Vulnerabilities of *McEliece in the World of Escher*

Ray Perlner, Dustin Moody

National Institute of Standards and Technology

Gaithersburg, Maryland, USA

Ray.Perlner@NIST.gov; Dustin.Moody@NIST.gov

Outline

- "McEliece in the World of Escher"
  [Gligoroski, Samardjiska, Jacobsen, Bezzateev 2014]

  - A McEliece variant promising encryption and signatures

  - New ideas:
    - Error sets
    - List decoding

  - Encryption and signature schemes:
    - Private key
    - Decoding algorithm

- Major Results
  - ISD for the error vector (signature forgery)
  - ISD for the private key (key recovery for both encryption and signature)

- Countermeasures?

Commonalities with other McEliece Variants.

- Private key operation is a decoding problem
  - Generator Matrix form:

$$mG_{pub} + e = c \qquad\qquad sG_{pub} + e = H(m)$$

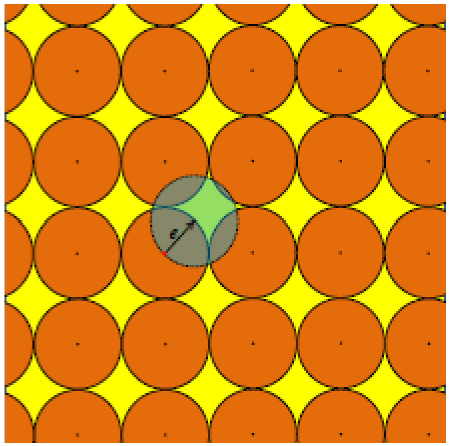
  - Parity Check Matrix form:

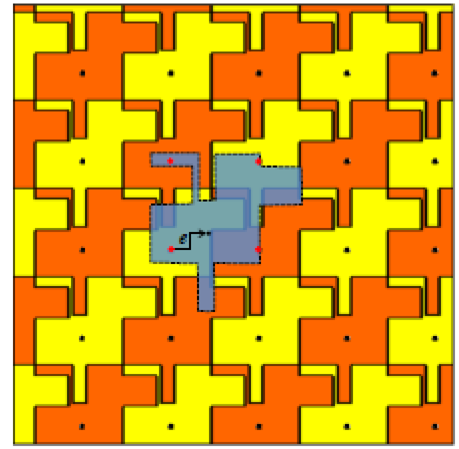
$$H_{pub}(c - e)^T = 0 \qquad\qquad H_{pub}(H(m) - e)^T = 0$$

- Public Generator/Parity Check matrix is disguised Private Generator/Parity Check matrix

$$G_{pub} = SGP \qquad\qquad H_{pub} = S'HP$$

Escher McEliece
Error Sets





- Standard Code-based crypto
  - Error vector is a biased sample from the 1-bit alphabet (0,1)
  - E.g. (00|10|11|00|00|10|00|01)
    - Mean: 0.33
    - **44% 00; 22% 01; 22% 10; 11% 11**

- Error sets
  - Error set is an unbiased sample from a limited ℓ-bit alphabet. E.g. (00,01,10)
  - E.g. (01|00|00|01|00|10|01|01)
    - Mean: 0.33
    - **33% 00; 33% 01; 33% 10; 0% 11**

Error Set Density, ρ

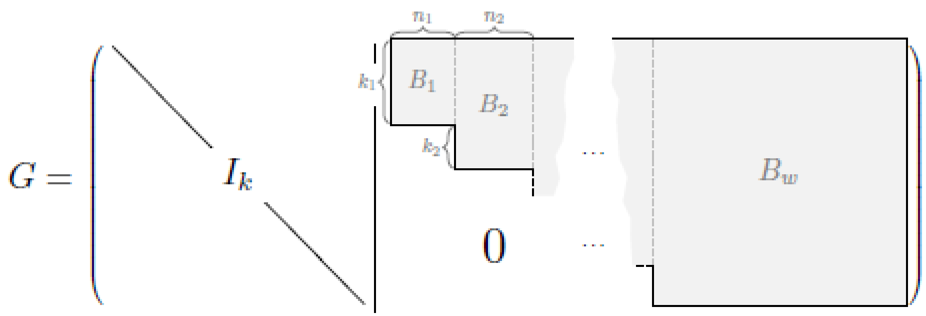
- For an error set of block size ℓ bits, Gligoroski et al define the density as:

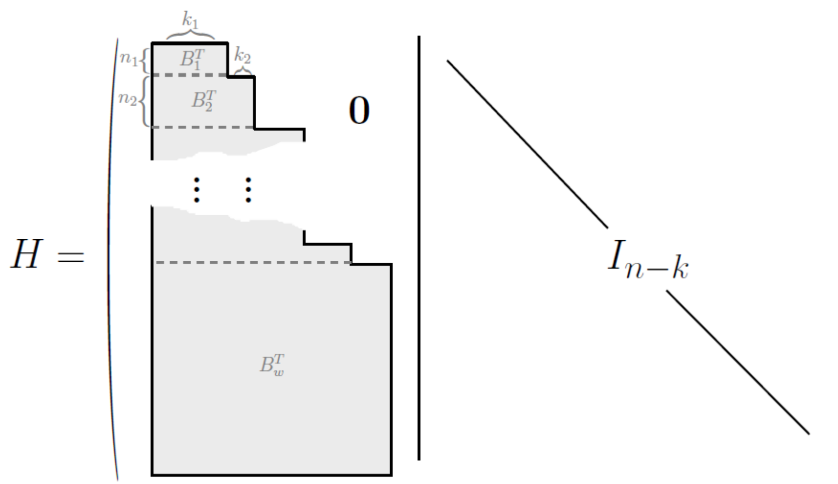
$$\rho_\ell = D(\ell) = |E_\ell|^{1/\ell}$$

- For example for the error set (00,01,10)

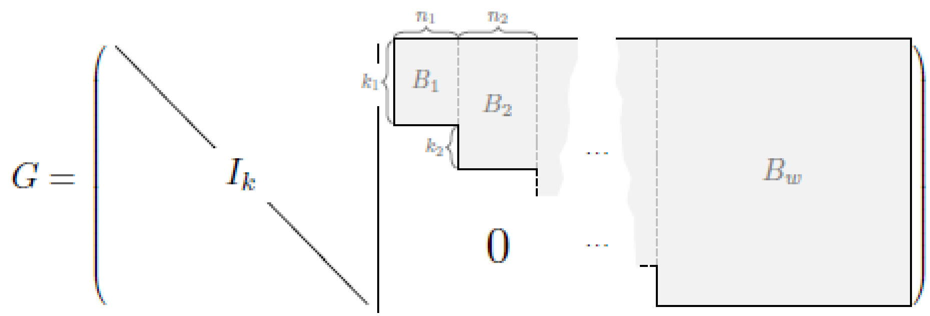
$$\rho_\ell = 3^{1/2}$$

Escher McEliece
The Private key





Escher McEliece Decoding Encryption



- Divide Message as $x_1|x_2|\ldots|x_w$, where x_i has length k_i
- Divide Ciphertext as $y_0|y_1|y_2|\ldots|y_w$, where y_0 has length k and the other y_i have length n_i
- Divide y_0 as $y_0[1]|y_0[2]|\ldots|y_0[w]$, where $y_0[i]$ has length k_i

- Step 0: Compile a list of all the possible decodings of the first k_1 bits of y

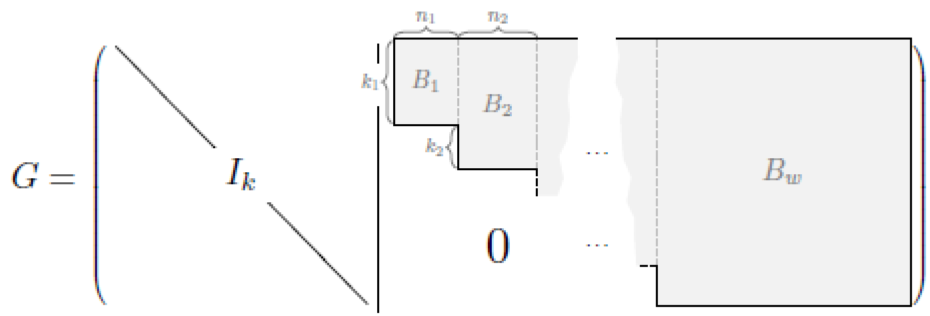
$$x_1 = y_0[1] + e_0[1]$$

- Step $1 \le i \le w$: Update by checking consistency and (if necessary) extending the decoding.

