

Code-Based Cryptography

Tanja Lange

with some slides by Tung Chou and Christiane Peters

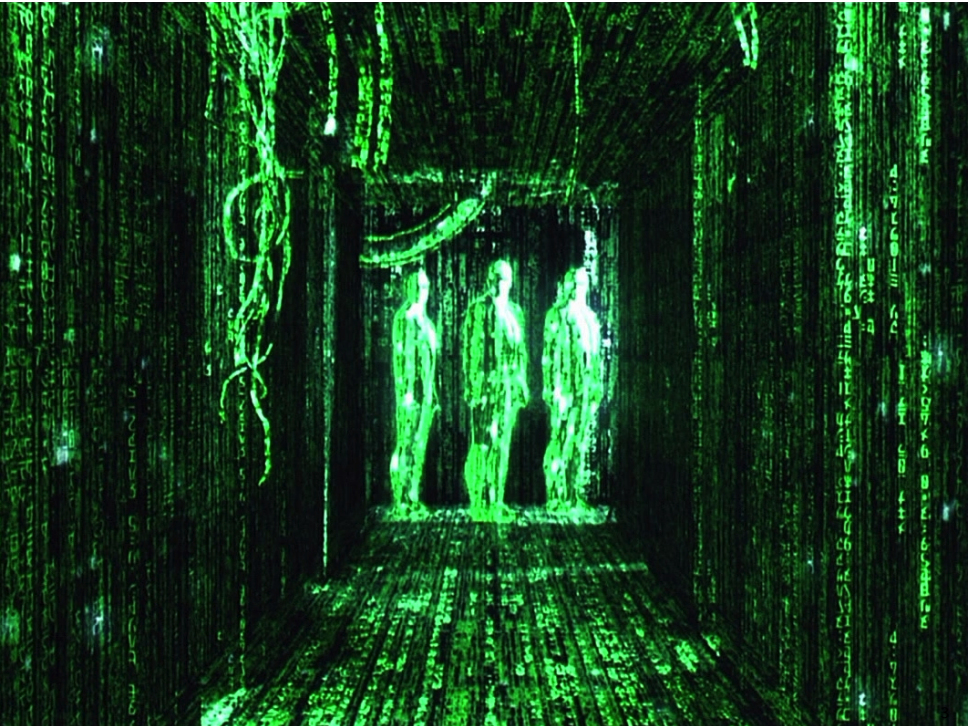
Technische Universiteit Eindhoven

Executive School on Post-Quantum Cryptography

02 July 2019

Error correction

- ▶ Digital media is exposed to memory corruption.
- ▶ Many systems check whether data was corrupted in transit:
 - ▶ ISBN numbers have check digit to detect corruption.
 - ▶ ECC RAM detects up to two errors and can correct one error. 64 bits are stored as 72 bits: extra 8 bits for checks and recovery.
- ▶ In general, k bits of data get stored in n bits, adding some redundancy.
- ▶ If no error occurred, these n bits satisfy $n - k$ parity check equations; else can correct errors from the error pattern.
- ▶ Good codes can correct many errors without blowing up storage too much; offer guarantee to correct t errors (often can correct or at least detect more).
- ▶ To represent these check equations we need a matrix.



Hamming code

Parity check matrix ($n = 7, k = 4$):

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

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{rcccccccl} b_0 & +b_1 & & +b_3 & +b_4 & & & = & 0 \\ b_0 & & +b_2 & +b_3 & & +b_5 & & = & 0 \\ & b_1 & +b_2 & +b_3 & & & +b_6 & = & 0 \end{array}$$

If one error occurred, at least one of these equations will not hold. Failure pattern uniquely identifies the error location, e.g., 1, 0, 1 means

Hamming code

Parity check matrix ($n = 7, k = 4$):

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

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{rcccccc} b_0 & +b_1 & & +b_3 & +b_4 & & = & 0 \\ b_0 & & +b_2 & +b_3 & & +b_5 & = & 0 \\ & b_1 & +b_2 & +b_3 & & & +b_6 & = & 0 \end{array}$$

If one error occurred, at least one of these equations will not hold. Failure pattern uniquely identifies the error location, e.g., 1,0,1 means b_1 flipped.

Hamming code

Parity check matrix ($n = 7, k = 4$):

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

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{rcccccc} b_0 & +b_1 & & +b_3 & +b_4 & & = & 0 \\ b_0 & & +b_2 & +b_3 & & +b_5 & = & 0 \\ & b_1 & +b_2 & +b_3 & & & +b_6 & = & 0 \end{array}$$

If one error occurred, at least one of these equations will not hold. Failure pattern uniquely identifies the error location, e.g., 1, 0, 1 means b_1 flipped.

In math notation, the failure pattern is $H \cdot \mathbf{b}$.

Coding theory

- ▶ Names: code word \mathbf{c} , error vector \mathbf{e} , received word $\mathbf{b} = \mathbf{c} + \mathbf{e}$.
- ▶ Very common to transform the matrix so that the right part has just 1 on the diagonal (no need to store that).

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

- ▶ Many special constructions discovered in 65 years of coding theory:
 - ▶ Large matrix H .
 - ▶ Fast decoding algorithm to find \mathbf{e} given $\mathbf{s} = H \cdot (\mathbf{c} + \mathbf{e})$, whenever \mathbf{e} does not have too many bits set.
- ▶ Given large H , usually very hard to find fast decoding algorithm.
- ▶ Use this difference in complexities for encryption.

Code-based encryption

- ▶ 1971 Goppa: Fast decoders for many matrices H .
- ▶ 1978 McEliece: Use Goppa codes for public-key crypto.
 - ▶ Original parameters designed for 2^{64} security.
 - ▶ 2008 Bernstein–Lange–Peters: broken in $\approx 2^{60}$ cycles.
 - ▶ Easily scale up for higher security.
- ▶ 1986 Niederreiter: Simplified and smaller version of McEliece.
- ▶ 1962 Prange: simple attack idea guiding sizes in 1978 McEliece.

The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$ -bit keys as $\lambda \rightarrow \infty$ to achieve 2^λ security against Prange's attack. Here $c_0 \approx 0.7418860694$.

Security analysis

Some papers studying algorithms for attackers:

1962 Prange; 1981 Clark–Cain, crediting Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk; 1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg; 1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau; 1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne; 1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier; 2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilburg; 2009 Bernstein (**post-quantum**); 2009 Finiasz–Sendrier; 2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae; 2012 Becker–Joux–May–Meurer; 2013 Hamdaoui–Sendrier; 2015 May–Ozerov; 2016 Canto Torres–Sendrier; 2017 Kachigar–Tillich (**post-quantum**); 2017 Both–May; 2018 Both–May; 2018 Kirshanova (**post-quantum**).

Consequence of security analysis

- ▶ The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$ -bit keys as $\lambda \rightarrow \infty$ to achieve 2^λ security against all these attacks.

Consequence of security analysis

- ▶ The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$ -bit keys as $\lambda \rightarrow \infty$ to achieve 2^λ security against all these attacks. Here $c_0 \approx 0.7418860694$.
- ▶ 256 KB public key for 2^{146} pre-quantum security.
- ▶ 512 KB public key for 2^{187} pre-quantum security.
- ▶ 1024 KB public key for 2^{263} pre-quantum security.

Consequence of security analysis

- ▶ The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$ -bit keys as $\lambda \rightarrow \infty$ to achieve 2^λ security against all these attacks. Here $c_0 \approx 0.7418860694$.
- ▶ 256 KB public key for 2^{146} pre-quantum security.
- ▶ 512 KB public key for 2^{187} pre-quantum security.
- ▶ 1024 KB public key for 2^{263} pre-quantum security.
- ▶ Post-quantum (Grover): below 2^{263} , above 2^{131} .

