Norwegian University of
Science and Technology

CONCRETE NTRU SECURITY AND ADVANCES IN PRACTICAL LATTICE-BASED ELECTRONIC VOTING

Presentation for NaCl

Patrick Hough, **Caroline Sandsbråten**, Tjerand Silde

November 8, 2023

Contents

Contents

Overview

Overview

- We do a generalised analysis of the NTRU Fatigue point to be better able to set good parameters for our voting scheme

Overview

- We do a generalised analysis of the NTRU Fatigue point to be better able to set good parameters for our voting scheme
- We adapt the E-Voting scheme from Aranha et al (CCS 2023) using NTRU to reduce ciphertext size and enc/dec time

Overview

- We do a generalised analysis of the NTRU Fatigue point to be better able to set good parameters for our voting scheme
- We adapt the E-Voting scheme from Aranha et al (CCS 2023) using NTRU to reduce ciphertext size and enc/dec time
- We implement this scheme to obtain timings

Overview

- We do a generalised analysis of the NTRU Fatigue point to be better able to set good parameters for our voting scheme
- We adapt the E-Voting scheme from Aranha et al (CCS 2023) using NTRU to reduce ciphertext size and enc/dec time
- We implement this scheme to obtain timings
- Our resulting scheme is on average faster and smaller in size than previous work, but with a larger proof of boundedness.

Contents

The NTRU problem

Definition

Let $q > 2$ be a prime, d be the ring dimension and $D_{\sigma_{NTRU}}$ be a distribution over R_q. Sample $(f, g) \leftarrow D_{\sigma_{NTRU}}$, reject if f is not invertible in R_q, let $h = g/f \in R_q$

The NTRU problem

Definition

Let $q > 2$ be a prime, d be the ring dimension and $D_{\sigma_{NTRU}}$ be a distribution over R_q. Sample $(f, g) \leftarrow D_{\sigma_{NTRU}}$, reject if f is not invertible in R_q, let $h = g/f \in R_q$

Search-NTRU: given h, recover any rotation $(X^i f, X^i g)$ of (f, g).

The NTRU problem

Definition

Let $q > 2$ be a prime, d be the ring dimension and $D_{\sigma_{NTRU}}$ be a distribution over R_q. Sample $(f, g) \leftarrow D_{\sigma_{NTRU}}$, reject if f is not invertible in R_q, let $h = g/f \in R_q$

Search-NTRU: given h, recover any rotation $(X^i f, X^i g)$ of (f, g).

Decision-NTRU: given h, decide if h is computed as $h = g/f$, or if h is uniformly sampled from R_q.

NTRU Encryption

We make 2 changes from traditional NTRU to achieve perfectly correct decryption:

NTRU Encryption

We make 2 changes from traditional NTRU to achieve perfectly correct decryption:

- Encryption randomness sampled from a bounded distribution

NTRU Encryption

We make 2 changes from traditional NTRU to achieve perfectly correct decryption:

- Encryption randomness sampled from a bounded distribution
- f and g are rejected unless their 2-norm is below some bound

NTRU KeyGen

Definition

$KeyGen(d, p, q, \sigma_{NTRU}, t, \nu)$:

NTRU KeyGen

Definition

$KeyGen(d, p, q, \sigma_{NTRU}, t, \nu)$:

1. Sample f from $D_{\sigma_{NTRU}}$. If $(f \mod q) \notin R_q^\times$ or $f \not\equiv 1 \in R_p$, resample.

NTRU KeyGen

Definition

$KeyGen(d, p, q, \sigma_{NTRU}, t, \nu)$:

1. Sample f from $D_{\sigma_{NTRU}}$. If $(f \mod q) \notin R_q^\times$ or $f \not\equiv 1 \in R_p$, resample.
2. Sample g from $D_{\sigma_{NTRU}}$. If $(g \mod q) \notin R_q^\times$

NTRU KeyGen

Definition

$KeyGen(d, p, q, \sigma_{NTRU}, t, \nu)$:

1. Sample f from $D_{\sigma_{NTRU}}$. If $(f \mod q) \notin R_q^\times$ or $f \not\equiv 1 \in R_p$, resample.
2. Sample g from $D_{\sigma_{NTRU}}$. If $(g \mod q) \notin R_q^\times$
3. If $\|f\|_2 > t \cdot \sqrt{d} \cdot \sigma_{NTRU}$ or $\|g\|_2 > t \cdot \sqrt{d} \cdot \sigma_{NTRU}$, restart.

