Threshold Lattice-Based Zero-Knowledge Proofs

Scott Griffy[1], Victor Youdom Kemmoe[1], Ngoc Khanh Nguyen[2], Tjerand Silde[3]

[1]Brown University [2]King's College London [3]Norwegian University of Science and Technology

Presented on 26-October-DD at WPEC 2026

NIST Workshop on Privacy-Enhancing Cryptography 2026

October 29, 2026

Overview

- Background
  - Zero-knowledge proofs
  - Lattices
  - Threshold cryptography
- Contribution Summary
- Threshold ZKP Definition
  - Syntax
  - Zero Knowledge
  - Knowledge Extraction
- Basic Framework
- Rejection sampling
- Exact proof
- Conclusion

Background

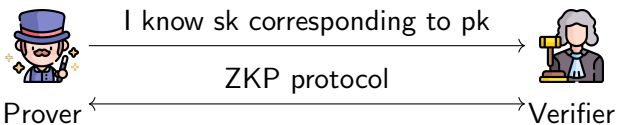
Zero-knowledge proofs

- Let a prover convince a verifier that a statement is true while revealing nothing about the witness

Background

Zero-knowledge proofs

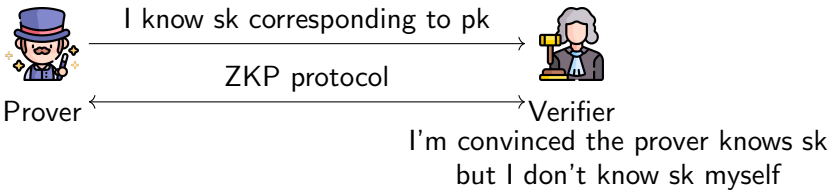
- Let a prover convince a verifier that a statement is true while revealing nothing about the witness



Background

Zero-knowledge proofs

- Let a prover convince a verifier that a statement is true while revealing nothing about the witness

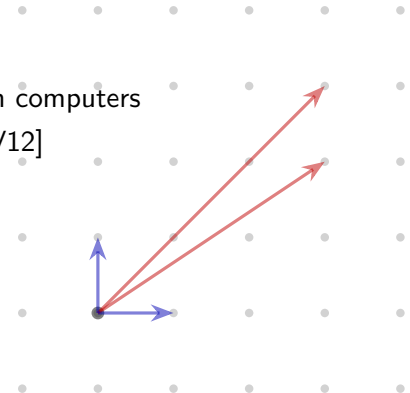


- Underlie many privacy-preserving applications such as
  - Anonymous credentials [CL03]
  - Verifiable encryption [LN17]
  - Private payments [BCG+14]

Background

Lattices

- Not known to be broken by any theoretical quantum computers
- Supports fully homomorphic encryption [BV11, BGV12]

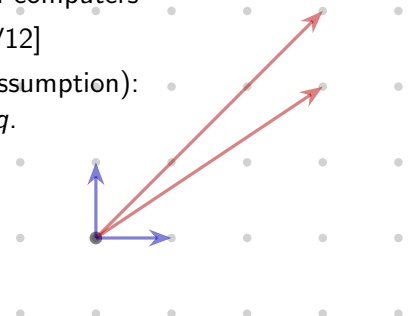


Background

Lattices

- Not known to be broken by any theoretical quantum computers
- Supports fully homomorphic encryption [BV11, BGV12]

Finding short solutions to linear equations is hard (SIS Assumption):
Given $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, find $\vec{\mathbf{r}}$ such that: $\mathbf{A}\vec{\mathbf{r}} = \vec{0}$ and $||\vec{\mathbf{r}}|| << q$.

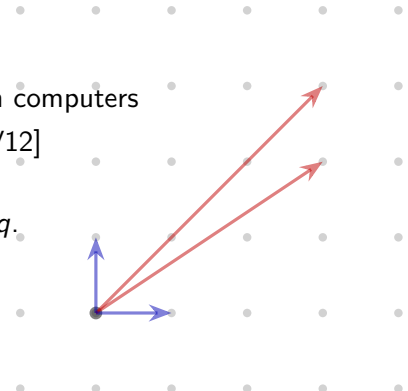


Background

Lattices

- Not known to be broken by any theoretical quantum computers
- Supports fully homomorphic encryption [BV11, BGV12]

Generalize to arbitrary rings (MSIS):
Given $\mathbf{A} \in R_q^{m \times n}$, find $\vec{\mathbf{r}}$ such that: $\mathbf{A}\vec{\mathbf{r}} = \vec{0}$ and $||\vec{\mathbf{r}}|| << q$.
Generally use $R_q = \mathbb{Z}_q/(x^d + 1)$.

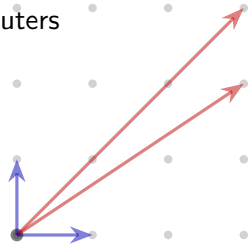


Background

Lattices

- Not known to be broken by any theoretical quantum computers
- Supports fully homomorphic encryption [BV11, BGV12]

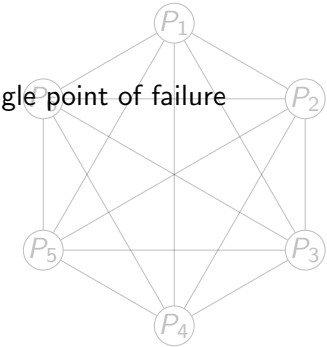
Generalize to arbitrary rings (MSIS):
Given $\mathbf{A} \in R_q^{m \times n}$, find $\vec{\mathbf{r}}$ such that: $\mathbf{A}\vec{\mathbf{r}} = \vec{0}$ and $||\vec{\mathbf{r}}|| << q$.
Generally use $R_q = \mathbb{Z}_q/(x^d + 1)$.
Related assumption (MLWE):
$(\mathbf{A}, \mathbf{A}\vec{\mathbf{r}} + \vec{\mathbf{e}}) \overset{c}{\approx} (\mathbf{A}, \vec{\mathbf{u}})$
Where $\vec{\mathbf{r}}, \vec{\mathbf{e}} \leftarrow\!\$\ \mathcal{D}$, $\vec{\mathbf{u}} \leftarrow\!\$\ R_q$



Background

Threshold cryptography

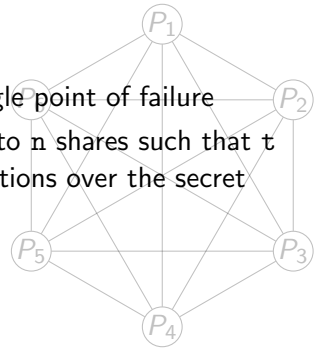
- Problem: trusting one person (or "party") introduces a single point of failure



Background

Threshold cryptography

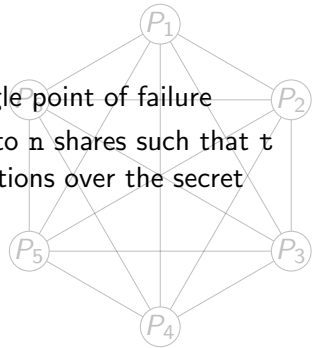
- Problem: trusting one person (or "party") introduces a single point of failure
- Solution: secret material (maybe encryption keys) is split into n shares such that t parties must come together to perform cryptographic operations over the secret



Background

Threshold cryptography

- Problem: trusting one person (or "party") introduces a single point of failure
- Solution: secret material (maybe encryption keys) is split into n shares such that t parties must come together to perform cryptographic operations over the secret
- Applications:
  - Blockchain and cryptocurrencies [DGNW20, MPSW19]
  - Multi-holder anonymous credentials (MHAC) [FLL25]
  - IoT and embedded devices



Contribution Summary

- Most efficient quantum-resistant threshold ZKP

Contribution Summary

- Most efficient quantum-resistant threshold ZKP
  - Based on efficient lattice-based ZKPs (Attema, Lyubashevsky, Seiler, CRYPTO 2020) and (Esgin, Nguyen, Seiler, Asiacrypt 2020).

