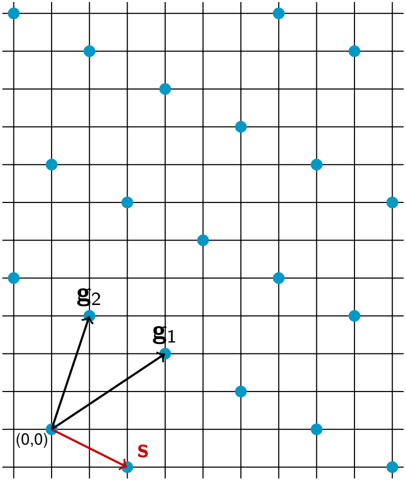


More recent candidate: Lattice problems

- A lattice in $\mathbb{R}^n$ is defined by a basis $\mathbf{g}_i$ with k elements, $k \leq n$
- It consists of the points

$$\sum_{i=1}^{k} x_i \mathbf{g}_i = \mathbf{x}^t G, \quad \forall x_i \in \mathbb{Z}$$

- The Shortest Vector Problem (SVP) is to find the shortest nonzero vector in a lattice

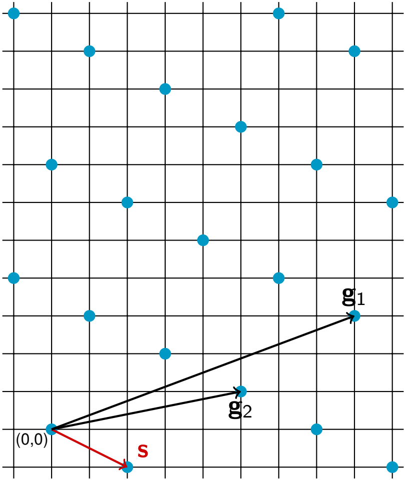


More recent candidate: Lattice problems

- A lattice in $\mathbb{R}^n$ is defined by a basis $\mathbf{g}_i$ with k elements, $k \leq n$
- It consists of the points

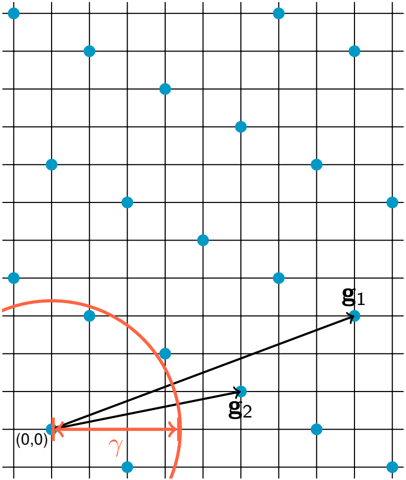
$$\sum_{i=1}^{k} x_i \mathbf{g}_i = \mathbf{x}^t G, \quad \forall x_i \in \mathbb{Z}$$

- The Shortest Vector Problem (SVP) is to find the shortest nonzero vector in a lattice



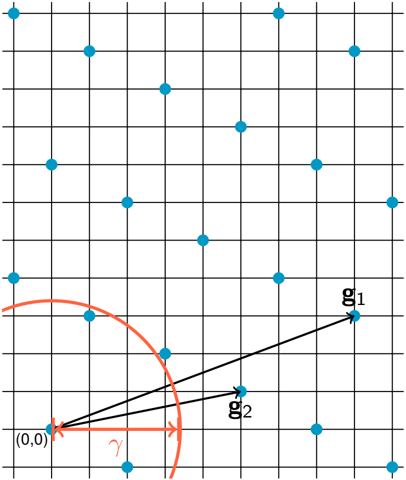
The Shortest Vector Problem

- SVP is NP-hard
- Checking if there exists a vector shorter than γ, or GapSVP$_\gamma$, is also NP-hard
- But this is worst-case hardness
- A generic instance may have an efficient solution
- Another knapsack crypto?



