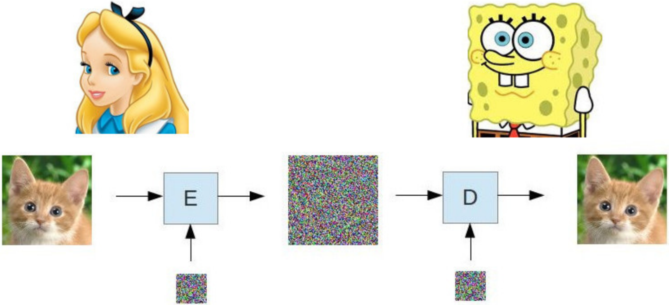
Public-key cryptography in the pre- and post-quantum world

Gabriel Chênevert

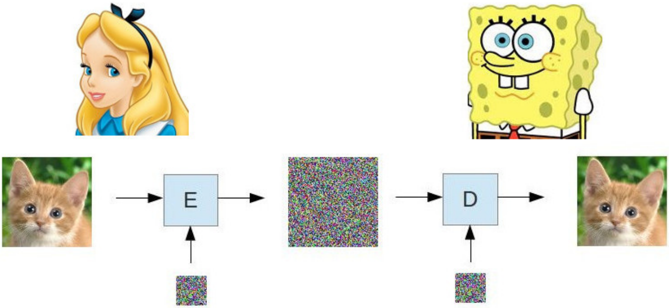
October 18, 2018

Catholic University of Lille

Symmetric cryptography

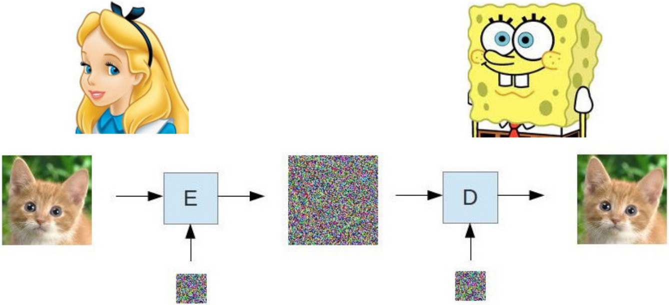


Symmetric cryptography



- encryption and decryption (knowing the key) should be fast

Symmetric cryptography



- encryption and decryption (knowing the key) should be fast
- decryption without the proper key should be **LONG**

Symmetric cryptography

Advanced Encryption Standard

- A federal standard since 2001

Symmetric cryptography

Advanced Encryption Standard

- A federal standard since 2001
- Proposed as Rijndael by a team from KU Leuven

Symmetric cryptography

Advanced Encryption Standard

- A federal standard since 2001
- Proposed as Rijndael by a team from KU Leuven
- Based on permutations and bit operations

Symmetric cryptography

Advanced Encryption Standard

- A federal standard since 2001
- Proposed as Rijndael by a team from KU Leuven
- Based on permutations and bit operations
- Accounts for the vast majority of all encryption in use today

Symmetric cryptography

Advanced Encryption Standard

- A federal standard since 2001
- Proposed as Rijndael by a team from KU Leuven
- Based on permutations and bit operations
- Accounts for the vast majority of all encryption in use today
- Uses 128, 192 or 256−bit secret keys

Symmetric cryptography

Advanced Encryption Standard

- A federal standard since 2001
- Proposed as Rijndael by a team from KU Leuven
- Based on permutations and bit operations
- Accounts for the vast majority of all encryption in use today
- Uses 128, 192 or 256−bit secret keys
- The best attack is essentially brute force : try to decrypt with all (2^{128}, 2^{192}, 2^{256}) possible keys

Symmetric cryptography
Advanced Encryption Standard

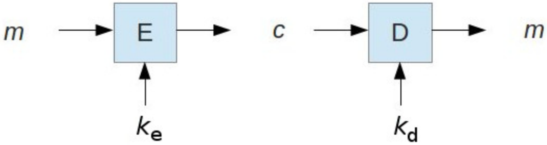
- A federal standard since 2001
- Proposed as Rijndael by a team from KU Leuven
- Based on permutations and bit operations
- Accounts for the vast majority of all encryption in use today
- Uses 128, 192 or 256−bit secret keys
- The best attack is essentially brute force : try to decrypt with all (2^{128}, 2^{192}, 2^{256}) possible keys
- Key distribution might be a problem...