Contribution Summary

- Most efficient quantum-resistant threshold ZKP
  - Based on exact ZKPs [ALS20, ENS20]
  - Uses threshold homomorphic encryption (Brakerski, Gentry, Vaikuntanathan, ITCS 2012). Similar to GKS framework (Gür, Katz, Silde, PQCRYPTO 2024).

Contribution Summary

- Most efficient quantum-resistant threshold ZKP
  - Based on exact ZKPs [ALS20, ENS20]
  - Uses threshold homomorphic encryption [GKS24].
  - Improves on (Boneh, Gennaro, Goldfeder, Jain, Kim, Rasmussen, Sahai, CRYPTO 2018) by not computing the hash homomorphically (instead adding rounds).

Contribution Summary

- Most efficient quantum-resistant threshold ZKP
  - Based on exact ZKPs [ALS20, ENS20]
  - Uses threshold homomorphic encryption [GKS24].
  - Improves on [BGG$^+$18]
  - Uses HintMLWE to avoid rejection sampling (Kim, Lee, Seo, Song, CRYPTO 2023).

Contribution Summary

- Most efficient quantum-resistant threshold ZKP
  - Based on exact ZKPs [ALS20, ENS20]
  - Uses threshold homomorphic encryption [GKS24].
  - Improves on [BGG$^+$18]
  - Uses HintMLWE to avoid rejection sampling [KLSS23].
- Also prove non-interactive variants of [ALS20, ENS20] without Fiat-Shamir heuristic using predicate special-soundness (Aardal, Aranha, Boudgoust, Kolby, Takahashi, CRYPTO 2024).

Contribution Summary

- Most efficient quantum-resistant threshold ZKP
  - Based on exact ZKPs [ALS20, ENS20]
  - Uses threshold homomorphic encryption [GKS24].
  - Improves on [BGG$^+$18]
  - Uses HintMLWE to avoid rejection sampling [KLSS23].
- Also prove non-interactive variants of [ALS20, ENS20] without Fiat-Shamir heuristic using [AAB$^+$24].
- Prove that MLWE is hard with non-uniform secrets (specifically those fixed by automorphisms) similar to (Goldwasser, Kalai, Peikert, Vaikuntanathan, ITCS 2010) or more recently (Boudgoust, Jeudy, Roux-Langlois, Wen, JoC 2023).

Contribution Summary

- Most efficient quantum-resistant threshold ZKP
  - Based on exact ZKPs [ALS20, ENS20]
  - Uses threshold homomorphic encryption [GKS24].
  - Improves on [BGG$^+$18]
  - Uses HintMLWE to avoid rejection sampling [KLSS23].
- Also prove non-interactive variants of [ALS20, ENS20] without Fiat-Shamir heuristic using [AAB$^+$24].
- Prove that MLWE is hard with secrets in lower rings similar to [BJRW23]

Threshold ZKP Definition

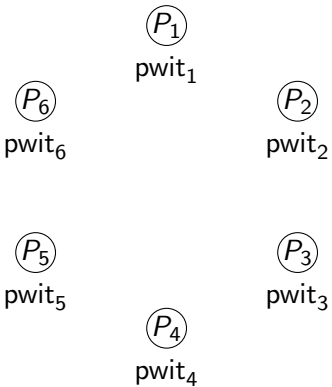
Syntax

Defined over a relation with witnesses and statements $(\mathbb{w}, \mathbb{x}) \in \mathcal{R}$
Function: Setup, WitnessShare, Commit (interactive), Prove (interactive), and Verify.

Threshold ZKP Definition

Syntax

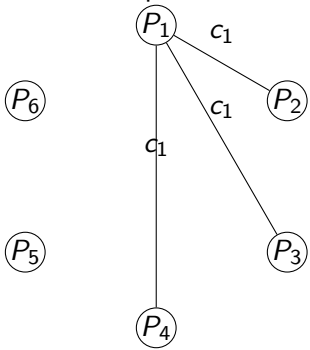
- WitnessShare: Split a witness $\mathbb{w}$ into n shares $\{\text{pwit}_i\}_{i\in[n]}$



Threshold ZKP Definition

Syntax

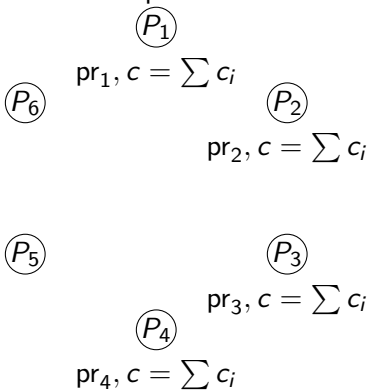
- WitnessShare: Split a witness $\mathbb{w}$ into n shares $\{\text{pwit}_i\}_{i\in[n]}$
- Commit: t parties interact to produce a commitment, c



Threshold ZKP Definition

Syntax

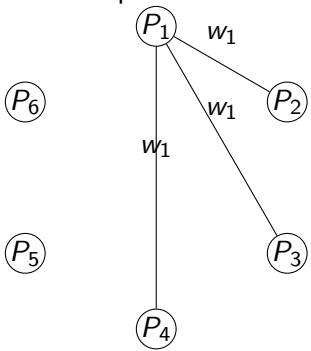
- WitnessShare: Split a witness $\mathbb{w}$ into n shares $\{\text{pwit}_i\}_{i\in[n]}$
- Commit: t parties interact to produce a commitment, c



Threshold ZKP Definition

Syntax

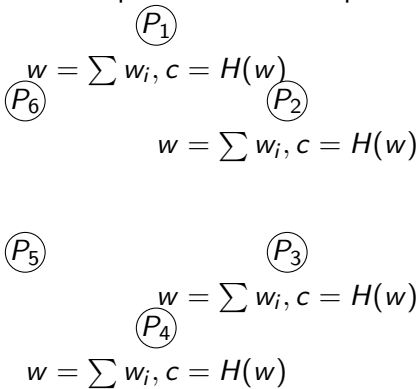
- WitnessShare: Split a witness $\mathbb{w}$ into n shares $\{\text{pwit}_i\}_{i\in[n]}$
- Commit: t parties interact to produce a commitment, c
- Prove: t parties interact to produce a proof π.



Threshold ZKP Definition

Syntax

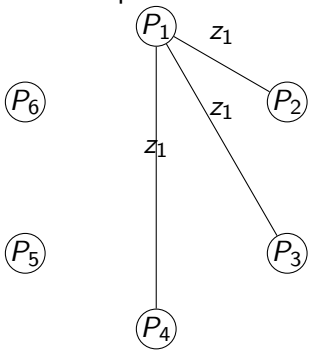
- WitnessShare: Split a witness $\mathbb{w}$ into n shares $\{\text{pwit}_i\}_{i\in[n]}$
- Commit: t parties interact to produce a commitment, c
- Prove: t parties interact to produce a proof π.



Threshold ZKP Definition

Syntax

- WitnessShare: Split a witness $\mathbb{w}$ into n shares $\{\text{pwit}_i\}_{i\in[n]}$
- Commit: t parties interact to produce a commitment, c
- Prove: t parties interact to produce a proof π.



Threshold ZKP Definition

Syntax

- WitnessShare: Split a witness $\mathbb{w}$ into n shares $\{\text{pwit}_i\}_{i\in[n]}$
- Commit: t parties interact to produce a commitment, c
- Prove: t parties interact to produce a proof π.
- Verify: Takes in the statement $\mathbb{x}$ and proof π and outputs whether $\mathbb{x} \in \mathcal{L}$.

