



Bocconi



Schmivitz: VOLEitH based ZK gadgets for threshold cryptography

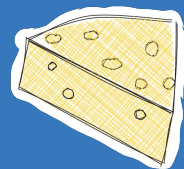
MPTS 2026

January 29, 2026

Carsten Baum, Emanuela Orsini, Peter Scholl, Benoit Razet, Marcella Hastings, Shibam Mukherjee, Christian Rechberger, **James Parker**

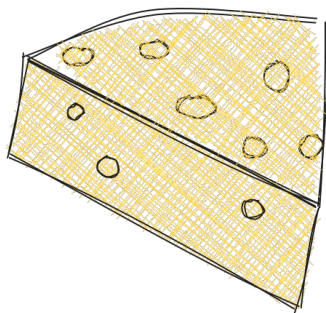
nist-vole@galois.com

Thanks to Carsten and Lance for slides



Introduction

Do you need a fast, linear-time ZKP?



Schmivitz highlights:

- Linear runtime and proof size (small constants)
- Flexible choice of field (binary/arithmetic)
- Non-interactive
- Post-Quantum (PQ) secure

Potential use cases

Smaller ZK statements

- Ex: FAEST - PQ digital signature algorithm, proves AES in ~1ms

Likely compatible with existing VOLE protocols

- Field conversions [BBMRS21]
- Free disjunctions [BMRS21]
- RAM [GHK24]
- Succinct loops [WYYXW22]
- Ex: Succinct PQ threshold ring signatures [CDDEKS25]

Overview

Presenting Schmivitz, a VOLEith based ZKP

- Generalization of the FAEST PQ signature scheme for arbitrary statements [BBGKORS23, BBMORRRS24, BBBGKMMORRRS25]
- Submitting two ZK gadgets for threshold and multiparty cryptography:
 - a. Weak VOLEs
 - b. VOLEith

Security properties

- Zero-knowledge argument of knowledge
- No trusted setup
- Publicly verifiable
- Target: 128 bits of security

Conservative assumptions:

- Standard, symmetric-key primitives
- Random oracle: SHA3
- PRG: AES

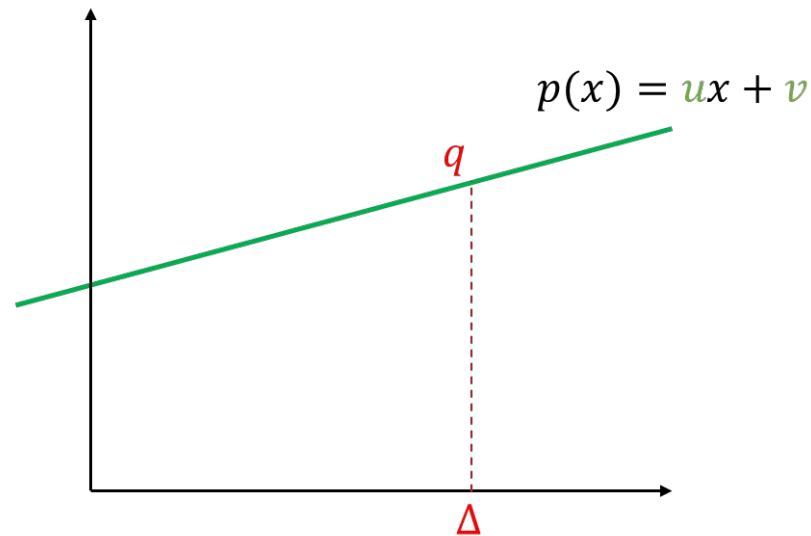
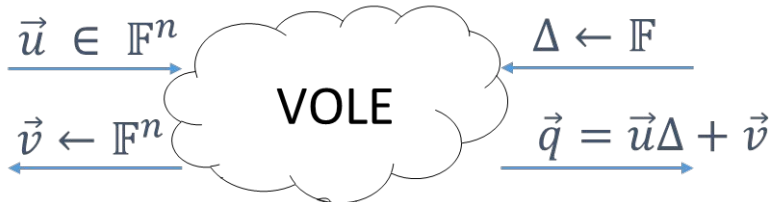
Background: VOLE ZK



