On the Learning Parity with Noise Problem

Luca Melis

Università degli Studi di Firenze

Århus Universitet

19 Aprile 2013

Advisors:
Prof. Alessandro Piva
Prof. Fabrizio Argenti

Co-advisors:
Dr. Claudio Orlandi
Prof. Ivan Damgård

Scenario

Cryptography schemes

- addresse the security of communication across an insecure medium
- are usually based only on complexity assumptions (standard model)

Near Future:

- Problem: What if someone constructs large quantum computers?
- Cryptography world may fall apart:
    1. cryptographic assumptions broken by efficient quantum algorithms
    2. proof of security becomes invalid

Post-Quantum cryptography

Schemes that are believed to resist classical computers and quantum computers

- **Hash-based cryptography**
- **Code-based cryptography**
- **Lattice-based cryptography**

Some Issues:

- **Efficiency** time and space
- **Confidence** cryptanalysts experience
- **Usability** infrastructure

Our contribution

- We investigate about the Learning Parity with Noise LPN Problem
- We propose a *Threshold Public-Key Encryption* scheme based on LPN
- We propose a *Commitment protocol* based on LPN

Learning Parity with Noise Problem LPN

- Dimension ℓ (security parameter), $q \gg \ell$, $\tau \in \left(0, \frac{1}{2}\right]$
- **Search**: find $\boldsymbol{s} \in \mathbb{Z}_2^{\ell}$ given "noisy random inner products"

  Errors $e_i \leftarrow \mathrm{Ber}_{\tau}$, i.e. $\Pr(e_i = 1) = \tau$
- **Decision**: distinguish $(\boldsymbol{a_i}, \boldsymbol{b_i})$ from uniform $(\boldsymbol{a_i}, b_i)$
- LPN becomes trivial with no error $\tau = 0$ (Gaussian elimination)
- *decisional* and *search* LPN are *"polinomially equivalent"*

Learning Parity with Noise Problem LPN

- Dimension ℓ (security parameter), $q \gg \ell$, $\tau \in \left(0, \frac{1}{2}\right]$
- **Search**: find $\boldsymbol{s} \in \mathbb{Z}_2^\ell$ given "noisy random inner products"

$$\boldsymbol{a_1} \xleftarrow{R} \mathbb{Z}_2^\ell \quad , \quad b_1 =< \boldsymbol{a_1} \, , \, \boldsymbol{s} > \, \oplus \, e_1$$

Errors $e_i \leftarrow \mathrm{Ber}_\tau$, i.e. $\Pr(e_i = 1) = \tau$

- **Decision**: distinguish $(\boldsymbol{a_i}, \boldsymbol{b_i})$ from uniform $(\boldsymbol{a_i}, b_i)$
- LPN becomes trivial with no error $\tau = 0$ (Gaussian elimination)
- *decisional* and *search* LPN are *"polinomially equivalent"*

Learning Parity with Noise Problem LPN

- Dimension ℓ (security parameter), $q \gg \ell$, $\tau \in \left(0, \frac{1}{2}\right]$
- **Search**: find $\boldsymbol{s} \in \mathbb{Z}_2^\ell$ given "noisy random inner products"

$$\begin{array}{ccc} \boldsymbol{a_1} \xleftarrow{R} \mathbb{Z}_2^\ell & , & b_1 =< \boldsymbol{a_1} \,,\, \boldsymbol{s} > \oplus e_1 \\ \boldsymbol{a_2} \xleftarrow{R} \mathbb{Z}_2^\ell & , & b_2 =< \boldsymbol{a_2} \,,\, \boldsymbol{s} > \oplus e_2 \\ & \vdots & \\ \boldsymbol{a_q} \xleftarrow{R} \mathbb{Z}_2^\ell & , & b_q =< \boldsymbol{a_q} \,,\, \boldsymbol{s} > \oplus e_q \end{array}$$

Errors $e_i \leftarrow \mathrm{Ber}_\tau$, i.e. $\Pr(e_i = 1) = \tau$

- **Decision**: distinguish $(\boldsymbol{a_i}, \boldsymbol{b_i})$ from uniform $(\boldsymbol{a_i}, b_i)$
- LPN becomes trivial with no error $\tau = 0$ (Gaussian elimination)
- *decisional* and *search* LPN are *"polinomially equivalent"*

