Solving LPN Using Covering Codes

Qian Guo[1,2] Thomas Johansson[1] Carl Löndahl[1]

[1]Dept. of Electrical and Information Technology, Lund University
[2]School of Computer Science, Fudan University

Outline

Outline

Learning Parity with Noise(LPN)

We have access to an oracle (LPN oracle):

The LPN oracle with parameters (k, η):

1. Picks a secret $\mathbf{s}$ in $\mathbb{Z}_2^k$.
2. Randomly picks $\mathbf{r}$ from $\mathbb{Z}_2^k$.
3. Picks a 'noise' $e \leftarrow \mathrm{Ber}_\eta$ (i.e. $e = 0$ w.p. $1 - \eta$ and 1 w.p. η).
4. Outputs the pair $(\mathbf{r},\ v = \langle \mathbf{r}, \mathbf{s} \rangle + e)$ as a sample.

Learning Parity with Noise(LPN)

We have access to an oracle (LPN oracle):

The LPN oracle with parameters (k, η):

1. Picks a secret $\mathbf{s}$ in $\mathbb{Z}_2^k$.
2. Randomly picks $\mathbf{r}$ from $\mathbb{Z}_2^k$.
3. Picks a 'noise' $e \leftarrow \mathrm{Ber}_\eta$ (i.e. $e = 0$ w.p. $1 - \eta$ and 1 w.p. η).
4. Outputs the pair $(\mathbf{r},\ v = \langle \mathbf{r}, \mathbf{s} \rangle + e)$ as a sample.

The goal (informal):

Find $\mathbf{s}$ after collecting enough samples.

Motivation

- Fundamental in theory.
  1. Central problem in learning theory.
  2. A special case of the learning with errors (LWE) problem.
  3. A close connection to the problem of decoding binary random linear codes.
  4. Believed to be hard: no polynomial time algorithm is known.

Motivation

- Fundamental in theory.
  1. Central problem in learning theory.
  2. A special case of the learning with errors (LWE) problem.
  3. A close connection to the problem of decoding binary random linear codes.
  4. Believed to be hard: no polynomial time algorithm is known.
- Many cryptographic applications in practice.
  1. User authentication, encryption, MACs, etc..
  2. Suitable for light-weight crypto: easy to implement, fast to evaluate, hard to break.
  3. Post-quantum cryptography.

Key Parameters

Three important parameters:

1. Dimension k
2. Noise rate η
3. $\#$ of samples n

A very common LPN instance

The LPN instance with $k = 512$, $\eta = 1/8$ and unbounded number of samples.

(Widely adopted for achieving 80-bit security.)

Related Works (1)

The BKW (Blum, Kalai, and Wasserman) algorithm:

Related Works (1)

The BKW (Blum, Kalai, and Wasserman) algorithm:

- The best asymptotic algorithm with sub-exponential complexity $2^{\mathcal{O}(k/\log(k))}$.

Related Works (1)

The BKW (Blum, Kalai, and Wasserman) algorithm:

- The best asymptotic algorithm with sub-exponential complexity $2^{\mathcal{O}(k/\log(k))}$.
- Goal: recover the first bit of $\mathbf{s}$.

The BKW (Blum, Kalai, and Wasserman) algorithm:

- The best asymptotic algorithm with sub-exponential complexity $2^{\mathcal{O}(k/\log(k))}$.
- Goal: recover the first bit of $\mathbf{s}$.
- Main idea:
  - Divide the length k vector into a parts, each with size $b = \lceil k/a \rceil$.
  - Merge and Sort (called one BKW step)—A trade-off:

$$\begin{aligned} v_1 &= \langle [\mathbf{r}_1, \mathbf{r}_0], \mathbf{s} \rangle + e_1 \\ v_2 &= \langle [\mathbf{r}_2, \mathbf{r}_0], \mathbf{s} \rangle + e_2 \\ v_1 + v_2 &= \langle [\mathbf{r}_1 + \mathbf{r}_2, \mathbf{0}], \mathbf{s} \rangle + e_1 + e_2 \end{aligned}$$

  - Do $a-1$ BKW steps to zero out the bottom $a-1$ blocks.
  - With certain probability, find an output sample with $\mathbf{r} = [1, 0, \ldots, 0]$, and then iterate with fresh LPN samples.

Related Works (2)

The Levieil and Fouque (LF) algorithm:

Test on the last block by Fast Walsh-Hadamard Transform.

