Optimization of LPN Solving Algorithms

Sonia Bogos Serge Vaudenay

EPFL

08 December 2016

Now Hiring!

mailto: `job_lasec@epfl.ch`

Now Hiring!

mailto: `job_lasec@epfl.ch`

Motivation

- LPN can be defined as a noisy system of linear equations in the binary domain

Motivation

- LPN can be defined as a noisy system of linear equations in the binary domain
- believed to be quantum resistant

Motivation

- LPN can be defined as a noisy system of linear equations in the binary domain
- believed to be quantum resistant
- used in authentication protocols and cryptosystems

Motivation

- LPN can be defined as a noisy system of linear equations in the binary domain
- believed to be quantum resistant
- used in authentication protocols and cryptosystems
- special case of LWE, but its hardness is not proven so far

Motivation

- LPN can be defined as a noisy system of linear equations in the binary domain
- believed to be quantum resistant
- used in authentication protocols and cryptosystems
- special case of LWE, but its hardness is not proven so far

Best way to study its hardness is by improving the algorithms that solve it

Our Results

- analyse the existing LPN algorithms and study its building blocks
- improve the theory behind the covering code reduction
- optimise the order and the parameters used in LPN solving algorithms
- improve the best existing algorithms from ASIACRYPT'14 and EUROCRYPT'16

Outline

1. LPN
2. Code Reduction
3. Our Algorithm
4. Results

Outline

1. LPN
2. Code Reduction
3. Our Algorithm
4. Results

Learning Parity with Noise (LPN)

LPN Oracle

Learning Parity with Noise (LPN)

LPN Oracle

secret random vector s

Learning Parity with Noise (LPN)

LPN Oracle

secret random vector s

$c_1 = \langle v_1, s \rangle \oplus d_1$

Learning Parity with Noise (LPN)

LPN Oracle

secret random vector s

$c_1 = \langle v_1, s \rangle \oplus d_1$

random vector

Learning Parity with Noise (LPN)

LPN Oracle

secret random vector s

$c_1 = \langle v_1, s \rangle \oplus d_1$

noise

random vector

Learning Parity with Noise (LPN)

LPN Oracle

secret random vector s $\longrightarrow (v_1, c_1)$

$c_1 = \langle v_1, s \rangle \oplus d_1$

noise

random vector

Learning Parity with Noise (LPN)

LPN Oracle

secret random vector s $\longrightarrow$ (v_2, c_2)

$c_2 = \langle v_2, s \rangle \oplus d_2$

random vector

noise

Learning Parity with Noise (LPN)

LPN Oracle

secret random vector s $\longrightarrow$ (v_i, c_i)

$c_i = \langle v_i, s \rangle \oplus d_i$

noise

random vector

Learning Parity with Noise (LPN)

LPN Oracle

secret random vector s $\longrightarrow$ (v_i, c_i)

$c_i = \langle v_i, s \rangle \oplus d_i$

random vector

noise

Definition (LPN)

Given independent queries from the LPN oracle, find the secret s.

LPN Solving Algorithm

Definition (LPN solving algorithm)

We say that an algorithm $\mathcal{M}$ solves the LPN problem if

$$\Pr[\mathcal{M} \text{ recovers the secret } s] \geq \frac{1}{2},$$

The performance of $\mathcal{M}$ is measured by the running time t, memory m and **number of queries** n from the LPN oracle

Define $\delta = \Pr[d_i = 0] - \Pr[d_i = 1]$ as the **noise bias**

General Structure

To recover a secret s of k bits:

- reduce to a secret s' of $k' \leq k$ bits
- recover the secret s'
- update the queries & repeat the steps

General Structure

To recover a secret s of k bits:

- reduce to a secret s' of $k' \leq k$ bits through **reduction techniques**
- recover the secret s'
- update the queries & repeat the steps

General Structure

To recover a secret s of k bits:

- reduce to a secret s' of $k' \leq k$ bits through **reduction techniques**
- recover the secret s' through **solving techniques**
- update the queries & repeat the steps