Threshold ZKP Definition

Syntax

- WitnessShare: Split a witness $\mathbb{w}$ into n shares $\{\text{pwit}_i\}_{i\in[n]}$
- Commit: t parties interact to produce a commitment, c
- Prove: t parties interact to produce a proof π.
- Verify: Takes in the statement $\mathbb{x}$ and proof π and outputs whether $\mathbb{x} \in \mathcal{L}$.

Want this to be complete and have *Threshold Zero Knowledge* and *Threshold Simulation Extraction*

Threshold ZKP Definition

Zero Knowledge

t-out-of-n Threshold Zero Knowledge

$(\mathcal{C}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

Ensure $(\mathbb{w}, \mathbb{x}) \in \mathcal{R}$

Real-world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{P}} \leftarrow \mathsf{WitnessShare}(\mathbb{x}, \mathbb{w})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Commit}}(\mathsf{pwit}_i))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Prove}}(\mathsf{pr}_i, \mathsf{pwit}_i))_{i \in \mathcal{H}}}(c)$

Simulated world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{C}} \leftarrow \mathsf{SimWitnessShare}(\mathbb{x}, \mathcal{C})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimCom}}(i, \mathbb{x}, \{\mathsf{pwit}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimProve},i}(i, \mathbb{x}, \{\mathsf{pwit}_i, \mathsf{pr}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(c, \{\mathsf{pr}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}$ wins if they can guess which world they're in.

Threshold ZKP Definition

Zero Knowledge

t-out-of-n Threshold Zero Knowledge

$(\mathcal{C}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

Ensure $(\mathbb{w}, \mathbb{x}) \in \mathcal{R}$ ("weak" ZK)

Real-world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{P}} \leftarrow \mathsf{WitnessShare}(\mathbb{x}, \mathbb{w})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Commit}}(\mathsf{pwit}_i))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Prove}}(\mathsf{pr}_i, \mathsf{pwit}_i))_{i \in \mathcal{H}}}(c)$

Simulated world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{C}} \leftarrow \mathsf{SimWitnessShare}(\mathbb{x}, \mathcal{C})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimCom}}(i, \mathbb{x}, \{\mathsf{pwit}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimProve},i}(i, \mathbb{x}, \{\mathsf{pwit}_i, \mathsf{pr}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(c, \{\mathsf{pr}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}$ wins if they can guess which world they're in.

Threshold ZKP Definition

Zero Knowledge

t-out-of-n Threshold Zero Knowledge

$(\mathcal{C}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

Ensure $(\mathbb{w}, \mathbb{x}) \in \mathcal{R}$

Real-world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{P}} \leftarrow \mathsf{WitnessShare}(\mathbb{x}, \mathbb{w})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Commit}}(\mathsf{pwit}_i))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Prove}}(\mathsf{pr}_i, \mathsf{pwit}_i))_{i \in \mathcal{H}}}(c)$

Simulated world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{C}} \leftarrow \mathsf{SimWitnessShare}(\mathbb{x}, \mathcal{C})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimCom}}(i, \mathbb{x}, \{\mathsf{pwit}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimProve},i}(i, \mathbb{x}, \{\mathsf{pwit}_i, \mathsf{pr}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(c, \{\mathsf{pr}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}$ wins if they can guess which world they're in.

Threshold ZKP Definition

Zero Knowledge

t-out-of-n Threshold Zero Knowledge

$(\mathcal{C}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

Ensure $(\mathbb{w}, \mathbb{x}) \in \mathcal{R}$

Real-world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{P}} \leftarrow \mathsf{WitnessShare}(\mathbb{x}, \mathbb{w})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Commit}}(\mathsf{pwit}_i))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Prove}}(\mathsf{pr}_i, \mathsf{pwit}_i))_{i \in \mathcal{H}}}(c)$

Simulated world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{C}} \leftarrow \mathsf{SimWitnessShare}(\mathbb{x}, \mathcal{C})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimCom}}(i, \mathbb{x}, \{\mathsf{pwit}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimProve},i}(i, \mathbb{x}, \{\mathsf{pwit}_i, \mathsf{pr}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(c, \{\mathsf{pr}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}$ wins if they can guess which world they're in.

Threshold ZKP Definition

Zero Knowledge

t-out-of-n Threshold Zero Knowledge

$(\mathcal{C}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

Ensure $(\mathbb{w}, \mathbb{x}) \in \mathcal{R}$

Real-world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{P}} \leftarrow \mathsf{WitnessShare}(\mathbb{x}, \mathbb{w})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Commit}}(\mathsf{pwit}_i))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{Prove}}(\mathsf{pr}_i, \mathsf{pwit}_i))_{i \in \mathcal{H}}}(c)$

Simulated world:

$\{\mathsf{pwit}_i\}_{i \in \mathcal{C}} \leftarrow \mathsf{SimWitnessShare}(\mathbb{x}, \mathcal{C})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimCom}}(i, \mathbb{x}, \{\mathsf{pwit}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(\{\mathsf{pwit}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}^{\mathcal{O}(\mathsf{P}_{\mathsf{SimProve},i}(i, \mathbb{x}, \{\mathsf{pwit}_i, \mathsf{pr}_i\}_{i \in \mathcal{C}}))_{i \in \mathcal{H}}}(c, \{\mathsf{pr}_i\}_{i \in \mathcal{C}})$

$\mathcal{A}$ wins if they can guess which world they're in.

Threshold ZKP Definition

Knowledge Extraction

t-out-of-*n* Threshold Simulation Extraction

$(\mathcal{C}) \leftarrow \mathcal{A}(\mathsf{pp});\ |\mathcal{C}| < t$

$(\mathbb{x}^*, c^*, \pi^*) \leftarrow \mathcal{A}^{\{\mathcal{O}(\mathsf{P}_{\mathsf{SimWitnessShare}}(\cdot)),\mathcal{O}(\mathsf{P}_{\mathsf{SimCommit}}(\mathsf{pwit}_i)),\mathcal{O}(\mathsf{P}_{\mathsf{SimProve}}(\mathsf{pr}_i,\mathsf{pwit}_i))\}_{i\in\mathcal{P}\setminus\mathcal{C}}}$

$(\mathbb{w}^*, o^*) \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathbb{x}^*, c^*, \pi^*)$

$\mathcal{A}$ wins if $\mathsf{Verify}(\mathbb{x}^*, c^*, \pi^*) = 1$

$\wedge(\mathbb{x}^*, c^*, \pi^*) \notin Q_{\mathbb{x}}$ (the queries to the simulated prover)

$\wedge((\mathbb{x}^*, \mathbb{w}^*) \notin \mathcal{R} \vee \mathsf{Open}(c^*, \mathbb{w}^*, o^*) \neq 1)$

Threshold ZKP Definition

Knowledge Extraction

t-out-of-*n* Threshold Simulation Extraction

$(\mathcal{C}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

$(\mathbb{x}^*, c^*, \pi^*) \leftarrow \mathcal{A}^{\{\mathcal{O}(\mathsf{P}_{\mathsf{SimWitnessShare}}(\cdot)),\mathcal{O}(\mathsf{P}_{\mathsf{SimCommit}}(\mathsf{pwit}_i)),\mathcal{O}(\mathsf{P}_{\mathsf{SimProve}}(\mathsf{pr}_i,\mathsf{pwit}_i))\}_{i\in\mathcal{P}\setminus\mathcal{C}}}$

$(\mathbb{w}^*, o^*) \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathbb{x}^*, c^*, \pi^*)$

$\mathcal{A}$ wins if $\mathsf{Verify}(\mathbb{x}^*, c^*, \pi^*) = 1$

$\wedge(\mathbb{x}^*, c^*, \pi^*) \notin Q_{\mathbb{x}}$ (the queries to the simulated prover)

$\wedge((\mathbb{x}^*, \mathbb{w}^*) \notin \mathcal{R} \vee \mathsf{Open}(c^*, \mathbb{w}^*, o^*) \neq 1)$