NTRU KeyGen

Definition

$KeyGen(d, p, q, \sigma_{NTRU}, t, \nu)$:

1. Sample f from $D_{\sigma_{NTRU}}$. If $(f \mod q) \notin R_q^\times$ or $f \not\equiv 1 \in R_p$, resample.
2. Sample g from $D_{\sigma_{NTRU}}$. If $(g \mod q) \notin R_q^\times$
3. If $\|f\|_2 > t \cdot \sqrt{d} \cdot \sigma_{NTRU}$ or $\|g\|_2 > t \cdot \sqrt{d} \cdot \sigma_{NTRU}$, restart.
4. Return secret key $sk = f$, public key $pk = h := g/f$

NTRU Encryption

Definition

$Enc(m \in R_p, pk = h)$:

NTRU Encryption

Definition

$Enc(m \in R_p, pk = h)$:

1. Sample encryption randomness $s, e \leftarrow S_\nu$

NTRU Encryption

Definition

$Enc(m \in R_p, pk = h)$:

1. Sample encryption randomness $s, e \leftarrow S_\nu$
2. $c = p \cdot (hs + e) + m \in R_q$

NTRU Encryption

Definition

$Enc(m \in R_p, pk = h)$:

1. Sample encryption randomness $s, e \leftarrow S_\nu$
2. $c = p \cdot (hs + e) + m \in R_q$
3. Return ciphertext c

NTRU Decryption

Definition

$Dec(c \in R_q, sk = f)$:

NTRU Decryption

Definition

$Dec(c \in R_q, sk = f)$:

1. $m = (f \cdot c \mod q) \mod p$

NTRU Decryption

Definition

$Dec(c \in R_q, sk = f)$:

1. $m = (f \cdot c \mod q) \mod p$
2. Return message m

NTRU Fatigue

Recall: NTRU and the NTRU problem.

NTRU Fatigue

Recall: NTRU and the NTRU problem.

- Intuitively NTRU key recovery is to find f, g given h

NTRU Fatigue

Recall: NTRU and the NTRU problem.

- Intuitively NTRU key recovery is to find f, g given h
- The hardness of this grows exponentially with dimension d

NTRU Fatigue

Recall: NTRU and the NTRU problem.

- Intuitively NTRU key recovery is to find f, g given h
- The hardness of this grows exponentially with dimension d
- BUT...

NTRU Fatigue

- Analysis have led to an attack on what is called *overstretched* NTRU

NTRU Fatigue

- Analysis have led to an attack on what is called *overstretched* NTRU
- Revealing that the NTRU problems becomes much easier to solve when σ is small and $q >> d$

NTRU Fatigue

- Analysis have led to an attack on what is called *overstretched* NTRU
- Revealing that the NTRU problems becomes much easier to solve when σ is small and $q >> d$
- This “point” is referred to as the *fatigue* point.

NTRU Fatigue

- Analysis have led to an attack on what is called *overstretched* NTRU
- Revealing that the NTRU problems becomes much easier to solve when σ is small and $q >> d$
- This “point” is referred to as the *fatigue* point.
- This is done by figuring out at which point it is easier to discover a dense sublattice generated by the secret key using BKZ than to find a secret key in the reduced basis

Our NTRU Fatigue Experiments

Our NTRU Fatigue Experiments

- Previous work by by Ducas and van Woerden found experimentally that the fatigue point was when $q \approx d^{2.484}$ for ternary case NTRU

