Low weight polynomials in crypto

Thomas Johansson

Dept of EIT,
Lund University,
Sweden

FSE 2014

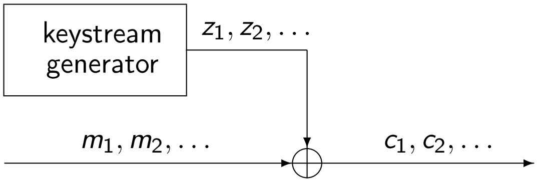
Contents

- PART I: Applications of low weight polynomials in crypto
  1. Fast correlation attacks (cryptanalysis)
  2. TCHo(design)
  3. MDPC (design)
- PART II: How to find a low weight multiple of a polynomial
  1. Weight 3,4,5 and finding all existing multiples
  2. Larger weight and finding all existing multiples

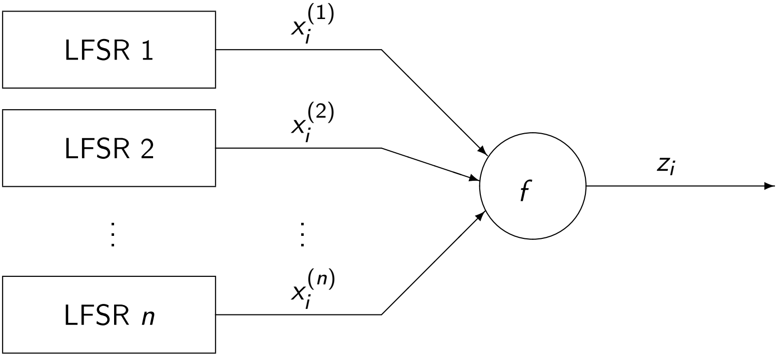
Problem: Low-Weight Polynomial Multiple (LWPM)

Given a polynomial $P(x) \in \mathbb{F}_2[x]$ of degree d_P.

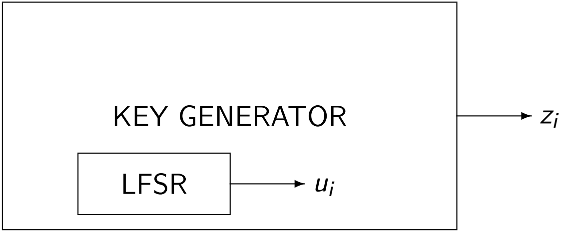
Find all multiples of $P(x)$ of degree $\leq d$ (if such exists) with w nonzero coefficients.



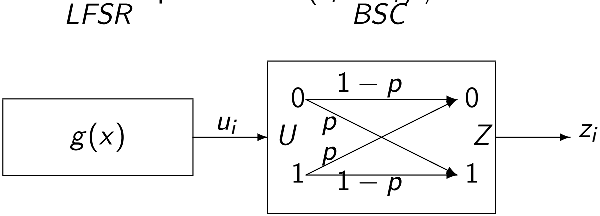
- The keystream generator contains one or several LFSRs.
- Observed keystream sequence $z_1, z_2, \ldots, z_N$.



A nonlinear combining generator



- A correlation attack is possible if $P(z_i = u_i) \neq 0.5$.



Meier-Staffelbach original approach

- The feedback polynomial

$$g(x) = 1 + g_1 x + g_2 x^2 + \ldots + x^l.$$

- Recurrence relation

$$u_n = g_1 u_{n-1} + g_2 u_{n-2} + \ldots + u_{n-l}.$$

- Assume a *low weight* of $g(x)$, weight w.
- We get in this way w different low weight parity check equations for u_n.

Finding more low weight parity checks

- Any multiple of $g(x)$ gives a recurrence relation.
- Use $g(x)^j = g(x^j)$ for $j = 2^i$,
- Create new polynomials by

$$g_{k+1}(x) = g_k(x)^2, \quad k = 1, 2, \ldots.$$

- This squaring is continued until the degree of $g_k(x)$ is greater than the length N of the observed keystream.
- Each $g_k(x)$ is of weight w and hence each gives w new parity check equations for a fixed position u_n.