Threshold ZKP Definition

Knowledge Extraction

t-out-of-*n* Threshold Simulation Extraction

$(\mathcal{C}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

$(\mathbb{x}^*, c^*, \pi^*) \leftarrow \mathcal{A}^{\{\mathcal{O}(\mathsf{P}_{\mathsf{SimWitnessShare}}(\cdot)),\mathcal{O}(\mathsf{P}_{\mathsf{SimCommit}}(\mathsf{pwit}_i)),\mathcal{O}(\mathsf{P}_{\mathsf{SimProve}}(\mathsf{pr}_i,\mathsf{pwit}_i))\}_{i\in\mathcal{P}\setminus\mathcal{C}}}$

$(\mathbb{w}^*, o^*) \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathbb{x}^*, c^*, \pi^*)$

$\mathcal{A}$ wins if $\mathsf{Verify}(\mathbb{x}^*, c^*, \pi^*) = 1$

$\wedge(\mathbb{x}^*, c^*, \pi^*) \notin Q_{\mathbb{x}}$ (the queries to the simulated prover)

$\wedge((\mathbb{x}^*, \mathbb{w}^*) \notin \mathcal{R} \vee \mathsf{Open}(c^*, \mathbb{w}^*, o^*) \neq 1)$

Threshold ZKP Definition

Knowledge Extraction

t-out-of-*n* Threshold Simulation Extraction

$(\mathcal{C}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

$(\mathbb{x}^*, c^*, \pi^*) \leftarrow \mathcal{A}^{\{\mathcal{O}(\mathsf{P}_{\mathsf{SimWitnessShare}}(\cdot)),\mathcal{O}(\mathsf{P}_{\mathsf{SimCommit}}(\mathsf{pwit}_i)),\mathcal{O}(\mathsf{P}_{\mathsf{SimProve}}(\mathsf{pr}_i,\mathsf{pwit}_i))\}_{i\in\mathcal{P}\setminus\mathcal{C}}}$

$(\mathbb{w}^*, o^*) \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathbb{x}^*, c^*, \pi^*)$

$\mathcal{A}$ wins if $\mathsf{Verify}(\mathbb{x}^*, c^*, \pi^*) = 1$

$\wedge(\mathbb{x}^*, c^*, \pi^*) \notin Q_{\mathbb{x}}$ (the queries to the simulated prover)

$\wedge((\mathbb{x}^*, \mathbb{w}^*) \notin \mathcal{R} \vee \mathsf{Open}(c^*, \mathbb{w}^*, o^*) \neq 1)$

Threshold ZKP Definition

Knowledge Extraction

t-out-of-*n* Threshold Simulation Extraction

$(\mathcal{C}) \leftarrow \mathcal{A}(\mathsf{pp})$; $|\mathcal{C}| < t$

$(\mathbb{x}^*, c^*, \pi^*) \leftarrow \mathcal{A}^{\{\mathcal{O}(\mathsf{P}_{\mathsf{SimWitnessShare}}(\cdot)),\mathcal{O}(\mathsf{P}_{\mathsf{SimCommit}}(\mathsf{pwit}_i)),\mathcal{O}(\mathsf{P}_{\mathsf{SimProve}}(\mathsf{pr}_i,\mathsf{pwit}_i))\}_{i\in\mathcal{P}\setminus\mathcal{C}}}$

$(\mathbb{w}^*, o^*) \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathbb{x}^*, c^*, \pi^*)$

$\mathcal{A}$ wins if $\mathsf{Verify}(\mathbb{x}^*, c^*, \pi^*) = 1$

$\wedge(\mathbb{x}^*, c^*, \pi^*) \notin Q_{\mathbb{x}}$ (the queries to the simulated prover)

$\wedge((\mathbb{x}^*, \mathbb{w}^*) \notin \mathcal{R} \vee \mathsf{Open}(c^*, \mathbb{w}^*, o^*) \neq 1)$

Threshold ZKP Definition

Knowledge Extraction

t-out-of-*n* Threshold Simulation Extraction

$(\mathcal{C}) \leftarrow \mathcal{A}(\mathsf{pp});\ |\mathcal{C}| < t$

$(\mathbb{x}^*, c^*, \pi^*) \leftarrow \mathcal{A}^{\{\mathcal{O}(\mathsf{P}_{\mathsf{SimWitnessShare}}(\cdot)),\mathcal{O}(\mathsf{P}_{\mathsf{SimCommit}}(\mathsf{pwit}_i)),\mathcal{O}(\mathsf{P}_{\mathsf{SimProve}}(\mathsf{pr}_i,\mathsf{pwit}_i))\}_{i\in\mathcal{P}\setminus\mathcal{C}}}$

$(\mathbb{w}^*, o^*) \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathbb{x}^*, c^*, \pi^*)$

$\mathcal{A}$ wins if $\mathsf{Verify}(\mathbb{x}^*, c^*, \pi^*) = 1$

$\wedge (\mathbb{x}^*, c^*, \pi^*) \notin Q_{\mathbb{x}}$ (the queries to the simulated prover)

$\wedge((\mathbb{x}^*, \mathbb{w}^*) \notin \mathcal{R} \vee \mathsf{Open}(c^*, \mathbb{w}^*, o^*) \neq 1)$

Basic Framework

Public key: SIS Sample $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$

($\mathbf{B} \in R_q^{m \times n}$, $R_q = \mathbb{Z}_q/(x^d + 1)$)

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$

Prover commits to $\vec{\mathbf{y}}$ with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$ (the statement is the public key in this case)

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$, where $\vec{\mathbf{y}}$ is a masking vector and $\mathbf{c}$ is the challenge.

Basic Framework

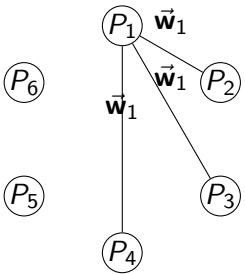
Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$ s.t. $\mathbf{B}\vec{\mathbf{r}}^* = \vec{\mathbf{t}}$

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$
Threshold ZK:
Parties hold $\vec{\mathbf{r}}_i$ s.t. $\sum \vec{\mathbf{r}}_i = \vec{\mathbf{r}}$ and $\vec{\mathbf{y}}_i$ s.t. $\sum \vec{\mathbf{y}}_i = \vec{\mathbf{y}}$

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$
Threshold ZK:
Parties hold $\vec{\mathbf{r}}_i$ s.t. $\sum \vec{\mathbf{r}}_i = \vec{\mathbf{r}}$ and $\vec{\mathbf{y}}_i$ s.t. $\sum \vec{\mathbf{y}}_i = \vec{\mathbf{y}}$
Broadcast: $\vec{\mathbf{w}}_i = \mathbf{B}\vec{\mathbf{r}}_i$



Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$
Threshold ZK:
Parties hold $\vec{\mathbf{r}}_i$ s.t. $\sum \vec{\mathbf{r}}_i = \vec{\mathbf{r}}$ and $\vec{\mathbf{y}}_i$ s.t. $\sum \vec{\mathbf{y}}_i = \vec{\mathbf{y}}$
Broadcast: $\vec{\mathbf{w}}_i = \mathbf{B}\vec{\mathbf{r}}_i$
Compute: $\vec{\mathbf{w}} = \sum \vec{\mathbf{w}}_i$, $c = H(\vec{\mathbf{w}}, \mathbb{x})$

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$
Threshold ZK:
Parties hold $\vec{\mathbf{r}}_i$ s.t. $\sum \vec{\mathbf{r}}_i = \vec{\mathbf{r}}$ and $\vec{\mathbf{y}}_i$ s.t. $\sum \vec{\mathbf{y}}_i = \vec{\mathbf{y}}$
Broadcast: $\vec{\mathbf{w}}_i = \mathbf{B}\vec{\mathbf{r}}_i$
Compute: $\vec{\mathbf{w}} = \sum \vec{\mathbf{w}}_i$, $c = H(\vec{\mathbf{w}}, \mathbb{x})$
Each compute: $\vec{\mathbf{z}}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$, broadcast, and sum $\vec{\mathbf{z}} = \sum(\vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i)$