General Structure

To recover a secret s of k bits:

- reduce to a secret s' of $k' \leq k$ bits through **reduction techniques**
- recover the secret s' through **solving techniques**
- update the queries & repeat the steps until the **entire s is recovered**

General Structure

To recover a secret s of k bits:

- reduce to a secret s' of $k' \leq k$ bits through **reduction techniques**
- recover the secret s' through **solving techniques**
- update the queries & repeat the steps until the **entire s is recovered**

$$\mathsf{LPN}_s \rightarrow \boxed{\text{reduction}} \rightarrow \mathsf{LPN}_{s_1} \rightarrow \ \ldots \rightarrow \mathsf{LPN}_{s_i} \rightarrow \boxed{\text{solve}} \rightarrow s_i$$

General Structure

To recover a secret s of k bits:

- reduce to a secret s' of $k' \leq k$ bits through **reduction techniques**
- recover the secret s' through **solving techniques**
- update the queries & repeat the steps until the **entire s is recovered**

$$\mathrm{LPN}_s \longrightarrow \boxed{\text{reduction}} \longrightarrow \mathrm{LPN}_{s_1} \longrightarrow \ \ldots \longrightarrow \mathrm{LPN}_{s_i} \longrightarrow \boxed{\text{solve}} \longrightarrow s_i$$

Optimise the use of the reduction techniques

Reduction Techniques

- *sparse-secret*
- *partition-reduce*(b)
- *xor-reduce*(b)
- *drop-reduce*(b)
- *code-reduce*(k, k', params)
- *guess-secret*(b, w)

Reduction Techniques

- *sparse-secret*
- *partition-reduce*(b)
- *xor-reduce*(b)
- *drop-reduce*(b)
- *code-reduce*(k, k', params)
- *guess-secret*(b, w)

Reduction Techniques

- *sparse-secret*
- *partition-reduce*(b)
- *xor-reduce*(b)
- *drop-reduce*(b)
- *code-reduce*(k, k', params)
- *guess-secret*(b, w)

Keep track of the:

- secret size
- number of queries
- noise bias
- secret bias

Reduction *sparse-secret*

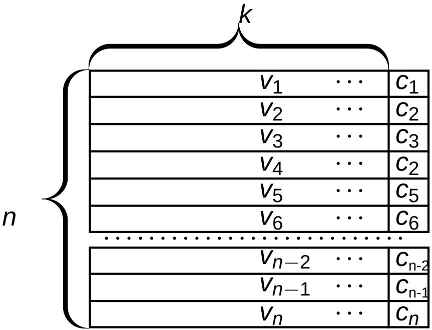
v_1	$\cdots$	c_1
v_2	$\cdots$	c_2
v_3	$\cdots$	c_3
v_4	$\cdots$	c_2
v_5	$\cdots$	c_5
v_6	$\cdots$	c_6
$\cdots$		
v_{n-2}	$\cdots$	$c_{\text{n-2}}$
v_{n-1}	$\cdots$	$c_{\text{n-1}}$
v_n	$\cdots$	c_n

Reduction *sparse-secret*

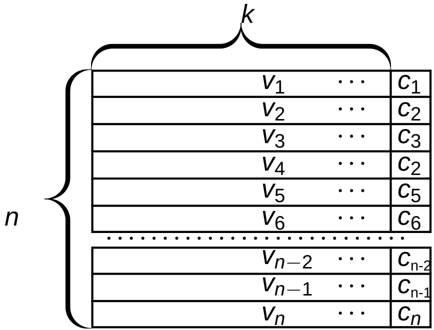
n

v_1	$\cdots$	c_1
v_2	$\cdots$	c_2
v_3	$\cdots$	c_3
v_4	$\cdots$	c_2
v_5	$\cdots$	c_5
v_6	$\cdots$	c_6
$\cdots$		
v_{n-2}	$\cdots$	$c_{\text{n-2}}$
v_{n-1}	$\cdots$	$c_{\text{n-1}}$
v_n	$\cdots$	c_n