Asymmetric encryption

Asymmetric encryption

- k_e public, k_d private : public-key encryption

Asymmetric encryption



- k_e public, k_d private : public-key encryption
- k_e private, k_e public : digital signature

Asymmetric cryptography

A solution to the key distribution problem

- Alice comes up with the secret key k

Asymmetric cryptography

A solution to the key distribution problem

- Alice comes up with the secret key k
- encrypts it with Bob's public key and sends it to him

Asymmetric cryptography

A solution to the key distribution problem

- Alice comes up with the secret key k
- encrypts it with Bob's public key and sends it to him
- Bob can then recover k using his private decryption key

Asymmetric cryptography

A solution to the key distribution problem

- Alice comes up with the secret key k
- encrypts it with Bob's public key and sends it to him
- Bob can then recover k using his private decryption key
- and they may now use AES for the rest of their discussion.

There are even (widely used schemes) in which Alice and Bob both come up with part of the key (Diffie-Hellman).

Asymmetric cryptography

A solution to the key distribution problem

- Alice comes up with the secret key k
- encrypts it with Bob's public key and sends it to him
- Bob can then recover k using his private decryption key
- and they may now use AES for the rest of their discussion.

There are even (widely used schemes) in which Alice and Bob both come up with part of the key (Diffie-Hellman).

They could also digitally sign their exchanges to avoid man-in-the-middle attacks.

Modular arithmetic

Almost all algorithms in use today are based on modular arithmetic.

Modular arithmetic

Almost all algorithms in use today are based on modular arithmetic.

Definition

If a, b and n are integers, $a \underset{n}{\equiv} b$ means that $n \mid (b - a)$.

Modular arithmetic

Almost all algorithms in use today are based on modular arithmetic.

Definition

If a, b and n are integers, $a \underset{n}{\equiv} b$ means that $n \mid (b - a)$.

In others words : $a \underset{n}{\equiv} b$ when there exists some integer k for which

$$a = b + kn.$$

Modular arithmetic

Almost all algorithms in use today are based on modular arithmetic.

Definition
If a, b and n are integers, $a \underset{n}{\equiv} b$ means that $n \mid (b - a)$.

In others words : $a \underset{n}{\equiv} b$ when there exists some integer k for which

$$a = b + kn.$$

Lemma
If $a \underset{n}{\equiv} a'$ and $b \underset{n}{\equiv} b'$,

Modular arithmetic

Almost all algorithms in use today are based on modular arithmetic.

Definition

If a, b and n are integers, $a \underset{n}{\equiv} b$ means that $n \mid (b - a)$.

In others words : $a \underset{n}{\equiv} b$ when there exists some integer k for which

$$a = b + kn.$$

Lemma

If $a \underset{n}{\equiv} a'$ *and* $b \underset{n}{\equiv} b'$*, then* $a + b \underset{n}{\equiv} a' + b'$

Modular arithmetic

Almost all algorithms in use today are based on modular arithmetic.

Definition

If a, b and n are integers, $a \underset{n}{\equiv} b$ means that $n \mid (b - a)$.

In others words : $a \underset{n}{\equiv} b$ when there exists some integer k for which

$$a = b + kn.$$

Lemma

If $a \underset{n}{\equiv} a'$ *and* $b \underset{n}{\equiv} b'$*, then* $a + b \underset{n}{\equiv} a' + b'$ *and* $a \cdot b \underset{n}{\equiv} a' \cdot b'$.

Modular arithmetic

Example : clock arithmetic

Instead of going to the colloquium, I start watching the full Star Wars saga (official episodes only) ; what time will it be when I'm done ?

Modular arithmetic

Example : clock arithmetic

Instead of going to the colloquium, I start watching the full Star Wars saga (official episodes only) ; what time will it be when I'm done ?

$$4 + 8 \cdot 2 = 4 + 16 \underset{12}{\equiv} 4 + 4 = 8$$

Modular arithmetic

Example : clock arithmetic

Instead of going to the colloquium, I start watching the full Star Wars saga (official episodes only) ; what time will it be when I'm done ?

$$4 + 8 \cdot 2 = 4 + 16 \underset{12}{\equiv} 4 + 4 = 8$$

With a 24-hour clock :

$$16 + 8 \cdot 2 = 16 + 16 = 32 \underset{24}{\equiv} 8$$

Modular arithmetic

Taking powers

The operation of taking modular powers can be computed efficiently by repeated squarings.

Example : to compute $y \underset{541}{\equiv} 176^{40}$.

Modular arithmetic

Taking powers

The operation of taking modular powers can be computed efficiently by repeated squarings.

Example : to compute $y \underset{541}{\equiv} 176^{40}$.