Prover



Verifier



Linearly homomorphic commitments

Hiding u_i since v_i is random

Binding since guessing Δ is $1/|\mathbb{F}|$

Background: Extended witness commitment



Prover



Verifier

$$\vec{u} \in \mathbb{F}^n \quad \vec{v} \in \mathbb{F}^n \quad \vec{w} \in \mathbb{F}^l$$

$$\vec{q} = \vec{u} \Delta + \vec{v} \quad \Delta \in \mathbb{F}$$

$$d_i = w_i - u_i$$

d_i



$$\begin{aligned} q_i' &= q_i + d_i \Delta \\ &= (u_i + d_i) \Delta + v_i \\ &= (u_i + w_i - u_i) \Delta + v_i \\ &= w_i \Delta + v_i \end{aligned}$$

Background: Addition gates



Prover

$$\vec{u} \in \mathbb{F}^n \quad \vec{v} \in \mathbb{F}^n \quad \vec{w} \in \mathbb{F}^l$$

$$w_k = w_i + w_j$$

$$v_k = v_i + v_j$$



Verifier

$$\vec{q} = \vec{u} \Delta + \vec{v} \quad \Delta \in \mathbb{F}$$

$$\begin{aligned} q_k &= q_i + q_j \\ &= (w_i \Delta + v_i) + (w_j \Delta + v_j) \\ &= (w_i + w_j) \Delta + (v_i + v_j) \end{aligned}$$

Note: No communication

Background: Multiplication gates

[DIO21, YSWW21]



Prover



Verifier

$$\vec{u} \in \mathbb{F}^n \quad \vec{v} \in \mathbb{F}^n \quad \vec{w} \in \mathbb{F}^l$$

$$\vec{q} = \vec{u} \Delta + \vec{v} \quad \Delta \in \mathbb{F}$$

$$d_k = w_i * w_j - u_k \quad \xrightarrow{d_k} \quad \begin{aligned} q_k' &= q_k + d_k \Delta \\ &= (u_k + d_k) \Delta + v_k \\ &= (u_k + w_i * w_j - u_k) \Delta + v_k \\ v_k &= (w_i * w_j) \Delta + v_k \end{aligned}$$

Quicksilver check: Efficient (batched) check that d_k was computed correctly

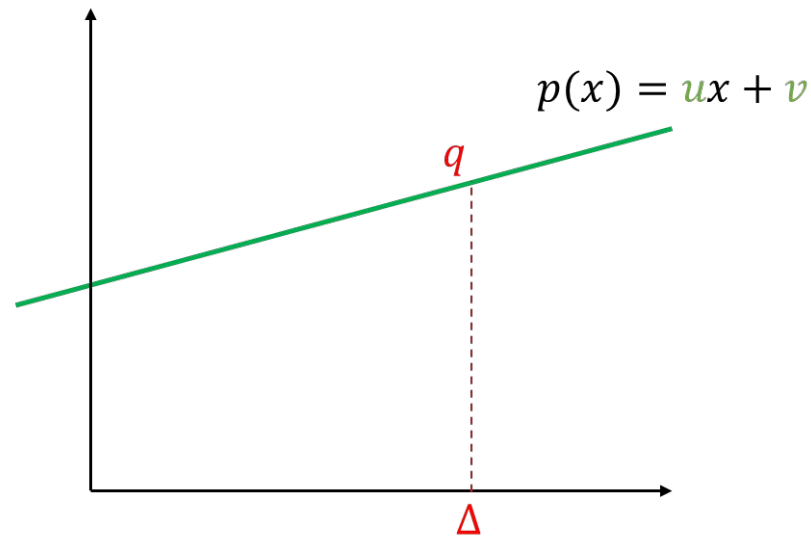
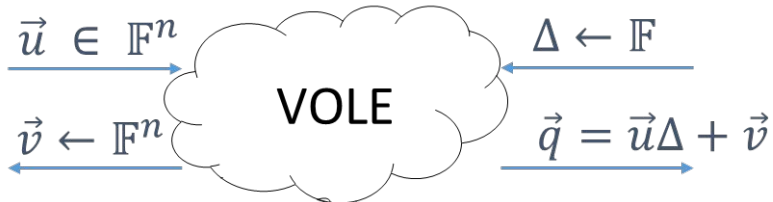
Gadget 1: Weak VOLEs