The Shortest Vector Problem

- GapSVP_γ is NP-hard, worst-case complexity
- The way forward is to randomize the lattice
- Problem is, how do we randomize the basis?
- No upper limit to $||\mathbf{g}_i||$
- No simple link between properties of the basis, γ, and instance complexity



Reformulate: Mod-q-vector problems

- Instead use m vectors $\mathbf{h}_i$ with n coordinates mod q, both m and $q > n$
- An *Integer Solution* is a nontrivial solution to the equation

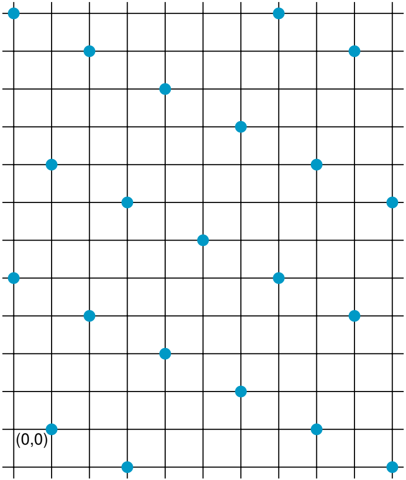
$$\sum_{i=1}^{m} \mathbf{h}_i z_i = H\mathbf{z} = \mathbf{0} \mod q$$

- Gaussian elimination works, if no restrictions are added
- The Short Integer Solution problem (SIS) is to find a “short” nonzero solution (such that $||\mathbf{z}|| < \beta$)

Reformulate: Error-correcting codes (mod q)

- Compare with error-correcting codes
- Code words are solutions to the *parity check* equation

$$Hz = \mathbf{0} \mod q$$

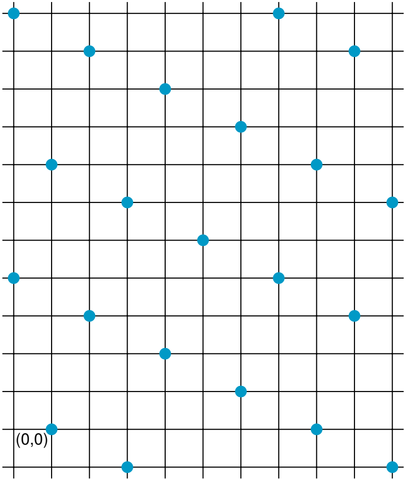


Reformulate: Error-correcting codes (mod q)

- Compare with error-correcting codes
- Code words are solutions to the *parity check* equation

$$H\mathbf{z} = \mathbf{0} \mod q$$

- An error is detected if the *syndrome* is nonzero

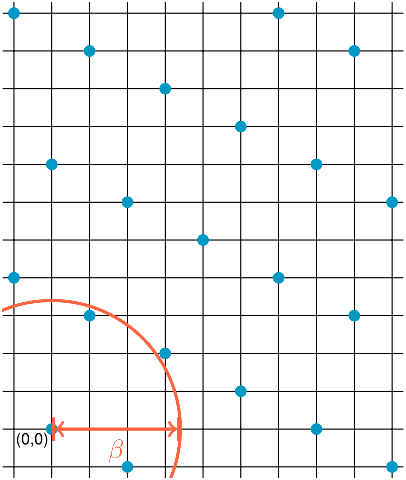


Reformulate: Error-correcting codes (mod q)

- Compare with error-correcting codes
- Code words are solutions to the *parity check* equation

  $$H\mathbf{z} = \mathbf{0} \mod q$$

- An error is detected if the *syndrome* is nonzero
- An SIS solution is a short code word $\mathbf{z}$

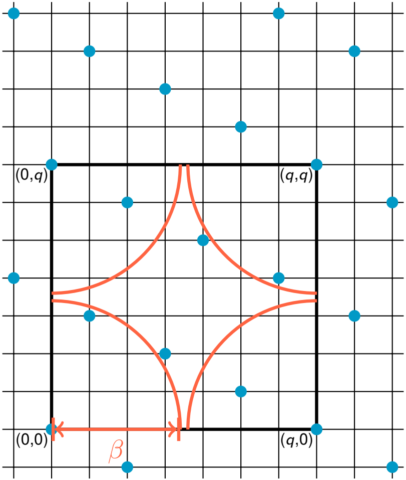


Reformulate: Error-correcting codes (mod q)

- Compare with error-correcting codes
- Code words are solutions to the *parity check* equation