Our NTRU Fatigue Experiments

- Previous work by by Ducas and van Woerden found experimentally that the fatigue point was when $q \approx d^{2.484}$ for ternary case NTRU

- We repeat their experiments/estimates, adjusting for different values of std. variation σ when sampling secrets from a gaussian

Our NTRU Fatigue Experiments

- Previous work by by Ducas and van Woerden found experimentally that the fatigue point was when $q \approx d^{2.484}$ for ternary case NTRU
- We repeat their experiments/estimates, adjusting for different values of std. variation σ when sampling secrets from a gaussian
- We estimate that the concrete fatigue point of NTRU can be described by: $q = 0.0058 \cdot \sigma^2 \cdot d^{2.484}$, which we also confirm experimentally.

Hardness Results

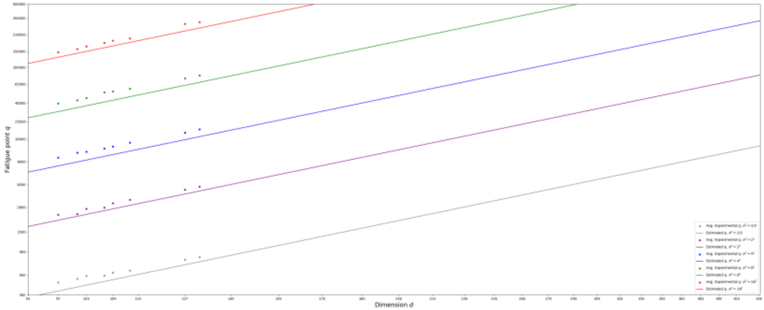


Fig. 2. Average experimental fatigue point q values plotted against estimated fatigue point using progressive BKZ with 8 tours on matrix NTRU instances with variance $\sigma^2 \in \{2/3, 4, 16, 64, 256\}$. The straight colored lines show the estimated values using the (modified) estimator from [DvW21]. The colored dots show the experimental results, where a DSD event has a 50% chance of triggering before an SKR event. The plot is scaled to $\log q$ and $\log d$.

Contents

What is E-Voting?

What is E-Voting?

- For us: employ cryptographic protocols to make secure and anonymous voting possible

What is E-Voting?

- For us: employ cryptographic protocols to make secure and anonymous voting possible

  Important features for e-voting:

What is E-Voting?

- For us: employ cryptographic protocols to make secure and anonymous voting possible

Important features for e-voting:

- End-to-End verifiability

What is E-Voting?

- For us: employ cryptographic protocols to make secure and anonymous voting possible

Important features for e-voting:

- End-to-End verifiability
- Anonymity and Unlinkability

What is E-Voting?

- For us: employ cryptographic protocols to make secure and anonymous voting possible

Important features for e-voting:

- End-to-End verifiability
- Anonymity and Unlinkability
- Integrity

A Voting Scheme

A Voting Scheme

Defined in terms of the algorithms needed for the tasks:

A Voting Scheme

Defined in terms of the algorithms needed for the tasks:

- Election Setup

A Voting Scheme

Defined in terms of the algorithms needed for the tasks:

- Election Setup
- Ballot Casting

A Voting Scheme

Defined in terms of the algorithms needed for the tasks:

- Election Setup
- Ballot Casting
- Ballot Counting

A Voting Scheme

Defined in terms of the algorithms needed for the tasks:

- Election Setup
- Ballot Casting
- Ballot Counting

Therefore, we need algorithms for:

A Voting Scheme

Defined in terms of the algorithms needed for the tasks:

- Election Setup
- Ballot Casting
- Ballot Counting

Therefore, we need algorithms for:

- Shuffling

A Voting Scheme

Defined in terms of the algorithms needed for the tasks:

- Election Setup
- Ballot Casting
- Ballot Counting

Therefore, we need algorithms for:

- Shuffling
- Distributed Decryption

A Voting Scheme

Defined in terms of the algorithms needed for the tasks:

- Election Setup
- Ballot Casting
- Ballot Counting

Therefore, we need algorithms for:

- Shuffling
- Distributed Decryption
- Mechanisms to verify that encryption, decryption and shuffling are computed honestly/correct

Breakdown: E-Voting

Breakdown: E-Voting

What happens in each of the phases previously mentioned?

