Code Based Cryptography

Alain Couvreur

INRIA & LIX, École Polytechnique

École de Printemps *Post Scryptum* 2018

Outline

1. Introduction
2. A bit coding theory
   - First definitions
   - A difficult problem
   - Easy instances
3. Code based cryptography
   - McEliece scheme
   - Some examples and proposals
4. Security analysis
   - Message recovery attacks
   - Key recovery attacks

1 Introduction

There is frequently a confusion between:

- Coding theory : resisting to noise;
- Cryptography : resisting to bad persons.

Why is it so confusing?

French language is misleading:

Why is it so confusing?

French language is misleading:

- “message codé”,

Why is it so confusing?

French language is misleading:

- "message codé", we should say **message chiffré**;

Why is it so confusing?

French language is misleading:

- "message codé", we should say **message chiffré**;
- "code secret",

Why is it so confusing?

French language is misleading:

- "message codé", we should say **message chiffré**;
- "code secret", we should say **mot de passe**;

Why is it so confusing?

French language is misleading:

- "message codé", we should say **message chiffré**;
- "code secret", we should say **mot de passe**;
- "décoder un message",

Why is it so confusing?

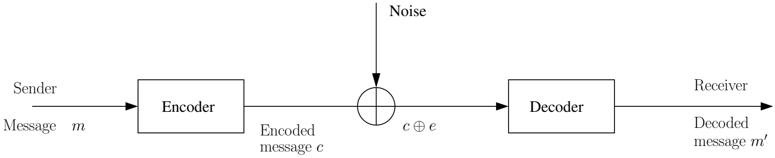
French language is misleading:

- "message codé", we should say **message chiffré**;
- "code secret", we should say **mot de passe**;
- "décoder un message", we should say **déchiffrer** if you are the owner of the key and **décrypter** if you're an eavesdropper.

2. A bit coding theory
   - First definitions
   - A difficult problem
   - Easy instances

Encoders, error correcting codes

Fundamental idea: add redundancy to information.



Definition

An *encoder* is a linear injective map: $\mathbb{F}_{q^k} \hookrightarrow \mathbb{F}_{q^n}$.
An *error correcting code*, or *code* is the image of such a map, i.e. a subspace of dimension k of $\mathbb{F}_q^n$.

Hamming metric

Definition

The *Hamming weight* of $\boldsymbol{x} \in \mathbb{F}_q^n$ is defined as

$$w_H(\boldsymbol{x}) \stackrel{\text{def}}{=} |\{i \in \{1, \dots, n\} \mid x_i \neq 0\}|$$

The *Hamming distance* on $\mathbb{F}_q^n$ is defined by

$$d_H(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} |\{i \mid x_i \neq y_i\}| = w_H(\boldsymbol{x} - \boldsymbol{y}).$$

For instance, $d_H((0,1,0,1,0,1),(0,1,1,1,0,0)) = 2$.

Hamming metric

Definition

The *Hamming weight* of $\boldsymbol{x} \in \mathbb{F}_q^n$ is defined as

$$w_H(\boldsymbol{x}) \stackrel{\text{def}}{=} |\{i \in \{1, \dots, n\} \mid x_i \neq 0\}|$$

The *Hamming distance* on $\mathbb{F}_q^n$ is defined by

$$d_H(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} |\{i \mid x_i \neq y_i\}| = w_H(\boldsymbol{x} - \boldsymbol{y}).$$

For instance, $d_H((0, 1, 0, 1, 0, 1), (0, 1, 1, 1, 0, 0)) = 2$.

Decoders

Definition

Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ and $t \leqslant n$. A (deterministic) *t–decoder* is a map $\mathcal{D} : \mathbb{F}_q^n \longrightarrow \mathscr{C} \cup \{?\}$ such that $\forall \boldsymbol{c} \in \mathscr{C}$ and all $\boldsymbol{e} \in \mathbb{F}_q^n$ with $w_H(\boldsymbol{e}) \leqslant t$, we have

$$\mathcal{D}(\boldsymbol{c} + \boldsymbol{e}) = \boldsymbol{c}.$$

Decoders

Definition

Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ and $t \leqslant n$. A (deterministic) *t–decoder* is a map $\mathcal{D} : \mathbb{F}_q^n \longrightarrow \mathscr{C} \cup \{?\}$ such that $\forall \boldsymbol{c} \in \mathscr{C}$ and all $\boldsymbol{e} \in \mathbb{F}_q^n$ with $w_H(\boldsymbol{e}) \leqslant t$, we have

$$\mathcal{D}(\boldsymbol{c} + \boldsymbol{e}) = \boldsymbol{c}.$$

Definition

Let $\mathscr{C} \subseteq \mathbb{F}_q^n$, $t \leqslant n$ and $0 < \varepsilon < 1$. A *probabilistic t–decoder of failure probability* ε is a map $\mathcal{D} : \mathbb{F}_q^n \longrightarrow \mathscr{C} \cup \{?\}$ such that for a uniformly random $\boldsymbol{c} \in \mathscr{C}$ and a uniformly random $\boldsymbol{e} \in \mathbb{F}_q^n$ of weight t, we have

$$\mathbb{P}(\mathcal{D}(\boldsymbol{c} + \boldsymbol{e}) = \boldsymbol{c}) \geqslant 1 - \varepsilon.$$

Almost every code is good but...

Shannon Theorem (very informally) asserts that for almost any code $\mathscr{C}$ of dimension $k = Rn$ for some $R > 0$, there is a probabilistic t–decoder with $t = \tau n$ for some $\tau > 0$ (τ depends on R) with failure probability $2^{-O(n)}$.

Almost every code is good but...

Shannon Theorem (very informally) asserts that for almost any code $\mathscr{C}$ of dimension $k = Rn$ for some $R > 0$, there is a probabilistic t–decoder with $t = \tau n$ for some $\tau > 0$ (τ depends on R) with failure probability $2^{-O(n)}$.

But the theorem does **not** assert the existence of a **polynomial time** decoder.

Almost every code is good but...

Shannon Theorem (very informally) asserts that for almost any code $\mathscr{C}$ of dimension $k = Rn$ for some $R > 0$, there is a probabilistic t–decoder with $t = \tau n$ for some $\tau > 0$ (τ depends on R) with failure probability $2^{-O(n)}$.

But the theorem does **not** assert the existence of a **polynomial time** decoder.

Moreover, we have:

Theorem (Berlekamp, McEliece, Van Tilbørg, 1978)

The following problem is NP–complete.

