Code-based cryptanalysis

Daniel J. Bernstein & Tanja Lange
with some slides by Tung Chou and Christiane Peters

Guest lecture at National Taiwan University

27 April 2026

Basic concepts and McEliece system

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).

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 $G \in \mathbb{F}_2^{k \times n}$

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

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

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

  Leaving out the $^\mathsf{T}$ from now on.
- Names: code word $\mathbf{c}$, error vector $\mathbf{e}$, received word $\mathbf{b} = \mathbf{c} + \mathbf{e}$.

Example: 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:

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

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

Example: 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:

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

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.

Example: 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:

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

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 $H \cdot \mathbf{b}$.

Linear codes are linear

Example with generator matrix:

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

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

Linear codes are linear

Example with generator matrix:

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

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

Linear codes are linear:
The sum of two code words is a code word:

Linear codes are linear

Example with generator matrix:

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

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

Linear codes are linear:
The sum of two code words is a code word:

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

Same with parity-check matrix:

Linear codes are linear

Example with generator matrix:

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

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

Linear codes are linear:
The sum of two code words is a code word:

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

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.

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

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

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

Hamming weight and distance

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

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

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

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

Hamming weight and distance

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

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

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

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

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

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

Minimum distance

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

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

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

Decoding problem

Decoding problem: find the closest code word $\mathbf{c} \in C$ to a given $\mathbf{x} \in \mathbb{F}_2^n$, assuming that there is a unique closest code word. 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

- Due to Robert McEliece 1978.
- 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

- The McEliece public key is the $k \times n$ matrix $G' = SGP$.
- 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

- The McEliece public key is the $k \times n$ matrix $G' = SGP$.
- 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 code word $\mathbf{m}G'$ in the code generated by G'.
  This is general decoding if G' does not expose any structure.

Niederreiter system and schoolbook attacks

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 $H = (Q^\intercal|I_{n-k})$.
  Then

$$H(\mathbf{m}G)^\intercal = HG^\intercal\mathbf{m}^\intercal = (Q^\intercal|I_{n-k})(I_k|Q)^\intercal\mathbf{m}^\intercal = 0.$$

- Can reduce storage / transmission bandwidth by leaving out the identity matrix part. E.g. for the parity-check matrix:

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

  Any use of H includes the identity matrix in the computations.

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 $H = (Q^\mathsf{T} | I_{n-k})$.

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 $H = (Q^\mathsf{T} | I_{n-k})$.
  Then $\mathbf{x} = (00\ldots0)||\mathbf{s}$ 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 an n-bit vector $\mathbf{e}$ with $\mathrm{wt}(\mathbf{e}) = t$ 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 $\mathrm{wt}(P\mathbf{e}) = t$, 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 Shestakov.
- 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 had several entries on QCMDPC codes.
  HQC chosen to be standardized.
- Ongoing NIST competition on signatures received several code-based entries as well.
- But not everything code-based is secure (see last part).

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 $\mathrm{wt}(\mathbf{e}) \leq t$.
- 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 $\mathrm{wt}(\mathbf{e}) = t$. (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 $\mathrm{wt}(\mathbf{e}) \leq t$.
- 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 $\mathrm{wt}(\mathbf{e}) = t$. (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 $\mathrm{wt}(\bar{\mathbf{e}}) = 2t$:

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 $\mathrm{wt}(\bar{\mathbf{e}}) = 2t$:
  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 $\mathrm{wt}(\bar{\mathbf{e}}) = 2t$:
  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 $2w = \mathrm{wt}(\bar{\mathbf{e}}) < 2t$ 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 CCA2 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 CCA2 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 CCA2 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 hash($\mathbf{e}$) as secret key to encrypt and authenticate (for McEliece or Niederreiter).

Goppa codes: definition and usage

Binary Goppa code

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

- a list $L = (a_1, \ldots, a_n)$ 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

$$\left\{ \mathbf{c} \in \mathbb{F}_2^n \,\middle|\, S(\mathbf{c}) = \frac{c_1}{x - a_1} + \frac{c_2}{x - a_2} + \cdots + \frac{c_n}{x - a_n} \equiv 0 \bmod g(x) \right\}$$

- This code is linear $S(\mathbf{b} + \mathbf{c}) = S(\mathbf{b}) + S(\mathbf{c})$ and has length n.
- What can we say about the dimension and minimum distance?

Dimension of $\Gamma(L, g)$

- $g(a_i) \neq 0$ implies $\gcd(x - a_i, g(x)) = 1$, thus get polynomials

$$(x - a_i)^{-1} \equiv f_i(x) \equiv \sum_{j=0}^{t-1} f_{i,j} x^j \bmod g(x)$$

  via XGCD. All this is over $\mathbb{F}_q = \mathbb{F}_{2^m}$.

- In this form, $S(\mathbf{c}) \equiv 0 \bmod g(x)$ means

$$\sum_{i=1}^{n} c_i \left(\sum_{j=0}^{t-1} f_{i,j} x^j \right) = \sum_{j=0}^{t-1} \left(\sum_{i=1}^{n} c_i f_{i,j} \right) x^j = 0,$$

  meaning that for each $0 \leq j \leq t - 1$:

$$\sum_{i=1}^{n} c_i f_{i,j} = 0.$$

- These are t conditions over $\mathbb{F}_q$, so tm conditions over $\mathbb{F}_2$. Giving an $tm \times n$ parity check matrix over $\mathbb{F}_2$.
- Some rows might be linearly dependent, so $k \geq n - tm$.

Nice parity check matrix

Assume $g(x) = \sum_{i=0}^{t} g_i x^i$ monic, i.e., $g_t = 1$.

$$H = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ g_{t-1} & 1 & 0 & \dots & 0 \\ g_{t-2} & g_{t-1} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \dots & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ a_1 & a_2 & a_3 & \cdots & a_n \\ a_1^2 & a_2^2 & a_3^2 & \cdots & a_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1^{t-1} & a_2^{t-1} & a_3^{t-1} & \cdots & a_n^{t-1} \end{pmatrix}$$

$$\cdot \begin{pmatrix} \frac{1}{g(a_1)} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{g(a_2)} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{g(a_3)} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{1}{g(a_n)} \end{pmatrix}$$

To learn about the minimum distance of a Goppa code and how to decode them, watch Tanja's PQC YouTube channel, in particular lecture 4 from the code-based crypto playlist.
We cover one decoder here later.

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}.

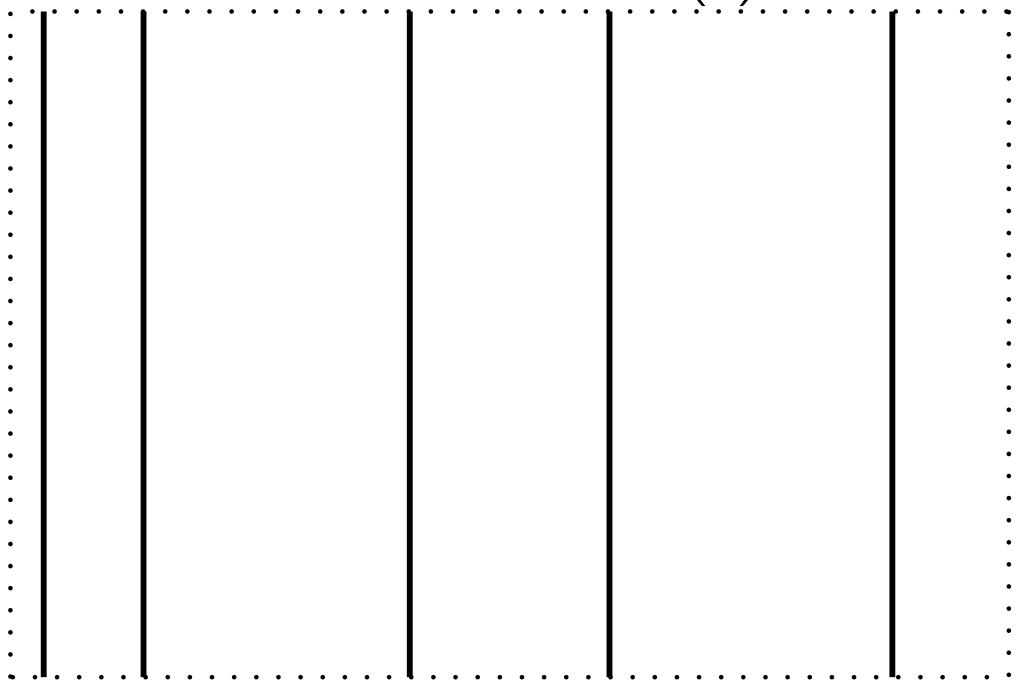
How to hide nice code?

- 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 public key;
  Store only K' part.
  - This implicitly applies S.
  - No need to remember S because decoding does not use H. (see Code-based crypto IV).
  - Public key size decreased to $(n-k) \times k$.
- Private key is polynomial g and support $L = (a_1, \ldots, a_n)$.