Breakdown: E-Voting

What happens in each of the phases previously mentioned?

- **Election Setup:** KeyGen is run by tursted party, the public params are given to each participant, decryption shares are distributed to decryption servers

Breakdown: E-Voting

What happens in each of the phases previously mentioned?

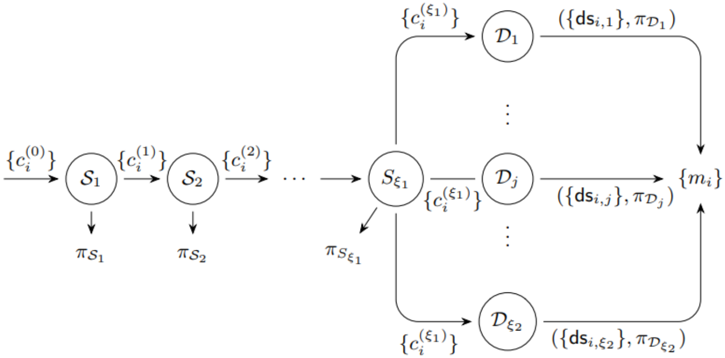
- **Election Setup:** KeyGen is run by tursted party, the public params are given to each participant, decryption shares are distributed to decryption servers
- **Ballot Casting:** Voters cast their ballot, their device encrypts it, along with a ballot proof

Breakdown: E-Voting

What happens in each of the phases previously mentioned?

- **Election Setup:** KeyGen is run by tursted party, the public params are given to each participant, decryption shares are distributed to decryption servers
- **Ballot Casting:** Voters cast their ballot, their device encrypts it, along with a ballot proof
- **Ballot Counting:** Encrypted ballots are shuffled, a shuffle proof is created, the decryption servers receives ballots, verifies the shuffle proofs, computes decryption shares and proof of correct decryption, the decryption shares are then combined after decryption proofs are verified

Components of our E-Voting Scheme



Shuffle

Shuffle

Definition

Shuffle$(\{c_i\}_{i\in[\tau]})$

Shuffle

Definition

Shuffle$(\{c_i\}_{i \in [\tau]})$

1. For each i, compute $c'_i \leftarrow Enc(pk, 0)$

Shuffle

Definition

Shuffle$(\{c_i\}_{i\in[\tau]})$

1. For each i, compute $c_i' \leftarrow Enc(pk, 0)$
2. For each i, compute $\hat{c}_i = c_i + c_i' \mod q$

Shuffle

Definition

Shuffle$(\{c_i\}_{i\in[\tau]})$

1. For each i, compute $c_i' \leftarrow Enc(pk, 0)$
2. For each i, compute $\hat{c}_i = c_i + c_i' \mod q$
3. Sample a random permutation $\pi \leftarrow Perm[\tau]$

Shuffle

Definition

Shuffle$(\{c_i\}_{i \in [\tau]})$

1. For each i, compute $c_i' \leftarrow Enc(pk, 0)$
2. For each i, compute $\hat{c}_i = c_i + c_i' \mod q$
3. Sample a random permutation $\pi \leftarrow Perm[\tau]$
4. Return new set of ballots $\left\{\hat{c}_{(\pi(i))}\right\}_{i \in [\tau]}$

ZK: Π_{SHUF}

ZK: Π_{SHUF}

- We use the shuffle by Aranha et. al. (CCS 23)

ZK: Π_{SHUF}

- We use the shuffle by Aranha et. al. (CCS 23)
- The proof of shuffle consists of $\tau - 1$ linearity proofs

ZK: Π_{SHUF}

- We use the shuffle by Aranha et. al. (CCS 23)
- The proof of shuffle consists of $\tau - 1$ linearity proofs
- The proof of shuffle is verified if all proofs of linearity is verified