Linear codes

A **binary linear code** C of length n and dimension k is a k -dimensional subspace of \mathbb{F}_2^n .

C is usually specified as

- ▶ the row space of a **generating matrix** $G \in \mathbb{F}_2^{k \times n}$

$$C = \{\mathbf{m}G \mid \mathbf{m} \in \mathbb{F}_2^k\}$$

- ▶ the kernel space of a **parity-check matrix** $H \in \mathbb{F}_2^{(n-k) \times n}$

$$C = \{\mathbf{c} \mid H\mathbf{c}^T = 0, \mathbf{c} \in \mathbb{F}_2^n\}$$

Leaving out the T from now on.

Example

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$\mathbf{c} = (111)G = (10011)$ is a codeword.

Example

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$\mathbf{c} = (111)G = (10011)$ is a codeword.

Linear codes are linear:

The sum of two codewords is a codeword:

Example

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$\mathbf{c} = (111)G = (10011)$ is a codeword.

Linear codes are linear:

The sum of two codewords is a codeword:

$$\mathbf{c}_1 + \mathbf{c}_2 = \mathbf{m}_1 G + \mathbf{m}_2 G = (\mathbf{m}_1 + \mathbf{m}_2)G.$$

Same with parity-check matrix:

Example

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$\mathbf{c} = (111)G = (10011)$ is a codeword.

Linear codes are linear:

The sum of two codewords is a codeword:

$$\mathbf{c}_1 + \mathbf{c}_2 = \mathbf{m}_1 G + \mathbf{m}_2 G = (\mathbf{m}_1 + \mathbf{m}_2)G.$$

Same with parity-check matrix:

$$H(\mathbf{c}_1 + \mathbf{c}_2) = H\mathbf{c}_1 + H\mathbf{c}_2 = 0 + 0 = 0.$$

Hamming weight and distance

- ▶ The **Hamming weight** of a word is the number of nonzero coordinates.

$$\text{wt}(1, 0, 0, 1, 1) = 3$$

- ▶ The **Hamming distance** between two words in \mathbb{F}_2^n is the number of coordinates in which they differ.

$$d((1, 1, 0, 1, 1), (1, 0, 0, 1, 0)) =$$

Hamming weight and distance

- ▶ The **Hamming weight** of a word is the number of nonzero coordinates.

$$\text{wt}(1, 0, 0, 1, 1) = 3$$

- ▶ The **Hamming distance** between two words in \mathbb{F}_2^n is the number of coordinates in which they differ.

$$d((1, 1, 0, 1, 1), (1, 0, 0, 1, 0)) = 2$$

Hamming weight and distance

- ▶ The **Hamming weight** of a word is the number of nonzero coordinates.

$$\text{wt}(1, 0, 0, 1, 1) = 3$$

- ▶ The **Hamming distance** between two words in \mathbb{F}_2^n is the number of coordinates in which they differ.

$$d((1, 1, 0, 1, 1), (1, 0, 0, 1, 0)) = 2$$

The Hamming distance between \mathbf{x} and \mathbf{y} equals the Hamming weight of $\mathbf{x} + \mathbf{y}$:

$$d((1, 1, 0, 1, 1), (1, 0, 0, 1, 1)) = \text{wt}(0, 1, 0, 0, 0).$$

Minimum distance

- ▶ The **minimum distance** of a linear code C is the smallest Hamming weight of a nonzero codeword in C .

$$d = \min_{\mathbf{0} \neq \mathbf{c} \in C} \{\text{wt}(\mathbf{c})\} = \min_{\mathbf{b} \neq \mathbf{c} \in C} \{d(\mathbf{b}, \mathbf{c})\}$$

- ▶ In code with minimum distance $d = 2t + 1$, any vector $\mathbf{x} = \mathbf{c} + \mathbf{e}$ with $\text{wt}(\mathbf{e}) \leq t$ is uniquely decodable to \mathbf{c} ;
i. e. there is no closer code word.

Decoding problem

Decoding problem: find the closest codeword $\mathbf{c} \in C$ to a given $\mathbf{x} \in \mathbb{F}_2^n$, assuming that there is a unique closest codeword. Let $\mathbf{x} = \mathbf{c} + \mathbf{e}$. Note that finding \mathbf{e} is an equivalent problem.

- ▶ If \mathbf{c} is t errors away from \mathbf{x} , i.e., the Hamming weight of \mathbf{e} is t , this is called a t -error correcting problem.
- ▶ There are lots of code families with fast decoding algorithms, e.g., Reed–Solomon codes, Goppa codes/alternant codes, etc.
- ▶ However, the **general decoding problem** is hard:
Information-set decoding (see later) takes exponential time.

The McEliece cryptosystem I

- ▶ Let C be a length- n binary Goppa code Γ of dimension k with minimum distance $2t + 1$ where $t \approx (n - k)/\log_2(n)$; original parameters (1978) $n = 1024$, $k = 524$, $t = 50$.
- ▶ The **McEliece secret key** consists of a generator matrix G for Γ , an efficient t -error correcting decoding algorithm for Γ ; an $n \times n$ permutation matrix P and a nonsingular $k \times k$ matrix S .
- ▶ n, k, t are public; but Γ, P, S are randomly generated secrets.
- ▶ The **McEliece public key** is the $k \times n$ matrix $G' = SGP$.

The McEliece cryptosystem II

- ▶ Encrypt: Compute $\mathbf{m}G'$ and add a random error vector \mathbf{e} of weight t and length n . Send $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$.
- ▶ Decrypt: Compute $\mathbf{y}P^{-1} = \mathbf{m}G'P^{-1} + \mathbf{e}P^{-1} = (\mathbf{m}S)G + \mathbf{e}P^{-1}$.
This works because $\mathbf{e}P^{-1}$ has the same weight as \mathbf{e}

The McEliece cryptosystem II

- ▶ Encrypt: Compute $\mathbf{m}G'$ and add a random error vector \mathbf{e} of weight t and length n . Send $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$.
- ▶ Decrypt: Compute $\mathbf{y}P^{-1} = \mathbf{m}G'P^{-1} + \mathbf{e}P^{-1} = (\mathbf{m}S)G + \mathbf{e}P^{-1}$. This works because $\mathbf{e}P^{-1}$ has the same weight as \mathbf{e} because P is a permutation matrix. Use fast decoding to find $\mathbf{m}S$ and \mathbf{m} .
- ▶ Attacker is faced with decoding \mathbf{y} to nearest codeword $\mathbf{m}G'$ in the code generated by G' . This is general decoding if G' does not expose any structure.

Systematic form

- ▶ A **systematic generator matrix** is a generator matrix of the form $(I_k|Q)$ where I_k is the $k \times k$ identity matrix and Q is a $k \times (n - k)$ matrix (**redundant part**).
- ▶ Classical decoding is about recovering m from $c = mG$; without errors m equals the first k positions of c .

Systematic form

- ▶ A **systematic generator matrix** is a generator matrix of the form $(I_k|Q)$ where I_k is the $k \times k$ identity matrix and Q is a $k \times (n - k)$ matrix (**redundant part**).
- ▶ Classical decoding is about recovering m from $c = mG$; without errors m equals the first k positions of c .
- ▶ Easy to get parity-check matrix from systematic generator matrix, use $H = (Q^T|I_{n-k})$.