Related Works (2)

The Levieil and Fouque (LF) algorithm:

Test on the last block by Fast Walsh-Hadamard Transform.

Two concepts about the realization of the BKW step in practice: LF1 (inherited from the BKW paper) and LF2.

- Sort and partition the samples.

  Instance: Suppose in a partition, that we have three samples $(\mathbf{r}_1, v_1)$,$(\mathbf{r}_2, v_2)$,$(\mathbf{r}_3, v_3)$.
- LF1: In each partition, choose one sample and then add it to the rest in the same partition.
  - Output $(\mathbf{r}_2 + \mathbf{r}_1, v_2 + v_1)$,$(\mathbf{r}_3 + \mathbf{r}_1, v_3 + v_1)$.
  - The number of samples reduces about 2^b after each BKW step.
- LF2: In each partition, add any pair in the same partition.
  - Output $(\mathbf{r}_2 + \mathbf{r}_1, v_2 + v_1)$,$(\mathbf{r}_3 + \mathbf{r}_1, v_3 + v_1)$, $(\mathbf{r}_3 + \mathbf{r}_2, v_3 + v_2)$.
  - The number of samples is preserved if we set it to be $3 * 2^b$.

Related Works (3)

Kirchner; Bernstein and Lange (BL):

Kirchner:

Transform the distribution of the secret by Gaussian Elimination:

- Collecting all n samples and representing them in the matrix form $\mathbf{v} = \mathbf{s}\mathbf{R} + \mathbf{e}$, where we choose the first k columns of $\mathbf{R}$ (denoted by $\mathbf{R}_0$) to be invertible.
- Let $\hat{\mathbf{s}} = \mathbf{s}\mathbf{R}_0 + \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix}$ and $\hat{\mathbf{v}} = \mathbf{v} + \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix} \mathbf{R}_0^{-1}\mathbf{R}$.
- Then,

$$\hat{\mathbf{v}} = \begin{bmatrix} \mathbf{0} & \hat{v}_{k+1} & \hat{v}_{k+2} \cdots & \hat{v}_n \end{bmatrix} = \hat{\mathbf{s}}\mathbf{R}_0^{-1}\mathbf{R} + \mathbf{e}. \tag{1}$$

- Since $\mathbf{R}_0^{-1}\mathbf{R}$ is in systematic form, $\hat{\mathbf{s}}$ is then the same as the first k entries of $\mathbf{e}$.

Related Works (3)

Kirchner; Bernstein and Lange (BL):

Bernstein and Lange (BL):

A Ring-LPN solving algorithm. (Easily applied to general LPN)

- Combine partial guessing and Fast Walsh-Hadamard Transform.

Advantage:

Improves the query and memory complexity.

Until now it is the *best-known method* to solve the LPN problem.

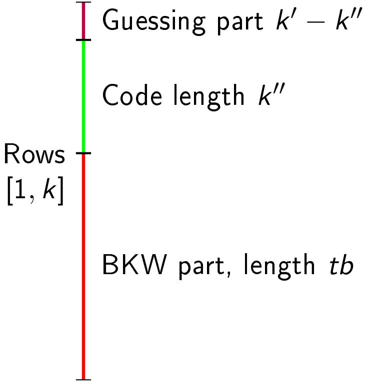
Outline

1. Introduction
   - The LPN Problem
   - Motivation
   - Related Works
2. **A New Algorithm**
   - Linear Approximation Using Covering Codes
   - Subspace Hypothesis Testing
3. Results
   - Complexity
   - Applications
4. Conclusions and Discussions

New Algorithm

Main Steps:

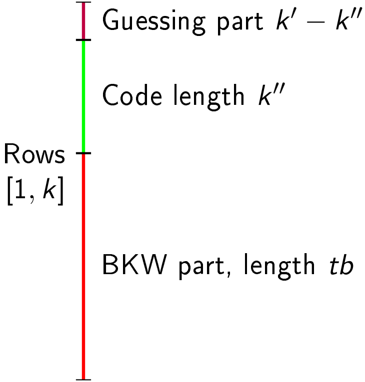
1. Gaussian Elimination.
2. The BKW step.
3. Partial secret guessing.
   - Try every vector with weight less than w_0.
4. Decoding a covering code.
5. Subspace hypothesis testing.