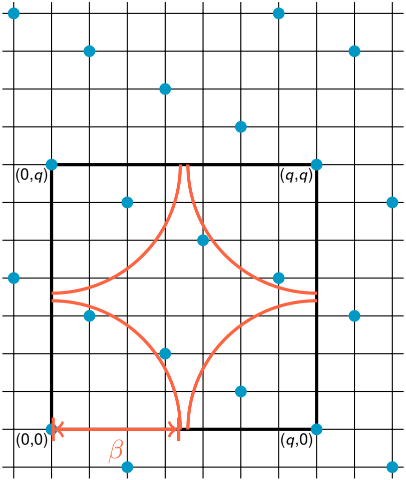
$$Hz = \mathbf{0} \mod q$$

- An error is detected if the *syndrome* is nonzero
- An SIS solution is a short code word $\mathbf{z}$



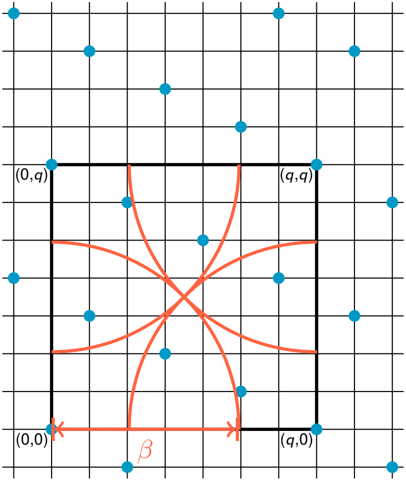
Reformulate: Short Integer Solution problem

- SIS is almost “GapSVP mod q”



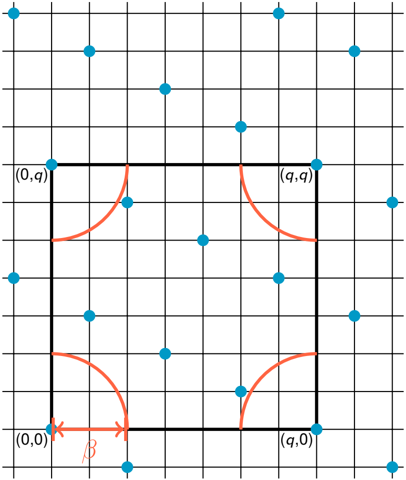
Reformulate: Short Integer Solution problem

- SIS is almost “GapSVP mod q”
- For large β Gaussian elimination gives an efficient solution



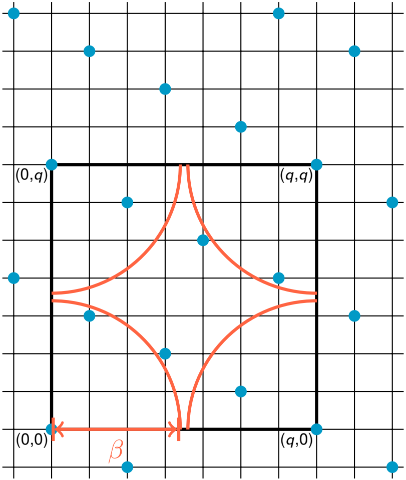
Reformulate: Short Integer Solution problem

- SIS is almost "GapSVP mod q"
- For large β Gaussian elimination gives an efficient solution
- For small β there are no solutions



Reformulate: Short Integer Solution problem

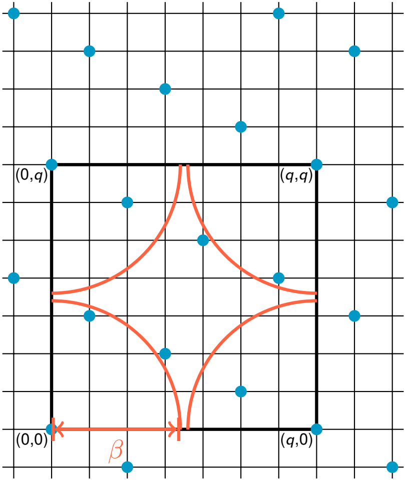
- SIS is almost "GapSVP mod q"
- For large β Gaussian elimination gives an efficient solution
- For small β there are no solutions
- Perhaps there is a region in between where the problem is hard, on average?