Information-set decoding

Generic attack: Brute force

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

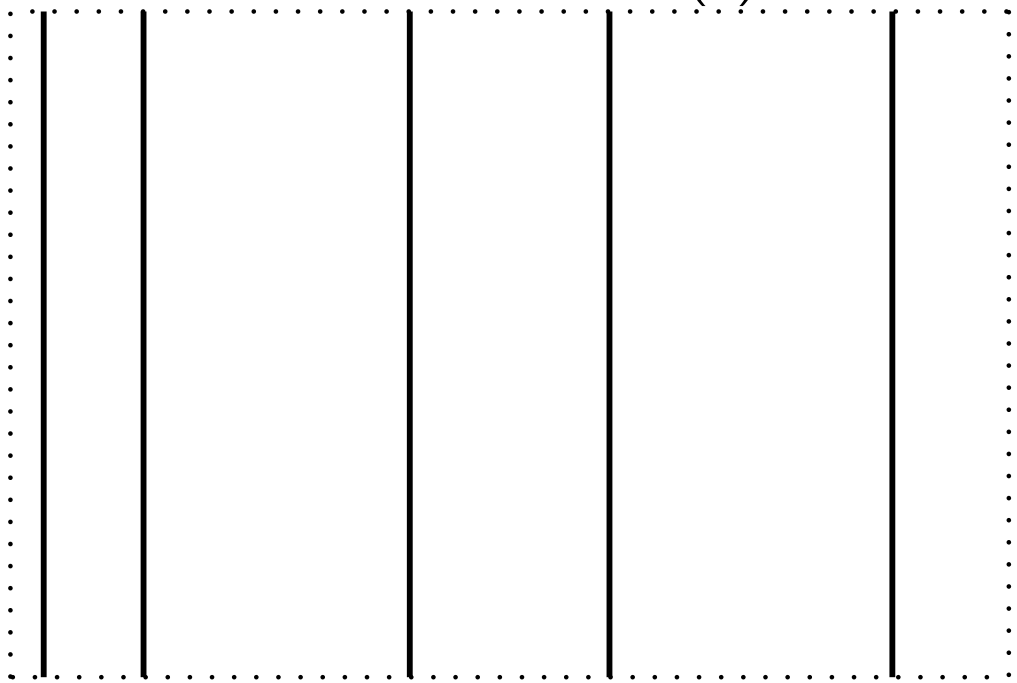
$K =$ 

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

Cost:

Generic attack: Brute force

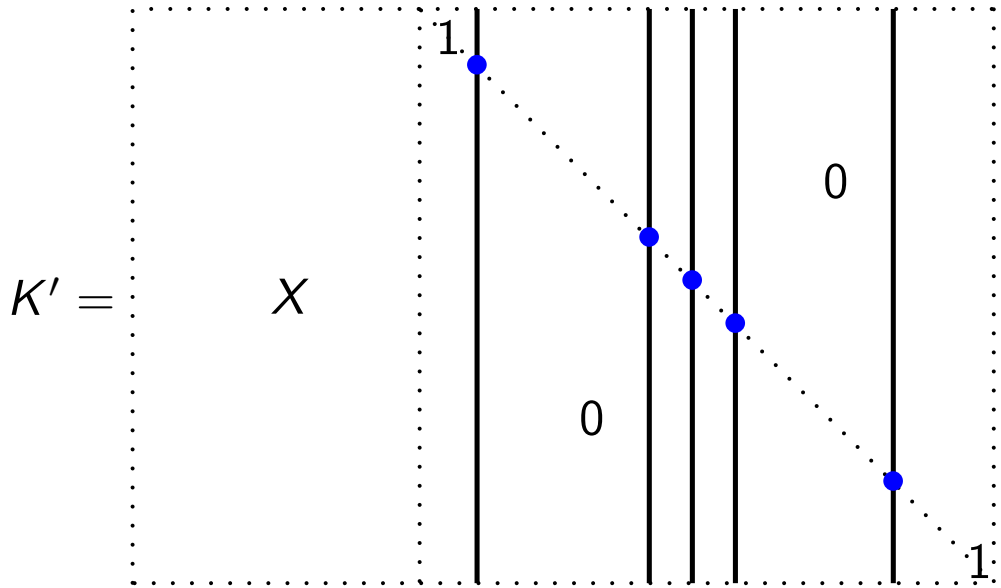
Given K and $\mathbf{s} = K\mathbf{e}$, find $\mathbf{e}$ with $\mathrm{wt}(\mathbf{e}) = t$.

$K =$ 

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: $O\binom{n}{t}$ additions of 1 column.

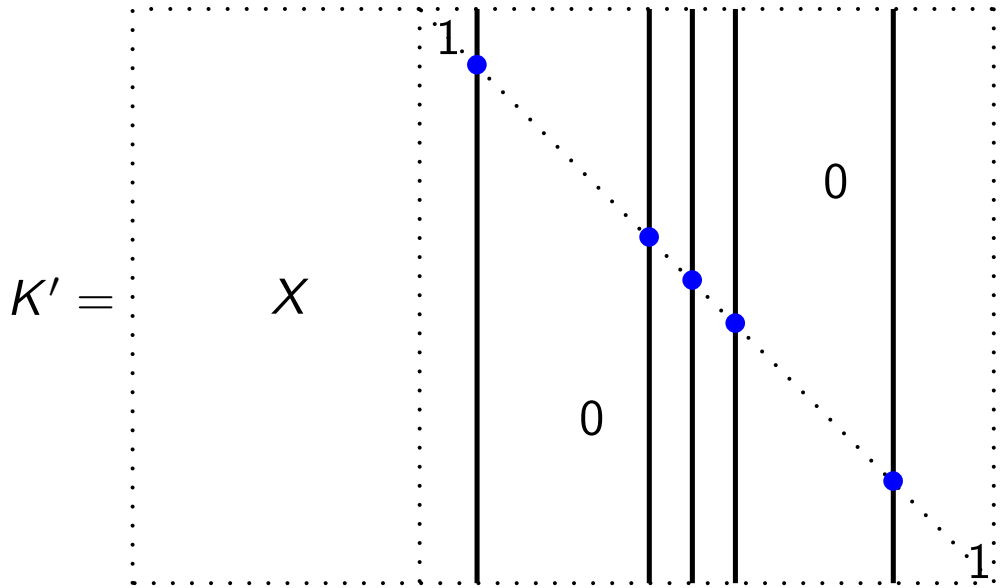
Generic attack: Information-set decoding, 1962 Prange



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 $\mathrm{wt}(U\mathbf{s}) = t$ then $\mathbf{e}' = (00\ldots0)||U\mathbf{s}$.
   Output unpermuted version of $\mathbf{e}'$.
5. Else return to 1 to rerandomize.

Cost:

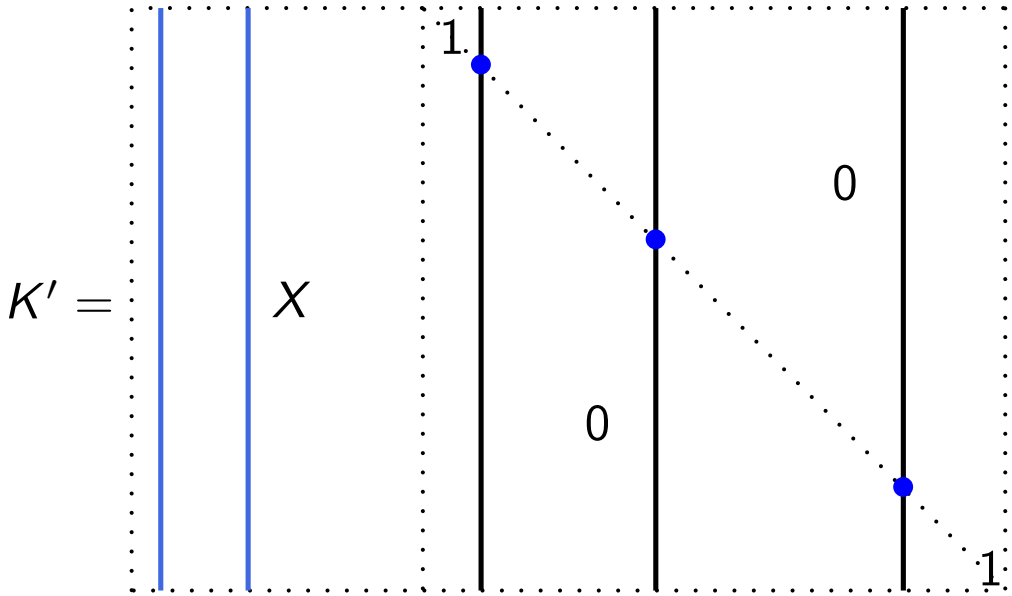
Generic attack: Information-set decoding, 1962 Prange



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 $\mathrm{wt}(U\mathbf{s}) = t$ then $\mathbf{e}' = (00\ldots0)||U\mathbf{s}$.
   Output unpermuted version of $\mathbf{e}'$.
5. Else return to 1 to rerandomize.

Cost: $O(\binom{n}{t}/\binom{n-k}{t})$ matrix operations.