A simple distinguisher

- $z_n = u_n + e_n, \; n = 1, 2, \ldots$.
- $\Pr(e_n = 0) = 1 - p = \frac{1}{2}(1 + \epsilon)$.
- Recurrence relations of weight w,

$$u_n + g_1 u_{n-1} + g_2 u_{n-2} + \ldots + u_{n-l} = 0.$$

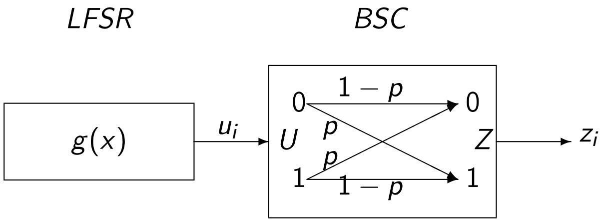
- Form

$$S_n = z_n + g_1 z_{n-1} + g_2 z_{n-2} + \ldots + z_{n-l}.$$

- Verify that

$$P(S_n = 0) = P(e_n + g_1 e_{n-1} + g_2 e_{n-2} + \ldots + e_{n-l} = 0) = 1/2 + \epsilon^w.$$

- Collect $1/\epsilon^{2w}$ such samples to distinguish $z_1, z_2, \ldots, z_N$ from a random sequence.



- General case: $g(x)$ is not of low weight.
- How can we attack in this case?
  One answer: Find a low weight multiple of $g(x)$.
- How do we find a multiple of $g(x)$ of weight $3, 4, 5$?
- Example of an instance: If length of LFSR=90, length of received sequence $N = 2^{33}$, what is the cost of finding a weight $w = 4$ multiple of $g(x)$?

- TCHo is a public-key cryptosystem based on the low weight polynomial multiple problem (Aumasson, Finiasz, Meier, Vaudenay, 2006-2007).
- Public key: polynomial $P(x)$,
- Secret key: a multiple $K(x) = q(x)P(x)$, where $w_H(K(x)) = w$ is low.

- $\mathbf{G}_{\text{rep}}$, generator matrix of a repetition code of length n.
- Plaintext $\mathbf{m} \in \mathbb{F}_2^k$.
- Generate a random string $\mathbf{r} = \begin{bmatrix} r_0 & r_1 & \cdots & r_{n-1} \end{bmatrix}$ with bias $\mathbf{Pr}[r_i = 0] = \frac{1}{2}(1+\gamma)$.
- Generate an LFSR sequence $\mathbf{p}$ with feedback polynomial $P(x)$ and a random starting state.

Ciphertext generated as

$$\mathbf{c} = \mathbf{m}\mathbf{G}_{\text{rep}} + \mathbf{r} + \mathbf{p}.$$

$$\mathbf{M} = \begin{bmatrix} k_0 & k_1 & \cdots & k_{d_K} & & & & \\ & k_0 & k_1 & \cdots & k_{d_K} & & & \\ & & \ddots & \ddots & & & \ddots & \\ & & & k_0 & & k_1 & \cdots & k_{d_K} \end{bmatrix}.$$

$P(x)$ divides $K(x)$, so $\mathbf{pM}^\mathsf{T} = \mathbf{0}$.
Compute $\mathbf{t} = \mathbf{cM}^\mathsf{T}$.

$\mathbf{t} = (\mathbf{mG}_{\text{rep}} + \mathbf{r} + \mathbf{p})\mathbf{M}^\mathsf{T} = \mathbf{mG}_{\text{rep}}\mathbf{M}^\mathsf{T} + \mathbf{rM}^\mathsf{T} + \mathbf{pM}^\mathsf{T} = \mathbf{mG}_{\text{rep}}\mathbf{M}^\mathsf{T} + \mathbf{rM}^\mathsf{T}$.

