Solving LPN Using Large Covering Codes

Thom Wiggers

Radboud University, Nijmegen, The Netherlands

8th August 2019

Outline

Post-quantum cryptography

Cryptography based on problems that are hard both for classical and quantum computers.

Post-quantum cryptography

Cryptography based on problems that are hard both for classical and quantum computers.

Categories of mathematical problems

- Lattice-based
- Code-based
- Hash-based
- Multivariate
- Isogenies

Post-quantum cryptography

Cryptography based on problems that are hard both for classical and quantum computers.

Categories of mathematical problems

- Lattice-based
- Code-based
- Hash-based
- Multivariate
- Isogenies

Learning Parity with Noise falls in the code-based category.

Post-quantum cryptography

Cryptography based on problems that are hard both for classical and quantum computers.

Categories of mathematical problems

- Lattice-based
- Code-based
- Hash-based
- Multivariate
- Isogenies

Learning Parity with Noise falls in the code-based category.
We want to qualify how hard the LPN problem is.

Primary school mathematics

Primary school mathematics

All maths in this talk will be in base two:

- $0 + 0 = 0$

Primary school mathematics

All maths in this talk will be in base two:

- $0 + 0 = 0$

Primary school mathematics

All maths in this talk will be in base two:

- $0 + 0 = 0$
- $1 + 0 = 0 + 1 = 1$

Primary school mathematics

All maths in this talk will be in base two:

- $0 + 0 = 0$
- $1 + 0 = 0 + 1 = 1$

Primary school mathematics

All maths in this talk will be in base two:

- $0 + 0 = 0$
- $1 + 0 = 0 + 1 = 1$
- $1 + 1 = 0$

Primary school mathematics

All maths in this talk will be in base two:

- $0 + 0 = 0$
- $1 + 0 = 0 + 1 = 1$
- $1 + 1 = 0$

Primary school mathematics

All maths in this talk will be in base two:

- $0 + 0 = 0$
- $1 + 0 = 0 + 1 = 1$
- $1 + 1 = 0$

So $a + b + b = a$.

Primary school mathematics

All maths in this talk will be in base two:

- $0 + 0 = 0$
- $1 + 0 = 0 + 1 = 1$
- $1 + 1 = 0$

So $a + b + b = a$.

Also, $1-1 = 0 = 1+1$

Outline

Learning Parity *without* Noise

$$\mathsf{s} \cdot \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Learning Parity *without* Noise

Through the magic of *Gaussian elimination*

$$\mathsf{s} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Learning Parity with Noise [Reg05]

We add some noise to the computations. We flip a bit using a biased coin (Bernoulli distribution) that gives head (1) with probability τ.

Learning Parity with Noise [Reg05]

We add some noise to the computations. We flip a bit using a biased coin (Bernoulli distribution) that gives head (1) with probability τ.

$$\mathbf{s} \cdot \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} + \mathbf{e} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Suddenly, finding $\mathbf{s}$ is hard.
Hardness related to *decoding random codes*.

The LPN problem

Definition (LPN Oracle samples)

We have some LPN problem with secret $\mathbf{s}$ of length k bits. Our biased 'coin' Ber_τ gives $e = 1$ with probability τ.

The LPN problem

Definition (LPN Oracle samples)

We have some LPN problem with secret $\mathbf{s}$ of length k bits. Our biased 'coin' Ber_τ gives $e = 1$ with probability τ.
We obtain samples $(\mathbf{a}, c)$ such that

The LPN problem

Definition (LPN Oracle samples)

We have some LPN problem with secret $\mathbf{s}$ of length k bits. Our biased 'coin' Ber_τ gives $e = 1$ with probability τ.
We obtain samples $(\mathbf{a}, c)$ such that

$$\langle \mathbf{a}, \mathbf{s} \rangle + e = c$$

where $\mathbf{a}$ is a k-bit uniformly random vector and $e \leftarrow \mathrm{Ber}_\tau$.

The LPN problem

Definition (LPN Oracle samples)

We have some LPN problem with secret $\mathbf{s}$ of length k bits. Our biased 'coin' Ber_τ gives $e = 1$ with probability τ.
We obtain samples $(\mathbf{a}, c)$ such that

$$\langle \mathbf{a}, \mathbf{s} \rangle + e = c$$

where $\mathbf{a}$ is a k-bit uniformly random vector and $e \leftarrow \mathrm{Ber}_\tau$.

$$\begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \\ \mathbf{a}_4 \\ \mathbf{a}_5 \\ \mathbf{a}_6 \end{pmatrix} \cdot \mathbf{s} + \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix}$$

The LPN problem

Definition (LPN Oracle samples)

We have some LPN problem with secret $\mathbf{s}$ of length k bits. Our biased 'coin' Ber_τ gives $e = 1$ with probability τ.
We obtain samples $(\mathbf{a}, c)$ such that

$$\langle \mathbf{a}, \mathbf{s} \rangle + e = c$$

where $\mathbf{a}$ is a k-bit uniformly random vector and $e \leftarrow \mathrm{Ber}_\tau$.

$$A \cdot \mathbf{s} + \mathbf{e} = \mathbf{c}$$

The LPN problem

Definition (LPN Oracle samples)

We have some LPN problem with secret $\mathbf{s}$ of length k bits. Our biased 'coin' Ber_τ gives $e = 1$ with probability τ.
We obtain samples $(\mathbf{a}, c)$ such that

$$\langle \mathbf{a}, \mathbf{s} \rangle + e = c$$

where $\mathbf{a}$ is a k-bit uniformly random vector and $e \leftarrow \mathrm{Ber}_\tau$.

Definition (Search LPN Problem)

Given n samples $(\mathbf{a}, c)$, recover (information on) $\mathbf{s}$.
We want to do this using at most t amount of time, n samples and m memory.

The LPN problem

Definition (LPN Oracle samples)

We have some LPN problem with secret $\mathbf{s}$ of length k bits. Our biased 'coin' Ber_τ gives $e = 1$ with probability τ.
We obtain samples $(\mathbf{a}, c)$ such that

$$\langle \mathbf{a}, \mathbf{s} \rangle + e = c$$

where $\mathbf{a}$ is a k-bit uniformly random vector and $e \leftarrow \mathrm{Ber}_\tau$.

Definition (Search LPN Problem)

Given n samples $(\mathbf{a}, c)$, recover (information on) $\mathbf{s}$.
We want to do this using at most t amount of time, n samples and m memory.

Familiar? LWE is the same problem over $\mathbb{Z}^q$.

The classic BKW algorithm [BKW00]

The classic BKW algorithm [BKW00]

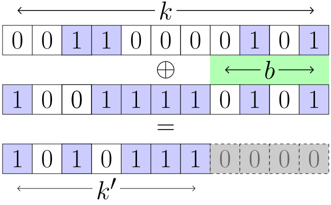
1. Repeat until small enough

The classic BKW algorithm [BKW00]

1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.

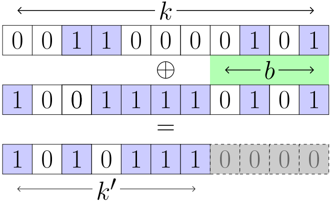
The classic BKW algorithm [BKW00]

1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.
   1.2 Pick one $(\mathbf{a}', c')$ from V_j and add it to all the other samples in V_j.



The classic BKW algorithm [BKW00]

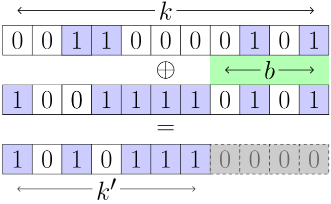
1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.
   1.2 Pick one $(\mathbf{a}', c')$ from V_j and add it to all the other samples in V_j.



2. Throw out all samples that have more than one bit set in $\mathbf{a}$.

The classic BKW algorithm [BKW00]

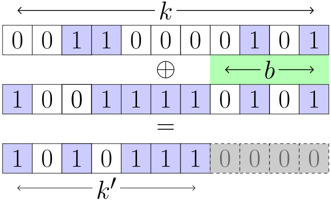
1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.
   1.2 Pick one $(\mathbf{a}', c')$ from V_j and add it to all the other samples in V_j.



