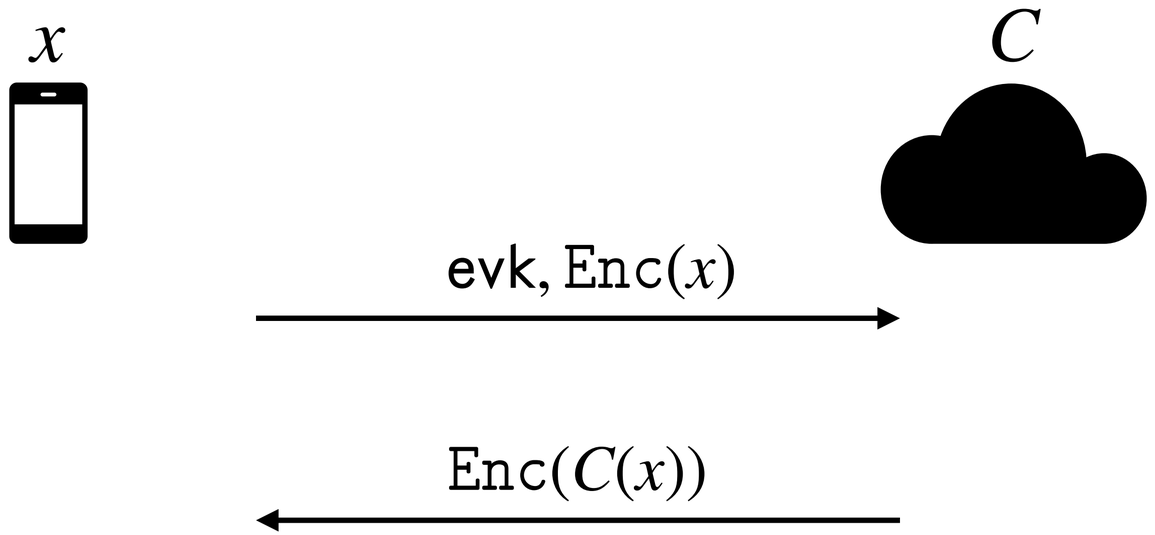
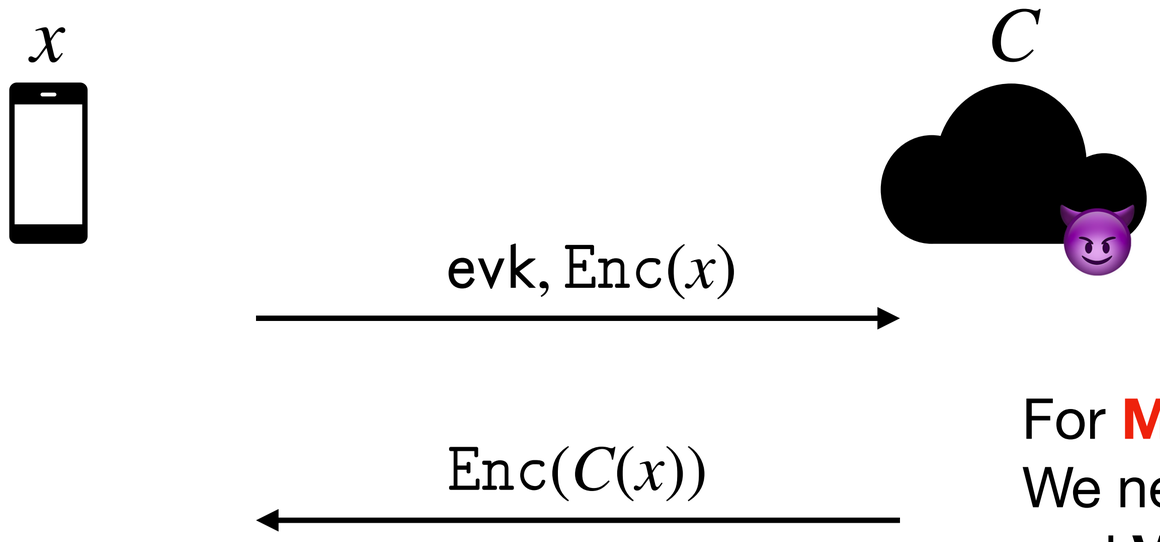
Practical Zero-Knowledge PIOP for Fully Homomorphic Encryption

Intak Hwang, Hyeonbum Lee, Seonhong Min, Jinyeong Seo and Yongsoo Song

Secure Protocols from FHE



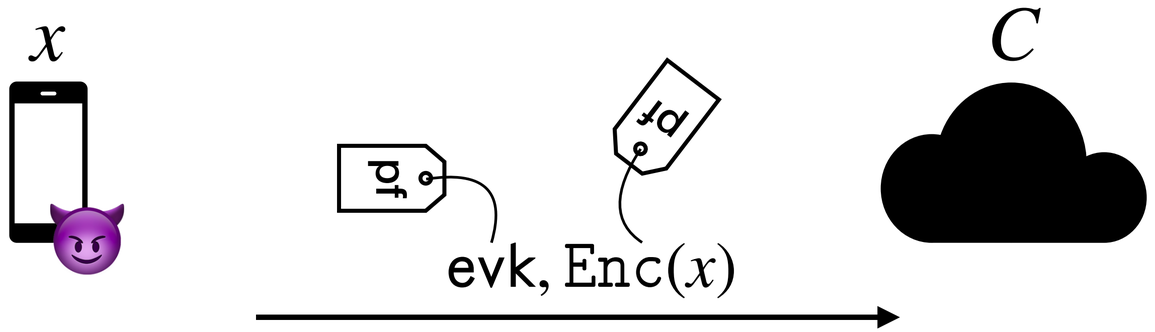
Secure Protocols from FHE



For **Malicious Servers**,
We need **Circuit Privacy**
and **Verifiable FHE**.

(See Previous Talk!)