Systematic form

- ▶ A **systematic generator matrix** is a generator matrix of the form $(I_k|Q)$ where I_k is the $k \times k$ identity matrix and Q is a $k \times (n - k)$ matrix (**redundant part**).
- ▶ Classical decoding is about recovering m from $c = mG$; without errors m equals the first k positions of c .
- ▶ Easy to get parity-check matrix from systematic generator matrix, use $H = (Q^T|I_{n-k})$.
Then

$$H(mG)^T = HG^T m^T = (Q^T|I_{n-k})(I_k|Q)^T m^T = 0.$$

Different views on decoding

- ▶ The **syndrome** of $\mathbf{x} \in \mathbb{F}_2^n$ is $\mathbf{s} = H\mathbf{x}$.
Note $H\mathbf{x} = H(\mathbf{c} + \mathbf{e}) = H\mathbf{c} + H\mathbf{e} = H\mathbf{e}$ depends only on \mathbf{e} .
- ▶ The **syndrome decoding problem** is to compute $\mathbf{e} \in \mathbb{F}_2^n$ given $\mathbf{s} \in \mathbb{F}_2^{n-k}$ so that $H\mathbf{e} = \mathbf{s}$ and \mathbf{e} has minimal weight.
- ▶ Syndrome decoding and (regular) decoding are equivalent:

Different views on decoding

- ▶ The **syndrome** of $\mathbf{x} \in \mathbb{F}_2^n$ is $\mathbf{s} = H\mathbf{x}$.
Note $H\mathbf{x} = H(\mathbf{c} + \mathbf{e}) = H\mathbf{c} + H\mathbf{e} = H\mathbf{e}$ depends only on \mathbf{e} .
- ▶ The **syndrome decoding problem** is to compute $\mathbf{e} \in \mathbb{F}_2^n$ given $\mathbf{s} \in \mathbb{F}_2^{n-k}$ so that $H\mathbf{e} = \mathbf{s}$ and \mathbf{e} has minimal weight.
- ▶ Syndrome decoding and (regular) decoding are equivalent:
To decode \mathbf{x} with syndrome decoder, compute \mathbf{e} from $H\mathbf{x}$, then $\mathbf{c} = \mathbf{x} + \mathbf{e}$.
To expand syndrome, assume $H = (Q^T | I_{n-k})$.

Different views on decoding

- ▶ The **syndrome** of $\mathbf{x} \in \mathbb{F}_2^n$ is $\mathbf{s} = H\mathbf{x}$.
Note $H\mathbf{x} = H(\mathbf{c} + \mathbf{e}) = H\mathbf{c} + H\mathbf{e} = H\mathbf{e}$ depends only on \mathbf{e} .
- ▶ The **syndrome decoding problem** is to compute $\mathbf{e} \in \mathbb{F}_2^n$ given $\mathbf{s} \in \mathbb{F}_2^{n-k}$ so that $H\mathbf{e} = \mathbf{s}$ and \mathbf{e} has minimal weight.
- ▶ Syndrome decoding and (regular) decoding are equivalent:
To decode \mathbf{x} with syndrome decoder, compute \mathbf{e} from $H\mathbf{x}$, then $\mathbf{c} = \mathbf{x} + \mathbf{e}$.
To expand syndrome, assume $H = (Q^T | I_{n-k})$.
Then $\mathbf{x} = (00 \dots 0) || \mathbf{s}$ satisfies $\mathbf{s} = H\mathbf{x}$.
- ▶ Note that this \mathbf{x} is not a solution to the syndrome decoding problem, unless it has very low weight.

The Niederreiter cryptosystem I

Developed in 1986 by Harald Niederreiter as a variant of the McEliece cryptosystem. This is the schoolbook version.

- ▶ Use $n \times n$ permutation matrix P and $(n - k) \times (n - k)$ invertible matrix S .
- ▶ Public Key: a scrambled parity-check matrix
 $K = SHP \in \mathbb{F}_2^{(n-k) \times n}$.
- ▶ Encryption: The plaintext \mathbf{e} is an n -bit vector of weight t . The ciphertext \mathbf{s} is the $(n - k)$ -bit vector

$$\mathbf{s} = K\mathbf{e}.$$

- ▶ Decryption: Find a n -bit vector \mathbf{e} with $\text{wt}(\mathbf{e}) = t$ such that $\mathbf{s} = K\mathbf{e}$.
- ▶ The passive attacker is facing a t -error correcting problem for the public key, which seems to be random.

The Niederreiter cryptosystem II

- ▶ Public Key: a scrambled parity-check matrix $K = SHP$.
- ▶ Encryption: The plaintext \mathbf{e} is an n -bit vector of weight t . The ciphertext \mathbf{s} is the $(n - k)$ -bit vector

$$\mathbf{s} = K\mathbf{e}.$$

- ▶ Decryption using secret key: Compute

$$\begin{aligned} S^{-1}\mathbf{s} &= S^{-1}K\mathbf{e} = S^{-1}(SHP)\mathbf{e} \\ &= H(P\mathbf{e}) \end{aligned}$$

and observe that $\text{wt}(P\mathbf{e}) = t$, because P permutes. Use efficient syndrome decoder for H to find $\mathbf{e}' = P\mathbf{e}$ and thus $\mathbf{e} = P^{-1}\mathbf{e}'$.

Note on codes

- ▶ McEliece proposed to use binary Goppa codes. These are still used today.
- ▶ Niederreiter described his scheme using Reed-Solomon codes. These were broken in 1992 by Sidelnikov and Chestakov.
- ▶ More corpses on the way: concatenated codes, Reed-Muller codes, several Algebraic Geometry (AG) codes, Gabidulin codes, several LDPC codes, cyclic codes.
- ▶ Some other constructions look OK (for now). NIST competition has several entries on QCMDPC codes.

Binary Goppa code

Let $q = 2^m$. A binary Goppa code is often defined by

- ▶ a list $L = (a_1, \dots, a_n)$ of n distinct elements in \mathbb{F}_q , called the **support**.
- ▶ a square-free polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t such that $g(a) \neq 0$ for all $a \in L$. $g(x)$ is called the **Goppa polynomial**.
- ▶ E.g. choose $g(x)$ irreducible over \mathbb{F}_q .

The corresponding binary Goppa code $\Gamma(L, g)$ is

$$\left\{ \mathbf{c} \in \mathbb{F}_2^n \mid S(\mathbf{c}) = \frac{c_1}{x - a_1} + \frac{c_2}{x - a_2} + \dots + \frac{c_n}{x - a_n} \equiv 0 \pmod{g(x)} \right\}$$

- ▶ This code is linear $S(\mathbf{b} + \mathbf{c}) = S(\mathbf{b}) + S(\mathbf{c})$ and has length n .
- ▶ Bounds on dimension $k \geq n - mt$ and minimum distance $t \geq 2t + 1$.

Reminder: How to hide nice code?

- ▶ Do not reveal matrix H related to nice-to-decode code.
- ▶ Pick a random invertible $(n - k) \times (n - k)$ matrix S and random $n \times n$ permutation matrix P . Put

$$K = SHP.$$

- ▶ K is the public key and S and P together with a decoding algorithm for H form the private key.
- ▶ For suitable codes K looks like random matrix.
- ▶ How to decode syndrome $\mathbf{s} = K\mathbf{e}$?

Reminder: How to hide nice code?