2. Throw out all samples that have more than one bit set in $\mathbf{a}$.
3. For $0 < j < b$, sort into sets W_j that have $\mathbf{a}_j = 1$.

The classic BKW algorithm [BKW00]

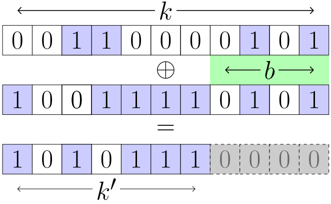
1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.
   1.2 Pick one $(\mathbf{a}', c')$ from V_j and add it to all the other samples in V_j.



2. Throw out all samples that have more than one bit set in $\mathbf{a}$.
3. For $0 < j < b$, sort into sets W_j that have $\mathbf{a}_j = 1$.
4. Decide $\mathbf{s}_j$ by the majority of the c in W_j

The classic BKW algorithm [BKW00]

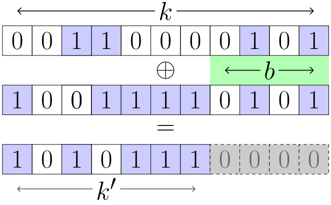
1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.
   1.2 Pick one $(\mathbf{a}', c')$ from V_j and add it to all the other samples in V_j.



2. Throw out all samples that have more than one bit set in $\mathbf{a}$.
3. For $0 < j < b$, sort into sets W_j that have $\mathbf{a}_j = 1$.
4. Decide $\mathbf{s}_j$ by the majority of the c in W_j

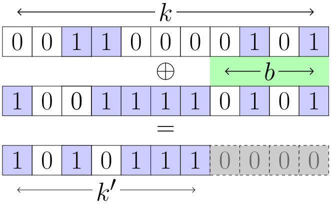
The LF1 algorithm [LF06]

1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.
   1.2 Pick one $(\mathbf{a}', c')$ from V_j and add it to all the other samples in V_j.



The LF1 algorithm [LF06]

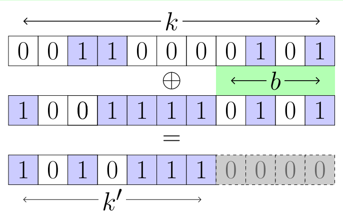
1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.
   1.2 Pick one $(\mathbf{a}', c')$ from V_j and add it to all the other samples in V_j.



2. Apply mathematical magic (Walsh-Hadamard transform) to recover $\mathbf{s}_1, \ldots, \mathbf{s}_b$.

The LF1 algorithm [LF06]

1. Repeat until small enough
   1.1 Sort queries into sets V_j that have the same b bits at the end.
   1.2 Pick one $(\mathbf{a}', c')$ from V_j and add it to all the other samples in V_j.



2. Apply mathematical magic (Walsh-Hadamard transform) to recover $\mathbf{s}_1, \ldots, \mathbf{s}_b$.

Generic solving algorithm

Solve an $\mathrm{LPN}_{k,\tau}$ problem, given n samples.

1. Apply reduction algorithm and obtain $\mathrm{LPN}_{k',\tau'}$ problem with n' samples.

Generic solving algorithm

Solve an $\mathrm{LPN}_{k,\tau}$ problem, given n samples.

1. Apply reduction algorithm and obtain $\mathrm{LPN}_{k',\tau'}$ problem with n' samples.
2. Apply solving algorithm consuming n' samples.

Generic solving algorithm

Solve an $\mathsf{LPN}_{k,\tau}$ problem, given n samples.

1. Apply reduction algorithm and obtain $\mathsf{LPN}_{k',\tau'}$ problem with n' samples.
2. Apply solving algorithm consuming n' samples.

$\rightarrow$ obtain information on $\mathbf{s}$.

Generic solving algorithm

Solve an $\mathsf{LPN}_{k,\tau}$ problem, given n samples.

1. Apply reduction algorithm and obtain $\mathsf{LPN}_{k',\tau'}$ problem with n' samples.
2. Apply solving algorithm consuming n' samples.

$\rightarrow$ obtain information on $\mathbf{s}$.

Generic solving algorithm

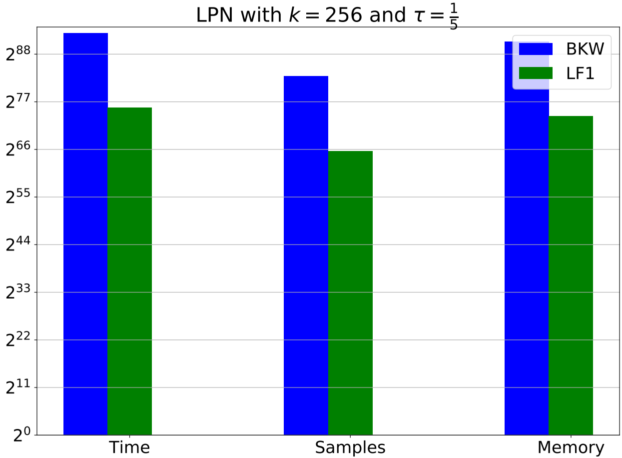
Solve an $\mathsf{LPN}_{k,\tau}$ problem, given n samples.

1. Apply reduction algorithm and obtain $\mathsf{LPN}_{k',\tau'}$ problem with n' samples.
2. Apply solving algorithm consuming n' samples.

$\rightarrow$ obtain information on $\mathbf{s}$.

We may apply several reductions algorithms in sequence!

Complexity


LPN with $k = 256$ and $\tau = \frac{1}{5}$
2^{88}
2^{77}
2^{66}
2^{55}
2^{44}
2^{33}
2^{22}
2^{11}
2^{0}
BKW
LF1
Time
Samples
Memory

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.
2. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.
2. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
3. If $\mathtt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.
2. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
3. If $\texttt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.
2. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
3. If $\texttt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

The $\texttt{Test}(\mathbf{s}')$ algorithm is as follows:

1. Take m samples $(\mathbf{a}, c)$ and write them as matrix $A_{\text{test}}, \mathbf{c}_{\text{test}}$.

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.
2. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
3. If $\texttt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

The $\texttt{Test}(\mathbf{s}')$ algorithm is as follows:

1. Take m samples $(\mathbf{a}, c)$ and write them as matrix $A_{\text{test}}, \mathbf{c}_{\text{test}}$.
2. Compute $\mathbf{e}' = A_{\text{test}} \cdot \mathbf{s}' + \mathbf{c}_{\text{test}}$.

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.
2. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
3. If $\texttt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

The $\texttt{Test}(\mathbf{s}')$ algorithm is as follows:

1. Take m samples $(\mathbf{a}, c)$ and write them as matrix $A_{\text{test}}, \mathbf{c}_{\text{test}}$.
2. Compute $\mathbf{e}' = A_{\text{test}} \cdot \mathbf{s}' + \mathbf{c}_{\text{test}}$.
   - We know $A_{\text{test}} \cdot \mathbf{s} + \mathbf{e} = \mathbf{c}_{\text{test}}$

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.
2. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
3. If $\texttt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

The $\texttt{Test}(\mathbf{s}')$ algorithm is as follows:

1. Take m samples $(\mathbf{a}, c)$ and write them as matrix $A_{\text{test}}, \mathbf{c}_{\text{test}}$.
2. Compute $\mathbf{e}' = A_{\text{test}} \cdot \mathbf{s}' + \mathbf{c}_{\text{test}}$.
    - We know $A_{\text{test}} \cdot \mathbf{s} + \mathbf{e} = \mathbf{c}_{\text{test}}$
    - We also know $\mathbf{e}$ will have roughly $m \cdot \tau$ bits flipped

The Gauss algorithm [EKM17]

Main idea: Try to find an error-free set of samples and then simply apply Gaussian elimination.