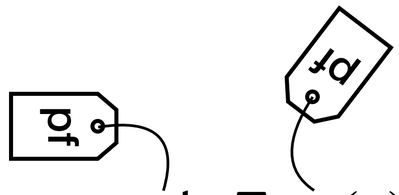
Secure Protocols from FHE



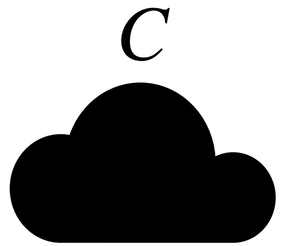
For **Malicious Clients**,
We need **Proofs of Keys and Ciphertexts**.

Secure Protocols from FHE

x



C



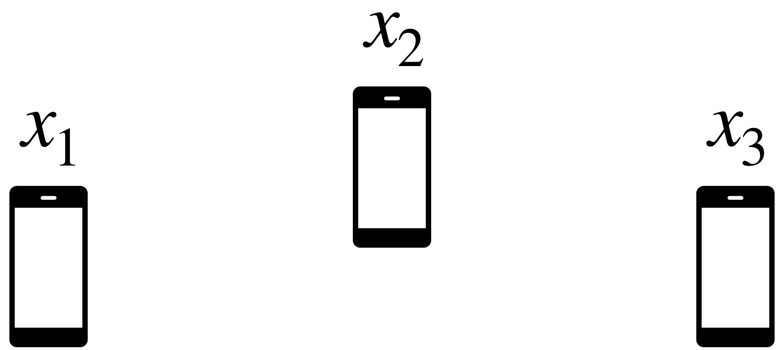
$\mathsf{evk}, \mathrm{Enc}(x)$

$\mathrm{Enc}(C(x))$

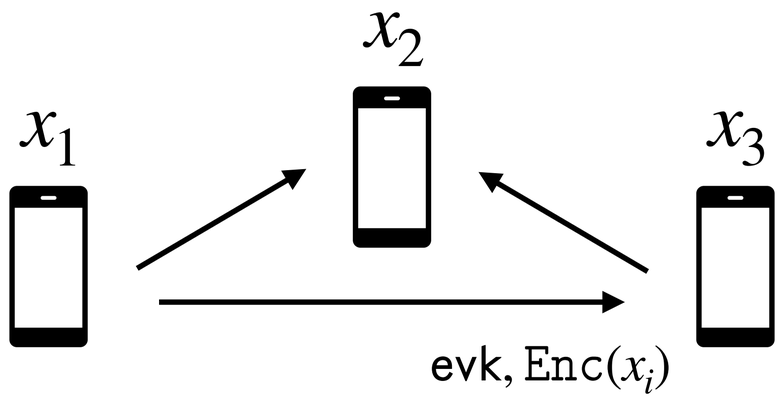
This Talk!

