On the Learning Parity with Noise Problem

Luca Melis

Università degli Studi di Firenze

Århus Universitet

22 April 2013

Advisors:
Prof. Alessandro Piva
Prof. Fabrizio Argenti

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

Scenario

Cryptography schemes

- address 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
     e.g. *factoring and discrete-logarithm broken by Shor's algorithm*
  2. proof of security becomes unuseful (hypothesis is not valid)

Scenario

Cryptography schemes

- address 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
     e.g. *factoring and discrete-logarithm broken by Shor's algorithm*
  2. proof of security becomes unuseful (hypothesis is not valid)

Scenario

Cryptography schemes

- address 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
     e.g. ***factoring*** *and* ***discrete-logarithm*** *broken by Shor's algorithm*
  2. proof of security becomes unuseful (hypothesis is not valid)

Scenario

Cryptography schemes

- address 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 e.g. ***factoring*** *and* ***discrete-logarithm*** *broken by Shor's algorithm*
  2. proof of security becomes unuseful (hypothesis is not valid)

Scenario

Cryptography schemes

- address 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
     e.g. ***factoring*** *and* ***discrete-logarithm*** *broken by Shor's algorithm*
  2. proof of security becomes unuseful (hypothesis is not valid)

Post-Quantum cryptography

Schemes that are believed to resist classical & quantum computers

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

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

Post-Quantum cryptography

Schemes that are believed to resist classical & quantum computers

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

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

Post-Quantum cryptography

Schemes that are believed to resist classical & quantum computers

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

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

Post-Quantum cryptography

Schemes that are believed to resist classical & quantum computers

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

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 $(\boldsymbol{a}_i, b_i)$ from uniform $(\boldsymbol{a}_i, b_i)$

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**: <u>find</u> $\boldsymbol{s} \in \mathbb{Z}_2^\ell$ 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: <u>distinguish</u> (a_i, b_i) from uniform (a_i, b_i)

decisional and search LPN are "polynomially equivalent"

Learning Parity with Noise Problem LPN

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

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

$$\vdots$$

$$\boldsymbol{a_q} \xleftarrow{R} \mathbb{Z}_2^\ell \quad , \quad b_q =< \boldsymbol{a_q} \, , \, \boldsymbol{s} > \oplus e_q$$

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

- **Decision**: <u>distinguish</u> (a_i, b_i) from uniform (a_i, b_i)

Learning Parity with Noise Problem LPN

- Dimension ℓ (security parameter), $q \gg \ell$, $\tau \in \left(0, \frac{1}{2}\right)$
- **Search**: <u>find</u> $\boldsymbol{s} \in \mathbb{Z}_2^\ell$ 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**: <u>distinguish</u> (a_i, b_i) from uniform (a_i, b_i)
- *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 $(\boldsymbol{a_i}, b_i)$
- *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 $(\boldsymbol{a_i}, b_i)$
- *decisional* and *search* LPN are *"polinomially equivalent"*

Learning Parity with Noise Problem LPN

Hardness of LPN

The best known attacks against search LPN problem takes

- $2^{\Theta(\ell/\log \ell)}$ 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

where ℓ is the security parameter

Interesting features

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

Learning Parity with Noise Problem LPN

Hardness of LPN

The best known attacks against search LPN problem takes

- $2^{\Theta(\ell/\log \ell)}$ 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

where ℓ is the security parameter

Interesting features

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

Learning Parity with Noise Problem LPN

Hardness of LPN

The best known attacks against search LPN problem takes

- $2^{\Theta(\ell/\log \ell)}$ 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

where ℓ is the security parameter

Interesting features

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

Learning Parity with Noise Problem LPN

Hardness of LPN

The best known attacks against search LPN problem takes

- $2^{\Theta(\ell/\log \ell)}$ 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

where ℓ is the security parameter

Interesting features

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

Learning Parity with Noise Problem LPN

Hardness of LPN

The best known attacks against search LPN problem takes

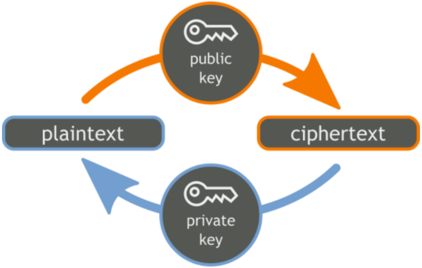
- $2^{\Theta(\ell/\log \ell)}$ having the same number of samples q
- $2^{\Theta(\ell/\log \log \ell)}$ having $q = poly(\ell)$ samples
- $2^{\Theta(\ell)}$ having $q = \Theta(\ell)$ samples