ZK: Π_{SMALL}

ZK: Π_{SMALL}

- The Π_{SMALL} protocol is quite involved, but in short it does the following:

ZK: Π_{SMALL}

- The Π_{SMALL} protocol is quite involved, but in short it does the following:
- Create a proof that a batch of equations: $As_i = t_i$ for $i \in [l]$ is satisfied for a set of secret vectors s_i with ∞-norm bounded by ν

Distributed Decryption

Distributed Decryption

- Recall the NTRU scheme from earlier

Distributed Decryption

- Recall the NTRU scheme from earlier
- Imagine it outputs decryption shares dk_j as well

Distributed Decryption

- Recall the NTRU scheme from earlier
- Imagine it outputs decryption shares dk_j as well
- Then we define $DDec_j(\{c_i\}_{i \in [\tau]}, dk_j)$

Distributed Decryption

- Recall the NTRU scheme from earlier
- Imagine it outputs decryption shares dk_j as well
- Then we define $DDec_j(\{c_i\}_{i \in [\tau]}, dk_j)$
- For each i sample $E_{ij} \leftarrow S_{B_{Drown}}$, compute $ds_{ij} = dk_j \cdot c_i + p \cdot E_{ij}$

Distributed Decryption

- Recall the NTRU scheme from earlier
- Imagine it outputs decryption shares dk_j as well
- Then we define $DDec_j(\{c_i\}_{i\in[\tau]}, dk_j)$
- For each i sample $E_{ij} \leftarrow S_{B_{Drown}}$, compute $ds_{ij} = dk_j \cdot c_i + p \cdot E_{ij}$
- For each i compute $([\![E_{ij}]\!], r_{E_{ij}}) \leftarrow Com(E_{ij}, pk_C)$

Distributed Decryption

- Recall the NTRU scheme from earlier
- Imagine it outputs decryption shares dk_j as well
- Then we define $DDec_j(\{c_i\}_{i\in[\tau]}, dk_j)$
- For each i sample $E_{ij} \leftarrow S_{B_{Drown}}$, compute $ds_{ij} = dk_j \cdot c_i + p \cdot E_{ij}$
- For each i compute $([\![E_{ij}]\!], r_{E_{ij}}) \leftarrow Com(E_{ij}, pk_C)$
- Compute π_{LIN} for the linear relation $ds_{ij} = dk_j \cdot c_i + p \cdot E_{ij}$

Distributed Decryption

- Recall the NTRU scheme from earlier
- Imagine it outputs decryption shares dk_j as well
- Then we define $DDec_j(\{c_i\}_{i \in [\tau]}, dk_j)$
- For each i sample $E_{ij} \leftarrow S_{B_{Drown}}$, compute $ds_{ij} = dk_j \cdot c_i + p \cdot E_{ij}$
- For each i compute $([\![E_{ij}]\!], r_{E_{ij}}) \leftarrow Com(E_{ij}, pk_C)$
- Compute π_{LIN} for the linear relation $ds_{ij} = dk_j \cdot c_i + p \cdot E_{ij}$
- Compute π_{BND} to prove that for all i, $\|E_{ij}\|_\infty \leq B_{Drown}$

Distributed Decryption

- Recall the NTRU scheme from earlier
- Imagine it outputs decryption shares dk_j as well
- Then we define $DDec_j(\{c_i\}_{i\in[\tau]}, dk_j)$
- For each i sample $E_{ij} \leftarrow S_{B_{Drown}}$, compute $ds_{ij} = dk_j \cdot c_i + p \cdot E_{ij}$
- For each i compute $(\llbracket E_{ij} \rrbracket, r_{E_{ij}}) \leftarrow Com(E_{ij}, pk_C)$
- Compute π_{LIN} for the linear relation $ds_{ij} = dk_j \cdot c_i + p \cdot E_{ij}$
- Compute π_{BND} to prove that for all i, $\|E_{ij}\|_\infty \leq B_{Drown}$
- Return $ds_j = (\{ds_{ij}\}_{i\in\tau}, \pi_D)$