Malicious Clients, **Servers need Proofs of Keys and Ciphertexts.**

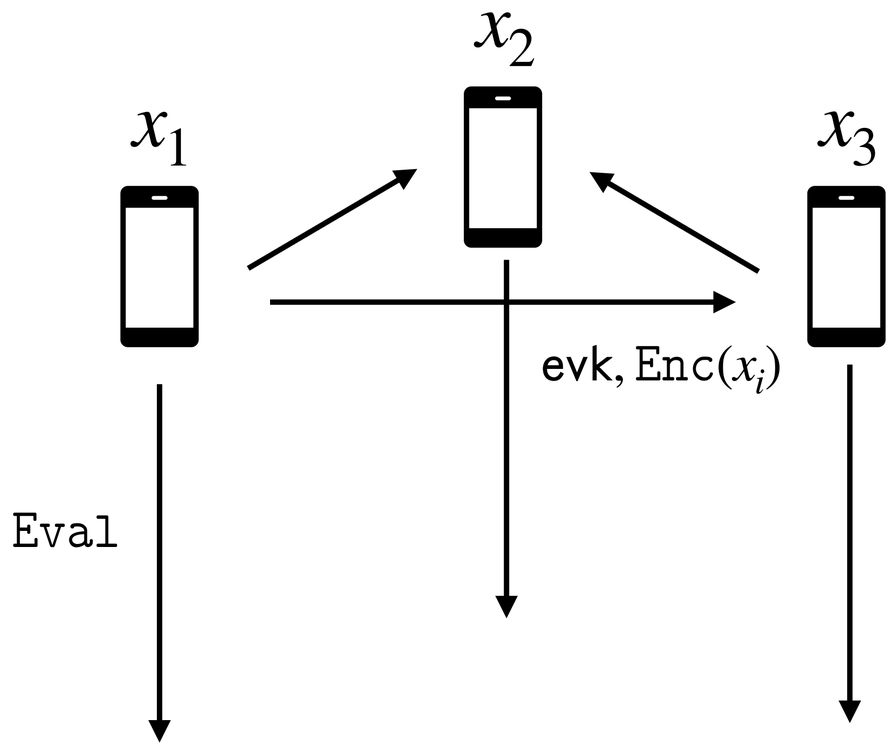
Zero-Knowledge Proofs for FHE



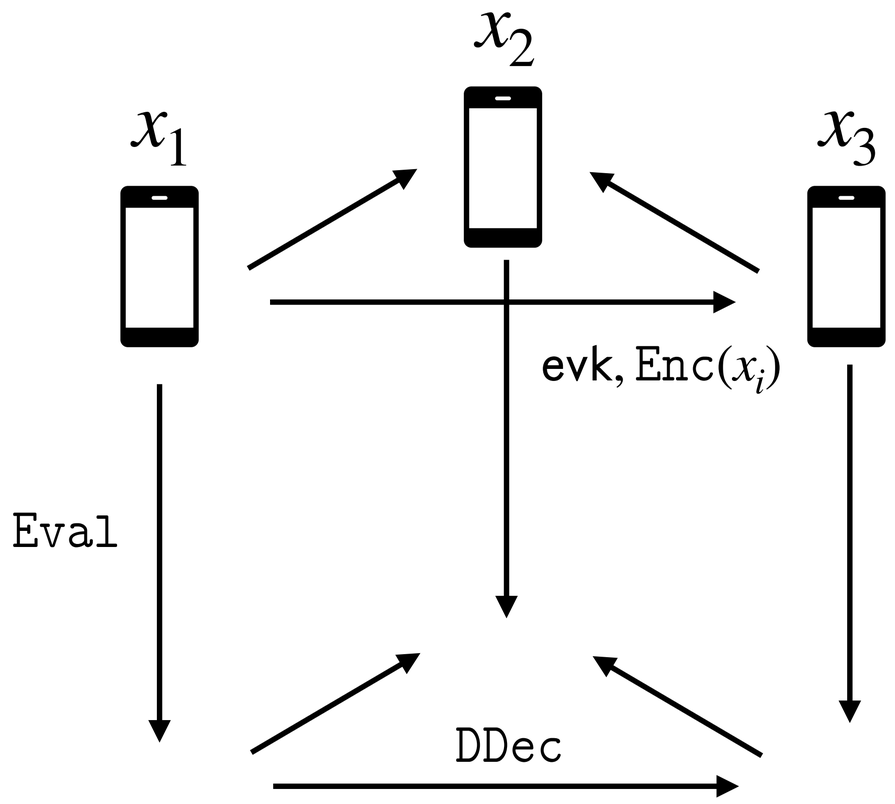
Zero-Knowledge Proofs for FHE



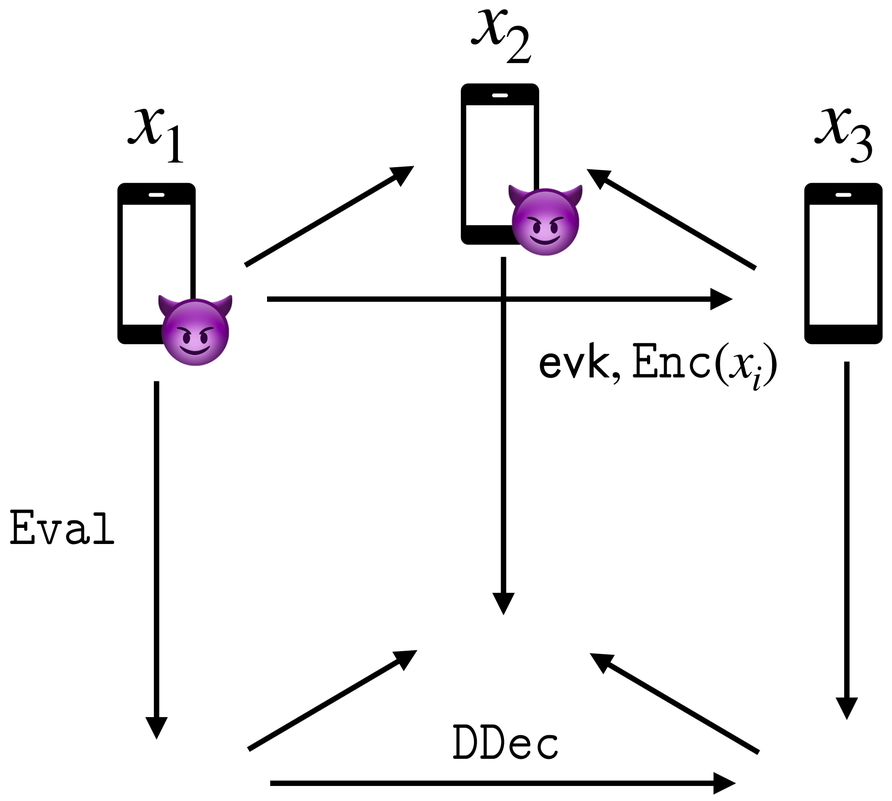
Zero-Knowledge Proofs for FHE



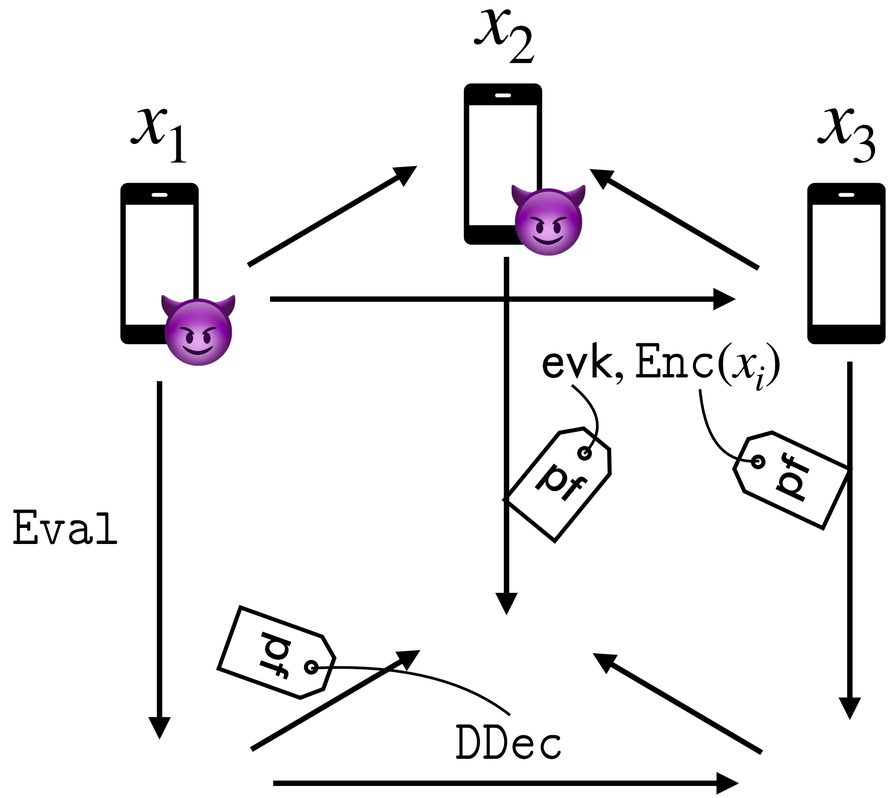
Zero-Knowledge Proofs for FHE



Zero-Knowledge Proofs for FHE



Zero-Knowledge Proofs for FHE



In Multi-Party / Threshold FHE, Proof of Keys & Ciphertexts are enough for active security [AJLA+12]

Zero-Knowledge Proofs for FHE

Zero-Knowledge Proofs for FHE

- Long lines of work to prove validity of Public Key & Ciphertexts [CMS+23,Bot24,…]
  - No known benchmarks for proving evaluation keys