1. Take k samples $(\mathbf{a}, c)$ as invertible matrix A and vector $\mathbf{c}$.
2. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
3. If $\texttt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

The $\texttt{Test}(\mathbf{s}')$ algorithm is as follows:

1. Take m samples $(\mathbf{a}, c)$ and write them as matrix $A_{\text{test}}, \mathbf{c}_{\text{test}}$.
2. Compute $\mathbf{e}' = A_{\text{test}} \cdot \mathbf{s}' + \mathbf{c}_{\text{test}}$.
   - We know $A_{\text{test}} \cdot \mathbf{s} + \mathbf{e} = \mathbf{c}_{\text{test}}$
   - We also know $\mathbf{e}$ will have roughly $m \cdot \tau$ bits flipped
3. If $\mathbf{e}'$ has approximately $m \cdot \tau$ bits flipped, probably $\mathbf{s}' = \mathbf{s}$

Variant: Pooled Gauss

1. Take $n = k^2 \log^2 k$ samples as a sample pool.

Variant: Pooled Gauss

1. Take $n = k^2 \log^2 k$ samples as a sample pool.
2. Randomly take k samples $(\mathbf{a}, c)$ from the pool as invertible matrix A and vector $\mathbf{c}$.

Variant: Pooled Gauss

1. Take $n = k^2 \log^2 k$ samples as a sample pool.
2. Randomly take k samples $(\mathbf{a}, c)$ from the pool as invertible matrix A and vector $\mathbf{c}$.
3. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.

Variant: Pooled Gauss

1. Take $n = k^2 \log^2 k$ samples as a sample pool.
2. Randomly take k samples $(\mathbf{a}, c)$ from the pool as invertible matrix A and vector $\mathbf{c}$.
3. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
4. If `Test`$(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

Variant: Pooled Gauss

1. Take $n = k^2 \log^2 k$ samples as a sample pool.
2. Randomly take k samples $(\mathbf{a}, c)$ from the pool as invertible matrix A and vector $\mathbf{c}$.
3. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
4. If `Test`$(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

Variant: Pooled Gauss

1. Take $n = k^2 \log^2 k$ samples as a sample pool.
2. Randomly take k samples $(\mathbf{a}, c)$ from the pool as invertible matrix A and vector $\mathbf{c}$.
3. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
4. If $\texttt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

$\rightsquigarrow$ Needs much less samples.

Variant: Pooled Gauss

1. Take $n = k^2 \log^2 k$ samples as a sample pool.
2. Randomly take k samples $(\mathbf{a}, c)$ from the pool as invertible matrix A and vector $\mathbf{c}$.
3. Compute $\mathbf{s}' = A^{-1} \cdot \mathbf{c}$.
4. If $\texttt{Test}(\mathbf{s}')$ confirms it's error-free, we're done, else `goto 1`.

$\rightsquigarrow$ Needs much less samples.

This algorithm is an *Information-Set Decoding* algorithm: it finds an error-free index set in the pool. Notably, it resembles the [Pra62] algorithm.

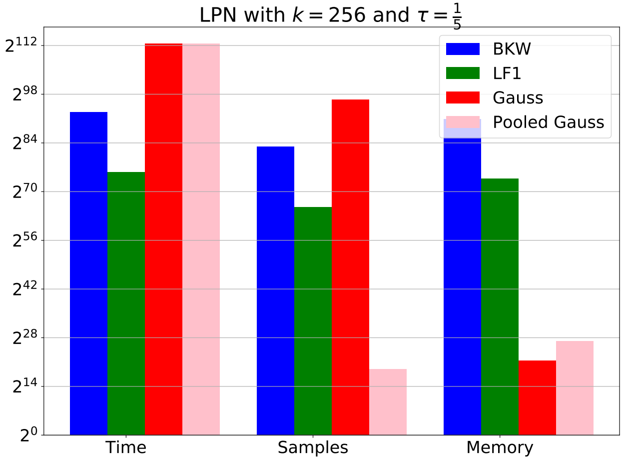
Complexities

To solve $\mathrm{LPN}_{k,\tau}$:

	n **Samples**	**Time**	**Memory**
BKW	$20 \cdot \ln(4k) \cdot 2^b \cdot (1-2\tau)^{-2^a}$	kan	kn
LF1	$(8b+2000)(1-2\tau)^{-2^a} + (a-1)2^b$	$kan + b2^b$	$kn + b2^b$
Gauss	$k \cdot I + m$	$\left(k^3 + km\right) I$	$k^2 + km$
Pooled Gauss	$k^2 \log^2 k + m$	$\left(k^3 + km\right) I$	$k^2 \log^2 k + km$

Gauss needs $I = O\left(\frac{\log_2^2 k}{(1-\tau)^k}\right)$ iterations to find a solution.

Complexity


LPN with $k = 256$ and $\tau = \frac{1}{5}$
2^{112}
2^{98}
2^{84}
2^{70}
2^{56}
2^{42}
2^{28}
2^{14}
2^{0}
Time
Samples
Memory
BKW
LF1
Gauss
Pooled Gauss

Covering Codes

$$\begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline 1 \\ \hline 0 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|c|c|c|}\hline 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

$$\downarrow$$

$$\begin{array}{|c|c|c|c|c|c|c|}\hline 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ \hline \end{array}$$

Covering-codes reduction

$$
k' \left\{ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right. \leftarrow
\begin{array}{c}
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{bmatrix} \\
\uparrow \\
\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \\
\longleftarrow k \longrightarrow
\end{array}
$$

Covering-codes reduction

$$
k'\left\{\begin{array}{|c|}\hline 0\\\hline 0\\\hline 1\\\hline 0\\\hline\end{array}\right. \leftarrow
\begin{array}{c}
\begin{array}{|c|c|c|c|c|c|c|}\hline 1&0&0&0&0&1&1\\\hline 0&1&0&0&1&0&1\\\hline 0&0&1&0&1&1&0\\\hline 0&0&0&1&1&1&1\\\hline\end{array}\\
\uparrow\\
\begin{array}{|c|c|c|c|c|c|c|}\hline 0&1&0&1&1&0&1\\\hline\end{array}\\
\longleftarrow k \longrightarrow
\end{array}
$$

This allows us to reduce a k-size LPN problem with noise τ to a k'-sized LPN problem with noise τ'.
This new noise τ' is strongly dependent on the code used. We measure the impact as bc. ($0 \leq \text{bc} \leq 1$, larger is better)

Finding codes for the reduction

- We need a code that allows us to reduce from k to k'.

$$k' \left\{ \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \right. \leftarrow \begin{bmatrix}1&0&0&0&0&1&1\\0&1&0&0&1&0&1\\0&0&1&0&1&1&0\\0&0&0&1&1&1&1\end{bmatrix}$$

$$\uparrow$$

$$\underbrace{\begin{bmatrix}0&1&0&1&1&0&1\end{bmatrix}}_{k}$$

Finding codes for the reduction

- We need a code that allows us to reduce from k to k'.
- We could use random codes, but they are hard to decode.

$$
\begin{array}{cccc}
k' \updownarrow & \begin{array}{|c|}\hline 0\\\hline 0\\\hline 1\\\hline 0\\\hline\end{array} & \leftarrow & \begin{array}{|c|c|c|c|c|c|c|}\hline 1&0&0&0&0&1&1\\\hline 0&1&0&0&1&0&1\\\hline 0&0&1&0&1&1&0\\\hline 0&0&0&1&1&1&1\\\hline\end{array} \\
 & & & \uparrow \\
 & & & \begin{array}{|c|c|c|c|c|c|c|}\hline 0&1&0&1&1&0&1\\\hline\end{array} \\
 & & & \longleftarrow k \longrightarrow
\end{array}
$$

Finding codes for the reduction

- We need a code that allows us to reduce from k to k'.
- We could use random codes, but they are hard to decode.
- (Quasi-)Perfect codes give the best bc, but only few are known.

$$\underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}}_{k'} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \uparrow \underbrace{\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}}_{k}$$

Coded Gauss

1. Apply covering-codes reduction to reduce problem size from k to k'.
2. Recover secret using Gauss

Coded Gauss

1. Apply covering-codes reduction to reduce problem size from k to k'.
2. Recover secret using Gauss

The complexity of this algorithm:

- We will need $I = \mathcal{O}\left(\frac{\log_2^2 k}{(1-\tau')^{k'}}\right)$ attempts before we find k' error-free samples.