Each bit in $\mathbf{r}$ was γ-biased. $K(x)$ has weight w and consequently, each element in $\mathbf{rM}^\mathsf{T}$ will be γ^w-biased.
Majority decision decoding can be used to decode

$$\mathbf{t} = \mathbf{m}\left(\mathbf{G}_{\text{rep}}\mathbf{M}^\mathsf{T}\right) + \mathbf{rM}^\mathsf{T}.$$

Example of an instance:

- $K(x)$ of degree $d_K = 44677$ and weight $w = 25$,
- Known polynomial $P(x)$ of degree $d_P = 4433$.
- How do we find a weight 25 multiple of $P(x)$ of degree 44677?

- Public-key cryptosystem (Misoczki, Tillich, Sendrier, Barreto)
- Secret key:

$$H = \begin{pmatrix} H_0 & H_1 & \cdots & H_{n_0-1} \end{pmatrix},$$

  where each H_i is a circulant $r \times r$ matrix with weight w_i in each row and with $w = \sum w_i$.
- Public key:

$$G = \begin{pmatrix} I & P \end{pmatrix},$$

  where

$$P = \begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_{n_0-2} \end{pmatrix} = \begin{pmatrix} \left(H_{n_0-1}^{-1} H_0\right)^T \\ \left(H_{n_0-1}^{-1} H_1\right)^T \\ \vdots \\ \left(H_{n_0-1}^{-1} H_{n_0-2}\right)^T \end{pmatrix}.$$

The McEliece PKC using QC-MDPC codes

- $\mathbf{m} \in \mathbb{F}_2^{(n-r)}$ plaintext.
  Multiply $\mathbf{m}$ with the public key G and add errors within the correction radius t of the code, i.e.,

  $$\mathbf{c} = \mathbf{m}G + \mathbf{e},$$

  where $w_H(\mathbf{e}) \leq t$.
- Decoding: Given the secret low-weight parity check matrix H, a low-complexity decoding procedure is used to obtain the plaintext $\mathbf{m}$.

- The scheme can be rewritten in polynomial form
- For $n_0 = 2$: Let $h_1(x)$ represent H_1 and $h_0(x)$ represent H_0.
- Known P_0 is represented by $P(x)$, we have

$$h_1(x)P(x) \equiv h_0(x) \bmod (x^r + 1). \tag{1}$$

Example of an instance:

- r =degree of $h_i(x) = 4801$. Weight $w_H(h_0(x)) = w_H(h_1(x)) = 45$.
- For given $P(x)$ find h_0 and h_1 such that

$$h_1(x)P(x) \equiv h_0(x) \bmod (x^{4801} + 1).$$

II.1 Algorithms for finding low weight polynomial multiples

- Many different approaches have been given.
- We are looking for multiples of the type

$$q(x)P(x) = 1 + x^{i_1} + \ldots + x^{i_{w-1}},$$

where $i_j \leq N$.

- When the algorithm finds expressions like

$$x^{i'_0} + x^{i'_1} + \ldots + x^{i'_{w-1}}$$

it can be shifted to produce a multiple of the desired form.

- $d_P = l$
- With $a, b, c, d \leq 2^{l/4}$, create $2^{l/2}$ polynomials $x^a + x^b \mod P(x)$, and equally many $x^c + x^d \mod P(x)$. From the birthday paradox, collisions between the lists is expected, yielding $g(x) | (x^a + x^b + x^c + x^d)$.
- Golić pointed out that a collision $x^a + x^b = x^c + x^d \pmod{P(x)}$ also yields $x^{a+\gamma} + x^{b+\gamma} + x^{c+\gamma} + x^{d+\gamma} = 0 \pmod{P(x)}$ for all $\gamma > 0$, thus creating additional collisions. But the birthday paradox does not suggest this many collisions.
- For random polynomials, multiples of weight w start showing up at degrees around $\alpha_t \cdot 2^{l/(w-1)}$, where $\alpha_t \approx 1$.