- ▶ Do not reveal matrix H related to nice-to-decode code.
- ▶ Pick a random invertible $(n - k) \times (n - k)$ matrix S and random $n \times n$ permutation matrix P . Put

$$K = SHP.$$

- ▶ K is the public key and S and P together with a decoding algorithm for H form the private key.
- ▶ For suitable codes K looks like random matrix.
- ▶ How to decode syndrome $\mathbf{s} = K\mathbf{e}$?
- ▶ Computes $S^{-1}\mathbf{s} = S^{-1}(SHP)\mathbf{e} = H(P\mathbf{e})$.
- ▶ P permutes, thus $P\mathbf{e}$ has same weight as \mathbf{e} .
- ▶ Decode to recover $P\mathbf{e}$, then multiply by P^{-1} .

How to hide nice code?

- ▶ For Goppa code use secret polynomial $g(x)$.
- ▶ Use secret permutation of the a_i , this corresponds to secret permutation of the n positions; this replaces P .
- ▶ Use systematic form $K = (K'|I)$ for key;
 - ▶ This implicitly applies S .
 - ▶ No need to remember S because decoding does not use H .
 - ▶ Public key size decreased to $(n - k) \times k$.
- ▶ Secret key is polynomial g and support $L = (a_1, \dots, a_n)$.

McBits (Bernstein, Chou, Schwabe, CHES 2013)

- ▶ Encryption is super fast anyways (just a vector-matrix multiplication).
- ▶ Main step in decryption is decoding of Goppa code. The McBits software achieves this in **constant time**.
- ▶ Decoding speed at 2^{128} pre-quantum security:
 $(n; t) = (4096; 41)$ uses 60493 Ivy Bridge cycles.
- ▶ Decoding speed at 2^{263} pre-quantum security:
 $(n; t) = (6960; 119)$ uses 306102 Ivy Bridge cycles.
- ▶ Grover speedup is less than halving the security level, so the latter parameters offer at least 2^{128} post-quantum security.
- ▶ More at <https://binary.cr.yp.to/mcbits.html>.

NIST submission Classic McEliece

- ▶ Security asymptotics unchanged by 40 years of cryptanalysis.
- ▶ Efficient and straightforward conversion OW-CPA PKE \rightarrow IND-CCA2 KEM.
- ▶ Open-source (public domain) implementations.
 - ▶ Constant-time software implementations.
 - ▶ FPGA implementation of full cryptosystem.
- ▶ No patents.

Metric	mceliece6960119	mceliece8192128
Public-key size	1047319 bytes	1357824 bytes
Secret-key size	13908 bytes	14080 bytes
Ciphertext size	226 bytes	240 bytes
Key-generation time	1108833108 cycles	1173074192 cycles
Encapsulation time	153940 cycles	188520 cycles
Decapsulation time	318088 cycles	343756 cycles

See <https://classic.mceliece.org> for more details.

More parameters in round 2.

Key issues for McEliece

- ▶ Very conservative system, expected to last; has strongest security track record.
- ▶ Ciphertexts are among the shortest.
- ▶ Secret keys can be compressed.
- ▶ But public keys are really, really big!
- ▶ Sending 1MB takes time and bandwidth.

Key issues for McEliece

- ▶ Very conservative system, expected to last; has strongest security track record.
 - ▶ Ciphertexts are among the shortest.
 - ▶ Secret keys can be compressed.
 - ▶ But public keys are really, really big!
 - ▶ Sending 1MB takes time and bandwidth.
 - ▶ **Google–Cloudflare experiment:**
 - in some cases the public-key + ciphertext size was too large to be viable in the context of TLS*
- and even 10KB messages dropped.

Key issues for McEliece

- ▶ Very conservative system, expected to last; has strongest security track record.
- ▶ Ciphertexts are among the shortest.
- ▶ Secret keys can be compressed.
- ▶ But public keys are really, really big!
- ▶ Sending 1MB takes time and bandwidth.
- ▶ [Google–Cloudflare experiment](#):
 - in some cases the public-key + ciphertext size was too large to be viable in the context of TLS*
 - and even 10KB messages dropped.
- ▶ If server accepts 1MB of public key from any client, an attacker can easily flood memory. This invites DoS attacks.

Goodness, what big keys you have!

- ▶ Public keys look like this:

$$K = \begin{pmatrix} 1 & 0 & \dots & 0 & 1 & \dots & 1 & 0 & 1 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 & \dots & 1 & 1 & 0 \\ 0 & 0 & \dots & 1 & 0 & \dots & 1 & 1 & 1 \end{pmatrix}$$

Left part is $(n - k) \times (n - k)$ identity matrix (no need to send)

right part is random-looking $(n - k) \times k$ matrix.

E.g. $n = 6960$, $k = 5413$, so $n - k = 1547$.

Goodness, what big keys you have!

- Public keys look like this:

$$K = \begin{pmatrix} 1 & 0 & \dots & 0 & 1 & \dots & 1 & 0 & 1 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 & \dots & 1 & 1 & 0 \\ 0 & 0 & \dots & 1 & 0 & \dots & 1 & 1 & 1 \end{pmatrix}$$

Left part is $(n - k) \times (n - k)$ identity matrix (no need to send)
right part is random-looking $(n - k) \times k$ matrix.

E.g. $n = 6960$, $k = 5413$, so $n - k = 1547$.

- Encryption xors secretly selected columns, e.g.

$$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Can servers avoid storing big keys?

$$K = \begin{pmatrix} 1 & 0 & \dots & 0 & 1 & \dots & 1 & 0 & 1 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 & \dots & 1 & 1 & 0 \\ 0 & 0 & \dots & 1 & 0 & \dots & 1 & 1 & 1 \end{pmatrix} = (I_{n-k} | K')$$

- ▶ Encryption xors secretly selected columns.
- ▶ With some storage and trusted environment:
Receive columns of K' one at a time, store and update partial sum.

Can servers avoid storing big keys?

$$K = \begin{pmatrix} 1 & 0 & \dots & 0 & 1 & \dots & 1 & 0 & 1 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 & \dots & 1 & 1 & 0 \\ 0 & 0 & \dots & 1 & 0 & \dots & 1 & 1 & 1 \end{pmatrix} = (I_{n-k} | K')$$

- ▶ Encryption xors secretly selected columns.
- ▶ With some storage and trusted environment:
Receive columns of K' one at a time, store and update partial sum.
- ▶ On the real Internet, without per-client state:

Can servers avoid storing big keys?

$$K = \begin{pmatrix} 1 & 0 & \dots & 0 & 1 & \dots & 1 & 0 & 1 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 & \dots & 1 & 1 & 0 \\ 0 & 0 & \dots & 1 & 0 & \dots & 1 & 1 & 1 \end{pmatrix} = (I_{n-k} | K')$$

- ▶ Encryption xors secretly selected columns.
- ▶ With some storage and trusted environment:
Receive columns of K' one at a time, store and update partial sum.
- ▶ On the real Internet, without per-client state:
Don't reveal intermediate results!
Which columns are picked is the secret message!
Intermediate results show whether a column was used or not.

McTiny (Bernstein/Lange)

Partition key

$$K' = \begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}$$

- ▶ Each submatrix $K_{i,j}$ small enough to fit + cookie into network packet.
- ▶ Server does computation on $K_{i,j}$, puts partial result into cookie.
- ▶ Cookies are encrypted by server to itself using some temporary symmetric key (same key for all server connections).
No per-client memory allocation.
- ▶ Client feeds the $K_{i,j}$ to server & handles storage for the server.
- ▶ Cookies also encrypted & authenticated to client.
- ▶ More stuff to avoid replay & similar attacks.