Zero-Knowledge Proofs for FHE

- Long lines of work to prove validity of Public Key & Ciphertexts [CMS+23,Bot24,…]
  - No known benchmarks for proving evaluation keys

- Problems:
  - No Efficient “arithmetization” for RLWE relations
  - Incompatibility between FHE and ZKP modulus
  - Lack of “zero-knowledge”

Our Work

Our Work

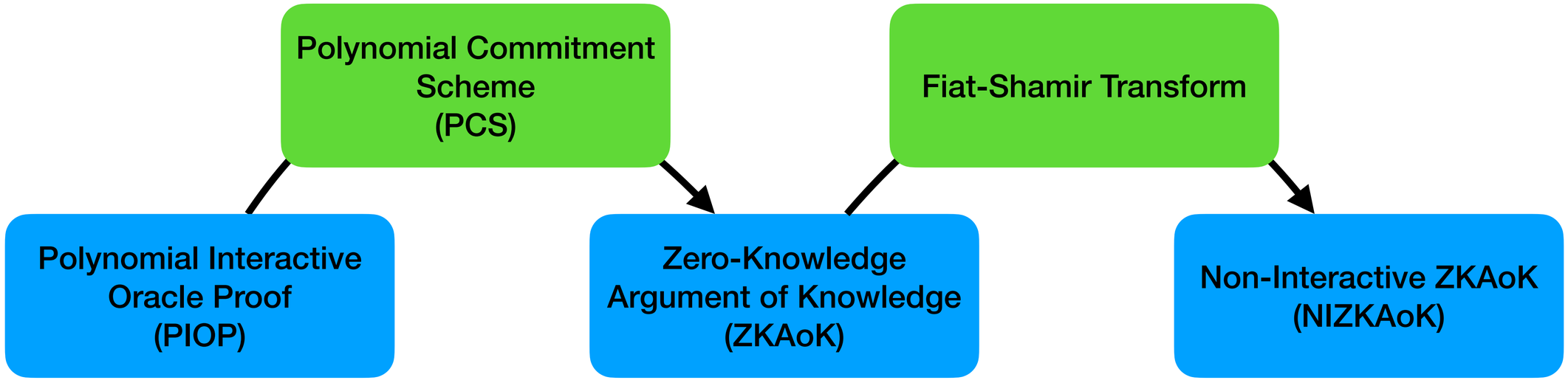
- We introduce a general proof system for FHE Keys & Ciphertexts
  - General enough to prove all widely-used RLWE relations
  - Specifically targets FHE parameter regime
  - Efficiently supports zero-knowledge

Our Work

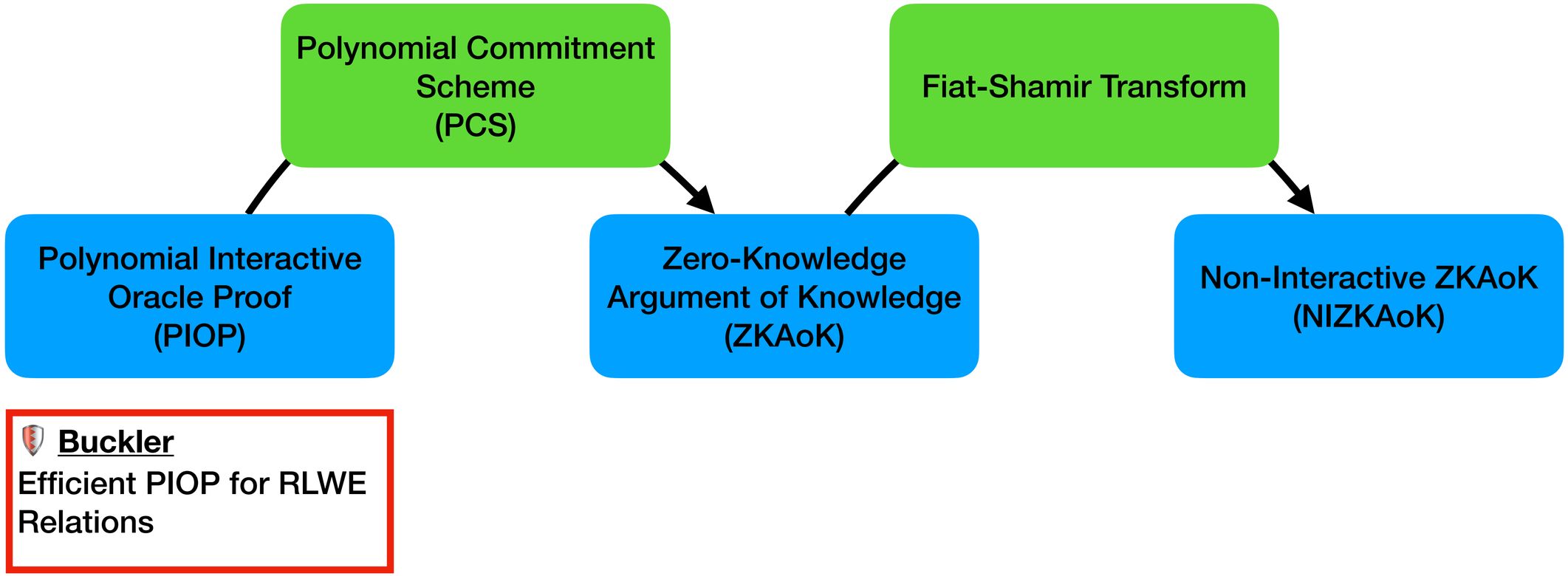
- We introduce a general proof system for FHE Keys & Ciphertexts
  - General enough to prove all widely-used RLWE relations
  - Specifically targets FHE parameter regime
  - Efficiently supports zero-knowledge

- First work to give concrete results for all client-sent materials, including:
  - Relinearization & Automorphism Keys for BFV, BGV, CKKS
  - Blind Rotation & Keyswitch Keys for TFHE