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$
Threshold ZK:
Parties hold $\vec{\mathbf{r}}_i$ s.t. $\sum \vec{\mathbf{r}}_i = \vec{\mathbf{r}}$ and $\vec{\mathbf{y}}_i$ s.t. $\sum \vec{\mathbf{y}}_i = \vec{\mathbf{y}}$
Broadcast: $\vec{\mathbf{w}}_i = \mathbf{B}\vec{\mathbf{r}}_i$
Compute: $\vec{\mathbf{w}} = \sum \vec{\mathbf{w}}_i$, $c = H(\vec{\mathbf{w}}, \mathbb{x})$
Each compute: $\vec{\mathbf{z}}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$, broadcast, and sum $\vec{\mathbf{z}} = \sum(\vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i)$
Works for t-out-of-n proofs by reducing to single prover:

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$
Threshold ZK:
Parties hold $\vec{\mathbf{r}}_i$ s.t. $\sum \vec{\mathbf{r}}_i = \vec{\mathbf{r}}$ and $\vec{\mathbf{y}}_i$ s.t. $\sum \vec{\mathbf{y}}_i = \vec{\mathbf{y}}$
Broadcast: $\vec{\mathbf{w}}_i = \mathbf{B}\vec{\mathbf{r}}_i$
Compute: $\vec{\mathbf{w}} = \sum \vec{\mathbf{w}}_i$, $c = H(\vec{\mathbf{w}}, \mathbb{x})$
Each compute: $\vec{\mathbf{z}}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$, broadcast, and sum $\vec{\mathbf{z}} = \sum(\vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i)$
Works for t-out-of-n proofs by reducing to single prover:
Single prover: $\vec{\mathbf{z}}' = \vec{\mathbf{y}}' + \mathbf{c}\vec{\mathbf{r}}'$.

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$
Threshold ZK:
Parties hold $\vec{\mathbf{r}}_i$ s.t. $\sum \vec{\mathbf{r}}_i = \vec{\mathbf{r}}$ and $\vec{\mathbf{y}}_i$ s.t. $\sum \vec{\mathbf{y}}_i = \vec{\mathbf{y}}$
Broadcast: $\vec{\mathbf{w}}_i = \mathbf{B}\vec{\mathbf{r}}_i$
Compute: $\vec{\mathbf{w}} = \sum \vec{\mathbf{w}}_i$, $c = H(\vec{\mathbf{w}}, \mathbb{x})$
Each compute: $\vec{\mathbf{z}}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$, broadcast, and sum $\vec{\mathbf{z}} = \sum(\vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i)$
Works for t-out-of-n proofs by reducing to single prover:
Single prover: $\vec{\mathbf{z}}' = \vec{\mathbf{y}}' + \mathbf{c}\vec{\mathbf{r}}'$.
Combined with $t-1$ adversary shares: $\vec{\mathbf{z}} = \vec{\mathbf{z}}' + \sum_{\mathcal{A}} \vec{\mathbf{z}}_i = \sum(\vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i)$

Basic Framework

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Prover commits with $\vec{\mathbf{w}} = \mathbf{B}\vec{\mathbf{y}}$.
Compute the hash: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Last message is $\vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$,
Extract potential opening: $\vec{\mathbf{r}}^* = (\vec{\mathbf{z}} - \vec{\mathbf{z}}')/(\mathbf{c} - \mathbf{c}')$
Threshold ZK:
Parties hold $\vec{\mathbf{r}}_i$ s.t. $\sum \vec{\mathbf{r}}_i = \vec{\mathbf{r}}$ and $\vec{\mathbf{y}}_i$ s.t. $\sum \vec{\mathbf{y}}_i = \vec{\mathbf{y}}$
Broadcast: $\vec{\mathbf{w}}_i = \mathbf{B}\vec{\mathbf{r}}_i$
Compute: $\vec{\mathbf{w}} = \sum \vec{\mathbf{w}}_i$, $c = H(\vec{\mathbf{w}}, \mathbb{x})$
Each compute: $\vec{\mathbf{z}}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$, broadcast, and sum $\vec{\mathbf{z}} = \sum(\vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i)$
Works for t-out-of-n proofs by reducing to single prover:
Single prover: $\vec{\mathbf{z}}' = \vec{\mathbf{y}}' + \mathbf{c}\vec{\mathbf{r}}'$.
Combined with $t-1$ adversary shares: $\vec{\mathbf{z}} = \vec{\mathbf{z}}' + \sum_{\mathcal{A}} \vec{\mathbf{z}}_i = \sum(\vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i)$
Each $\vec{\mathbf{y}}_i$ needs to hide $\mathbf{c}\vec{\mathbf{r}}'$ and $\vec{\mathbf{z}}$ needs to be small enough to verify ($\sqrt{t}$ extra slack due to sums of discrete Gaussians)

Rejection sampling

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Commitment: $\vec{\mathbf{w}} = \sum \mathbf{B}\vec{\mathbf{y}}_i$
Challenge: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Partial final messages: $\mathbf{z}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$ and sum $\vec{\mathbf{z}} = \sum \vec{\mathbf{y}}_i + \mathbf{c} \sum \vec{\mathbf{r}}_i$

Rejection sampling

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Commitment: $\vec{\mathbf{w}} = \sum \mathbf{B}\vec{\mathbf{y}}_i$
Challenge: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Partial final messages: $\mathbf{z}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$ and sum $\vec{\mathbf{z}} = \sum \vec{\mathbf{y}}_i + \mathbf{c} \sum \vec{\mathbf{r}}_i$
Prover i may want to reject if $\vec{\mathbf{z}}_i$ is too large[Lyu09].

Rejection sampling

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Commitment: $\vec{\mathbf{w}} = \sum \mathbf{B}\vec{\mathbf{y}}_i$
Challenge: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Partial final messages: $\mathbf{z}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$ and sum $\vec{\mathbf{z}} = \sum \vec{\mathbf{y}}_i + \mathbf{c} \sum \vec{\mathbf{r}}_i$
Prover i may want to reject if $\vec{\mathbf{z}}_i$ is too large[Lyu09].
In the threshold setting, rejection sampling requires extra rounds.

Rejection sampling

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Commitment: $\vec{\mathbf{w}} = \sum \mathbf{B}\vec{\mathbf{y}}_i$
Challenge: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Partial final messages: $\mathbf{z}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$ and sum $\vec{\mathbf{z}} = \sum \vec{\mathbf{y}}_i + \mathbf{c}\sum \vec{\mathbf{r}}_i$
Prover i may want to reject if $\vec{\mathbf{z}}_i$ is too large[Lyu09].
In the threshold setting, rejection sampling requires extra rounds.
Intead, reduce to Hint-MLWE:

$$(\mathbf{A}, [\mathbf{I}|\mathbf{A}]\vec{\mathbf{r}}, \mathbf{c}, \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}) \overset{c}{\approx} (\mathbf{A}, \vec{\mathbf{u}}, \mathbf{c}, \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}})$$

$\vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}}$ is the "hint"

Rejection sampling

Public key: $\vec{\mathbf{t}} := \mathbf{B}\vec{\mathbf{r}}$
Commitment: $\vec{\mathbf{w}} = \sum \mathbf{B}\vec{\mathbf{y}}_i$
Challenge: $\mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x})$
Partial final messages: $\mathbf{z}_i = \vec{\mathbf{y}}_i + \mathbf{c}\vec{\mathbf{r}}_i$ and sum $\vec{\mathbf{z}} = \sum \vec{\mathbf{y}}_i + \mathbf{c}\sum \vec{\mathbf{r}}_i$
Prover i may want to reject if $\vec{\mathbf{z}}_i$ is too large[Lyu09].
In the threshold setting, rejection sampling requires extra rounds.
Intead, reduce to Hint-MLWE:

$$(\mathbf{A}, [\mathbf{I}|\mathbf{A}]\vec{\mathbf{r}}, (\mathbf{c}_i, \vec{\mathbf{y}}_i + \mathbf{c}_i\vec{\mathbf{r}})_{i\in[\ell]}) \overset{c}{\approx} (\mathbf{A}, \vec{\mathbf{u}}, (\mathbf{c}_i, \vec{\mathbf{y}}_i + \mathbf{c}_i\vec{\mathbf{r}})_{i\in[\ell]})$$

Efficient for one-time commitments.