Reduction *sparse-secret*



Reduction *sparse-secret*



$$c_i = \langle v_i, s \rangle \oplus d_i$$

Reduction *sparse-secret*

$$
n\left\{
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\multicolumn{8}{c}{\overbrace{\hspace{14em}}^{k}} & \multicolumn{1}{c}{} \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots & 0 \\
\hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots & 0 \\
\hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 1 \\
\hline
0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 1 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & 0 \\
\hline
\multicolumn{9}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\
\hline
0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots & 0 \\
\hline
1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots & 1 \\
\hline
1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\
\hline
\end{array}
\right.
$$

$$c_i = \langle v_i, s \rangle \oplus d_i$$

Reduction *sparse-secret*

$$
\begin{array}{c}
\hphantom{n\{}\overbrace{\hphantom{1\;0\;1\;0\;1\;0\;1\;\cdots\cdots}}^{k}\hphantom{0}\\
n\left\{\begin{array}{|c|c|c|c|c|c|c|c|c|}
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots & 0\\ \hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots & 0\\ \hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots & 0\\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 1\\ \hline
0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 1\\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & 0\\ \hline
\multicolumn{9}{c}{\cdots\cdots\cdots\cdots\cdots\cdots}\\ \hline
0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots & 0\\ \hline
1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots & 1\\ \hline
1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1\\ \hline
\end{array}\right.
\end{array}
\qquad c_i = \langle v_i, s\rangle \oplus d_i
$$

Change the distribution of the secret

Reduction *sparse-secret*

$$
n\left\{
\overbrace{\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots \\ \hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots \\ \hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots \\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\ \hline
0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots \\ \hline
\multicolumn{8}{c}{\cdots\cdots\cdots\cdots\cdots\cdots} \\ \hline
0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots \\ \hline
1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots \\ \hline
1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots \\ \hline
\end{array}}^{k}
\begin{array}{|c|}
\hline
0 \\ \hline
0 \\ \hline
0 \\ \hline
1 \\ \hline
1 \\ \hline
0 \\ \hline
\cdots \\ \hline
0 \\ \hline
1 \\ \hline
1 \\ \hline
\end{array}
\right.
\qquad
c_i = \langle v_i, s \rangle \oplus d_i
$$

Change the distribution of the secret

- from s being uniformly distributed
- to an s where each bit has the same distribution as the noise

Reduction *sparse-secret*

$$
n-k\left\{
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\multicolumn{8}{c}{\overbrace{\hphantom{1\;0\;1\;0\;1\;0\;1\;\cdots\cdots}}^{k}} & \multicolumn{1}{c}{} \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots & 0 \\ \hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots & 0 \\ \hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 1 \\ \hline
0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 1 \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & 0 \\ \hline
\multicolumn{9}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\ \hline
0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots & 0 \\ \hline
1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots & 1 \\ \hline
1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ \hline
\end{array}
\right.
\qquad c'_i = \langle v'_i, s' \rangle \oplus d_i
$$

Change the distribution of the secret

- from s being uniformly distributed
- to an s where each bit has the same distribution as the noise

Complexity: $O(min_{\chi\in\mathbb{N}}(k(n-k)\lceil\frac{k}{\chi}\rceil + k^3 + k\chi 2^{\chi}))$

