NTRUReEncrypt

An Efficient Proxy Re-Encryption Scheme based on NTRU

David Nuñez, Isaac Agudo, and Javier Lopez

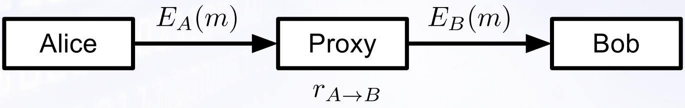
Network, Information and Computer Security Laboratory (NICS Lab)
Universidad de Málaga, Spain
Email: dnunez@lcc.uma.es

ACM AsiaCCS 2015 – Singapore

1. Proxy Re-Encryption
2. NTRU
3. NTRUReEncrypt
4. PS-NTRUReEncrypt
5. Experimental results
6. Conclusions

Proxy Re-Encryption: Overview

- A Proxy Re-Encryption scheme is a public-key encryption scheme that permits a proxy to transform ciphertexts under Alice's public key into ciphertexts under Bob's public key
- The proxy needs a re-encryption key $r_{A \to B}$ to make this transformation possible, generated by the delegating entity
- Proxy Re-Encryption enables delegation of decryption rights



Syntax of Bidirectional Proxy Re-Encryption

Definition. A bidirectional proxy re-encryption scheme is a tuple of algorithms (Setup, KeyGen, ReKeyGen, Enc, ReEnc, Dec):

- KeyGen() $\rightarrow (pk_A, sk_A)$
- ReKeyGen$(sk_A, sk_B) \rightarrow rk_{A\rightarrow B}$
- Enc$(pk_A, M) \rightarrow C_A$
- ReEnc$(rk_{A\rightarrow B}, C_A) \rightarrow C_B$
- Dec$(sk_A, C_A) \rightarrow M$

Correctness

Definition: Multihop Correctness. A bidirectional PRE scheme (Setup, KeyGen, ReKeyGen, Enc, ReEnc, Dec) is multihop correct with respect to plaintext space $\mathcal{M}$ if:

- (*Encrypted Ciphertexts*) For all (pk_A, sk_A) output by KeyGen and all messages $M \in \mathcal{M}$, it holds that:

$$\mathsf{Dec}(sk_A, \mathsf{Enc}(pk_A, M)) = M$$

- (*Re-Encrypted Ciphertexts*) For any sequence of pairs (pk_i, sk_i) output by KeyGen, with $0 \leq i \leq N$, all re-encryption keys $rk_{j \to j+1}$ output by ReKeyGen(sk_j, sk_{j+1}), with $j < N$, all messages $M \in \mathcal{M}$, and all ciphertexts C_1 output by Enc(pk_1, M), it holds that:

$$\mathsf{Dec}(sk_N, \mathsf{ReEnc}(rk_{N-1 \to N}, ...\mathsf{ReEnc}(rk_{1 \to 2}, C_1))) = M$$

Bidirectional CPA-security game

Let us assume:

- k is the security parameter
- $\mathcal{A}$ is a polynomial-time adversary
- $\mathcal{H}, \mathcal{C}$ are the sets of indices of honest and corrupt users

The IND-CPA game consists of an execution of $\mathcal{A}$ with the following oracles, which can be invoked multiple times in any order, subject to the constraints below:

Bidirectional CPA-security game

Phase 0:

- The challenger obtains global parameters $params \leftarrow \text{Setup}(1^k)$ and initializes sets $\mathcal{H}, \mathcal{C}$ to $\emptyset$.
- The challenger generates the public key pk^* of target user i^*, adds i^* to $\mathcal{H}$, and sends pk^* to the adversary.

Phase 1:

- Uncorrupted key generation $\mathcal{O}_{honest}$: On input an index i, where $i \notin \mathcal{H} \cup \mathcal{C}$, the oracle obtains a new keypair $(pk_i, sk_i) \leftarrow \text{KeyGen}()$ and adds index i to $\mathcal{H}$. The adversary receives pk_i.
- Corrupted key generation $\mathcal{O}_{corrupt}$: On input an index i, where $i \notin \mathcal{H} \cup \mathcal{C}$, the oracle obtains a new keypair $(pk_i, sk_i) \leftarrow \text{KeyGen}()$ and adds index i to $\mathcal{C}$. The adversary receives (pk_i, sk_i).

