Post-quantum cryptography
Why?

Kristian Gjøsteen

Department of Mathematical Sciences, NTNU

Finse, May 2017

Background

I will use:

- Linear algebra.
  - Vectors $\mathbf{x}$.
  - Matrices $\mathbf{A}$, matrix multiplication $\mathbf{AB}$, $\mathbf{xA}$, $\mathbf{By}$.
  - Linear maps can be described by matrices.
- Abstract algebra.
  - Finite fields.
  - Polynomials, in one or several variables.
- Number theory.
  - Primes p.
  - Modular arithmetic.
- Complex numbers.
  - Polar coordinates.

Factoring

We want to factor an integer $n = pq$.

If we can factor integers quickly, we can break a lot of current cryptography.

If we can find (a multiple) of the period r of the function

$$f(x) = a^x \bmod n,$$

then we can factor n by computing the *gcd* of n and a power of a.

If we can find a fraction close to a multiple of $1/r$, then we can find r by finding a *rational approximation*, e.g. using *continued fractions*.

Factoring

We want to factor an integer $n = pq$.

If we can factor integers quickly, we can break a lot of current cryptography.

If we can find (a multiple) of the period r of the function

$$f(x) = a^x \bmod n,$$

then we can factor n by computing the gcd of n and a power of a.

If we can find a fraction close to a multiple of $1/r$, then we can find r by finding a *rational approximation*, e.g. using *continued fractions*.

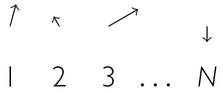
Finding fractions close to multiples of $1/r$ breaks a lot of current cryptography.

Quantum Computation

1 2 3 . . . N

— *Superposition*: a system can be in more than one state at the same time.

Quantum Computation



— *Superposition*: a system can be in more than one state at the same time.

— A system has a *complex amplitude* for each state, a kind of “probability”.

Quantum Computation

↖

2

— *Superposition*: a system can be in more than one state at the same time.

— A system has a *complex amplitude* for each state, a kind of “probability”.

— When we look at the system (*measure*), we see just one state.

— Which state we see is a random choice, each state chosen proportional to its complex amplitude (probability).

Quantum Computation

$$\begin{array}{ccccccccccc} \uparrow & \nwarrow & \nearrow & & \downarrow & & \uparrow & \nwarrow & \nearrow & & \downarrow \\ 1 & 2 & 3 & \dots & N & \longmapsto & a^1 \bmod n & a^2 \bmod n & a^3 \bmod n & \dots & a^N \bmod n \end{array}$$

— We can do things with the superpositions that affect each state individually, like computing a function (say $a^x \bmod n$). The result is a superposition of the function values.

Quantum Computation

$$\begin{array}{ccccc} \nearrow & \nwarrow & \nearrow & & \downarrow \\ 1 & 2 & 3 & \dots & N \end{array} \quad \longmapsto \quad \begin{array}{ccccc} \nearrow & \nwarrow & \nearrow & & \downarrow \\ a^1 \bmod n & a^2 \bmod n & a^3 \bmod n & \dots & a^N \bmod n \end{array}$$

— We can do things with the superpositions that affect each state individually, like computing a function (say $a^x \bmod n$). The result is a superposition of the function values.

— Yes, this is a massively parallel computation.

Quantum Computation

$$\begin{array}{ccc} \nwarrow & & \nwarrow \\ 2 & \longmapsto & a^2 \bmod n \end{array}$$

- We can do things with the superpositions that affect each state individually, like computing a function (say $a^x \bmod n$). The result is a superposition of the function values.
- Yes, this is a massively parallel computation.
  - Since we can only see one of the results, we cannot use this directly.

Quantum Computation

$$\overset{\uparrow}{1}\quad \overset{\nwarrow}{2}\quad \overset{\nearrow}{3}\quad \dots\quad \overset{\downarrow}{N} \qquad \longmapsto \qquad \overset{\uparrow}{a^1 \bmod n}\quad \overset{\nwarrow}{a^2 \bmod n}\quad \overset{\nearrow}{a^3 \bmod n}\quad \dots\quad \overset{\downarrow}{a^N \bmod n}$$

— We can do things with the superpositions that affect each state individually, like computing a function (say $a^x \bmod n$). The result is a superposition of the function values.

— Yes, this is a massively parallel computation.

- Since we can only see one of the results, we cannot use this directly.
- We have to manipulate the probabilities somehow…

Quantum Computation

$$\overset{\uparrow}{1} \quad \overset{\nwarrow}{2} \quad \overset{\nearrow}{3} \quad \dots \quad \overset{\downarrow}{N} \qquad \longmapsto \qquad \overset{\uparrow}{a^1 \bmod n} \quad \overset{\nwarrow}{a^2 \bmod n} \quad \overset{\nearrow}{a^3 \bmod n} \quad \dots \quad \overset{\downarrow}{a^N \bmod n}$$

- We can do things with the superpositions that affect each state individually, like computing a function (say $a^x \bmod n$). The result is a superposition of the function values.
- Yes, this is a massively parallel computation.
  - Since we can only see one of the results, we cannot use this directly.
  - We have to manipulate the probabilities somehow…
- **Quantum interference.**

Discrete Fourier Transform

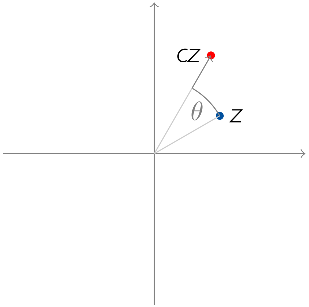
Given the numbers $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \ldots, \beta_{N-1}$ using the formula

$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i\, kj/N}.$$

Discrete Fourier Transform

Given the numbers $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \dots, \beta_{N-1}$ using the formula

$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i\, kj/N}.$$



Recall: a complex number's polar form is $c = ae^{i\theta}$.

Multiplication by this c is

— rotation by θ and

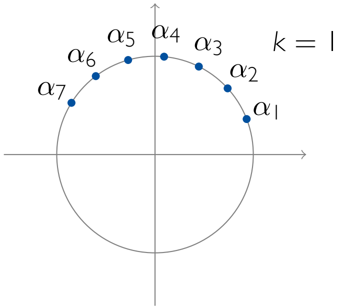
— scaling by a.

Multiplication by $e^{2\pi i\, \rho}$ is rotation by ρ full circles.

Discrete Fourier Transform

Given the numbers $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \dots, \beta_{N-1}$ using the formula

$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i\, kj/N}.$$

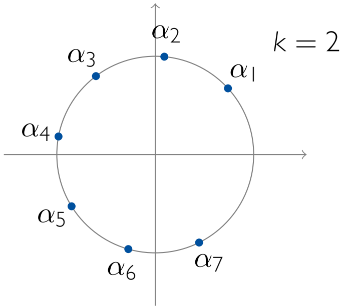


The elements of the α sequence are rotated before they are summed.

Discrete Fourier Transform

Given the numbers $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \dots, \beta_{N-1}$ using the formula

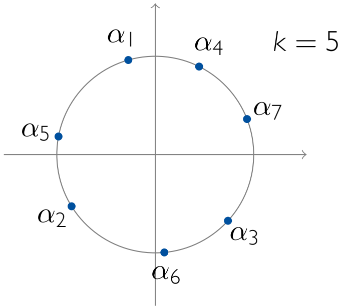
$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i\, kj/N}.$$



Discrete Fourier Transform

Given the numbers $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \ldots, \beta_{N-1}$ using the formula

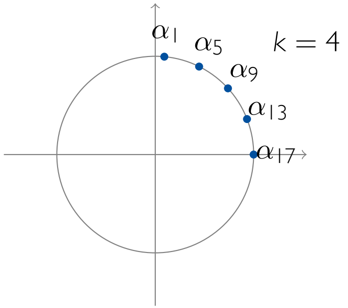
$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i\, kj/N}.$$



Discrete Fourier Transform

Given the numbers $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \ldots, \beta_{N-1}$ using the formula

$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i \, kj/N}.$$



When the α sequence has period r:

If k is close to a multiple of N/r

— they tend to reinforce each other.

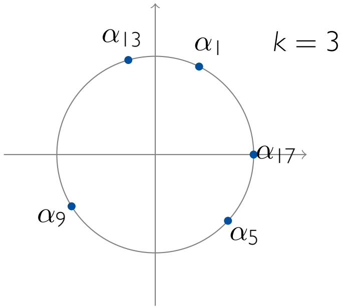
Otherwise

— they tend to cancel out.

Discrete Fourier Transform

Given the numbers $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \dots, \beta_{N-1}$ using the formula

$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i\, kj/N}.$$



When the α sequence has period r:

If k is close to a multiple of N/r

— they tend to reinforce each other.

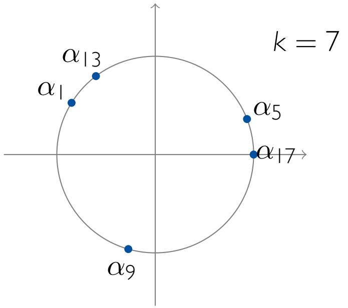
Otherwise

— they tend to cancel out.

Discrete Fourier Transform

Given the numbers $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \ldots, \beta_{N-1}$ using the formula

$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i\, kj/N}.$$



When the α sequence has period r:

If k is close to a multiple of N/r

— they tend to reinforce each other.

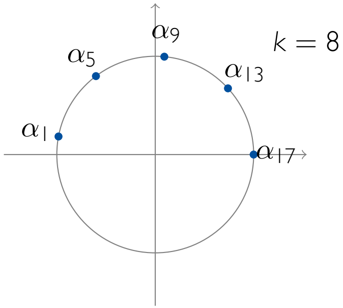
Otherwise

— they tend to cancel out.

Discrete Fourier Transform

Given the numbers $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}$, we compute $\beta_0, \beta_1, \ldots, \beta_{N-1}$ using the formula

$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} \alpha_j e^{2\pi i\, kj/N}.$$



When the α sequence has period r:

If k is close to a multiple of N/r

— they tend to reinforce each other.

Otherwise

— they tend to cancel out.

Shor's algorithm

We first organize a quantum computation resulting in a superposition of

$$u, u + r, u + 2r, u + 3r, \ldots$$

all equiprobable. All other states have zero probability of being measured.

Shor's algorithm

We first organize a quantum computation resulting in a superposition of

$$u, u+r, u+2r, u+3r, \dots$$

all equiprobable. All other states have zero probability of being measured.

- We compute a *quantum Fourier transform* (basically DFT on the probabilities):

$$0, 0, \dots, 0, \alpha_u, 0, \dots, 0, \alpha_{u+r}, 0, \dots \qquad \longmapsto \qquad \beta_0, \beta_1, \beta_2, \beta_3, \dots$$

Shor's algorithm

We first organize a quantum computation resulting in a superposition of

$$u, u+r, u+2r, u+3r, \ldots$$

all equiprobable. All other states have zero probability of being measured.

— We compute a *quantum Fourier transform* (basically DFT on the probabilities):

$$0, 0, \ldots, 0, \alpha_u, 0, \ldots, 0, \alpha_{u+r}, 0, \ldots \qquad \longmapsto \qquad \beta_0, \beta_1, \beta_2, \beta_3, \ldots$$

— Periodicity: states close to a multiple of N/r cause reinforcing and are more probable.

Shor's algorithm

We first organize a quantum computation resulting in a superposition of

$$u, u + r, u + 2r, u + 3r, \ldots$$

all equiprobable. All other states have zero probability of being measured.

— We compute a *quantum Fourier transform* (basically DFT on the probabilities):

$$0, 0, \ldots, 0, \alpha_u, 0, \ldots, 0, \alpha_{u+r}, 0, \ldots \qquad \longmapsto \qquad \beta_0, \beta_1, \beta_2, \beta_3, \ldots$$

— Periodicity: states close to a multiple of N/r cause reinforcing and are more probable.

— Other states cause cancellation and are less probable.