Golić formulated an algorithm that searches for checks of weights $2v$ and $2v+1$

- Create a list of the $\binom{N}{v}$ residues $x^{i_1} + \ldots + x^{i_v} \bmod P(x)$.
- Sort and look for 0-matches and 1-matches, i.e.,

$$(x^{i_1^1} + \ldots + x^{i_v^1}) + (x^{i_1^2} + \ldots + x^{i_v^2}) = b \pmod{P(x)},$$

  giving rise to a multiple of weight at most $2v + b$.
- This algorithm requires time and memory about $\binom{N}{v}$.
- If $w = 2v = 4$ then we need time and memory about $2^{2l/3}$.

- Penzhorn and Kühn
- Create $\mathbb{F}_{2^l}$ using $P(x)$. Use Zech's logarithm defined from a primitive element $\alpha \in \mathbb{F}_{2^l}$.
- Zech's logarithm $z(i)$ is defined through

$$\alpha^{z(i)} = \alpha^i + 1.$$

- Multiples of weight 3 can be found by observing that $x^{z(i)} + x^i + 1$ is a multiple of $g(x)$. Therefore, logarithms $z(i)$ for $i = 1, 2, \ldots, T$ are computed until $z(i) \leq N$ is found.
- Logarithms can be computed rather efficiently, using e.g. a method by Coppersmith. Aiming at an overall success probability of $1 - e^{-1}$, one might e.g., use $N = 2^{l/2}$, $T = 2^{l/2}$.

- Multiples of weight 4 can be found by observing that if (i, j) are found such that $z(i) - z(j) = \delta > 0$, then

$$x^{z(i)} + x^i + 1 + x^\delta(x^{z(j)} + x^j + 1) = x^i + x^{j+\delta} + x^\delta + 1$$

  because $x^{z(i)} = x^{\delta + z(j)}$. Aiming at $N = 2^{l/3}$ gives $T = 2^{l/3}$, which compares favourably to previous methods.
- Computational complexity: the number of discrete logarithms that must be computed is $2^{l/3}$. Table size $T = 2^{l/3}$.

1. Create all $x^{i_1} \bmod P(x)$, for $0 \le i_1 < 2^{d_P/3+\alpha}$ and put $(x^{i_1} \bmod P(x), i_1)$ in a table T_1. Sort T_1 according to the residue value.
2. Create all $x^{i_1} + x^{i_2} \bmod P(x)$ such that $\mathrm{lsb}_{d_P/3}(x^{i_1} + x^{i_2} \bmod P(x)) = 0$, for $0 \le i_1 < i_2 < 2^{d_P/3+\alpha}$ and put in a table T_2. Here $\mathrm{lsb}_{d_P/3}()$ means the $d_P/3$ least significant bits.
   This is done by merging the sorted table T_1 by itself and keeping only residues $x^{i_1} + x^{i_2} \bmod P(x)$ with the last $d_P/3$ bits all zero. The table T_2 is sorted according to the residue value.
3. Merge the sorted table T_2 with itself keeping only residues $x^{i_1} + x^{i_2} + x^{i_3} + x^{i_4} = 0 \bmod P(x)$. Output these weight 4 multiples.

Complexity analysis

- Assume $K(x)$ is the multiple of lowest degree, around $d_P/3$.
- The algorithm creates all weight 4 multiples up to degree $2^{d_P/3+\alpha}$, that include two monomials x^{i_1} and x^{i_2} such that $\mathrm{lsb}_{d_P/3}(x^{i_1}+x^{i_2} \bmod P(x)) = 0$.
- Any polynomial $x^{i_1}K(x)$ is of weight 4. Since we consider all weight 4 multiples up to degree $2^{d_P/3+\alpha}$ we will consider $2^{d_P/3+\alpha} - 2^{d_P/3}$ such weight 4 polynomials, i.e. about $2^{d_P/3}(2^\alpha - 1)$ duplicates of $K(x)$.
- As the probability for a single weight 4 polynomial to have the condition $\mathrm{lsb}_{d_P/2}(x^{i_1}+x^{i_2} \bmod P(x)) = 0$ can be approximated to be around $2^{-d_P/3}$, there will be a large probability that at least one duplicate $x^{i_1}K(x)$ will survive in Step 2 in the above algorithm and will be included in the output.