Lee–Brickell attack

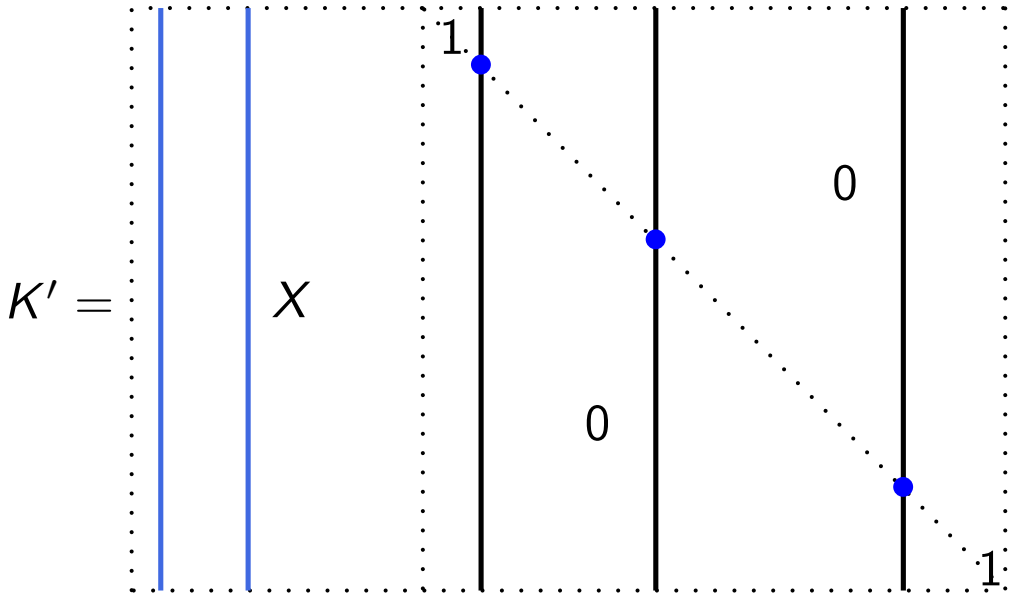


$\mathbf{s}' = K'\mathbf{e}'$

1. Permute K and bring to systematic form $K' = (X|I_{n-k})$.
   (If this fails, repeat with other permutation). $\mathbf{s}$ is updated to $\mathbf{s}'$.
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 $\mathrm{wt}(\mathbf{s}' + X\mathbf{p}) = t - p$ 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

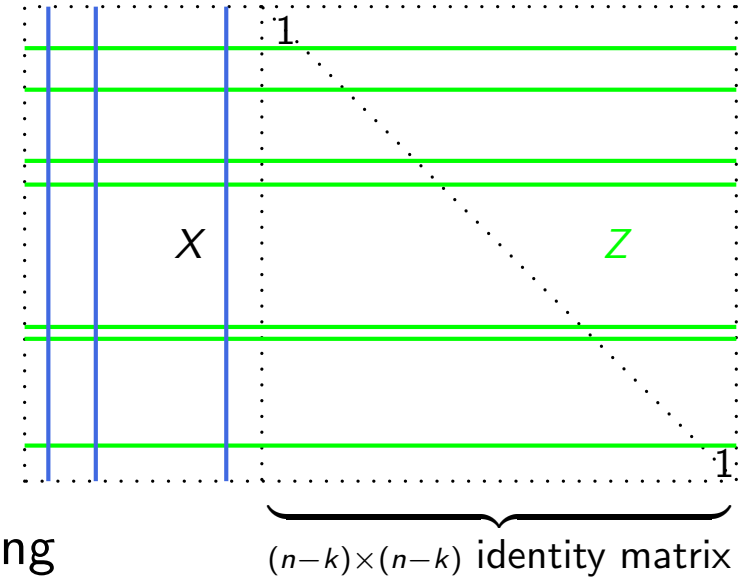


$\mathbf{s}' = K'\mathbf{e}'$

1. Permute K and bring to systematic form $K' = (X|I_{n-k})$.
   (If this fails, repeat with other permutation). $\mathbf{s}$ is updated to $\mathbf{s}'$.
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 $\mathrm{wt}(\mathbf{s}' + X\mathbf{p}) = t - p$ 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.

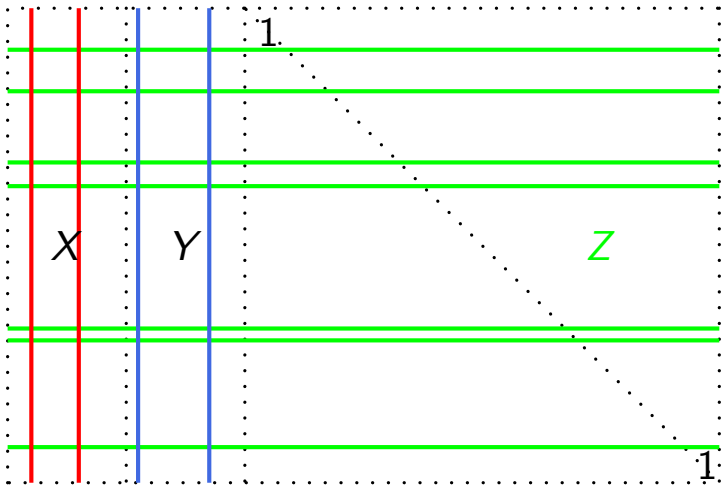
Cost: $O(\binom{n}{t}/(\binom{k}{p}\binom{n-k}{t-p}))$ [matrix operations$+\binom{k}{p}$ column additions].

Leon's attack



- Setup similar to Lee-Brickell's attack.
- Random combinations of p vectors will be dense, so have $\mathrm{wt}(\mathbf{s}' + X\mathbf{p}) \sim (n-k)/2$.
- 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 $\mathrm{wt}(\mathbf{p}) = p$.
  2. Compute $X_Z\mathbf{p}$.
  3. If $\mathbf{s}'_Z + X_Z\mathbf{p} \neq 0$ goto 1.
  4. Else compute $X\mathbf{p}$.
     1. If $\mathrm{wt}(\mathbf{s}' + X\mathbf{p}) = t - p$ then put $\mathbf{e}' = \mathbf{p}||(\mathbf{s}' + X\mathbf{p})$. Output unpermuted version of $\mathbf{e}'$.
     2. Else return to 1 or rerandomize K.
- Note that $\mathbf{s}'_Z + X_Z\mathbf{p} = 0$ 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 $A = \{\mathbf{a} \in \mathbb{F}_2^{k/2} | \mathrm{wt}(\mathbf{a}) = p\}$, $B = \{\mathbf{b} \in \mathbb{F}_2^{k/2} | \mathrm{wt}(\mathbf{b}) = p\}$.
  - Search for words having exactly p ones in X and p ones in Y and exactly $t - 2p$ 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 $\mathrm{wt}(\mathbf{s}' + X\mathbf{a} + Y\mathbf{b}) = t - 2p$ is found for some X, Y, Z.
- Select p, ℓ, and the subset of A to minimize overall work.

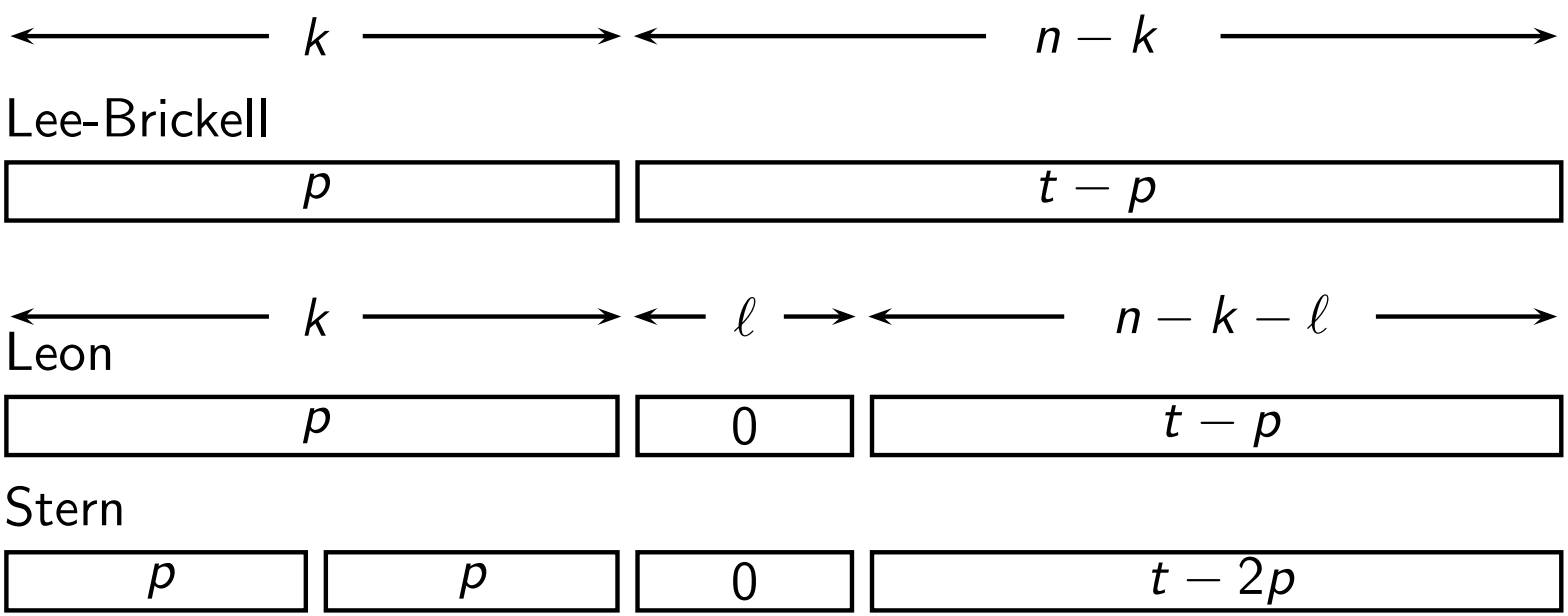
Running time in practice

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

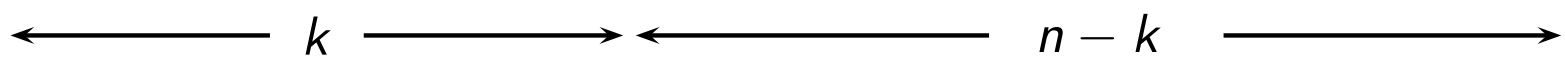
Methods differ in where the errors are allowed to be.