The Short Integer Solution problem is hard on average

- Use $m \geq n \log q$ and $\beta \geq \sqrt{m}$
- Problem is trivial if $\beta \geq q$, a common choice is $q \approx n^3 \gg \beta$

- Solving SIS_β for uniformly random H with high probability
  $\Rightarrow$
  solving $\mathrm{GapSVP}_{\beta\sqrt{n}}$ with probability exponentially close to 1



SIS hash function (Ajtai 1996)

- Let $m \geq n \log q$, choose random n-by-m matrix H of integers mod q, and let
$$h_H(\mathbf{x}) = H\mathbf{x}$$
- The hash function h_H from bitstrings length m to bitstrings length n is collision-resistant

SIS hash function (Ajtai 1996)

- Let $m \geq n \log q$, choose random n-by-m matrix H of integers mod q, and let
$$h_H(\mathbf{x}) = H\mathbf{x}$$
- The hash function h_H from bitstrings length m to bitstrings length n is collision-resistant

- A collision $h_H(\mathbf{x}) = h_H(\mathbf{x}')$ implies that $H(\mathbf{x} - \mathbf{x}') = \mathbf{0}$
- Then $\mathbf{z} = \mathbf{x} - \mathbf{x}'$ is a solution to $\mathrm{SIS}_{\sqrt{m}}$, because $||z|| \leq \sqrt{m}$
- If it is easy to find collisions, it is easy to solve $\mathrm{SIS}_{\sqrt{m}}$

How to choose random H that has a short solution

- It is simple to choose a random matrix mod q, but that does not guarantee existence of a short solution
- The trick is to use a random matrix $\overline{H}$ and a random vector $\bar{\mathbf{x}}$, and then generate a larger matrix H with a short solution
- The process is called “reducing $\bar{\mathbf{x}}$ modulo the lattice”

How to choose random H that has a short solution

- It is simple to choose a random matrix mod q, but that does not guarantee existence of a short solution
- The trick is to use a random matrix $\overline{H}$ and a random vector $\bar{\mathbf{x}}$, and then generate a larger matrix H with a short solution
- The process is called “reducing $\bar{\mathbf{x}}$ modulo the lattice”

- Let $\overline{m} = m - 1 \geq n \log q$ and draw uniform random $n \times \overline{m}$ matrix $\overline{H}$ and binary $\overline{m}$-element vector $\bar{\mathbf{x}}$

How to choose random H that has a short solution

- It is simple to choose a random matrix mod q, but that does not guarantee existence of a short solution
- The trick is to use a random matrix $\overline{H}$ and a random vector $\overline{\mathbf{x}}$, and then generate a larger matrix H with a short solution
- The process is called "reducing $\overline{\mathbf{x}}$ modulo the lattice"

- Let $\overline{m} = m - 1 \geq n \log q$ and draw uniform random $n \times \overline{m}$ matrix $\overline{H}$ and binary $\overline{m}$-element vector $\overline{\mathbf{x}}$
- Add a column to $\overline{H}$ and a row to $\overline{\mathbf{x}}$,

$$H = \left[\overline{H} \middle| -h_{\overline{H}}(\overline{\mathbf{x}})\right] = \left[\overline{H} \middle| -\overline{H}\overline{\mathbf{x}}\right], \qquad \mathbf{x} = \begin{bmatrix} \overline{\mathbf{x}} \\ 1 \end{bmatrix}$$

How to choose random H that has a short solution

- It is simple to choose a random matrix mod q, but that does not guarantee existence of a short solution
- The trick is to use a random matrix $\overline{H}$ and a random vector $\overline{\mathbf{x}}$, and then generate a larger matrix H with a short solution
- The process is called "reducing $\overline{\mathbf{x}}$ modulo the lattice"

- Let $\overline{m} = m - 1 \geq n \log q$ and draw uniform random $n \times \overline{m}$ matrix $\overline{H}$ and binary $\overline{m}$-element vector $\overline{\mathbf{x}}$
- Add a column to $\overline{H}$ and a row to $\overline{\mathbf{x}}$,

$$H = \left[\overline{H} \middle| -h_{\overline{H}}(\overline{\mathbf{x}})\right] = \left[\overline{H} \middle| -\overline{H}\overline{\mathbf{x}}\right], \qquad \mathbf{x} = \begin{bmatrix} \overline{\mathbf{x}} \\ 1 \end{bmatrix}$$

- Then $||\mathbf{x}|| \leq \sqrt{m}$, and $H\mathbf{x} = \overline{H}\overline{\mathbf{x}} - \overline{H}\overline{\mathbf{x}} = \mathbf{0}$

What is the distribution of H?