Reduction *xor-reduce*

$$
n\left\{
\begin{array}{c}
\overbrace{\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots \\ \hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots \\ \hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots \\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\ \hline
0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots \\ \hline
\end{array}}^{k}
\end{array}
\right.
\begin{array}{|c|}
\hline
0 \\ \hline
0 \\ \hline
0 \\ \hline
1 \\ \hline
1 \\ \hline
0 \\ \hline
\end{array}
$$

$$
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\multicolumn{9}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\ \hline
0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots & 0 \\ \hline
1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots & 1 \\ \hline
1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ \hline
\end{array}
$$

$$c_i = \langle v_i, s \rangle \oplus d_i$$

Find collisions on a window of b bits

Reduction *xor-reduce*

$$n\left\{\begin{array}{c|c} \overbrace{\begin{array}{ccccccccc} 1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots \end{array}}^{k} & \begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ \cdots \\ 0 \\ 1 \\ 1 \end{array} \end{array}\right.$$

$$c_i = \langle v_i, s\rangle \oplus d_i$$

Find collisions on a window of b bits

- group queries in equivalence classes
- xor each pair of queries from the same equivalence class

Reduction *xor-reduce*

$$\frac{n(n-1)}{2^{b+1}}\left\{\begin{array}{ccc|cccc c|c}
\multicolumn{3}{c}{} & \multicolumn{5}{c}{\overbrace{\qquad\qquad\qquad\qquad\qquad}^{k-b}} & \\
0&0&0&0&1&0&1&\cdots&0\\
0&0&0&0&1&0&0&\cdots&0\\
0&0&0&1&1&1&0&\cdots&0\\
0&0&0&0&0&1&0&\cdots&1\\
0&0&0&1&0&0&1&\cdots&1\\
0&0&0&0&1&1&0&\cdots&0\\
\vdots&&&&&&&&\vdots\\
0&0&0&0&0&1&0&\cdots&0\\
0&0&0&0&0&1&1&\cdots&1\\
0&0&0&1&1&0&0&\cdots&1
\end{array}\right.$$

$$c_i \oplus c_j = \langle v_i \oplus v_j, s\rangle \oplus d_i \oplus d_j$$

Find collisions on a window of b bits

- group queries in equivalence classes
- xor each pair of queries from the same equivalence class

Complexity: $O(k \cdot \max(n, \frac{n(n-1)}{2^{b+1}}))$

Reduction *xor-reduce*

$$\frac{n(n-1)}{2^{b+1}}\left\{\begin{array}{ccc|cccc|c|c}
\multicolumn{3}{c}{} & \multicolumn{5}{c}{\overbrace{\hspace{10em}}^{k-b}} & \\
\hline
0&0&0&0&1&0&1&\cdots&0\\
0&0&0&0&1&0&0&\cdots&0\\
0&0&0&1&1&1&0&\cdots&0\\
0&0&0&0&0&1&0&\cdots&1\\
0&0&0&1&0&0&1&\cdots&1\\
0&0&0&0&1&1&0&\cdots&0\\
\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&&\vdots\\
0&0&0&0&0&1&0&\cdots&0\\
0&0&0&0&0&1&1&\cdots&1\\
0&0&0&1&1&0&0&\cdots&1\\
\hline
\end{array}\right.$$

$$c_i \oplus c_j = \langle v_i \oplus v_j, s\rangle \oplus d_i \oplus d_j$$

Find collisions on a window of b bits

- group queries in equivalence classes
- xor each pair of queries from the same equivalence class

Complexity: $O(k \cdot \max(n, \frac{n(n-1)}{2^{b+1}}))$

When $n \approx 1 + 2^{b+1}$, the number of queries stay constant

Reduction *drop-reduce*

$$
\begin{array}{c}
\overbrace{\hphantom{1\;0\;1\;0\;1\;0\;1\;\cdots\cdots\cdots}}^{k} \hphantom{\;0}\\
n\left\{
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots & 0 \\ \hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots & 0 \\ \hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 1 \\ \hline
0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 1 \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & 0 \\ \hline
\multicolumn{9}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\ \hline
0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots & 0 \\ \hline
1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots & 1 \\ \hline
1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ \hline
\end{array}
\right.
\end{array}
\qquad c_i = \langle v_i, s\rangle \oplus d_i
$$