Prover



Verifier



Linearly homomorphic commitments

Hiding u_i since v_i is random

Binding since guessing Δ is $1/|\mathbb{F}|$

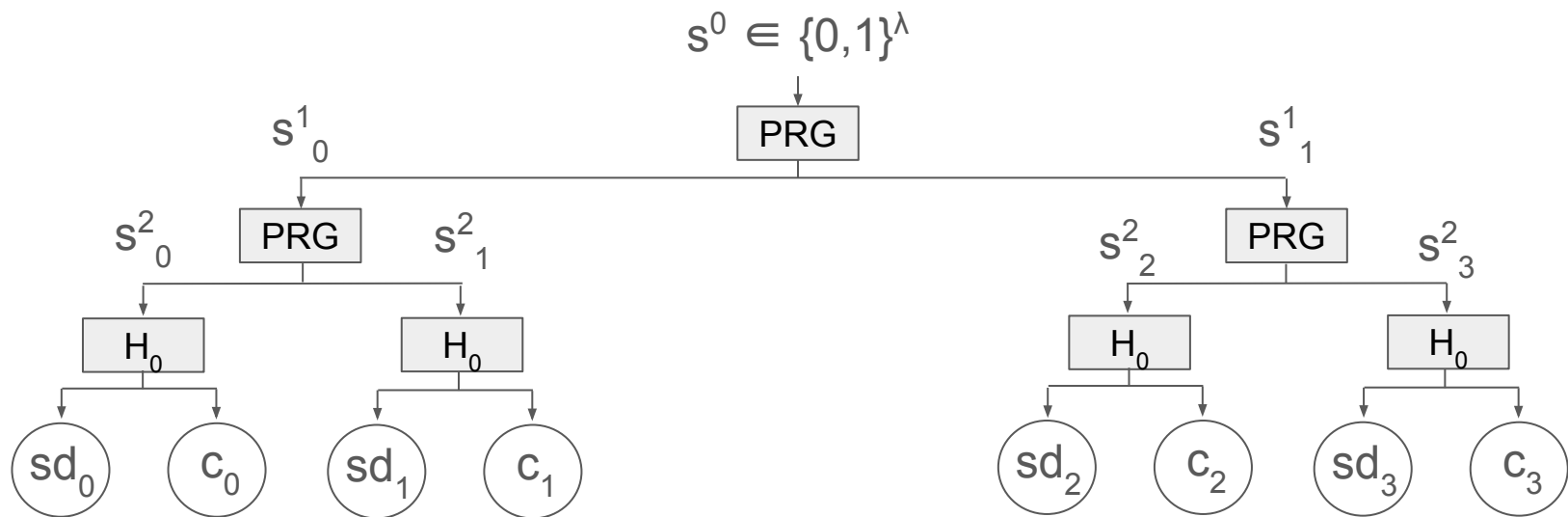
Requires prover does not know Δ .
Is this necessary?