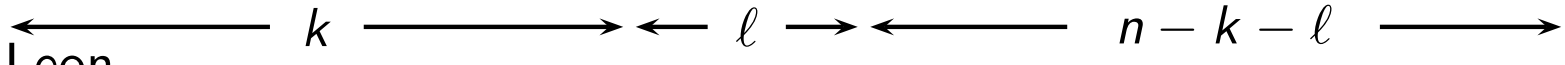
Running time is exponential for Goppa parameters n, k, d.

Information-set decoding

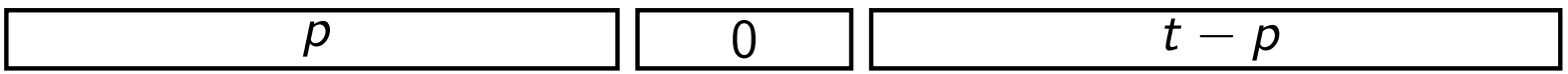
Methods differ in where the errors are allowed to be.



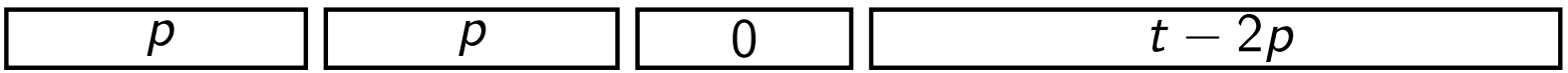
Lee-Brickell



Leon



Stern



Ball-collision decoding/Dumer/Finiasz-Sendrier

2011 May-Meurer-Thomae and 2012 Becker-Joux-May-Meurer refine multi-level collision search.

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 Tilborg;
2009 Bernstein (**post-quantum**); 2009 Finiasz–Sendrier; 2010 Bernstein–Lange–Peters;
2009 Bernstein–Lange–Peters–van Tilborg; 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**); 2020 Debris-Alazard–Ducas–van Woerden;
2021 Esser–Bellini; 2021 Perriello–Barenghi–Pelosi (**post-quantum**);
2021 Esser–Ramos-Calderer–Bellini–Latorre–Manzano (**post-quantum**);
2021 Esser–May–Zweydinger; 2022 Carrier–Debris-Alazard–Meyer-Hilfiger–Tillich; 2022 Esser;
2022 Esser–Zweydinger; 2022 Wakasugi–Tada; 2023 Guo–Johansson–Nguyen;
2023 Narisada–Fukushima–Kiyomoto; 2023 Li–Wang;
2023 Kimura–Takayasu–Takagi (**post-quantum**); 2023 Bernstein–Chou;
2023 Perriello–Barenghi–Pelosi (**post-quantum**); 2023 Ducas–Esser–Etinski–Kirshanova;
2023 Bhattacharyya–Sarkar; 2024 Narisada–Uemura–Okada–Furue–Aikawa–Fukushima;
2024 Chevignard–Fouque–Schrottenloher (**post-quantum**);
2024 Perriello–Barenghi–Pelosi (**post-quantum**); 2024 Engelberts–Etinski–Loyer (**post-quantum**);
2024 Yoshiguchi–Aikawa–Takagi; 2025 Elbro–Santini; 2026 May–Diogo;
2026 Yu–Jiang–Wang–Chen–Cheng–Huang–Zhu.

"Code-based" is not necessarily secure!

Here are examples of fast attacks we found against two code-based proposals:
2017 against NIST submission RaCoSS;
2024 against KpqC submission PALOMA.

RaCoSS – Random Code-based Signature Scheme

- 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: $T = H \cdot S$. (Not obvious how to recover S.)
- 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 Scheme

- 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: $T = H \cdot S$. (Not obvious how to recover S.)
- 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 $\mathrm{weight}(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

RaCoSS – Random Code-based Signature Scheme

- 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: $T = H \cdot S$. (Not obvious how to recover S.)
- 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 $\mathrm{weight}(z) \leq 1564$.
  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 Scheme

- 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: $T = H \cdot S$. (Not obvious how to recover S.)
- 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 $\mathrm{weight}(z) \leq 1564$.
  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 Scheme

- 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: $T = H \cdot S$. (Not obvious how to recover S.)
- 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 $\mathrm{weight}(z) \leq 1564$.
  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 Scheme

- 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: $T = H \cdot S$. (Not obvious how to recover S.)
- 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 $\mathrm{weight}(z) \leq 1564$.
  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 sparse.

RaCoSS – Random Code-based Signature Scheme

- 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: $T = H \cdot S$. (Not obvious how to recover S.)
- 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 $\mathrm{weight}(z) \leq 1564$.
  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.

RaCoSS

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 */
```

RaCoSS

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 */
```

RaCoSS

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 $c[0...299] = 0$ 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 $\mathrm{weight}(z) \leq 1564$.
  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 $\mathrm{weight}(z) \leq 1564$.
  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 $\mathrm{weight}(z) \leq 1564$.
  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 $\mathrm{weight}(z) \leq 1564$.
  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 $\mathrm{weight}(z) \leq 1564$.
  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 $z = (z_1 || 00 \ldots 0)$ by linear algebra.
- Expected weight of z is $\approx (n - k)/2 = 170 \ll 1564$.
- Properly generated signatures have 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!

PALOMA – submission to Korean KpqC competition

- PALOMA is a code-based cryptosystem using Goppa codes.
- Parameters: m, n, and t.
  - $m = 13$.
  - $n < 2^m$ is code length.
    $n \in \{3904, 5568, 6592\}$.
  - t is number of errors code can efficiently correct.
    $t \in \{64, 128\}$.
- pk is a $mt \times (n - mt)$ matrix M over $\mathbb{F}_2$.
- pk expands to $mt \times n$ matrix $\hat{H} = [I|M]$.
- Encryption: $\hat{s} = \hat{H}\hat{e}$, for $\mathrm{wt}(\hat{e}) = t$.

PALOMA – submission to Korean KpqC competition

- PALOMA is a code-based cryptosystem using Goppa codes.
- Parameters: m, n, and t.
  - $m = 13$.
  - $n < 2^m$ is code length.
    $n \in \{3904, 5568, 6592\}$.
  - t is number of errors code can efficiently correct.
    $t \in \{64, 128\}$.
- pk is a $mt \times (n - mt)$ matrix M over $\mathbb{F}_2$.
- pk expands to $mt \times n$ matrix $\hat{H} = [I|M]$.
- Encryption: $\hat{s} = \hat{H}\hat{e}$, for $\mathrm{wt}(\hat{e}) = t$.
- Decryption uses Goppa decoder to retrieve $\hat{e}$ from $\hat{s}$.
- Assumption: $\hat{H} = SHP'$ hides structured Goppa matrix H,
  P' random permutation matrix expanded from some seed $r_{\hat{\mathcal{C}}}$,
  S invertible matrix to get $\hat{H} = [I|M]$.

KeyGen in PALOMA

PALOMA chooses

$$g(x) = \prod_{\alpha \in T} (x - \alpha)$$

for $T \subseteq \mathbb{F}_q \setminus \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ with $|T| = t$. Hence, $g(x)$ splits completely over $\mathbb{F}_q$.

PALOMA KeyGen, main secret is string r:

1. $(\alpha_1, \alpha_2, \ldots, \alpha_q) = \text{SHUFFLE}_r(\mathbb{F}_q)$.
2. $L = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, $T = (\alpha_{n+1}, \alpha_{n+2}, \ldots, \alpha_{n+t})$.
3. Compute g and parity-check matrix H.
4. Pick random permutation matrix P', compute HP' & bring to systematic form, repeat this step if fails.

Secrets are L, g, and P'; sk includes S with $\hat{H} = SHP' = [I|M]$.
Public key is M, the rightmost $n - mt$ columns of $\hat{H}$.
P' effectively changes order of elements in L.

Patterson decoding of $\mathbf{c} + \mathbf{e}$ in $\Gamma(L, g)$ if $\gcd(g, s) = 1$

$$s(x) = \sum_{i=1}^{n} (c_i + e_i)/(x - \alpha_i)$$

Patterson decoding of $\mathbf{c} + \mathbf{e}$ in $\Gamma(L, g)$ if $\gcd(g, s) = 1$

$$s(x) = \sum_{i=1}^{n} (c_i + e_i)/(x - \alpha_i) \equiv \left(\sum_{i=1}^{n} e_i \prod_{j \neq i} (x - \alpha_j) \right) / \prod_{i=1}^{n} (x - \alpha_i) \bmod g(x).$$

Patterson decoding of $\mathbf{c} + \mathbf{e}$ in $\Gamma(L, g)$ if $\gcd(g, s) = 1$

$$s(x) = \sum_{i=1}^{n} (c_i + e_i)/(x - \alpha_i) \equiv \left(\sum_{i=1}^{n} e_i \prod_{j \neq i} (x - \alpha_j) \right) / \prod_{i=1}^{n} (x - \alpha_i) \bmod g(x).$$

- Put $\sigma(x) = \prod_{i=1}^{n} (x - \alpha_i)^{e_i}$ with $e_i \in \{0, 1\}$, then $\sigma'(x) = \sum_{i=1}^{n} e_i \prod_{j \neq i} (x - \alpha_j)^{e_j}$.
- Thus $(x) \equiv \sigma'(x)/\sigma(x) \bmod g(x)$. We want to find σ.