Bidirectional CPA-security game

Phase 2:

- Re-encryption key generation $\mathcal{O}_{rkgen}$: On input (i, j), where $i \neq j$, and either $i, j \in \mathcal{H}$ or $i, j \in \mathcal{C}$, the oracle returns $rk_{i \to j} \leftarrow \text{ReKeyGen}(sk_i, sk_j)$.
- Challenge oracle $\mathcal{O}_{challenge}$: This oracle can be queried only once. On input (M_0, M_1), the oracle chooses a bit $b \leftarrow \{0, 1\}$ and returns the challenge ciphertext $C^* \leftarrow \text{Enc}(pk^*, M_b)$, where pk^* corresponds to the public key of target user i^*.

Phase 3:

- Decision: $\mathcal{A}$ outputs guess $b' \in \{0, 1\}$. $\mathcal{A}$ wins the game if and only if $b' = b$.

Other remarks

- Static corruption model
- We only allow queries to $\mathcal{O}_{rkgen}$ where users are either both corrupt or both honest
- Otherwise, these queries would corrupt honest users
- Re-encryption oracle is not necessary in CPA

NTRUEncrypt: Overview

- Originally proposed by Hoffstein, Pipher and Silverman
- One of the first PKE schemes based on lattices
- NTRU Encryption is very efficient, orders of magnitude faster than other PKE schemes
- IEEE Standard 1363.1-2008 and ANSI X9.98-2010
- It is conjectured to be based on hard problems over lattices
- Post-quantum cryptography
- It lacks a formal proof in the form of a reduction to a hard problem (i.e. not provably-secure)

NTRUEncrypt: Basics

- Defined over the quotient ring $\mathcal{R}_{NTRU} = \mathbb{Z}[x]/(x^n - 1)$, where n is a prime parameter
- Other parameters of NTRU:
  - Integer q, which is a small power of 2 of the same order of magnitude than n
  - Small polynomial $p \in \mathcal{R}_{NTRU}$, which usually takes values $p = 3$ or $p = x + 2$
- In general, operations over polynomials will be performed in $\mathcal{R}_{NTRU}/q$ or $\mathcal{R}_{NTRU}/p$

NTRUEncrypt: Key Generation

Private key: $sk = f \in \mathcal{R}_{NTRU}$

- f is chosen at random, with a determined number of coefficients equal to 0, -1, and 1
- f must be invertible in $\mathcal{R}_{NTRU}/q$ and $\mathcal{R}_{NTRU}/p \Rightarrow f_q^{-1}, f_p^{-1}$
- For efficiency, f can be chosen to be $1 \mod p$

Public key: $pk = h = p \cdot g \cdot f_q^{-1} \mod q$

- $g \in \mathcal{R}_{NTRU}$ is chosen at random

NTRUEncrypt: Encryption and Decryption

Encryption:

- plaintext M from message space $\mathcal{R}_{NTRU}/p$
- ciphertext $C = h \cdot s + M \mod q$
- noise term s is a small random polynomial in $\mathcal{R}_{NTRU}$

Decryption:

- Compute $C' = f \cdot C \mod q$
- Compute $m = f_p^{-1} \cdot C' \mod p$

Why does it work?

- $C' = f \cdot (p \cdot g \cdot f_q^{-1} \cdot s + M) \mod q = p \cdot g \cdot s + f \cdot M \mod q$
- This equation holds if $f \cdot C$ is "*small enough*"
- $f_p^{-1} \cdot (\cancel{p \cdot g \cdot s} + f \cdot M) \mod p = f_p^{-1} \cdot f \cdot M \mod p = M$
- If $f = 1 \mod p$, then the last step is simply $m = C' \mod p$

NTRUReEncrypt

- We extended NTRUEncrypt to support re-encryption $\Rightarrow$ **NTRUReEncrypt**
- New requirement: secret polynomial $f = 1 \mod p$
- Not for efficiency reasons, but necessary to correctly decrypt re-encrypted ciphertexts

NTRUReEncrypt: Key Generation

Private key: $sk_A = f_A \in \mathcal{R}_{NTRU}$

- f_A is chosen at random, with a determined number of coefficients equal to 0, -1, and 1
- f_A must be invertible in $\mathcal{R}_{NTRU}/q \Rightarrow f_A^{-1}$
- Since f is chosen to be $1 \mod p$, its inverse $\mod p$ is not necessary