Keep only the queries with 0 on a window of b bits

Reduction *drop-reduce*

$$
n\left\{
\begin{array}{c}
\overbrace{
\begin{array}{cccccccc}
1&0&1&0&1&0&1&\cdots\\
1&1&1&0&1&0&0&\cdots\\
0&0&0&1&1&1&0&\cdots\\
0&0&0&0&0&1&0&\cdots\\
0&0&1&1&0&0&1&\cdots\\
1&0&1&0&0&1&1&\cdots\\
\cdots&\cdots&\cdots&\cdots&\cdots&\cdots&\cdots&\cdots\\
0&0&1&1&0&1&1&\cdots\\
1&0&1&0&1&1&0&\cdots\\
1&0&0&1&1&0&0&\cdots
\end{array}
}^{k}
\end{array}
\right.
\begin{array}{|c}
0\\0\\0\\1\\1\\0\\\cdots\\0\\1\\1
\end{array}
\qquad c_i = \langle v_i, s\rangle \oplus d_i
$$

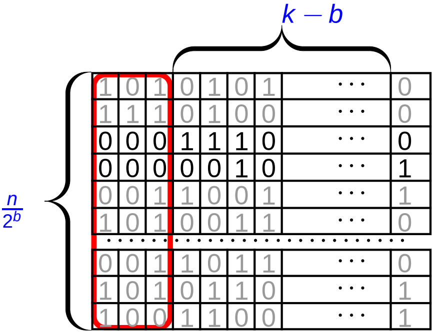
Keep only the queries with 0 on a window of b bits

Reduction *drop-reduce*

$$n\left\{\begin{array}{c} \overbrace{\begin{array}{ccccccccc|c} 1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots & & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots & & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots & & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & & 0 \\ \vdots & & & & & & & & & \vdots \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots & & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots & & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & & 1 \end{array}}^{k} \end{array}\right. \qquad c_i = \langle v_i, s \rangle \oplus d_i$$

Keep only the queries with 0 on a window of b bits

Reduction *drop-reduce*



$c_i = \langle v_i, s \rangle \oplus d_i$

Keep only the queries with 0 on a window of b bits

Complexity: $\mathcal{O}(n(1 + \frac{1}{2} + \ldots + \frac{1}{2^{b-1}}))$

Reduction *code-reduce*

$$
n\left\{
\overbrace{\begin{array}{cccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots \\
\vdots & & & & & & & \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots
\end{array}}^{k}
\right.
\begin{array}{c}
0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 1
\end{array}
$$

$$c_i = \langle v_i, s \rangle \oplus d_i$$

Introduced at ASIACRYPT'14 [GJL]

Use a linear code $\mathcal{C}[k, k', D]$ with generator matrix G, where $g = g'G \in \mathcal{C}$

Approximate each vector v_i to the nearest neighbour in the code $\mathcal{C}$

Reduction *code-reduce*

$$
n\left\{
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\multicolumn{8}{c}{\overbrace{\hspace{14em}}^{k}} & \multicolumn{1}{c}{} \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots & 0 \\ \hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots & 0 \\ \hline
0 & 0 & 0 & 1 & 1 & 1 & 0 & \cdots & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 1 \\ \hline
0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 1 \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & 0 \\ \hline
\multicolumn{9}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\ \hline
0 & 0 & 1 & 1 & 0 & 1 & 1 & \cdots & 0 \\ \hline
1 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots & 1 \\ \hline
1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ \hline
\end{array}
\right.
$$

$$
\begin{aligned}
c_i &= \langle v_i, s \rangle \oplus d_i \\
&= \langle g, s \rangle \oplus \langle v_i - g, s \rangle \oplus d_i \\
&= \langle g'G, s \rangle \oplus \langle v_i - g, s \rangle \oplus d_i \\
&= \langle g', sG^T \rangle \oplus \langle v_i - g, s \rangle \oplus d_i
\end{aligned}
$$

Introduced at ASIACRYPT'14 [GJL]