Shor's algorithm

We first organize a quantum computation resulting in a superposition of

$$u, u + r, u + 2r, u + 3r, \ldots$$

all equiprobable. All other states have zero probability of being measured.

— We compute a *quantum Fourier transform* (basically DFT on the probabilities):

$$0, 0, \ldots, 0, \alpha_u, 0, \ldots, 0, \alpha_{u+r}, 0, \ldots \qquad \longmapsto \qquad \beta_0, \beta_1, \beta_2, \beta_3, \ldots$$

— Periodicity: states close to a multiple of N/r cause reinforcing and are more probable.

— Other states cause cancellation and are less probable.

— When we measure, we will likely see something close to a multiple of N/r.

Shor's algorithm

We first organize a quantum computation resulting in a superposition of

$$u, u + r, u + 2r, u + 3r, \ldots$$

all equiprobable. All other states have zero probability of being measured.

— We compute a *quantum Fourier transform* (basically DFT on the probabilities):

$$0, 0, \ldots, 0, \alpha_u, 0, \ldots, 0, \alpha_{u+r}, 0, \ldots \qquad \longmapsto \qquad \beta_0, \beta_1, \beta_2, \beta_3, \ldots$$

— Periodicity: states close to a multiple of N/r cause reinforcing and are more probable.

— Other states cause cancellation and are less probable.

— When we measure, we will likely see something close to a multiple of N/r.

— Dividing by N, we get something close to a multiple of $1/r$.

Factoring

We want to factor an integer $n = pq$.

If we can factor integers quickly, we can break a lot of current cryptography.

If we can find (a multiple) of the period r of the function

$$f(x) = a^x \bmod n,$$

then we can factor n by computing the gcd of n and a power of a.

If we can find a fraction close to a multiple of $1/r$, then we can find r by finding a *rational approximation*, e.g. using *continued fractions*.

Finding fractions close to multiples of $1/r$ breaks a lot of current cryptography.

Grover's algorithm

Given $f : S \to \{0, 1\}$, *Grover's algorithm* finds $s \in S$ such that $f(s) = 1$ within $\sqrt{|S|}$ iterations.

For constants m and c, we can define a function f as

$$f(k) = \begin{cases} 1 & AES(k, m) = c, \\ 0 & \text{otherwise.} \end{cases}$$

In other words, Grover's algorithm can find a 128-bit AES key using only 2^{64} iterations.

Which is why AES has a 256-bit variant.

Can Quantum Computers be Built?

Wuantum computers have been built already.

Can Quantum Computers be Built?

Wuantum computers have been built already.

- But they are very small. They can only factor 15 or other very small numbers.

Can Quantum Computers be Built?

Wuantum computers have been built already.

- But they are very small. They can only factor 15 or other very small numbers.
- We do not know if it is possible to build a large enough quantum computer.

Can Quantum Computers be Built?

Wuantum computers have been built already.

- But they are very small. They can only factor 15 or other very small numbers.
- We do not know if it is possible to build a large enough quantum computer.
- We do not know if it is impossible to build such a computer.

Can Quantum Computers be Built?

Wuantum computers have been built already.

- But they are very small. They can only factor 15 or other very small numbers.
- We do not know if it is possible to build a large enough quantum computer.
- We do not know if it is impossible to build such a computer.
- We need some new cryptography.

Quantum Key Growth

Quantum cryptography is about using the properties of the physical universe to get security. The most famous example is the *key growth protocol*, often called quantum key distribution.

Quantum Key Growth

Quantum cryptography is about using the properties of the physical universe to get security. The most famous example is the *key growth protocol*, often called quantum key distribution.

In its simplest form, the underlying physical idea is that we can emit single photons with any polarization angle. But we can reliably measure polarization along just one orthogonal basis:

— If the photon is polarized along our orthogonal basis, we measure it correctly.

— Otherwise, we get a random measurement.

If you do not know in advance know which basis to use, you cannot measure the photon with certainty, so you cannot reliably copy the photon.

Quantum Key Growth

The protocol begins with a quantum phase:

- The sender emits a stream of single photons, either with 0° or 90° polarization, or with ±45° polarization. The sender remembers both the exact polarization and the choice.
- The receiver measures this stream of photons, orienting the detector so that it measures in either a 0°/90° basis or a ±45° basis, at random. The receiver remembers both the orientation of the receiver and the measurements.

Quantum Key Growth

The protocol begins with a quantum phase:

- The sender emits a stream of single photons, either with $0°$ or $90°$ polarization, or with $\pm 45°$ polarization. The sender remembers both the exact polarization and the choice.
- The receiver measures this stream of photons, orienting the detector so that it measures in either a $0°/90°$ basis or a $\pm 45°$ basis, at random. The receiver remembers both the orientation of the receiver and the measurements.

If nobody interfered, when the sender and receiver orientations coincide, the receiver should measure exactly what the sender sent.

If somebody looked at the sender's photons, the receiver will often not measure what the sender sent.

Quantum Key Growth

The protocol begins with a quantum phase:

- The sender emits a stream of single photons, either with 0° or 90° polarization, or with $\pm 45^\circ$ polarization. The sender remembers both the exact polarization and the choice.
- The receiver measures this stream of photons, orienting the detector so that it measures in either a $0^\circ/90^\circ$ basis or a $\pm 45^\circ$ basis, at random. The receiver remembers both the orientation of the receiver and the measurements.

The protocol continues with a first classical phase:

- The receiver reveals the orientation of his detector for each measurement.
- The sender reveals when their orientations were the same.

Quantum Key Growth

The protocol begins with a quantum phase:

- The sender emits a stream of single photons, either with 0° or 90° polarization, or with $\pm 45^\circ$ polarization. The sender remembers both the exact polarization and the choice.
- The receiver measures this stream of photons, orienting the detector so that it measures in either a $0^\circ/90^\circ$ basis or a $\pm 45^\circ$ basis, at random. The receiver remembers both the orientation of the receiver and the measurements.

The protocol continues with a first classical phase:

- The receiver reveals the orientation of his detector for each measurement.
- The sender reveals when their orientations were the same.

Finally:

- The receiver and the sender then use an error detection protocol to decide if the receiver measured exactly what the sender sent.

Quantum Key Growth

There is a “theorem” that says this can be done with information-theoretical security. Physical realisations have been insecure.

Quantum Key Growth

There is a “theorem” that says this can be done with information-theoretical security. Physical realisations have been insecure.

Warning: Opinions follow.

Quantum Key Growth

There is a “theorem” that says this can be done with information-theoretical security. Physical realisations have been insecure.

Warning: Opinions follow.

- Quantum key growth is impractical.
- We don’t need it in practice.

NIST process

We need:

— Public key encryption.

— Digital signatures.

— Key exchange.

NIST in the US has begun a process to find suitable post-quantum primitives and improve our understanding of their security. This is a multi-year process, and it will probably be at least a decade until implementations are common.

NIST process

We need:

- Public key encryption.
- Digital signatures.
- Key exchange.

NIST in the US has begun a process to find suitable post-quantum primitives and improve our understanding of their security. This is a multi-year process, and it will probably be at least a decade until implementations are common.

And that is if there are no surprises.

Types of post-quantum cryptography

It is commonly said that there are four types of post-quantum cryptography, based on:

- Hash functions
- Error-correcting codes
- Lattices
- Systems of multivariate polynomial equations

Some people also mention isogenies.

Hash-based digital signatures

Hash-based digital signatures have their origin in Lamport's one-time signature scheme. The underlying idea is that we have a hash function h that is one-way (hard to invert).

We choose n pairs of values $(x_{1,0}, x_{1,1}), (x_{2,0}, x_{2,1}), \dots, (x_{n,0}, x_{n_1})$. We compute $y_{i,j} = h(x_{i,j})$. The public key is the n pairs

$$(y_{1,0}, y_{1,1}), (y_{2,0}, y_{2,1}), \dots, (y_{n,0}, y_{n,1}).$$

The signature on a message $(m_1, m_2, \dots, m_n) \in \{0, 1\}^n$ is $(x_{1,m_1}, x_{2,m_2}, \dots, x_{n,m_n})$.

A signature $(x'_1, x'_2, \dots, x'_n)$ on a message $(m_1, m_2, \dots, m_n)$ is valid if $h(x'_i) = y_{i,m_i}$ for all i.

This scheme is trivially insecure if more than one message is signed.

We can use various tree-based structures to create reasonably efficient schemes that can sign more than one message.

Isogeny-based cryptography

Isogenies are “nice” maps between algebraic varieties.

We get a graph where the vertices are algebraic varieties and the edges are isogenies.

Finding paths between given points in this graph seems to be difficult.

It is easy to create hash functions and Diffie-Hellman analogues.

Isogeny-based cryptography is somewhat obscure.

Post-quantum cryptography
Error-correcting codes

Kristian Gjøsteen

Department of Mathematical Sciences, NTNU

Finse, May 2017

Background: Error correction

We want to transmit a *message* $\mathbf{m}$ through an *unreliable channel*.

The unreliable channel introduces *random errors* to symbols sent through it:

— $\mathbf{m}$ goes in, $\mathbf{y}$ comes out, but $\mathbf{y}$ may be different from $\mathbf{m}$.

— $\mathbf{e} = \mathbf{y} - \mathbf{m}$ is called the *error vector*.

This is bad.

Background: Error correction

We want to transmit a *message* $\mathbf{m}$ through an *unreliable channel*.

The unreliable channel introduces *random errors* to symbols sent through it:

— $\mathbf{m}$ goes in, $\mathbf{y}$ comes out, but $\mathbf{y}$ may be different from $\mathbf{m}$.

— $\mathbf{e} = \mathbf{y} - \mathbf{m}$ is called the *error vector*.

This is bad.

The idea is to encode messages as longer, distinct *code words*. The recipient receives something that is similar, but not quite the same as the sent code word. He must then decide what the most likely message sent was.

Background: Error correction

We want to transmit a *message* $\mathbf{m}$ through an *unreliable channel*.

The unreliable channel introduces *random errors* to symbols sent through it:

— $\mathbf{m}$ goes in, $\mathbf{y}$ comes out, but $\mathbf{y}$ may be different from $\mathbf{m}$.

— $\mathbf{e} = \mathbf{y} - \mathbf{m}$ is called the *error vector*.

This is bad.

The idea is to encode messages as longer, distinct *code words*. The recipient receives something that is similar, but not quite the same as the sent code word. He must then decide what the most likely message sent was.

If we can encode in such a way that any two code words can be distinguished, even after a few changes have been introduced, the recipient simply finds the code word that is "closest" to whatever was received and then inverts the encoding map to recover the message.

Block codes

A *block code* is a set $\mathcal{C} \subseteq \mathbb{F}^n$ of *code words*.

Block codes

A *block code* is a set $\mathcal{C} \subseteq \mathbb{F}^n$ of *code words*.

We have an *encoding function* $\mathbf{G}$ from $\mathbb{F}^k$ into $\mathcal{C}$. We want k to be large relative to n.

Block codes

A *block code* is a set $\mathcal{C} \subseteq \mathbb{F}^n$ of *code words*.

We have an *encoding function* $\mathbf{G}$ from $\mathbb{F}^k$ into $\mathcal{C}$. We want k to be large relative to n.

The *Hamming distance* $\mathrm{d}(\mathbf{c}, \mathbf{y})$ of two vectors counts the number of differences.

Block codes

A *block code* is a set $\mathcal{C} \subseteq \mathbb{F}^n$ of *code words*.

We have an *encoding function* $\mathbf{G}$ from $\mathbb{F}^k$ into $\mathcal{C}$. We want k to be large relative to n.

The *Hamming distance* $\mathrm{d}(\mathbf{c}, \mathbf{y})$ of two vectors counts the number of differences.

The *minimum distance* d of a code $\mathcal{C}$ is the minimum distance between two distinct code words. We want d to be large.

Block codes

A *block code* is a set $\mathcal{C} \subseteq \mathbb{F}^n$ of *code words*.