Public key: $pk_A = h_A = p \cdot g_A \cdot f_A^{-1} \mod q$

- $g_A \in \mathcal{R}_{NTRU}$ is chosen at random

NTRUReEncrypt: Encryption and Decryption

Encryption:

- plaintext M from message space $\mathcal{R}_{NTRU}/p$
- ciphertext $C_A = h_A \cdot s + M \mod q$
- noise term s is a small random polynomial in $\mathcal{R}_{NTRU}$

Decryption:

- Compute $C'_A = f \cdot C_A \mod q$
- Compute $m = C'_A \mod p$

NTRUReEncrypt: Re-Encryption Key Generation

Re-Encryption Key Generation:

- Input: secret keys $sk_A = f_A$ and $sk_B = f_B$
- The re-encryption key between users A and B is

$$rk_{A \to B} = sk_A \cdot sk_B^{-1} = f_A \cdot f_B^{-1}$$

- Three-party protocol, so neither A, B nor the proxy learns any secret key.
  - A selects a random $r \in \mathcal{R}_{NTRU}/q$
  - A sends $r \cdot f_A \mod q$ to B and r to the proxy
  - B sends $r \cdot f_A \cdot f_B^{-1} \mod q$ to the proxy
  - The proxy computes $rk_{A \to B} = f_A \cdot f_B^{-1} \mod q$

NTRUReEncrypt: Re-Encryption

Re-Encryption

- Input: a re-encryption key $rk_{A\to B}$ and a ciphertext C_A
- Samples a random polynomial $e \in \mathcal{R}_{NTRU}$
- Output re-encrypted ciphertext

$$C_B = C_A \cdot rk_{A\to B} + pe$$

- The noise e prevents B from extracting A's private key

NTRUReEncrypt: Re-Encryption

Why does it work?

- Re-encrypted ciphertext:

$$\begin{aligned} C_B &= C_A \cdot rk_{A\to B} + p \cdot e \mod q \\ &= (p \cdot g \cdot f_A^{-1} \cdot s + M) \cdot f_A \cdot f_B^{-1} + p \cdot e \mod q \\ &= p \cdot g \cdot f_B^{-1} \cdot s + f_A \cdot f_B^{-1} \cdot M + p \cdot e \mod q \end{aligned}$$

- Decrypting a re-encrypted ciphertext:

$$\begin{aligned} f_B \cdot C_B \mod p &= \cancel{(p \cdot g \cdot s + p \cdot e)} + f_A \cdot M \mod p \\ &= f_A \cdot M \mod p \\ &= M \end{aligned}$$

NTRUReEncrypt: Re-Encryption

Limited Multihop:

- The scheme does not support unlimited re-encryptions
- The noise e added during the re-encryption accumulates on each hop, until eventually, decryption fails
- This depends heavily on the choice of parameters

NTRUReEncrypt: Analysis

Computational costs:

- The core operation in NTRU is the multiplication of polynomials
- It can be done in $O(n \log n)$ time using the Fast Fourier Transform (FFT)
- Encryption, decryption and re-encryption only need a **single multiplication**

NTRUReEncrypt: Analysis

Space costs:

- Keys and ciphertexts are polynomials of size $O(n \cdot \log_2 q)$ bits
- Ciphertext expansion is $O(\log_2 q)$
- Other lattice-based schemes have ciphertexts of size $O(n^2)$

Table : Comparison of space costs (in KB)

Size	Aono et al.	NTRUReEncrypt
Public keys	60.00	1.57
Secret key	60.00	1.57
Re-Encryption key	2520.00	1.57
Ciphertext	0.66	1.57

NTRUReEncrypt: Analysis

- Bidirectional: Given $rk_{A \to B} = f_A f_B^{-1}$, one can easily compute

$$rk_{B \to A} = (rk_{A \to B})^{-1} = f_B f_A^{-1}$$

- Limited multihop
- Not collusion-safe: Secret keys can be extracted from the re-encryption key if the proxy colludes with a user involved

$$f_A = rk_{B \to A} \cdot f_B$$

- This is common in interactive bidirectional PRE schemes

PS-NTRUReEncrypt

- A second proxy re-encryption scheme, called **PS-NTRUReEncrypt**
- Provable secure under the **Ring-LWE assumption**
- Extends the NTRU variant proposed by Stehlé and Steinfeld [Eurocrypt'11], which is proven **IND-CPA secure**