where ℓ is the security parameter

Interesting features

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

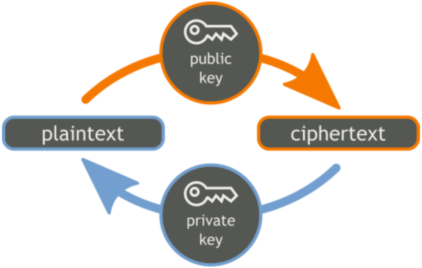
Threshold Public-Key Encryption schemes



Public-key cryptography

- the ability of decrypting or signing is restricted to the owner of the secret key.
- ⇒ only one person has all the power

Threshold Public-Key Encryption schemes



Public-key cryptography

- the ability of decrypting or signing is restricted to the owner of the secret key.
- $\Rightarrow$ only one person has all the power

Threshold Public-Key Encryption schemes

Solution: Threshold PKE

- The secret key is split into shares and each share is given to a group of users.
- users can decrypt or sign only if enough, a *threshold*, cooperate

Our contribution

A Threshold Public-Key Encryption scheme which is:

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

Threshold Public-Key Encryption schemes

Solution: Threshold PKE

- The secret key is split into shares and each share is given to a group of users.
- users can decrypt or sign only if enough, a *threshold*, cooperate

Our contribution

A Threshold Public-Key Encryption scheme which is:

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

Threshold Public-Key Encryption schemes

Solution: Threshold PKE

- The secret key is split into shares and each share is given to a group of users.
- users can decrypt or sign only if enough, a *threshold*, cooperate

Our contribution

A Threshold Public-Key Encryption scheme which is:

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

Threshold Public-Key Encryption schemes

Solution: Threshold PKE

- The secret key is split into shares and each share is given to a group of users.
- users can decrypt or sign only if enough, a *threshold*, cooperate

Our contribution

A Threshold Public-Key Encryption scheme which is:

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

Threshold Public-Key Encryption schemes

Solution: Threshold PKE

- The secret key is split into shares and each share is given to a group of users.
- users can decrypt or sign only if enough, a *threshold*, cooperate

Our contribution

A Threshold Public-Key Encryption scheme which is:

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

Alekhnovich PKE scheme

Key Generation

The receiver R 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 <u>S</u>		Receiver <u>R</u>
choose a vector $\boldsymbol{f} \leftarrow \mathrm{Ber}_{\tau}^{q}$		
compute $\boldsymbol{u} = \boldsymbol{f} \cdot \boldsymbol{A}$		
$c = \langle \boldsymbol{f}, \boldsymbol{b} \rangle \oplus m$	$\xrightarrow{(\boldsymbol{u}, c)}$	

Alekhnovich PKE scheme

Key Generation

The receiver R 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 $m \in \mathbb{Z}_2$

$$\begin{array}{ccc}
\text{Sender } \underline{\mathtt{S}} & & \text{Receiver } \underline{\mathtt{R}} \\
\text{choose a vector } \boldsymbol{f} \leftarrow \mathrm{Ber}_\tau^q & & \\
\text{compute } \boldsymbol{u} = \boldsymbol{f} \cdot \boldsymbol{A} & & \\
c = \langle \boldsymbol{f}, \boldsymbol{b} \rangle \oplus m & \xrightarrow{\quad (\boldsymbol{u}, c) \quad} &
\end{array}$$

Alekhnovich PKE scheme

Key Generation

The receiver $\mathtt{R}$ 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 $\underline{\mathtt{S}}$		Receiver $\underline{\mathtt{R}}$
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 $\mathtt{R}$ computes $d = c \oplus \langle \boldsymbol{s}, \boldsymbol{u} \rangle$

Alekhnovich PKE scheme

Key Generation

The receiver R 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 <u>S</u> Receiver <u>R</u>