We have an *encoding function* $\mathbf{G}$ from $\mathbb{F}^k$ into $\mathcal{C}$. We want k to be large relative to n.

The *Hamming distance* $\mathrm{d}(\mathbf{c}, \mathbf{y})$ of two vectors counts the number of differences.

The *minimum distance* d of a code $\mathcal{C}$ is the minimum distance between two distinct code words. We want d to be large.

The *nearest neighbour decoding* is about finding the codeword $\mathbf{c}$ closest to $\mathbf{y}$ and then inverting $\mathbf{G}$.

Block codes

A *block code* is a set $\mathcal{C} \subseteq \mathbb{F}^n$ of *code words*.

We have an *encoding function* $\mathbf{G}$ from $\mathbb{F}^k$ into $\mathcal{C}$. We want k to be large relative to n.

The *Hamming distance* $\mathrm{d}(\mathbf{c}, \mathbf{y})$ of two vectors counts the number of differences.

The *minimum distance* d of a code $\mathcal{C}$ is the minimum distance between two distinct code words. We want d to be large.

The *nearest neighbour decoding* is about finding the codeword $\mathbf{c}$ closest to $\mathbf{y}$ and then inverting $\mathbf{G}$.

In general, all of this is impractical or plain impossible.

Linear block codes

A *linear block code* is a subspace $\mathcal{C}$ of $\mathbb{F}^n$. Our encoding function $\mathbf{G} : \mathbb{F}^k \to \mathcal{C}$ is a linear function, which means that $\mathbf{G}$ can be described by a matrix, a *generator matrix*.

So the encoding function maps $\mathbf{m}$ to the code word $\mathbf{c} = \mathbf{mG}$.

The *weight* of a code word is the number of non-zero coordinates. The distance between two code words is the weight of their difference.

Linear block codes

A *linear block code* is a subspace $\mathcal{C}$ of $\mathbb{F}^n$. Our encoding function $\mathbf{G} : \mathbb{F}^k \to \mathcal{C}$ is a linear function, which means that $\mathbf{G}$ can be described by a matrix, a *generator matrix*.

So the encoding function maps $\mathbf{m}$ to the code word $\mathbf{c} = \mathbf{mG}$.

The *weight* of a code word is the number of non-zero coordinates. The distance between two code words is the weight of their difference.

A generator matrix may be systematic, in which case the message symbols are included "as-is" in the code word.

Linear block codes

A *linear block code* is a subspace $\mathcal{C}$ of $\mathbb{F}^n$. Our encoding function $\mathbf{G} : \mathbb{F}^k \to \mathcal{C}$ is a linear function, which means that $\mathbf{G}$ can be described by a matrix, a *generator matrix*.

So the encoding function maps $\mathbf{m}$ to the code word $\mathbf{c} = \mathbf{mG}$.

The *weight* of a code word is the number of non-zero coordinates. The distance between two code words is the weight of their difference.

A generator matrix may be systematic, in which case the message symbols are included "as-is" in the code word.

If the generator matrix is non-systematic, how can we invert the encoding map?

Information set

Let $\mathcal{C}$ be a linear code with generator matrix $\mathbf{G}$. Given $\mathbf{c} \in \mathcal{C}$, find $\mathbf{m}$ such that $\mathbf{c} = \mathbf{mG}$.

Information set

Let $\mathcal{C}$ be a linear code with generator matrix $\mathbf{G}$. Given $\mathbf{c} \in \mathcal{C}$, find $\mathbf{m}$ such that $\mathbf{c} = \mathbf{mG}$.

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_k\}$ be a subset of $\{1, 2, \ldots, n\}$. We define the projection map $\mathbf{N}_{\mathcal{I}}$ taking $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ to $(c_{i_1}, c_{i_2}, \ldots, c_{i_k})$.

Information set

Let $\mathcal{C}$ be a linear code with generator matrix $\mathbf{G}$. Given $\mathbf{c} \in \mathcal{C}$, find $\mathbf{m}$ such that $\mathbf{c} = \mathbf{m}\mathbf{G}$.

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_k\}$ be a subset of $\{1, 2, \ldots, n\}$. We define the projection map $\mathbf{N}_{\mathcal{I}}$ taking $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ to $(c_{i_1}, c_{i_2}, \ldots, c_{i_k})$. The action on a matrix is to select a set of columns:

$$\mathbf{G}\mathbf{N}_{\mathcal{I}} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix} \mathbf{N}_{\mathcal{I}} = \begin{bmatrix} g_{1i_1} & g_{1i_2} & \cdots & g_{1i_k} \\ g_{2i_1} & g_{2i_2} & \cdots & g_{2i_k} \\ \vdots & \vdots & & \vdots \\ g_{ki_1} & g_{ki_2} & \cdots & g_{ki_k} \end{bmatrix}.$$

Information set

Let $\mathcal{C}$ be a linear code with generator matrix $\mathbf{G}$. Given $\mathbf{c} \in \mathcal{C}$, find $\mathbf{m}$ such that $\mathbf{c} = \mathbf{mG}$.

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_k\}$ be a subset of $\{1, 2, \ldots, n\}$. We define the projection map $\mathbf{N}_{\mathcal{I}}$ taking $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ to $(c_{i_1}, c_{i_2}, \ldots, c_{i_k})$. The action on a matrix is to select a set of columns:

$$\mathbf{GN}_{\mathcal{I}} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix} \mathbf{N}_{\mathcal{I}} = \begin{bmatrix} g_{1i_1} & g_{1i_2} & \cdots & g_{1i_k} \\ g_{2i_1} & g_{2i_2} & \cdots & g_{2i_k} \\ \vdots & \vdots & & \vdots \\ g_{ki_1} & g_{ki_2} & \cdots & g_{ki_k} \end{bmatrix}.$$

$\mathcal{I}$ is an *information set* if $\mathbf{GN}_{\mathcal{I}}$ is an invertible matrix.

Information set

Let $\mathcal{C}$ be a linear code with generator matrix $\mathbf{G}$. Given $\mathbf{c} \in \mathcal{C}$, find $\mathbf{m}$ such that $\mathbf{c} = \mathbf{mG}$.

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_k\}$ be a subset of $\{1, 2, \ldots, n\}$. We define the projection map $\mathbf{N}_{\mathcal{I}}$ taking $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ to $(c_{i_1}, c_{i_2}, \ldots, c_{i_k})$. The action on a matrix is to select a set of columns:

$$\mathbf{GN}_{\mathcal{I}} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix} \mathbf{N}_{\mathcal{I}} = \begin{bmatrix} g_{1i_1} & g_{1i_2} & \cdots & g_{1i_k} \\ g_{2i_1} & g_{2i_2} & \cdots & g_{2i_k} \\ \vdots & \vdots & & \vdots \\ g_{ki_1} & g_{ki_2} & \cdots & g_{ki_k} \end{bmatrix}.$$

$\mathcal{I}$ is an *information set* if $\mathbf{GN}_{\mathcal{I}}$ is an invertible matrix. In which case, if $\mathbf{M}$ is an inverse of $\mathbf{GN}_{\mathcal{I}}$, then

$$\mathbf{cN}_{\mathcal{I}}\mathbf{M} = \mathbf{mGN}_{\mathcal{I}}\mathbf{M} = \mathbf{m}.$$

Different generator matrices

Our linear code $\mathcal{C}$ is a subspace of dimension k.

There are many maps from $\mathbb{F}^k$ into $\mathcal{C}$. In fact, if we have any map $\mathbf{G}$ and any $k \times k$ invertible matrix $\mathbf{S}$, the matrix $\mathbf{SG}$ describes another map from $\mathbb{F}^k$ into $\mathcal{C}$.

Different generator matrices

Our linear code $\mathcal{C}$ is a subspace of dimension k.

There are many maps from $\mathbb{F}^k$ into $\mathcal{C}$. In fact, if we have any map $\mathbf{G}$ and any $k \times k$ invertible matrix $\mathbf{S}$, the matrix $\mathbf{SG}$ describes another map from $\mathbb{F}^k$ into $\mathcal{C}$.

In other words, if $\mathbf{G}$ is a generator matrix for $\mathcal{C}$ and $\mathbf{S}$ is an invertible matrix, then $\mathbf{SG}$ is also a (different) generator matrix for $\mathcal{C}$.

Permutation-equivalent codes

When are two codes “the same”?

Permutation-equivalent codes

When are two codes "the same"?

Codes are vector spaces. We know that two vector spaces are "the same" if we have an invertible linear map between them (a vector space isomorphism).

But codes are not just vector spaces, since we very much care about the minimum distance of codes. And isomorphic vector spaces can have very different minimum distances, so as codes they will be very different.

Permutation-equivalent codes

When are two codes "the same"?

Some invertible linear maps do not change the weight of vectors. For example, if we permute the order of the coordinates in the code words:

$$(c_1, c_2, c_3, c_4, c_5) \longmapsto (c_3, c_1, c_5, c_4, c_2).$$

Permutation matrices describe exactly these linear maps.

Permutation-equivalent codes

When are two codes "the same"?

For any code $\mathcal{C}$ and any $n \times n$ permutation matrix $\mathbf{P}$,

$$\mathcal{C}\mathbf{P} = \{\mathbf{cP} \mid \mathbf{c} \in \mathcal{C}\}$$

is a code with the same dimension and the same minimum distance as $\mathcal{C}$, a code that is in some sense *equivalent* to $\mathcal{C}$.

Permutation-equivalent codes

When are two codes “the same”?

For any code $\mathcal{C}$ and any $n \times n$ permutation matrix $\mathbf{P}$,

$$\mathcal{C}\mathbf{P} = \{\mathbf{cP} \mid \mathbf{c} \in \mathcal{C}\}$$

is a code with the same dimension and the same minimum distance as $\mathcal{C}$, a code that is in some sense *equivalent* to $\mathcal{C}$.

If $\mathcal{C}$ has generator matrix $\mathbf{G}$, then $\mathcal{C}\mathbf{P}$ has generator matrix $\mathbf{GP}$.

Generalized Reed-Solomon codes

Let $\mathbb{F}_q$ be a finite field with $q > n$, let $\alpha_1, \alpha_2, \dots, \alpha_n$ be distinct, non-zero field elements, let $\beta_1, \beta_2, \dots, \beta_n$ be non-zero field elements, and let $\mathcal{F}$ be the polynomials of degree less than k.

We define the *Generalized Reed-Solomon code* defined by $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ to be the code

$$\mathcal{C} = \{(\beta_1 f(\alpha_1), \beta_2 f(\alpha_2), \dots, \beta_n f(\alpha_n)) \mid f(X) \in \mathcal{F}\} \subseteq \mathbb{F}_q^n.$$

It is easy to show that this code has dimension k and minimum distance $n - k + 1$.

You can get a generator matrix from the vectors associated to the monomials $1, X, X^2, \dots, X^{k-1}$.

(These codes meet the singleton bound, so they are MDS codes.)

Goppa codes

Let $q = 2^r \geq n$, let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be distinct field elements, let $g(X)$ be an irreducible polynomial over $\mathbb{F}_q$ of degree t, and let $\mathcal{F}$ be the polynomials of degree less than k.

Let

$$\tilde{\mathcal{C}} = \{(\beta_1 f(\alpha_1), \beta_2 f(\alpha_2), \ldots, \beta_n f(\alpha_n)) \in \mathbb{F}_q \mid f(X) \in \mathcal{F}\},$$

where $\beta_i = g(\alpha_i) \prod_{j=1, j\neq i}^{n} (\alpha_i - \alpha_j)$. Then our binary *Goppa code* $\mathcal{C}$ defined by $(\boldsymbol{\alpha}, g(X))$ is the $\mathbb{F}_2$ subfield code of $\tilde{\mathcal{C}}$.

Goppa codes

Let $q = 2^r \geq n$, let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be distinct field elements, let $g(X)$ be an irreducible polynomial over $\mathbb{F}_q$ of degree t, and let $\mathcal{F}$ be the polynomials of degree less than k.