$$(x_1|\ldots|x_i) \cdot B_i + y_i = e_i$$
$$x_{i+1} = y_0[i+1] + e_0[i+1]$$

- *Note the complexity of decoding is set by the list size at step 1: ρ^{k_1}, so k_1 can't be too big.*

Escher McEliece Decoding Signatures



- Divide Message as $x_1|x_2|\ldots|x_w$, where x_i has length k_i
- Divide Ciphertext as $y_0|y_1|y_2|\ldots|y_w$, where y_0 has length k and the other y_i have length n_i
- Divide y_0 as $y_0[1]|y_0[2]|\ldots|y_0[w]$, where $y_0[i]$ has length k_i

- Step 0: Compile a list of ***some of*** the possible decodings of the first k_1 bits of y
$$x_1 = y_0[1] + e_0[1]$$
- Step $1 \leq i \leq w$: Update by checking consistency and (if necessary) extending the decoding.
$$(x_1|\ldots|x_i) \cdot B_i + y_i = e_i$$
$$x_{i+1} = y_0[i+1] + e_0[i+1]$$

- *Note the complexity of decoding is set by the list size at step w. Needs to be at least $\left(\frac{2}{\rho}\right)^{n_w}$ to survive consistency checks. Thus n_w can't be too big.*

On to attacks!

Information set Decoding for Errors

1. Permute the columns of G_{pub}
$$G'_{pub} = G_{pub}P' = (A|B)$$
2. Check that A is invertible
3. Guess the first k bits, v, of the permuted error vector eP. If so:
$$yP' = m(A|B) + eP' = (c|d) = (mA + v|D)$$
$$(c - v)A^{-1} = m$$
4. Check the guess by computing the weight/pattern of:
$$y - (c - v)A^{-1}G_{pub}$$
   – If the guess fails, repeat.

Using ISD for Errors to Forge Signatures

- The efficiency of ISD depends on the probability of guessing k bits of the error vector of a valid signature
  - Note that there is not a *unique* valid signature for each message.

- For the error set (00, 01, 10)
  - We can guess a single bit (0), and the other bit is guaranteed to be valid.
  - By choosing the permutation such that all k information-set bits come from different blocks, we guarantee that $2k$ of the n bits form a valid error vector.
  - The probability the remaining $n - 2k$ bits are also in the error set is:

$$p = \left(\frac{\sqrt{3}}{2}\right)^{n-2k}.$$

- Examples:
  - Code (650,306): Claimed security $2^{87.54}$; $p = 2^{-7.88}$
  - Code (1578,786): Claimed security $2^{137.11}$; $p = 2^{-1.25}$

Can this forgery be avoided?

- This attack can be avoided by
  - Only accepting signature error vectors with hamming weight $\sim n/3$.
    - This only gets $p = \left(\frac{\sqrt{3}}{2}\right)^{n-1.5k}$ … not enough.
  - Increasing the ratio $\frac{n}{k}$
    - But this will enable/worsen other attacks. More later …

Information Set Decoding for the Private Key.