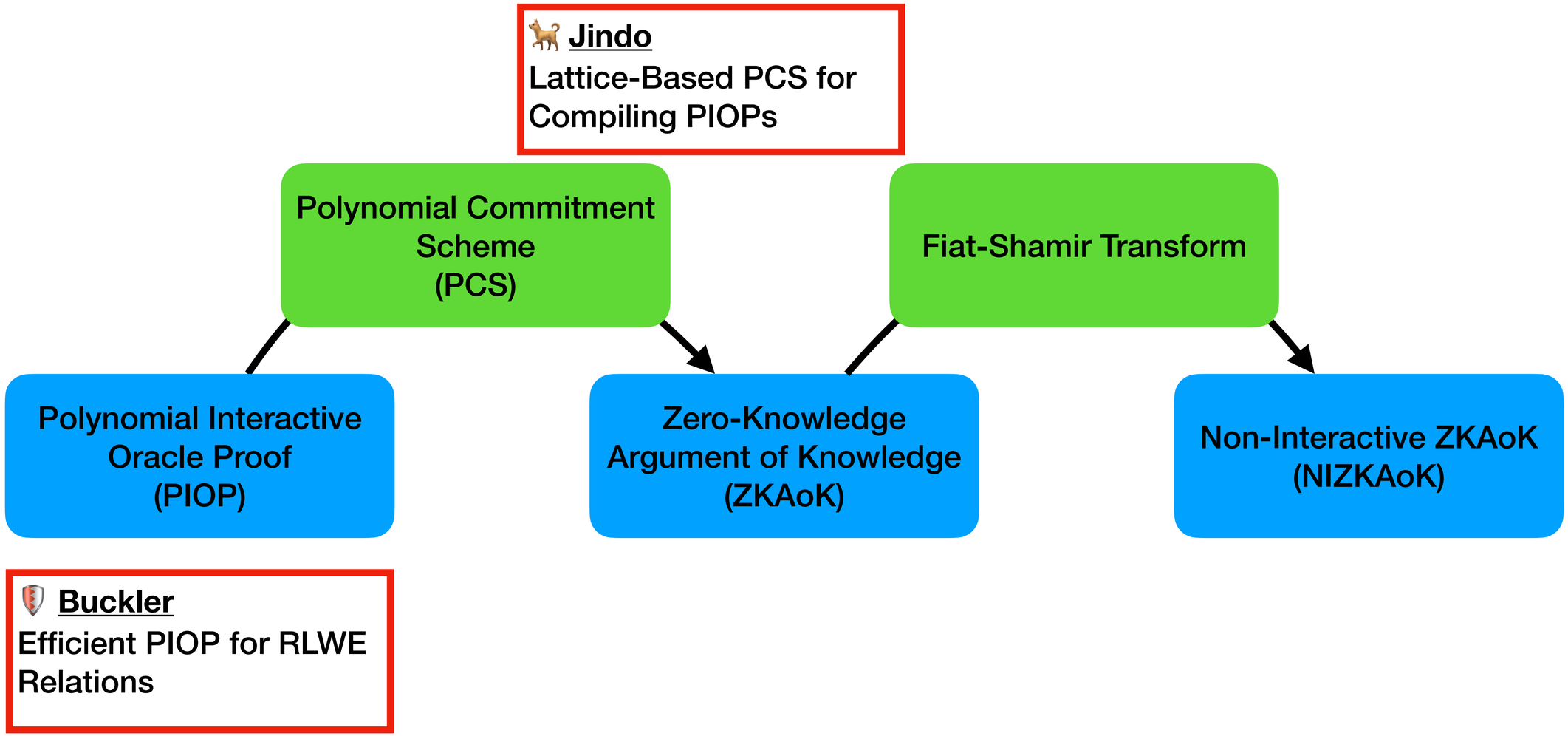
The PIOP Framework



The PIOP Framework



The PIOP Framework



Sneak-Peek at Univariate PIOPs

Sneak-Peek at Univariate PIOPs

- Most ZK-SNARKs proves the R1CS constraints:

$$A\vec{z} \odot B\vec{z} = C\vec{z} \pmod{p}$$

Sneak-Peek at Univariate PIOPs

- Most ZK-SNARKs proves the R1CS constraints:

$$A\vec{z} \odot B\vec{z} = C\vec{z} \pmod{p}$$

- Univariate PIOP-based SNARKs proves R1CS relation by checking:
  - Linear Check: $\vec{a} = A\vec{z},\ \vec{b} = B\vec{z},\ \vec{c} = C\vec{z}$
  - Arithmetic Check: $\vec{a} \odot \vec{b} = \vec{c}$

Buckler: Efficient PIOP for RLWE Relations

 Buckler: Efficient PIOP for RLWE Relations

- Our Observation: Arithmetic & Linear Check is enough to prove RLWE relations!

Buckler: Efficient PIOP for RLWE Relations

- Our Observation: Arithmetic & Linear Check is enough to prove RLWE relations!

Target Relation: $b = -as + Bt + e,\ \|s\|_\infty, \|e\|_\infty \leq 1$

Buckler: Efficient PIOP for RLWE Relations

- Our Observation: Arithmetic & Linear Check is enough to prove RLWE relations!

Target Relation: $b = -as + Bt + e,\ \|s\|_\infty, \|e\|_\infty \leq 1$

Arithmetic Checks

- $\vec{b}_{\text{NTT}} = -\vec{a}_{\text{NTT}} \odot \vec{s}_{\text{NTT}} + B\vec{s}^{\phi}_{\text{NTT}} + \vec{e}_{\text{NTT}}$

Buckler: Efficient PIOP for RLWE Relations

- Our Observation: Arithmetic & Linear Check is enough to prove RLWE relations!

Target Relation: $b = -as + Bt + e,\ \|s\|_\infty, \|e\|_\infty \leq 1$

Arithmetic Checks

- $\vec{b}_{\text{NTT}} = -\vec{a}_{\text{NTT}} \odot \vec{s}_{\text{NTT}} + B\vec{s}^{\phi}_{\text{NTT}} + \vec{e}_{\text{NTT}}$
- $(\vec{s} - \vec{1}) \odot \vec{s} \odot (\vec{s} + \vec{1}) = 0$
- $(\vec{e} - \vec{1}) \odot \vec{e} \odot (\vec{e} + \vec{1}) = 0$

Buckler: Efficient PIOP for RLWE Relations

- Our Observation: Arithmetic & Linear Check is enough to prove RLWE relations!