A more convenient description is

$$\mathcal{C} = \left\{ (c_1, c_2, \ldots, c_n) \in \mathbb{F}_2^n \;\middle|\; \sum_{i=1}^{n} \frac{c_i}{X - \alpha_i} \equiv 0 \pmod{g(X)} \right\}.$$

It can be shown that this code has

— dimension at least $n - tr$, and

— minimum distance least $2t + 1$.

Decoding problem

The *decoding problem* is to find the nearest codeword to a given vector.

For general block codes, the decoding problem is impossible. (This doesn't matter, because even encoding is impossible.)

For random linear block codes, the decoding problem is merely very difficult (NP-complete).

Information set decoding

We have $\mathbf{y} = \mathbf{c} + \mathbf{e}$. We want to find $\mathbf{c}$.

Choose an information set $\mathcal{I} \subseteq \{1, 2, \dots, n\}$, such that $\mathbf{G}\mathbf{N}_\mathcal{I}$ is invertible with inverse $\mathbf{M}$. Compute

$$\mathbf{z} = \mathbf{y}\mathbf{N}_\mathcal{I}\mathbf{M}\mathbf{G} = \mathbf{c}\mathbf{N}_\mathcal{I}\mathbf{M}\mathbf{G} + \mathbf{e}\mathbf{N}_\mathcal{I}\mathbf{M}\mathbf{G} = \mathbf{c} + \mathbf{e}\mathbf{N}_\mathcal{I}\mathbf{M}\mathbf{G}.$$

If we are lucky, $\mathbf{e}\mathbf{N}_\mathcal{I} = \mathbf{0}$, and we get that $\mathrm{d}(\mathbf{y}, \mathbf{z})$ is small and therefore that $\mathbf{z} = \mathbf{c}$.

Information set decoding

We have $\mathbf{y} = \mathbf{c} + \mathbf{e}$. We want to find $\mathbf{c}$.

Choose an information set $\mathcal{I} \subseteq \{1, 2, \ldots, n\}$, such that $\mathbf{G}\mathbf{N}_{\mathcal{I}}$ is invertible with inverse $\mathbf{M}$. Compute

$$\mathbf{z} = \mathbf{y}\mathbf{N}_{\mathcal{I}}\mathbf{M}\mathbf{G} = \mathbf{c}\mathbf{N}_{\mathcal{I}}\mathbf{M}\mathbf{G} + \mathbf{e}\mathbf{N}_{\mathcal{I}}\mathbf{M}\mathbf{G} = \mathbf{c} + \mathbf{e}\mathbf{N}_{\mathcal{I}}\mathbf{M}\mathbf{G}.$$

If we are lucky, $\mathbf{e}\mathbf{N}_{\mathcal{I}} = \mathbf{0}$, and we get that $\mathrm{d}(\mathbf{y}, \mathbf{z})$ is small and therefore that $\mathbf{z} = \mathbf{c}$. Otherwise, try a new information set.

Information set decoding

We have $\mathbf{y} = \mathbf{c} + \mathbf{e}$. We want to find $\mathbf{c}$.

Choose an information set $\mathcal{I} \subseteq \{1, 2, \dots, n\}$, such that $\mathbf{G}\mathbf{N}_{\mathcal{I}}$ is invertible with inverse $\mathbf{M}$. Compute

$$\mathbf{z} = \mathbf{y}\mathbf{N}_{\mathcal{I}}\mathbf{M}\mathbf{G} = \mathbf{c}\mathbf{N}_{\mathcal{I}}\mathbf{M}\mathbf{G} + \mathbf{e}\mathbf{N}_{\mathcal{I}}\mathbf{M}\mathbf{G} = \mathbf{c} + \mathbf{e}\mathbf{N}_{\mathcal{I}}\mathbf{M}\mathbf{G}.$$

If we are lucky, $\mathbf{e}\mathbf{N}_{\mathcal{I}} = \mathbf{0}$, and we get that $\mathrm{d}(\mathbf{y}, \mathbf{z})$ is small and therefore that $\mathbf{z} = \mathbf{c}$. Otherwise, try a new information set.

With t errors, the odds of choosing an information set that does not contain an error is $\binom{n-t}{k} / \binom{n}{k}$, so the expected number of information sets we have to try before we get lucky is

$$\frac{\binom{n}{k}}{\binom{n-t}{k}} = \frac{n!(n-t-k)!k!}{(n-t)!(n-k)!k!} = \frac{n(n-1)\dots(n-t+1)}{(n-k)(n-k-1)\dots(n-k-t+1)}$$

$$\geq \left(\frac{n}{n-k}\right)^{t} = \left(1 - \frac{k}{n}\right)^{-t}.$$

Finding low-weight code words

Again, $\mathbf{y} = \mathbf{c} + \mathbf{e}$.

Consider the code $\mathcal{C}'$ generated by $\mathcal{C} \cup \{\mathbf{y}\}$.

This code has a single code word $\mathbf{e} = \mathbf{y} - \mathbf{c}$ of weight t.

If we can find a low-weight code word in $\mathcal{C}'$, we have found the error.

Finding low-weight code words

Again, $\mathbf{y} = \mathbf{c} + \mathbf{e}$.

Consider the code $\mathcal{C}'$ generated by $\mathcal{C} \cup \{\mathbf{y}\}$.

This code has a single code word $\mathbf{e} = \mathbf{y} - \mathbf{c}$ of weight t.

If we can find a low-weight code word in $\mathcal{C}'$, we have found the error.

These algorithms are good, but not good enough.

Decoding problem

The *decoding problem* is to find the nearest codeword to a given vector.

For general block codes, the decoding problem is impossible. (This doesn't matter, because even encoding is impossible.)

For random linear block codes, the decoding problem is merely very difficult (NP-complete).

However, it is not hard for every linear block code.

Decoding Generalized Reed-Solomon codes

GRS codes are equivalent to Reed-Solomon codes.

Reed-Solomon codes can be efficiently decoded e.g. by using the theory for BCH codes.

Decoding Goppa codes

Again, $\mathbf{y} = \mathbf{c} + \mathbf{e}$, with $\mathrm{wt}(\mathbf{e}) \leq t$.

1. Compute: $s(X) = \sum_{i=1}^{n} \frac{y_i}{X-\alpha_i} \bmod g(X) = \sum_{i=1}^{n} \frac{e_i}{X-\alpha_i} \bmod g(X)$.
2. Find $v(X)$ s.t. $v(X)^2 \equiv \frac{1}{s(X)} - X \pmod{g(X)}$.
3. Use a "half-way" extended Euclidian algorithm to find $a(X)$ and $b(X)$ s.t.

$$a(X) \equiv b(X)v(X) \pmod{g(X)} \qquad \deg a(X) \leq t/2 \text{ and } \deg b(X) < t/2.$$

Then $\sigma(X) = a(X)^2 + X\,b(X)^2$ is an *error locator polynomial*: $e_i = 1 \iff \sigma(\alpha_i) = 0$.

Decoding Goppa codes

Again, $\mathbf{y} = \mathbf{c} + \mathbf{e}$, with $\mathrm{wt}(\mathbf{e}) \leq t$.

1. Compute: $s(X) = \sum_{i=1}^{n} \frac{y_i}{X-\alpha_i} \bmod g(X) = \sum_{i=1}^{n} \frac{e_i}{X-\alpha_i} \bmod g(X)$.
2. Find $v(X)$ s.t. $v(X)^2 \equiv \frac{1}{s(X)} - X \pmod{g(X)}$.
3. Use a "half-way" extended Euclidian algorithm to find $a(X)$ and $b(X)$ s.t.

$$a(X) \equiv b(X)v(X) \pmod{g(X)} \qquad \deg a(X) \leq t/2 \text{ and } \deg b(X) < t/2.$$

Then $\sigma(X) = a(X)^2 + X\,b(X)^2$ is an *error locator polynomial*: $e_i = 1 \iff \sigma(\alpha_i) = 0$.

Why? Note that $\sigma'(X) = b(X)^2$, so modulo $g(X)$

$$\frac{\sigma(X)}{\sigma'(X)} \equiv \frac{a(X)^2}{b(X)^2} + X \equiv v(X)^2 + X \equiv \frac{1}{s(X)} - X + X \equiv \frac{1}{s(X)} \equiv \frac{\prod_{\substack{i \\ e_i=1}} (X - \alpha_i)}{\text{something}}.$$

First attempt at code-based cryptography

We will first try to do secret-key cryptography.

Choose a random code from some suitable family. Choose a generator matrix $\mathbf{G}$ for the code with an information set $\mathcal{I}$.

First attempt at code-based cryptography

We will first try to do secret-key cryptography.

Choose a random code from some suitable family. Choose a generator matrix $\mathbf{G}$ for the code with an information set $\mathcal{I}$.

We encrypt a message $\mathbf{m}$ by encoding the message as $\mathbf{c} = \mathbf{m}\mathbf{G}$, and then add a random error to it, so our ciphertext is

$$\mathbf{y} = \mathbf{c} + \mathbf{e}.$$

First attempt at code-based cryptography

We will first try to do secret-key cryptography.

Choose a random code from some suitable family. Choose a generator matrix $\mathbf{G}$ for the code with an information set $\mathcal{I}$.

We encrypt a message $\mathbf{m}$ by encoding the message as $\mathbf{c} = \mathbf{m}\mathbf{G}$, and then add a random error to it, so our ciphertext is

$$\mathbf{y} = \mathbf{c} + \mathbf{e}.$$

We decrypt by finding the nearest code word $\mathbf{z}$ to $\mathbf{y}$, and then compute

$$\mathbf{m} = \mathbf{z}(\mathbf{G}\mathbf{N}_{\mathcal{I}})^{-1}.$$

First attempt at code-based cryptography

We will first try to do secret-key cryptography.

Choose a random code from some suitable family. Choose a generator matrix $\mathbf{G}$ for the code with an information set $\mathcal{I}$.

We encrypt a message $\mathbf{m}$ by encoding the message as $\mathbf{c} = \mathbf{mG}$, and then add a random error to it, so our ciphertext is

$$\mathbf{y} = \mathbf{c} + \mathbf{e}.$$

We decrypt by finding the nearest code word $\mathbf{z}$ to $\mathbf{y}$, and then compute

$$\mathbf{m} = \mathbf{z}(\mathbf{GN}_{\mathcal{I}})^{-1}.$$

Note: If $\mathbf{G}$ is systematic, most of $\mathbf{m}$ will be plainly visible in the ciphertext.

First attempt at code-based cryptography

We will probably not use this as a general encryption scheme. Instead we will use it as a *key encapsulation mechanism* (KEM):

- Encrypt randomness.
- Hash the randomness to get a symmetric key. (We may want to hash the error vector too.)
- Encrypt the message with the symmetric key.

McEliece’s idea

Idea: We have a “nice” secret code that we can decode. We give away a “not-so-nice” generator matrix for an equivalent code, which is hard to decode.

McEliece’s idea

We have a “nice” code $\mathcal{C}$ with a generator matrix $\mathbf{G}$. Choose an invertible matrix $\mathbf{S}$ and a permutation matrix $\mathbf{P}$, both random. Let $\mathbf{G}' = \mathbf{SGP}$, which is a random generator matrix for an equivalent code $\mathcal{C}'$.

McEliece's idea

We have a "nice" code $\mathcal{C}$ with a generator matrix $\mathbf{G}$. Choose an invertible matrix $\mathbf{S}$ and a permutation matrix $\mathbf{P}$, both random. Let $\mathbf{G}' = \mathbf{SGP}$, which is a random generator matrix for an equivalent code $\mathcal{C}'$.

The sender has $\mathbf{G}'$ and encrypts a message $\mathbf{m}$ as

$$\mathbf{y} = \mathbf{m}\mathbf{G}' + \mathbf{e}$$

where $\mathbf{e}$ has weight t.