Problem. *(*Bounded decoding Problem*) Let* $\mathscr{C} \subseteq \mathbb{F}_q^n$, $\mathbf{y} \in \mathbb{F}_q^n$ *and* $t \in \{0, \ldots, n\}$. *Decide whether there exists* $\mathbf{c} \in \mathscr{C}$ *such that* $d_H(\mathbf{c}, \mathbf{y}) \leqslant t$.

Easy instances

- **Algebraic coding (with deterministic decoders)**
  - Reed–Muller Codes;
  - Reed–Solomon and alternant codes;
  - Algebraic geometry codes;
  - etc...
- **Probabilistic coding (with probabilistic decoders)**
  - LDPC codes (Gallager codes);
  - Turbo-codes.

Reed–Solomon Codes

Definition

Let $x_1, \ldots, x_n$ be distinct elements of $\mathbb{F}_q$. The *Reed Solomon code* of dimension k and *support* $\boldsymbol{x}$ is:

$$\mathbf{RS}_k(\boldsymbol{x}) \stackrel{\text{def}}{=} \left\{(f(x_1), \ldots, f(x_n)) \mid f \in \mathbb{F}_q[X]_{<k}\right\}.$$

Generalised Reed–Solomon Codes and Altenant codes

Definition

Let $x_1, \ldots, x_n$ be distinct elements of $\mathbb{F}_q$ and $y_1, \ldots, y_n$ be nonzero elements of $\mathbb{F}_q$. The *generalised Reed Solomon code* of dimension k, *support* $\boldsymbol{x}$ and *multiplier* $\boldsymbol{y}$ is:

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} \{(y_1 f(x_1), \ldots, y_n f(x_n)) \mid f \in \mathbb{F}_q[X]_{<k}\}.$$

Generalised Reed–Solomon Codes and Altenant codes

Definition

Let $x_1, \ldots, x_n$ be distinct elements of $\mathbb{F}_q$ and $y_1, \ldots, y_n$ be nonzero elements of $\mathbb{F}_q$. The *generalised Reed Solomon code* of dimension k, *support* $\boldsymbol{x}$ and *multiplier* $\boldsymbol{y}$ is:

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} \{(y_1 f(x_1), \ldots, y_n f(x_n)) \mid f \in \mathbb{F}_q[X]_{<k}\}.$$

Definition

An *Alternant code* is a code over $\mathbb{F}_q$ of the form

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \cap \mathbb{F}_q^n$$

for some GRS code over $\mathbb{F}_{q^m}$.

Generalised Reed–Solomon Codes and Altenant codes

Definition

Let $x_1, \ldots, x_n$ be distinct elements of $\mathbb{F}_q$ and $y_1, \ldots, y_n$ be nonzero elements of $\mathbb{F}_q$. The *generalised Reed Solomon code* of dimension k, *support* $\boldsymbol{x}$ and *multiplier* $\boldsymbol{y}$ is:

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} \{(y_1 f(x_1), \ldots, y_n f(x_n)) \mid f \in \mathbb{F}_q[X]_{<k}\}.$$

Definition

An *Alternant code* is a code over $\mathbb{F}_q$ of the form

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \cap \mathbb{F}_q^n$$

for some GRS code over $\mathbb{F}_{q^m}$.

Most of the families of algebraic codes **are alternant codes** : *Goppa codes, Srivastava codes, BCH codes, etc...*

Decoding RS codes – Berlekamp Welch algorithm

Let $\boldsymbol{c} = (f(x_1), \dots, f(x_n)) \in \mathbf{RS}_k(\boldsymbol{x})$. Let $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ with $d_H(\boldsymbol{r}, \boldsymbol{c}) \leqslant t$.

- $\boldsymbol{r}$ is known;
- We aim at computing $\boldsymbol{c}$.

Decoding RS codes – Berlekamp Welch algorithm

Let $\boldsymbol{c} = (f(x_1), \dots, f(x_n)) \in \mathbf{RS}_k(\boldsymbol{x})$. Let $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ with $d_H(\boldsymbol{r}, \boldsymbol{c}) \leqslant t$.

- $\boldsymbol{r}$ is known;
- We aim at computing $\boldsymbol{c}$.

Step 1 Compute the polynomial $P = P_0(X) + P_1(X)Y \in \mathbb{F}_q[X, Y]$ satisfying

(i) $\deg P_0 < n - t$, $\deg P_1 < n - k - t$

(ii) $\forall i \in \{1, \dots, n\}, P(x_i, r_i) = 0$.

Decoding RS codes – Berlekamp Welch algorithm

Let $\boldsymbol{c} = (f(x_1), \dots, f(x_n)) \in \mathbf{RS}_k(\boldsymbol{x})$. Let $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ with $d_H(\boldsymbol{r}, \boldsymbol{c}) \leqslant t$.

- $\boldsymbol{r}$ is known;
- We aim at computing $\boldsymbol{c}$.

Step 1 Compute the polynomial $P = P_0(X) + P_1(X)Y \in \mathbb{F}_q[X, Y]$ satisfying

(i) $\deg P_0 < n - t$, $\deg P_1 < n - k - t$

(ii) $\forall i \in \{1, \dots, n\}, P(x_i, r_i) = 0$.

Step 2 If $t \leqslant \frac{n-k}{2}$, then $f = -\frac{P_0}{P_1}$

Decoding RS codes – Berlekamp Welch algorithm

Let $\boldsymbol{c} = (f(x_1), \ldots, f(x_n)) \in \mathbf{RS}_k(\boldsymbol{x})$. Let $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ with $d_H(\boldsymbol{r}, \boldsymbol{c}) \leqslant t$.

- $\boldsymbol{r}$ is known;
- We aim at computing $\boldsymbol{c}$.

Step 1 Compute the polynomial $P = P_0(X) + P_1(X)Y \in \mathbb{F}_q[X, Y]$ satisfying

(i) $\deg P_0 < n - t$, $\deg P_1 < n - k - t$

(ii) $\forall i \in \{1, \ldots, n\}, P(x_i, r_i) = 0$.

Step 2 If $t \leqslant \frac{n-k}{2}$, then $f = -\frac{P_0}{P_1}$

Proof.

$P(X, f(X))$ has degree $< n - t$ and has $\geqslant n - t$ roots. Hence is zero. □

LDPC codes

Low **D**ensity **P**arity **C**heck codes. Informally they are codes which are the kernel of a "sparse" matrix.

Definition

A sequence $(\mathscr{C}_s)_{s \in \mathbb{N}}$ of codes whose length sequence $(n_s)_s$ tends to infinity is said to be LDPC (resp MDPC[a]) is for any s, $\mathscr{C}_s$ is the kernel of a matrix $\boldsymbol{H}_s$ whose row weight is in $O(1)$ (resp $O(\sqrt{n_s})$).