Use a linear code $\mathcal{C}[k, k', D]$ with generator matrix G, where $g = g'G \in \mathcal{C}$

Approximate each vector v_i to the nearest neighbour in the code $\mathcal{C}$

Reduction *code-reduce*

$$
n\left\{
\begin{array}{c}
\overbrace{\begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 0 & 1 & \cdots \\ \hline \end{array}}^{k'}
\end{array}
\right.
\begin{array}{|c|c|c|c|c|c|}
\hline
1 & 1 & 0 & 1 & \cdots & 0 \\ \hline
0 & 1 & 0 & 0 & \cdots & 0 \\ \hline
0 & 1 & 1 & 1 & \cdots & 0 \\ \hline
1 & 0 & 1 & 0 & \cdots & 1 \\ \hline
0 & 0 & 1 & 0 & \cdots & 1 \\ \hline
0 & 0 & 1 & 1 & \cdots & 0 \\ \hline
\multicolumn{6}{c}{\cdots\cdots\cdots\cdots} \\ \hline
1 & 1 & 0 & 1 & \cdots & 0 \\ \hline
1 & 0 & 1 & 1 & \cdots & 1 \\ \hline
0 & 0 & 1 & 0 & \cdots & 1 \\ \hline
\end{array}
$$

$$
\begin{aligned}
c_i &= \langle v_i, s\rangle \oplus d_i \\
&= \langle g, s\rangle \oplus \langle v_i - g, s\rangle \oplus d_i \\
&= \langle g'G, s\rangle \oplus \langle v_i - g, s\rangle \oplus d_i \\
&= \langle g', sG^T\rangle \oplus \langle v_i - g, s\rangle \oplus d_i
\end{aligned}
$$

Introduced at ASIACRYPT'14 [GJL]

Use a linear code $\mathcal{C}[k, k', D]$ with generator matrix G, where $g = g'G \in \mathcal{C}$

Approximate each vector v_i to the nearest neighbour in the code $\mathcal{C}$

Complexity: $\mathcal{O}(k \cdot n)$

Solving Technique

Define

$$f(x) = \sum_i 1_{v_i = x} (-1)^{\langle v_i, s \rangle \oplus d_i}$$

and apply the Walsh Hadamard Transform (WHT) to obtain

$$\hat{f}(\nu) = \sum_x (-1)^{\langle \nu, x \rangle} f(x) = \sum_i (-1)^{\langle v_i, s + \nu \rangle \oplus d_i}$$

$|\hat{f}(s)|$ is large; In order to be the largest value in the table of $\hat{f}$, we require certain amount of queries

Complexity: $\mathcal{O}(k 2^k \frac{\log_2 n + 1}{2} + kn)$, when WHT is applied for a secret of k bits on n queries

Outline

1 LPN

2 Code Reduction

3 Our Algorithm

4 Results

Bias of the Code Reduction

For *code-reduce* we have

$$c_i = \langle v_i, s\rangle \oplus d_i = \langle g', s'\rangle \oplus \langle v_i - g, s\rangle \oplus d_i$$

Bias of the Code Reduction

For *code-reduce* we have

$$c_i = \langle v_i, s\rangle \oplus d_i = \langle g', s'\rangle \oplus \langle v_i - g, s\rangle \oplus d_i$$

$$\begin{aligned} \text{bc} &= E((-1)^{\langle v_i - g, s\rangle}) = \sum_{e \in \{0,1\}^k} \Pr[v_i - g = e] E((-1)^{\langle e,s\rangle}) \\ &= E\left(\delta_s^{\text{HW}(v_i - g)}\right), \end{aligned}$$

where δ_s is the secret bias

Bias of the Code Reduction

For *code-reduce* we have

$$c_i = \langle v_i, s\rangle \oplus d_i = \langle g', s'\rangle \oplus \langle v_i - g, s\rangle \oplus d_i$$

$$\mathrm{bc} = E((-1)^{\langle v_i - g, s\rangle}) = \sum_{e \in \{0,1\}^k} \Pr[v_i - g = e] E((-1)^{\langle e, s\rangle})$$
$$= E\left(\delta_s^{\mathrm{HW}(v_i - g)}\right),$$

where δ_s is the secret bias
We analyse:

- perfect codes
- quasi-perfect codes
- random codes

Perfect Codes

- Repetition code $[k, 1, \frac{k-1}{2}]$ with k odd

$$\mathrm{bc} = \sum_{w=0}^{\frac{k-1}{2}} \frac{1}{2^{k-1}} \binom{k}{w} \delta_s^w$$

- Golay code $[23, 12, 7]$

$$\mathrm{bc} = 2^{-11} \sum_{w=0}^{3} \binom{23}{w} \delta_s^w$$

- Hamming code $[2^\ell - 1, 2^\ell - \ell, 3]$

$$\mathrm{bc} = 2^{-\ell} \sum_{w=0}^{1} \binom{2^\ell - 1}{w} \delta_s^w$$

Optimal Concatenated Code

Not every code $\mathcal{C}[k, k', D]$ is perfect or quasi-perfect

Optimal Concatenated Code

Not every code $\mathcal{C}[k, k', D]$ is perfect or quasi-perfect

$$\Downarrow$$

Concatenate codes

Optimal Concatenated Code

Not every code $\mathcal{C}[k,k',D]$ is perfect or quasi-perfect

$\Downarrow$

Concatenate codes

Take the $\mathcal{C}$ $[k,k',D]$ code as the concatenation of $\mathcal{C}_1$ $[k-\ell,k'-\ell',D_1]$ and $\mathcal{C}_2$ $[\ell,\ell',D_2]$ with $\mathsf{bc} = \mathsf{bc}_1 \cdot \mathsf{bc}_2$

Optimal Concatenated Code

Not every code $\mathcal{C}[k,k',D]$ is perfect or quasi-perfect

$\Downarrow$

Concatenate codes

Take the $\mathcal{C}\ [k,k',D]$ code as the concatenation of $\mathcal{C}_1\ [k-\ell,k'-\ell',D_1]$ and $\mathcal{C}_2\ [\ell,\ell',D_2]$ with $\mathsf{bc} = \mathsf{bc}_1 \cdot \mathsf{bc}_2$

Computation:

- compute the biases for perfect, quasi-perfect and random codes
- for each $[k,k',D]$, check if $\mathsf{bc}[k,k',D] < \mathsf{bc}[k-\ell,k'-\ell',D_1] \cdot \mathsf{bc}[\ell,\ell',D_2]$

Outline

1. LPN
2. Code Reduction
3. Our Algorithm
4. Results

LPN Solving Automaton

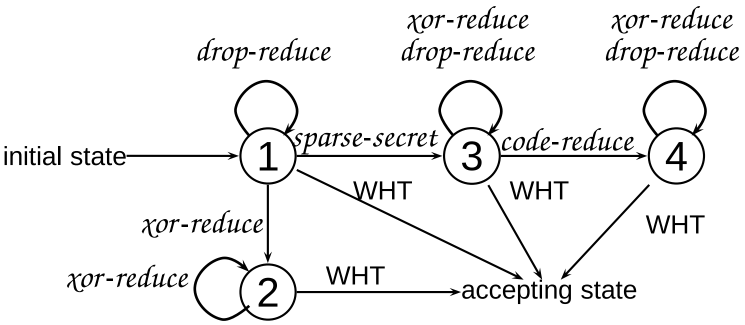
LPN solving algorithms = chains of reductions + WHT

$$\mathrm{LPN}_{s} \longrightarrow \boxed{\text{reduction}} \longrightarrow \mathrm{LPN}_{s_1} \longrightarrow \ \dots \longrightarrow \mathrm{LPN}_{s_i} \longrightarrow \boxed{\text{solve}} \longrightarrow s_i$$