Coded Gauss

1. Apply covering-codes reduction to reduce problem size from k to k'.
2. Recover secret using Gauss

The complexity of this algorithm:

- We will need $I = \mathcal{O}\left(\frac{\log_2^2 k}{(1-\tau')^{k'}}\right)$ attempts before we find k' error-free samples.
- Gauss needs $n = k' \cdot I + m$ samples.

Coded Gauss

1. Apply covering-codes reduction to reduce problem size from k to k'.
2. Recover secret using Gauss

The complexity of this algorithm:

- We will need $I = \mathcal{O}\left(\frac{\log_2^2 k}{(1-\tau')^{k'}}\right)$ attempts before we find k' error-free samples.
- Gauss needs $n = k' \cdot I + m$ samples.
- In each Gauss iteration we do $k^3 + k \cdot m$ work

Coded Gauss

1. Apply covering-codes reduction to reduce problem size from k to k'.
2. Recover secret using Gauss

The complexity of this algorithm:

- We will need $I = \mathcal{O}\left(\frac{\log_2^2 k}{(1-\tau')^{k'}}\right)$ attempts before we find k' error-free samples.
- Gauss needs $n = k' \cdot I + m$ samples.
- In each Gauss iteration we do $k^3 + k \cdot m$ work
- We will need to decode all the n samples as well

Coded Gauss

1. Apply covering-codes reduction to reduce problem size from k to k'.
2. Recover secret using Gauss

The complexity of this algorithm:

- We will need $I = \mathcal{O}\left(\frac{\log_2^2 k}{(1-\tau')^{k'}}\right)$ attempts before we find k' error-free samples.
- Gauss needs $n = k' \cdot I + m$ samples.
- In each Gauss iteration we do $k^3 + k \cdot m$ work
- We will need to decode all the n samples as well

$\rightarrow$ Time complexity $O(n + (k^3 + k \cdot m) \cdot I)$

How 'good' should a code be?

Let's assume we have arbitrary $[k, k']$ codes.

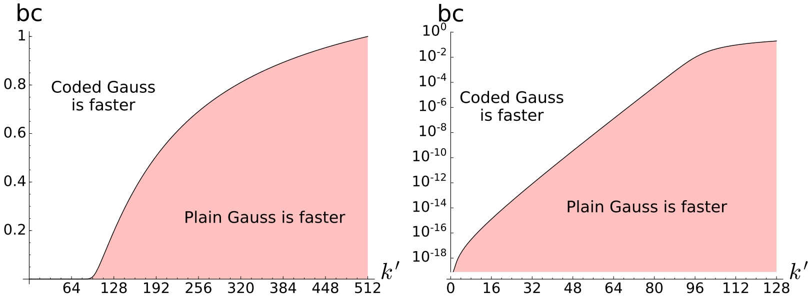
How ‘good’ should a code be?

Let’s assume we have arbitrary $[k, k']$ codes. We have the following inequality

$$T_{\text{Gauss}}(k, \tau) \geq T_{\text{Coded Gauss}}(k, k', \tau, \text{bc})$$

How ‘good’ should a code be?

Let’s assume we have arbitrary $[k, k']$ codes.



(a) Lower bound for bc

(b) More detailed look at $0 < k' \leq 128$.

Figure: Minimal bc before Coded Gauss is faster than applying Gauss to the full problem. $k = 512, \tau = \frac{1}{8}$.

The best-case scenario for bc

Assume we have arbitrary, (quasi-)perfect codes.

The best-case scenario for bc

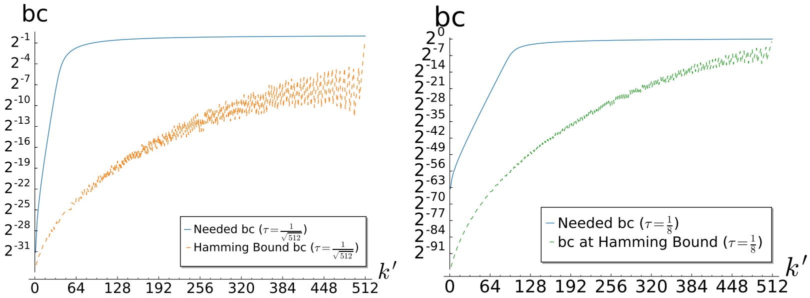
Assume we have arbitrary, (quasi-)perfect codes.

$$\mathrm{bc} \leq 2^{k'-k} \sum_{w=0}^{R} \binom{k}{w} \left(\delta_s^w - \delta_s^{R+1} \right) + \delta_s^{R+1}.$$

Here, R is a property we can bound for quasi-perfect codes (Hamming Bound [Ham50]) and $\delta_s = 1 - 2\tau$.

The best-case scenario for bc

Assume we have arbitrary, (quasi-)perfect codes.

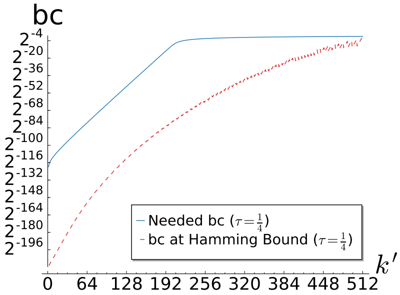


(a) $\tau = \frac{1}{\sqrt{512}}$ (b) $\tau = \frac{1}{8}$

Figure: Minimal bc and the bc obtained at the Hamming bound for various τ. $k = 512, \delta = \delta_s = 1 - 2\tau$.

The best-case scenario for bc

Assume we have arbitrary, (quasi-)perfect codes.



(a) $\tau = \frac{1}{4}$ (b) $\tau = \frac{4\,999\,999}{10\,000\,000}$

Figure: Minimal bc and the bc obtained at the Hamming bound for various τ. $k = 512, \delta = \delta_s = 1 - 2\tau$.

Coded Gauss doesn't work

In conclusion, the following:

1. Apply covering code to reduce to k'-sized problem
2. Use Gauss to solve the problem

isn't faster than only applying step 2.

Coded Gauss doesn't work

In conclusion, the following:

1. Apply covering code to reduce to k'-sized problem
2. Use Gauss to solve the problem

isn't faster than only applying step 2.

Note

Our analysis was limited to the above algorithm. We have results that show the following *may* work

1. Apply some reduction to reduce to a k'-sized problem with noise τ'
2. Apply covering code to reduce to k''-sized problem with noise τ''
3. Use Gauss to solve the problem

Coded Gauss doesn't work

In conclusion, the following:

1. Apply covering code to reduce to k'-sized problem
2. Use Gauss to solve the problem

isn't faster than only applying step 2.

Note

Our analysis was limited to the above algorithm. We have results that show the following *may* work

1. Apply some reduction to reduce to a k'-sized problem with noise τ'
2. Apply covering code to reduce to k''-sized problem with noise τ''
3. Use Gauss to solve the problem

However, we would also need to include the complexity of step 1 when analysing this combination.

Outline

Improving the performance of the covering-codes reduction

The covering-codes reduction as originally proposed:

1. Apply covering-codes reduction
2. Recover information on $\mathbf{s}$ using Walsh-Hadamard Transform.

Improving the performance of the covering-codes reduction

The covering-codes reduction as originally proposed:

1. Apply covering-codes reduction
2. Recover information on $\mathbf{s}$ using Walsh-Hadamard Transform.

Picking codes is hard. Much of the work around this attack has been on finding the right codes to instantiate attacks.

Concatenated Codes

Current attacks use concatenations of small perfect codes to construct larger $[k, k']$ codes.

Example

We construct the following $[12, 4]$ code from $[3, 1]$ repetion codes with generator $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$:

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

Concatenated Codes

Current attacks use concatenations of small perfect codes to construct larger $[k, k']$ codes.

Example

We construct the following $[12, 4]$ code from $[3, 1]$ repetion codes with generator $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$:

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

The bc of concatenated codes is the product of the bc of the smaller codes.