Patterson decoding of $\mathbf{c}+\mathbf{e}$ in $\Gamma(L,g)$ if $\gcd(g,s)=1$

$$s(x)=\sum_{i=1}^{n}(c_i+e_i)/(x-\alpha_i)\equiv\left(\sum_{i=1}^{n}e_i\prod_{j\neq i}(x-\alpha_j)\right)/\prod_{i=1}^{n}(x-\alpha_i) \bmod g(x).$$

- Put $\sigma(x)=\prod_{i=1}^{n}(x-\alpha_i)^{e_i}$ with $e_i\in\{0,1\}$, then $\sigma'(x)=\sum_{i=1}^{n}e_i\prod_{j\neq i}(x-\alpha_j)^{e_j}$.
- Thus $(x)\equiv\sigma'(x)/\sigma(x) \bmod g(x)$. We want to find σ.
- Split $\sigma(x)$ into odd and even terms: $\sigma(x)=A^2(x)+xB^2(x)$ with $\sigma'(x)=B^2(x)$.
- Thus $B^2(x)\equiv\sigma(x)s(x)\equiv(A^2(x)+xB^2(x))s(x) \bmod g(x)$

  $$B^2(x)(x+1/s(x))\equiv A^2(x) \bmod g(x)$$

- Put $v(x)\equiv\sqrt{x+1/s(x)} \bmod g(x)$, then $A(x)\equiv B(x)v(x) \bmod g(x)$.
- Can compute $v(x)$ from $s(x)$.
- Use XGCD on v and g, stop when $\deg(A)\leq\lfloor t/2\rfloor,\deg(B)\leq\lfloor(t-1)/2\rfloor$ in

  $$A(x)=B(x)v(x)+h(x)g(x).$$

Extended Patterson decoder to handle $\gcd(g, s) \neq 1$

PALOMA uses extended Patterson decoder for reducible g.

- Let $\tilde{s} = 1 + xs$ and $g_1 = \gcd(g, s), g_2 = \gcd(g, \tilde{s}), g_{12} = g/(g_1 g_2)$.
- Compute $\tilde{s}_2 = \tilde{s}/g_2$ and $s_1 = s/g_1$.
- Replace $u(x) = x + 1/s(x)$ by

$$u = g_1 \tilde{s}_2/(g_2 s_1) \bmod g_{12}$$

Extended Patterson decoder to handle $\gcd(g, s) \neq 1$

PALOMA uses extended Patterson decoder for reducible g.

- Let $\tilde{s} = 1 + xs$ and $g_1 = \gcd(g, s), g_2 = \gcd(g, \tilde{s}), g_{12} = g/(g_1 g_2)$.
- Compute $\tilde{s}_2 = \tilde{s}/g_2$ and $s_1 = s/g_1$.
- Replace $u(x) = x + 1/s(x)$ by

$$u = g_1 \tilde{s}_2/(g_2 s_1) \bmod g_{12}$$

- Deal with complication of computing $v = \sqrt{u} \bmod g_{12}$ for reducible g_{12}.
- Let half-gcd return A', B', put

$$\sigma(x) = (A' g_2)^2 + x(B' g_1)^2.$$

Extended Patterson decoder to handle $\gcd(g, s) \neq 1$

PALOMA uses extended Patterson decoder for reducible g.

- Let $\tilde{s} = 1 + xs$ and $g_1 = \gcd(g, s), g_2 = \gcd(g, \tilde{s}), g_{12} = g/(g_1 g_2)$.
- Compute $\tilde{s}_2 = \tilde{s}/g_2$ and $s_1 = s/g_1$.
- Replace $u(x) = x + 1/s(x)$ by

$$u = g_1 \tilde{s}_2/(g_2 s_1) \bmod g_{12}$$

- Deal with complication of computing $v = \sqrt{u} \bmod g_{12}$ for reducible g_{12}.
- Let half-gcd return A', B', put

$$\sigma(x) = (A' g_2)^2 + x(B' g_1)^2.$$

- Put $e = (00 \cdots 0)$.
- For j in $1, 2, \ldots, n$: if $\sigma(\alpha_j) = 0$ put $e = e + e_j$.

Extended Patterson decoder to handle $\gcd(g, s) \neq 1$

PALOMA uses extended Patterson decoder for reducible g.

- Let $\tilde{s} = 1 + xs$ and $g_1 = \gcd(g, s), g_2 = \gcd(g, \tilde{s}), g_{12} = g/(g_1 g_2)$.
- Compute $\tilde{s}_2 = \tilde{s}/g_2$ and $s_1 = s/g_1$.
- Replace $u(x) = x + 1/s(x)$ by

$$u = g_1 \tilde{s}_2/(g_2 s_1) \bmod g_{12}$$

- Deal with complication of computing $v = \sqrt{u} \bmod g_{12}$ for reducible g_{12}.
- Let half-gcd return A', B', put

$$\sigma(x) = (A' g_2)^2 + x(B' g_1)^2.$$

- Put $e = (00 \cdots 0)$.
- For j in $1, 2, \ldots, n$: if $\sigma(\alpha_j) = 0$ put $e = e + e_j$.
- $u = g_1 \tilde{s}_2/(g_2 s_1) \bmod g_{12}$ should fail for $s = s_1 = 0$ but doesn't.

Extended Patterson decoder to handle $\gcd(g, s) \neq 1$

PALOMA uses extended Patterson decoder for reducible g.

- Let $\tilde{s} = 1 + xs$ and $g_1 = \gcd(g, s), g_2 = \gcd(g, \tilde{s}), g_{12} = g/(g_1 g_2)$.
- Compute $\tilde{s}_2 = \tilde{s}/g_2$ and $s_1 = s/g_1$.
- Replace $u(x) = x + 1/s(x)$ by

$$u = g_1 \tilde{s}_2/(g_2 s_1) \bmod g_{12}$$

- Deal with complication of computing $v = \sqrt{u} \bmod g_{12}$ for reducible g_{12}.
- Let half-gcd return A', B', put

$$\sigma(x) = (A' g_2)^2 + x(B' g_1)^2.$$

- Put $e = (00 \cdots 0)$.
- For j in $1, 2, \ldots, n$: if $\sigma(\alpha_j) = 0$ put $e = e + e_j$.
- $u = g_1 \tilde{s}_2/(g_2 s_1) \bmod g_{12}$ should fail for $s = s_1 = 0$ but doesn't.
- Random polynomial has 0 roots in L with probability $\approx (1 - 1/q)^n$.

Extended Patterson decoder – another corner case

Remember, $s(x) = \sigma'(x)/\sigma(x)$.

Extended Patterson decoder – another corner case

Remember, $s(x) = \sigma'(x)/\sigma(x)$.
Single error at $\alpha_i = 0$ means $\sigma(x) = x \to s(x) \equiv 1/x \bmod g(x)$.

Extended Patterson decoder – another corner case

Remember, $s(x) = \sigma'(x)/\sigma(x)$.
Single error at $\alpha_i = 0$ means $\sigma(x) = x \to s(x) \equiv 1/x \bmod g(x)$.

- Let $\tilde{s} = 1 + xs$ and $g_1 = \gcd(g, s), g_2 = \gcd(g, \tilde{s}), g_{12} = g/(g_1 g_2)$.

Extended Patterson decoder – another corner case

Remember, $s(x) = \sigma'(x)/\sigma(x)$.
Single error at $\alpha_i = 0$ means $\sigma(x) = x \to s(x) \equiv 1/x \bmod g(x)$.

- Let $\tilde{s} = 1 + xs$ and $g_1 = \gcd(g, s), g_2 = \gcd(g, \tilde{s}), g_{12} = g/(g_1 g_2)$.
  This means, $\tilde{s} = 0, g_1 = 1, g_2 = g, g_{12} = 1$.
- Compute $\tilde{s}_2 = \tilde{s}/g_2$ and $s_1 = s/g_1$.
- Replace $u(x) = x + 1/s(x)$ by

$$u = g_1 \tilde{s}_2/(g_2 s_1) \bmod g_{12}$$

  Here $g_{12} = 1$, so $u = 0$
- Algorithm 5 returns $(1, 0)$ (should be $(0, 1)$), hence

$$\sigma(x) = (A' g_2)^2 + x(B' g_1)^2 = g_2^2 = g^2$$

  which has no roots in L (by definition of Goppa codes).
- Returns $e = (00 \cdots 0)$. This is an example for $\deg(\sigma) > t$.