[a] the 'M' stands for *moderate*

Decoding LDPC codes – the bit flipping algorithm

Toy example. Consider the binary code defined as the kernel of:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

Consider a codeword $\boldsymbol{c}$.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{c} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

Consider a codeword $\boldsymbol{c}$... with some errors.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & \color{red}{1} & 0 & 0 & 1 & \color{red}{0} \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

Color the rows of $\boldsymbol{H}$ which have an odd number of 1 in common with $\boldsymbol{y}$.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

For each index, count the number of blue 1's in the corresponding column.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 \\ \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 \\ \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}1 \\ \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}0 & \color{blue}1 & \color{blue}1 \end{pmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & \color{red}1 & 0 & 0 & 1 & \color{red}0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

Flip bits matching with the largest number of unsatisfied rows.

$$
\boldsymbol{H} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
$$

$$
\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}
$$

$$
\begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 \end{pmatrix}
$$

Decoding LDPC codes – the bit flipping algorithm

Flip bits matching with the largest number of unsatisfied rows.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

Flip bits matching with the largest number of unsatisfied rows.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

Flip bits matching with the largest number of unsatisfied rows.

$$
\boldsymbol{H} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
$$

$$
\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 \end{pmatrix}
$$

Decoding LDPC codes – the bit flipping algorithm

Reset counters.

$$
\boldsymbol{H} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
$$

$$
\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} ? & ? & ? & ? & ? & ? & ? & ? & ? \end{pmatrix}
$$

Decoding LDPC codes – the bit flipping algorithm

An error remains.

$$
\boldsymbol{H} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
$$

$$
\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & \color{red}{1} & 0 & 0 & 1 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} ? & ? & ? & ? & ? & ? & ? & ? & ? \end{pmatrix}
$$

Decoding LDPC codes – the bit flipping algorithm

One more time!

$$
\boldsymbol{H} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
$$

$$
\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & {\color{red}1} & 0 & 0 & 1 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} ? & ? & ? & ? & ? & ? & ? & ? & ? \end{pmatrix}
$$

Decoding LDPC codes – the bit flipping algorithm

Color unsatisfied rows in blue.

$$
\boldsymbol{H} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 \\
\color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
$$

$$
\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & \color{red}1 & 0 & 0 & 1 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} ? & ? & ? & ? & ? & ? & ? & ? & ? \end{pmatrix}
$$

Decoding LDPC codes – the bit flipping algorithm

Count number of blue ones per column.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

Flip bits...

$$
\boldsymbol{H} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 \\
\color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
$$

$$
\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & \color{red}1 & 0 & 0 & 1 & 1 \end{pmatrix}
$$

$$
\begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \end{pmatrix}
$$

Decoding LDPC codes – the bit flipping algorithm

Flip bits...

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 \\ \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}0 & \color{blue}1 & \color{blue}1 & \color{blue}1 & \color{blue}0 & \color{blue}0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & \color{red}0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Decoding LDPC codes – the bit flipping algorithm

We did it!!!

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

LDPC/MDPC codes – How many errors?

- A random LDPC code (row weight w in $O(1)$) corrects $\Theta(n)$ errors w.h.p. For $\dim C = \frac{n}{2}$, standard LDPC codes correct $\approx 0.10n$ errors (Shannon limit is $\approx 0.11n$).
- Almost any MDPC code (row weight $O(\sqrt{n})$) corrects **any** pattern of $\Omega\left(\frac{\sqrt{n}\log\log n}{\log n}\right)$ (Tillich, 2017).

3 Code based cryptography

- McEliece scheme
- Some examples and proposals

McEliece Scheme (1978)

It is a public key encryption scheme based on the hardness of the bounded decoding problem.

McEliece Scheme (1978)

It is a public key encryption scheme based on the hardness of the bounded decoding problem.

- **Public key:** $(\boldsymbol{G}, t)$;
  `// (The rows of` $\boldsymbol{G}$ `span some code` $\mathscr{C}$`).`

McEliece Scheme (1978)

It is a public key encryption scheme based on the hardness of the bounded decoding problem.

- **Public key:** $(\boldsymbol{G}, t)$;
  `// (The rows of` $\boldsymbol{G}$ `span some code` $\mathscr{C}$`).`
- **Secret key:** An efficient decoding algorithm $\mathcal{A}$ for $\mathscr{C}$.

McEliece Scheme (1978)

It is a public key encryption scheme based on the hardness of the bounded decoding problem.

- **Public key:** $(\boldsymbol{G}, t)$;
  `// (The rows of` $\boldsymbol{G}$ `span some code` $\mathscr{C}$`).`
- **Secret key:** An efficient decoding algorithm $\mathcal{A}$ for $\mathscr{C}$.
- **Encryption:** Plaintext: $\boldsymbol{m} \in \mathbb{F}_q^k$.
  - $\boldsymbol{mG} \in \mathscr{C}$

McEliece Scheme (1978)

It is a public key encryption scheme based on the hardness of the bounded decoding problem.

- **Public key:** $(\boldsymbol{G}, t)$;
  `// (The rows of` $\boldsymbol{G}$ `span some code` $\mathscr{C}$`).`
- **Secret key:** An efficient decoding algorithm $\mathcal{A}$ for $\mathscr{C}$.
- **Encryption:** Plaintext: $\boldsymbol{m} \in \mathbb{F}_q^k$.
  - $\boldsymbol{mG} \in \mathscr{C}$
  - $\boldsymbol{e} \in \mathbb{F}_q^n$ is a uniformly random word of weight t;

McEliece Scheme (1978)

It is a public key encryption scheme based on the hardness of the bounded decoding problem.

- **Public key:** $(\boldsymbol{G}, t)$;
  `// (The rows of` $\boldsymbol{G}$ `span some code` $\mathscr{C}$`).`
- **Secret key:** An efficient decoding algorithm $\mathcal{A}$ for $\mathscr{C}$.
- **Encryption:** Plaintext: $\boldsymbol{m} \in \mathbb{F}_q^k$.
  - $\boldsymbol{mG} \in \mathscr{C}$
  - $\boldsymbol{e} \in \mathbb{F}_q^n$ is a uniformly random word of weight t;
  - Ciphertext:
$$\boldsymbol{c} \stackrel{\text{def}}{=} \boldsymbol{mG} + \boldsymbol{e}.$$

McEliece Scheme (1978)

It is a public key encryption scheme based on the hardness of the bounded decoding problem.