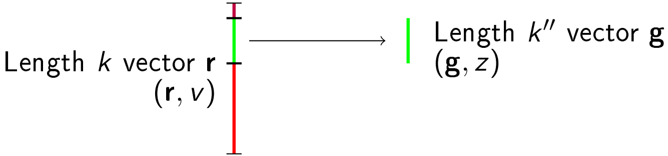


New Algorithm

Main Steps:

1. Gaussian Elimination.
2. The BKW step.
3. Partial secret guessing.
   - Try every vector with weight less than w_0.
4. Decoding a covering code.
5. Subspace hypothesis testing.





Linear Approximation Using Covering Codes

Covering Coding Methods:

- Use a $[k'', l]$ linear code $\mathcal{C}$ with covering radius d_C to group the columns $\mathbf{g}_i$.
  - That is, we rewrite
    $$\mathbf{g}_i = \mathbf{c}_i + \mathbf{e}'_i,$$
    where $\mathbf{c}_i$ is the nearest codeword in $\mathcal{C}$, and $wt(\mathbf{e}'_i) \leq d_C$.
  - The code is characterized by a systematic generator matrix $\mathbf{F}$.
  - Use syndrome decoding by a lookup table.

Linear Approximation Using Covering Codes

Covering Coding Methods:

- Use a $[k'', l]$ linear code $\mathcal{C}$ with covering radius d_C to group the columns $\mathbf{g}_i$.
  - That is, we rewrite
  $$\mathbf{g}_i = \mathbf{c}_i + \mathbf{e}'_i,$$
  where $\mathbf{c}_i$ is the nearest codeword in $\mathcal{C}$, and $wt(\mathbf{e}'_i) \leq d_C$.
  - The code is characterized by a systematic generator matrix $\mathbf{F}$.
  - Use syndrome decoding by a lookup table.

A Concatenated Construction: Due to the memory limit (the size of the syndrome table), we use a concatenation of two $[l_1, l_2]$ linear codes.

- Thus, here $k'' = 2l_1$ and $l = 2l_2$.

Subspace Hypothesis Testing

- Group the samples $(\mathbf{g}_i, z_i)$ in sets $L(\mathbf{c}_i)$ according to their nearest codewords and define the function $f_L(\mathbf{c}_i)$ as

$$f_L(\mathbf{c}_i) = \sum_{(\mathbf{g}_i, z_i) \in L(\mathbf{c}_i)} (-1)^{z_i}.$$

- Define a new function $g(\mathbf{u}) = f_L(\mathbf{c}_i)$, where $\mathbf{u}$ is the information part of $\mathbf{c}_i$ and exhausts all the vectors in $\mathbb{Z}_2^l$.
- The Walsh transform of g is defined as

$$G(\mathbf{y}) = \sum_{\mathbf{u} \in \mathbb{Z}_2^l} g(\mathbf{u})(-1)^{\langle \mathbf{y}, \mathbf{u} \rangle}.$$

Here we exhaust all candidates of $\mathbf{y} \in \mathbb{Z}_2^l$ by computing the Walsh transform.

Subspace Hypothesis Testing

- Group the samples $(\mathbf{g}_i, z_i)$ in sets $L(\mathbf{c}_i)$ according to their nearest codewords and define the function $f_L(\mathbf{c}_i)$ as

$$f_L(\mathbf{c}_i) = \sum_{(\mathbf{g}_i, z_i) \in L(\mathbf{c}_i)} (-1)^{z_i}.$$

- Define a new function $g(\mathbf{u}) = f_L(\mathbf{c}_i)$, where $\mathbf{u}$ is the information part of $\mathbf{c}_i$ and exhausts all the vectors in $\mathbb{Z}_2^l$.
- The Walsh transform of g is defined as

$$G(\mathbf{y}) = \sum_{\mathbf{u} \in \mathbb{Z}_2^l} g(\mathbf{u})(-1)^{\langle \mathbf{y}, \mathbf{u} \rangle}.$$

  Here we exhaust all candidates of $\mathbf{y} \in \mathbb{Z}_2^l$ by computing the Walsh transform.

Observation: Given the candidate $\mathbf{y}$, $G(\mathbf{y})$ is the difference between the number of predicted 0 and the number of predicted 1 for the bit $z_i + \langle \mathbf{y}, \mathbf{u} \rangle$.