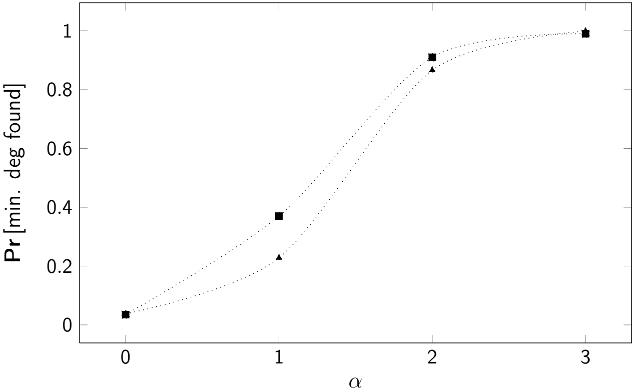


Figure: The probability of finding the minimum degree polynomial multiple as a function of algorithm parameter α.

Example, weight 4

Finding the weight 4 multiple with lowest degree

- For $d_P = 90, N = 2^{30}$, the complexity of the classical approach is 2^{60}.
- or solving 2^{30} discrete log instances in $\mathbb{F}_{2^{90}}$.
- Proposed algorithm with $\alpha = 3$ yields complexity around 2^{33}, with very low probability of not finding the lowest degree polynomial multiple.

Wagner's Generalized Birthday Problem

- One of several applications
- Each list L_j is populated with elements $x^i \bmod P(x)$. Finding a set of $v_j \in L_j$, where $v_j = x^{i_j} \bmod P(x)$, such that $v_1 + \ldots + v_t = 1$ yields the multiple $x^{i_1} + \ldots + x^{i_t} + 1$.
- Problem size $t = 2^x$: reducing the problem by joining pairs of lists fixing the s least significant bits. Repeat again for remaining lists and fixing the next least significant bits, etc.
- One needs $N \approx 2^{d_P/(1+\log t)}$ to expect to find a multiple. The weight will be $t+1$, the degree will be about $2^{d_P/(1+\lfloor \log t \rfloor)}$ and the work about $t \cdot 2^{d_P/(1+\lfloor \log t \rfloor)}$.
- For $w = 5$ we will get a multiple of degree $2^{d_P/3}$ (first weight 5 multiple will appear around degree $2^{d_P/4}$).

- What happens when w is a bit larger?
- Assume we know there is a low weight multiple of degree d.
- The problem can be turned into a coding theory problem.
- Finding a low weight multiple is the same as finding a low weight codeword in a certain code.
- Low weight codewords in a code can be found by decoding algorithms for general codes, in particular *information set decoding* (ISD) algorithms.

Problem: Subspace Weight (minimum weight codeword)

With $\mathbf{G}$ being a random $k \times n$ matrix find $\mathbf{u}$ in

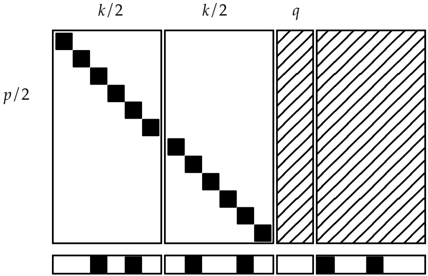
$$\mathbf{v} = \mathbf{u}\mathbf{G}$$

such that $w_H(\mathbf{v}) = w$.

- Decision problem is $\mathcal{NP}$-complete.