Use the fact that $40 = 32 + 8$ and compute by successive squarings :

$b = 176$,

Modular arithmetic

Taking powers

The operation of taking modular powers can be computed efficiently by repeated squarings.

Example : to compute $y \underset{541}{\equiv} 176^{40}$.

Use the fact that $40 = 32 + 8$ and compute by successive squarings :

$b = 176, \; b^2 \underset{541}{\equiv} 139,$

Modular arithmetic

Taking powers

The operation of taking modular powers can be computed efficiently by repeated squarings.

Example : to compute $y \underset{541}{\equiv} 176^{40}$.

Use the fact that $40 = 32 + 8$ and compute by successive squarings :

$b = 176, \; b^2 \underset{541}{\equiv} 139, \; b^4 \underset{541}{\equiv} 386,$

Modular arithmetic

Taking powers

The operation of taking modular powers can be computed efficiently by repeated squarings.

Example : to compute $y \underset{541}{\equiv} 176^{40}$.

Use the fact that $40 = 32 + 8$ and compute by successive squarings :

$b = 176,\ b^2 \underset{541}{\equiv} 139,\ b^4 \underset{541}{\equiv} 386,\ b^8 \underset{541}{\equiv} 221,$

Modular arithmetic

Taking powers

The operation of taking modular powers can be computed efficiently by repeated squarings.

Example : to compute $y \underset{541}{\equiv} 176^{40}$.

Use the fact that $40 = 32 + 8$ and compute by successive squarings :

$$b = 176,\ b^2 \underset{541}{\equiv} 139,\ b^4 \underset{541}{\equiv} 386,\ b^8 \underset{541}{\equiv} 221,\ b^{16} \underset{541}{\equiv} 151,$$

Modular arithmetic

Taking powers

The operation of taking modular powers can be computed efficiently by repeated squarings.

Example : to compute $y \underset{541}{\equiv} 176^{40}$.

Use the fact that $40 = 32 + 8$ and compute by successive squarings :

$$b = 176,\ b^2 \underset{541}{\equiv} 139,\ b^4 \underset{541}{\equiv} 386,\ b^8 \underset{541}{\equiv} 221,\ b^{16} \underset{541}{\equiv} 151,\ b^{32} \underset{541}{\equiv} 79$$

Modular arithmetic
Taking powers

The operation of taking modular powers can be computed efficiently by repeated squarings.

Example : to compute $y \underset{541}{\equiv} 176^{40}$.

Use the fact that $40 = 32 + 8$ and compute by successive squarings :

$$b = 176,\ b^2 \underset{541}{\equiv} 139,\ b^4 \underset{541}{\equiv} 386,\ b^8 \underset{541}{\equiv} 221,\ b^{16} \underset{541}{\equiv} 151,\ b^{32} \underset{541}{\equiv} 79$$

and then

$$b^{40} \underset{541}{\equiv} b^{32} \cdot b^8 \equiv 79 \cdot 221 \equiv 147.$$

Modular arithmetic

Taking powers

Much quicker than first computing

$$\begin{aligned} b^{40} = {} & 66146476117266938411 \\ & 57619437514125541994 \\ & 77293169155435203018 \\ & 62889699022438451615 \\ & 53331941376 \end{aligned}$$

then taking the remainder modulo 541 !

Modular arithmetic

Taking powers

Much quicker than first computing

$$\begin{aligned} b^{40} = {} & 661464761172669384115 \\ & 76194375141255419947 \\ & 72931691554352030186 \\ & 28896990224384516155 \\ & 3331941376 \end{aligned}$$

then taking the remainder modulo 541 !

Number of steps needed is proportional to $\log_2$ of the exponent.

Rivest-Shamir-Adleman (1977)

Given a (large) integer n :

$$\begin{cases} E(e, m) \underset{n}{\equiv} m^e \\ D(d, c) \underset{n}{\equiv} c^d \end{cases}$$

Rivest-Shamir-Adleman (1977)

Given a (large) integer n :

$$\begin{cases} E(e,m) \underset{n}{\equiv} m^e \\ D(d,c) \underset{n}{\equiv} c^d \end{cases}$$

For this to work : we ask that n is square-free and that

$$de \underset{\phi(n)}{\equiv} 1$$

where $\phi(n)$ is the number of integers between 1 and n that are coprime with n (Euler's ϕ function).