McEliece's idea

We have a "nice" code $\mathcal{C}$ with a generator matrix $\mathbf{G}$. Choose an invertible matrix $\mathbf{S}$ and a permutation matrix $\mathbf{P}$, both random. Let $\mathbf{G}' = \mathbf{SGP}$, which is a random generator matrix for an equivalent code $\mathcal{C}'$.

The sender has $\mathbf{G}'$ and encrypts a message $\mathbf{m}$ as

$$\mathbf{y} = \mathbf{mG}' + \mathbf{e}$$

where $\mathbf{e}$ has weight t.

Now $\mathbf{y}$ is close to a code word in $\mathcal{C}'$, which we cannot decode. However,

$$\mathbf{yP}^{-1} = \mathbf{mSGPP}^{-1} + \mathbf{eP}^{-1} = (\mathbf{mS})\mathbf{G} + \mathbf{eP}^{-1}.$$

This is now an encoding of the message $\mathbf{mS}$ under $\mathbf{G}$. The errors have changed positions, but we still have the same number of errors.

McEliece's idea

We have a "nice" code $\mathcal{C}$ with a generator matrix $\mathbf{G}$. Choose an invertible matrix $\mathbf{S}$ and a permutation matrix $\mathbf{P}$, both random. Let $\mathbf{G}' = \mathbf{SGP}$, which is a random generator matrix for an equivalent code $\mathcal{C}'$.

The sender has $\mathbf{G}'$ and encrypts a message $\mathbf{m}$ as

$$\mathbf{y} = \mathbf{m}\mathbf{G}' + \mathbf{e}$$

where $\mathbf{e}$ has weight t.

We decode $\mathbf{y}\mathbf{P}^{-1} \in \mathcal{C}$ to get $\mathbf{m}' = \mathbf{mS}$ (and probably $\mathbf{e}\mathbf{P}^{-1}$), from which we recover $\mathbf{m}$ (and probably $\mathbf{e}$).

McEliece's idea

We have a "nice" code $\mathcal{C}$ with a generator matrix $\mathbf{G}$. Choose an invertible matrix $\mathbf{S}$ and a permutation matrix $\mathbf{P}$, both random. Let $\mathbf{G}' = \mathbf{SGP}$, which is a random generator matrix for an equivalent code $\mathcal{C}'$.

Why should this be secure?

Hopefully, $\mathbf{G}'$ looks like a random linear code.

We know that random linear codes are hard to decode. So to the extent that $\mathbf{G}'$ looks like a random linear code, this should be secure.

Can we use Generalized Reed-Solomon codes?

The dimension of the square code is the same for all permutation-equivalent codes.

It turns out that if $\mathcal{C}$ is a Generalized Reed-Solomon code, the square code has fairly low dimension.

For random linear codes, the square code has fairly high dimension.

In other words, the Generalized Reed-Solomon codes do not look like random linear codes, even when described by a random generator matrix.

In fact, parameters for the Generalized Reed-Solomon code can be recovered from a random generator matrix.

What about Goppa codes?

There is no proof that Goppa codes look like random linear codes. Or that the McEliece idea is secure when used with Goppa codes.

But so far, nobody has broken McEliece with Goppa codes.

However, general decoding algorithms have improved, so old parameter sets have now become insecure.

Post-quantum cryptography
Lattices

Kristian Gjøsteen

Department of Mathematical Sciences, NTNU

Finse, May 2017

Subset sum

Given a list of positive integers $s_1, s_2, \ldots, s_n$ find out which that sum to a given integer z.

Subset sum

Given a list of positive integers $s_1, s_2, \ldots, s_n$ find out which that sum to a given integer z.

Alternative: Find $a_1, \ldots, a_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} a_i s_i = z$.

Subset sum

Given a list of positive integers $s_1, s_2, \ldots, s_n$ find out which that sum to a given integer z.

Alternative: Find $a_1, \ldots, a_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} a_i s_i = z$.

Note that this solution satisfies

$$(a_1, a_2, \ldots, a_n, 1)\mathbf{B} = (a_1, a_2, \ldots, a_n, 0) \qquad \text{where } \mathbf{B} = \begin{pmatrix} 1 & & & s_1 \\ & \ddots & & \vdots \\ & & 1 & s_n \\ & & & -z \end{pmatrix}.$$

Subset sum

Given a list of positive integers $s_1, s_2, \ldots, s_n$ find out which that sum to a given integer z.

Alternative: Find $a_1, \ldots, a_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} a_i s_i = z$.

Note that this solution satisfies

$$(a_1, a_2, \ldots, a_n, 1)\mathbf{B} = (a_1, a_2, \ldots, a_n, 0) \qquad \text{where } \mathbf{B} = \begin{pmatrix} 1 & & & s_1 \\ & \ddots & & \vdots \\ & & 1 & s_n \\ & & & -z \end{pmatrix}.$$

"Most" sums involving $s_1, s_2, \ldots, s_n$ are big, so "most" *integer linear combinations* of the rows in $\mathbf{B}$ are long vectors, while the solution $(a_1, a_2, \ldots, a_n, 0)$ we want is short.

Subset sum

Given a list of positive integers $s_1, s_2, \dots, s_n$ find out which that sum to a given integer z.

Alternative: Find $a_1, \dots, a_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} a_i s_i = z$.

Note that this solution satisfies

$$(a_1, a_2, \dots, a_n, 1)\mathbf{B} = (a_1, a_2, \dots, a_n, 0) \qquad \text{where } \mathbf{B} = \begin{pmatrix} 1 & & & s_1 \\ & \ddots & & \vdots \\ & & 1 & s_n \\ & & & -z \end{pmatrix}.$$

"Most" sums involving $s_1, s_2, \dots, s_n$ are big, so "most" *integer linear combinations* of the rows in $\mathbf{B}$ are long vectors, while the solution $(a_1, a_2, \dots, a_n, 0)$ we want is short.

If we can find short integer linear combinations, there is a reasonable possibility that $(a_1, a_2, \dots, a_n, 0)$ is among them.

Small roots

We have a polynomial $f(X) = \sum_{i=0}^{d} f_i X^i$ which has a zero x_0 modulo N med $|x_0| < T$. We want to find x_0.

Small roots

We have a polynomial $f(X) = \sum_{i=0}^{d} f_i X^i$ which has a zero x_0 modulo N med $|x_0| < T$. We want to find x_0.

If we can find an integer linear combination

$$h(X) = \sum_{i=0}^{d} h_i X^i = cf(X) + \sum_{i=0}^{d-1} a_i N X^i$$

such that $\sum_i |h_i| T^i \leq N$, then

$$h(x_0) = 0$$

because $h(x_0) \equiv 0 \pmod{N}$ and $|h(x_0)| \leq \sum_i |h_i||x_0|^i \leq \sum_i |h_i| T^i < N$.

Now we can use Newton's method to find x_0.

Small roots

We have a polynomial $f(X) = \sum_{i=0}^{d} f_i X^i$ which has a zero x_0 modulo N med $|x_0| < T$. We want to find x_0.

We want to find $h(X) = cf(X) + \sum_{i=0}^{d-1} a_i N X^i$ such that $h(x_0) = 0$ over the integers.

Look at the matrix

$$\mathbf{B} = \begin{pmatrix} N & & & & \\ & NT & & & \\ & & NT^2 & & \\ & & & \ddots & \\ f_0 & f_1 T & f_2 T^2 & \dots & f_d T^d \end{pmatrix}.$$

If we can find integers $a_0, a_1, \dots, a_{d-1}$ and c such that the vector $(a_0, a_1, \dots, a_{d-1}, c)\mathbf{B}$ is short, then we have found $h(X)$.

Lattices

Let $\mathbf{B}$ be a (usually square) matrix with linearly independent rows (maximal rank).

A *lattice* Λ generated by $\mathbf{B}$ is every integer linear combination of the rows of $\mathbf{B}$:

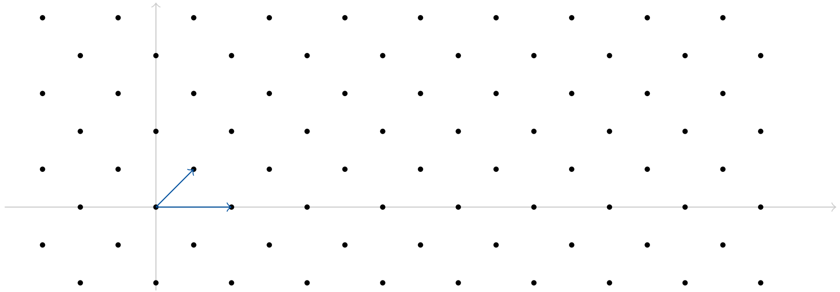
$$\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}.$$

We say that $\mathbf{B}$ is a *basis* for Λ.

Often we talk about the rows $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ in $\mathbf{B}$ as basis vectors.

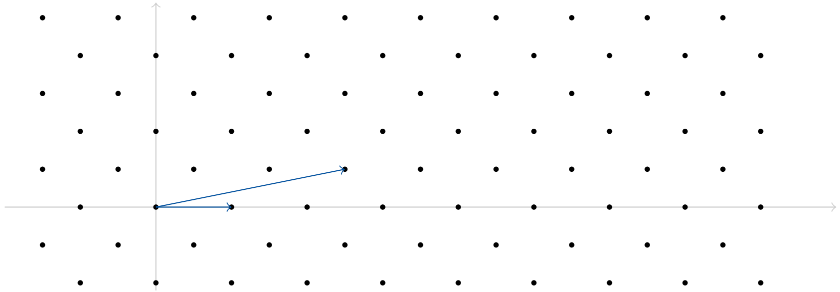
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



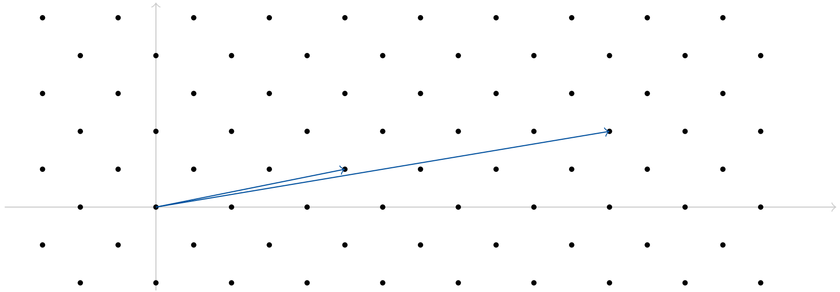
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



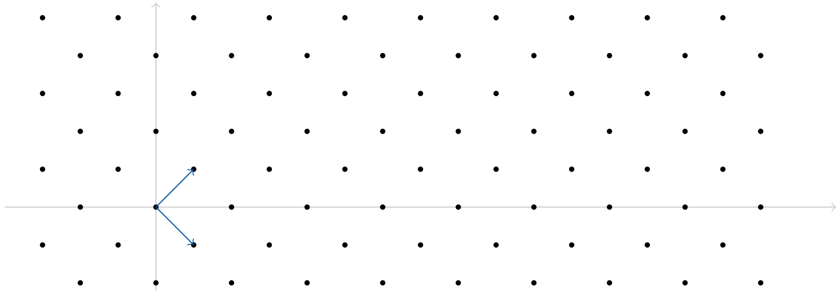
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.

In higher dimensions: If $\mathbf{B}$ and $\mathbf{C}$ are two bases for the same lattice, then the rows in $\mathbf{B}$ can be written as integer linear combinations of the rows in $\mathbf{C}$, and vice versa, so

$$\mathbf{B} = \mathbf{UC} = \mathbf{UVB}.$$

It follows that $\mathbf{U}$ and $\mathbf{V}$ are inverses and integer matrices, so they have determinant ± 1.

Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.