Subspace Hypothesis Testing (Cont'd)

Lemma

There exits a unique vector $\mathbf{y} \in \mathbb{Z}_2^l$ *s.t.,*

$$\langle \mathbf{y}, \mathbf{u} \rangle = \langle \mathbf{s}, \mathbf{c}_i \rangle, \quad \forall \mathbf{u} \in \mathbb{Z}_2^l.$$

Sketch of Proof.

As $\mathbf{c}_i = \mathbf{u}\mathbf{F}$, we obtain that $\langle \mathbf{s}, \mathbf{c}_i \rangle = \mathbf{s}\mathbf{F}^\mathsf{T}\mathbf{u}^\mathsf{T} = \langle \mathbf{s}\mathbf{F}^\mathsf{T}, \mathbf{u} \rangle$. □

Observation: Given the candidate $\mathbf{y}$, $G(\mathbf{y})$ is the difference between the number of predicted 0 and the number of predicted 1 for the bit $z_i + \langle \mathbf{y}, \mathbf{u} \rangle$.

Subspace Hypothesis Testing (Cont'd)

Lemma

There exits a unique vector $\mathbf{y} \in \mathbb{Z}_2^l$ *s.t.,*

$$\langle \mathbf{y}, \mathbf{u} \rangle = \langle \mathbf{s}, \mathbf{c}_i \rangle, \quad \forall \mathbf{u} \in \mathbb{Z}_2^l.$$

Sketch of Proof.

As $\mathbf{c}_i = \mathbf{u}\mathbf{F}$, we obtain that $\langle \mathbf{s}, \mathbf{c}_i \rangle = \mathbf{s}\mathbf{F}^\mathsf{T}\mathbf{u}^\mathsf{T} = \langle \mathbf{s}\mathbf{F}^\mathsf{T}, \mathbf{u} \rangle$. □

Observation': Given the candidate $\mathbf{y}$, $G(\mathbf{y})$ is the difference between the number of predicted 0 and the number of predicted 1 for the bit $z_i + \langle \mathbf{s}, \mathbf{c}_i \rangle$.

A Graphic Illustration

Rewrite $\mathbf{g}_i$ as codeword $\mathbf{c}_i = \mathbf{uF}$ and discrepancy $\mathbf{e}'_i$

$$
\begin{bmatrix} * \\ \vdots \\ * \\ \hline z_i + \color{red}{e_i} \\ \hline * \\ \vdots \end{bmatrix}^{\mathsf{T}}
=
\underbrace{\begin{bmatrix} \color{red}{s_1} \\ \vdots \\ \color{red}{s_{k''}} \end{bmatrix}^{\mathsf{T}}}_{\textbf{Secret } \color{red}{\mathbf{s}}}
\underbrace{\left[\begin{array}{cc|c|c} * & * & g_{1,i} & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & g_{k'',i} & * \end{array}\right]}_{\textbf{Query matrix}}
=
\begin{bmatrix} \color{red}{s_1} \\ \vdots \\ \color{red}{s_{k''}} \end{bmatrix}^{\mathsf{T}}
\left[\begin{array}{cc|c|c} * & * & (\mathbf{uF} + \mathbf{e}'_i)_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & (\mathbf{uF} + \mathbf{e}'_i)_{k''} & * \end{array}\right].
$$

A Graphic Illustration

Rewrite $\mathbf{g}_i$ as codeword $\mathbf{c}_i = \mathbf{uF}$ and discrepancy $\mathbf{e}'_i$

$$\begin{bmatrix} * \\ \vdots \\ * \\ \hline z_i + e_i \\ \hline * \\ \vdots \end{bmatrix}^{\mathsf{T}} = \underbrace{\begin{bmatrix} s_1 \\ \vdots \\ s_{k''} \end{bmatrix}^{\mathsf{T}}}_{\textbf{Secret } \mathbf{s}} \underbrace{\left[\begin{array}{cc|c|c} * & * & g_{1,i} & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & g_{k'',i} & * \end{array}\right]}_{\textbf{Query matrix}} = \begin{bmatrix} s_1 \\ \vdots \\ s_{k''} \end{bmatrix}^{\mathsf{T}} \left[\begin{array}{cc|c|c} * & * & (\mathbf{uF} + \mathbf{e}'_i)_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & (\mathbf{uF} + \mathbf{e}'_i)_{k''} & * \end{array}\right].$$