Learning Parity with Noise Problem LPN

- Dimension ℓ (security parameter), $q \gg \ell$, $\tau \in \left(0, \frac{1}{2}\right]$
- **Search**: find $\boldsymbol{s} \in \mathbb{Z}_2^\ell$ given "noisy random inner products"

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a_1} \\ \vdots \\ \boldsymbol{a_q} \end{pmatrix}, \boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{s} \oplus \boldsymbol{e}$$

Errors $e_i \leftarrow \mathrm{Ber}_\tau$, i.e. $\Pr(e_i = 1) = \tau$

- **Decision**: distinguish $(\boldsymbol{a_i}, \boldsymbol{b_i})$ from uniform $(\boldsymbol{a_i}, b_i)$
- LPN becomes trivial with no error $\tau = 0$ (Gaussian elimination)
- *decisional* and *search* LPN are *"polinomially equivalent"*

Learning Parity with Noise Problem LPN

- Dimension ℓ (security parameter), $q \gg \ell$, $\tau \in \left(0, \frac{1}{2}\right]$
- **Search**: find $\boldsymbol{s} \in \mathbb{Z}_2^\ell$ given "noisy random inner products"

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a_1} \\ \vdots \\ \boldsymbol{a_q} \end{pmatrix}, \boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{s} \oplus \boldsymbol{e}$$

Errors $e_i \leftarrow \mathrm{Ber}_\tau$, i.e. $\Pr(e_i = 1) = \tau$

- **Decision**: distinguish $(\boldsymbol{a_i}, \boldsymbol{b_i})$ from uniform $(\boldsymbol{a_i}, b_i)$
- LPN becomes trivial with no error $\tau = 0$ (Gaussian elimination)
- *decisional* and *search* LPN are *"polinomially equivalent"*

Learning Parity with Noise Problem LPN

LPN variants

- Ring LPN
- Subspace LPN
- Exact LPN

Hardness of LPN

Breaking the search LPN problem takes time

- $2^{\Theta(\ell/\log \ell)}$ having the same number of samples q
- $2^{\Theta(\ell/\log\log \ell)}$ having $q = poly(\ell)$ samples
- $2^{\Theta(\ell)}$ having $q = \Theta(\ell)$ samples

Interesting features

- Efficiency $\Rightarrow$ suitable for limited computing power devices (e.g. RFID).
- quantum algorithm resistance

Threshold Public-Key Encryption schemes

Scenario

- In public-key cryptography in general, the ability of decrypting or signing is restricted to the owner of the secret key.
- $\Rightarrow$ only one person has all the power

Solution

- Threshold PKE shares trust among a group of users, such that *enough* of them, the *threshold*, is needed to sign or decrypt
- The secret key is split into shares and each share is given to a group of users.

Our contribution

A Threshold Public-Key Encryption scheme which is:

- based on LPN
- secure in the *Semi-honest* model

Alekhnovich Public Key Encryption scheme

Key Generation

The sender S chooses

- a secret key $\boldsymbol{s} \xleftarrow{R} \mathbb{Z}_2^{\ell}$
- $\boldsymbol{A} \xleftarrow{R} \mathbb{Z}_2^{q \times \ell}$ and the error $\boldsymbol{e} \leftarrow \mathrm{Ber}_{\tau}^{q}$, where $\tau \in \Theta(\frac{1}{\sqrt{\ell}})$ and computes the pk as $(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} \oplus \boldsymbol{e})$

Encryption of a message bit $m \in \mathbb{Z}_2$

Sender S — Receiver R

choose a vector $\boldsymbol{f} \leftarrow \mathrm{Ber}_{\tau}^{q}$

compute $\boldsymbol{u} = \boldsymbol{f} \cdot \boldsymbol{A}$

$c = \langle \boldsymbol{f}, \boldsymbol{b} \rangle \oplus m \xrightarrow{(\boldsymbol{u}, c)}$

Decryption

The receiver R computes $d = c \oplus \langle \boldsymbol{s}, \boldsymbol{u} \rangle = \cdots = \langle \boldsymbol{f}, \boldsymbol{e} \rangle \oplus m$

Protocol phases:

- **Key Generation**
- **Key Assembly**
- **Encryption**
- **Partial Decryption**
- **Finish Decryption**

Protocol phases:

- **Key Generation**
- **Key Assembly**
- **Encryption**
- **Partial Decryption**
- **Finish Decryption**

Protocol phases:

- **Key Generation**
- **Key Assembly**
- **Encryption**
- **Partial Decryption**
- **Finish Decryption**

Protocol phases:

- **Key Generation**
- **Key Assembly**
- **Encryption**
- **Partial Decryption**
- **Finish Decryption**

Protocol phases:

- **Key Generation**
- **Key Assembly**
- **Encryption**
- **Partial Decryption**
- **Finish Decryption**

Key Generation

- All the receivers share a matrix $\boldsymbol{A} \stackrel{R}{\longleftarrow} \mathbb{Z}_2^{q \times \ell}$
- Each receiver $\mathtt{R_i}$ indipendently choose a secret key $\boldsymbol{s_i} \stackrel{R}{\longleftarrow} \mathbb{Z}_2^{\ell}$ and an error $\boldsymbol{e_i} \leftarrow \mathrm{Ber}_\tau^q$
- the public key for $\mathtt{R_i}$ is the pair $(\boldsymbol{A}, \boldsymbol{b_i} = \boldsymbol{A}\boldsymbol{s_i} \oplus \boldsymbol{e_i})$

Key Assembly

The combined public key is the pair $(\boldsymbol{A}, \boldsymbol{b})$, where $\boldsymbol{b} = \bigoplus_{i \in I} \boldsymbol{b_i}$ (I is the users subset)

Encryption Phase

Sender $\underline{\mathtt{S}}$ | Receivers $\underline{\mathtt{R_i}, \mathtt{R_j}}$

$(\boldsymbol{C_1}, \boldsymbol{c_2}) \leftarrow \texttt{ThLPN.Enc}(m, \boldsymbol{b})$

Encryption function (Alekhnovich scheme)

$$\boldsymbol{C_1} = \boldsymbol{F} \cdot \boldsymbol{A},$$

$$\boldsymbol{c_2} = \boldsymbol{F} \cdot \boldsymbol{b} \oplus \begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix} \cdot m$$

where $\boldsymbol{F} := \begin{bmatrix} \boldsymbol{f_1} \\ \dots \\ \boldsymbol{f_q} \end{bmatrix}$, $\boldsymbol{f_i} \leftarrow \mathrm{Ber}_\tau^q$

Encryption Phase

Sender $\underline{\mathtt{S}}$

Receivers $\underline{\mathtt{R_i}, \mathtt{R_j}}$

$(\boldsymbol{C_1}, \boldsymbol{c_2}) \leftarrow \mathtt{ThLPN.Enc}(m, \boldsymbol{b})$

Encryption function (Alekhnovich scheme)

$$\boldsymbol{C_1} = \boldsymbol{F} \cdot \boldsymbol{A},$$

$$\boldsymbol{c_2} = \boldsymbol{F} \cdot \boldsymbol{b} \oplus \begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix} \cdot m$$

where $\boldsymbol{F} := \begin{bmatrix} \boldsymbol{f_1} \\ \dots \\ \boldsymbol{f_q} \end{bmatrix}$, $\boldsymbol{f_i} \leftarrow \mathrm{Ber}_\tau^q$

Encryption Phase

Sender $\underline{\mathtt{S}}$ | Receivers $\underline{\mathtt{R_i}, \mathtt{R_j}}$

$(\boldsymbol{C_1}, \boldsymbol{c_2}) \leftarrow \mathtt{ThLPN.Enc}(m, \boldsymbol{b})$ $\xrightarrow{(\boldsymbol{C_1}, \boldsymbol{c_2})}$

Encryption function (Alekhnovich scheme)

$$\boldsymbol{C_1} = \boldsymbol{F} \cdot \boldsymbol{A},$$

$$\boldsymbol{c_2} = \boldsymbol{F} \cdot \boldsymbol{b} \oplus \begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix} \cdot m$$

where $\boldsymbol{F} := \begin{bmatrix} \boldsymbol{f_1} \\ \dots \\ \boldsymbol{f_q} \end{bmatrix}$, $\boldsymbol{f_i} \leftarrow \mathrm{Ber}_\tau^q$