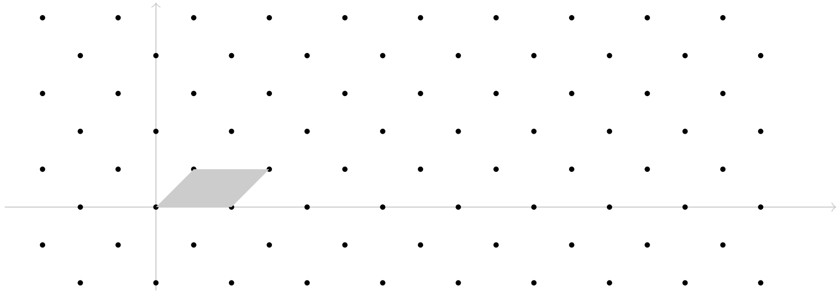
The fundamental domain of a lattice is the parallelogram defined by the basis vectors. The shape of the fundamental domain depends on the basis, but the area/volume is independent.

We can always write any vector $\mathbf{z}$ in space as a sum of a vector in the lattice and a vector in the fundamental domain.

$$\boldsymbol{\alpha} = \mathbf{zB}^{-1} \rightarrow \mathbf{x} = \lfloor \boldsymbol{\alpha} \rceil \mathbf{B}.$$

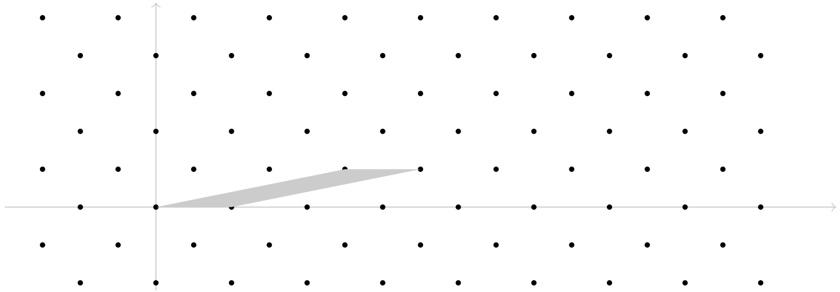
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



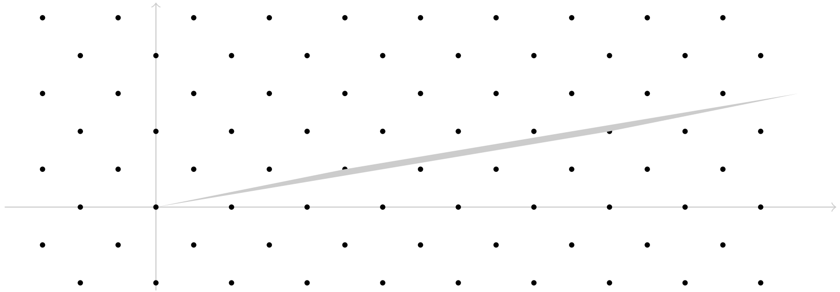
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



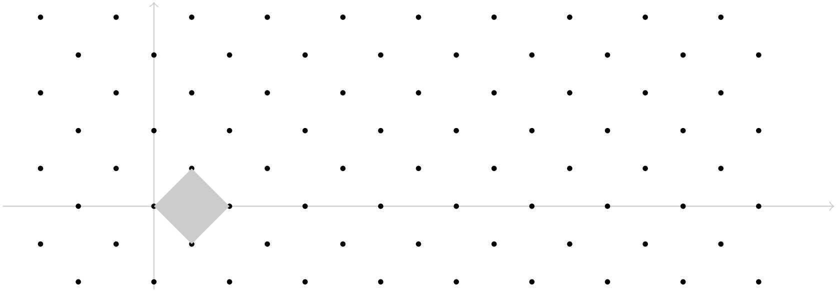
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



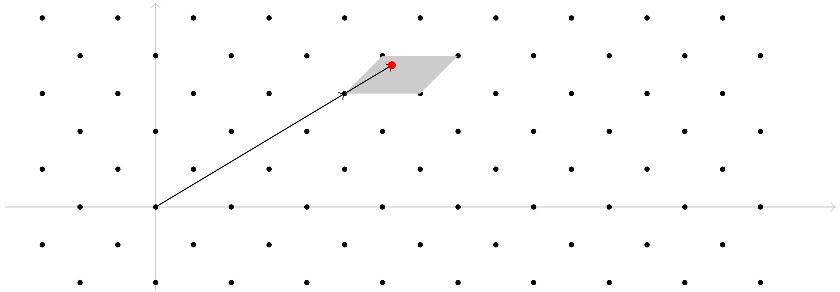
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



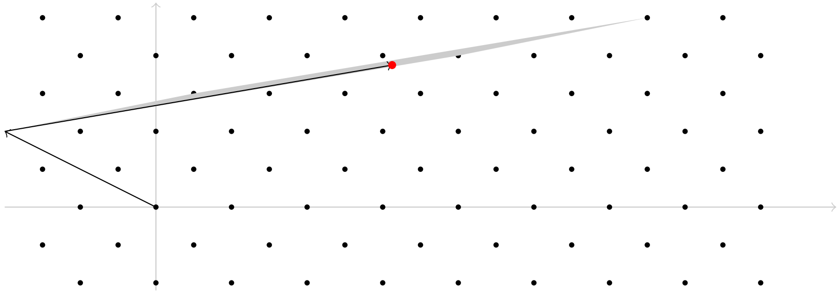
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



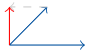
Lattices

A *lattice* Λ generated by $\mathbf{B}$ is $\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ is a *basis* for Λ.



Gram-Schmidt

Gram-Schmidt makes an orthogonal basis:

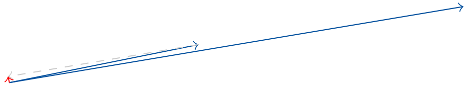


In higher dimensions we begin with a basis $\mathbf{b}_1, \dots, \mathbf{b}_n$ and construct $\mathbf{b}_1^*, \dots, \mathbf{b}_n^*$ stepwise by $\mathbf{b}_i^*$ being $\mathbf{b}_i$ minus the projection of $\mathbf{b}_i$ onto the span of $\mathbf{b}_1^*, \dots, \mathbf{b}_{i-1}^*$.

Note: $\mathbf{b}_i^*$ is orthogonal to not just $\mathbf{b}_1^*, \dots, \mathbf{b}_{i-1}^*$, but also $\mathbf{b}_1, \dots, \mathbf{b}_{i-1}$.

Gram-Schmidt

Gram-Schmidt makes an orthogonal basis:



In higher dimensions we begin with a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ and construct $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ stepwise by $\mathbf{b}_i^*$ being $\mathbf{b}_i$ minus the projection of $\mathbf{b}_i$ onto the span of $\mathbf{b}_1^*, \ldots, \mathbf{b}_{i-1}^*$.

Note: $\mathbf{b}_i^*$ is orthogonal to not just $\mathbf{b}_1^*, \ldots, \mathbf{b}_{i-1}^*$, but also $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$.

Lattice-problems

Find a shortest vector. Given a lattice Λ, find a vector $\mathbf{x} \in \Lambda$ such that $\mathbf{x} \neq \mathbf{0}$ and no other non-zero lattice vector is shorter than $\mathbf{x}$.

Find a closest point. Given a lattice Λ and a point $\mathbf{z}$ in $\mathbb{R}^n$, find $\mathbf{x} \in \Lambda$ such that no other lattice vector is closer to $\mathbf{z}$ than $\mathbf{x}$.

There is a huge variety of lattice problems, and there are many subtleties with the probability distributions involved.

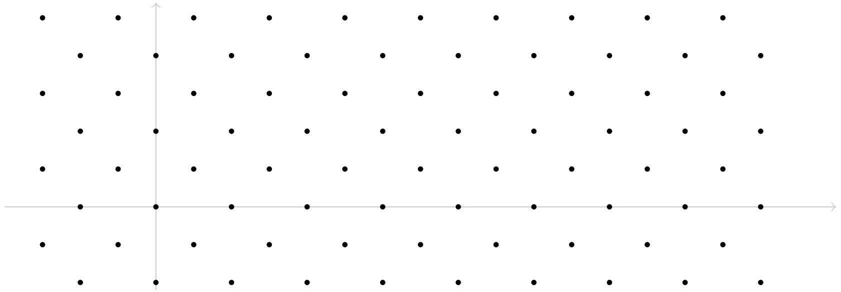
Rounding

Given a lattice Λ with basis $\mathbf{B}$ and a point $\mathbf{z} \in \mathbb{R}^n$, we want to find a vector in Λ that is as close as possible to $\mathbf{z}$.

It is tempting to keep it simple. We write $\mathbf{z}$ as a (non-integral) linear combination of the basis vectors and round the coefficients to the nearest integer.

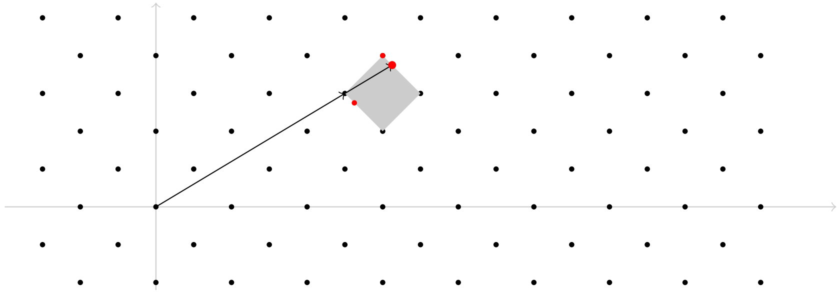
Rounding

Given a lattice Λ with basis $\mathbf{B}$ and a point $\mathbf{z} \in \mathbb{R}^n$, we want to find a vector in Λ that is as close as possible to $\mathbf{z}$.



Rounding

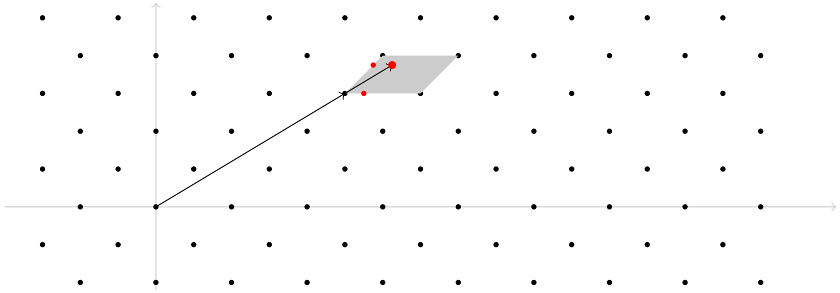
Given a lattice Λ with basis $\mathbf{B}$ and a point $\mathbf{z} \in \mathbb{R}^n$, we want to find a vector in Λ that is as close as possible to $\mathbf{z}$.



This is perfect if the basis vectors are orthogonal.

Rounding

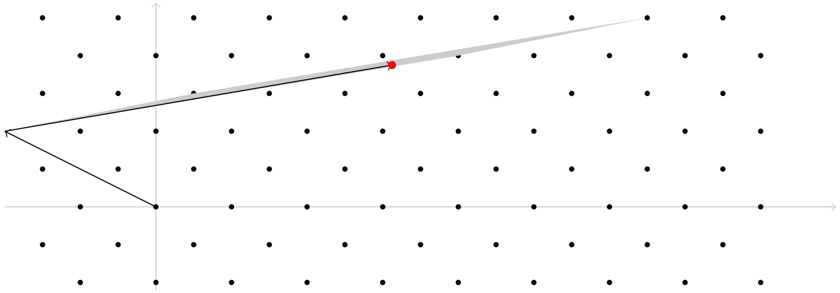
Given a lattice Λ with basis $\mathbf{B}$ and a point $\mathbf{z} \in \mathbb{R}^n$, we want to find a vector in Λ that is as close as possible to $\mathbf{z}$.



It can work reasonably well if the basis vectors are nearly orthogonal, at least as a starting point for a search among nearby vectors in the lattice.

