Non-Interactive Zero-Knowledge Proofs to Multiple Verifiers

Kang Yang

State Key Laboratory of Cryptology

Xiao Wang

Northwestern University

ASIACRYPT 2022

2022.12.07

Zero-Knowledge (ZK) Proofs

Prover

(x, w)

$(x, w) \in R$

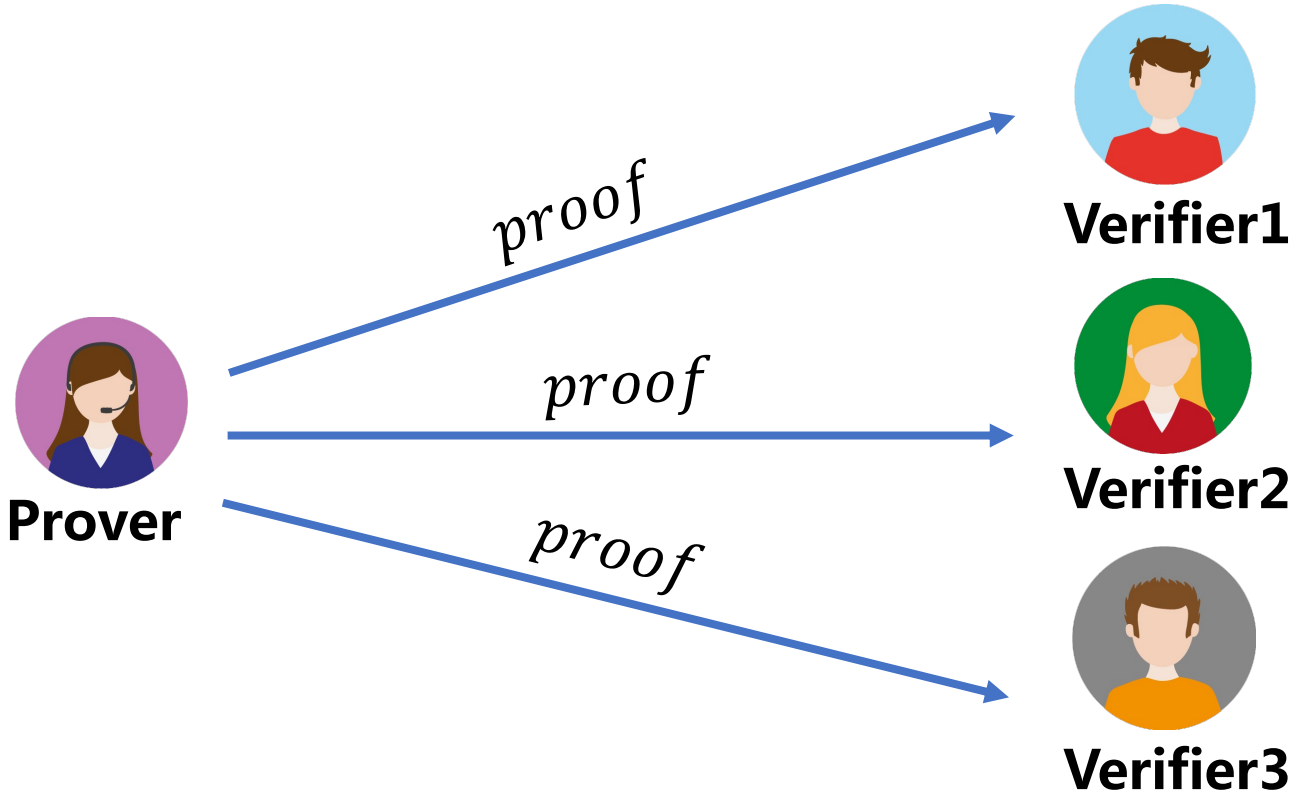
Verifier

x

- Statement x is often represented by a circuit C
- w is called witness
- $(x, w) \in R \Leftrightarrow C(w) = 0$