Gadget 1: Weak VOLEs

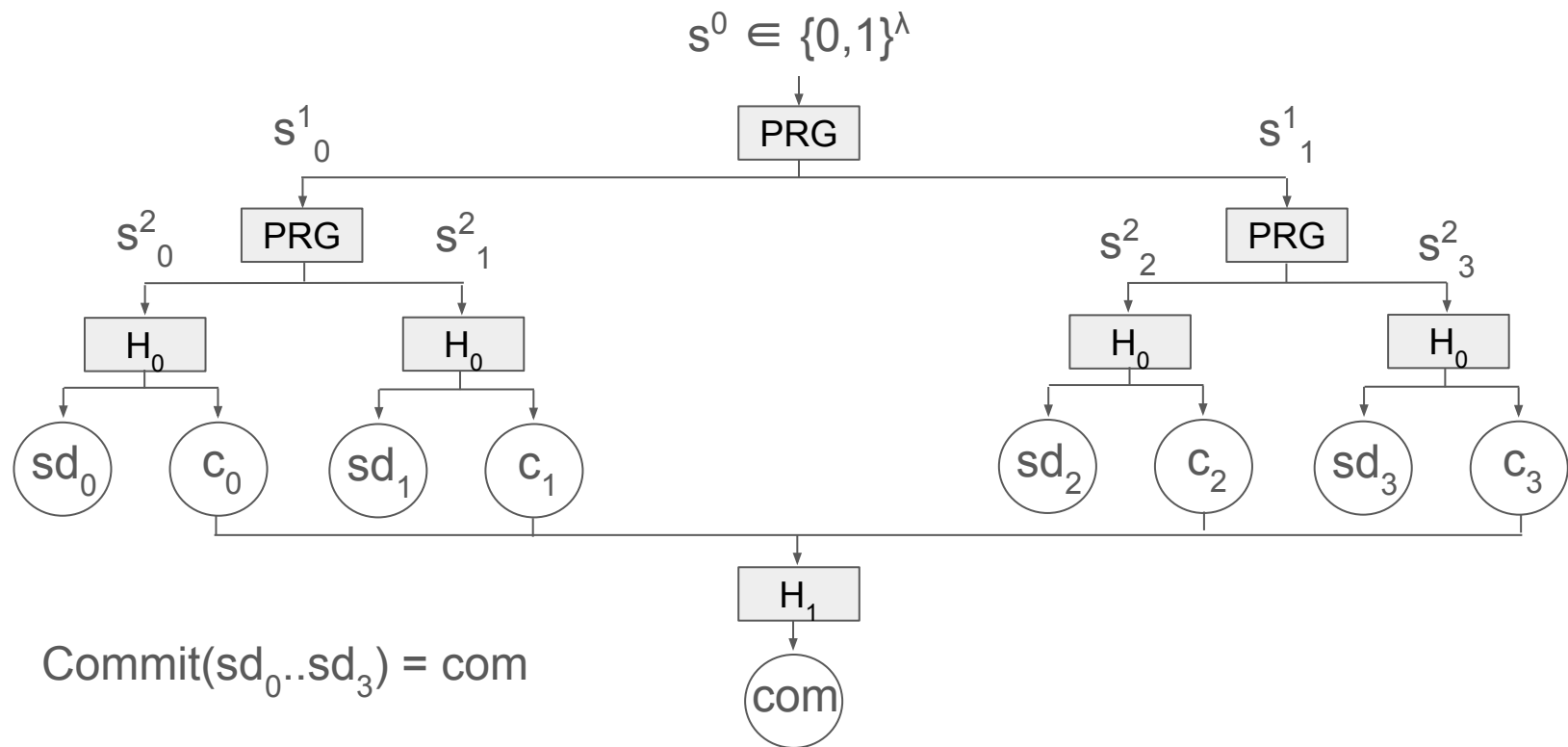
$$\vec{q} = \vec{u}\Delta + \vec{v}$$

- Instead of learning VOLE shares simultaneously, prover learns its shares, commits to them, and then Δ is generated via Fiat-Shamir
- Enables publicly verifiable, non-interactive ZKPs
- Likely useful in other applications like MPC