PALOMA encapsulation v1.1

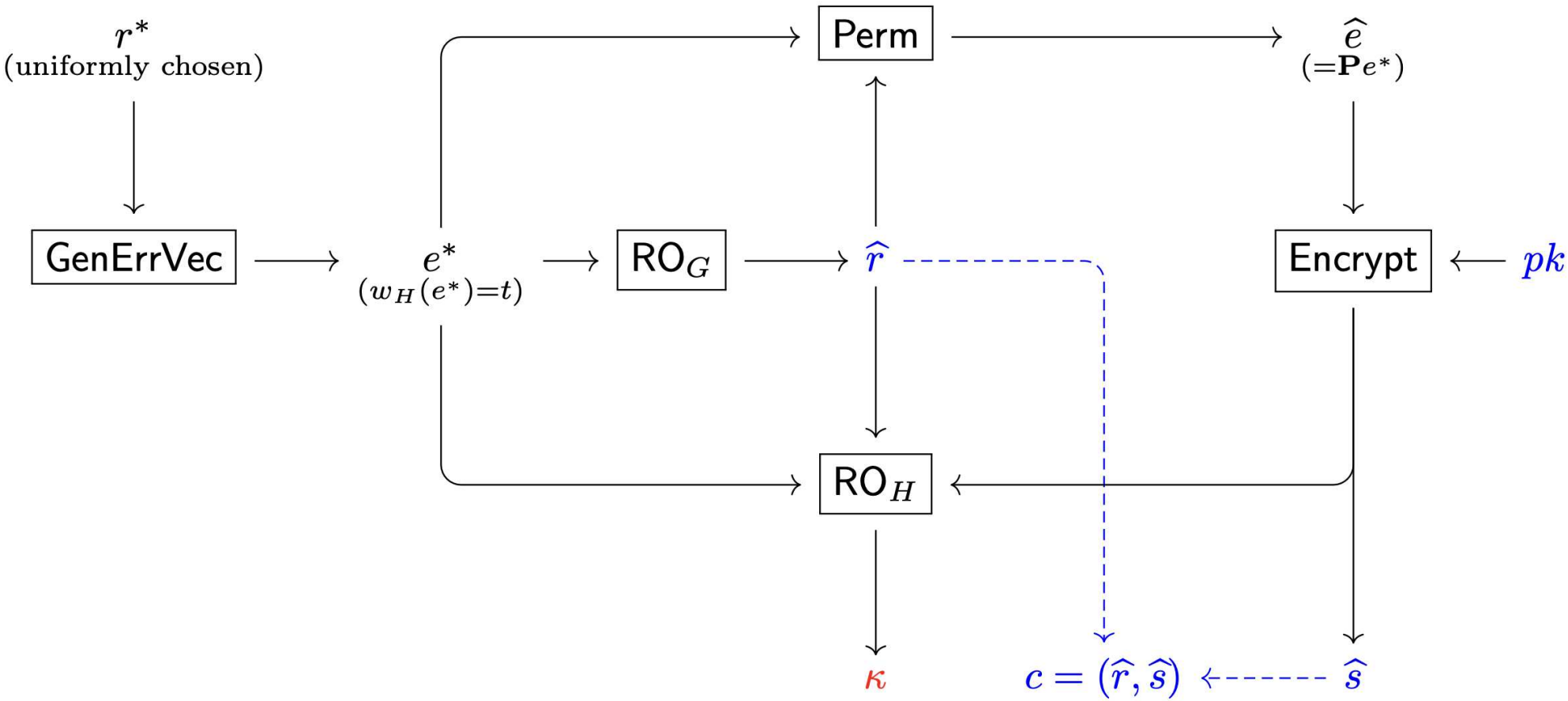


Image credit: PALOMA Team

(a) $\kappa, c = (\widehat{r}, \widehat{s}) \leftarrow \mathsf{Encap}(pk)$

PALOMA decapsulation v1.1

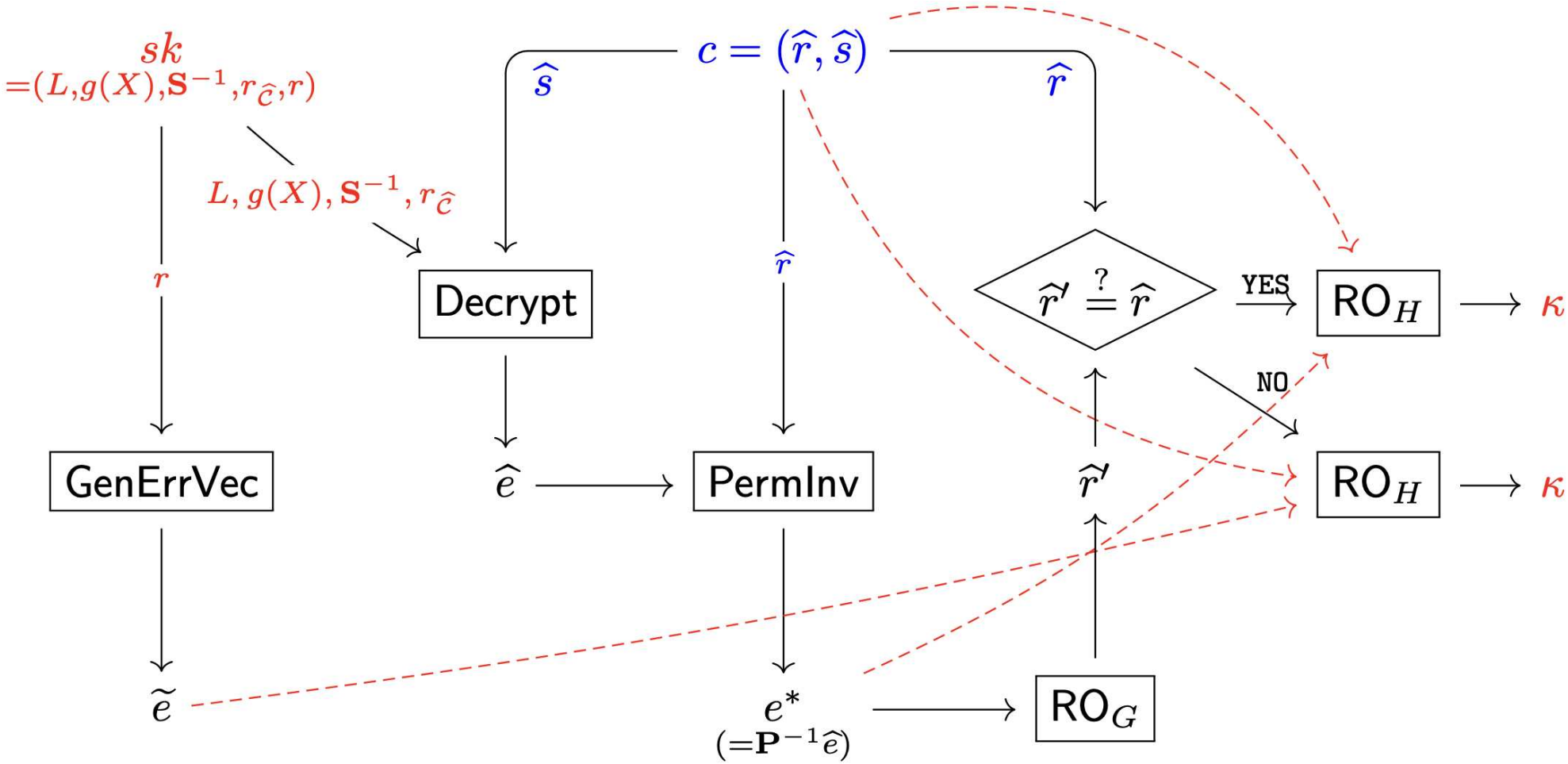


Image credit: PALOMA Team

(b) $\kappa \leftarrow \mathsf{Decap}(sk; c = (\widehat{r}, \widehat{s}))$

Reaction attacks on v1.1

- Remember "Sloppy Alice" / "reaction" attacks?
- Send specially crafted ciphertext, watch for reaction.

Reaction attacks on v1.1

- Remember "Sloppy Alice" / "reaction" attacks?
- Send specially crafted ciphertext, watch for reaction.
- Learn encrypted message (for codes) or key (lattices) from adaptive queries.

Reaction attacks on v1.1

- Remember "Sloppy Alice" / "reaction" attacks?
- Send specially crafted ciphertext, watch for reaction.
- Learn encrypted message (for codes) or key (lattices) from adaptive queries.
- Goppa decoder decodes **up to and including** t errors, fails for more.
- Change $\hat{s} = \hat{H}\hat{e}$ to $s' = \hat{s} + h_1$, where h_1 is the first column of $\hat{H}$.

Reaction attacks on v1.1

- Remember "Sloppy Alice" / "reaction" attacks?
- Send specially crafted ciphertext, watch for reaction.
- Learn encrypted message (for codes) or key (lattices) from adaptive queries.
- Goppa decoder decodes **up to and including** t errors, fails for more.
- Change $\hat{s} = \hat{H}\hat{e}$ to $s' = \hat{s} + h_1$, where h_1 is the first column of $\hat{H}$.
- s' is encryption of $e' = \hat{e} + e_1$.

Reaction attacks on v1.1

- Remember "Sloppy Alice" / "reaction" attacks?
- Send specially crafted ciphertext, watch for reaction.
- Learn encrypted message (for codes) or key (lattices) from adaptive queries.
- Goppa decoder decodes **up to and including** t errors, fails for more.
- Change $\hat{s} = \hat{H}\hat{e}$ to $s' = \hat{s} + h_j$, where h_j is the j-th column of $\hat{H}$.
- s' is encryption of $e' = \hat{e} + e_j$.

Reaction attacks on v1.1

- Remember "Sloppy Alice" / "reaction" attacks?
- Send specially crafted ciphertext, watch for reaction.
- Learn encrypted message (for codes) or key (lattices) from adaptive queries.
- Goppa decoder decodes **up to and including** t errors, fails for more.
- Change $\hat{s} = \hat{H}\hat{e}$ to $s' = \hat{s} + h_j$, where h_j is the j-th column of $\hat{H}$.
- s' is encryption of $e' = \hat{e} + e_j$.
- Decryption works iff e' has weight $\leq t$, i.e., if position j flipped from 1 to 0.

Reaction attacks on v1.1

- Remember "Sloppy Alice" / "reaction" attacks?
- Send specially crafted ciphertext, watch for reaction.
- Learn encrypted message (for codes) or key (lattices) from adaptive queries.
- Goppa decoder decodes **up to and including** t errors, fails for more.
- Change $\hat{s} = \hat{H}\hat{e}$ to $s' = \hat{s} + h_j$, where h_j is the j-th column of $\hat{H}$.
- s' is encryption of $e' = \hat{e} + e_j$.
- Decryption works iff e' has weight $\leq t$, i.e., if position j flipped from 1 to 0.
- Learn $\hat{e}$ in at most $n - 1$ steps.