- $\overline{H}$ and $\overline{\mathbf{x}}$ are uniform (mod q and mod 2), so what about H?

What is the distribution of H?

- $\overline{H}$ and $\bar{\mathbf{x}}$ are uniform (mod q and mod 2), so what about H?

- Observation: $h_{\overline{H}}$ is not only collision resistant, the parameter $\overline{H}$ indexes a Universal-2 hash function family

- Leftover hash lemma: If $h_{\overline{H}}$ is a Universal-2 hash function family, $\overline{H}$ is uniform and $\bar{\mathbf{x}}$ has high min-entropy, then the pair $\overline{H}$, $h_{\overline{H}}(\bar{\mathbf{x}})$ is close (in statistical distance) to being uniform, i.e.,

$$A = \left[\overline{H} \middle| - h_{\overline{H}}(\bar{\mathbf{x}})\right] \overset{s}{\approx} \text{uniform}$$

(Remember:) (Trapdoor) One-way functions

A (trapdoor) one-way function is a function that is easy to compute but computationally hard to reverse. Examples:

- RSA (factoring)
- Knapsack (NP-complete but insecure with trapdoor)
- Diffie-Hellman + ElGamal (discrete log)
- EC Diffie-Hellman + EC ElGamal (EC discrete log)

The hash function family $\{h_H\}$ is Universal-2, so with H (almost) uniformly distributed, hash functions are collision resistant ($\subset$ one-way-hash), so we have

- Linear random hash mod q (SIS)

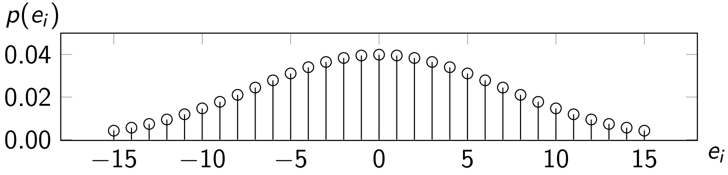
Actually, this is the only one-way (hash) function used in all of lattice cryptography

Learning With Errors (Regev 2005)

- Use m vectors $\mathbf{g}_i$ with n coordinates mod q, both m and $q > n$
- You are now given noisy data on the form

$$\mathbf{c}^t = \mathbf{s}^t G + \mathbf{e}^t, \quad \text{or } c_i = \mathbf{s}^t \mathbf{g}_i + e_i = \sum_{j=1}^{m} s_j g_{ij} + e_i, \quad \text{mod } q$$

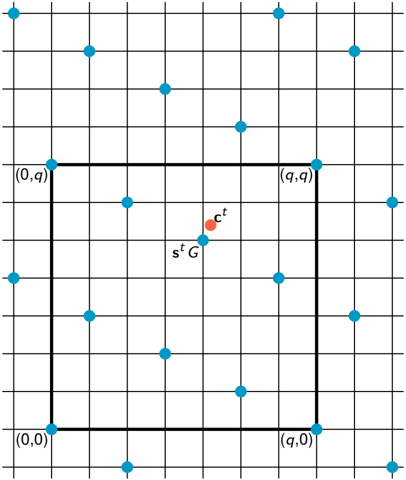
- The "noise" $\mathbf{e}$ is normally integer-Gaussian, stdev $\alpha q > \sqrt{n}$



- Learning With Errors (LWE) is the mathematical problem to find s

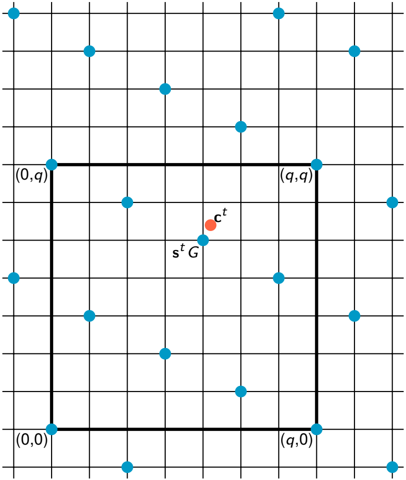
Learning With Errors

- LWE: Find s given noisy data on the form $\mathbf{c}^t = \mathbf{s}^t G + \mathbf{e}^t$
- Decision-LWE (DLWE): Is there an s so that $\mathbf{c}$ has the distribution of $\mathbf{s}^t G + \mathbf{e}^t$?
- (D)LWE is NP-hard
- Solving (D)LWE in the average case $\Rightarrow$ solving (D)LWE in the hard case