Concatenated Codes (cont.)

Example

We construct the following $[12, 4]$ code from $[3, 1]$ repetion codes with generator $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$:

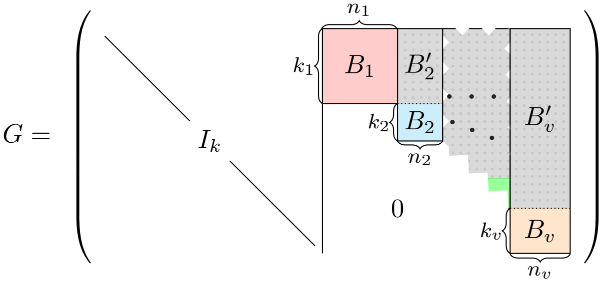
$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Decoding Algorithm

1. Generate look up tables for the small codes
2. Split your vector along the small codes
3. Look up the codewords for the individual pieces in the lookup tables
4. Concatenate

StGen codes

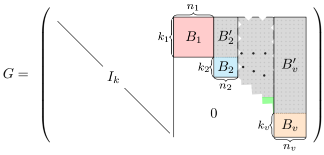
Samardjiska and Gliogoski proposed an improvement on these concatenations of codes. We add random noise on top of the blocks.



(1)

Simona proposed using these codes with the covering-codes reduction at a department lunch talk.

Decoding StGen Codes

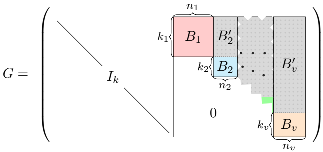


(2)

Decoding algorithm sketch

1. Set maximum error weights and limits

Decoding StGen Codes

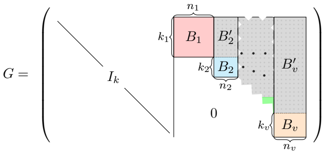


(2)

Decoding algorithm sketch

1. Set maximum error weights and limits
2. Split vector into pieces

Decoding StGen Codes

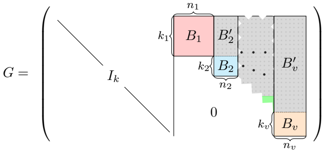


(2)

Decoding algorithm sketch

1. Set maximum error weights and limits
2. Split vector into pieces
3. Produce all candidate codewords and error vectors for first block B_1

Decoding StGen Codes

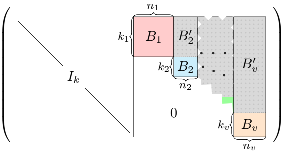


(2)

Decoding algorithm sketch

1. Set maximum error weights and limits
2. Split vector into pieces
3. Produce all candidate codewords and error vectors for first block B_1
4. Multiply each of these by B_2' to account for that random noise

Decoding StGen Codes

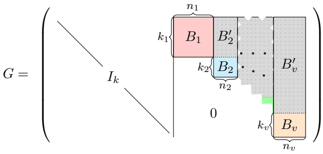
$G =$ 

(2)

Decoding algorithm sketch

1. Set maximum error weights and limits
2. Split vector into pieces
3. Produce all candidate codewords and error vectors for first block B_1
4. Multiply each of these by B'_2 to account for that random noise
5. Generate all the candidates for B_2

Decoding StGen Codes



(2)

Decoding algorithm sketch

1. Set maximum error weights and limits
2. Split vector into pieces
3. Produce all candidate codewords and error vectors for first block B_1
4. Multiply each of these by B_2' to account for that random noise
5. Generate all the candidates for B_2
6. Increase maximum weights if you have few candidates for the next round.

Complications of StGen codes

- Decoding is not trivial

Complications of StGen codes

- Decoding is not trivial
  - Decoding algorithm based on list decoding [SG17]

Complications of StGen codes

- Decoding is not trivial
    - Decoding algorithm based on list decoding [SG17]
    - Highly tweakable

Complications of StGen codes

- Decoding is not trivial
  - Decoding algorithm based on list decoding [SG17]
  - Highly tweakable
- Because of the random elements we can no longer directly compute bc.

Complications of StGen codes

- Decoding is not trivial
  - Decoding algorithm based on list decoding [SG17]
  - Highly tweakable
- Because of the random elements we can no longer directly compute bc.
  - For random codes computing bc is a hugely expensive operation.

Complications of StGen codes

- Decoding is not trivial
  - Decoding algorithm based on list decoding [SG17]
  - Highly tweakable
- Because of the random elements we can no longer directly compute bc.
  - For random codes computing bc is a hugely expensive operation.
  - We instead estimate it over a number of random vectors

Outline

Finding reduction chains

Bogos and Vaudenay propose a search algorithm for finding combinations of reductions:

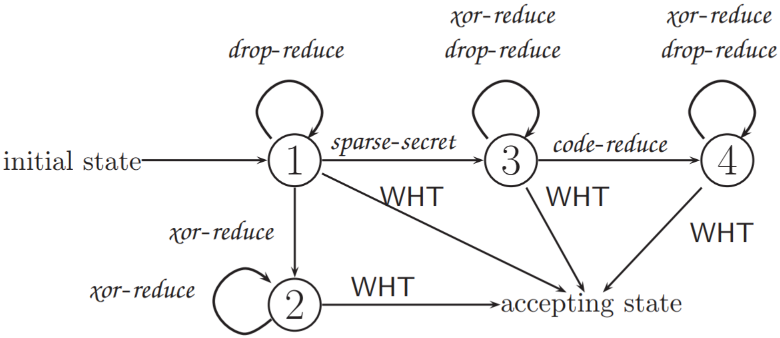


Figure: Finding chains of reductions [Bog17].

Improving the performance of an attack

Bogos and Vaudenay propose a combination of reductions to solve $\mathrm{LPN}_{512,\frac{1}{8}}$ in $\mathcal{O}(2^{78.85})$ time using $2^{63.3}$ samples.

Improving the performance of an attack

Bogos and Vaudenay propose a combination of reductions to solve $\mathrm{LPN}_{512,\frac{1}{8}}$ in $\mathcal{O}(2^{78.85})$ time using $2^{63.3}$ samples.

Step	k	$\log_2 n$	$1-2\tau$	δ_s	Algorithm
1	512	63.3	0.75	0	`sparse-secret`
2	512	63.3	0.75	0.75	`xor-reduce` ($b = 59$)
3	453	66.6	0.5625	0.75	`xor-reduce` ($b = 65$)
4	388	67.2	0.3164	0.75	`xor-reduce` ($b = 66$)
5	322	67.4	0.1001	0.75	`xor-reduce` ($b = 66$)
6	256	67.8	0.0100	0.75	`xor-reduce` ($b = 67$)
7	189	67.6	0.0001	0.75	`covering-codes`
8	64	67.6	$8.8 \cdot 10^{-10}$		`FWHT`

Table: The full solving chain of Bogos and Vaudenay [BV16; BV] on $\mathrm{LPN}_{512,\frac{1}{8}}$. In step 7 they apply a $[189, 64]$ covering code with $\mathrm{bc} = 8.78 \cdot 10^{-6}$.

The code used by Bogos and Vaudenay

The last reduction applied uses a number of random codes.

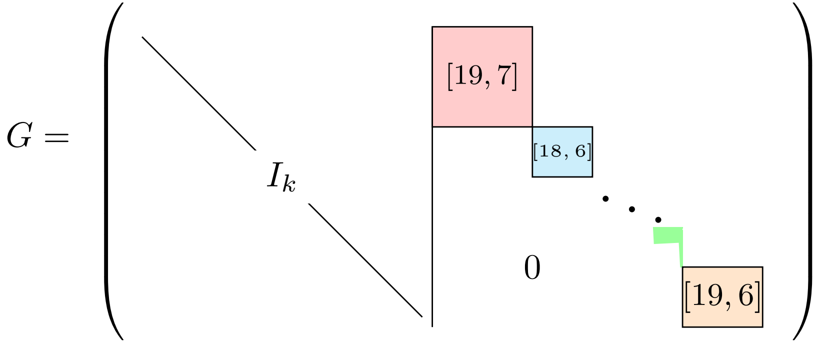
The code used by Bogos and Vaudenay

The last reduction applied uses a number of random codes.