Reaction attacks on v1.1

- Remember "Sloppy Alice" / "reaction" attacks?
- Send specially crafted ciphertext, watch for reaction.
- Learn encrypted message (for codes) or key (lattices) from adaptive queries.
- Goppa decoder decodes **up to and including** t errors, fails for more.
- Change $\hat{s} = \hat{H}\hat{e}$ to $s' = \hat{s} + h_j$, where h_j is the j-th column of $\hat{H}$.
- s' is encryption of $e' = \hat{e} + e_j$.
- Decryption works iff e' has weight $\leq t$, i.e., if position j flipped from 1 to 0.
- Learn $\hat{e}$ in at most $n - 1$ steps.
- Our first attack with Alex Pellegrini from 2024 against the PALOMA software used the reaction that some decryption attempts crashed the program.

PALOMA decapsulation v1.1

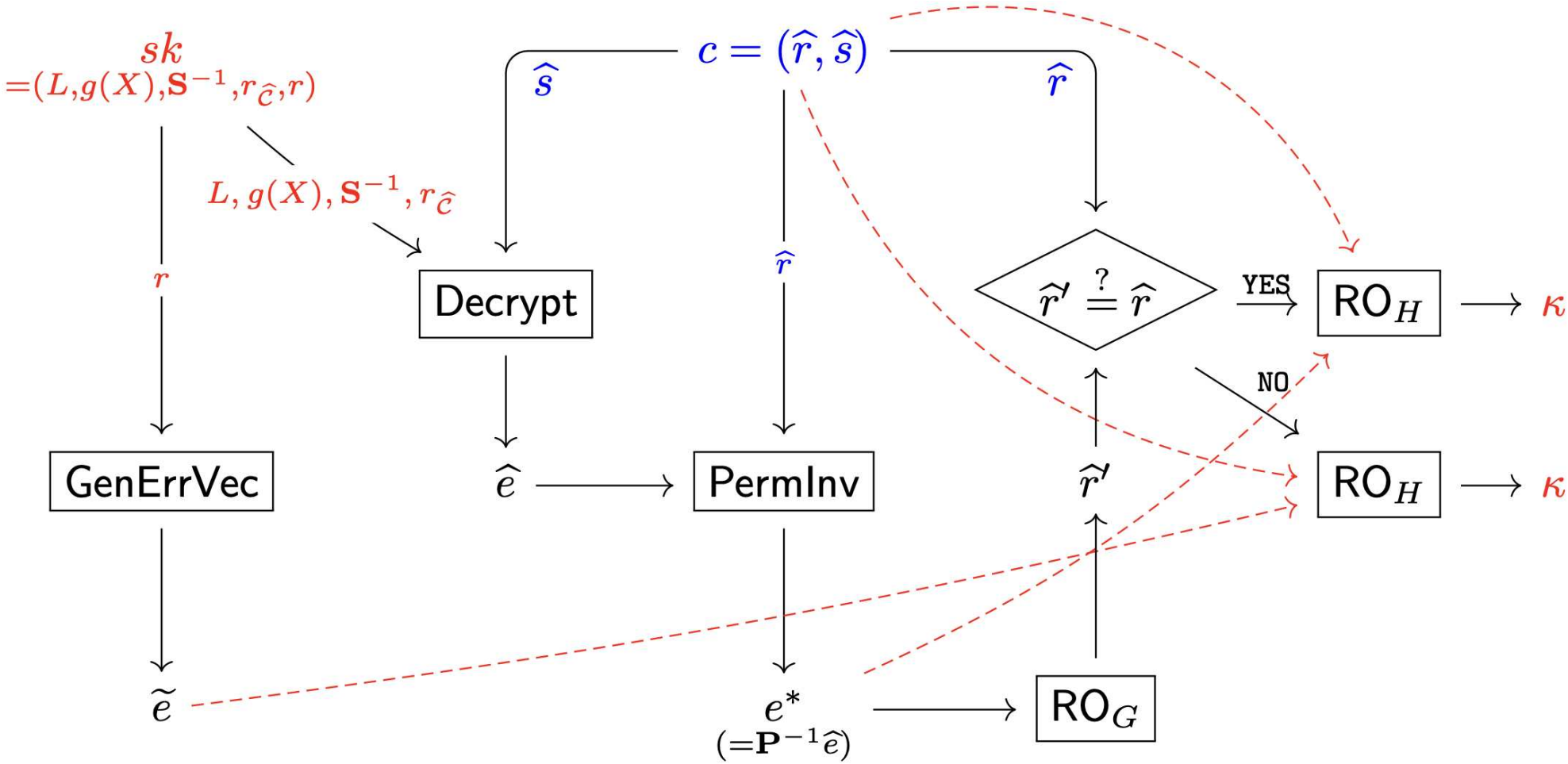


Image credit: PALOMA Team

(b) $\kappa \leftarrow \mathsf{Decap}(sk; c=(\widehat{r},\widehat{s}))$

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.
- Observation: Goppa decoder often returns $(00\cdots0)$ on random input.

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.
- Observation: Goppa decoder often returns $(00\cdots0)$ on random input.
- Any permutation of $(00\cdots0)$ remains $(00\cdots0)$.

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.
- Observation: Goppa decoder often returns $(00\cdots0)$ on random input.
- Any permutation of $(00\cdots0)$ remains $(00\cdots0)$.
- $\kappa = \mathrm{RO}_H(e^*, \hat{r}, \hat{s})$ with $\hat{r} = \mathrm{RO}_G(e^*)$ is computable for guessed e^*.

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.
- Observation: Goppa decoder often returns $(00\cdots0)$ on random input.
- Any permutation of $(00\cdots0)$ remains $(00\cdots0)$.
- $\kappa = \mathrm{RO}_H(e^*, \hat{r}, \hat{s})$ with $\hat{r} = \mathrm{RO}_G(e^*)$ is computable for guessed e^*.
- Let $e = (00\cdots0)$, $r = \mathrm{RO}_G(e)$.
- For j in $0, 1, 2, ..., n$
  - if decapsulation of $(r, \hat{s} + h_j)$ returns $\mathrm{RO}_H(e, r, \hat{s} + h_j)$ then we know $\hat{e}_j = 0$.
- Now know $\hat{e}_j = 0$ for 40 – 70% of all j.

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.
- Observation: Goppa decoder often returns $(00\cdots0)$ on random input.
- Any permutation of $(00\cdots0)$ remains $(00\cdots0)$.
- $\kappa = \mathrm{RO}_H(e^*, \hat{r}, \hat{s})$ with $\hat{r} = \mathrm{RO}_G(e^*)$ is computable for guessed e^*.
- Let $e = (00\cdots0)$, $r = \mathrm{RO}_G(e)$.
- For j in $0, 1, 2, ..., n$
  - if decapsulation of $(r, \hat{s} + h_j)$ returns $\mathrm{RO}_H(e, r, \hat{s} + h_j)$ then we know $\hat{e}_j = 0$.
- Now know $\hat{e}_j = 0$ for 40 – 70% of all j.
- $\hat{s} + h_i + h_j$ matches e' of weight t iff exactly one of positions i and j is 1.

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.
- Observation: Goppa decoder often returns $(00\cdots0)$ on random input.
- Any permutation of $(00\cdots0)$ remains $(00\cdots0)$.
- $\kappa = \mathrm{RO}_H(e^*, \hat{r}, \hat{s})$ with $\hat{r} = \mathrm{RO}_G(e^*)$ is computable for guessed e^*.
- Let $e = (00\cdots0)$, $r = \mathrm{RO}_G(e)$.
- For j in $0, 1, 2, ..., n$
  - if decapsulation of $(r, \hat{s} + h_j)$ returns $\mathrm{RO}_H(e, r, \hat{s} + h_j)$ then we know $\hat{e}_j = 0$.
- Now know $\hat{e}_j = 0$ for 40 – 70% of all j.
- $\hat{s} + h_i + h_j$ matches e' of weight t iff exactly one of positions i and j is 1.
- Use pairs of columns to identify all positions in original $\hat{e}$.
  Obtain e^* using $\hat{r}$.

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.
- Observation: Goppa decoder often returns $(00\cdots0)$ on random input.
- Any permutation of $(00\cdots0)$ remains $(00\cdots0)$.
- $\kappa = \mathrm{RO}_H(e^*, \hat{r}, \hat{s})$ with $\hat{r} = \mathrm{RO}_G(e^*)$ is computable for guessed e^*.
- Let $e = (00\cdots0)$, $r = \mathrm{RO}_G(e)$.
- For j in $0, 1, 2, ..., n$
  - if decapsulation of $(r, \hat{s} + h_j)$ returns $\mathrm{RO}_H(e, r, \hat{s} + h_j)$ then we know $\hat{e}_j = 0$.
- Now know $\hat{e}_j = 0$ for 40 – 70% of all j.
- $\hat{s} + h_i + h_j$ matches e' of weight t iff exactly one of positions i and j is 1.
- Use pairs of columns to identify all positions in original $\hat{e}$.
  Obtain e^* using $\hat{r}$. Our attack software takes 0.5 – 7 minutes.

CCA attack on PALOMA v1.1

- Attack seems to be stopped: we don't know e' and $\hat{r}$ is obtained from e^* by one-way function RO_G.
- Observation: Goppa decoder often returns $(00\cdots0)$ on random input.
- Any permutation of $(00\cdots0)$ remains $(00\cdots0)$.
- $\kappa = \mathrm{RO}_H(e^*, \hat{r}, \hat{s})$ with $\hat{r} = \mathrm{RO}_G(e^*)$ is computable for guessed e^*.
- Let $e = (00\cdots0)$, $r = \mathrm{RO}_G(e)$.
- For j in $0, 1, 2, ..., n$
  - if decapsulation of $(r, \hat{s} + h_j)$ returns $\mathrm{RO}_H(e, r, \hat{s} + h_j)$ then we know $\hat{e}_j = 0$.
- Now know $\hat{e}_j = 0$ for 40 – 70% of all j.
- $\hat{s} + h_i + h_j$ matches e' of weight t iff exactly one of positions i and j is 1.
- Use pairs of columns to identify all positions in original $\hat{e}$. Obtain e^* using $\hat{r}$. Our attack software takes 0.5 – 7 minutes.
- Same recovery as with Pellegrini. New: **valid** fake ciphertexts, predicting κ.