We can separate the discrepancy $\mathbf{e}'_i$ from $\mathbf{uF}$, which yields

$$\begin{bmatrix} s_1 \\ \vdots \\ s_{k''} \end{bmatrix}^{\mathsf{T}} \left[\begin{array}{cc|c|c} * & * & (\mathbf{uF})_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & (\mathbf{uF})_{k''} & * \end{array}\right] = \begin{bmatrix} * \\ \vdots \\ * \\ \hline z_i + e_i + \langle \mathbf{s}, \mathbf{e}'_i \rangle \\ \hline * \\ \vdots \end{bmatrix}.$$

A Graphic Illustration (Cont'd)

Finally, we note that $\mathbf{s}\mathbf{F}^\mathsf{T} \in \mathbb{Z}_2^l$, where $l < k''$. A simple transformation yields,

$$\begin{bmatrix} (\mathbf{s}\mathbf{F}^\mathsf{T})_1 \\ \vdots \\ (\mathbf{s}\mathbf{F}^\mathsf{T})_l \end{bmatrix}^\mathsf{T} \left[\begin{array}{cc|c|c} * & * & u_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & u_l & * \end{array}\right] = \left[\begin{array}{c} * \\ \vdots \\ * \\ \hline z_i + e_i + \langle \mathbf{s}, \mathbf{e}'_i \rangle \\ \hline * \\ \vdots \end{array}\right].$$

A Graphic Illustration (Cont'd)

Finally, we note that $\mathbf{s}\mathbf{F}^{\mathsf{T}} \in \mathbb{Z}_2^l$, where $\underline{l < k''}$. A simple transformation yields,

$$\begin{bmatrix} (\mathbf{s}\mathbf{F}^{\mathsf{T}})_1 \\ \vdots \\ (\mathbf{s}\mathbf{F}^{\mathsf{T}})_l \end{bmatrix}^{\mathsf{T}} \left[\begin{array}{cc|c|c} * & * & u_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & u_l & * \end{array}\right] = \left[\begin{array}{c} * \\ \vdots \\ * \\ \hline z_i + e_i + \langle \mathbf{s}, \mathbf{e}_i' \rangle \\ \hline * \\ \vdots \end{array}\right].$$

- If right guess, then $z_i + \langle \mathbf{s}, \mathbf{c}_i \rangle$ is equal to $e_i + \langle \mathbf{s}, \mathbf{e}_i' \rangle$.
  - Distinguishable when the bias of $\langle \mathbf{s}, \mathbf{e}_i' \rangle$ is large enough.
- Otherwise: random.

A Graphic Illustration (Cont'd)

Finally, we note that $\mathbf{sF}^{\mathsf{T}} \in \mathbb{Z}_2^l$, where $l < k''$. A simple transformation yields,

$$\begin{bmatrix} (\mathbf{sF}^{\mathsf{T}})_1 \\ \vdots \\ (\mathbf{sF}^{\mathsf{T}})_l \end{bmatrix}^{\mathsf{T}} \left[\begin{array}{cc|c|c} * & * & u_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & u_l & * \end{array}\right] = \left[\begin{array}{c} * \\ \vdots \\ * \\ \hline z_i + e_i + \langle \mathbf{s}, \mathbf{e}'_i \rangle \\ \hline * \\ \vdots \end{array}\right].$$

- If right guess, then $z_i + \langle \mathbf{s}, \mathbf{c}_i \rangle$ is equal to $e_i + \langle \mathbf{s}, \mathbf{e}'_i \rangle$.
  - Distinguishable when the bias of $\langle \mathbf{s}, \mathbf{e}'_i \rangle$ is large enough.
- Otherwise: random.

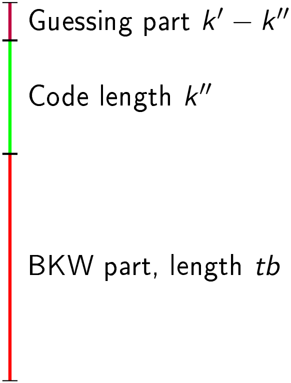
Advantage: $l < k''$.

Outline

Complexity

The complexity consists of three parts:

1. Inner complexity $C_{one-iteration}$.
   - Adding the complexity of each step.
2. Guessing factor F_g.
   - Knowing the probability that w_0 ones occur in the top $k' - k''$ dimensional vector.
3. Testing factor F_t.
   - Knowing the probability that the subspace hypothesis testing succeeds for all different information patterns, after presetting a bias ϵ_{set}.