LPN Solving Automaton

LPN solving algorithms = chains of reductions + WHT

$$\mathsf{LPN}_s \longrightarrow \boxed{\text{reduction}} \longrightarrow \mathsf{LPN}_{s_1} \longrightarrow \ \ldots \longrightarrow \mathsf{LPN}_{s_i} \longrightarrow \boxed{\text{solve}} \longrightarrow s_i$$



Graph of Reduction Chains

Construct a graph of all possible reduction chains

- the vertex stores the secret size and the number of queries
- the edge stores the bias change for a reduction

Graph of Reduction Chains

Construct a graph of all possible reduction chains

- the vertex stores the secret size and the number of queries
- the edge stores the bias change for a reduction

Find the reductions that optimize the bias

Graph of Reduction Chains

Construct a graph of all possible reduction chains

- the vertex stores the secret size and the number of queries
- the edge stores the bias change for a reduction

Find the reductions that optimize the bias

The **time complexity** of a chain is the sum of the complexities of each reduction step + cost of WHT

Graph of Reduction Chains

Construct a graph of all possible reduction chains

- the vertex stores the secret size and the number of queries
- the edge stores the bias change for a reduction

Find the reductions that optimize the bias

The **time complexity** of a chain is the sum of the complexities of each reduction step + cost of WHT

Use **max-complexity** as an approximation for the time complexity

Graph of Reduction Chains

Construct a graph of all possible reduction chains

- the vertex stores the secret size and the number of queries
- the edge stores the bias change for a reduction

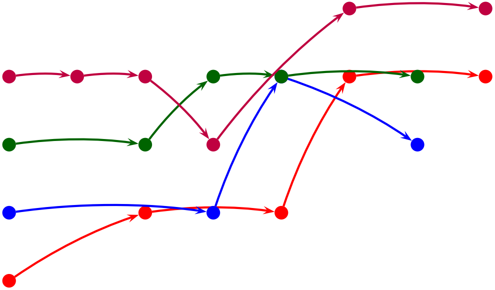
Find the reductions that optimize the bias

The **time complexity** of a chain is the sum of the complexities of each reduction step + cost of WHT

Use **max-complexity** as an approximation for the time complexity

Find the chain with the smallest max-complexity and compute its total time complexity

Graph of Reduction Chains



Find the chain with the smallest max-complexity and compute its total time complexity

Outline

1. LPN
2. Code Reduction
3. Our Algorithm
4. Results

Results

(k,τ)	ASIACRYPT'14 [GJL]	EUROCRYPT'16 [ZJW]	our results
$(512, 0.125)$	81.90	80.09	78.84
$(532, 0.125)$	88.62	82.17	81.02
$(592, 0.125)$	97.71	89.32	87.57

Table: Logarithmic time complexity to solve LPN (in bit operations)

k **- secret size**

τ **- noise level**

Results

τ	k						
	32	48	64	100	256	512	768
0.05	13.89	14.52	16.04	20.47	36.75	57.77	76.63
0.1	15.04	18.58	21.58	27.61	46.75	73.68	98.97
0.125	15.66	19.29	22.94	28.91	49.90	78.85	105.89
0.2	17.01	21.25	24.42	32.06	56.31	89.04	121.04
0.25	18.42	22.34	26.86	32.94	59.47	94.66	127.35

Table: Logarithmic time complexity to solve LPN

Results

τ	k						
	32	48	64	100	256	512	768
0.05	13.89	14.52	16.04	20.47	36.75	57.77	76.63
0.1	15.04	18.58	21.58	27.61	46.75	73.68	98.97
0.125	15.66	19.29	22.94	28.91	49.90	78.85	105.89
0.2	17.01	21.25	24.42	32.06	56.31	89.04	121.04
0.25	18.42	22.34	26.86	32.94	59.47	94.66	127.35

Table: Logarithmic time complexity to solve LPN

Conclusion

- Create an algorithm that automatizes the LPN solving algorithms
- Improve the best existing results
- New reduction techniques can be integrated later on

Thank you for your kind attention!