Preliminaries

- $\Phi(x)$ is the cyclotomic polynomial $x^n + 1$, with n a power of 2
- q is a prime integer such that $q = 1 \mod 2n$
- $\mathcal{R}$ is the ring $\mathbb{Z}[x]/\Phi(x)$
- $\mathcal{R}_q = \mathcal{R}/q = \mathbb{Z}_q[x]/\Phi(x)$
- $\mathcal{R}_q^\times$ is the set of invertible elements of $\mathcal{R}_q$

The Ring-LWE problem

- The **Ring Learning With Errors** (Ring-LWE) problem is a hard decisional problem based on lattices
- We use a variant of this problem proposed by Stehlé and Steinfeld.
- $s \in \mathcal{R}_q$ and ψ a distribution over $\mathcal{R}_q^{\times}$
- $A_{s,\psi}^{\times}$ is the distribution that samples pairs of the form (a, b)
  - a is chosen uniformly from $\mathcal{R}_q^{\times}$
  - $b = a \cdot s + e$, for some e sampled from ψ
- The Ring-LWE problem is to distinguish distribution $A_{s,\psi}^{\times}$ from a uniform distribution over $\mathcal{R}_q^{\times} \times \mathcal{R}_q$
- The Ring-LWE assumption is that this problem is computationally infeasible

PS-NTRUReEncrypt: Setup and Key Generation

Setup:

- Global parameters: $(n, q, p, \alpha, \sigma)$

Key Generation:

- $D_{\mathbb{Z}^n,\sigma}$ is a Gaussian distribution over $\mathbb{Z}^n$ with standard deviation σ
- The keys are computed as follows:
  1. Sample f' from $D_{\mathbb{Z}^n,\sigma}$
     Let $f_A = 1 + p \cdot f'$; if $(f_A \mod q) \notin \mathcal{R}_q^\times$, resample
  2. Sample g_A from $D_{\mathbb{Z}^n,\sigma}$; if $(g_A \mod q) \notin \mathcal{R}_q^\times$, resample
  3. Compute $h_A = p \cdot g_A \cdot f_A^{-1}$
  4. Return secret key $sk_A = f_A$ and $pk_A = h_A$

PS-NTRUReEncrypt: Encryption and Decryption

Encryption:

- Input: public key pk_A and message $M \in \mathcal{M}$
- Sample noise polynomials s, e from a distribution Ψ_α
- Output ciphertext:

$$C_A = h_A s + pe + M \in \mathcal{R}_q$$

Decryption:

- Input: secret key $sk_A = f_A$ and ciphertext C_A
- Compute $C'_A = C_A \cdot f_A$
- Output the message $M = (C'_A \mod p) \in \mathcal{M}$

PS-NTRUReEncrypt: Re-Encryption Key Generation and Re-Encryption

Re-Encryption Key Generation:

- Input: secret keys $sk_A = f_A$ and $sk_B = f_B$
- The re-encryption key between users A and B is

$$rk_{A\to B} = sk_A \cdot sk_B^{-1} = f_A \cdot f_B^{-1}$$

Re-Encryption:

- Input: a re-encryption key $rk_{A\to B}$ and a ciphertext C_A
- Samples a random polynomial e' from a distribution Ψ_α
- Output re-encrypted ciphertext

$$C_B = C_A \cdot rk_{A\to B} + pe'$$

Multihop Correctness

Ciphertext re-encrypted N times:

$$\begin{aligned} C_N &= pg_0 f_N^{-1} s + pe_0 f_0 f_N^{-1} + pe_1 f_1 f_N^{-1} + \dots \\ &\quad + pe_{N-1} f_{N-1} f_N^{-1} + pe_N + M f_0 f_N^{-1} \\ &= pg_0 f_N^{-1} s + \left[\sum_{i=0}^{N-1} pe_i f_i f_N^{-1} \right] + pe_N + M f_0 f_N^{-1} \end{aligned}$$

When decrypting C_N (*assuming no decryption failures*):

$$C'_N = C_N \cdot f_N = pg_0 s + \left[\sum_{i=0}^{N} pe_i f_i \right] + M f_0$$