Rivest-Shamir-Adleman (1977)

Given a (large) integer n :

$$\begin{cases} E(e,m) \underset{n}{\equiv} m^e \\ D(d,c) \underset{n}{\equiv} c^d \end{cases}$$

For this to work : we ask that n is square-free and that

$$de \underset{\phi(n)}{\equiv} 1$$

where $\phi(n)$ is the number of integers between 1 and n that are coprime with n (Euler's ϕ function).

If $n = p_1 \cdots p_\ell$, then $\phi(n) = (p_1 - 1) \cdots (p_\ell - 1)$.

RSA

A working example

Take

$$n = 3670486004008413084113 77,$$

$$m = 1010101010101010101010,$$

$$e = 3,$$

$$d = 2446990669330863306993 07.$$

RSA

A working example

Take

$$n = 36704860040084130841137 7,$$

$$m = 10101010101010101010101 0,$$

$$e = 3,$$

$$d = 24469906693308633069930 7.$$

Then Alice computes

$$c \underset{n}{\equiv} m^e \underset{n}{\equiv} 28017227544946476129772 7$$

RSA

A working example

Take

$$n = 367048600400841308411377,$$

$$m = 101010101010101010101010,$$

$$e = 3,$$

$$d = 244699066933086330699307.$$

Then Alice computes

$$c \underset{n}{\equiv} m^e \underset{n}{\equiv} 280172275449464761297727$$

and Bob is able to decrypt this to

$$c^d \underset{n}{\equiv} 101010101010101010101010.$$

Knowing e (or d), it is easy to recover the other key in $\log(n)$ steps by using Euclid's algorithm backwards :

$$de + k\phi(n) = 1.$$

Knowing e (or d), it is easy to recover the other key in $\log(n)$ steps by using Euclid's algorithm backwards :

$$de + k\phi(n) = 1.$$

Thus we better make the computation of $\phi(n)$ knowing n as long as possible.

Knowing e (or d), it is easy to recover the other key in $\log(n)$ steps by using Euclid's algorithm backwards :

$$de + k\phi(n) = 1.$$

Thus we better make the computation of $\phi(n)$ knowing n as long as possible.

Best currently known algorithm : first **factor** n as

$$n = p_1 \cdots p_\ell$$

then (trivially) compute

$$\phi(n) = (p_1 - 1) \cdots (p_\ell - 1).$$

RSA
Factoring n

Assume $n = p_1 \cdots p_\ell$ with $p_1 < \ldots < p_\ell$.

Assume $n = p_1 \cdots p_\ell$ with $p_1 < \ldots < p_\ell$.

Naive factorisation algorithm : try dividing n by all successive integers until p_1 is found (then repeat with $\frac{n}{p_1}$).

Assume $n = p_1 \cdots p_\ell$ with $p_1 < \ldots < p_\ell$.

Naive factorisation algorithm : try dividing n by all successive integers until p_1 is found (then repeat with $\frac{n}{p_1}$).

Worst case : have all prime factors as large as possible, $p_i \approx \sqrt[\ell]{n}$.

Assume $n = p_1 \cdots p_\ell$ with $p_1 < \ldots < p_\ell$.

Naive factorisation algorithm : try dividing n by all successive integers until p_1 is found (then repeat with $\frac{n}{p_1}$).

Worst case : have all prime factors as large as possible, $p_i \approx \sqrt[\ell]{n}$.

Might as well take $\ell = 2$! Then $n = pq$ with p, q prime and

$$\phi(n) = (p-1)(q-1).$$

For $c \in \mathbf{R}$ and $\alpha \in [0, 1]$,

$$L_\alpha(n) := \exp\left(c\,(\log n)^\alpha\,(\log\log n)^{1-\alpha}\right).$$

For $c \in \mathbf{R}$ and $\alpha \in [0, 1]$,

$$L_\alpha(n) := \exp\left(c\,(\log n)^\alpha\,(\log\log n)^{1-\alpha}\right).$$

- $L_0(n) = (\log n)^c$

For $c \in \mathbf{R}$ and $\alpha \in [0,1]$,

$$L_\alpha(n) := \exp\left(c\,(\log n)^\alpha\,(\log\log n)^{1-\alpha}\right).$$

- $L_0(n) = (\log n)^c$
- $L_1(n) = n^c$

For $c \in \mathbf{R}$ and $\alpha \in [0,1]$,

$$L_\alpha(n) := \exp\left(c\,(\log n)^\alpha\,(\log\log n)^{1-\alpha}\right).$$

- $L_0(n) = (\log n)^c$
- $L_1(n) = n^c$
- in general $L_\alpha(n)$ is somewhere between these two.