- **Public key:** $(\boldsymbol{G}, t)$;
  `// (The rows of` $\boldsymbol{G}$ `span some code` $\mathscr{C}$`).`
- **Secret key:** An efficient decoding algorithm $\mathcal{A}$ for $\mathscr{C}$.
- **Encryption:** Plaintext: $\boldsymbol{m} \in \mathbb{F}_q^k$.
  - $\boldsymbol{mG} \in \mathscr{C}$
  - $\boldsymbol{e} \in \mathbb{F}_q^n$ is a uniformly random word of weight t;
  - Ciphertext:
  $$\boldsymbol{c} \stackrel{\text{def}}{=} \boldsymbol{mG} + \boldsymbol{e}.$$
- **Decryption:** Apply $\mathcal{A}$ to recover $\boldsymbol{m}$.

McEliece presented in the literature

- **Secret key.**
  - $\boldsymbol{G}$, a structured $k \times n$ matrix whose rows span a code $\mathscr{C}$;
  - $\boldsymbol{S} \in \mathbf{GL}_k$;
  - $\boldsymbol{P} \in \mathfrak{S}_n$.
- **Public key.** $(\boldsymbol{SGP}, t)$;
- **Encryption** $\boldsymbol{m} \mapsto \boldsymbol{mSGP} + \boldsymbol{e}$ for a uniformly random $\boldsymbol{e}$ of weight t;
- **Decryption**
  - Right multiply by $\boldsymbol{P}^{-1}$: $\boldsymbol{mSGP} + \boldsymbol{e} \longmapsto \boldsymbol{mSG} + e\boldsymbol{P}^{-1}$;
  - decode to get $\boldsymbol{mS}$;
  - right multiply it by $\boldsymbol{S}^{-1}$ to get $\boldsymbol{m}$.

I prefer this presentation

It is a public key encryption scheme based on the hardness of the bounded decoding problem.

- **Public key:** $(\boldsymbol{G}, t)$;
  `# (`$\boldsymbol{G}$` is a generator matrix of some code `$\mathscr{C}$`, i.e. its rows span `$\mathscr{C}$`).`
- **Secret key:** An efficient decoding algorithm $\mathcal{A}$.
- **Encryption:** Plaintext: $\boldsymbol{m} \in \mathbb{F}_q^k$.
  - $\boldsymbol{mG} \in \mathscr{C}$
  - $\boldsymbol{e} \in \mathbb{F}_q^n$ is a uniformly random word of weight t;
  - Ciphertext:

$$\boldsymbol{c} \stackrel{\text{def}}{=} \boldsymbol{mG} + \boldsymbol{e}$$

- **Decryption:** Apply $\mathcal{A}$ to recover $\boldsymbol{m}$.

Advantages and drawbacks

Avantages

- Fast encryption and decryption.
- Post quantum.

Advantages and drawbacks

Avantages

- Fast encryption and decryption.
- Post quantum.

Drawbacks

- Requires large key sizes : Historical proposal (1978) 32 kB key.
- But, impressive improvements in the last decades.

NIST's call

	Signatures		KEM/Encryption		Overall
Lattice-based	CRYSTALS-DILITHIUM DRS FALCON pqNTRUSign qTESLA	5	Compact LWE CRYSTALS-KYBER Ding Key Exchange EMBLEM and R.EMBLEM FrodoKEM Giophantus HILA5 KINDI LAC LIMA Lizard LOTUS NewHope NTRUEncrypt NTRU-HRSS-KEM NTRU Prime Odd Manhattan KCL (pka OKCN/AKCN/CNKE) Round2 SABER Titanium	21	26
Code-based	pqsigRM RaCoSS RankSign	3	BIG QUAKE BIKE Classic McEliece DAGS Edon-K HQC LAKE LEDAkem LEDApkc Lepton LOCKER McNie NTS-KEM Ouroboros-R QC-MDPC KEM RLCE-KEM RQC	17	20
Multi-variate	DualModeMS GeMSS Gui HiMQ-3 MQDSS LUOV Rainbow	7	CFPKM DME	2	9
Hash-based	Gravity-SPHINCS Picnic SPHINCS+	3			3
Others	Post-quantum RSA-Signature WalnutDSA	2	Guess Again Mersenne-756839 Post-quantum RSA-Encryption Ramstake SIKE Three Bears	6	8
Total		20		46	66
withdrawn			*HK17 *RVB *SRTPI	3	3

Classic McEliece

Historical proposal with binary Goppa codes.

- Goppa codes are Alternant codes $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \cap \mathbb{F}_2^n$ with a particular relation between $\boldsymbol{x}$ and $\boldsymbol{y}$;
- They permit to correct twice the number of errors that can correct other binary alternant codes.

Classic McEliece

Historical proposal with binary Goppa codes.

- Goppa codes are Alternant codes $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \cap \mathbb{F}_2^n$ with a particular relation between $\boldsymbol{x}$ and $\boldsymbol{y}$;
- They permit to correct twice the number of errors that can correct other binary alternant codes.
- Specifications:
  - **Public key.** Some basis of $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \cap \mathbb{F}_q^n$ and a number of errors t you can correct (namely $t = \frac{n-k}{2}$).
  - **Secret key.** The pair $(\boldsymbol{x}, \boldsymbol{y})$: it permits to construct $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})$ which is used for decoding.

Classic McEliece

Notation

By an $[n, k]_q$ code, we mean "a code of dimension k in $\mathbb{F}_q^n$".

[1] According to R. Canto Torres C Library **CaWoF**

Classic McEliece

Notation

By an $[n, k]_q$ code, we mean "a code of dimension k in $\mathbb{F}_q^n$".

- Historical proposal. McEliece 1978. A $[1024, 524]_2$ 50–correcting Goppa code.
    - Key size : 32,8 kB;
    - Security : 54 bits (in 1978, $\approx$ 46 bits today[1]).

[1] According to R. Canto Torres C Library **CaWoF**

Classic McEliece

Notation

By an $[n, k]_q$ code, we mean "a code of dimension k in $\mathbb{F}_q^n$".

- Historical proposal. McEliece 1978. A $[1024, 524]_2$ 50–correcting Goppa code.
  - Key size : 32,8 kB;
  - Security : 54 bits (in 1978, $\approx$ 46 bits today[1]).
- Bernstein, Lange, Peters 2008. A $[2048, 1751]_2$ 27–correcting Goppa code.
  - Key size : 65 kB;
  - Security : 80 bits (actually $\approx$ 61 bits today[1]).

[1] According to R. Canto Torres C Library **CaWoF**

Classic McEliece

Notation

By an $[n, k]_q$ code, we mean "a code of dimension k in $\mathbb{F}_q^n$".