Rounding

Given a lattice Λ with basis $\mathbf{B}$ and a point $\mathbf{z} \in \mathbb{R}^n$, we want to find a vector in Λ that is as close as possible to $\mathbf{z}$.



It can work badly if the basis is far from orthogonal.

LLL

A lattice basis is Lenstra-Lenstra-Lovász-reduced if

- The component of $\mathbf{b}_i$ in the direction of $\mathbf{b}_j^*$ is at most half as long as $\mathbf{b}_j^*$.
- The ration of the lengths of $\mathbf{b}_{i-1}^*$ and $\mathbf{b}_i^*$ is at most $\sqrt{2}$.

LLL

A lattice basis is Lenstra-Lenstra-Lovász-reduced if

— The component of $\mathbf{b}_i$ in the direction of $\mathbf{b}_j^*$ is at most half as long as $\mathbf{b}_j^*$.

— The ration of the lengths of $\mathbf{b}_{i-1}^*$ and $\mathbf{b}_i^*$ is at most $\sqrt{2}$.

If $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is LLL-reduced, then:

— The length of $\mathbf{b}_1$ is at most $2^{(n-1)/2}$ times the length of the shortest vector in the lattice.

— The distance from $\mathbf{z}$ to the estimate the rounding method gives us is at most $1 + 2n(9/2)^{n/2}$ times the distance from $\mathbf{z}$ to a closest lattice vector.

LLL

A lattice basis is Lenstra-Lenstra-Lovász-reduced if

- The component of $\mathbf{b}_i$ in the direction of $\mathbf{b}_j^*$ is at most half as long as $\mathbf{b}_j^*$.
- The ration of the lengths of $\mathbf{b}_{i-1}^*$ and $\mathbf{b}_i^*$ is at most $\sqrt{2}$.

The LLL-algorithm gives us an LLL-reduced basis.

- Make sure the first requirement holds.
- If $\mathbf{b}_{i-1}^*$ is too long relative to $\mathbf{b}_i^*$, change their order.

Repeat until no further order changes are needed.

LLL

A lattice basis is Lenstra-Lenstra-Lovász-reduced if

— The component of $\mathbf{b}_i$ in the direction of $\mathbf{b}_j^*$ is at most half as long as $\mathbf{b}_j^*$.

— The ration of the lengths of $\mathbf{b}_{i-1}^*$ and $\mathbf{b}_i^*$ is at most $\sqrt{2}$.

The LLL-algorithm gives us an LLL-reduced basis.

— Make sure the first requirement holds.

— If $\mathbf{b}_{i-1}^*$ is too long relative to $\mathbf{b}_i^*$, change their order.

Repeat until no further order changes are needed.

It is relatively easy to show that the number of iterations is polynomial. The hard part is to show that the precision needed in the computations isn't too big.

Searching for short vectors

We want to find all

$$\mathbf{z} = \sum_i a_i \mathbf{b}_i$$

such that $\mathbf{z}$ is shorter than some bound T.

Searching for short vectors

We want to find all

$$\mathbf{z} = \sum_i a_i \mathbf{b}_i$$

such that $\mathbf{z}$ is shorter than some bound T.

Using the Gram-Schmidt-basis we can write $\mathbf{z}$ as

$$\mathbf{z} = \sum_i \alpha_i \mathbf{b}_i^* \text{ of length } ||\mathbf{z}||^2 = \sum_i |\alpha_i|^2 ||\mathbf{b}_i^*||^2.$$

We know that $\mathbf{b}_n^*$ is orthogonal to $\mathbf{b}_1, \dots, \mathbf{b}_{n-1}$, so $\alpha_n = a_n$. Given a_n we know that $||\mathbf{z}||$ is at least $|a_n| ||\mathbf{b}_n^*||$. In other words, we can limit our search to $|a_n| \leq T / ||\mathbf{b}_n^*||$.

Searching for short vectors

We want to find all

$$\mathbf{z} = \sum_i a_i \mathbf{b}_i$$

such that $\mathbf{z}$ is shorter than some bound T.

Now we try every possible a_n systematically, so we begin with the starting point $a_n\mathbf{b}_n$. The next coefficient is a_{n-1}, which in the same way will contribute $|a_{n-1}|\|\mathbf{b}^*_{n-1}\|$ to the length, but $a_n\mathbf{b}_n$ could have a component in the direction of $\mathbf{b}^*_{n-1}$ which we must include when we compute the search range for a_{n-1}.

Now we have $a_{n-1}\mathbf{b}_{n-1} + a_n\mathbf{b}_n$. Again, we need to include the components of $a_{n-1}\mathbf{b}_{n-1}$ ant $a_n\mathbf{b}_n$ along $\mathbf{b}^*_{n-2}$ when we compute the search range for a_{n-2}.

Searching for short vectors

We want to find all

$$\mathbf{z} = \sum_i a_i \mathbf{b}_i$$

such that $\mathbf{z}$ is shorter than some bound T.

Now we try every possible a_n systematically, so we begin with the starting point $a_n\mathbf{b}_n$. The next coefficient is a_{n-1}, which in the same way will contribute $|a_{n-1}|\|\mathbf{b}^*_{n-1}\|$ to the length, but $a_n\mathbf{b}_n$ could have a component in the direction of $\mathbf{b}^*_{n-1}$ which we must include when we compute the search range for a_{n-1}.

Now we have $a_{n-1}\mathbf{b}_{n-1} + a_n\mathbf{b}_n$. Again, we need to include the components of $a_{n-1}\mathbf{b}_{n-1}$ ant $a_n\mathbf{b}_n$ along $\mathbf{b}^*_{n-2}$ when we compute the search range for a_{n-2}.

And so on …

Searching for short vectors

We want to find all

$$\mathbf{z} = \sum_i a_i \mathbf{b}_i$$

such that $\mathbf{z}$ is shorter than some bound T.

This is a slightly careful exhaustive search, but the runtime is exponential. The runtime is especially sensitive to the length of $\mathbf{b}_n^*$. And it is very sensitive to the T we use.

Lattice-based cryptography: GGH

We choose an "almost" orthogonal basis $\mathbf{B}$ for a lattice Λ and a «bad» basis $\mathbf{C}$ for the same lattice.

To encrypt, we choose a lattice point $\mathbf{x} = \mathbf{a}\mathbf{C}$ and a not too long «noise vector» $\mathbf{e}$. Then we compute the ciphertext as

$$\mathbf{z} = \mathbf{x} + \mathbf{e}.$$

To decrypt compute $\mathbf{x} = \lfloor \mathbf{z}\mathbf{B}^{-1} \rceil \mathbf{B}$ and $\mathbf{e} = \mathbf{z} - \mathbf{x}$.

(We could recover $\mathbf{a}$ as $\mathbf{x}\mathbf{C}^{-1}$, but it may be easier to use $\mathbf{x}$ and $\mathbf{e}$ in a KEM design.)

This works since $\mathbf{z} \in \mathbb{R}^n$ is not too far away from a lattice vector, so rounding works well and

$$\mathbf{x} = \lfloor \mathbf{z}\mathbf{B}^{-1} \rceil \mathbf{B}$$

will be the closest lattice point.

Lattice-based cryptography: NTRU

Let q be a prime and consider the two rings $R = \mathbb{Z}[X]/\langle X^n - 1\rangle$, $R_q = R/qR$ and $R_2 = R/2R$.

Choose two "short" polynomials $g(X)$ and $f(X)$, where the second polynomial should have inverses in R_q and R_2. The public key is $h(X) = 2g(X)/f(X) \bmod q$.

To encrypt a polynomial $\mathrm{e}(X)$ with all coefficients 0 or 1, choose a "short" polynomial $r(X)$ and compute the ciphertext as

$$y(X) = h(X)r(X) + \mathrm{e}(X) \bmod q.$$

To decrypt, compute $z(X) = y(X)f(X) \bmod q$ and then compute $\mathrm{e}(X) = z(X)/f(X) \bmod 2$.

This works because

$$y(X)f(X) \equiv h(X)r(X)f(X) + \mathrm{e}(X)f(X) \equiv 2g(X)r(X) + \mathrm{e}(X)f(X)$$

and since all of these are "short", we have equality in R, so the calculation modulo 2 works.

Lattice-based cryptography: Regev's LWE-based system

Let q be a prime. Choose a secret vector $\mathbf{s} \in \mathbb{F}_q^n$, a matrix $\mathbf{A} \in \mathbb{F}_q^{k \times n}$ and a noise vector $\mathbf{e}$ (from a subtle distribution). Compute $\mathbf{b} = \mathbf{sA} + \mathbf{e}$. The public key is $(\mathbf{A}, \mathbf{b})$

To encrypt a message $m \in \{0, 1\}$, choose a "short" vector $\mathbf{a}$ and compute

$$\mathbf{y} = \mathbf{aA}^T \qquad w = \mathbf{ba}^T + m \left\lfloor \frac{q}{2} \right\rfloor .$$

To decrypt, compute

$$z = w - \mathbf{sy}^T .$$

If z is small, the decryption is 0, otherwise it is 1.

This works because $\mathbf{e}$ and $\mathbf{a}$ are "short" vectors and

$$w - \mathbf{sy}^T = \mathbf{s}(\mathbf{aA}^T)^T + \mathbf{ea}^T + m \left\lfloor \frac{q}{2} \right\rfloor - \mathbf{sAa}^T = \mathbf{ea}^T + m \left\lfloor \frac{q}{2} \right\rfloor$$

Post-quantum cryptography
Multivariate cryptography

Kristian Gjøsteen

Department of Mathematical Sciences, NTNU

Finse, May 2017

Systems of equations

We all know how to solve linear equations:

$$\alpha X = \gamma.$$

Systems of equations

We all know how to solve linear equations:

$$\alpha X = \gamma.$$

If we have more than one unknown, there are too many solutions to the linear equation:

$$\alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_n X_n = \gamma.$$

Systems of equations

We all know how to solve linear equations:

$$\alpha X = \gamma.$$

If we have more than one unknown, there are too many solutions to the linear equation:

$$\alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_n X_n = \gamma.$$

However, if we have a *system of linear equations*:

$$\begin{aligned}
\alpha_{11} X_1 + \alpha_{12} X_2 + \cdots + \alpha_{1n} X_n &= \gamma_1 \\
\alpha_{21} X_1 + \alpha_{22} X_2 + \cdots + \alpha_{2n} X_n &= \gamma_2 \\
&\vdots \\
\alpha_{m1} X_1 + \alpha_{m2} X_2 + \cdots + \alpha_{mn} X_n &= \gamma_m
\end{aligned}$$

Systems of equations

We all know how to find the solutions to a polynomial equation:

$$\alpha_0 + \alpha_1 X + \alpha_2 X^2 + \cdots + \alpha_t X^t = \gamma.$$

Systems of equations

We all know how to find the solutions to a polynomial equation:

$$\alpha_0 + \alpha_1 X + \alpha_2 X^2 + \cdots + \alpha_t X^t = \gamma.$$

If we have more than one unknown, there are typically too many solutions to the multivariate polynomial equation:

$$\sum \alpha_{i_1, i_2, \ldots, i_n} X_1^{i_1} X_2^{i_2} \ldots X_n^{i_n} = \gamma.$$

Systems of equations

We all know how to find the solutions to a polynomial equation:

$$\alpha_0 + \alpha_1 X + \alpha_2 X^2 + \cdots + \alpha_t X^t = \gamma.$$

If we have more than one unknown, there are typically too many solutions to the multivariate polynomial equation:

$$\sum \alpha_{i_1,i_2,\ldots,i_n} X_1^{i_1} X_2^{i_2} \ldots X_n^{i_n} = \gamma.$$

However, if we have a *system of multivariate polynomial equations*:

$$\begin{gathered}\sum \alpha_{1,i_1,i_2,\ldots,i_n} X_1^{i_1} X_2^{i_2} \ldots X_n^{i_n} = \gamma_1 \\ \vdots \\ \sum \alpha_{m,i_1,i_2,\ldots,i_n} X_1^{i_1} X_2^{i_2} \ldots X_n^{i_n} = \gamma_m\end{gathered}$$

Systems of equations

We all know how to find the solutions to a polynomial equation:

$$\alpha_0 + \alpha_1 X + \alpha_2 X^2 + \cdots + \alpha_t X^t = \gamma.$$

If we have more than one unknown, there are typically too many solutions to the multivariate polynomial equation:

$$\sum \alpha_{i_1,i_2,\dots,i_n} X_1^{i_1} X_2^{i_2} \dots X_n^{i_n} = \gamma.$$

However, if we have a *system of multivariate polynomial equations*:

$$\begin{aligned} \sum \alpha_{1,i_1,i_2,\dots,i_n} X_1^{i_1} X_2^{i_2} \dots X_n^{i_n} &= \gamma_1 \\ &\vdots \\ \sum \alpha_{m,i_1,i_2,\dots,i_n} X_1^{i_1} X_2^{i_2} \dots X_n^{i_n} &= \gamma_m \end{aligned}$$

How easy is it to find a solution to such a system? Over a finite field?