- Completeness
- Soundness
- Zero Knowledge

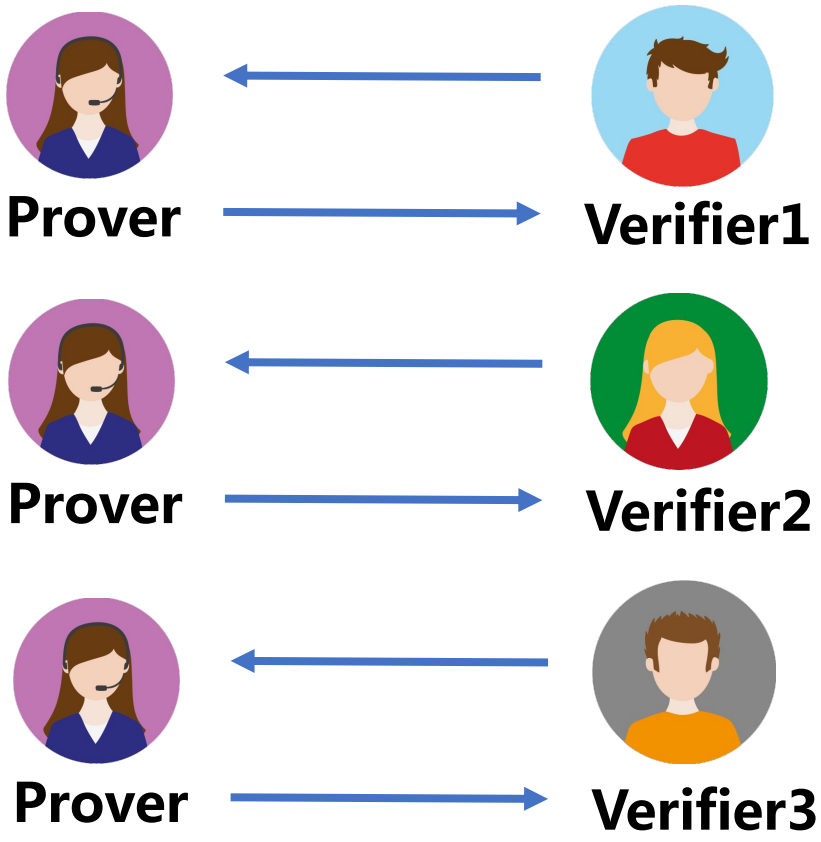
Non-Interactive ZK (NIZK)



- NIZK is **publicly verifiable**, and a *proof* can be **reused** to convince multiple verifiers
- NIZK efficiency has been significantly improved under different frameworks

  (e.g., MPC-in-the-head [IKOS07], GKR [GKR08], IOP [BCS16], …)

Designated-Verifier ZK (DVZK)



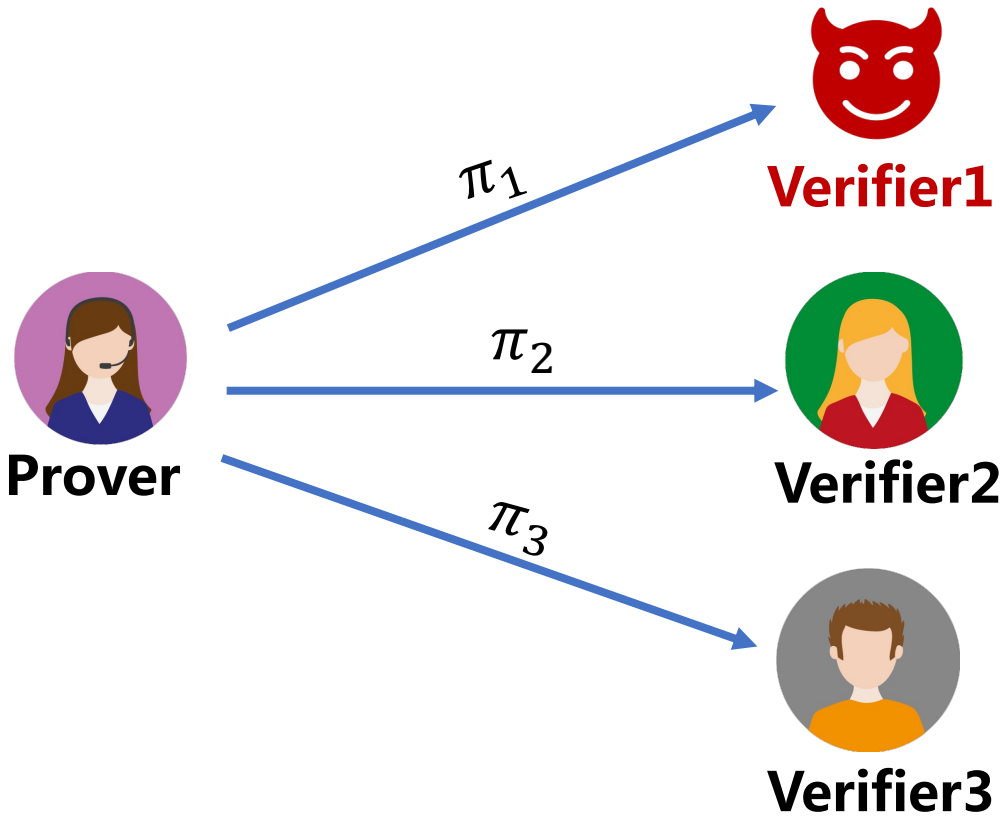
Different efficiency features, suitable for different applications