Target Relation: $b = -as + Bt + e,\ \|s\|_\infty, \|e\|_\infty \leq 1$

Arithmetic Checks

- $\vec{b}_{\text{NTT}} = -\vec{a}_{\text{NTT}} \odot \vec{s}_{\text{NTT}} + B\vec{s}^{\phi}_{\text{NTT}} + \vec{e}_{\text{NTT}}$
- $(\vec{s} - \vec{1}) \odot \vec{s} \odot (\vec{s} + \vec{1}) = 0$
- $(\vec{e} - \vec{1}) \odot \vec{e} \odot (\vec{e} + \vec{1}) = 0$

Linear Checks

- $\vec{s}_{\text{NTT}} = \text{NTT} \cdot \vec{s}$
- $\vec{e}_{\text{NTT}} = \text{NTT} \cdot \vec{e}$

Buckler: Efficient PIOP for RLWE Relations

- Our Observation: Arithmetic & Linear Check is enough to prove RLWE relations!

Target Relation: $b = -as + Bt + e,\ \|s\|_\infty, \|e\|_\infty \leq 1$

Arithmetic Checks

- $\vec{b}_{\text{NTT}} = -\vec{a}_{\text{NTT}} \odot \vec{s}_{\text{NTT}} + B\vec{s}^{\phi}_{\text{NTT}} + \vec{e}_{\text{NTT}}$
- $(\vec{s} - \vec{1}) \odot \vec{s} \odot (\vec{s} + \vec{1}) = 0$
- $(\vec{e} - \vec{1}) \odot \vec{e} \odot (\vec{e} + \vec{1}) = 0$
- $\vec{t}_{\text{NTT}} = \vec{s}_{\text{NTT}} \odot \vec{s}_{\text{NTT}}$ (Relinearization)

Linear Checks

- $\vec{s}_{\text{NTT}} = \text{NTT} \cdot \vec{s}$
- $\vec{e}_{\text{NTT}} = \text{NTT} \cdot \vec{e}$

Buckler: Efficient PIOP for RLWE Relations

- Our Observation: Arithmetic & Linear Check is enough to prove RLWE relations!

Target Relation: $b = -as + Bt + e,\ \|s\|_\infty, \|e\|_\infty \leq 1$

Arithmetic Checks

- $\vec{b}_{\text{NTT}} = -\vec{a}_{\text{NTT}} \odot \vec{s}_{\text{NTT}} + B\vec{s}^{\phi}_{\text{NTT}} + \vec{e}_{\text{NTT}}$
- $(\vec{s} - \vec{1}) \odot \vec{s} \odot (\vec{s} + \vec{1}) = 0$
- $(\vec{e} - \vec{1}) \odot \vec{e} \odot (\vec{e} + \vec{1}) = 0$
- $\vec{t}_{\text{NTT}} = \vec{s}_{\text{NTT}} \odot \vec{s}_{\text{NTT}}$ (Relinearization)

Linear Checks

- $\vec{s}_{\text{NTT}} = \text{NTT} \cdot \vec{s}$
- $\vec{e}_{\text{NTT}} = \text{NTT} \cdot \vec{e}$
- $\vec{t}_{\text{NTT}} = \phi \cdot \vec{s}_{\text{NTT}}$ (Automorphism)