- Information set decoding algorithms find ***low weight vectors*** in the ***row space*** of a matrix

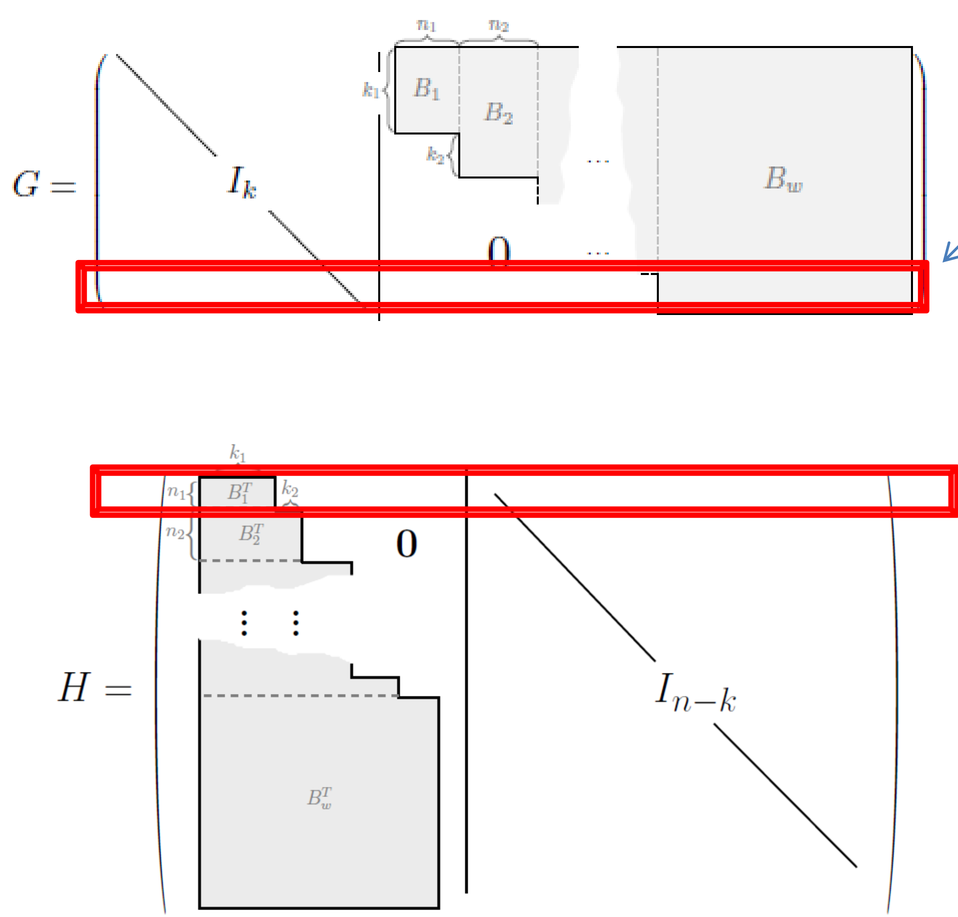
- Can be applied to G_{pub} or H_{pub}.

- Note that G_{pub} and H_{pub} have the same row spaces as G and H up to a Permutation.

$$wt.(vG) = wt.(vGP) = wt(vS^{-1}SGP) = wt((vS^{-1})G_{pub})$$

$$wt.(vH) = wt.(vHP) = wt(vS'^{-1}S'HP) = wt((vS'^{-1})H_{pub})$$

Where are the low-weight targets?