- DVZK : an **interactive** protocol between a prover and a verifier in general
- DVZK : convince **only one verifier** for every time
- Efficient frameworks : garbled circuit (GC) [JKO13], VOLE/COT [BMRS21, DIO21, WYKW21], …

	NIZK (zk-SNARKs)	DVZK (VOLE/COT)
Prover time	slower	**faster**
Verifier time	**sublinear**	linear
Total time	slower	**faster**
Proof size	**sublinear**	linear
Memory	Large	**small**
Rounds	**1**	constant
#Verifiers	**any**	one for every time

Multi-Verifier ZK (MVZK)

NIZK ⟷ MVZK ⟷ DVZK



- A prover wants to convince $\boldsymbol{n}$ verifiers
- $\boldsymbol{t}$ verifiers allow to be corrupted and **collude** with the prover
- We focus on the case of $\boldsymbol{t < n/2}$

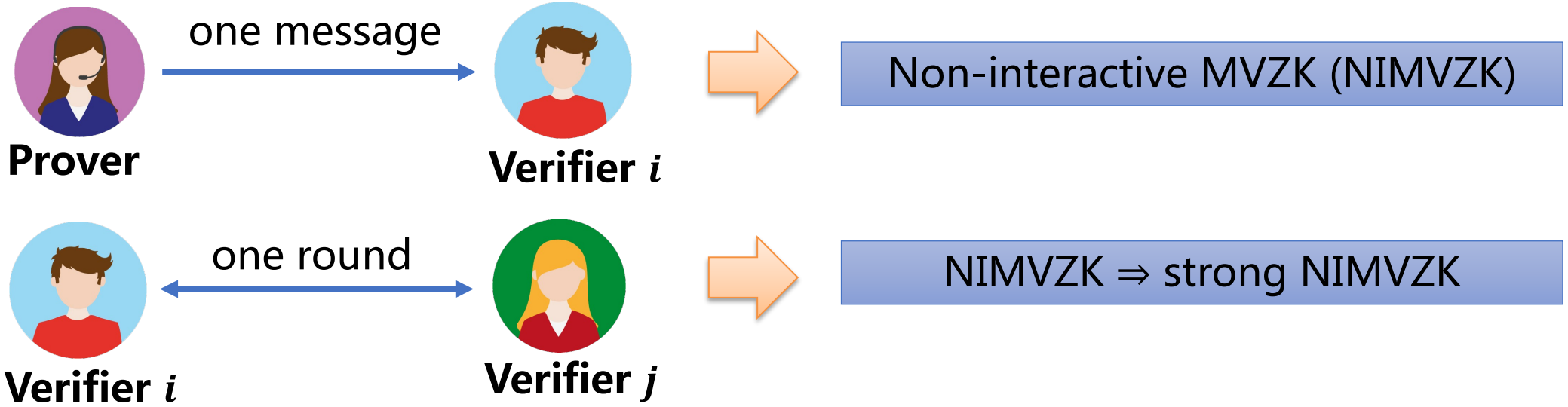
Our design goal

- Concretely **efficient** MVZK proofs
- **Less communication** than n DVZKs
- Rounds **same/close** to NIZK
- **Streamable** ⇒ small memory

Streamable : the prover can generate and send the proof on-the-fly (the circuit is proved batch-by-batch)

Non-Interactive MVZK

In the MVZK setting, there are two types of communications



- We allow the verifiers to communicate for one round
- Without any communication between verifiers, constructing NIMVZKs are as difficult as NIZKs

Applications for MVZK

- **Drop-in replacement to NIZK and DVZK:** MVZK can be used in normal ZK applications as long as the identifies of the verifiers are known ahead of time
- **Honest-majority MPC with input predicate check:** more efficient to check correctness of the input of every party
- **Private aggregation systems:**
  - Systems like Prio [CB17] use a set of servers to collect and aggregate data of users
  - To prevent mistakes/attacks, users prove to the servers that their data is valid in ZK
  - **MVZK can be used in the systems** when allowing users to **collude** with a minority of servers
- **……**