Complexity Formula

1. $C_{one-iteration} = \log_2(C_1 + C_2 + C_3 + C_4 + C_5)$;
   - C_1 is the complexity of Gaussian Elimination.
   - C_2 is the complexity of t BKW steps.
   - C_3 is the complexity of partial guessing.
   - C_4 is the complexity of syndrome decoding.
   - C_5 is the complexity of subspace hypothesis testing.
2. $F_g = -\log_2(P(w_0, k' - k''))$;[1]

[1] Let $P(w, m)$ be the probability of having at most w errors in m positions, i.e.,

$$P(w, m) = \sum_{i=0}^{w} (1 - \eta)^{m-i} \eta^i \binom{m}{i}.$$

Complexity Formula (Cont'd)

- $F_t = -\log_2(P(c, k''))$, where c is the largest weight of $\mathbf{s}$ that the bias[2] $\epsilon(c)$ is not smaller than ϵ_{set}.

[2] In the proceeding's version, we estimate $\epsilon(c)$ as ϵ'^{c}, where $\epsilon' = 1 - \frac{2d_{\mathcal{C}}}{k''}$. Assume for a code with optimal covering, then we compute the bias accurately as follows ([Vaudenay] Private communication):

$$\Pr\left[\left\langle \mathbf{s}, \mathbf{e}'_i \right\rangle = 1 | wt(\mathbf{s}) = c\right] = \frac{1}{S(k'', d_{\mathcal{C}})} \sum_{i \leq d_{\mathcal{C}}, i \textbf{ odd}} \binom{c}{i} S(k'' - c, d_{\mathcal{C}} - i),$$

where $S(k'', d_{\mathcal{C}})$ is the number of k''-bit strings with weight at most $d_{\mathcal{C}}$. This exact value is $4-5$ times larger on the instance $(512, 1/8)$.

Complexity Formula (Cont'd)

Theorem

The bit complexity (in $\log_2$) of the new algorithm is,

$$C = C_{one-iteration} + F_g + F_t \quad (1)$$

under the condition that[3] $N \geq \frac{4\ln 2 \cdot l}{(\epsilon_{BKW} \cdot \epsilon_{set})^2}$, *where* $\epsilon_{BKW} = (1-2\eta)^{2^t}$.

- Here N is the number of samples after the t BKW steps.
- Using this setting, the error probability P_e is less than 2^{-10}([Selçuk08][4]).

[3] In our proceeding's version, we use a too optimistic estimation on N, i.e., using a constant factor before the term $1/\epsilon^2$. This rough estimation also appears in the previous works.

[4] [Selçuk08] Ali Aydin Selçuk, On Probability of Success in Linear and Differential Cryptanalysis, Journal of cryptology, 2008.

Complexity Formula for Concatenated Construction

- $C_{one-iteration} = \log_2(C_1 + C_2 + C_3 + C_4^{*5} + C_5)$
- Formula F_g is the same.
- Formula F_t changes.
  - Accumulate the probability of all the »good« pair[6], denoted by Pr_t.
    - Important: $wt(\mathbf{s_1})$ and $wt(\mathbf{s_2})$.
    - $$Pr_t = \sum_{(\mathbf{s_1},\mathbf{s_2}) \in \mathbf{Z}_2^{k''}, good} (1-\eta)^{l_1 - wt(\mathbf{s_1})} \eta^{wt(\mathbf{s_1})} \cdot (1-\eta)^{l_1 - wt(\mathbf{s_2})} \eta^{wt(\mathbf{s_2})}.$$
  - The testing factor F_t is equal to $-\log_2(Pr_t)$.

[5] Since we use a concatenated code, the complexity formula of the syndrome decoding step differs and we use C_4^* to denote the new complexity of this step.

[6] For a possible secret vector $(\mathbf{s_1}, \mathbf{s_2})$ in $\mathbb{Z}_2^{k''}$, if the corresponding $\epsilon_{(\mathbf{s_1}, \mathbf{s_2})}$ is larger than a preset bias ϵ_{set}, then we call the pair $(\mathbf{s_1}, \mathbf{s_2})$ a »good« pair.