Table: bc for the small random codes used in the solving algorithm for $\mathrm{LPN}_{512,\frac{1}{8}}$ [BV16; BV].

Code	**Count**	**bc** $(\tau = \frac{1}{8})$
$[18, 6]$	1	0.323782920837402
$[19, 6]$	5	0.291754990816116
$[19, 7]$	4	0.336303114891052

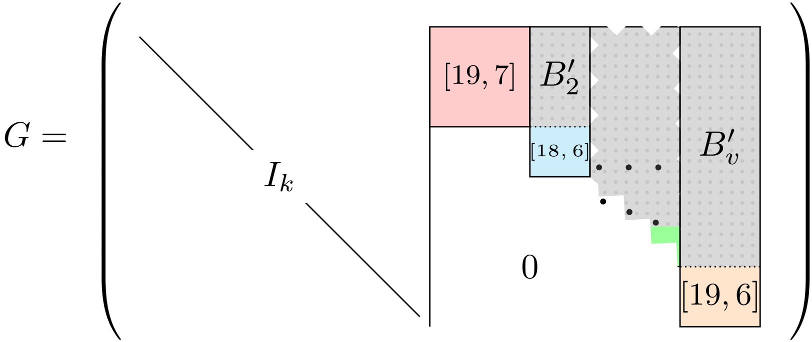
The code used by Bogos and Vaudenay



The bc of the concatenated code is

$$\mathrm{bc} = 0.323^1 \cdot 0.292^5 \cdot 0.336^4 = 8.78 \cdot 10^{-6}.$$

The code used by Bogos and Vaudenay



The bc of this StGen code is approximated to

$$\mathrm{bc} \approx 3.8 \cdot 10^{-5}.$$

Theoretical improvement

Using $\mathrm{bc} = 3.8 \cdot 10^{-5}$ we improve the performance of the algorithm.

Table: Improved attack on $LPN_{512,\frac{1}{8}}$

	Original	**With StGen code**
Time	$\mathcal{O}\left(2^{78.85}\right)$	$\mathcal{O}\left(2^{78.1}\right)$
Samples	$2^{63.3}$	$2^{63.2}$

Theoretical improvement

Using $\mathrm{bc} = 3.8 \cdot 10^{-5}$ we improve the performance of the algorithm.

Table: Improved attack on $LPN_{512,\frac{1}{8}}$

	Original	**With StGen code**
Time	$\mathcal{O}\left(2^{78.85}\right)$	$\mathcal{O}\left(2^{78.1}\right)$
Samples	$2^{63.3}$	$2^{63.2}$

But: we assumed that decoding takes $\mathcal{O}(1)$ time!

Theoretical improvement

Using $\mathrm{bc} = 3.8 \cdot 10^{-5}$ we improve the performance of the algorithm.

Table: Improved attack on $LPN_{512,\frac{1}{8}}$

	Original	**With StGen code**
Time	$\mathcal{O}\left(2^{78.85}\right)$	$\mathcal{O}\left(2^{78.1}\right)$
Samples	$2^{63.3}$	$2^{63.2}$

But: we assumed that decoding takes $\mathcal{O}(1)$ time!

Table: Decoding times

Base codes	**Concatenated**	**StGen**
B&V	0.2 ms	

Theoretical improvement

Using $\text{bc} = 3.8 \cdot 10^{-5}$ we improve the performance of the algorithm.

Table: Improved attack on $LPN_{512,\frac{1}{8}}$

	Original	**With StGen code**
Time	$\mathcal{O}\left(2^{78.85}\right)$	$\mathcal{O}\left(2^{78.1}\right)$
Samples	$2^{63.3}$	$2^{63.2}$

But: we assumed that decoding takes $\mathcal{O}(1)$ time!

Table: Decoding times

Base codes	**Concatenated**	**StGen**
B&V	0.2 ms	$\pm 500\,000$ ms

Theoretical improvement

Using bc $= 3.8 \cdot 10^{-5}$ we improve the performance of the algorithm.

Table: Improved attack on $LPN_{512,\frac{1}{8}}$

	Original	**With StGen code**
Time	$\mathcal{O}\left(2^{78.85}\right)$	$\mathcal{O}\left(2^{78.1}\right)$
Samples	$2^{63.3}$	$2^{63.2}$

But: we assumed that decoding takes $\mathcal{O}(1)$ time!

Table: Decoding times

Base codes	**Concatenated**	**StGen**
B&V	0.2 ms	±500 000 ms
Small perfect	0.009 ms	20–100 ms

Outline

Finding reduction chains

Bogos and Vaudenay propose a search algorithm for finding combinations of reductions:

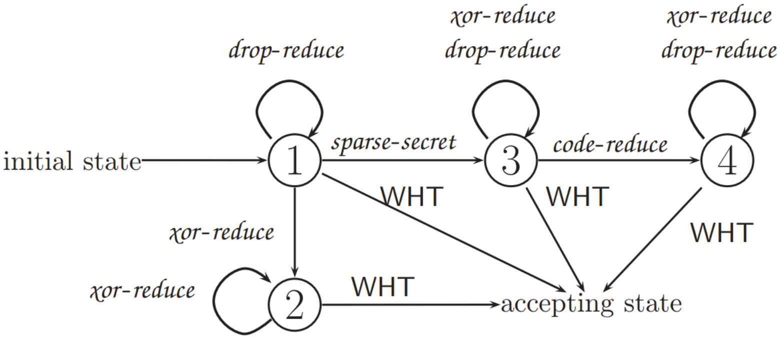


Figure: Finding chains of reductions [Bog17].

Finding new reduction chains

Bogos and Vaudenay propose a search algorithm for finding combinations of reductions:

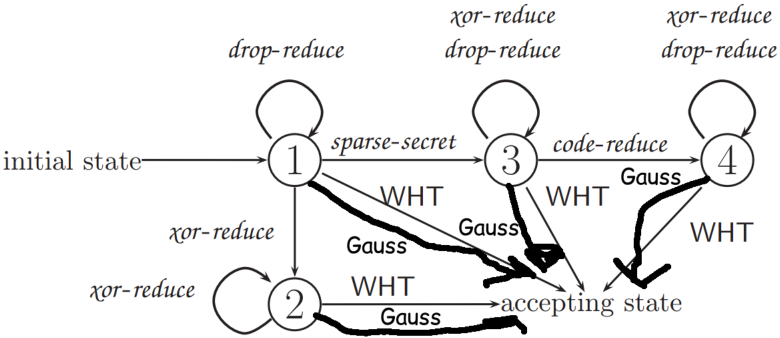


Figure: Finding chains of reductions with Gauss [Bog17].

Memory consumption of *m*

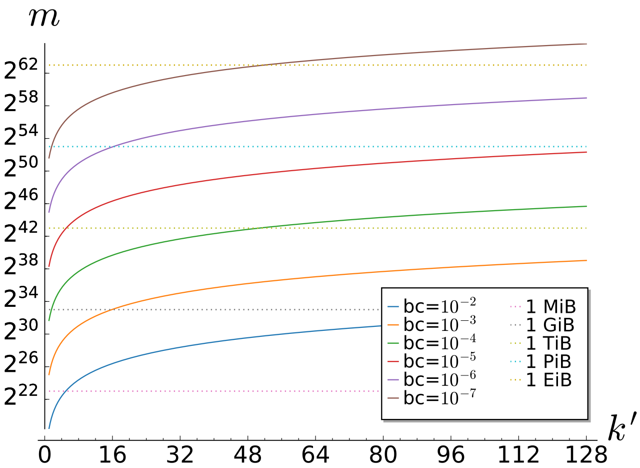


Figure: *m* for various small bc ($\tau = \frac{1}{8}$)

Software

We developed software that allows to implement LPN solving algorithms.

```
// Create LPN oracle with k=32 and tau=1/32
let mut oracle = LpnOracle::new(32, 1.0 / 32.0);
oracle.get_samples(1000);
// apply the LF2 `xor_reduce' reduction
// using b = 8 three times
xor_reduction(&mut oracle, 8);
xor_reduction(&mut oracle, 8);
xor_reduction(&mut oracle, 8);
// solve using two techniques
let fwht_solution = fwht_solve(oracle.clone());
let gauss_solution = pooled_gauss_solve(oracle);
```

Available via `https://thomwiggers.nl/research/msc-thesis/`.