Our results

Three new concrete-efficient MVZK proofs with streamable property

Tools	Security	Threshold	Communication per gate per verifier	Rounds ($P \leftrightarrow V$)	Rounds ($V \leftrightarrow V$)	Stream
Shamir SS	IT	$t < n/2$	1	**1**	$\log \lvert C \rvert + 3$	**yes**
Shamir SS	CS	$t < n/2$	**0.5**	**1**	**1**	**yes**
Packed SS	CS	$t < n(1/2 - \epsilon)$	$\boldsymbol{O(1/n)}$	**1**	**1**	**yes**

IT: information-theoretic, CS: computational security

main protocols

NIMVZK

Strong NIMVZK

- All protocols have the computational complexity **linear** to circuit size, **cheaper** than DVZK
- Streaming proofs ⇒ **small memory**; strong NIMVZK protocols keep the **rounds** among verifiers **unchanged**
- **Asymmetric :** NIMVZK proofs in the CS setting allow that t verifiers have sublinear comm.

Compared to previous work for MVZK

Require public-key operations，not concretely efficient

- Strong NIMVZK protocol with honest-majority verifiers was first proposed in [ACF02]
- For $t < n/2$, the protocol in [GO07] can be transformed into strong NIMVZK

Implicit in distributed ZK, not generic circuits

- Distributed ZK on low-degree polynomials (instead of generic circuits) [BBCGI19]
- Distributed ZK implied in MPC [BGIN20] only proves degree-2 relations

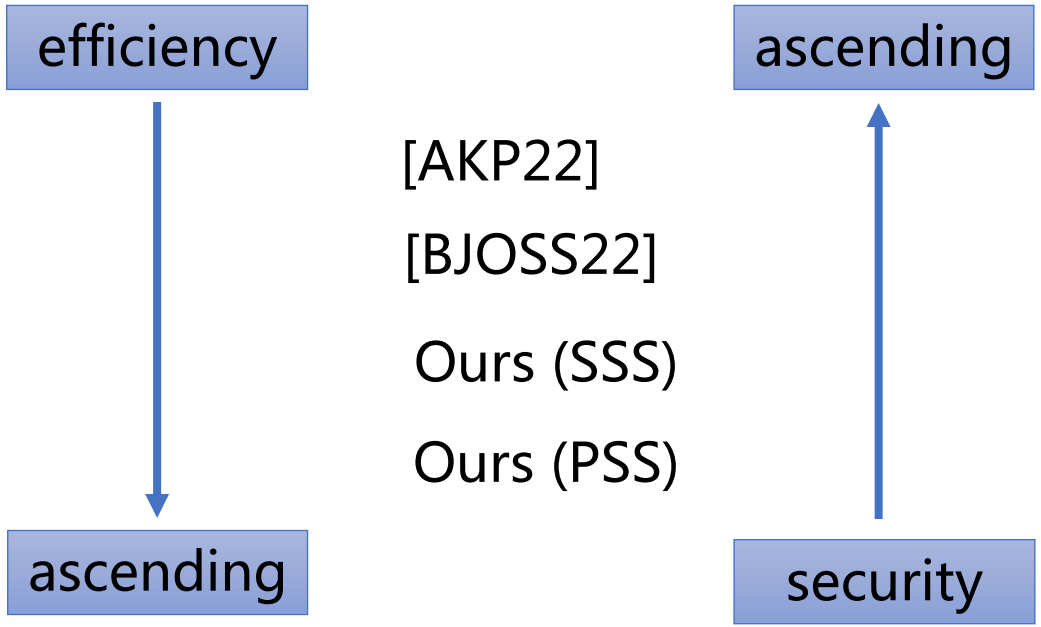
Prio, no collusion of the prover and verifiers

- Prio [CB17] does not allow the prover to collude with any verifier
- Prio requires quasi-linear (instead of linear) computation