Learning With Errors

- DLWE: is it possible to decode? (NP-hard)
- Solving DLWE in the average case
  $\Rightarrow$
  solving DLWE in the hard case
- We are still relying on the hardness of decoding general linear codes



Example of cryptosystem with LWE (GPV 2008)

- Bob draws a random $n \times m$ matrix $\overline{G}$ mod q and a random m-bit string $\bar{\mathbf{x}}$, and sets $G = \left[\overline{G} \middle| -h_{\overline{G}}(\bar{\mathbf{x}})\right] = \left[\overline{G} \middle| -\overline{G}\bar{\mathbf{x}}\right]$ and $\mathbf{x} = \begin{bmatrix} \bar{\mathbf{x}} \\ 1 \end{bmatrix}$, so that $G\mathbf{x} = 0$
- Bob makes G public and keeps $\mathbf{x}$ secret
- Alice draws a random n-integer $\mathbf{s}$ and random integer-Gaussian $\mathbf{e}$ (mod q), and encrypts the single bit b as $\mathbf{c}^t = \mathbf{s}^t G + \mathbf{e}^t + (0, 0, \ldots, b)\lfloor q/2 \rfloor$
- Bob decrypts $\mathbf{c}$ as

$$\mathbf{c}^t\mathbf{x} = s^t G\mathbf{x} + \mathbf{e}^t\mathbf{x} + b\lfloor q/2 \rfloor = \mathbf{e}^t\mathbf{x} + b\lfloor q/2 \rfloor \approx bq/2$$

Security of example cryptosystem with LWE (GPV 2008)

- Secret key is $\bar{\mathbf{x}}$, public key is $G = \left[\overline{G} \middle| -h_{\overline{G}}(\bar{\mathbf{x}})\right] = \left[\overline{G} \middle| -\overline{G}\bar{\mathbf{x}}\right]$
- Eve can't recover $\bar{\mathbf{x}}$ efficiently, because then SIS would be simple to solve

- Ciphertext is $\mathbf{c}^t = \mathbf{s}^t G + \mathbf{e}^t + (0, 0, ..., b)\lfloor q/2 \rfloor$
- If $b = 0$ then $\mathbf{c}$ is distributed as $\mathbf{s}^t G + \mathbf{e}^t$
  If $b = 1$ then no $\mathbf{s}$ makes $\mathbf{c}$ distributed as $\mathbf{s}^t G + \mathbf{e}^t$
- If Eve can recover b, she can also solve DLWE ("Is there an s so that...?")

- But DLWE is hard (on average) so cryptosystem is secure

McEliece vs LWE

McEliece:

- Security rests on hardness of decoding general linear code
- Encryption is into a code word, $\mathbf{c}^t = \mathbf{m}^t \widehat{G} + \mathbf{z}^t$
- Uses random code from one particular subfamily of codes
- Choice of code family is crucial for security

LWE-based encryption:

- Security rests on hardness of decoding general linear code
- Encryption is into the noise, $\mathbf{c}^t = \mathbf{s}^t G + \mathbf{e}^t + (0, 0, ..., b)\lfloor q/2 \rfloor$
- Uses random general linear code
- Average case hardness is the same as hardest case

(Trapdoor) One-way functions

A (trapdoor) one-way function is a function that is easy to compute but computationally hard to reverse. Examples:

- RSA (factoring)
- Knapsack (NP-complete but insecure with trapdoor)
- Diffie-Hellman + ElGamal (discrete log)
- EC Diffie-Hellman + EC ElGamal (EC discrete log)
- Lattice crypto (LWE, SIS, DLWE)

We also have a strongly collision-resistant hash function

- Linear random hash mod q (SIS)