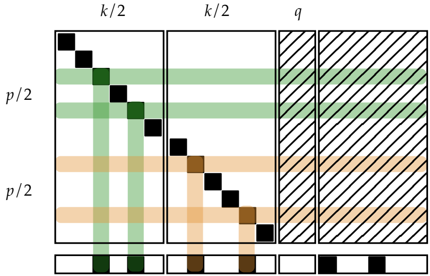
Given: a $k \times n$ matrix $\mathbf{G}$, p, q algorithm parameter

1. Pick a random column permutation π. Compute $\pi(\mathbf{G})$.
2. Make $\pi(\mathbf{G})$ systematic, forming $\hat{\mathbf{G}}$



$\pi(\mathbf{G})$ and $\hat{\mathbf{G}}$ represents the same code
$\Longrightarrow$ $\hat{\mathbf{v}}_{\min}$ remains the same.

3 (a) Create all sums $p/2$ of rows from the upper part of $\hat{\mathbf{G}}$ and put in a list L_1 indexed by q.
   (b) Equivalently, create all sums $p/2$ of rows from the lower part of $\hat{\mathbf{G}}$ and put in a list L_2 indexed by q.



4. Merge the two lists L_1 and L_2. A collision means that the q-field is all-zero.
5. If any vector has weight $w - p$ in the remaining positions, output it. If not, repeat 1. with a new permutation.

- The workfactor of one iteration in Stern's algorithm is given by

$$C = \frac{1}{2}(n-k)^2(n+k) + 2\binom{k/2}{p}pj + \binom{k/2}{p}^2 p(n-k)/2^{j-1}.$$

- q is the probability of success in one iteration.
- Total work factor: C/q

- Given the polynomial $P(x)$, we want to find a $u(x)$ such that $u(x) \cdot P(x) = K(x)$ where $K(x)$ has w nonzero coefficients.
-

$$
\begin{aligned}
K(x) = & \quad u(x) \cdot P(x) \\
= & \quad (u_0 + u_1 x + \cdots u_{d-d_P} x^{d-d_P}) \cdot (p_0 + p_1 x + p_{d_P} x^{d_P}) \\
= & \quad \begin{bmatrix} u_0 & u_1 & \cdots & u_{d-d_P} \end{bmatrix} \begin{bmatrix} P(x) \\ xP(x) \\ \vdots \\ x^{d-d_P} P(x) \end{bmatrix} = \mathbf{uG}(x)
\end{aligned}
$$

- We can represent

$$\mathbf{G}(x) = \begin{bmatrix} P(x) \\ xP(x) \\ \vdots \\ x^{d-d_P}P(x) \end{bmatrix}$$

as

$$\mathbf{G} = \begin{bmatrix} p_0 & p_1 & \cdots & p_{d_P} & & & & \\ & p_0 & p_1 & \cdots & p_{d_P} & & & \\ & & \ddots & \ddots & & & \ddots & \\ & & & p_0 & & p_1 & \cdots & p_{d_P} \end{bmatrix},$$

if each column is mapped to each degree of x.

- We can use ISD algorithms on $\mathbf{G}$.

$\mathbf{G}$ has dimension $k \times n$. The $[n, k]$-code generated by $\mathbf{G}$ has one single codeword of weight w, namely $\mathbf{K}$ (or $K(x)$).

Idea: The code is 'cyclic', so we can allow shifts of $K(x)$, i.e.

$$xK(x),\ x^2K(x), \ldots$$

By adding one row to $\mathbf{G}$,

$$\mathbf{G}' = \begin{bmatrix} \mathbf{G} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & \cdots & p_{d_P} & & & & \\ & p_0 & p_1 & \cdots & p_{d_P} & & & \\ & & \ddots & \ddots & & \ddots & & \\ & & & p_0 & p_1 & \cdots & p_{d_P} & \\ & & & & p_0 & p_1 & \cdots & p_{d_P} \end{bmatrix},$$

there are now two codewords of weight w.

G has dimensions $k \times n$. The $[n, k]$-code generated by **G** has one single codeword of weight w, namely **K** (or $K(x)$).