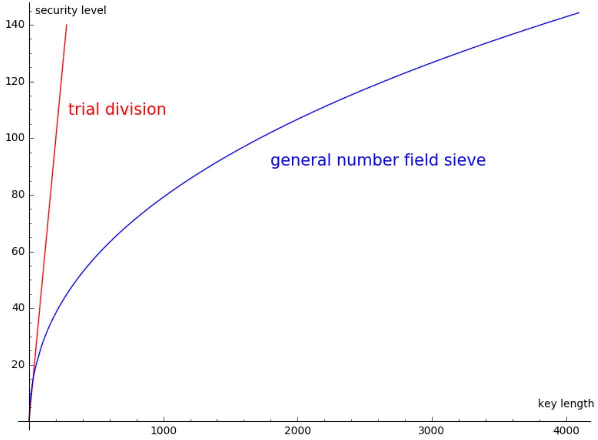
- Trial division (1202) : L_1 with $c = \frac{1}{2}$

- Trial division (1202) : L_1 with $c = \frac{1}{2}$
- Quadratic sieve (1981) : $L_{\frac{1}{2}}$ avec $d = 1$

- Trial division (1202) : L_1 with $c = \frac{1}{2}$
- Quadratic sieve (1981) : $L_{\frac{1}{2}}$ avec $d = 1$
- General number field sieve (1993) : $L_{\frac{1}{3}}$ avec $d \approx 1{,}923$.

RSA

Equivalent symmetric ciphers



Discrete logarithms

Another way to use modular exponentiation : $x \underset{n}{\equiv} g^y$

Discrete logarithms

Another way to use modular exponentiation : $x \underset{n}{\equiv} g^y$

Discrete logarithm problem : find $y = \ll \log_g(x) \gg$

Discrete logarithms

Another way to use modular exponentiation : $x \underset{n}{\equiv} g^y$

Discrete logarithm problem : find $y = \ll \log_g(x) \gg$

Same algorithmic complexity than factoring (DSA, Diffie-Hellman)

Discrete logarithms

Another way to use modular exponentiation : $x \underset{n}{\equiv} g^y$

Discrete logarithm problem : find $y = \ll \log_g(x) \gg$

Same algorithmic complexity than factoring (DSA, Diffie-Hellman)

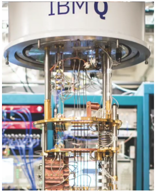
But : $\sqrt{n}$ in more general groups

$\implies$ methods based on elliptic curves (ECDSA, ECDH)

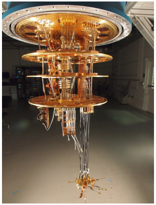
Quantum computers

State of the art

Simulated annealing



IBM : 50 qubits



Google : 70 qubits

Quantum computers

Grover's algorithm

Quantum computers

Grover's algorithm

Quantum computers

Grover's algorithm

Given an arbitrary function $f : A \to B$ with $|A| = n$ and $b \in f(A)$, looking for a preimage $a \in A$ such that $f(a) = b$ classically takes n evaluations.

Given an arbitrary function $f : A \to B$ with $|A| = n$ and $b \in f(A)$, looking for a preimage $a \in A$ such that $f(a) = b$ classically takes n evaluations.

A quantum computer could (*probably*) find one with only $\sqrt{n}$ evaluations of f.

Quantum computers

Grover's algorithm

Given an arbitrary function $f : A \to B$ with $|A| = n$ and $b \in f(A)$, looking for a preimage $a \in A$ such that $f(a) = b$ classically takes n evaluations.

A quantum computer could (*probably*) find one with only $\sqrt{n}$ evaluations of f.

$\implies$ symmetric keys will need to be twice as long

Quantum computers

Shor's algorithm

Quickly finds periods or arbitrary functions.

Quickly finds periods or arbitrary functions.

Look for integers a for which $f(x) \underset{n}{\equiv} a^x$ has even period r.

Quickly finds periods or arbitrary functions.

Look for integers a for which $f(x) \underset{n}{\equiv} a^x$ has even period r.

$$(a^{\frac{r}{2}})^2 \underset{p \cdot q}{\equiv} 1 \implies \begin{cases} a^{\frac{r}{2}} \underset{p}{\equiv} \pm 1 \\ a^{\frac{r}{2}} \underset{q}{\equiv} \pm 1 \end{cases}$$

There is a 50% chance that

$$\gcd(a^{\frac{r}{2}} - 1, n) \quad \text{and} \quad \gcd(a^{\frac{r}{2}} + 1, n)$$

are non-trivial factors of n.