Partial key recovery attack

- Goppa codes can efficiently correct up to t errors.
- Let $(\alpha'_1, \alpha'_2, \ldots, \alpha'_n) = P'L$.
- Observation: Decoder used in PALOMA has exception:

$$He_j \text{ decodes to } (00\cdots0) \text{ iff } \alpha'_j = 0.$$

Partial key recovery attack

- Goppa codes can efficiently correct up to t errors.
- Let $(\alpha'_1, \alpha'_2, \ldots, \alpha'_n) = P'L$.
- Observation: Decoder used in PALOMA has exception:

$$He_j \text{ decodes to } (00\cdots0) \text{ iff } \alpha'_j = 0.$$

- Let $e = (00\cdots0), r = \mathrm{RO}_G(e)$.
- Attack algorithm:
  1. For j in $1, 2, 3, ..., n$:
     - If decapsulation of (r, h_j) returns $\mathrm{RO}_H(e, r, h_j)$: return "$\alpha'_j = 0$".
  2. Return "0 is not in support".
- This takes at most n steps and will find the position of 0 if included.

PALOMA encapsulation v1.2

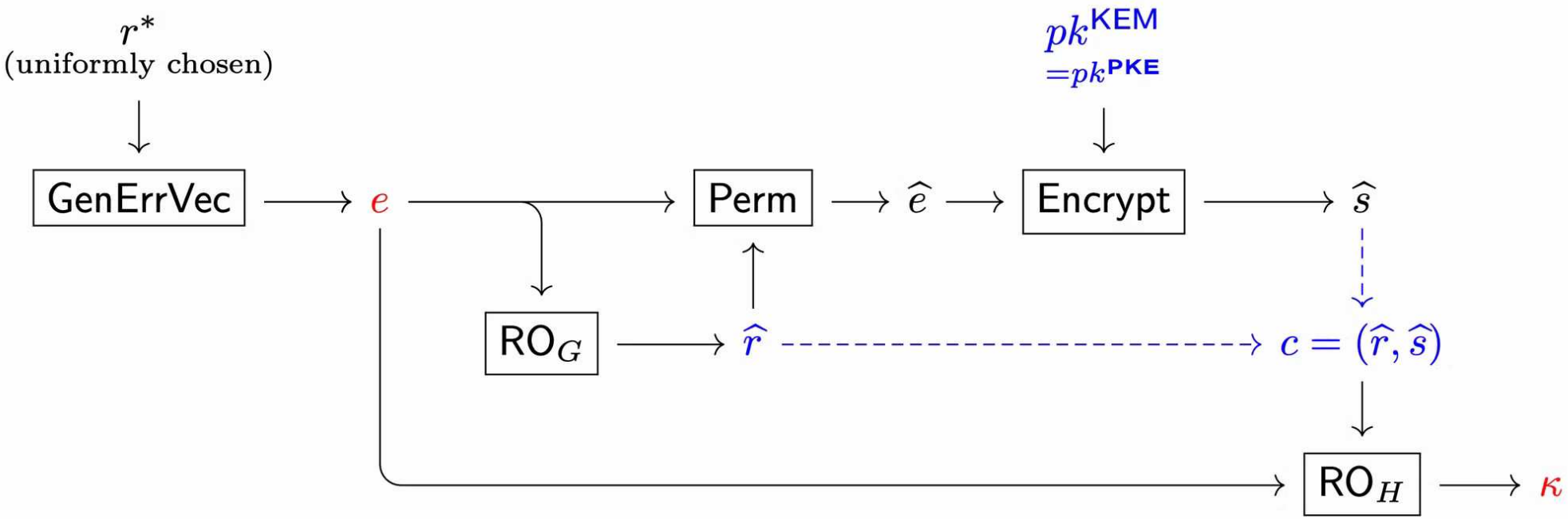


Image credit: PALOMA Team $\quad$ (a) $c = (\widehat{r}, \widehat{s}), \kappa \leftarrow \mathsf{Encap}(pk^{\mathsf{KEM}})$

PALOMA decapsulation v1.2

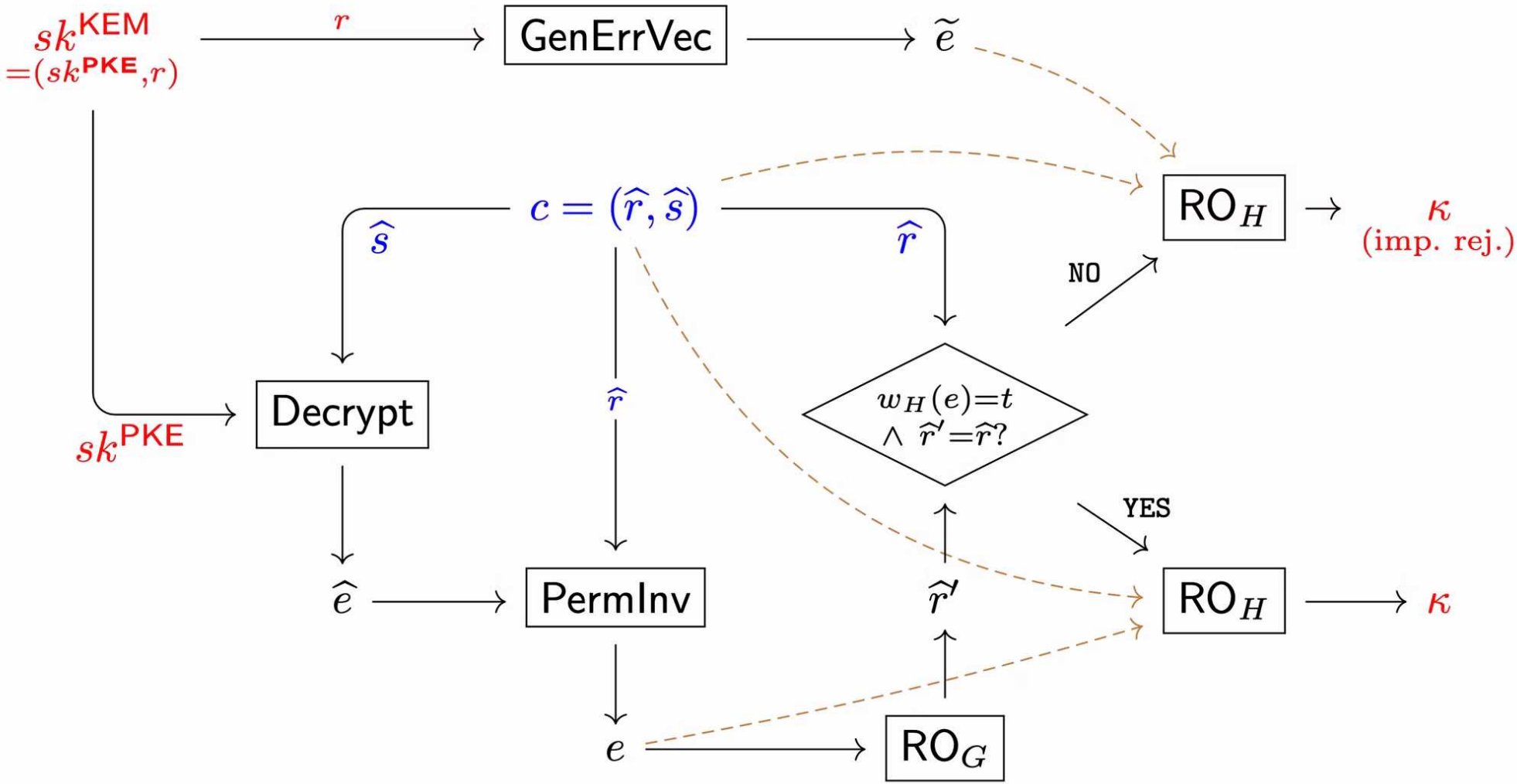


Image credit: PALOMA Team

(b) $\kappa \leftarrow \mathsf{Decap}(sk^{\mathsf{KEM}};\ c=(\widehat{r},\widehat{s}))$

Lesson: Look at the whole attack surface

- These are not the same problem:
  - Breaking one-wayness of McEliece's original system.
  - Breaking mathematical security of a cryptosystem.
  - Breaking software for a cryptosystem.

Lesson: Look at the whole attack surface

- These are not the same problem:
  - Breaking one-wayness of McEliece's original system.
  - Breaking mathematical security of a cryptosystem.
  - Breaking software for a cryptosystem.
- First problem has extensive analysis and a stable security level, but attacker is also happy breaking second or third problem. Any minor difference could allow a devastating attack.

Lesson: Look at the whole attack surface

- These are not the same problem:
  - Breaking one-wayness of McEliece's original system.
  - Breaking mathematical security of a cryptosystem.
  - Breaking software for a cryptosystem.
- First problem has extensive analysis and a stable security level, but attacker is also happy breaking second or third problem. Any minor difference could allow a devastating attack.
- If a cryptosystem or software claims "provable security":
  - Look for gaps between the theorem and security.
  - Look for gaps in the proof. The "theorem" could be wrong.
  - Or try breaking the system anyway. This often works!