Compared to concurrent work for MVZK

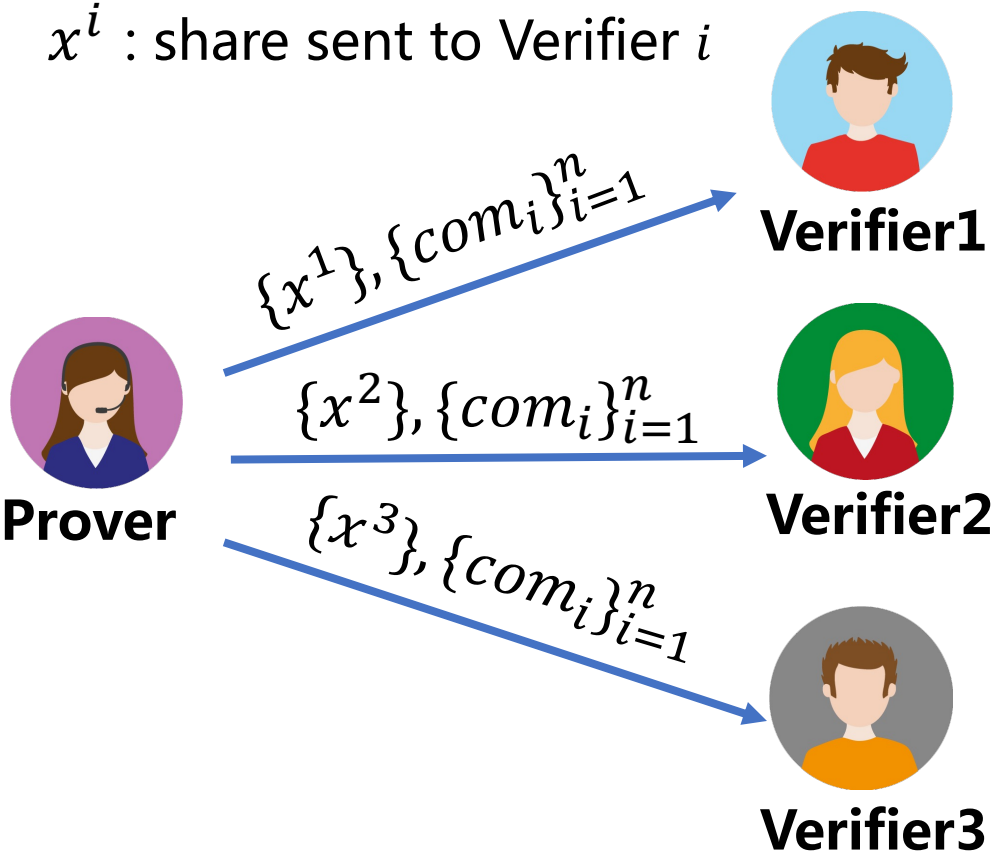
MVZK	Rounds	Security	Threshold	Assumption
[AKP22]	2	full security	$t < n(1/2 - \epsilon)$	NICOM
[BJOSS22]	2	identifiable abort	$t < n/3$ or $t < n/4$	RO
Ours (Shamir SS)	2	security with abort	$t < n/2$	RO
Ours (Packed SS)	2	security with abort	$t < n(1/2 - \epsilon')$	RO

NICOM can be based on injective one-way functions with sub-exponential hardness



Our framework for building NIMVZK protocols

- $[x]$: **Linear secret sharing** on secret x ;
- Shamir/packed secret sharings
- Replicated secret sharings also work
- x^i : share sent to Verifier i



Share the secret values for each input wire and output wire of MULT gates

Using **Distributed ZK (DZK)** technique to check correctness of sharings of MULT gates

Distributing Fiat-Shamir ⇒ strongly non-interactive

- Prover broadcasts $\{com_i\}_{i=1}^n$ to every verifier, com_i is a **commitment of shares** held by Verifier i
- Honest-majority verifiers ⇒ $\{com_i\}_{i=1}^n$ determine all secret wire values

Warm-Up : Information-Theoretic NIMVZK

Circuit evaluation

- For output x of each gate, the prover shares x into $[x]$
- Addition gates are free due to of linear property of secret sharings

Transformation

Check correctness of multiplication gates ⇒ Check inner-product tuple

$$\{([x_i],[y_i],[z_i])\}_{i\in[1,N]} \Rightarrow z_i = x_i \cdot y_i \text{ for } i \in [1,N]$$

coin-tossing to generate random challenge χ

Verification

- Using DZK [BBCGI19,GSZ20] to check correctness of $([\boldsymbol{x}],[\boldsymbol{y}],[z])$
- **The prover generates random sharings** to mask sharings of secrets

$$([\boldsymbol{x}],[\boldsymbol{y}],[z]) \Rightarrow z = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$$

$$[\boldsymbol{x}] = ([x_1], \chi \cdot [x_2], \cdots, \chi^{N-1} \cdot [x_N])$$

$$[\boldsymbol{y}] = ([y_1],[y_2],\cdots,[y_N])$$

$$[z] = \sum_{i\in[1,N]} \chi^i \cdot [z_i]$$

Strong NIMVZK from Shamir Sharings

- The framework is the same as information-theoretic NIMVZK
- The challenge is to compute a **public message** used as the input of RO
  - Only **secret** shares are sent by the prover to every verifier
  - The verifiers have **no way** to compute a public message