Since, $f_0 = 1 \mod p$ and $pg_0 s = pe_i f_i = 0 \mod p$, then:

$$C'_N \mod p = M$$

Experimental setting

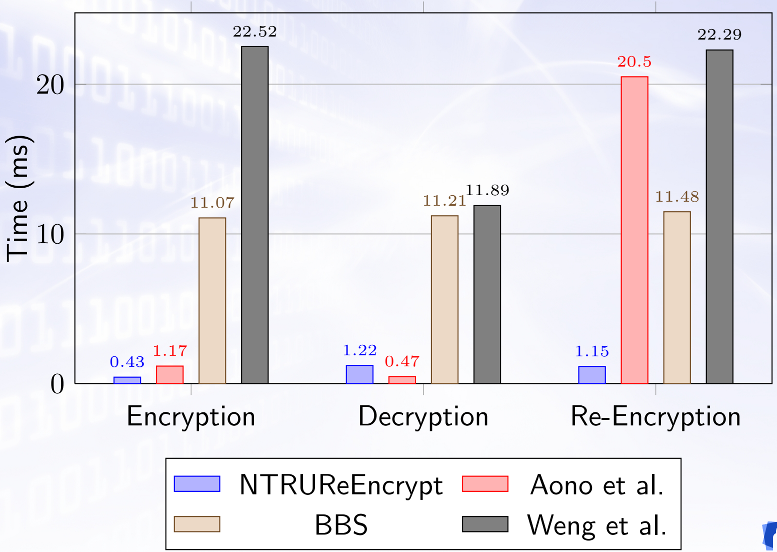
- Implementation of our proposals:
  - NTRUReEncrypt is implemented on top of an available open-source Java implementation of NTRU
  - PS-NTRUReEncrypt was coded from scratch, using the Java Lattice-Based Cryptography (jLBC) library
- Execution enviroment: Intel Core 2 Duo @ 2.66 GHz

Performance of NTRUReEncrypt

Table : Computation time (in ms) and number of hops of NTRUReEncrypt for different parameters

Parameters	Enc.	Dec.	Re-Enc.	# Hops
(439, no, 128)	0.64	0.30	0.24	5
(439, yes, 128)	0.16	0.30	0.23	5
(1087, no, 256)	1.39	1.25	1.05	21
(1087, yes, 256)	0.48	1.26	1.07	15
(1171, no, 256)	0.80	1.12	1.14	21
(1171, yes, 256)	**0.43**	**1.22**	**1.15**	**14**
(1499, no, 256)	0.74	1.78	1.73	50
(1499, yes, 256)	0.32	1.67	1.66	42

Comparison of NTRUReEncrypt to other schemes



Comparison of NTRUReEncrypt to other schemes

Table : Computation time of several proxy re-encryption schemes (in ms)

Scheme	Enc.	Dec.	Re-Enc.
NTRUReEncrypt	0.43	1.22	1.15
Aono et al	1.17	0.47	20.5
BBS	11.07	11.21	11.48
Weng et al	22.52	11.89	22.29
Ateniese et al	22.76	13.76	83.52
Libert and Vergnaud	155.27	443.87	386.93

Performance of PS-NTRUReEncrypt

Table : Computation time (in ms) and size (in KB) of PS-NTRUReEncrypt for different parameters

n	$\log_2 q$	Enc.	Dec.	Re-Enc.	Size
32	23	0.93	0.99	1.05	0.09
64	28	4.53	4.23	4.32	0.22
128	32	17.28	17.32	17.45	0.50
256	37	80.64	81.045	86.56	1.16
512	41	333.75	334.07	359.54	2.56
1024	46	1333.03	1344.10	1461.46	5.75

Conclusions

- **NTRUReEncrypt** is a highly-efficient proxy re-encryption scheme based on the NTRU cryptosystem
- This scheme is bidirectional and multihop, but not collusion-resistant
- The key strength of this scheme is its performance: outperforms other schemes by an order of magnitude
- Potential improvement with parallelization techniques
- Opens up new practical applications of PRE in constrained environments
- We also propose **PS-NTRUReEncrypt**, a provably-secure variant that is CPA-secure under the Ring-LWE assumption

Future Work

- Achieve CCA-security
- Definition of a unidirectional and collision-resistant scheme
- Fine-tune the parameters of NTRUReEncrypt for decreasing the probability of decryption failures after multiple re-encryptions
- Better bounds for the provably-secure version
- Analysis of the selection of parameters based on best known lattice attacks

Thank you!