Buckler: Efficient PIOP for RLWE Relations

- Problem: Mismatch between FHE and ZKP space

Buckler: Efficient PIOP for RLWE Relations

- Problem: Mismatch between FHE and ZKP space

FHE

- Works over R_q
- $q = \prod_i q_i$
- $q_i < 2^{64}$

Buckler: Efficient PIOP for RLWE Relations

- Problem: Mismatch between FHE and ZKP space

FHE

- Works over R_q
- $q = \prod_i q_i$
- $q_i < 2^{64}$

ZKP

- Works over $\mathbb{F}_p$
- p: power-of-prime
- $p \geq 2^{\lambda}$

Buckler: Efficient PIOP for RLWE Relations

- Problem: Mismatch between FHE and ZKP space

FHE

- Works over R_q
- $q = \prod_i q_i$
- $q_i < 2^{64}$

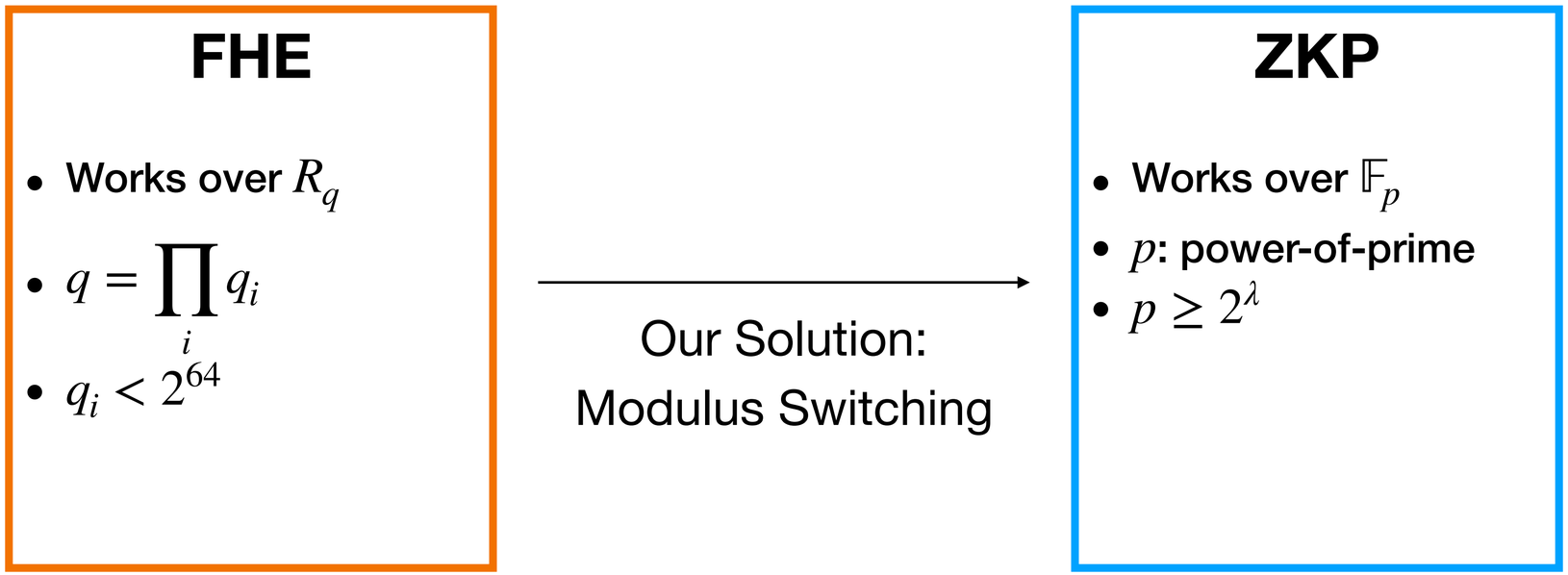
Our Solution:
Modulus Switching

ZKP

- Works over $\mathbb{F}_p$
- p: power-of-prime
- $p \geq 2^{\lambda}$

Buckler: Efficient PIOP for RLWE Relations

- Problem: Mismatch between FHE and ZKP space



$$b = -as + e \pmod{q},\ \|e\|_\infty \leq \beta$$

$$b' = -a's + e' \pmod{p},\ \|e'\|_\infty \leq \beta'$$

$\beta \approx \beta'$ if $q \approx p$

Example: Proving Evaluation Keys

- We prove the validity of Relinearization Key & Two Automorphism Keys
  - Automorphism Keys for $X \mapsto X^{-1}, X^5$ are sufficient [LLK+22]

Example: Proving Evaluation Keys

- We prove the validity of Relinearization Key & Two Automorphism Keys
  - Automorphism Keys for $X \mapsto X^{-1}, X^5$ are sufficient [LLK+22]

- Ternary Secret Key:

$$\|s\|_\infty \leq 1$$

- Relinearization Key:

$$\vec{b} = -\vec{a}s + \vec{g}s^2 + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

- Automorphism Key(s) for $X \mapsto X^{-1}, X^5$:

$$\vec{b} = -\vec{a}s + \vec{g}s(X^i) + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

Example: Proving Evaluation Keys

- We prove the validity of Relinearization Key & Two Automorphism Keys
  - Automorphism Keys for $X \mapsto X^{-1}, X^5$ are sufficient [LLK+22]

- Ternary Secret Key:

$$\|s\|_\infty \leq 1$$

- Relinearization Key:

$$\vec{b} = -\vec{a}s + \vec{g}s^2 + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

- Automorphism Key(s) for $X \mapsto X^{-1}, X^5$:

$$\vec{b} = -\vec{a}s + \vec{g}s(X^i) + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

$$N = 2^{14},\ \log q = 440$$

Prover Time	Verifier Time	Proof Size	Key Size
9.6 s	0.60 s	1.14MB	37.85MB

All benchmarks were run in MacBook Air M4, single-core.

Example: Proving CKKS Ciphertext

- Target Relation:

  - Ternary Secret Key:

    $$\|s\|_\infty \leq 1$$

  - Ciphertext:

    $$b = -as + m + e \wedge \|e\|_\infty \leq \beta$$

  - Norm Bound of message:

    $$|m(\zeta^{2i+1})| \leq \Delta \text{ for } i = 0, \ldots, N-1$$

- $m(\zeta^{2i+1}) \in \mathbb{C}$, which PIOP cannot efficiently prove

Example: Proving CKKS Ciphertext

- Our Solution: Use the Coeff-To-Slot transformation
  - **Prover**: Encodes the messages to *coefficients*, prove that $\|m\|_\infty \leq \Delta$
  - **Verifier**: Performs Coeff-To-Slot on the ciphertext after verification

Example: Proving CKKS Ciphertext

- Our Solution: Use the Coeff-To-Slot transformation
  - **Prover**: Encodes the messages to *coefficients*, prove that $\|m\|_\infty \leq \Delta$
  - **Verifier**: Performs Coeff-To-Slot on the ciphertext after verification

- Ternary Secret Key:

$$\|s\|_\infty \leq 1$$

- Ciphertext:

$$b = -as + m + e \wedge \|e\|_\infty \leq \beta$$

- Norm Bound of message:

$$\|m\|_\infty \leq \Delta$$

Example: Proving CKKS Ciphertext

- Our Solution: Use the Coeff-To-Slot transformation
  - **Prover**: Encodes the messages to *coefficients*, prove that $\|m\|_\infty \leq \Delta$
  - **Verifier**: Performs Coeff-To-Slot on the ciphertext after verification

- Ternary Secret Key:

$$\|s\|_\infty \leq 1$$

- Ciphertext:

$$b = -as + m + e \wedge \|e\|_\infty \leq \beta$$

- Norm Bound of message:

$$\|m\|_\infty \leq \Delta$$

$$N = 2^{14},\ \log q = 440,\ \log \Delta = 40$$

Prover Time	Verifier Time	Proof Size	Ct Size
7.9s	0.07s	1.13MB	1.80MB

All benchmarks were run in MacBook Air M4, single-core.