$n^2\lambda$ bits extra communication

① Prover sends (M_i, r_i) to Verifier i where M_i consists of secret shares

② Prover computes $com_i = H(M_i, r_i)$ where H is a RO

③ Prover sends $(com_1, \cdots, com_n)$ to every verifier

④ Verifier i checks $com_i = H(M_i, r_i)$; $t < n/2 \Rightarrow$ a majority of commitments are correct

⑤ Challenge $\chi = H'(com_1, \cdots, com_n)$

H, H'

- **It is unnecessary to be collision-resistant**
- **Random output is sufficient**
- **Adopt λ (rather than 2λ) as the output length**

Construction [DNNR17] based on fixed-key AES

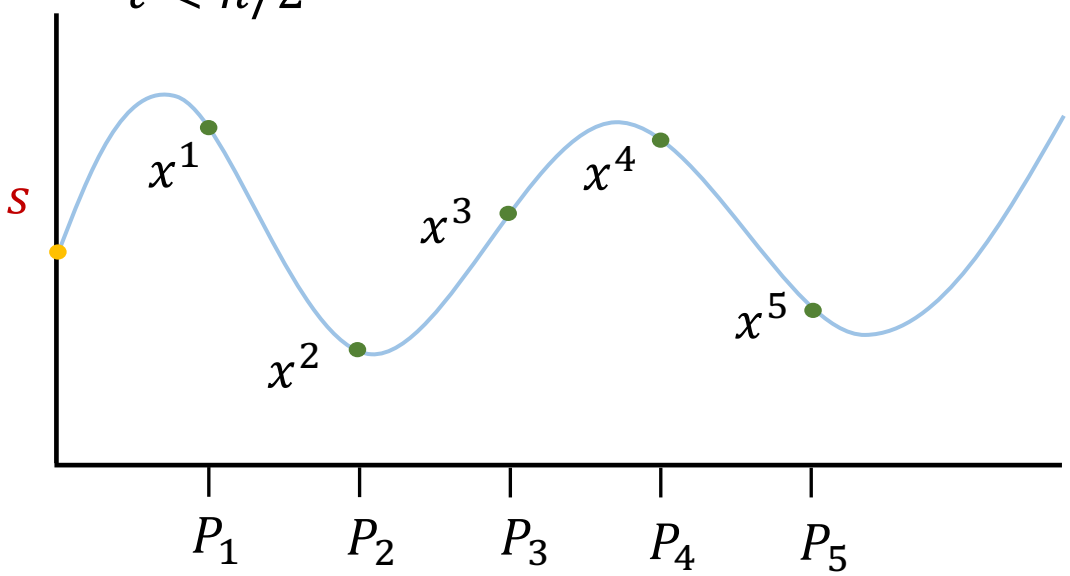
Strong NIMVZK from Packed Sharings

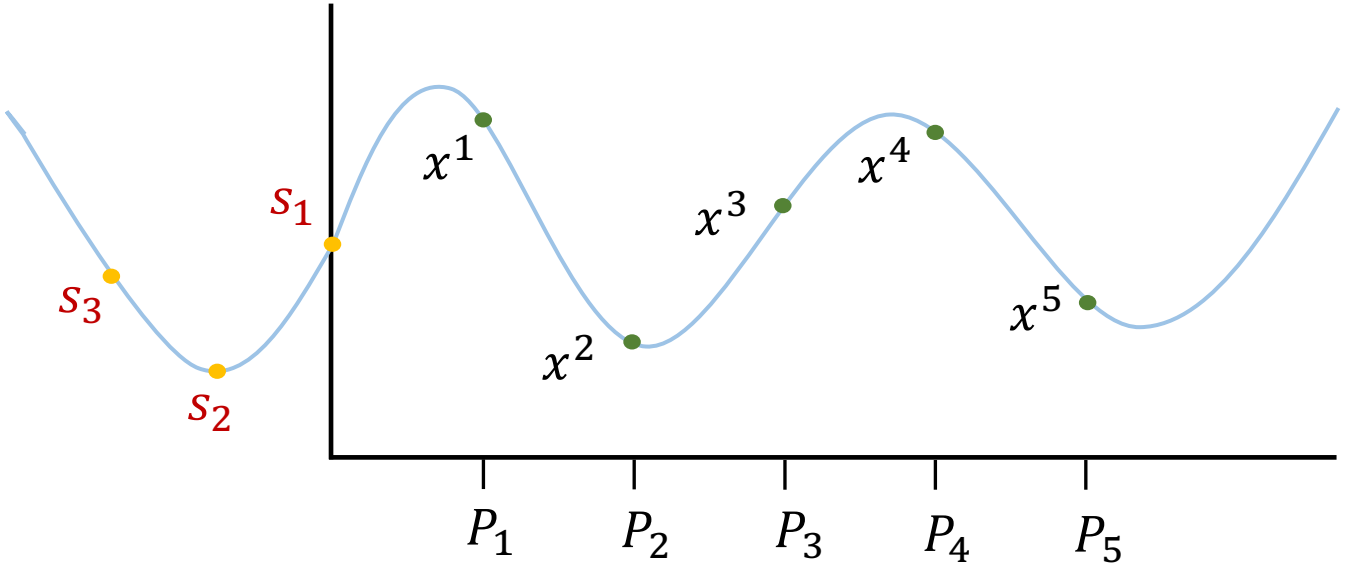
- Shamir sharings : $f(0) = s$
- $f(\alpha_i) = x^i$
- $t < n/2$