Partial Decryption Phase

Receiver $\mathtt{R_i}$

Receiver $\mathtt{R_j}$

$\boldsymbol{d_i} \leftarrow \texttt{ThLPN.Pdec}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_i})$

Partial Decryption Phase

Receiver $\mathtt{R_i}$ Receiver $\mathtt{R_j}$

$\boldsymbol{d_i} \leftarrow \color{red}{\mathtt{ThLPN.Pdec}}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_i})$

Partial decryption function (Alekhnovich scheme)

$$\boldsymbol{d_i} = \boldsymbol{C_1} \cdot \boldsymbol{s_i} \oplus \boldsymbol{\nu_i}$$

where $\boldsymbol{\nu_i} \leftarrow \mathrm{Ber}_\sigma^q$

Partial Decryption Phase

Receiver $\mathtt{R_i}$ Receiver $\mathtt{R_j}$

$\boldsymbol{d_i} \leftarrow \mathtt{ThLPN.Pdec}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_i}) \xrightarrow{\boldsymbol{d_i}}$

Partial decryption function (Alekhnovich scheme)

$$\boldsymbol{d_i} = \boldsymbol{C_1} \cdot \boldsymbol{s_i} \oplus \boldsymbol{\nu_i}$$

where $\boldsymbol{\nu_i} \leftarrow \mathrm{Ber}_\sigma^q$

Partial Decryption Phase

Receiver $\underline{\mathtt{R_i}}$ | Receiver $\underline{\mathtt{R_j}}$

$\boldsymbol{d_i} \leftarrow \texttt{ThLPN.Pdec}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_i}) \xrightarrow{\boldsymbol{d_i}}$

$\boldsymbol{d_j} \leftarrow \texttt{ThLPN.Pdec}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_j})$

Partial decryption function (Alekhnovich scheme)

$$\boldsymbol{d_i} = \boldsymbol{C_1} \cdot \boldsymbol{s_i} \oplus \boldsymbol{\nu_i}$$

where $\boldsymbol{\nu_i} \leftarrow \mathrm{Ber}_\sigma^q$

Partial Decryption Phase

Receiver $\underline{\mathtt{R_i}}$		Receiver $\underline{\mathtt{R_j}}$
$\boldsymbol{d_i} \leftarrow \texttt{ThLPN.Pdec}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_i})$	$\xrightarrow{\boldsymbol{d_i}}$	
	$\xleftarrow{\boldsymbol{d_j}}$	$\boldsymbol{d_j} \leftarrow \texttt{ThLPN.Pdec}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_j})$

Partial decryption function (Alekhnovich scheme)

$$\boldsymbol{d_i} = \boldsymbol{C_1} \cdot \boldsymbol{s_i} \oplus \boldsymbol{\nu_i}$$

where $\boldsymbol{\nu_i} \leftarrow \mathrm{Ber}_\sigma^q$

Partial Decryption Phase

Receiver $\underline{\mathtt{R_i}}$ | Receiver $\underline{\mathtt{R_j}}$

$\boldsymbol{d_i} \leftarrow \mathtt{ThLPN.Pdec}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_i})$ $\xrightarrow{\boldsymbol{d_i}}$

$\xleftarrow{\boldsymbol{d_j}}$ $\boldsymbol{d_j} \leftarrow \mathtt{ThLPN.Pdec}(\boldsymbol{C_1}, \boldsymbol{c_2}, \boldsymbol{s_j})$

Finish decryption

- Each receiver indipendently computes the vector

$$\boldsymbol{d} = \boldsymbol{c_2} \bigoplus_{i \in I} (\boldsymbol{d_i}) = \boldsymbol{F} \cdot \boldsymbol{e} \oplus \begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix} \cdot m \bigoplus_{i \in I} (\boldsymbol{\nu_i}) \, .$$

- the bit in the vector $\boldsymbol{d}$ that is in majority is separately chosen by each receiver as the plaintext m

Protocol Security Analysis

Semi-honest model

We make the following two assumptions:

1. The semi-honest party will indeed toss a fair coin
2. The semi-honest party will send all messages as instructed by the protocol

Security

- **Encryption:** from the Alekhnovich's scheme security
- **Decryption:** from the LPN hardness assumption, as each $\mathtt{R}_i$ is generating LPN samples

$$\boldsymbol{d_i} = \boldsymbol{C_1} \cdot \boldsymbol{s_i} \oplus \boldsymbol{\nu_i}$$