Exact proof

If $\vec{m} \odot (\vec{m} - \vec{1}) = \vec{0}$, it must be that $\vec{m}$ is a binary vector (where $\odot$ is a pair-wise multiplication).
If $\vec{m}$ includes 2 then $2 * (2 - 1) = 2 \neq 0$

Exact proof

If $\vec{m} \odot (\vec{m} - \vec{1}) = \vec{0}$, it must be that $\vec{m}$ is a binary vector (where $\odot$ is a pair-wise multiplication).
If we are working over a ring, $R_q = \mathbb{Z}_q[X]/(x^d + 1) \cong \mathbb{Z}_q^d$, we can use NTT to represent $\odot$ as ring multiplication.

Exact proof

If $\vec{m} \odot (\vec{m} - \vec{1}) = \vec{0}$, it must be that $\vec{m}$ is a binary vector (where $\odot$ is a pair-wise multiplication).
If we are working over a ring, $R_q = \mathbb{Z}_q[X]/(x^d + 1) \cong \mathbb{Z}_q^d$, we can use NTT to represent $\odot$ as ring multiplication.
Take $\vec{m} = (m_1, \ldots, m_d) \in \mathbb{Z}_q^d$, compute the NTT inverse, $\mathbf{m} \in R_q$.

Exact proof

If $\vec{m} \odot (\vec{m} - \vec{1}) = \vec{0}$, it must be that $\vec{m}$ is a binary vector (where $\odot$ is a pair-wise multiplication).
If we are working over a ring, $R_q = \mathbb{Z}_q[X]/(x^d + 1) \cong \mathbb{Z}_q^d$, we can use NTT to represent $\odot$ as ring multiplication.
Take $\vec{m} = (m_1, \ldots, m_d) \in \mathbb{Z}_q^d$, compute the NTT inverse, $\mathbf{m} \in R_q$.
Prove that $\mathbf{m}(\mathbf{m} - 1) = 0 \in R_q$

Exact proof

If $\vec{m} \odot (\vec{m} - \vec{1}) = \vec{0}$, it must be that $\vec{m}$ is a binary vector (where $\odot$ is a pair-wise multiplication).
If we are working over a ring, $R_q = \mathbb{Z}_q[X]/(x^d + 1) \cong \mathbb{Z}_q^d$, we can use NTT to represent $\odot$ as ring multiplication.
Take $\vec{m} = (m_1, \ldots, m_d) \in \mathbb{Z}_q^d$, compute the NTT inverse, $\mathbf{m} \in R_q$.
Prove that $\mathbf{m}(\mathbf{m} - 1) = 0 \in R_q$

$$\vec{\mathbf{t}} := \begin{bmatrix} \vec{\mathbf{t}}_0 \\ \vec{\mathbf{t}}_1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \end{bmatrix} \vec{\mathbf{r}} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m} \\ \mathbf{m} - 1 \\ 0 \end{bmatrix}$$

Exact proof

If $\vec{m} \odot (\vec{m} - \vec{1}) = \vec{0}$, it must be that $\vec{m}$ is a binary vector (where $\odot$ is a pair-wise multiplication).
If we are working over a ring, $R_q = \mathbb{Z}_q[X]/(x^d + 1) \cong \mathbb{Z}_q^d$, we can use NTT to represent $\odot$ as ring multiplication.
Take $\vec{m} = (m_1, \ldots, m_d) \in \mathbb{Z}_q^d$, compute the NTT inverse, $\mathbf{m} \in R_q$.
Prove that $\mathbf{m}(\mathbf{m} - 1) = 0 \in R_q$

$$\vec{\mathbf{t}} := \begin{bmatrix} \vec{\mathbf{t}}_0 \\ \vec{\mathbf{t}}_1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \end{bmatrix} \vec{\mathbf{r}} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m} \\ \mathbf{m} - 1 \\ 0 \end{bmatrix}$$

Opening: $\begin{bmatrix} \mathbf{m} \\ \mathbf{m} - 1 \\ 0 \end{bmatrix} = \vec{\mathbf{t}}_1 - \mathbf{B}_1 \vec{\mathbf{r}}$

Exact proof

$$\vec{\mathbf{t}} := \begin{bmatrix} \vec{\mathbf{t}}_0 \\ \vec{\mathbf{t}}_1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \end{bmatrix} \vec{\mathbf{r}} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix}$$

Exact proof

$$\vec{\mathbf{t}} := \begin{bmatrix} \vec{\mathbf{t}}_0 \\ \vec{\mathbf{t}}_1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \end{bmatrix} \vec{\mathbf{r}} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix}$$

$$(\vec{\mathbf{w}} = \mathbf{B}_0 \vec{\mathbf{y}},\ \mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x}),\ \vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}})$$

Exact proof

$$\vec{\mathbf{t}} := \begin{bmatrix} \vec{\mathbf{t}}_0 \\ \vec{\mathbf{t}}_1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \end{bmatrix} \vec{\mathbf{r}} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix}$$

$(\vec{\mathbf{w}} = \mathbf{B}_0\vec{\mathbf{y}},\ \mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x}),\ \vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}})$

Masked openings: $\mathbf{f}_i = \vec{\mathbf{b}}_i\vec{\mathbf{y}} - \mathbf{c}\mathbf{m}_i$ for $i \in [3]$, the verifier can compute:

$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3$

Exact proof

$$\vec{\mathbf{t}} := \begin{bmatrix} \vec{\mathbf{t}}_0 \\ \vec{\mathbf{t}}_1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \end{bmatrix} \vec{\mathbf{r}} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix}$$

$(\vec{\mathbf{w}} = \mathbf{B}_0\vec{\mathbf{y}},\ \mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x}),\ \vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}})$

Masked openings: $\mathbf{f}_i = \vec{\mathbf{b}}_i\vec{\mathbf{y}} - \mathbf{cm}_i$ for $i \in [3]$, the verifier can compute:

$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 = (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{cm}_1)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle - \mathbf{cm}_2) + \mathbf{c}(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{cm}_3)$

Exact proof

$$\vec{\mathbf{t}} := \begin{bmatrix}\vec{\mathbf{t}}_0\\ \vec{\mathbf{t}}_1\end{bmatrix} \leftarrow \begin{bmatrix}\mathbf{B}_0\\ \mathbf{B}_1\end{bmatrix}\vec{\mathbf{r}} + \begin{bmatrix}\mathbf{0}^n\\ \mathbf{m}_1\\ \mathbf{m}_2\\ \mathbf{m}_3\end{bmatrix}$$

$(\vec{\mathbf{w}} = \mathbf{B}_0\vec{\mathbf{y}},\ \mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x}),\ \vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}})$

Masked openings: $\mathbf{f}_i = \vec{\mathbf{b}}_i\vec{\mathbf{y}} - \mathbf{c}\mathbf{m}_i$ for $i \in [3]$, the verifier can compute:

$$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 = (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle) + \mathbf{c}\left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right) + \mathbf{c}^2\left(\mathbf{m}_1\mathbf{m}_2 - \mathbf{m}_3\right).$$

Exact proof

$$\vec{\mathbf{t}} := \begin{bmatrix} \vec{\mathbf{t}}_0 \\ \vec{\mathbf{t}}_1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \end{bmatrix} \vec{\mathbf{r}} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix}$$