McTiny (Bernstein/Lange)

Partition key

$$K' = \begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}$$

- ▶ Each submatrix $K_{i,j}$ small enough to fit + cookie into network packet.
- ▶ Server does computation on $K_{i,j}$, puts partial result into cookie.
- ▶ Cookies are encrypted by server to itself using some temporary symmetric key (same key for all server connections).
No per-client memory allocation.
- ▶ Client feeds the $K_{i,j}$ to server & handles storage for the server.
- ▶ Cookies also encrypted & authenticated to client.
- ▶ More stuff to avoid replay & similar attacks.
- ▶ Several round trips, but no per-client state on the server.

Do not use the schoolbook versions!

Sloppy Alice attacks! 1998 Verheul, Doumen, van Tilborg

- ▶ Assume that the decoding algorithm decodes up to t errors, i. e. it decodes $\mathbf{y} = \mathbf{c} + \mathbf{e}$ to \mathbf{c} if $\text{wt}(\mathbf{e}) \leq t$.
- ▶ Eve intercepts ciphertext $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$.
Eve poses as Alice towards Bob and sends him tweaks of \mathbf{y} . She uses Bob's reactions (success of failure to decrypt) to recover \mathbf{m} .
- ▶ Assume $\text{wt}(\mathbf{e}) = t$. (Else flip more bits till Bob fails).
- ▶ Eve sends $\mathbf{y}_i = \mathbf{y} + \mathbf{e}_i$ for \mathbf{e}_i the i -th unit vector.
If Bob returns error, position i in \mathbf{e} is 0 (so the number of errors has increased to $t + 1$ and Bob fails).
Else position i in \mathbf{e} is 1.
- ▶ After k steps Eve knows the first k positions of $\mathbf{m}G'$ without error. Invert the $k \times k$ submatrix of G' to get \mathbf{m}

Sloppy Alice attacks! 1998 Verheul, Doumen, van Tilborg

- ▶ Assume that the decoding algorithm decodes up to t errors, i. e. it decodes $\mathbf{y} = \mathbf{c} + \mathbf{e}$ to \mathbf{c} if $\text{wt}(\mathbf{e}) \leq t$.
- ▶ Eve intercepts ciphertext $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$.
Eve poses as Alice towards Bob and sends him tweaks of \mathbf{y} . She uses Bob's reactions (success of failure to decrypt) to recover \mathbf{m} .
- ▶ Assume $\text{wt}(\mathbf{e}) = t$. (Else flip more bits till Bob fails).
- ▶ Eve sends $\mathbf{y}_i = \mathbf{y} + \mathbf{e}_i$ for \mathbf{e}_i the i -th unit vector.
If Bob returns error, position i in \mathbf{e} is 0 (so the number of errors has increased to $t + 1$ and Bob fails).
Else position i in \mathbf{e} is 1.
- ▶ After k steps Eve knows the first k positions of $\mathbf{m}G'$ without error. Invert the $k \times k$ submatrix of G' to get \mathbf{m} assuming it is invertible.
- ▶ Proper attack: figure out invertible submatrix of G' at beginning; recover matching k coordinates.

More on sloppy Alice

- ▶ This attack has Eve send Bob variations of the same ciphertext; so Bob will think that Alice is sloppy.
- ▶ Note, this is more complicated if \mathbb{F}_q instead of \mathbb{F}_2 is used.
- ▶ Other name: reaction attack.
(1999 Hall, Goldberg, and Schneier)
- ▶ Attack also works on Niederreiter version:

More on sloppy Alice

- ▶ This attack has Eve send Bob variations of the same ciphertext; so Bob will think that Alice is sloppy.
- ▶ Note, this is more complicated if \mathbb{F}_q instead of \mathbb{F}_2 is used.
- ▶ Other name: reaction attack.
(1999 Hall, Goldberg, and Schneier)
- ▶ Attack also works on Niederreiter version:
Bitflip corresponds to sending $\mathbf{s}_i = \mathbf{s} + K_i$,
where K_i is the i -th column of K .
- ▶ More involved but doable (for McEliece and Niederreiter)
if decryption requires exactly t errors.

Berson's attack

- ▶ Eve knows $\mathbf{y}_1 = \mathbf{m}G' + \mathbf{e}_1$ and $\mathbf{y}_2 = \mathbf{m}G' + \mathbf{e}_2$; these have the same \mathbf{m} .

Berson's attack

- ▶ Eve knows $\mathbf{y}_1 = \mathbf{m}G' + \mathbf{e}_1$ and $\mathbf{y}_2 = \mathbf{m}G' + \mathbf{e}_2$; these have the same \mathbf{m} .
- ▶ Then $\mathbf{y}_1 + \mathbf{y}_2 = \mathbf{e}_1 + \mathbf{e}_2 = \bar{\mathbf{e}}$. This has weight in $[0, 2t]$.
- ▶ If $\text{wt}(\bar{\mathbf{e}}) = 2t$:

Berson's attack

- ▶ Eve knows $\mathbf{y}_1 = \mathbf{m}G' + \mathbf{e}_1$ and $\mathbf{y}_2 = \mathbf{m}G' + \mathbf{e}_2$; these have the same \mathbf{m} .
- ▶ Then $\mathbf{y}_1 + \mathbf{y}_2 = \mathbf{e}_1 + \mathbf{e}_2 = \bar{\mathbf{e}}$. This has weight in $[0, 2t]$.
- ▶ If $\text{wt}(\bar{\mathbf{e}}) = 2t$:
All zero positions in $\bar{\mathbf{e}}$ are error free in both ciphertexts.
Invert G' in those columns to recover \mathbf{m} as in previous attack.
- ▶ Else:

Berson's attack

- ▶ Eve knows $\mathbf{y}_1 = \mathbf{m}G' + \mathbf{e}_1$ and $\mathbf{y}_2 = \mathbf{m}G' + \mathbf{e}_2$; these have the same \mathbf{m} .
- ▶ Then $\mathbf{y}_1 + \mathbf{y}_2 = \mathbf{e}_1 + \mathbf{e}_2 = \bar{\mathbf{e}}$. This has weight in $[0, 2t]$.
- ▶ If $\text{wt}(\bar{\mathbf{e}}) = 2t$:
All zero positions in $\bar{\mathbf{e}}$ are error free in both ciphertexts.
Invert G' in those columns to recover \mathbf{m} as in previous attack.
- ▶ Else: ignore the $2w = \text{wt}(\bar{\mathbf{e}}) < 2t$ positions in G' and \mathbf{y}_1 .
Solve decoding problem for $k \times (n - 2w)$ generator matrix G'' and vector \mathbf{y}'_1 with $t - w$ errors; typically much easier.

Formal security notions

- ▶ McEliece/Niederreiter are One-Way Encryption (OWE) schemes.
- ▶ However, the schemes as presented are not CCA-II secure:
 - ▶ Given challenge $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$, Eve can ask for decryptions of anything but \mathbf{y} .

Formal security notions

- ▶ McEliece/Niederreiter are One-Way Encryption (OWE) schemes.
- ▶ However, the schemes as presented are not CCA-II secure:
 - ▶ Given challenge $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$, Eve can ask for decryptions of anything but \mathbf{y} .
 - ▶ Eve picks a random code word $\mathbf{c} = \bar{\mathbf{m}}G'$, asks for decryption of $\mathbf{y} + \mathbf{c}$.
 - ▶ This is different from challenge \mathbf{y} , so Bob answers.