Relaxed Semi-honest model

- Semi-honest model not so realistic (*replay attacks* may occur)
- Problem: if the same message is encrypted multiple times then it is possible to recover information about the secret key from the ciphertexts

Possible solutions

1. implement the receivers as *stateful* machines (not good in resource-constrained devices)
2. make use of pseudorandom functions (i.e. deterministic algorithms that simulate truly random functions, given a "seed")

Commitment Protocols

Commitment protocol

- can be thought as the digital analogue of a sealed envelope
- **Commit:** the sender `S` commit to a message m and the receiver `R` does not learn any information about m (hiding property)
- **Open:** `S` can choose to open the commitment and reveal the content m, but no other value (binding property)

Our contribution

We presented a commitment protocol

- based on the commitment protocol by Jain et al
- based on Exact-LPN problem (where $\mathbf{wt}(\boldsymbol{e}) = w$)
- not in a *common reference string (CRS)* model

The commitment protocol

Setup Phase

In order to commit a message $\boldsymbol{m} \in \mathbb{Z}_2^k$ where $k \in \Theta(\ell + v)$
We let $\boldsymbol{A'} \xleftarrow{R} \mathbb{Z}_2^{k\times\ell}$ and $\boldsymbol{A''} \xleftarrow{R} \mathbb{Z}_2^{k\times v}$. We state $\boldsymbol{A} = [\boldsymbol{A'} \| \boldsymbol{A''}] \in \mathbb{Z}_2^{k\times(\ell+v)}$ as the common reference string (CRS). Finally, we set $w = \lfloor \tau k \rceil$.

Commitment phase

Sender $\underline{\mathtt{S}}$		Receiver $\underline{\mathtt{R}}$
chooses $\boldsymbol{r} \xleftarrow{R} \mathbb{Z}_2^{\ell}$, $\boldsymbol{e} \in \mathbb{Z}_2^k$ s.t. $\boldsymbol{wt}(\boldsymbol{e}) = w$		
computes $\boldsymbol{c} = \boldsymbol{A}(\boldsymbol{r}\|\boldsymbol{m}) \oplus \boldsymbol{e}$	$\xrightarrow{\quad \boldsymbol{c} \quad}$	

Opening phase

computes $\boldsymbol{d} = (\boldsymbol{m'}, \boldsymbol{r'})$	$\xrightarrow{\quad \boldsymbol{d} \quad}$	
		computes $\boldsymbol{e'} = \boldsymbol{c} \oplus \boldsymbol{A}(\boldsymbol{r'}\|\boldsymbol{m'})$
	$\xleftarrow{\quad Yes, No \quad}$	accepts iff $\boldsymbol{wt}(\boldsymbol{e'}) = w$

Problem

We need a trusted third party for the common matrix $\boldsymbol{A} = [\boldsymbol{A'} \| \boldsymbol{A''}]$

Solution:

Setup phase

Sender $\underline{\mathtt{S}}$		Receiver $\underline{\mathtt{R}}$
chooses $\boldsymbol{A'} \xleftarrow{R} \mathbb{Z}_2^{k \times \ell}$	$\xrightarrow{\boldsymbol{A'}}$	
	$\xleftarrow{\boldsymbol{A''}}$	chooses $\boldsymbol{A''} \xleftarrow{R} \mathbb{Z}_2^{k \times v}$

The Commitment and Opening phases are the same as in the original scheme

Theorem

Our commitment scheme is statistically binding and computationally hiding

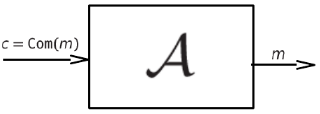
Proof

Statistically binding

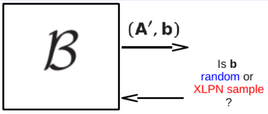
even if S is computationally unbounded she cannot cheat with probability greater than 2^{-k}