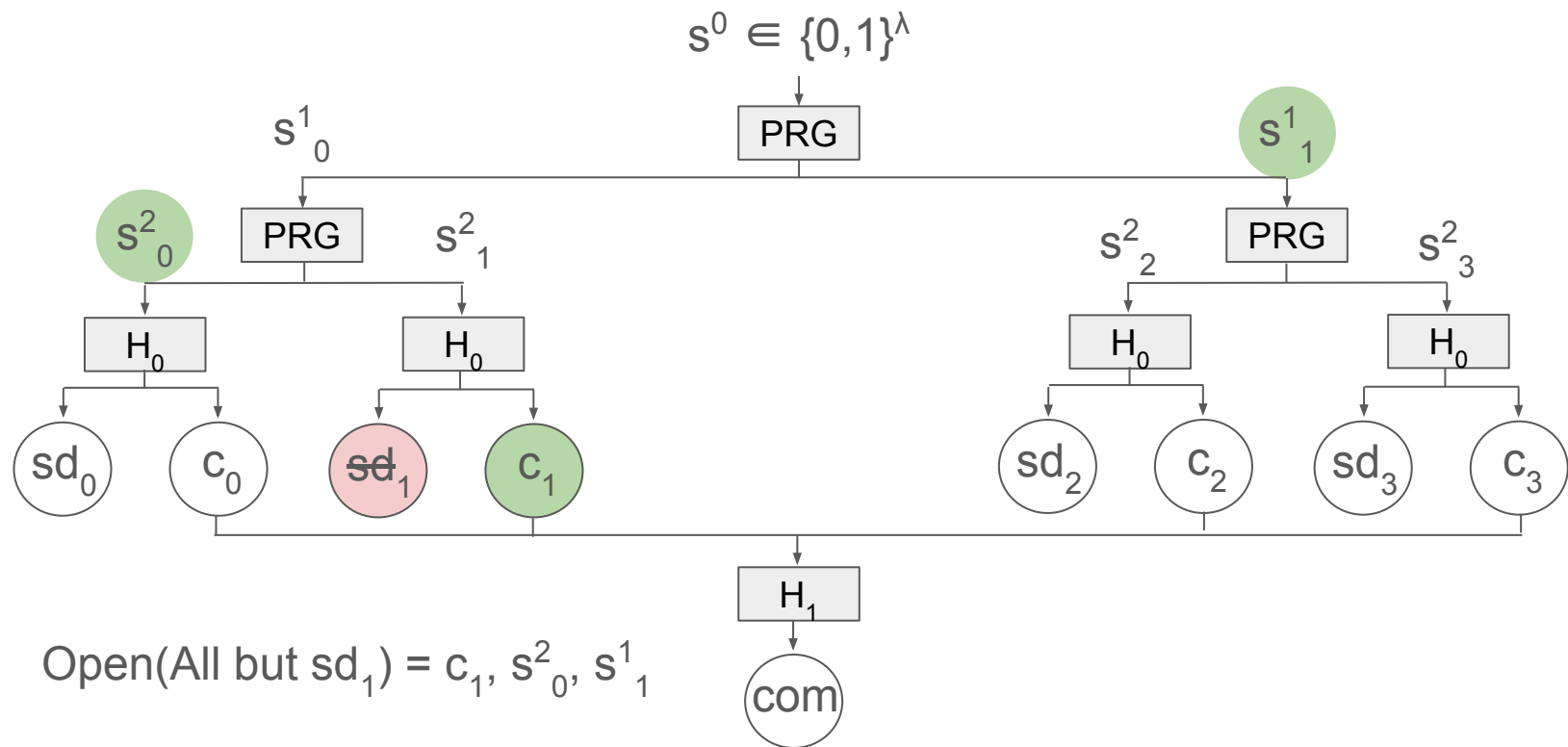
All-But-One Random Vector Commitments



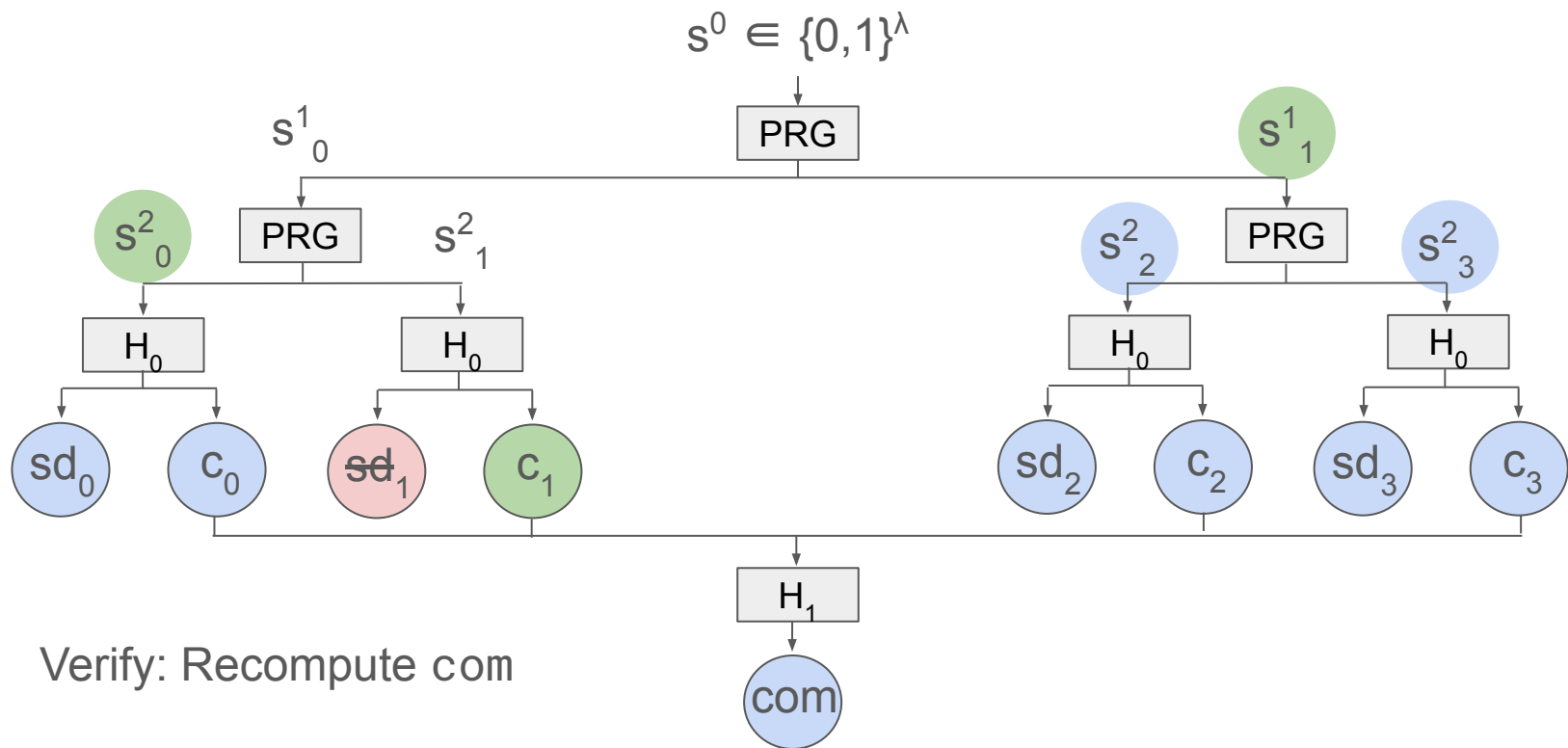
All-But-One Random Vector Commitments



All-But-One Random Vector Commitments