- Historical proposal. McEliece 1978. A $[1024, 524]_2$ 50–correcting Goppa code.
  - Key size : 32,8 kB;
  - Security : 54 bits (in 1978, $\approx$ 46 bits today[1]).
- Bernstein, Lange, Peters 2008. A $[2048, 1751]_2$ 27–correcting Goppa code.
  - Key size : 65 kB;
  - Security : 80 bits (actually $\approx$ 61 bits today[1]).
- Classic McEliece (Bernstein et al. NIST proposal). A $[6960, 5296]_2$ 119–correcting Goppa code.
  - Key :1.1MB
  - Security $>$ 256 bits.

[1] According to R. Canto Torres C Library **CaWoF**

McEliece with compact keys

First suggested by Gaborit in 2005, the use codes with a non trivial group automorphism G permits to divide the key size by $|G|$.

- Faugère et. al. 2016, Barelli 2017. *The security of the key of an alternant code with automorphism group G is not larger than that of the G–invariant subcode.*

McEliece with compact keys

First suggested by Gaborit in 2005, the use codes with a non trivial group automorphism G permits to divide the key size by $|G|$.

- Faugère et. al. 2016, Barelli 2017. *The security of the key of an alternant code with automorphism group G is not larger than that of the G–invariant subcode.*
- Actually, for the currently known attacks, the security w.r.t key-recovery attacks is much larger than the security w.r.t message recovery attacks.

McEliece with compact keys

NIST proposals :

- **DAGS** (E. Persichetti et. al.) Group $G = (\mathbb{Z}/2\mathbb{Z})^s$ with $s \in \{4, 5, 6\}$. Based on Generalised Srivastava codes (particular alternant).

security	n	k	$\mathbb{F}_q$	G	key size (kBytes)
128	832	416	$\mathbb{F}_{32}$	$(\mathbb{Z}/2\mathbb{Z})^4$	6.8
192	1216	512	$\mathbb{F}_{64}$	$(\mathbb{Z}/2\mathbb{Z})^5$	8.5
256	2112	704	$\mathbb{F}_{64}$	$(\mathbb{Z}/2\mathbb{Z})^6$	11.6

McEliece with compact keys

NIST proposals :

- **DAGS** (E. Persichetti et. al.) Group $G = (\mathbb{Z}/2\mathbb{Z})^s$ with $s \in \{4, 5, 6\}$. Based on Generalised Srivastava codes (particular alternant).

security	n	k	$\mathbb{F}_q$	G	key size (kBytes)
128	832	416	$\mathbb{F}_{32}$	$(\mathbb{Z}/2\mathbb{Z})^4$	6.8
192	1216	512	$\mathbb{F}_{64}$	$(\mathbb{Z}/2\mathbb{Z})^5$	8.5
256	2112	704	$\mathbb{F}_{64}$	$(\mathbb{Z}/2\mathbb{Z})^6$	11.6

- **BIG QUAKE** (C- et. al.) Group $\mathbb{Z}/\ell\mathbb{Z}$ with ℓ prime and primitive modulo 2. Based on Goppa codes.

security	n	k	$\mathbb{F}_q$	G	key size (kBytes)
128	3510	2418	$\mathbb{F}_2$	$\mathbb{Z}/13\mathbb{Z}$	25.4
192	7410	4674	$\mathbb{F}_2$	$\mathbb{Z}/19\mathbb{Z}$	84.1
256	10070	6650	$\mathbb{F}_2$	$\mathbb{Z}/19\mathbb{Z}$	149.6

LDPC/MDPC codes

History

- Monico, Rosenthal, Shokrollahi (2000). Suggest the use of LDPC codes for McEliece encryption.
- Baldi M., Bodrato M., Chiaraluce F. (2008).
  - use quasi-cyclic LDPC codes to get shorter keys.
  - "deforms" the LDPC structure to resist against a key-recovery attack by computing low weight codewords.
- Misoczki, Tillich, Sendrier, Baretto (2012). Use quasi–cyclic MDPC codes.

MDPC codes – BIKE (BIt flipping Key Encapsulation)

- The code is the row space of a **sparse** doubly circulant matrix:

$$\begin{pmatrix} f_0 & f_1 & \cdots & \cdots & f_{n-1} & g_0 & g_1 & \cdots & \cdots & g_{n-1} \\ & f_0 & f_1 & \cdots & f_{n-2} & & g_0 & g_1 & \cdots & g_{n-2} \\ & & \ddots & \ddots & \vdots & & & \ddots & \ddots & \vdots \\ & & & \ddots & f_1 & & & & \ddots & g_1 \\ f_1 & f_2 & \cdots & f_{n-1} & f_0 & g_1 & g_2 & \cdots & g_{n-1} & g_0 \end{pmatrix}$$

MDPC codes – BIKE (BIt flipping Key Encapsulation)

- The code is the row space of a **sparse** doubly circulant matrix:

$$\begin{pmatrix} f_0 & f_1 & \cdots & \cdots & f_{n-1} & g_0 & g_1 & \cdots & \cdots & g_{n-1} \\ & f_0 & f_1 & \cdots & f_{n-2} & & g_0 & g_1 & \cdots & g_{n-2} \\ & & \ddots & \ddots & \vdots & & & \ddots & \ddots & \vdots \\ & & & \ddots & f_1 & & & & \ddots & g_1 \\ f_1 & f_2 & \cdots & f_{n-1} & f_0 & g_1 & g_2 & \cdots & g_{n-1} & g_0 \end{pmatrix}$$

- It can be represented by two sparse (weight $O(\sqrt{n})$) polynomials in $\mathbb{F}_2[X]/(X^n - 1)$

$$(\ f(X)\ \mid\ g(X)\ )$$

MDPC codes – BIKE (BIt flipping Key Encapsulation)

- It can represented by two sparse (weight $O(\sqrt{n})$) polynomials in $\mathbb{F}_2[X]/(X^n - 1)$

$$\left(\ f(X)\ \middle|\ g(X)\ \right)$$

MDPC codes – BIKE (BIt flipping Key Encapsulation)

- It can represented by two sparse (weight $O(\sqrt{n})$) polynomials in $\mathbb{F}_2[X]/(X^n - 1)$

$$(\ f(X) \ \mid \ g(X) \ )$$

- The **public key** is the reduced row echelon form is of the form:

$$(\ 1 \ \mid \ h(X) \ )$$

where $h \equiv f^{-1}g \mod (X^n - 1)$.