Systems of multivariate quadratic equations

We are going to restrict attention to systems of multivariate quadratic polynomial equations:

$$\sum \alpha_{1,i,j} X_i X_j + \sum_i \beta_{1,i} X_i = \gamma_1$$

$$\vdots$$

$$\sum \alpha_{m,i,j} X_i X_j + \sum_i \beta_{m,i} X_i = \gamma_m$$

We need some notation. Let $\mathbf{A}_k = [\alpha_{k,i,j}]$, $\boldsymbol{\beta}_k = (\beta_{k,i})$ and let $\mathbf{X} = (X_1, X_2, \dots, X_n)$. Then

$$\mathbf{X}\mathbf{A}_k\mathbf{X}^T + \boldsymbol{\beta}_k\mathbf{X}^T = \gamma_k$$

describes the *k*th equation.

Equivalent systems: combining equations

One thing we can easily do to a system is to combine equations. For instance, if we add the first equation to the second equation, the solution space does not change. In general, we can replace each equation by a linear combination of all the equations without changing the solution space, as long as the linear map is invertible.

Let $\mathsf{T} = [\tau_{ij}]$ be an invertible matrix. Then we have new system where the *i*th equation is

$$\mathsf{x}\left(\sum_j \tau_{ij}\mathsf{A}_j\right)\mathsf{x}^T + \left(\sum_j \tau_{ij}\boldsymbol{\beta}_j\right)\mathsf{x}^T = \sum_j \tau_{ij}\gamma_j.$$

Equivalent systems: change of variables

Another thing we can easily do to a system is a change of variables. Replacing X_1 and X_2 by $Y_1 + Y_2$ and Y_2, respecively, gives us a different polynomial system, but it will still have the same degree, and we can easily recover solutions to the old system from solutions to the old system. In general, any linear change of variables will do.

Let $\mathbf{S}$ be an invertible matrix. With the change of variables $\mathbf{X} = \mathbf{YS}$, the the *i*th equation of our system becomes

$$(\mathbf{YS})\mathbf{A}_i(\mathbf{YS})^T + \boldsymbol{\beta}_i(\mathbf{YS})^T = \gamma_i$$

or

$$\mathbf{Y}(\mathbf{SA}_i\mathbf{S}^T)\mathbf{Y}^T + (\boldsymbol{\beta}_i\mathbf{S}^T)\mathbf{Y} = \gamma_i.$$

Triangular systems

Not all systems are difficult to solve. For instance, a system where the ith equation does not have any terms with $X_{i+1}, X_{i+2}, \ldots, X_n$, and only a linear term containing X_i.

In this case, the first equation is a linear equation with the only unknown X_1, which we can easily solve.

Once we know the solution for $X_1, \ldots, X_{i-1}$, we can insert this solution into the ith equation, which results in a linear equation in X_i only, which we again can easily solve.

Oil and vinegar

Consider a system where the matrices $\mathbf{A}_i$ has zero entries for $i = 1, 2, \ldots, s$. In other words, there are no quadratic terms with variables only from $X_1, X_2, \ldots, X_s$. The $\mathbf{A}_i$ matrices look like

$$\mathsf{A}_i = \left(\begin{array}{c|c} \mathsf{0} & \mathsf{U} \\ \hline \mathsf{V} & \mathsf{W} \end{array}\right)$$

Suppose we have $s < m$ equations of this form. To solve, we first choose random values for $X_{s+1}, \ldots, X_m$. This produces a linear system of equations in $X_1, X_2, \ldots, X_s$. If the system has a solution, we are done. Otherwise, choose new random values.

Solving: Linearisation

The idea is to replace the quadratic terms with new linear unknowns:

$$X + Y + XY = \gamma \qquad \rightsquigarrow \qquad X + Y + Z = \gamma.$$

This gives us a linear system of equations. If our non-linear system has a solution, then the linear system has a solution. Which means that if the linear system has a unique solution, we are done.

However, unless we have more equations than unknowns, we will get a linear system with many solutions, and we do not know which one corresponds to the solution of the non-linear system.

(In algebraic cryptanalysis, there are situations where you can get a large number of equations with relatively few terms, making this approach feasible.)

Ideals in polynomial rings

An *ideal* in a multivariate polynomial ring generated by polynomials $\{f_1(\mathbf{X}), f_2(\mathbf{X}), \dots, f_m(\mathbf{X})\}$ the the set of multivariate polynomials given by

$$\left\{ \sum_{i=1}^{m} \phi_i(\mathbf{X}) f_i(\mathbf{X}) \right\}.$$

If $\boldsymbol{\xi}$ is a zero of every generator $f_i(\mathbf{X})$, then $\boldsymbol{\xi}$ is a zero of every multivariate polynomial in the ideal.

Conversely, if we find some other set of polynomials that generate the same ideal and a zero $\boldsymbol{\xi}$ of all those polynomials, we know that $\boldsymbol{\xi}$ will be a zero of our original generators.

Monomial orders

A *monomial* is a product of the form $X_{i_1}^{r_1} X_{i_2}^{r_2} \dots X_{i_k}^{r_k}$. A multivariate polynomial is a sum of monomials.

Suppose we have a *total ordering* $<$ of these monomials. Then it makes sense to talk about the *leading term* of a multivariate polynomial, the largest monomial in the sum relative to $<$.

We denote the leading term of a multivariate polynomial $f(\mathsf{X})$ by $\mathrm{lt}(f(\mathsf{X}))$.

Gröbner basis

Suppose we have an ideal $\mathcal{I}$. A set of polynomials $\{g_1(\mathsf{X}), g_2(\mathsf{X}), \ldots, g_N\}$ is a *Gröbner basis* for $\mathcal{I}$ if it generates the ideal and for any $f(\mathsf{X}) \in \mathcal{I}$, there is a generator $g_i(\mathsf{X})$ such that $\mathrm{lt}(g_i(\mathsf{X}))$ divides $\mathrm{lt}(f(\mathsf{X}))$.

Solving: Gröbner basis

Once we have a Gröbner basis under a certain monomial ordering, there is a theorem that says that the first polynomials in the basis will be polynomials in X_1 only. Later, we get polynomials in X_1, X_2 only. And so on.

Finding a Gröbner basis

- Buchberger
- F4
- F5

They are good, but not good enough.

Recovering a triangular form

If we can find a linear combination of the matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_m$ that has lower rank, we can find a linear combination of our equations and a linear change of variables such that one variable only appears as a linear term.

This is the MinRank problem.

We can solve MinRank by creating a new system of equations and finding a solution to that system.

Multivariate cryptography: Triangular systems

The secret key is the left-hand side of a triangular system described by $\{\mathbf{A}_i\}$ and $\{\boldsymbol{\beta}_i\}$, and two invertible linear maps $\mathbf{T}$ and $\mathbf{S}$. The public key is then

$$\mathbf{A}'_i = \sum_j \tau_{ij}\mathbf{S}\mathbf{A}_j\mathbf{S}^T \qquad\qquad \boldsymbol{\beta}'_i = \sum_j \tau_{ij}\boldsymbol{\beta}_j\mathbf{S}^T.$$

To encrypt $\boldsymbol{\xi}$, we compute $\gamma_i = \boldsymbol{\xi}\mathbf{A}'_i\boldsymbol{\xi}^T + \boldsymbol{\beta}'_j\boldsymbol{\xi}^T$. Our ciphertexts is $\boldsymbol{\gamma}$.

To decrypt, compute $\boldsymbol{\gamma}' = \mathbf{T}^{-1}\boldsymbol{\gamma}$. Then find a solution $\boldsymbol{\xi}'$ to the triangular system with equations

$$\mathbf{X}\mathbf{A}_i\mathbf{X}^T + \boldsymbol{\beta}\mathbf{X}^T = \gamma'_i.$$

The decryption is then $\boldsymbol{\xi} = \boldsymbol{\xi}'\mathbf{S}^{-1}$.

This works because

$$\gamma'_i = \boldsymbol{\xi}'\mathbf{A}_i(\boldsymbol{\xi}')^T + \boldsymbol{\beta}_i(\boldsymbol{\xi}')^T = \boldsymbol{\xi}'\mathbf{S}^{-1}\mathbf{S}\mathbf{A}_i\mathbf{S}^T(\mathbf{S}^T)^{-1}(\boldsymbol{\xi}')^T + \boldsymbol{\beta}_i\mathbf{S}^T(\mathbf{S}^T)^{-1}(\boldsymbol{\xi}')^T = \boldsymbol{\xi}\mathbf{S}\mathbf{A}_i\mathbf{S}^T\boldsymbol{\xi}^T + \boldsymbol{\beta}_i\mathbf{S}^T\boldsymbol{\xi}^T.$$

Multivariate cryptography: Oil and vinegar

The secret key is the left-hand side of an oil and vinegar system described by $\{\mathbf{A}_i\}$ and $\{\boldsymbol{\beta}_i\}$ with parameter s, and two invertible linear maps $\mathbf{T}$ and $\mathbf{S}$. The public key is then

$$\mathbf{A}'_i = \sum_j \tau_{ij} \mathbf{S}\mathbf{A}_j\mathbf{S}^T \qquad \boldsymbol{\beta}'_i = \sum_j \tau_{ij} \boldsymbol{\beta}_j \mathbf{S}^T.$$

To sign a message $\boldsymbol{\gamma}$, compute $\boldsymbol{\gamma}' = \mathbf{T}^{-1}\boldsymbol{\gamma}$. Then find a solution $\boldsymbol{\xi}'$ to the oil and vinegar system

$$\mathbf{X}\mathbf{A}_i\mathbf{X}^T + \boldsymbol{\beta}\mathbf{X}^T = \gamma'_i.$$

The signature is $\boldsymbol{\xi} = \boldsymbol{\xi}'\mathbf{S}^{-1}$.

We verify the signature $\boldsymbol{\xi}$ on $\boldsymbol{\gamma}$ by verifying that $\gamma_i = \boldsymbol{\xi}\mathbf{A}'_i\boldsymbol{\xi}^T + \boldsymbol{\beta}'_i\boldsymbol{\xi}^T$.

This works because

$$\gamma'_i = \boldsymbol{\xi}'\mathbf{A}_i(\boldsymbol{\xi}')^T + \boldsymbol{\beta}_i(\boldsymbol{\xi}')^T = \boldsymbol{\xi}'\mathbf{S}^{-1}\mathbf{S}\mathbf{A}_i\mathbf{S}^T(\mathbf{S}^T)^{-1}(\boldsymbol{\xi}')^T + \boldsymbol{\beta}_i\mathbf{S}^T(\mathbf{S}^T)^{-1}(\boldsymbol{\xi}')^T = \boldsymbol{\xi}\mathbf{S}\mathbf{A}_i\mathbf{S}^T\boldsymbol{\xi}^T + \boldsymbol{\beta}_i\mathbf{S}^T\boldsymbol{\xi}^T.$$