generalize

- packed sharings : $f(\beta_i) = s_i$ for $i \in [1, k]$
- $t \leq n/2 - k$

denoted by $[\boldsymbol{x}]$ for $\boldsymbol{x} \in \mathbb{F}^{\boldsymbol{k}}$





- Using packed sharings, perform addition/multiplication for a group of k gates every time
- Prover generates and distributes $[\boldsymbol{y}]$ for every group, if the packed sharing does not exist
- There exists some wires that are in different groups

Consistency Check for Wire Tuples

- Wire tuple [GPS21] : $([\boldsymbol{x}], [\boldsymbol{y}], i, j)$ with $x_i = y_j$
- For each $i, j \in [1, k]$, check $([\boldsymbol{x}_1], [\boldsymbol{y}_1], i, j), \cdots, ([\boldsymbol{x}_m], [\boldsymbol{y}_m], i, j)$
- $[\boldsymbol{x}] = \Sigma_{h\in[1,m]} \alpha^h \cdot [\boldsymbol{x}_h] + [\boldsymbol{x}_0]$, $[\boldsymbol{y}] = \Sigma_{h\in[1,m]} \alpha^h \cdot [\boldsymbol{y}_h] + [\boldsymbol{y}_0]$
- Open($[\boldsymbol{x}]$, $[\boldsymbol{y}]$) to check $x_i = y_j$
- $[\boldsymbol{x}_0]$ and $[\boldsymbol{y}_0]$ are random sharings generated by Prover such that $x_{0,i} = y_{0,j}$
- Generate α using Fiat-Shamir : $\alpha = H'(\chi, \boldsymbol{u}, \boldsymbol{v}, i, j)$, where $\boldsymbol{u}, \boldsymbol{v}$ are public messages to generate $[\boldsymbol{x}_0]$ and $[\boldsymbol{y}_0]$
  - Prover sends random sharings $[\boldsymbol{r}]$ and $[\boldsymbol{s}]$ to verifiers
  - Prover broadcasts $\boldsymbol{u} = \boldsymbol{x}_0 + \boldsymbol{r}$ and $\boldsymbol{v} = \boldsymbol{y}_0 + \boldsymbol{s}$ to verifiers
  - Verifiers compute $[\boldsymbol{x}_0] = \boldsymbol{u} - [\boldsymbol{r}]$ and $[\boldsymbol{y}_0] = \boldsymbol{v} - [\boldsymbol{s}]$

- Using echo-broadcast protocol [GL05]
- The hashing of messages is sent when opening sharings ⇒ **keep one round**

Verification for Packed Inner-Product Tuples

Transform packed multiplication tuples into a packed inner-product tuple

$([\boldsymbol{x}_1], \cdots, [\boldsymbol{x}_N]), ([\boldsymbol{y}_1], \cdots, [\boldsymbol{y}_N]), [\boldsymbol{z}]$ with $\boldsymbol{z} = \Sigma_{i \in [1,N]} \boldsymbol{x}_i * \boldsymbol{y}_i$ and $*$ is component-wise product

Recursive reduction of dimension of the packed inner-product tuple

$\log N - 1$ iterations without interaction

For each iteration, split a packed tuple into two packed tuples, then compress two tuples with dimension m into one tuple with dimension $m/2$