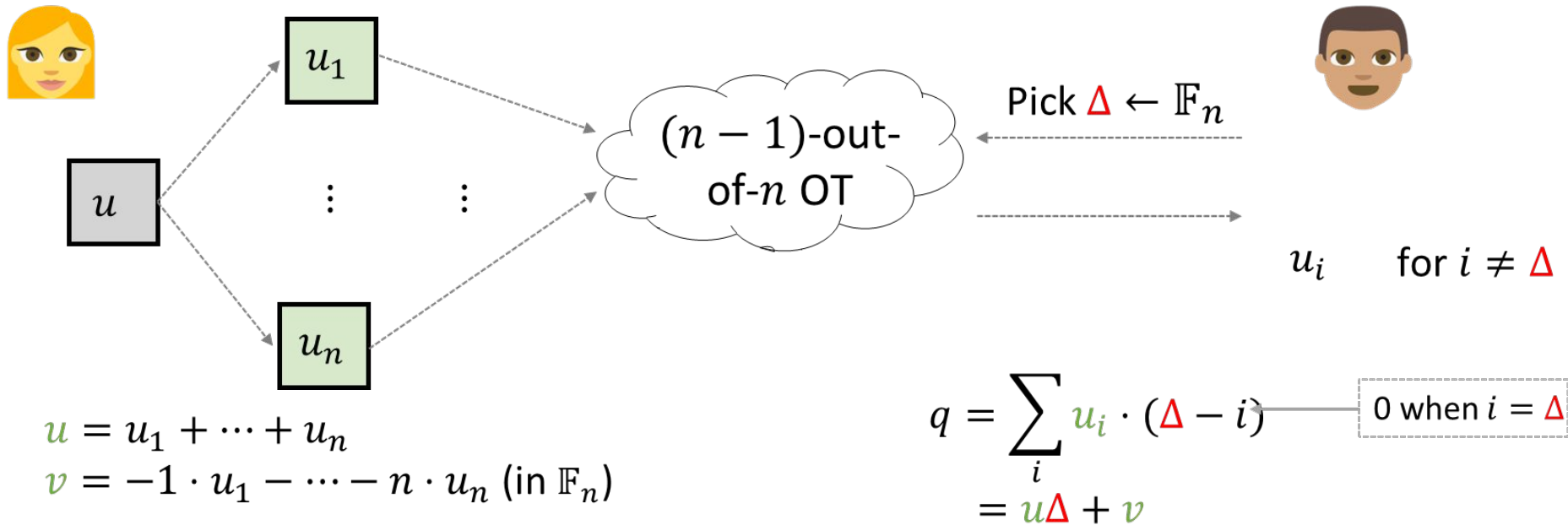
All-But-One Random Vector Commitments



VOLEs from OT

Key observation: n -out-of- n secret sharing + OT \Rightarrow VOLE in \mathbb{F}_n

(from [Roy 22])