$(\vec{\mathbf{w}} = \mathbf{B}_0\vec{\mathbf{y}},\ \mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x}),\ \vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}})$

Masked openings: $\mathbf{f}_i = \vec{\mathbf{b}}_i\vec{\mathbf{y}} - \mathbf{c}\mathbf{m}_i$ for $i \in [3]$, the verifier can compute:

$$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 = (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle) + \mathbf{c}\left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right) + \mathbf{c}^2\left(\mathbf{m}_1\mathbf{m}_2 - \mathbf{m}_3\right).$$

We can see that if $\mathbf{m}_1\mathbf{m}_2 = \mathbf{m}_3$, the quadratic term vanishes and we are left with:

$$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 = (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle) + \mathbf{c}\left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right)$$

Exact proof

$$\vec{\mathbf{t}} := \begin{bmatrix}\vec{\mathbf{t}}_0\\ \vec{\mathbf{t}}_1\end{bmatrix} \leftarrow \begin{bmatrix}\mathbf{B}_0\\ \mathbf{B}_1\end{bmatrix}\vec{\mathbf{r}} + \begin{bmatrix}\mathbf{0}^n\\ \mathbf{m}_1\\ \mathbf{m}_2\\ \mathbf{m}_3\end{bmatrix}$$

$(\vec{\mathbf{w}} = \mathbf{B}_0\vec{\mathbf{y}},\ \mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x}),\ \vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}})$

Masked openings: $\mathbf{f}_i = \vec{\mathbf{b}}_i\vec{\mathbf{y}} - \mathbf{c}\mathbf{m}_i$ for $i \in [3]$, the verifier can compute:

$$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 = (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle) + \mathbf{c}\left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right) + \mathbf{c}^2\left(\mathbf{m}_1\mathbf{m}_2 - \mathbf{m}_3\right).$$

We can see that if $\mathbf{m}_1\mathbf{m}_2 = \mathbf{m}_3$, the quadratic term vanishes and we are left with:

$$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 = (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle) + \mathbf{c}\left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right)$$

Need to commit to these "garbage terms" and prove it can be computed without $\mathbf{c}^2$ before seeing $\mathbf{c}$.

$$\mathbf{t}_4 = \langle\vec{\mathbf{b}}_4, \vec{\mathbf{r}}\rangle + \left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right)$$

$$\mathbf{t}_5 = \langle\vec{\mathbf{b}}_5, \vec{\mathbf{r}}\rangle + (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle)$$

$\mathbf{c} = H(\vec{\mathbf{w}}, \vec{\mathbf{t}}_0, \vec{\mathbf{t}}_1, \mathbf{t}_4, \mathbf{t}_5, \mathbb{x})$

Reduces to proving: $\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 - \mathbf{c}\mathbf{t}_4 - \mathbf{t}_5$ is a commitment to 0.

Exact proof

$$\vec{\mathbf{t}} := \begin{bmatrix}\vec{\mathbf{t}}_0\\ \vec{\mathbf{t}}_1\end{bmatrix} \leftarrow \begin{bmatrix}\mathbf{B}_0\\ \mathbf{B}_1\end{bmatrix}\vec{\mathbf{r}} + \begin{bmatrix}\mathbf{0}^n\\ \mathbf{m}_1\\ \mathbf{m}_2\\ \mathbf{m}_3\end{bmatrix}$$

$(\vec{\mathbf{w}} = \mathbf{B}_0\vec{\mathbf{y}},\ \mathbf{c} = H(\vec{\mathbf{w}}, \mathbb{x}),\ \vec{\mathbf{z}} = \vec{\mathbf{y}} + \mathbf{c}\vec{\mathbf{r}})$

Masked openings: $\mathbf{f}_i = \vec{\mathbf{b}}_i\vec{\mathbf{y}} - \mathbf{c}\mathbf{m}_i$ for $i \in [3]$, the verifier can compute:

$$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 = (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle) + \mathbf{c}\left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right) + \mathbf{c}^2\left(\mathbf{m}_1\mathbf{m}_2 - \mathbf{m}_3\right).$$

We can see that if $\mathbf{m}_1\mathbf{m}_2 = \mathbf{m}_3$, the quadratic term vanishes and we are left with:

$$\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 = (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle) + \mathbf{c}\left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right)$$

Need to commit to these "garbage terms" and prove it can be computed without $\mathbf{c}^2$ before seeing $\mathbf{c}$.

$$\mathbf{t}_4 = \langle\vec{\mathbf{b}}_4, \vec{\mathbf{r}}\rangle + \left(\langle\vec{\mathbf{b}}_3, \vec{\mathbf{y}}\rangle - \mathbf{m}_2\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle - \mathbf{m}_1\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle\right)$$

$$\mathbf{v} = \langle\vec{\mathbf{b}}_4, \vec{\mathbf{y}}\rangle + (\langle\vec{\mathbf{b}}_1, \vec{\mathbf{y}}\rangle)(\langle\vec{\mathbf{b}}_2, \vec{\mathbf{y}}\rangle)$$

Verifier checks $\mathbf{f}_1\mathbf{f}_2 + \mathbf{c}\mathbf{f}_3 - \mathbf{c}\mathbf{f}_4 = \mathbf{v}$

Exact proof

This is one reason we need homomorphic encryption (product of secrets $\mathbf{m}$ and $\vec{\mathbf{y}}$):

Exact proof

This is one reason we need homomorphic encryption (product of secrets $\mathbf{m}$ and $\vec{\mathbf{y}}$):
In setup, generate a BGV encryption scheme key.

Exact proof

This is one reason we need homomorphic encryption (product of secrets $\mathbf{m}$ and $\vec{\mathbf{y}}$):
In setup, generate a BGV encryption scheme key.
Each party receives $\mathrm{ctx}_{\vec{\mathbf{r}}}$, $\mathrm{ctx}_{\vec{\mathbf{y}}}$, $\mathrm{ctx}_{\mathbf{m}_i}$, sk_i

$$\mathrm{ctx}_{\mathbf{t}_4} = \langle \vec{\mathbf{b}}_4, \mathrm{ctx}_{\vec{\mathbf{r}}} \rangle + \left(\langle \vec{\mathbf{b}}_3, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle - \mathrm{ctx}_{\mathbf{m}_2} \langle \vec{\mathbf{b}}_1, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle - \mathrm{ctx}_{\mathbf{m}_1} \langle \vec{\mathbf{b}}_2, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle \right)$$

$$\mathrm{ctx}_{\mathbf{v}} = \langle \vec{\mathbf{b}}_4, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle + (\langle \vec{\mathbf{b}}_1, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle)(\langle \vec{\mathbf{b}}_2, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle)$$

Exact proof

This is one reason we need homomorphic encryption (product of secrets $\mathbf{m}$ and $\vec{\mathbf{y}}$):
In setup, generate a BGV encryption scheme key.
Each party receives $\mathrm{ctx}_{\vec{\mathbf{r}}}$, $\mathrm{ctx}_{\vec{\mathbf{y}}}$, $\mathrm{ctx}_{\mathbf{m}_i}$, sk_i

$$\mathrm{ctx}_{\mathbf{t}_4} = \langle \vec{\mathbf{b}}_4, \mathrm{ctx}_{\vec{\mathbf{r}}} \rangle + \left(\langle \vec{\mathbf{b}}_3, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle - \mathrm{ctx}_{\mathbf{m}_2} \langle \vec{\mathbf{b}}_1, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle - \mathrm{ctx}_{\mathbf{m}_1} \langle \vec{\mathbf{b}}_2, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle \right)$$

$$\mathrm{ctx}_{\mathbf{v}} = \langle \vec{\mathbf{b}}_4, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle + (\langle \vec{\mathbf{b}}_1, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle)(\langle \vec{\mathbf{b}}_2, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle)$$