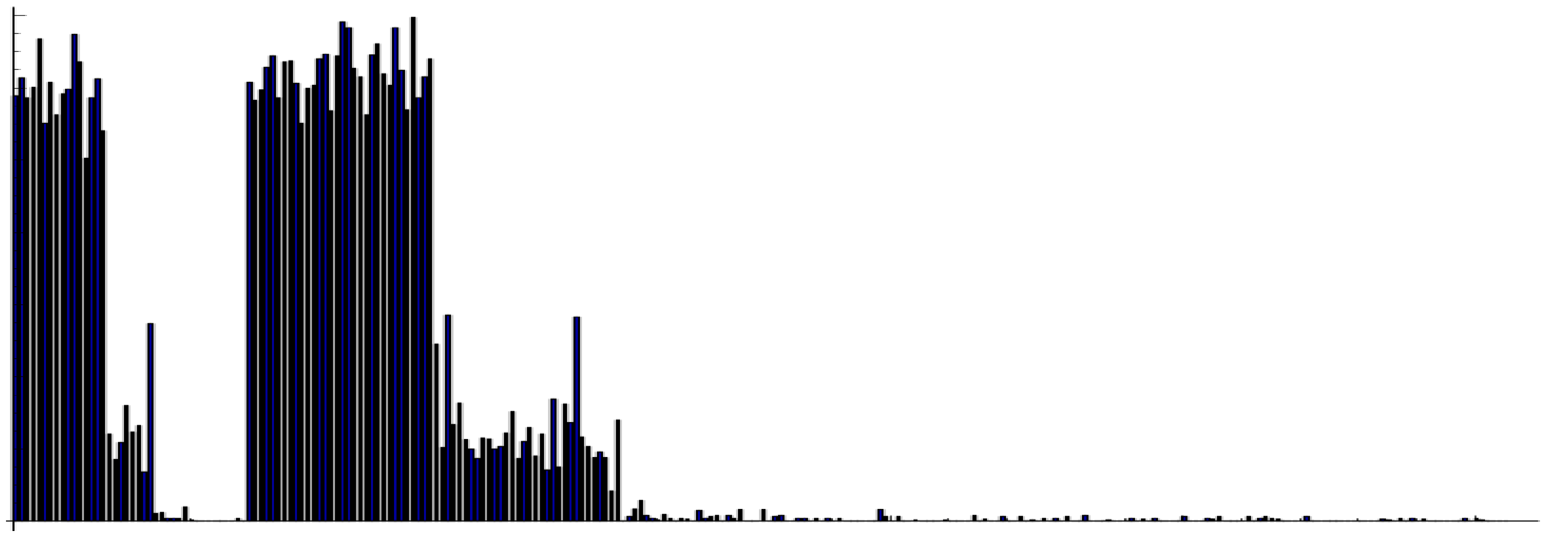
This has to be small for signature

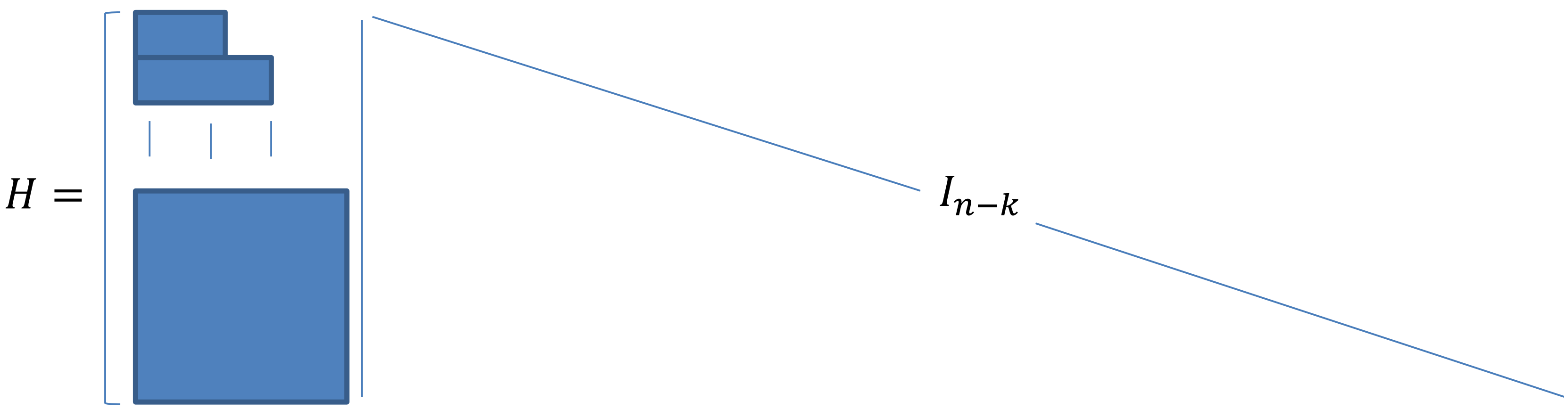
This has to be small for encryption

Once we have low weight vectors, then what?