Computationally hiding

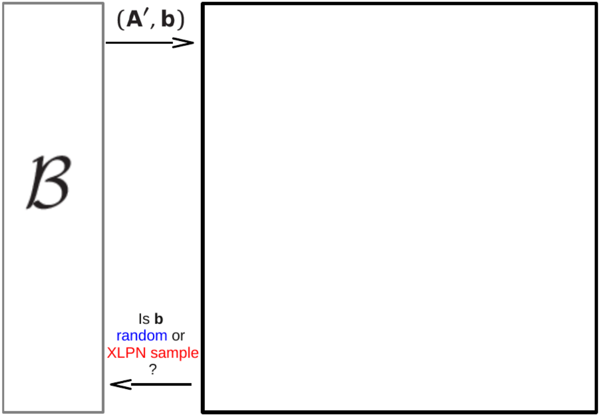
- proof for reduction (single bit message)
- we assume that $\mathcal{A}$ is able to break the commitment scheme

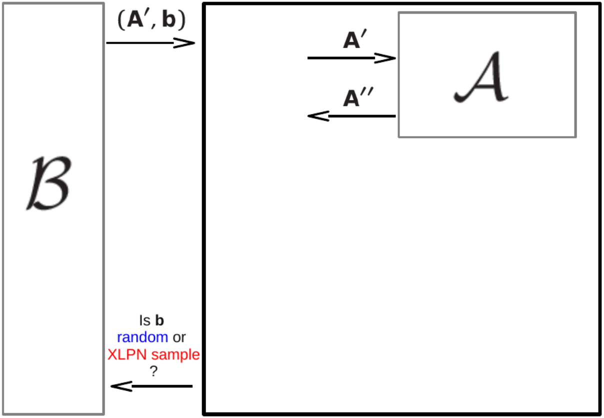


- Let $\mathcal{B}$ an oracle

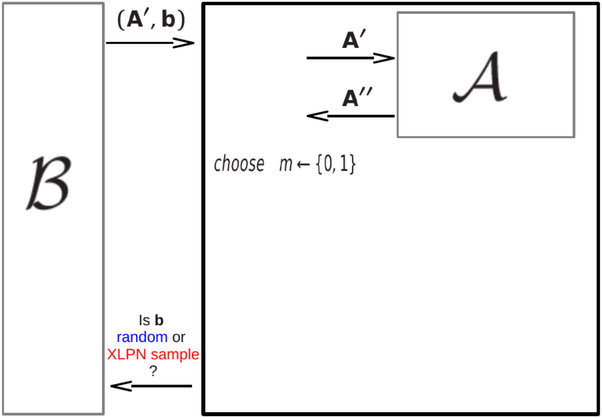


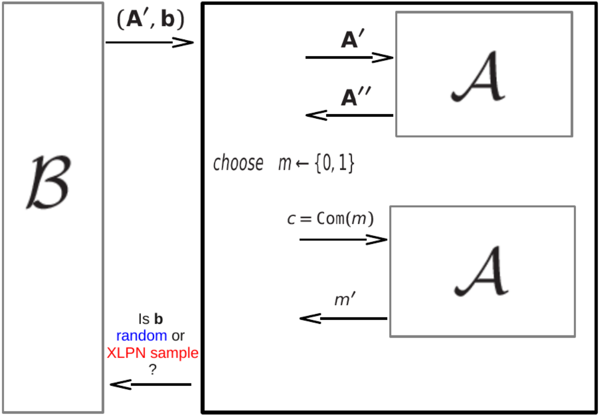
where $\boldsymbol{b} = \begin{cases} \text{random} & w.p.\ 1/2 \\ \boldsymbol{A's} \oplus \boldsymbol{e} & w.p.\ 1/2 \end{cases}$


$(\mathbf{A}', \mathbf{b})$
$\mathcal{B}$
Is **b**
random or
XLPN sample
?

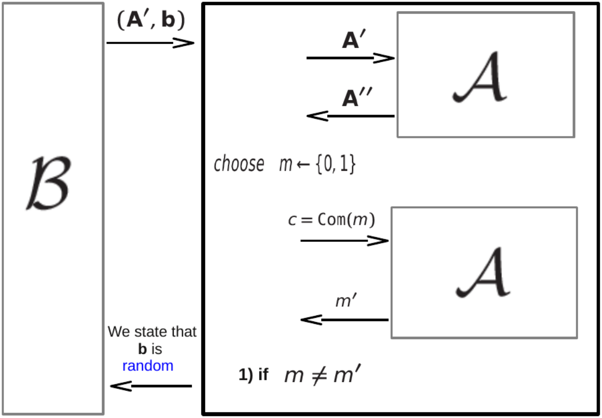


$(\mathbf{A}', \mathbf{b})$
$\mathbf{A}'$
$\mathbf{A}''$
$\mathcal{A}$
$\mathcal{B}$
Is $\mathbf{b}$
random or
XLPN sample
?