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.

Work in progress

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.

Work in progress

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.
  - At least, in the combinations we presented

Work in progress

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.
  - At least, in the combinations we presented
- We can improve the theoretical performance of solving algorithms using StGen codes

Work in progress

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.
  - At least, in the combinations we presented
- We can improve the theoretical performance of solving algorithms using StGen codes
  - But StGen codes decode so much slower that it's not faster in practice

Work in progress

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.
  - At least, in the combinations we presented
- We can improve the theoretical performance of solving algorithms using StGen codes
  - But StGen codes decode so much slower that it's not faster in practice

Work in progress

- Find combinations of reductions that do work with Gauss

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.
  - At least, in the combinations we presented
- We can improve the theoretical performance of solving algorithms using StGen codes
  - But StGen codes decode so much slower that it's not faster in practice

Work in progress

- Find combinations of reductions that do work with Gauss
- Adapt Bogos and Vaudenay's reduction chain finding algorithm to consider memory consumption

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.
  - At least, in the combinations we presented
- We can improve the theoretical performance of solving algorithms using StGen codes
  - But StGen codes decode so much slower that it's not faster in practice

Work in progress

- Find combinations of reductions that do work with Gauss
- Adapt Bogos and Vaudenay's reduction chain finding algorithm to consider memory consumption
- Add StGen codes to the reduction finding algorithm to find better-optimised solving algorithms.

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.
  - At least, in the combinations we presented
- We can improve the theoretical performance of solving algorithms using StGen codes
  - But StGen codes decode so much slower that it's not faster in practice

Work in progress

- Find combinations of reductions that do work with Gauss
- Adapt Bogos and Vaudenay's reduction chain finding algorithm to consider memory consumption
- Add StGen codes to the reduction finding algorithm to find better-optimised solving algorithms.

Wrapping up

Conclusions

- Solving LPN not only costs a lot of time, but also a lot of memory.
- Combining the covering-codes reduction with Gauss does not work.
  - At least, in the combinations we presented
- We can improve the theoretical performance of solving algorithms using StGen codes
  - But StGen codes decode so much slower that it's not faster in practice

Work in progress

- Find combinations of reductions that do work with Gauss
- Adapt Bogos and Vaudenay's reduction chain finding algorithm to consider memory consumption
- Add StGen codes to the reduction finding algorithm to find better-optimised solving algorithms.

Thank you for your attention

Outline

The BKW algorithm

Input: A set V of n samples $(\mathbf{a}, c)$ from $\mathcal{O}^{\mathsf{LPN}}_{\mathbf{s},t}$, a, b s.t. $k \geq ab$

for $i = 1$ **to** $a - 1$ **do**

  // Reduction (partition-reduce):

  Partition $V = V_1 \cup \cdots \cup V_{2^b}$ s.t. they all have the same bit values on the last ib bits

  foreach V_j **do**

    Choose a $(\mathbf{a}', c') \in V_j$

    Replace all other $(\mathbf{a}, c) \in V_j$ by $(\mathbf{a} + \mathbf{a}', c + c')$

    Discard $(\mathbf{a}', c')$

// Solving phase (majority):

Discard all samples $(\mathbf{a}, c)$ from V where $HW(\mathbf{a}) \neq 1$

Divide V into b partitions, such that vectors $\mathbf{a} \in V_j$ have $\mathbf{a}_j = 1$

for $i = 1$ **to** b **do**

  $\mathsf{s}_i = \text{majority}(c)$, for all $(\mathbf{a}, c) \in V_i$

return $\mathsf{s}_1, \ldots, \mathsf{s}_b$

LF1 algorithm

Algorithm 1: The LF1 algorithm as presented in [BTV15]

Input: A set V of n samples $(\mathbf{a}, c)$ from $\mathcal{O}^{\mathsf{LPN}}_{\mathbf{s},t}$, a, b s.t. $k = ab$

Output: $(\mathbf{s}_1, \ldots, \mathbf{s}_a)$ from $\mathbf{s}$

Run $a - 1$ iterations of `partition-reduce` as in the BKW algorithm

// Solving Phase (FWHT):

$f(\mathbf{x}) = \sum_{(\mathbf{a},c)\in V} 1_{V_{1,\ldots,b}=\mathbf{x}}(-1)^c$

$\hat{f}(\mathbf{x}) = \sum_x (-1)^{\langle \mathbf{a},\mathbf{x}\rangle} f(x)$

return $(\mathbf{s}_1, \ldots, \mathbf{s}_b) = \arg\max_{\mathbf{a}\in\mathbb{Z}_2^b}(\hat{f}(\mathbf{a}))$

LF2 algorithm

Algorithm 2: The LF2 algorithm [LF06]

Input: A set V of n samples $(\mathbf{a}, c)$ from $\mathcal{O}^{\mathsf{LPN}}_{\mathbf{s},t}$,
a, b s.t. $k = ab$

Output: $(\mathbf{s}_1, \ldots, \mathbf{s}_b)$ from $\mathbf{s}$

for $i = 1$ **to** $a - 1$ **do**
  Partition $V = V_1 \cup \cdots \cup V_{2^b}$ s.t. they all have the same bit values on the last ib bits
  foreach V_j **do**
    $V'_j = \emptyset$
    for $(\mathbf{a}, c), (\mathbf{a}', c') \in V_j, (\mathbf{a}, c) \neq (\mathbf{a}', c')$ **do**
      $V'_j = V'_j \cup \{(\mathbf{a} + \mathbf{a}', c + c')\}$
  $V = V'_1 \cup \cdots \cup V'_{2^b}$

// Solving Phase (FWHT) [..]

return $(\mathbf{s}_1, \ldots, \mathbf{s}_b) = \arg\max(\hat{f}(\mathbf{a}))$

Gausss

Function `Gauss`$(\mathcal{O}^{\mathsf{LPN}}_{\mathbf{s},\tau}, \tau)$

  repeat

    repeat

      $(A, \mathbf{c}) \leftarrow \left(\mathcal{O}^{\mathsf{LPN}}_{\mathbf{s},\tau}\right)^k$

    until *A is full rank*

    $\mathbf{s}' = A^{-1}\mathbf{c}$

  until `Test`$(\mathbf{s}', \tau, \frac{1}{2^k}, \left(\frac{1-\tau}{2}\right)^k)$

  return $\mathbf{s}'$

Function `Test`$(\mathbf{s}', \tau, \alpha, \beta)$

  $m = \left(\frac{\sqrt{\frac{3}{2}\ln\left(\frac{1}{\alpha}\right)}+\sqrt{\ln\frac{1}{\beta}}}{\frac{1}{2}-\tau}\right)^2;$

  $c = \tau m + \sqrt{3\left(\frac{1}{2}-\tau\right)\ln\left(\frac{1}{\alpha}\right)m};$

  $(A, \mathbf{c}) \leftarrow \left(\mathcal{O}^{\mathsf{LPN}}_{\mathbf{s},\tau}\right)^m;$

  if $HW(A\mathbf{s}' + \mathbf{c}) \leq c$ **then**

    return *True*;

  else

    return *False*;

Pooled Gauss

Function $\texttt{PooledGauss}(\mathcal{O}^{\mathsf{LPN}}_{\mathbf{s},\tau}, \tau)$