Example: Proving TFHE Bootstrapping Key

- TFHE has much more complex key structure:

- Binary Secret Key:

$$s_{i\in[0,N)} \in \{0,1\} \wedge s'_{i\in[0,n)} \in \{0,1\}$$

- Blind Rotation Key:

$$\vec{b}_0 = -\vec{a}_0 s + \vec{g}s'_i + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

$$\vec{b}_1 = -\vec{a}s + \vec{g}ss'_i + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

- Key Switch Key:

$$\vec{b} = -\langle \vec{a}, \vec{s}' \rangle + \vec{g}s_i + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

Example: Proving TFHE Bootstrapping Key

Example: Proving TFHE Bootstrapping Key

- **Too many constraints!** → **Use compression techniques**
  - Blind Rotation Keys can be compressed to Relinearization + Galois Keys [KDE+23]
  - Key Switching Keys can be compressed to single RGSW ciphertext [XZW+24]

Example: Proving TFHE Bootstrapping Key

- **Too many constraints!** → **Use compression techniques**
  - Blind Rotation Keys can be compressed to Relinearization + Galois Keys [KDE+23]
  - Key Switching Keys can be compressed to single RGSW ciphertext [XZW+24]

- **Not Enough Soundness!** → **Modulus switch during decompression**
  - Guarantees sufficient soundness
  - Decompression noise is suppressed

Example: Proving TFHE Bootstrapping Key

- Binary Secret Key:

$$s_{i\in[0,N)} \in \{0,1\} \wedge s'_{i\in[0,n)} \in \{0,1\}$$

- Packed Blind Rotation Key:

$$\vec{b} = -\vec{a}s + \vec{g}s'_i + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

- Packed Key Switch Key:

$$\vec{b} = -\vec{a}s' + \vec{g}s + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

- Relinearize & Automorphism Keys as before

Example: Proving TFHE Bootstrapping Key

- Binary Secret Key:

$$s_{i\in[0,N)} \in \{0,1\} \wedge s'_{i\in[0,n)} \in \{0,1\}$$

- Packed Blind Rotation Key:

$$\vec{b} = -\vec{a}s + \vec{g}s'_i + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

- Packed Key Switch Key:

$$\vec{b} = -\vec{a}s' + \vec{g}s + \vec{e} \wedge \|\vec{e}\|_\infty \leq \beta$$

- Relinearize & Automorphism Keys as before

$$N = 2^{12}, n = 850$$

Prover Time	Verifier Time	Proof Size	Key Size
3.5s	0.08s	4.90MB	1.80MB

All benchmarks were run in MacBook Air M4, single-core.

🥁Ringo-SNARK

- Incorporating zero-knowledge proofs to existing libraries are cumbersome
- We introduce **Ringo-SNARK**:
  - Open-Source
  - Written in Go, can be seamlessly integrated to existing libraries (e.g. Lattigo)
  - Automatic parameter selection

Using Ringo-SNARK

```
type PublicKeyCircuit[Q bignum.Uint[Q]] struct {
    NTTChecker buckler.LinearTransformer[Q]

    PublicKeyNTT [2]buckler.PublicWitness[Q]

    SecretKey    buckler.Witness[Q]
    SecretKeyNTT buckler.Witness[Q]

    Noise    buckler.Witness[Q]
    NoiseNTT buckler.Witness[Q]
}
```

Using Ringo-SNARK

```
func (c *PublicKeyCircuit[Q]) Define(ctx *buckler.Context[Q]) {
    // skNTT = NTT * sk
    ctx.AddLinearConstraint(c.SecretKeyNTT, c.SecretKey, c.NTTTransformer)
    // eNTT = NTT * e
    ctx.AddLinearConstraint(c.NoiseNTT, c.Noise, c.NTTTransformer)

    // |s|, |e| <= 1
    ctx.AddInfNormConstraint(c.SecretKey, 1)
    ctx.AddInfNormConstraint(c.Noise, 1)
```

Using Ringo-SNARK

```
    // pk[1] + pk[0] * sk - e = 0
    var pkConstraint buckler.ArithmeticConstraint[Q]
    pkConstraint.AddTerm(c.PublicKeyNTT[1])
    pkConstraint.AddTerm(c.PublicKeyNTT[0], c.SecretKeyNTT)
    pkConstraint.SubTerm(nil, c.NoiseNTT)
    ctx.AddArithmeticConstraint(pkConstraint)
}
```

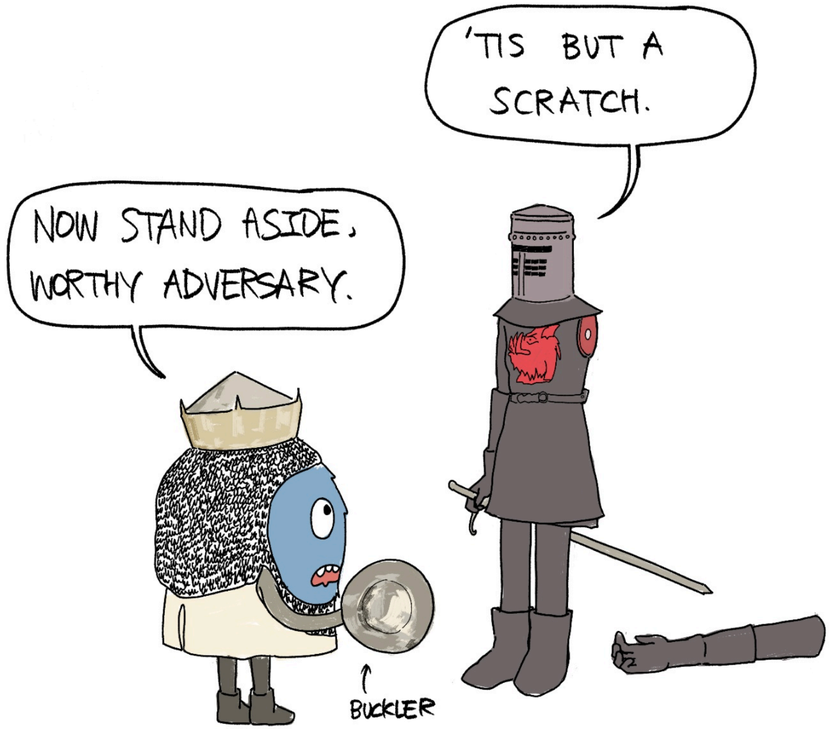
Using Ringo-SNARK

```
circuit := &PublicKeyCircuit[Q]{
    NTTTransformer: buckler.NewNTTTransformer[Q](N),
}

prv, vrf, err := buckler.Compile(N, circuit, crs)

assign := &PublicKeyCircuit[Q] { ... }
pf, err := prv.Prove(assign)

pubAssign := &PublicKeyCircuit[Q] { ... }
ok := vrf.Verify(assign, pf)
```



Thanks!

github.com/sp301415/ringo-snark

* No Generative AI used