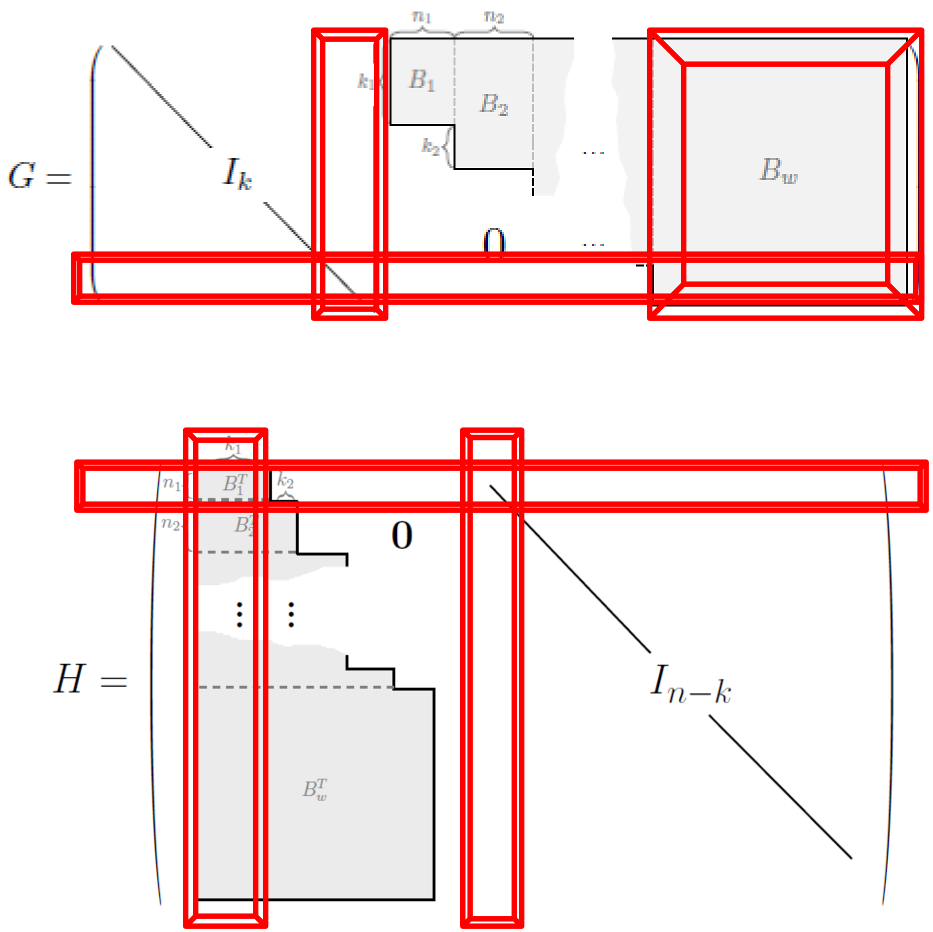
- The nonzero bits of low weight vectors all come from the same columns in the private matrix
  - Encryption
    - H: Columns 0, …, $k_1 - 1$; k, …, $k + n_1$
  - Signature
    - G: Columns 0, …, $k_1 - 1$; $n - n_w, \dots, n - 1$
- The nonzero bits of the low weight vectors will therefore come from the images of these columns under P
- So, we can simply find these columns and lop them off.
  - The result is a smaller matrix of the same form we started with.
  - Repeating the process is generally easier.

Detecting the Structure through Statistics.





Removing The Columns



Generic information set decoding

1. Permute the columns of H_{pub}

$$H'_{pub} = H_{pub}P' = (A|B)$$

2. Check that A is invertible
3. Left multiply by A^{-1}.

$$M = A^{-1}H'_{pub} = (I|Q)$$

4. Check for low weight x' in rowspace of M
   - Check weight of $x' = vM$.
   - If low, return $x = vMP'^{-1}$.

Why does this work?