  $P \leftarrow \left(\mathcal{O}^{\mathsf{LPN}}_{\mathbf{s},\tau}\right)^{k^2 \log_2 k}$

  repeat

    repeat

      $(A, \mathbf{c}) \xleftarrow{U} P$

    until *A is full rank*

    $\mathbf{s}' = A^{-1}\mathbf{c}$

  until $\texttt{Test}(\mathbf{s}', \tau, \frac{1}{2^k}, \left(\frac{1-\tau}{2}\right)^k)$

  return $\mathbf{s}'$

When is Coded Gauss faster

$$\frac{\left(k^3 + km\right)\log_2^2 k}{\left(\frac{1}{2} + \frac{1}{2}\delta\right)^k} \geq \frac{\left({k'}^3 + k'm\right)\log_2^2 k'}{\left(\frac{1}{2} + \frac{1}{2}\delta \mathrm{bc}\right)^{k'}} + m + n.$$

Memory consumption of *m*

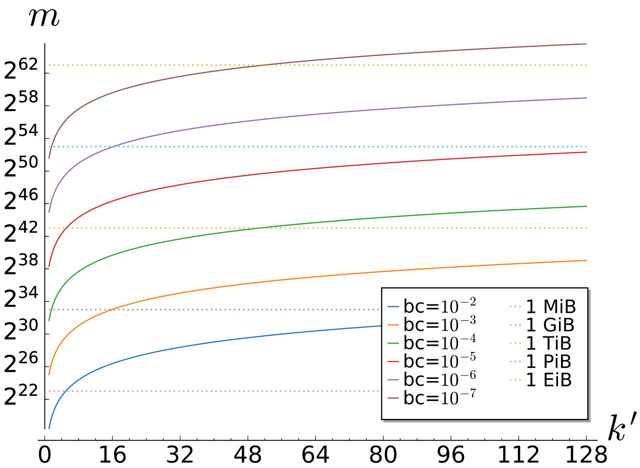


Figure: *m* for various small bc ($\tau = \frac{1}{8}$)

StGen decoding

Input: $w_1, w_b, w_{\text{inc}}, G, L_{\max}, \mathbf{c} \in \mathbb{F}_2^n$.
Output: A close codeword of $\mathbf{c}$
Let $K_i = \Sigma_{j=1}^{i} k_j$, $N_i = \Sigma_{j=1}^{i} n_j$ and let G_i be the 'small code' $(I_{k_i}|B_i)$.

```
L_0 = {(x_0, e_0)}, x_0, e_0 are zero-dimensional vectors.
for i = 1 to v do
    foreach (x_{i-1}, e_{i-1}) in L_{i-1} do
        b = (c_{K_{i-1}}, ..., c_{K_i}) || (x_{i-1} B'_i + (c_{k+N_{i-1}}, ..., c_{k+N_i}))
        max-wt = min(w_i - HW(e_{i-1}), w_b)
        foreach e' ∈ { v ∈ F_2^{n_i+k_i} | HW(v) ≤ max-wt } do
            Find x' s.t. x' G_i + b = e'
            e_new = ((e_{i-1})_1, ..., (e_{i-1})_{K_{i-1}}, e'_1, ..., e'_{k_i},
                     (e_{i-1})_{K_{i-1}}, ..., (e_{i-1})_{K_{i-1}+N_{i-1}}, e'_{k_i}, ..., e'_{k_i+n_i})
            Add (x_{i-1}||x', e_new) to L_i
    if |L_i| < L_max then w_{i+1} = w_i + w_inc  else w_{i+1} = w_i
return x from (x, e) ∈ L_v where HW(e) is minimal
```

Line 1: $L_0 = \{(\mathbf{x}_0, \mathbf{e}_0)\}$, $\mathbf{x}_0, \mathbf{e}_0$ are zero-dimensional vectors.

Line 4: $$\mathbf{b} = \left(\mathbf{c}_{K_{i-1}}, \ldots, \mathbf{c}_{K_i}\right) \,||\, \left(\mathbf{x}_{i-1} B'_i + \left(\mathbf{c}_{k+N_{i-1}}, \ldots, \mathbf{c}_{k+N_i}\right)\right)$$

Line 5: $\text{max-wt} = \min(w_i - HW(\mathbf{e}_{i-1}), w_b)$

Line 6: **foreach** $e' \in \left\{\mathbf{v} \in \mathbb{F}_2^{n_i+k_i} \mid HW(\mathbf{v}) \leq \textit{max-wt}\right\}$ **do**

Line 7: Find $\mathbf{x}'$ s.t. $\mathbf{x}' G_i + \mathbf{b} = \mathbf{e}'$

Line 8: $$\mathbf{e}_{\text{new}} = \Big((\mathbf{e}_{i-1})_1, \ldots, (\mathbf{e}_{i-1})_{K_{i-1}}, \mathbf{e}'_1, \ldots, \mathbf{e}'_{k_i}, (\mathbf{e}_{i-1})_{K_{i-1}}, \ldots, (\mathbf{e}_{i-1})_{K_{i-1}+N_{i-1}}, \mathbf{e}'_{k_i}, \ldots, \mathbf{e}'_{k_i+n_i}\Big)$$

Line 9: Add $(\mathbf{x}_{i-1}||\mathbf{x}', \mathbf{e}_{\text{new}})$ to L_i

Line 10: **if** $|L_i| < L_{\max}$ **then** $w_{i+1} = w_i + w_{\text{inc}}$ **else** $w_{i+1} = w_i$

Line 11: **return** $\mathbf{x}$ *from* $(\mathbf{x}, \mathbf{e}) \in L_v$ *where* $HW(\mathbf{e})$ *is minimal*

Algorithm 3: List-decoding StGen codes [SG15]

Outline

Bibliography I

[BKW00] Avrim Blum, Adam Kalai and Hal Wasserman. 'Noise-tolerant learning, the parity problem, and the statistical query model'. In: *32nd ACM STOC*. ACM Press, May 2000, pp. 435–440.

[Bog17] Sonia Bogos. 'LPN in Cryptography: an Algorithmic Study'. PhD thesis. École Polytechnique Fédérale De Lausanne, 2017. URL: https://infoscience.epfl.ch/record/228977/files/EPFL_TH7800.pdf.

[BTV15] Sonia Bogos, Florian Tramer and Serge Vaudenay. *On Solving LPN using BKW and Variants*. Cryptology ePrint Archive, Report 2015/049. http://eprint.iacr.org/2015/049. 2015.

[BV] Sonia Bogos and Serge Vaudenay. *Optimization of LPN Solving Algorithms: Additional material*. URL: https://infoscience.epfl.ch/record/223773/files/additional_material.pdf?version=1.

Bibliography II

[BV16] Sonia Bogos and Serge Vaudenay. 'Optimization of LPN Solving Algorithms'. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 703–728. DOI: `10.1007/978-3-662-53887-6_26`.

[EKM17] Andre Esser, Robert Kübler and Alexander May. 'LPN Decoded'. In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 486–514.

[Ham50] Richard W. Hamming. 'Error detecting and error correcting codes'. In: *The Bell System Technical Journal* 29.2 (Apr. 1950), pp. 147–160. DOI: `10.1002/j.1538-7305.1950.tb00463.x`.

Bibliography III

[LF06] Éric Levieil and Pierre-Alain Fouque. 'An Improved LPN Algorithm'. In: *SCN 06*. Ed. by Roberto De Prisco and Moti Yung. Vol. 4116. LNCS. Springer, Heidelberg, Sept. 2006, pp. 348–359.

[Pra62] Eugene Prange. 'The use of information sets in decoding cyclic codes'. In: *IRE Transactions on Information Theory* 8.5 (Sept. 1962), pp. 5–9. DOI: 10.1109/TIT.1962.1057777.

[Reg05] Oded Regev. 'On lattices, learning with errors, random linear codes, and cryptography'. In: *37th ACM STOC*. Ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93.

Bibliography IV

[SG15] Simona Samardjiska and Danilo Gligoroski. 'Approaching maximum embedding efficiency on small covers using Staircase-Generator codes'. In: *2015 IEEE International Symposium on Information Theory (ISIT)*. June 2015, pp. 2752–2756. DOI: 10.1109/ISIT.2015.7282957.

[SG17] Simona Samardjiska and Danilo Gligoroski. 'A Robust List-Decoding Algorithm for Maximizing Embedding Efficiency for Arbitrary Payloads'. 2017.