Formal security notions

- ▶ McEliece/Niederreiter are One-Way Encryption (OWE) schemes.
- ▶ However, the schemes as presented are not CCA-II secure:
 - ▶ Given challenge $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$, Eve can ask for decryptions of anything but \mathbf{y} .
 - ▶ Eve picks a random code word $\mathbf{c} = \bar{\mathbf{m}}G'$, asks for decryption of $\mathbf{y} + \mathbf{c}$.
 - ▶ This is different from challenge \mathbf{y} , so Bob answers.
 - ▶ Answer is $\mathbf{m} + \bar{\mathbf{m}}$.
- ▶ Fix by using CCA2 transformation (e.g. Fujisaki-Okamoto transform) or (easier) KEM/DEM version:
pick random \mathbf{e} of weight t , use $\text{hash}(\mathbf{e})$ as secret key to encrypt and authenticate (for McEliece or Niederreiter).

Generic attack: Brute force

Given K and $\mathbf{s} = K\mathbf{e}$, find \mathbf{e} with $\text{wt}(\mathbf{e}) = t$.

$K =$

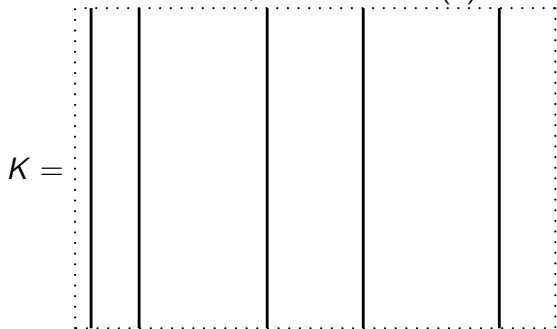


Pick any group of t columns of K , add them and compare with \mathbf{s} .

Cost:

Generic attack: Brute force

Given K and $\mathbf{s} = K\mathbf{e}$, find \mathbf{e} with $\text{wt}(\mathbf{e}) = t$.



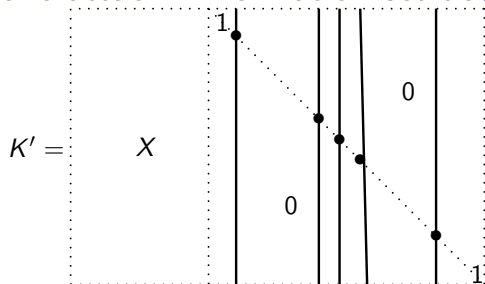
Pick any group of t columns of K , add them and compare with \mathbf{s} .

Cost: $\binom{n}{t}$ sums of t columns.

Can do better so that each try costs only 1 column addition (after some initial additions).

Cost: $O\left(\binom{n}{t}\right)$ additions of 1 column.

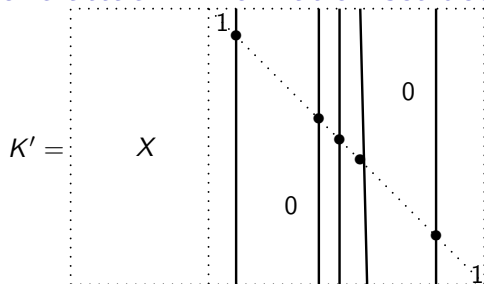
Generic attack: Information-set decoding, 1962 Prange



1. Permute K and bring to systematic form $K' = (X|I_{n-k})$.
(If this fails, repeat with other permutation).
2. Then $K' = UKP$ for some permutation matrix P and U the matrix that produces systematic form.
3. This updates \mathbf{s} to $U\mathbf{s}$.
4. If $\text{wt}(U\mathbf{s}) = t$ then $\mathbf{e}' = (00 \dots 0) \parallel U\mathbf{s}$.
Output unpermuted version of \mathbf{e}' .
5. Else return to **1** to rerandomize.

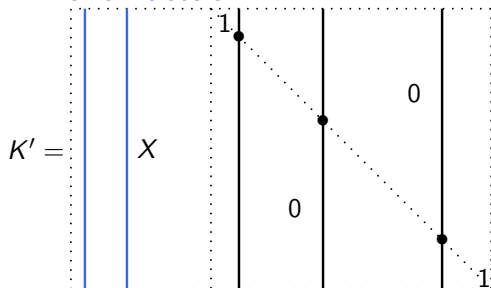
Cost:

Generic attack: Information-set decoding, 1962 Prange



1. Permute K and bring to systematic form $K' = (X|I_{n-k})$.
(If this fails, repeat with other permutation).
 2. Then $K' = UKP$ for some permutation matrix P and U the matrix that produces systematic form.
 3. This updates \mathbf{s} to $U\mathbf{s}$.
 4. If $\text{wt}(U\mathbf{s}) = t$ then $\mathbf{e}' = (00 \dots 0) \parallel U\mathbf{s}$.
Output unpermuted version of \mathbf{e}' .
 5. Else return to **1** to rerandomize.
- Cost: $O\left(\frac{\binom{n}{t}}{\binom{n-k}{t}}\right)$ matrix operations.

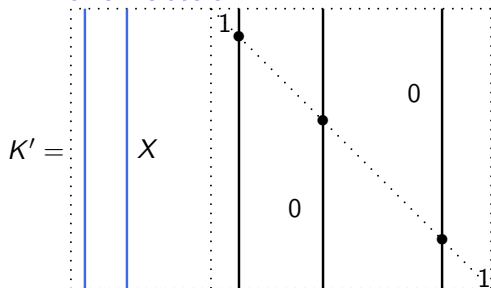
Lee–Brickell attack



1. Permute K and bring to systematic form $K' = (X|I_{n-k})$. (If this fails, repeat with other permutation). \mathbf{s} is updated.
2. For small p , pick p of the k columns on the left, compute their sum $X\mathbf{p}$. (\mathbf{p} is the vector of weight p).
3. If $\text{wt}(\mathbf{s} + X\mathbf{p}) = t - p$ then put $\mathbf{e}' = \mathbf{p} || (\mathbf{s} + X\mathbf{p})$. Output unpermuted version of \mathbf{e}' .
4. Else return to 2 or return to 1 to rerandomize.

Cost:

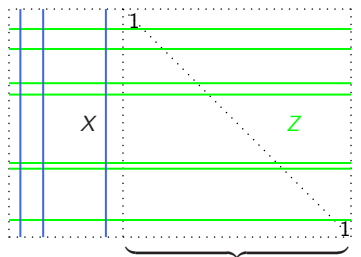
Lee–Brickell attack



1. Permute K and bring to systematic form $K' = (X|I_{n-k})$. (If this fails, repeat with other permutation). \mathbf{s} is updated.
 2. For small p , pick p of the k columns on the left, compute their sum $X\mathbf{p}$. (\mathbf{p} is the vector of weight p).
 3. If $\text{wt}(\mathbf{s} + X\mathbf{p}) = t - p$ then put $\mathbf{e}' = \mathbf{p} || (\mathbf{s} + X\mathbf{p})$. Output unpermuted version of \mathbf{e}' .
 4. Else return to 2 or return to 1 to rerandomize.
- Cost: $O\left(\binom{n}{t} / \left(\binom{k}{p} \binom{n-k}{t-p}\right)\right)$ [matrix operations + $\binom{k}{p}$ column additions].