Combining Decryption Shares

Combining Decryption Shares

- Parse all decryption shares and verify the proofs π_{LIN} and π_{BND}

Combining Decryption Shares

- Parse all decryption shares and verify the proofs π_{LIN} and π_{BND}
- If no verification errors, compute

$$v_i = (\sum_{j \in \xi_2} ds_{ij} \mod q) \mod p$$

Combining Decryption Shares

- Parse all decryption shares and verify the proofs π_{LIN} and π_{BND}
- If no verification errors, compute

$$v_i = (\sum_{j \in \xi_2} ds_{ij} \mod q) \mod p$$

- Return the set of votes $\{v_i\}_{i \in \tau}$

ZK: Π_{LIN}

ZK: Π_{LIN}

Π_{LIN} produces a proof that a committed value v is a multiple of another commited value u with respect to a public scalar g.

ZK: Π_{BND}

ZK: Π_{BND}

For proving boundedness we use a slightly adapted version of the Π_{SMALL} protocol.

ZK: Π_{BND}

For proving boundedness we use a slightly adapted version of the Π_{SMALL} protocol.
Compared to previous works we use exact proofs to get better parameters, though this leads to larger proof sizes.

Contents

Parameters

Parameters

Decryption Correctness A ciphertext after the mix-net of ξ_1 shuffle servers is on the form

$$c = p(h \sum_{k \in [\xi_1]} s_k + \sum_{k \in [\xi_1]} e_k) + m$$

Parameters

Decryption Correctness A ciphertext after the mix-net of ξ_1 shuffle servers is on the form

$$c = p(h \sum_{k \in [\xi_1]} s_k + \sum_{k \in [\xi_1]} e_k) + m$$

After decryption shares are calculated by ξ_2 decryption servers we get

$$v' = (\sum_{j \in [\xi_2]} ds_j \mod q) \mod p$$

Parameters

Decryption Correctness A ciphertext after the mix-net of ξ_1 shuffle servers is on the form

$$c = p(h \sum_{k \in [\xi_1]} s_k + \sum_{k \in [\xi_1]} e_k) + m$$

After decryption shares are calculated by ξ_2 decryption servers we get

$$v' = (\sum_{j \in [\xi_2]} ds_j \mod q) \mod p$$

So correct decryption would have to satisfy

$$p \cdot d \cdot t \cdot \sigma_{NTRU} \cdot (2\xi_1 \cdot \nu + 1/2)(1 + 2^{sec}) < \lfloor q/2 \rfloor$$

Parameters

Parameter	Explanation	Value
λ	Computational security parameter	128
d	Ring dimension	2048
q	Ciphertext and commitment modulus	$\approx 2^{59}$
sec	Statistical security parameter	40
p	Plaintext modulus	2
t	$KeyGen$ rejection parameter	1.058
ν	Infinity norm of encryption randomness	1
B_{Com}	Infinity norm of commitment randomness	1
ξ_1, ξ_2	Number of shuffle and decryption servers	4
σ_{NTRU}	Standard deviation for encryption secret key	7.12

Our Implementation

Build upon
https://github.com/dfaranha/lattice-voting-ctrsa21 and
https://github.com/dfaranha/lattice-verifiable-mixnet
by Diego Aranha

Size Comparison

Scheme	c_i	$[\![R_q]\!]$	π_{Shuf}	π_{Lin}	π_{Small}	π_{Bnd}
[7] [KB]	80	80/120	150	35	20	2
Our [KB]	15	30	63	18	22	22

Table 3: Ciphertext, commitment, and proof sizes per voter. Note that the two sizes in [7] reflect commitments to noise-drowning terms and ciphertexts, respectively.

Timing Comparison

Scheme	Com	Open	Enc	Dec	DDec
[7] [ms]	0.45	2.76	0.74	0.64	1.56
Our [ms]	0.17	0.80	0.20	0.21	0.45

Table 4: Ciphertext and commitment timings. Numbers were obtained averaging over 10^4 executions measured using the cycle counter available on the platform.

Questions?