MDPC codes – BIKE (BIt flipping Key Encapsulation)

- It can represented by two sparse (weight $O(\sqrt{n})$) polynomials in $\mathbb{F}_2[X]/(X^n - 1)$
$$(\ f(X) \ \mid \ g(X) \ )$$
- The **public key** is the reduced row echelon form is of the form:
$$(\ 1 \ \mid \ h(X) \ )$$
where $h \equiv f^{-1}g \mod (X^n - 1)$.
- The **secret key** is the pair (f, g).

MDPC codes – BIKE (BIt flipping Key Encapsulation)

- It can represented by two sparse (weight $O(\sqrt{n})$) polynomials in $\mathbb{F}_2[X]/(X^n - 1)$
$$(\ f(X) \ \mid \ g(X) \ )$$
- The **public key** is the reduced row echelon form is of the form:
$$(\ 1 \ \mid \ h(X) \ )$$
where $h \equiv f^{-1}g \mod (X^n - 1)$.
- The **secret key** is the pair (f, g).
- **Comment.** f and g are sparse, h has no apparent structure. (same as NTRU or Mersenne).

MDPC codes – BIKE (BIt flipping Key Encapsulation)

Parameters.

Security	n	k	row weight	t	Key size (kB)
128	20326	10163	142	134	1.25
192	39706	19853	206	199	2.5
256	65498	32749	274	264	4.1

Broken schemes

Broken schemes

- GRScodes, (Niederreiter 1986).

Broken schemes

- GRScodes, (Niederreiter 1986).
  [Polynomial time attack: Sidelnikov Shestakov 1992]

Broken schemes

- GRScodes, (Niederreiter 1986).
  [Polynomial time attack: Sidelnikov Shestakov 1992]
- Reed-Muller codes (Sidelinikov 1994)

Broken schemes

- GRScodes, (Niederreiter 1986).

  [Polynomial time attack: Sidelnikov Shestakov 1992]
- Reed-Muller codes (Sidelinikov 1994)

  [Sub-exponential time attack: Minder, Shokrollahi 2007]

  [Polynomial time attack: Borodin, Chizhov, 2013]

Broken schemes

- GRScodes, (Niederreiter 1986).
  [Polynomial time attack: Sidelnikov Shestakov 1992]
- Reed-Muller codes (Sidelinikov 1994)
  [Sub-exponential time attack: Minder, Shokrollahi 2007]
  [Polynomial time attack: Borodin, Chizhov, 2013]
- Algebraic Geometry codes, (Janwa, Moreno 1996):

Broken schemes

- GRScodes, (Niederreiter 1986).
  [Polynomial time attack: Sidelnikov Shestakov 1992]
- Reed-Muller codes (Sidelinikov 1994)
  [Sub-exponential time attack: Minder, Shokrollahi 2007]
  [Polynomial time attack: Borodin, Chizhov, 2013]
- Algebraic Geometry codes, (Janwa, Moreno 1996):
  [Curves of genus 1 and 2 : Faure, Minder 2009]
  [Any genus: C. Márquez–Corbella, Pellikaan 2014]

Broken schemes

- GRScodes, (Niederreiter 1986).
  [Polynomial time attack: Sidelnikov Shestakov 1992]
- Reed-Muller codes (Sidelinikov 1994)
  [Sub-exponential time attack: Minder, Shokrollahi 2007]
  [Polynomial time attack: Borodin, Chizhov, 2013]
- Algebraic Geometry codes, (Janwa, Moreno 1996):
  [Curves of genus 1 and 2 : Faure, Minder 2009]
  [Any genus: C. Márquez–Corbella, Pellikaan 2014]
- q–ary Goppa codes (Bernstein, Lange, Peters, 2011).

Broken schemes

- GRScodes, (Niederreiter 1986).
  [Polynomial time attack: Sidelnikov Shestakov 1992]
- Reed-Muller codes (Sidelinikov 1994)
  [Sub-exponential time attack: Minder, Shokrollahi 2007]
  [Polynomial time attack: Borodin, Chizhov, 2013]
- Algebraic Geometry codes, (Janwa, Moreno 1996):
  [Curves of genus 1 and 2 : Faure, Minder 2009]
  [Any genus: C. Márquez–Corbella, Pellikaan 2014]
- q–ary Goppa codes (Bernstein, Lange, Peters, 2011).
  [Extension degree 2 : C. Otmani, Tillich 2015]
  [Degree 2 and 3 over non prime fields : Faugère, Perret, de Portzamparc, 2015]

Broken schemes

- GRScodes, (Niederreiter 1986).
  [Polynomial time attack: Sidelnikov Shestakov 1992]
- Reed-Muller codes (Sidelinikov 1994)
  [Sub-exponential time attack: Minder, Shokrollahi 2007]
  [Polynomial time attack: Borodin, Chizhov, 2013]
- Algebraic Geometry codes, (Janwa, Moreno 1996):
  [Curves of genus 1 and 2 : Faure, Minder 2009]
  [Any genus: C. Márquez–Corbella, Pellikaan 2014]
- q–ary Goppa codes (Bernstein, Lange, Peters, 2011).
  [Extension degree 2 : C. Otmani, Tillich 2015]
  [Degree 2 and 3 over non prime fields : Faugère, Perret, de Portzamparc, 2015]
- etc...

4. Security analysis
   - Message recovery attacks
   - Key recovery attacks

Message Recovery attacks

The bounded decoding problem is hard. Decoding a random code is difficult, the best known generic algorithms have exponential complexity.

Prange Algorithm (Information Set Decoding) 1962

Let $\mathscr{C}$ be an $[n, k]$ code described as the row space of a matrix $\boldsymbol{G}$. Suppose we received $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{e}$ of weight $\leqslant t$.

Prange Algorithm (Information Set Decoding) 1962

Let $\mathscr{C}$ be an $[n, k]$ code described as the row space of a matrix $\boldsymbol{G}$. Suppose we received $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{e}$ of weight $\leqslant t$.

1. Take a k–tuple $i_1, \ldots, i_k$ of columns of $\boldsymbol{G}$ such that the corresponding $k \times k$ matrix is invertible and suppose that none of these positions have errors ($e_{i_j} = 0$ for any $j \in \{1, \ldots, k\}$).

Prange Algorithm (Information Set Decoding) 1962

Let $\mathscr{C}$ be an $[n, k]$ code described as the row space of a matrix $\boldsymbol{G}$. Suppose we received $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{e}$ of weight $\leqslant t$.