Use GKS threshold homomorphic encryption to compute decryption shares
$\mathrm{ds}_{\mathbf{v},i} = \mathrm{Dec}(\mathrm{sk}_i, \mathrm{ctx}_{\mathbf{v}})$ and broadcast them: $\mathbf{v} = \sum_{\mathcal{P}} \mathrm{ds}_{\mathrm{ctx}_{\mathbf{v}},i}$

Exact proof

This is one reason we need homomorphic encryption (product of secrets $\mathbf{m}$ and $\vec{\mathbf{y}}$):
In setup, generate a BGV encryption scheme key.
Each party receives $\mathrm{ctx}_{\vec{\mathbf{r}}}$, $\mathrm{ctx}_{\vec{\mathbf{y}}}$, $\mathrm{ctx}_{\mathbf{m}_i}$, sk_i

$$\mathrm{ctx}_{\mathbf{t}_4} = \langle \vec{\mathbf{b}}_4, \mathrm{ctx}_{\vec{\mathbf{r}}} \rangle + \left(\langle \vec{\mathbf{b}}_3, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle - \mathrm{ctx}_{\mathbf{m}_2} \langle \vec{\mathbf{b}}_1, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle - \mathrm{ctx}_{\mathbf{m}_1} \langle \vec{\mathbf{b}}_2, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle \right)$$

$$\mathrm{ctx}_{\mathbf{v}} = \langle \vec{\mathbf{b}}_4, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle + (\langle \vec{\mathbf{b}}_1, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle)(\langle \vec{\mathbf{b}}_2, \mathrm{ctx}_{\vec{\mathbf{y}}} \rangle)$$

Use GKS threshold homomorphic encryption to compute decryption shares
$\mathrm{ds}_{\mathbf{v},i} = \mathrm{Dec}(\mathrm{sk}_i, \mathrm{ctx}_{\mathbf{v}})$ and broadcast them: $\mathbf{v} = \sum_{\mathcal{P}} \mathrm{ds}_{\mathrm{ctx}_{\mathbf{v}},i}$
Another reason is when $t+1$ or more parties compute a proof, since we can't simply reduce to single prover.
$\mathrm{ctx}_{\mathbf{z}} = \mathrm{ctx}_{\vec{\mathbf{y}}} + \mathbf{c}\mathrm{ctx}_{\vec{\mathbf{r}}}$.

Exact proof

This is one reason we need homomorphic encryption (product of secrets $\mathbf{m}$ and $\vec{\mathbf{y}}$):
In setup, generate a BGV encryption scheme key.
Each party receives $\mathsf{ctx}_{\vec{\mathbf{r}}}$, $\mathsf{ctx}_{\vec{\mathbf{y}}}$, $\mathsf{ctx}_{\mathbf{m}_i}$, sk_i

$$\mathsf{ctx}_{\mathbf{t}_4} = \langle \vec{\mathbf{b}}_4, \mathsf{ctx}_{\vec{\mathbf{r}}} \rangle + \left(\langle \vec{\mathbf{b}}_3, \mathsf{ctx}_{\vec{\mathbf{y}}} \rangle - \mathsf{ctx}_{\mathbf{m}_2} \langle \vec{\mathbf{b}}_1, \mathsf{ctx}_{\vec{\mathbf{y}}} \rangle - \mathsf{ctx}_{\mathbf{m}_1} \langle \vec{\mathbf{b}}_2, \mathsf{ctx}_{\vec{\mathbf{y}}} \rangle \right)$$

$$\mathsf{ctx}_{\mathbf{v}} = \langle \vec{\mathbf{b}}_4, \mathsf{ctx}_{\vec{\mathbf{y}}} \rangle + (\langle \vec{\mathbf{b}}_1, \mathsf{ctx}_{\vec{\mathbf{y}}} \rangle)(\langle \vec{\mathbf{b}}_2, \mathsf{ctx}_{\vec{\mathbf{y}}} \rangle)$$

Use GKS threshold homomorphic encryption to compute decryption shares
$\mathsf{ds}_{\mathbf{v},i} = \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ctx}_{\mathbf{v}})$ and broadcast them: $\mathbf{v} = \sum_{\mathcal{P}} \mathsf{ds}_{\mathsf{ctx}_{\mathbf{v}},i}$
Another reason is when $t+1$ or more parties compute a proof, since we can't simply reduce to single prover.
$\mathsf{ctx}_{\mathbf{z}} = \mathsf{ctx}_{\vec{\mathbf{y}}} + \mathbf{c}\mathsf{ctx}_{\vec{\mathbf{r}}}$.
Thresholdizing [ENS20] also requires products of secret values.

Conclusion

Thanks!

- Most efficient quantum-resistant threshold ZKP
  - Based on exact ZKPs [ALS20, ENS20]
  - Uses threshold homomorphic encryption [GKS24].
  - Uses HintMLWE to avoid rejection sampling [KLSS23].
- Also prove non-interactive variants of [ALS20, ENS20] without Fiat-Shamir heuristic using [AAB+24].
- Prove that MLWE is hard with secrets in lower rings similar to [BJRW23]

`https://eprint.iacr.org/2026/1750.pdf`

Threshold Lattice-Based Zero-Knowledge Proofs

Scott Griffy[1], Victor Youdom Kemmoe[1], Ngoc Khanh Nguyen[2], Tjerand Silde[3]

[1]Brown University [2]King's College London [3]Norwegian University of Science and Technology

icons from FlatIcon


Marius A. Aardal, Diego F. Aranha, Katharina Boudgoust, Sebastian Kolby, and Akira Takahashi.
Aggregating falcon signatures with LaBRADOR.
In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 71–106. Springer, Cham, August 2024.


Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler.
Practical product proofs for lattice commitments.
In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 470–499. Springer, Cham, August 2020.


Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza.
Zerocash: Decentralized anonymous payments from Bitcoin.
In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.


Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai.

Threshold cryptosystems from threshold fully homomorphic encryption.
In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Cham, August 2018.


Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan.
(Leveled) fully homomorphic encryption without bootstrapping.
In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.


Katharina Boudgoust, Corentin Jeudy, Adeline Roux-Langlois, and Weiqiang Wen.
On the hardness of module learning with errors with short distributions.
Journal of Cryptology, 36(1):1, January 2023.


Zvika Brakerski and Vinod Vaikuntanathan.
Efficient fully homomorphic encryption from (standard) LWE.
In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.


Jan Camenisch and Anna Lysyanskaya.
A signature scheme with efficient protocols.

In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, Berlin, Heidelberg, September 2003.


Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee.
Pixel: Multi-signatures for consensus.
In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2093–2110. USENIX Association, August 2020.


Muhammed F. Esgin, Ngoc Khanh Nguyen, and Gregor Seiler.
Practical exact proofs from lattices: New techniques to exploit fully-splitting rings.
In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 259–288. Springer, Cham, December 2020.


Andrea Flamini, Eysa Lee, and Anna Lysyanskaya.
Multi-holder anonymous credentials from BBS signatures.
In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 325–357. Springer, Cham, August 2025.


Kamil Doruk Gür, Jonathan Katz, and Tjerand Silde.

Two-round threshold lattice-based signatures from threshold homomorphic encryption.
In Markku-Juhani Saarinen and Daniel Smith-Tone, editors, *PQCrypto 2024, Part II*, pages 266–300. Springer, Cham, June 2024.


Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song.
Toward practical lattice-based proof of knowledge from hint-MLWE.
In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580. Springer, Cham, August 2023.


Vadim Lyubashevsky and Gregory Neven.
One-shot verifiable encryption from lattices.
In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 293–323. Springer, Cham, April / May 2017.


Vadim Lyubashevsky.
Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures.
In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, December 2009.


Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille.
Simple Schnorr multi-signatures with applications to Bitcoin.
DCC, 87(9):2139–2164, 2019.