Quantum computers

Shor's algorithm

Factors n in time

$$(\log n)^2(\log \log n)(\log \log \log n)$$

Quantum computers

Shor's algorithm

Factors n in time

$$(\log n)^2(\log\log n)(\log\log\log n)$$

~~RSA~~ ~~(EC)DSA~~ ~~(EC)DH~~

Post-quantum encryption

It is necessary to start thinking today about potential alternatives that could be both efficient *and* secure against a quantum opponent.

Post-quantum encryption

It is necessary to start thinking today about potential alternatives that could be both efficient *and* secure against a quantum opponent.

4 main leads :

Post-quantum encryption

It is necessary to start thinking today about potential alternatives that could be both efficient *and* secure against a quantum opponent.

4 main leads :

- hash-based encryption

Post-quantum encryption

It is necessary to start thinking today about potential alternatives that could be both efficient *and* secure against a quantum opponent.

4 main leads :

- hash-based encryption
- based on error-correcting codes

Post-quantum encryption

It is necessary to start thinking today about potential alternatives that could be both efficient *and* secure against a quantum opponent.

4 main leads :

- hash-based encryption
- based on error-correcting codes
- lattice-based

Post-quantum encryption

It is necessary to start thinking today about potential alternatives that could be both efficient *and* secure against a quantum opponent.

4 main leads :

- hash-based encryption
- based on error-correcting codes
- lattice-based
- hidden field equations.

Post-quantum encryption

It is necessary to start thinking today about potential alternatives that could be both efficient *and* secure against a quantum opponent.

4 main leads :

- hash-based encryption
- based on error-correcting codes
- lattice-based
- hidden field equations.

Ongoing standardization process by NIST.

Consider a lattice $\Lambda = \{ \sum_i a_i \mathbf{v}_i \mid a_i \in \mathbf{Z} \}$ where $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are linearly independant vectors in $\mathbf{R}^n$.

Consider a lattice $\Lambda = \{ \sum_i a_i \mathbf{v}_i \mid a_i \in \mathbf{Z}\}$ where $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are linearly independant vectors in $\mathbf{R}^n$.

Encryption : we add to a message $\mathbf{v} \in \Lambda$ a small perturbation $\mathbf{e} \in \mathbf{R}^n$,

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

Consider a lattice $\Lambda = \{ \sum_i a_i \mathbf{v}_i \mid a_i \in \mathbf{Z} \}$ where $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are linearly independant vectors in $\mathbf{R}^n$.

Encryption : we add to a message $\mathbf{v} \in \Lambda$ a small perturbation $\mathbf{e} \in \mathbf{R}^n$,

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

Decryption : given $\mathbf{c}$, look for the nearest vector $\mathbf{v}$ in the lattice.

Consider a lattice $\Lambda = \{ \sum_i a_i \mathbf{v}_i \mid a_i \in \mathbf{Z}\}$ where $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are linearly independant vectors in $\mathbf{R}^n$.

Encryption : we add to a message $\mathbf{v} \in \Lambda$ a small perturbation $\mathbf{e} \in \mathbf{R}^n$,

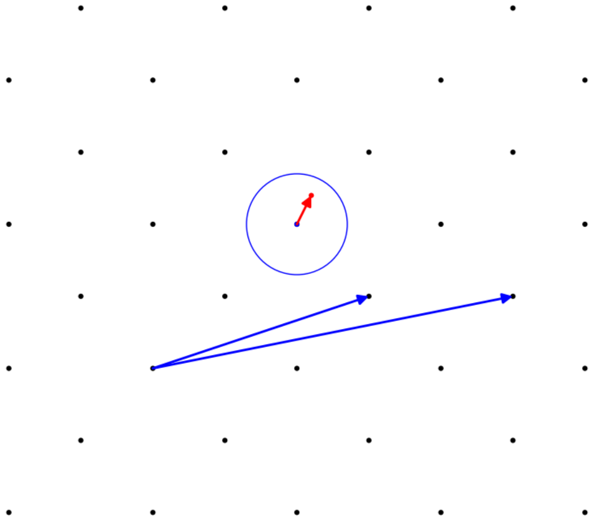
$$\mathbf{c} = \mathbf{v} + \mathbf{e}.$$

Decryption : given $\mathbf{c}$, look for the nearest vector $\mathbf{v}$ in the lattice.

Algorithmically difficult problem if an adapted basis is not known.

Post-quantum encryption

Lattice-based encryption



Thanks !