1. Take a k–tuple $i_1, \ldots, i_k$ of columns of $\boldsymbol{G}$ such that the corresponding $k \times k$ matrix is invertible and suppose that none of these positions have errors ($e_{i_j} = 0$ for any $j \in \{1, \ldots, k\}$).
2. We supposed that $y_{i_j} = c_{i_j}$ for any $j \in \{1, \ldots, k\}$. Then, $\boldsymbol{c}$ can be reconstructed from these k digits (since the $k \times k$ submatrix of $\boldsymbol{G}$ is invertible).

Prange Algorithm (Information Set Decoding) 1962

Let $\mathscr{C}$ be an $[n, k]$ code described as the row space of a matrix $\boldsymbol{G}$. Suppose we received $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{e}$ of weight $\leqslant t$.

1. Take a k–tuple $i_1, \ldots, i_k$ of columns of $\boldsymbol{G}$ such that the corresponding $k \times k$ matrix is invertible and suppose that none of these positions have errors ($e_{i_j} = 0$ for any $j \in \{1, \ldots, k\}$).
2. We supposed that $y_{i_j} = c_{i_j}$ for any $j \in \{1, \ldots, k\}$. Then, $\boldsymbol{c}$ can be reconstructed from these k digits (since the $k \times k$ submatrix of $\boldsymbol{G}$ is invertible).
3. This provides a word $\tilde{\boldsymbol{c}}$:

Prange Algorithm (Information Set Decoding) 1962

Let $\mathscr{C}$ be an $[n, k]$ code described as the row space of a matrix $\boldsymbol{G}$. Suppose we received $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{e}$ of weight $\leqslant t$.

1. Take a k–tuple $i_1, \ldots, i_k$ of columns of $\boldsymbol{G}$ such that the corresponding $k \times k$ matrix is invertible and suppose that none of these positions have errors ($e_{i_j} = 0$ for any $j \in \{1, \ldots, k\}$).
2. We supposed that $y_{i_j} = c_{i_j}$ for any $j \in \{1, \ldots, k\}$. Then, $\boldsymbol{c}$ can be reconstructed from these k digits (since the $k \times k$ submatrix of $\boldsymbol{G}$ is invertible).
3. This provides a word $\tilde{\boldsymbol{c}}$:
   - if $d(\tilde{\boldsymbol{c}}, \boldsymbol{y}) \leqslant t$, then return $\tilde{\boldsymbol{c}}$;

Prange Algorithm (Information Set Decoding) 1962

Let $\mathscr{C}$ be an $[n, k]$ code described as the row space of a matrix $\boldsymbol{G}$. Suppose we received $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{e}$ of weight $\leqslant t$.

1. Take a k–tuple $i_1, \ldots, i_k$ of columns of $\boldsymbol{G}$ such that the corresponding $k \times k$ matrix is invertible and suppose that none of these positions have errors ($e_{i_j} = 0$ for any $j \in \{1, \ldots, k\}$).
2. We supposed that $y_{i_j} = c_{i_j}$ for any $j \in \{1, \ldots, k\}$. Then, $\boldsymbol{c}$ can be reconstructed from these k digits (since the $k \times k$ submatrix of $\boldsymbol{G}$ is invertible).
3. This provides a word $\tilde{\boldsymbol{c}}$:
    - if $d(\tilde{\boldsymbol{c}}, \boldsymbol{y}) \leqslant t$, then return $\tilde{\boldsymbol{c}}$;
    - else, go to (1).

Complexity

Depends on the probability of finding k positions avoiding the error positions.

$$\mathbb{P} = \frac{\binom{n-t}{k}}{\binom{n}{k}}.$$

Complexity

Depends on the probability of finding k positions avoiding the error positions.

$$\mathbb{P} = \frac{\binom{n-t}{k}}{\binom{n}{k}}.$$

This leads to an average complexity

$$O\left(\frac{\binom{n}{k}}{\binom{n-t}{k}} n^{\omega}\right).$$

Complexity

Depends on the probability of finding k positions avoiding the error positions.

$$\mathbb{P} = \frac{\binom{n-t}{k}}{\binom{n}{k}}.$$

This leads to an average complexity

$$O\left(\frac{\binom{n}{k}}{\binom{n-t}{k}} n^{\omega}\right).$$

Remark

- If $t = \alpha n$ for some $\alpha > 0$ and $k = Rn$ for some $R > 0$, then the average complexity is in $2^{\Omega(n)}$.
- If $t = O(1)$, the the average complexity is polynomial: in $O(n^{t+\omega})$.

Improvements

- Lee Brickel 1988;
- Stern 1989 & Dumer 1991;
- Canteaut Chabaud 1998;
- May, Meurer, Thomae 2011;
- **Becker, Joux, May, Meurer, 2012**;
- May, Ozerov, 2015.

Improvements

- Lee Brickel 1988;
- Stern 1989 & Dumer 1991;
- Canteaut Chabaud 1998;
- May, Meurer, Thomae 2011;
- **Becker, Joux, May, Meurer, 2012**;
- May, Ozerov, 2015.

Example

For a binary code of length $0.5n$, correcting an amount of 10% of errors costs $O(2^{0.09n})$ using BJMM.

Improvements

- Lee Brickel 1988;
- Stern 1989 & Dumer 1991;
- Canteaut Chabaud 1998;
- May, Meurer, Thomae 2011;
- **Becker, Joux, May, Meurer, 2012**;
- May, Ozerov, 2015.

Example

For a binary code of length $0.5n$, correcting an amount of 10% of errors costs $O(2^{0.09n})$ using BJMM.

Remark

Most of the improvements concern binary codes.

Key recovery attacks : How to propose a family of codes?

The family should

- contain “large” enough codes to resist to message recovery attacks;
- be large enough (to avoid brute force search);
- The structure of the code should be easily hidden (which is the hardest task).

LDPC codes

Problem.

- Given a code $\mathscr{C}$ described as the row space of a matrix $\boldsymbol{G}$, find a sparse matrix $\boldsymbol{H}$ such that $\boldsymbol{G}\boldsymbol{H}^T = 0$.
- Equivalently, find a collection of low weight words $(\boldsymbol{h}_i)$ such that $\boldsymbol{G}\boldsymbol{h}_i^T = 0$.

LDPC codes

Problem.

- Given a code $\mathscr{C}$ described as the row space of a matrix $\boldsymbol{G}$, find a sparse matrix $\boldsymbol{H}$ such that $\boldsymbol{G}\boldsymbol{H}^T = 0$.
- Equivalently, find a collection of low weight words $(\boldsymbol{h}_i)$ such that $\boldsymbol{G}\boldsymbol{h}_i^T = 0$.
- This can be done by generic decoding : finding a low weight codeword is nothing but decoding the zero codeword.

LDPC codes

Problem.