Weak VOLEs based on Fiat-Shamir



All-but-one
vector commitment



$$\vec{u}, \vec{v}$$

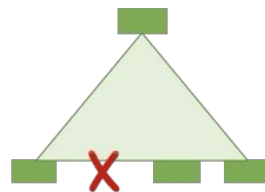
Commit to n random strings

⋮

Challenge Δ

Open $n - 1$

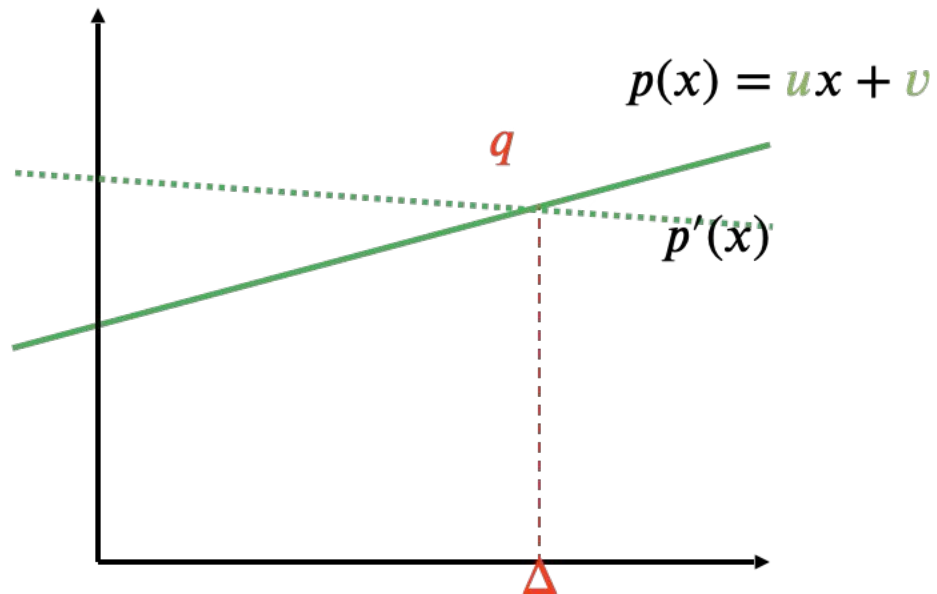
Convert to VOLE



$$\vec{q} = \vec{u} \Delta + \vec{v}$$

Gadget 2: VOLE in the Head

VOLE in the Head
=
Weak VOLEs
+
Quicksilver

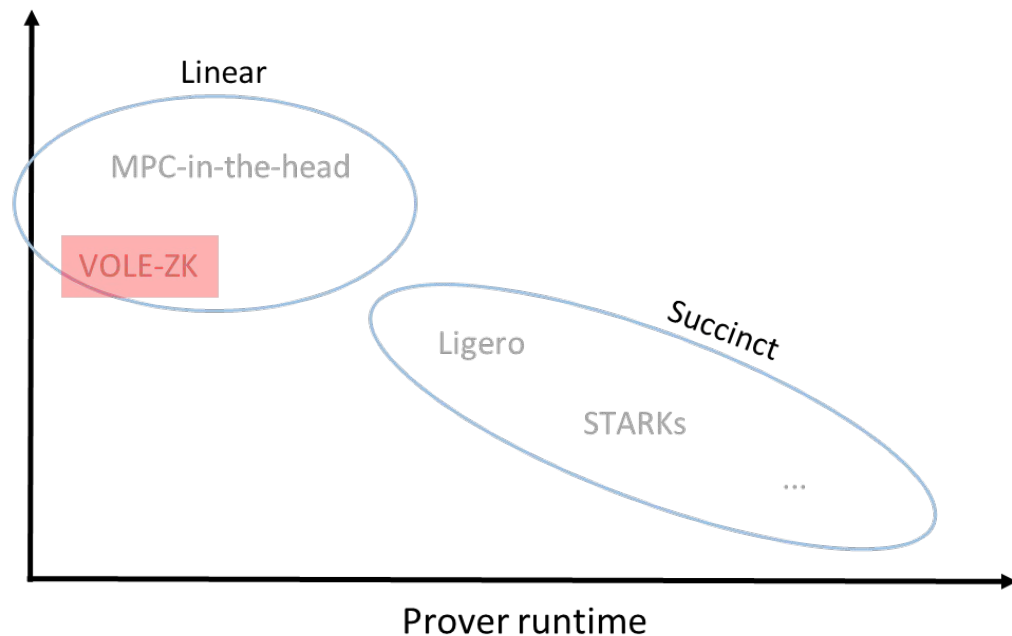


Efficient, non-interactive ZKPs

Gadget 2: VOLE in the Head

Security:
AES/SHA

Proof size: linear
Mod 2: 11-16x witness
Mod p: <2x witness



Schmivitz reference implementation

Open source Rust implementation

- Part of the Swanky secure computation library¹
- Crate dependencies include `aes`, `sha3`, `merlin`

Frontend statements:

- SIEVE IR
- Rust API (In progress)

1. <https://github.com/GaloisInc/swanky>

SIEVE IR

- Circuit format¹ defined during the DARPA SIEVE program
- Features
 - Field operations
 - Function gates
 - Multiple fields
 - Conversion gates
- Separates statement from ZK backend

```
version 2.0.0;
circuit;
@type field 2;
@begin
    $0 <- @private(0);
    $1 <- @mul(0: $0, $0);
    $2 <- @add(0: $1, $1);
    @assert_zero($2);
@end
```

1. <https://github.com/sieve-zk/ir>

Rust API (In progress)

```
pub trait Backend<F> {  
    type Wire;  
  