$(\mathbf{A}', \mathbf{b})$
$\mathbf{A}'$
$\mathbf{A}''$
$\mathcal{A}$
$\mathcal{B}$
choose $m \leftarrow \{0,1\}$
Is **b**
random or
XLPN sample
?



(A′, b)
A′
A″
$\mathcal{A}$
$\mathcal{B}$
choose $m \leftarrow \{0,1\}$
$c = \mathrm{Com}(m)$
$\mathcal{A}$
m'
Is **b**
random or
XLPN sample
?

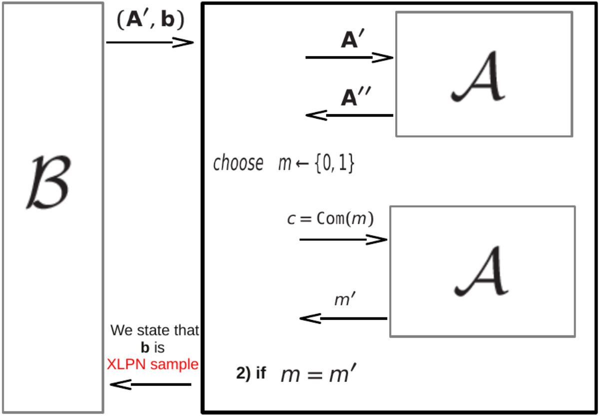


case 1)

$\boldsymbol{b}$ is random $\Rightarrow$ $\boldsymbol{c} = \boldsymbol{b} \oplus \boldsymbol{A}''m$ is a onetime-pad encryption

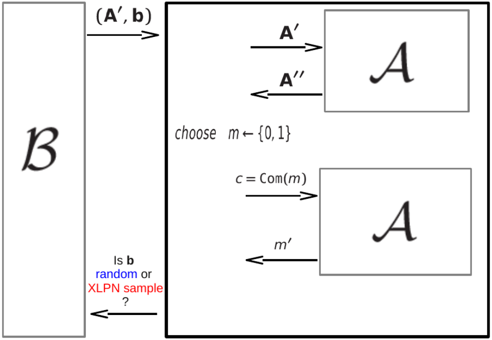
$\Rightarrow \mathcal{A}$ guesses w.p. $\frac{1}{2}$

Exact-LPN hardness $\Rightarrow$ Hiding commitment



case 2)

$\boldsymbol{b}$ is a Exact-LPN sample $\Rightarrow$ $\boldsymbol{c}$ is a well formed commitment
$\Rightarrow \mathcal{A}$ guesses w.p. 1 (by hypothesis)



case 1) and 2)

Let E = the reduction breaks the Exact-LPN problem,

$$\begin{aligned}\Pr(E) &= \Pr\left(E \middle|\, \boldsymbol{b} = \boldsymbol{A}'\boldsymbol{s} \oplus \boldsymbol{e}\right) \cdot \Pr\left(\boldsymbol{b} = \boldsymbol{A}'\boldsymbol{s} \oplus \boldsymbol{e}\right) + \Pr\left(E \middle|\, \boldsymbol{b} \text{ is random}\right) \cdot \Pr\left(\boldsymbol{b} \text{ is random}\right) \\ &= 1 \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4} \gg 2^{-k}\end{aligned}$$

Exact-LPN hardness $\Rightarrow$ Hiding commitment

Choice of parameters

According to

Levieil, Éric and Fouque, Pierre-Alain
An Improved LPN Algorithm
Springer Berlin Heidelberg, 2006

we choose $\ell = 768$ and noise rate $\tau = \frac{1}{8} \Rightarrow 2^{90}$ bytes of memory to solve LPN problem

Conclusions and Open Problems

LPN open problems

- relation between standard LPN and some variants
- LPN with noise rate τ imply anything about LPN with $\tau' < \tau$? Is there a threshold?
- how to get some basic primitives from standard LPN?

Our contribution

- study the security of our Threshold Public-Key Encryption scheme in the *malicious model*
- find statistically hiding commitments
- find efficient statistically binding commitments