- Given a code $\mathscr{C}$ described as the row space of a matrix $\boldsymbol{G}$, find a sparse matrix $\boldsymbol{H}$ such that $\boldsymbol{G}\boldsymbol{H}^T = 0$.
- Equivalently, find a collection of low weight words $(\boldsymbol{h}_i)$ such that $\boldsymbol{G}\boldsymbol{h}_i^T = 0$.
- This can be done by generic decoding : finding a low weight codeword is nothing but decoding the zero codeword.
- For LDPC codes, we know that $\boldsymbol{H}$ has row weight bounded by t, hence these row vectors can be recovered in polynomial time $O(n^{t+\omega})$.

LDPC/MDPC codes

Summary. Using generic decoding, the secret key can be recovered

- in polynomial time for LDPC codes;
- in $O(2^{\sqrt{n}})$ for MDPC codes.

LDPC/MDPC codes

Summary. Using generic decoding, the secret key can be recovered

- in polynomial time for LDPC codes;
- in $O(2^{\sqrt{n}})$ for MDPC codes.

Remark

For MDPC codes the rows of $\boldsymbol{H}$ such as the errors have weight $O(\sqrt{n})$. Therefore, both key recovery and message recovery attacks have complexity $O(2^{\sqrt{n}})$.

GRS codes – A distinguisher

Definition

Let $\mathscr{C}, \mathscr{D} \subseteq \mathbb{F}_q^n$ be two codes.

$$\mathscr{C} \star \mathscr{D} \stackrel{\text{def}}{=} \mathrm{Span}\{(c_1 d_1, \ldots, c_n d_n) \mid \boldsymbol{c} \in \mathscr{C},\ \boldsymbol{d} \in \mathscr{D}\}$$

GRS codes – A distinguisher

Theorem (Cascudo, Cramer, Mirandola, Zémor. 2014)

Let $\mathscr{A} \subseteq \mathbb{F}_q^n$ *be an* $[n, k]$ *random code with* $n > \binom{k+1}{2}$*. Then, for any* $0 < \ell < \binom{k+1}{2}$,

$$\mathbb{P}\left(\dim(\mathscr{A} \star \mathscr{A}) \leqslant \binom{k+1}{2} - \ell\right) = O(q^{-\ell} \cdot q^{-n-\binom{k+1}{2}})$$

Informally, $\dim \mathscr{A} \star \mathscr{A} = \binom{k+1}{2}$ *w.h.p.*

Proposition

Let $\mathscr{C}$ *be an* $[n, k]$ *GRS code with* $k < n/2$*, then*

$$\dim \mathscr{C} \star \mathscr{C} = 2 \dim \mathscr{C} + 1.$$

GRS codes – A distinguisher

Proposition

Let $\mathscr{C}$ be an $[n, k]$ GRS code with $k < n/2$, then

$$\dim \mathscr{C} \star \mathscr{C} = 2 \dim \mathscr{C} + 1.$$

Consequences. Attacks on schemes based on variants of GRS codes which resisted to a previous attack (that of Sidelnikov Shestakov 1992).

GRS codes – A distinguisher

Proposition

Let $\mathscr{C}$ be an $[n, k]$ GRS code with $k < n/2$, then

$$\dim \mathscr{C} \star \mathscr{C} = 2 \dim \mathscr{C} + 1.$$

Consequences. Attacks on schemes based on variants of GRS codes which resisted to a previous attack (that of Sidelnikov Shestakov 1992).

- Wieschebrink 2006 (Broke Berger Loidreau proposal).
- C., Gaborit, Gauthier–Umaña, Otmani, Tillich 2013. (Variants of Wieschebrink and Baldi et al.).
- C., Otmani, Tillich 2014. Goppa codes with extension degree 2.

Alternant codes – brute force attacks

An alternant code is a $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \cap \mathbb{F}_q^n$ where the GRS code is defined over some extension $\mathbb{F}_{q^m}$. The secret key is the pair $(\boldsymbol{x}, \boldsymbol{y})$ (its knowledge permits to correct errors).

- Brute force attack is in $O(q^{2nm})$.
- Actually, Sendrier's *Support Splitting algorithm* permits to determine the permutation relating two codes if exists. Hence it is sufficient to find the pair $(\boldsymbol{x}, \boldsymbol{y})$ up to permutation : which divides the cost by $n!$
- This leads to a cost

$$O(\frac{q^{2nm}}{n!})$$

  Actually $n \leqslant q^m$ and hence $\frac{q^{2nm}}{n!} > \frac{n^{2n}}{n!} \gg n^n$.

Alternant codes – Algebraic attacks

An alternant code can be defined as the kernel (whose entries are in some subfield) of a matrix of the form

$$\begin{pmatrix} y_1 & y_2 & \cdots & y_n \\ x_1 y_1 & x_2 y_2 & \cdots & x_n y_n \\ \vdots & \vdots & & \vdots \\ x_1^r y_1 & x_2^r y_2 & \cdots & x_n^r y_n \end{pmatrix}.$$

The secret key is the pair of vectors $(\boldsymbol{x}, \boldsymbol{y})$ and the public key a basis of this kernel.

Consequence. The x_i's and the y_i's are solutions of a polynomial system.

Alternant codes – Algebraic attacks

- Attacks on Goppa/alternant codes with a non trivial permutation group.
  - Faugère, Otmani, Perret, Tillich 2010.
  - Faugère, Otmani, Perret, de Portzamparc, Tillich 2016
- Attack on Goppa codes over non prime fields (with small degree)
  - Faugère, Perret, de Portzamparc, 2016.

Alternant codes – Algebraic attacks

Open questions :

- How to evaluate the complexity of this polynomial system resolution?
- How to get lower bounds for the complexity?

Alternant codes – Algebraic attacks

Open questions :

- How to evaluate the complexity of this polynomial system resolution?
- How to get lower bounds for the complexity?

Remark

Note that the polynomial system is overdetermined in general. Some relevant selection of a subset of equation may improve significantly the speed up of resolution! (See Faugère, Perret, de Portzamparc 2016)

Conclusion

- Code–based crypto is 40 years old.

Conclusion

- Code–based crypto is 40 years old.
- After being considered as unusable, it is coming back because of
  - a need of post quantum alternatives to number theoretic based primitives;
  - impressive improvements on the key sizes

Conclusion

- Code–based crypto is 40 years old.
- After being considered as unusable, it is coming back because of
  - a need of post quantum alternatives to number theoretic based primitives;
  - impressive improvements on the key sizes
- Promizing primitives
  - MDPC codes;
  - Binary Goppa/Alternant codes : 40 years and no polynomial time key recovery attack.

Merci!