    fn input_public(&mut self) -> Result<Wire>;  
    fn input_private(&mut self) -> Result<Wire>;  
  
    fn add(&mut self, l: &Wire, r: &Wire)  
        -> Result<Wire>;  
    fn mul(&mut self, l: &Wire, r: &Wire)  
        -> Result<Wire>;  
    fn assert_zero(&mut self, a: &Wire)  
        -> Result<()>;  
    ...  
}
```

```
fn example<ZK: Backend<F2>>(b: &mut ZK)  
    -> CResult<()> {  
    let v0 = b.input_private()?;  
    let v1 = b.mul(&v0, &v0)?;  
    let v2 = b.add(&v1, &v1)?;  
  
    b.assert_zero(&v2)?;  
}
```

- Programmatic way to define circuits in source code
- Matches SIEVE IR functionality

Preliminary performance results

Table 1: Benchmarks on AES-256 and SHA-256.

	#AND	#ADD	#VOLEs	Prover time	Verifier time	Proof size (B)
AES-256	8832	39008	9216	10.9ms	8.7ms	21680
SHA-256	22573	110644	23341	17.7ms	15.7ms	49920

Circuits:

- Off-the-shelf Bristol Fashion Boolean circuits for AES-256 and SHA-256¹
- Not optimal

Machine: Macbook Pro M2 Max with 12 cores, 32GB RAM

1. <https://nigelsmart.github.io/MPC-Circuits/>

Ongoing work

- Optimize circuits
- Finish mod p implementation
- (Possible) additional gadgets: conversions, extension fields, higher-degree constraints
- Security analysis
 - Largely follows from prior work in FAEST + VOLEitH
 - Some parts still need modularizing