Complexity in bit operation of the LPN instance with parameters $(512, 1/8)$:

Algorithm	Queries	Memory	Time
Levieil-Fouque	75.7[7]	84.8	87.5
Bernstein-Lange	69.6	78.6	85.8
New algorithm(LF1)	65.0	74.0	80.7
New algorithm(LF2)[8]	63.6	72.6	79.7

[7] The complexity is measured by $\log_2$.

[8] It is the complexity of the following instance: $t = 5$, $b = 62$, using the concatenation of two $[90, 30]$ linear codes, $w_0 = 2$, $\epsilon_{BKW} = 2^{-13.28}$, $\epsilon_{set} = 2^{-14.78}$, $C_{one-iteration} = 76.08$, $F_g = 1.09$ and $F_t = 2.55$.

Applications

1. We have proposed a new algorithm to solve the $(512, 1/8)$ LPN instance in $2^{79.7}$ bit operations (2^{74} word operations with word length 128). Thus, we recommend that the cryptographic schemes employing this instance for 80-bit security to use larger parameters.
   - HB family (HB$^+$, HB$^\#$, etc.)
   - LPN-C.
   - Others.

[9] The reducible case of Lapin designed for 80-bit security is broken within about 2^{70} bit operations: [GJL14] Qian Guo, Thomas Johansson, Carl Löndahl: A New Algorithm for Solving Ring-LPN with a Reducible Polynomial. CoRR abs/1409.0472 (2014).

Applications

1. We have proposed a new algorithm to solve the $(512, 1/8)$ LPN instance in $2^{79.7}$ bit operations (2^{74} word operations with word length 128). Thus, we recommend that the cryptographic schemes employing this instance for 80-bit security to use larger parameters.
   - HB family (HB^+, $HB^\#$, etc.)
   - LPN-C.
   - Others.
2. Better attack for Lapin authentication protocol.
   - The irreducible[9] Ring-LPN instance $(532, 1/8)$ employed in Lapin can be solved within 2^{82} bit operations using this generic algorithm.

[9] The reducible case of Lapin designed for 80-bit security is broken within about 2^{70} bit operations: [GJL14] Qian Guo, Thomas Johansson, Carl Löndahl: A New Algorithm for Solving Ring-LPN with a Reducible Polynomial. CoRR abs/1409.0472 (2014).

Outline

Conclusions and Discussions

Conclusions and Future Works:

1. Having introduced two novel techniques—linear approximation employing covering codes together with a subspace hypothesis testing technique—to form a new generalized-BKW-type LPN solving algorithm.
   - The new algorithm beats the best-known algorithm in query, memory and time complexity.
2. The novel idea inside may also be helpful to solve some other similar problems (e.g., LWE, ISD, etc.).

Conclusions and Discussions

Conclusions and Future Works:

1. Having introduced two novel techniques—linear approximation employing covering codes together with a subspace hypothesis testing technique—to form a new generalized-BKW-type LPN solving algorithm.
   - The new algorithm beats the best-known algorithm in query, memory and time complexity.
2. The novel idea inside may also be helpful to solve some other similar problems (e.g., LWE, ISD, etc.).

An Open Problem:

Improve the asymptotic behavior of the BKW algorithm using the covering coding idea.

Conclusions and Discussions

Conclusions and Future Works:

1. Having introduced two novel techniques—linear approximation employing covering codes together with a subspace hypothesis testing technique—to form a new generalized-BKW-type LPN solving algorithm.
   - The new algorithm beats the best-known algorithm in query, memory and time complexity.
2. The novel idea inside may also be helpful to solve some other similar problems (e.g., LWE, ISD, etc.).

Acknowledgement: We would like to thank Prof. Serge Vaudenay for his helpful comments that improve our work.

Thank you

Thank you for your attention!

Questions?

Testing Using Soft-Information

A Soft-Testing Version: Changing the formula:

Hard Version

$$f_L(\mathbf{c}_i) = \sum_{(\mathbf{g}_i, z_i) \in L(\mathbf{c}_i)} (-1)^{z_i}$$

Soft Version

$$f_L^s(\mathbf{c}_i) = \sum_{(\mathbf{g}_i, z_i) \in L(\mathbf{c}_i)} (-1)^{z_i} \cdot \epsilon_{wt(e_i')}$$

What is $\epsilon_{wt(e_i')}$? $\epsilon^{wt(e_i')}$.

- A Taylor approximation of Log-likelihood ratio (LLR) testing.
- The complexity changes little.