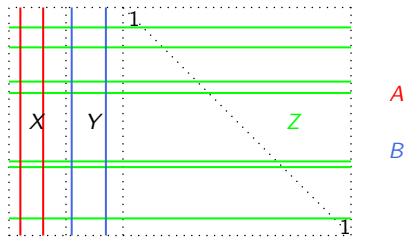
Leon's attack

- ▶ Setup similar to Lee-Brickell's attack.
- ▶ Random combinations of p vectors will be dense, so have $\text{wt}(\mathbf{s} + X\mathbf{p}) \sim k/2$.
- ▶ Idea: Introduce early abort by checking only ℓ positions (selected by set Z , green lines in the picture). This forms $\ell \times k$ matrix X_Z , length- ℓ vector \mathbf{s}_Z .
- ▶ Inner loop becomes:
 1. Pick \mathbf{p} with $\text{wt}(\mathbf{p}) = p$.
 2. Compute $X_Z\mathbf{p}$.
 3. If $\mathbf{s}_Z + X_Z\mathbf{p} \neq \mathbf{0}$ goto 1.
 4. Else compute $X\mathbf{p}$.
 - 4.1 If $\text{wt}(\mathbf{s} + X\mathbf{p}) = t - p$ then put $\mathbf{e}' = \mathbf{p} || (\mathbf{s} + X\mathbf{p})$.
Output unpermuted version of \mathbf{e}' .
 - 4.2 Else return to 1 or rerandomize K .
- ▶ Note that $\mathbf{s}_Z + X_Z\mathbf{p} = \mathbf{0}$ means that there are no ones in the positions specified by Z . Small loss in success, big speedup.



Stern's attack

- ▶ Setup similar to Leon's and Lee-Brickell's attacks.
- ▶ Use the early abort trick, so specify set Z .
- ▶ Improve chances of finding \mathbf{p} with $\mathbf{s} + X_Z \mathbf{p} = 0$:



- ▶ Split left part of K' into two disjoint subsets X and Y .
- ▶ Let $A = \{\mathbf{a} \in \mathbb{F}_2^{k/2} \mid \text{wt}(\mathbf{a}) = p\}$, $B = \{\mathbf{b} \in \mathbb{F}_2^{k/2} \mid \text{wt}(\mathbf{b}) = p\}$.
- ▶ Search for words having exactly p ones in X and p ones in Y and exactly $w - 2p$ ones in the remaining columns.
- ▶ Do the latter part as a collision search:
Compute $\mathbf{s}_Z + X_Z \mathbf{a}$ for all (many) $\mathbf{a} \in A$, sort.
Then compute $Y_Z \mathbf{b}$ for $\mathbf{b} \in B$ and look for collisions; expand.
- ▶ Iterate until word with $\text{wt}(\mathbf{s} + X\mathbf{a} + Y\mathbf{b}) = 2p$ is found for some X, Y, Z .
- ▶ Select p, ℓ , and the subset of A to minimize overall work.

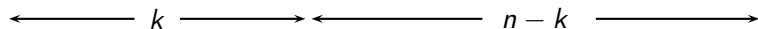
Running time in practice

2008 Bernstein, Lange, Peters.

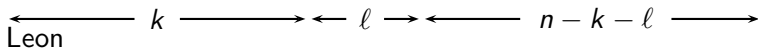
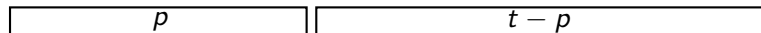
- ▶ Wrote attack software against original McEliece parameters, decoding 50 errors in a $[1024, 524]$ code.
- ▶ Lots of optimizations, e.g. cheap updates between $\mathbf{s}_Z + X_Z \mathbf{a}$ and next value for \mathbf{a} ; optimized frequency of K randomization.
- ▶ Attack on a single computer with a 2.4GHz Intel Core 2 Quad Q6600 CPU would need, on average, 1400 days (2^{58} CPU cycles) to complete the attack.
- ▶ About 200 computers involved, with about 300 cores.
- ▶ Most of the cores put in far fewer than 90 days of work; some of which were considerably slower than a Core 2.
- ▶ Computation used about 8000 core-days.
- ▶ Error vector found by Walton cluster at SFI/HEA Irish Centre of High-End Computing (ICHEC).

Information-set decoding

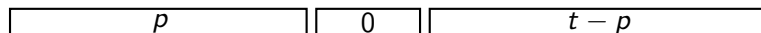
Methods differ in where the “errors” are allowed to be.



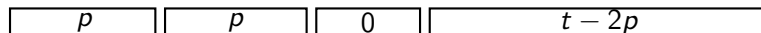
Lee-Brickell



Leon



Stern



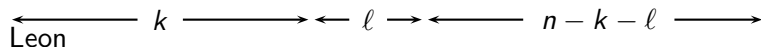
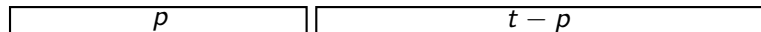
Running time is exponential for Goppa parameters n, k, d .

Information-set decoding

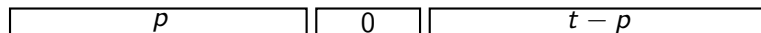
Methods differ in where the errors are allowed to be.



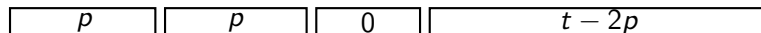
Lee-Brickell



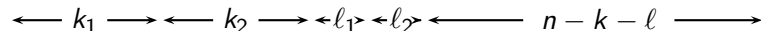
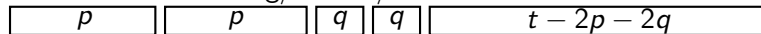
Leon



Stern



Ball-collision decoding/Dumer/Finiasz-Sendrier



2011 May-Meurer-Thomae and 2012 Becker-Joux-May-Meurer refine multi-level collision search. No change in exponent for [Goppa](#) parameters n, k, d .

Improvements

- ▶ Increase n : The most obvious way to defend McEliece's cryptosystem is to increase the code length n .
- ▶ Allow values of n between powers of 2: Get considerably better optimization of (e.g.) the McEliece public-key size.
- ▶ Use list decoding to increase t : Unique decoding is ensured by CCA2-secure variants.
- ▶ Decrease key size by using fields other than \mathbb{F}_2 (wild McEliece).
- ▶ Decrease key size & be faster by using other codes. **Needs security analysis**: some codes have too much structure.

More exciting codes

- ▶ We distinguish between generic attacks (such as information-set decoding) and structural attacks (that use the structure of the code).
- ▶ Gröbner basis computation is a generally powerful tool for structural attacks.
- ▶ Cyclic codes need to store only top row of matrix, rest follows by shifts. Quasi-cyclic: multiple cyclic blocks.
- ▶ QC Goppa: too exciting, too much structure.
- ▶ Interesting candidate: Quasi-cyclic Moderate-Density Parity-Check (QC-MDPC) codes, due to Misoczki, Tillich, Sendrier, and Barreto (2012).
Very efficient but practical problem if the key is reused (Asiacrypt 2016).
- ▶ Hermitian codes, general algebraic geometry codes.
- ▶ Please help us update <https://pqcrypto.org/code.html>.

Bonus slides

RaCoSS – Random Code-based Signature Schemes

- ▶ “Code-based” does not imply secure!

RaCoSS – Random Code-based Signature Schemes

- ▶ “Code-based” does not imply secure!
- ▶ System parameters: $n = 2400$, $k = 2060$.
Random matrix $H \in \mathbb{F}_2^{(n-k) \times n}$.
- ▶ Secret key: sparse $S \in \mathbb{F}_2^{n \times n}$.
- ▶ Public key: $T = H \cdot S$. (looks pretty random).
- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .

RaCoSS – Random Code-based Signature Schemes

- ▶ “Code-based” does not imply secure!
- ▶ System parameters: $n = 2400$, $k = 2060$.
Random matrix $H \in \mathbb{F}_2^{(n-k) \times n}$.
- ▶ Secret key: sparse $S \in \mathbb{F}_2^{n \times n}$.
- ▶ Public key: $T = H \cdot S$. (looks pretty random).
- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

RaCoSS – Random Code-based Signature Schemes

- ▶ “Code-based” does not imply secure!
- ▶ System parameters: $n = 2400$, $k = 2060$.
Random matrix $H \in \mathbb{F}_2^{(n-k) \times n}$.
- ▶ Secret key: sparse $S \in \mathbb{F}_2^{n \times n}$.
- ▶ Public key: $T = H \cdot S$. (looks pretty random).
- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- ▶ Why are these equal?
$$v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc$$

RaCoSS – Random Code-based Signature Schemes

- ▶ “Code-based” does not imply secure!
- ▶ System parameters: $n = 2400$, $k = 2060$.
Random matrix $H \in \mathbb{F}_2^{(n-k) \times n}$.
- ▶ Secret key: sparse $S \in \mathbb{F}_2^{n \times n}$.
- ▶ Public key: $T = H \cdot S$. (looks pretty random).
- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- ▶ Why are these equal?
 $v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc = Hy = v$
- ▶ Why does the weight restriction hold?

RaCoSS – Random Code-based Signature Schemes

- ▶ “Code-based” does not imply secure!
- ▶ System parameters: $n = 2400$, $k = 2060$.
Random matrix $H \in \mathbb{F}_2^{(n-k) \times n}$.
- ▶ Secret key: sparse $S \in \mathbb{F}_2^{n \times n}$.
- ▶ Public key: $T = H \cdot S$. (looks pretty random).
- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- ▶ Why are these equal?
 $v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc = Hy = v$
- ▶ Why does the weight restriction hold?
 S and y are sparse, but each entry in Sc is sum over n positions

$$z_i = y_i + \sum_{j=1}^n S_{ij}c_j.$$

RaCoSS – Random Code-based Signature Schemes

- ▶ “Code-based” does not imply secure!
- ▶ System parameters: $n = 2400$, $k = 2060$.
Random matrix $H \in \mathbb{F}_2^{(n-k) \times n}$.
- ▶ Secret key: sparse $S \in \mathbb{F}_2^{n \times n}$.
- ▶ Public key: $T = H \cdot S$. (looks pretty random).
- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- ▶ Why are these equal?
 $v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc = Hy = v$
- ▶ Why does the weight restriction hold?
 S and y are sparse, but each entry in Sc is sum over n positions

$$z_i = y_i + \sum_{j=1}^n S_{ij}c_j.$$

This needs a special hash function so that c is **very** sparse.

The weight-restricted hash function (wrhf)

- ▶ Maps to 2400-bit strings of weight 3.

The weight-restricted hash function (wrhf)

- ▶ Maps to 2400-bit strings of weight 3.
- ▶ Only

$$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$

possible outputs.

The weight-restricted hash function (wrhf)

- ▶ Maps to 2400-bit strings of weight 3.
- ▶ Only

$$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$

possible outputs.

- ▶ Slow: 600 to 800 hashes per second and core.
- ▶ Expected time for a preimage on ≈ 100 cores: **10 hours**.

Implementation bug:

```
unsigned char  c[RACOSS_N];
unsigned char  c2[RACOSS_N];

/* ... */

for( i=0 ; i<(RACOSS_N/8) ; i++ )
    if( c2[i] != c[i] )
        /* fail */

return 0; /* accept */
```

Implementation bug:

```
unsigned char  c[RACOSS_N];
unsigned char  c2[RACOSS_N];

/* ... */

for( i=0 ; i<(RACOSS_N/8) ; i++ )
    if( c2[i] != c[i] )
        /* fail */

return 0; /* accept */
```


Implementation bug:

```
unsigned char  c[RACOSS_N];
unsigned char  c2[RACOSS_N];

/* ... */

for( i=0 ; i<(RACOSS_N/8) ; i++ )
    if( c2[i] != c[i] )
        /* fail */

return 0; /* accept */
```

...compares only the first 300 coefficients!

Thus, a signature with $c[0\dots299] = 0$ is accepted for

$$\binom{2100}{3} / \binom{2400}{3} \approx 67\%$$

of all messages.

The weight-restricted hash function (wrhf)

- ▶ Maps to 2400-bit strings of weight 3.
- ▶ Only

$$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$

possible outputs.

- ▶ Slow: 600 to 800 hashes per second and core.
- ▶ Expected time for a preimage on ≈ 100 cores: 10 hours.
- ▶ crashed while brute-forcing: *memory leaks*
- ▶ another message signed by the first KAT:

NISTPQC is so much fun! 10900qmmP

Wait, there is more!

- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \\ \\ \end{pmatrix} = \begin{pmatrix} & & \\ & H & \\ & & \end{pmatrix} \begin{pmatrix} \\ \\ z \end{pmatrix}$$

- ▶ Sign without knowing S : $(c, y, z \in \mathbb{F}_2^n, v, Tc \in \mathbb{F}_2^{n-k})$.

Wait, there is more!

- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \\ \\ \end{pmatrix} = \begin{pmatrix} & & \\ & H & \\ & & \end{pmatrix} \begin{pmatrix} \\ \\ z \end{pmatrix}$$

- ▶ Sign without knowing S : ($c, y, z \in \mathbb{F}_2^n$, $v, Tc \in \mathbb{F}_2^{n-k}$).
Pick a low weight $y \in \mathbb{F}_2^n$. Compute $v = Hy$, $c = h(v, m)$.

Wait, there is more!

- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \\ \\ \\ \end{pmatrix} = \begin{pmatrix} & & & \\ & H & & \\ & & & \\ & & & \end{pmatrix} \begin{pmatrix} \\ \\ \\ z \end{pmatrix}$$

- ▶ Sign without knowing S : ($c, y, z \in \mathbb{F}_2^n$, $v, Tc \in \mathbb{F}_2^{n-k}$).
Pick a low weight $y \in \mathbb{F}_2^n$. Compute $v = Hy$, $c = h(v, m)$.
Pick $n - k$ columns of H that form an invertible matrix H_1 .

Wait, there is more!

- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \\ \\ \end{pmatrix} = \begin{pmatrix} & \\ H_1 & H_2 \end{pmatrix} \begin{pmatrix} z_1 \\ \\ z_2 \end{pmatrix}$$

- ▶ Sign without knowing S : ($c, y, z \in \mathbb{F}_2^n$, $v, Tc \in \mathbb{F}_2^{n-k}$).
Pick a low weight $y \in \mathbb{F}_2^n$. Compute $v = Hy$, $c = h(v, m)$.
Pick $n - k$ columns of H that form an invertible matrix H_1 .

Wait, there is more!

- ▶ Sign m : Pick a low weight $y \in \mathbb{F}_2^n$.
Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c) .
- ▶ Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \\ \\ \end{pmatrix} = \begin{pmatrix} & \\ H_1 & H_2 \end{pmatrix} \begin{pmatrix} z_1 \\ \\ \\ z_2 \end{pmatrix}$$

- ▶ Sign without knowing S : ($c, y, z \in \mathbb{F}_2^n$, $v, Tc \in \mathbb{F}_2^{n-k}$).
Pick a low weight $y \in \mathbb{F}_2^n$. Compute $v = Hy$, $c = h(v, m)$.
Pick $n - k$ columns of H that form an invertible matrix H_1 .
- ▶ Compute $z = (z_1 || 00 \dots 0)$ by linear algebra.
- ▶ Expected weight of z is $\approx (n - k)/2 = 170 \ll 1564$.
- ▶ Properly generated signatures have $\text{weight}(z) \approx 261$.

RaCoSS – Summary

- ▶ Bug in code: bit vs. byte confusion meant only every 8th bit verified.
- ▶ Preimages for RaCoSS' special hash function: only

$$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$

possible outputs.

- ▶ The code dimensions give a lot of freedom to the attacker – our forged signature is better than a real one!