Randomize the compressed inner-product tuple and then open

only one round

- Split $([\boldsymbol{x}_1], [\boldsymbol{x}_2]), ([\boldsymbol{y}_1], [\boldsymbol{y}_2]), [\boldsymbol{z}]$ into $([\boldsymbol{x}_1], [\boldsymbol{y}_1], [\boldsymbol{z}_1]), ([\boldsymbol{x}_2], [\boldsymbol{y}_2], [\boldsymbol{z}_2]), [\boldsymbol{z}] = [\boldsymbol{z}_1] + [\boldsymbol{z}_2]$
- Prover generates random tuple $([\boldsymbol{x}_0], [\boldsymbol{y}_0], [\boldsymbol{z}_0])$ with $\boldsymbol{z}_0 = \boldsymbol{x}_0 * \boldsymbol{y}_0$
- Compress $\{([\boldsymbol{x}_i], [\boldsymbol{y}_i], [\boldsymbol{z}_i])\}_{i \in [0,2]}$ into $([\boldsymbol{x}], [\boldsymbol{y}], [\boldsymbol{z}])$ $\Rightarrow$ open it to check $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$

Non-interactive compression of inner-product tuples

Based on the polynomial approach [BBCGI19,GPS21], we use Fiat-Shamir to realize **non-interactive** compression

Compress two packed inner-product tuples $([\boldsymbol{a}_{1,1}],\cdots,[\boldsymbol{a}_{1,m}]),([\boldsymbol{b}_{1,1}],\cdots,[\boldsymbol{b}_{1,m}]),[\boldsymbol{c}_1]$ and $([\boldsymbol{a}_{2,1}],\cdots,[\boldsymbol{a}_{2,m}]),([\boldsymbol{b}_{2,1}],\cdots,[\boldsymbol{b}_{2,m}]),[\boldsymbol{c}_2]$ into one tuple $([\boldsymbol{x}_1],\cdots,[\boldsymbol{x}_m]),([\boldsymbol{y}_1],\cdots,[\boldsymbol{y}_m]),[\boldsymbol{z}]$

- For $j \in [1,m]$, compute $[\boldsymbol{f}_j(\cdot)]$ such that $\boldsymbol{f}_j(i) = \boldsymbol{a}_{i,j}$ for $i \in [1,2]$
- For $j \in [1,m]$, compute $[\boldsymbol{g}_j(\cdot)]$ such that $\boldsymbol{g}_j(i) = \boldsymbol{b}_{i,j}$ for $i \in [1,2]$
- Compute $[h(\cdot)]$ such that $\boldsymbol{h}(i) = \boldsymbol{c}_i$ for $i \in [1,2]$
- Compute $\alpha = H'(\gamma, msg)$ where γ is the challenge used in the previous iteration and msg is the public message sent in the current iteration
- For $j \in [1,m]$, $[\boldsymbol{x}_j] = [\boldsymbol{f}_j(\alpha)]$, $[\boldsymbol{y}_j] = [\boldsymbol{g}_j(\alpha)]$, $[\boldsymbol{z}] = [\boldsymbol{h}(\alpha)]$

Stream proof without increasing rounds between verifiers

- Prove a large circuit **batch-by-batch**
  - Prover can prove $N = k \cdot M$ multiplication gates each time
- For one batch, **non-interactively** compress a packed inner-product tuple with dimension N into a packed tuple $IPtuple_1$ with dimension $N/2^c$ for some integer c
- For another batch, perform the same operations to obtain another packed tuple $IPtuple_2$ with dimension $N/2^c$
- **Non-interactively** compress $IPtuple_1$ and $IPtuple_2$ into one tuple with dimension $N/2^c$

- Non-interactively compression ⇒ **only one round** between verifiers for all batches
- Memory $O(N)$ where N is a parameter set according to the memory size
- Similar idea can used for consistency check of wire tuples

Future Works

- Recently, AntMan [WYYXW22] first achieved sublinear communication for two-party ZK proofs in the VOLE family
  - One open question is to construct an NIMVZK protocol with sublinear communication without requiring any computation-heavy tool such as FHE, if it is not impossible
- Our NIMVZK protocol works in the honest-majority setting
  - Another open question is to design an MVZK protocol for $t = n - 1$ that has significantly less communication than running n DVZKs
- A future work is to implement the NIMVZK protocol and apply it to construct a private aggregation system

Thanks for your attention!

Full version can be found at https://eprint.iacr.org/2022/063

If you have any questions, send emails to yangk@sklc.org