- $H' = HPP'$, H'_{pub}, M all have the same rowspace

- vM is the unique element of that rowspace with prefix v

- v is a guess for the first n-k bits (IS) of hPP'
  - h is a low weight element of the rowspace of H

- Best guess: *(almost) all zeroes*
  - All zeroes won't work, since $0M = 0$

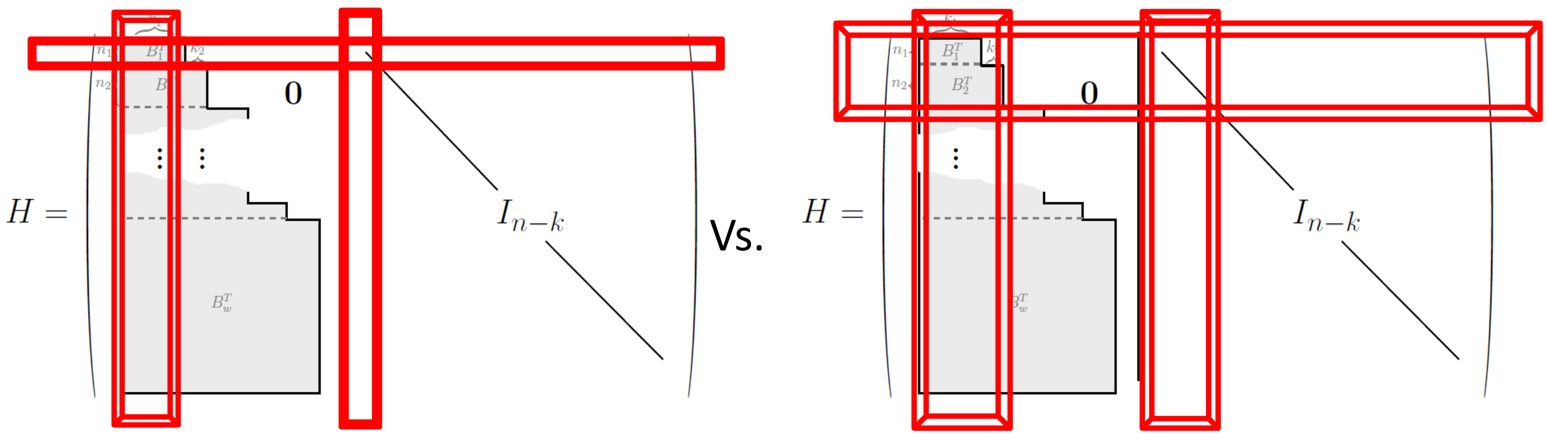
Optimization 1
Using a nonrandom P'

- The permutation P used to disguise G is non-random.
  - It needs to keep ℓ-bit blocks of columns intact.
- The permutation P' used for ISD should be non-random in the same way.
  - Note that if one column in a block is in the set we want to remove, the rest are too.

Optimization 2:

Using rank to check if we're removing the right columns

- Low weight vectors don't all come from the correct rows:



- But, we can tell which is which pretty quickly.
  - Removing most of the highlighted columns at left will reduce the rank.
  - Removing the same number of columns highlighted on the right won't.
- And we can find the rest of the columns pretty easily too.
  - Just check if removing a column reduces the rank more.

Key Recovery Results

- We implemented the key recovery attack in SAGE on a laptop.
- We applied it to the 80 bit parameters for signature and encryption.
- Block structure was correctly recovered in less than 2 hours in both cases.

Can the scheme be saved?

- Encryption:
  - Change the error set: *counterproductive*
  - Key recovery can be avoided by increasing $\frac{n}{k}$
    - However, the keys get ~300 times bigger to achieve claimed security.
- Signature:
  - Change the error set: *still counterproductive*
  - Key recovery can be avoided by increasing $\frac{n}{n-k}$
    - But this worsens the previously discussed forgery attack.
  - Doesn’t look like signature can be saved.

Conclusions

- We demonstrated two highly efficient attacks
  - Near trivial signature forgery.
  - Practical key recovery for both signature and encryption.
- It is possible the encryption scheme can be saved
  - Although at the cost of making an inefficient scheme several orders of magnitude worse.
  - Attack is only quasi-polynomially costlier than decryption.
- The signature scheme does not appear fixable

Thank You!