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

ThPKE: Protocol phases

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

ThPKE: Protocol phases

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

ThPKE: Protocol phases

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

Key Generation

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

ThPKE: Protocol phases

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

Key Generation

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

ThPKE: Protocol phases

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

Key Generation

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

ThPKE: Protocol phases

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

ThPKE: Protocol phases

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

Key Assembly

The combined public key is the pair $(\boldsymbol{A}, \boldsymbol{b})$, where

$$\boldsymbol{b} = \bigoplus_{i \in I} \boldsymbol{b}_i$$

and I is the users subset

ThPKE: Protocol phases

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

ThPKE: Protocol phases

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

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

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

ThPKE: Protocol phases

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

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)

$$\boldsymbol{C_1} = \boldsymbol{F} \cdot \boldsymbol{A},\ \boldsymbol{c_2} = \boldsymbol{F} \cdot \boldsymbol{b} \oplus \underbrace{\begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix}}_{q} \cdot m \qquad \text{where } \boldsymbol{F} := \begin{bmatrix} \boldsymbol{f_1} \\ \dots \\ \boldsymbol{f_q} \end{bmatrix}, \boldsymbol{f_i} \leftarrow \mathrm{Ber}_\tau^q$$

ThPKE: Protocol phases

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

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

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

Encryption function (Alekhnovich scheme)

$$\boldsymbol{C_1} = \boldsymbol{F} \cdot \boldsymbol{A},\ \boldsymbol{c_2} = \boldsymbol{F} \cdot \boldsymbol{b} \oplus \underbrace{\begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix}}_{q} \cdot m \qquad \text{where } \boldsymbol{F} := \begin{bmatrix} \boldsymbol{f_1} \\ \dots \\ \boldsymbol{f_q} \end{bmatrix}, \boldsymbol{f_i} \leftarrow \mathrm{Ber}_\tau^q$$

ThPKE: Protocol phases

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

ThPKE: Protocol phases

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

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

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

ThPKE: Protocol phases

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

Receiver $\mathtt{R_i}$

Receiver $\mathtt{R_j}$

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

Partial decryption function (Alekhnovich scheme)

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

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

ThPKE: Protocol phases

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

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

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

Partial decryption function (Alekhnovich scheme)

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

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

ThPKE: Protocol phases

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

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

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

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

Partial decryption function (Alekhnovich scheme)

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

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

ThPKE: Protocol phases

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

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

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

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

Partial decryption function (Alekhnovich scheme)

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

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

ThPKE: Protocol phases

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

ThPKE: Protocol phases

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

Finish decryption

- Each receiver indipendently computes the vector

$$\boldsymbol{d} = \boldsymbol{c_2} \bigoplus_{i \in I} (\boldsymbol{d}_i) = \boldsymbol{F} \cdot \boldsymbol{e} \oplus \underbrace{\begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix}}_{q} \cdot m \bigoplus_{i \in I} (\boldsymbol{\nu}_i) .$$

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

ThPKE: Protocol phases

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

Finish decryption

- Each receiver indipendently computes the vector

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

- 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

A semi-honest party:

1. Follows the protocol properly
2. Keeps a record of all its intermediate computations

Security

- **Encryption:** from the Alekhnovich's scheme security
- **Decryption:** from the LPN hardness assumption, as each P_i is generating LPN samples

$$d_i = C_1 \cdot s_i \oplus \nu_i$$

Protocol Security Analysis

Semi-honest model

A semi-honest party:

1. Follows the protocol properly
2. Keeps a record of all its intermediate computations

Security

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

$$\boldsymbol{d}_i = \boldsymbol{C}_1 \cdot \boldsymbol{s}_i \oplus \boldsymbol{\nu}_i$$

Protocol Security Analysis

Semi-honest model

A semi-honest party:

1. Follows the protocol properly
2. Keeps a record of all its intermediate computations

Security

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

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

Protocol Security Analysis

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

Proposed solutions

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

Protocol Security Analysis

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

Proposed 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")

Protocol Security Analysis

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

Proposed 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”)

Protocol Security Analysis

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

Proposed 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”)

Conclusions and Open Problems

Our contribution

- study the security of our Threshold Public-Key Encryption scheme in the *malicious model*

LPN open problems

- relation between standard LPN and some variants
- Does 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?

Conclusions and Open Problems

Our contribution

- study the security of our Threshold Public-Key Encryption scheme in the *malicious model*

LPN open problems

- relation between standard LPN and some variants
- Does 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?

Conclusions and Open Problems

Our contribution

- study the security of our Threshold Public-Key Encryption scheme in the *malicious model*

LPN open problems

- relation between standard LPN and some variants
- Does 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?

Conclusions and Open Problems

Our contribution

- study the security of our Threshold Public-Key Encryption scheme in the *malicious model*

LPN open problems

- relation between standard LPN and some variants
- Does 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?