Idea: The code is cyclic, so we can allow shifts of $K(x)$, i.e.

$$xK(x),\ x^2K(x), \ldots$$

By adding one row to **G**,

$$\mathbf{G}' = \begin{bmatrix} \mathbf{G} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & \cdots & p_{d_P} & & & & \\ & p_0 & p_1 & \cdots & p_{d_P} & & & \\ & & \ddots & \ddots & & \ddots & & \\ & & & p_0 & p_1 & \cdots & p_{d_P} & \\ & & & & p_0 & p_1 & \cdots & p_{d_P} \end{bmatrix},$$

there are now two codewords of weight w.

What is the effect?

- ISD algorithms have a complexity that is $\sim \frac{1}{q}$, where q is the probability of success in one iteration.
- If q is small and success events are independent, then y low weight codewords means success prob. $\approx y \cdot q$.
- The dimension k of the code increases with y, but if $k >> y$ it has little effect on complexity.
- Complexity decreases with increasing y, i.e., $\frac{C}{y \cdot q}$.

We know that the polynomial $K(x)$ has the form:

$$1+\underbrace{\cdots\cdots\cdots\cdots\cdots\cdots\cdots}_{w-2 \text{ nonzero coefficients}}+x^d$$

How can we use that information?

- Should be able to search over $w-2$ unknowns rather than w.
- Less weight leads to lower complexity.

For any polynomial $P(x)$, there exists a linear map Γ that transforms the code $\mathcal{C}_y$ into a new code given by $\mathbf{G}_y\Gamma$, such that all weight w codewords corresponding to shifts of $K(x)$ will have weight $w-2$ in the new code.

The result is a $(k+y) \times (n-1)$ matrix

$$\mathbf{G}' = \begin{bmatrix} p_0 & p_1 & \cdots & p_{d_P} & & \\ & p_0 & & \vdots & \ddots & \\ & & \ddots & \vdots & & p_{d_P} \\ p_{d_P} & & & p_0 & \cdots & p_{d_P-1} \\ \vdots & \ddots & & & \ddots & \vdots \\ p_0 & \cdots & p_{d_P} & & & p_0 \end{bmatrix}$$

with weight $w-2$ codewords

$$\begin{bmatrix} K_1 \\ K_2 \\ \vdots \\ K_y \end{bmatrix} = \begin{bmatrix} 0 & k_1 & \cdots & \cdots & \cdots & k_{d-1} \\ k_{d-1} & 0 & k_1 & \cdots & \cdots & k_{d-2} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ k_{d-y} & \cdots & k_{d-1} & 0 & k_1 & k_2 \end{bmatrix}$$

The last step is to simply apply an ISD-algorithm on $\mathbf{G}'$.

1. Build a matrix $\mathbf{G}$ from $P(x)$ according to the reduction.
2. Expand with y shifts of $K(x)$.
3. Perform weight-reduction.
4. Apply ISD to find weight $w-2$ codeword

How well does it perform?

80-bit security (in terms of key-recovery):

d	d_P	w	ISD	New algorithm	*Gain*	y_{opt}
24730	12470	67	$2^{85.75}$	$2^{77.65}$	$2^{8.20}$	230
44677	4433	25	$2^{96.47}$	$2^{84.15}$	$2^{12.32}$	250

The numbers refer to bit operations.

Ideally, 2^6 bit operations per word operation ($2^{3.3}$ in toy example implementation).

Coming back to the related problem (QC-MDPC codes)

Example: degree of $h_i(x) = 4801$. $w_H(h_0(x)) = w_H(h_1(x)) = 45$.

For given $P(x)$ find h_0 and h_1 such that

$$h_1(x)P(x) \equiv h_0(x) \bmod (x^{4801} + 1).$$

- ISD algorithms can be used to solve this problem.
- We know some improved ways when degree of $h_i(x)$ is even.
- Can CRT give improvements?

- Many interesting problems around low weight multiples.
- New primitives could be based on such problems.

T. Johansson, C. Löndahl, ”Improved Algorithms for Finding Low-Weight Polynomial Multiples and some cryptographic applications”, to appear in *Designs, Codes and Cryptography.*