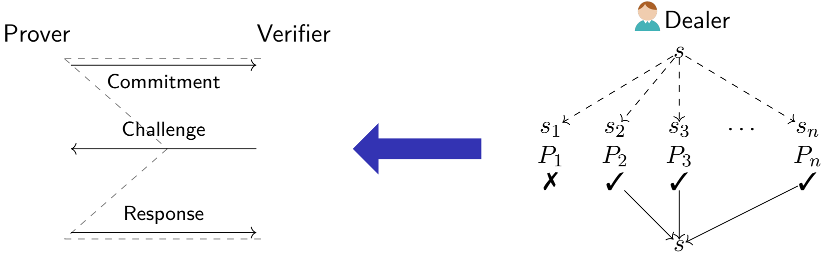
Sigma Protocols from Verifiable Secret Sharing and Their Applications



Min Zhang
Shandong University

joint work[1] with Yu Chen, Chuanzhou Yao and Zhichao Wang

[1] ASIACRYPT 2023: Sigma Protocols from Verifiable Secret Sharing and Their Applications. Min Zhang, Yu Chen, Chuanzhou Yao, Zhichao Wang.

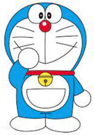
Outline

Outline

Sigma (Σ) Protocols (PhD Thesis 1996: Cramer)

$(x, w) \in \mathsf{R}$

Prover



message a →

← challenge e

response z →

Verifier

$\mathsf{Verify}(x, a, e, z) = 0/1$

- **Completeness:** $\Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x) \rangle = 1 | \ (x, w) \in \mathsf{R}] = 1$
- n-**Special soundness:** $\exists$ PPT Ext that given any x and any n accepting transcripts (a, e_i, z_i) with distinct e_i's can extract w s.t. $(x, w) \in \mathsf{R}$
- **Special honest verifier zero-knowledge (SHVZK):** $\exists$ PPT Sim s.t. for any x and e, $\mathsf{Sim}(x, e) \equiv \langle \mathcal{P}(x, w), \mathcal{V}(x, e) \rangle$

Attractive Properties of Sigma Protocols

- Efficient for algebraic statements
  - Schnorr protocol [Sch91]: $x = g^w$
  - Okamoto protocol [Oka92]: $x = g^w h^r$
  - Guillou-Quisquater (GQ) protocol [GQ88]: $x = w^e \mod N$
- Can be easily combined to prove compound statements, such as AND/OR
- Provide a simple way to establish proof-of-knowledge property
- Fiat-Shamir heuristic [FS86] helps to remove ~~interaction~~: SHVZK $\rightsquigarrow$ Full ZK
- Enable numerous real-world applications

Identification protocols	(Ring) Signature schemes
Anonymous credentials	Privacy-preserving cryptocurrency

Research on Sigma Protocols

Classic Σ protocols

- Schnorr [Sch91]
- Okamoto [Oka92]
- GQ [GQ88]

Improve efficiency

- Batch-Schnorr [GLSY04]

Enrich functionality

- Commitments to bits [Bou00, GK15, BCC+15]
- k-out-of-n proofs [CDS94, GK15, AAB+21]
- Lattice-based problems [YAZ+19, BLS19, LNP22]

Research on Sigma Protocols

Classic Σ protocols

- Schnorr [Sch91]
- Okamoto [Oka92]
- GQ [GQ88]

ingenious

but hand-crafted

Improve efficiency

- Batch-Schnorr [GLSY04]

Enrich functionality

- Commitments to bits [Bou00, GK15, BCC+15]
- k-out-of-n proofs [CDS94, GK15, AAB+21]
- Lattice-based problems [YAZ+19, BLS19, LNP22]

Research on Sigma Protocols

Classic Σ protocols

- Schnorr [Sch91]
- Okamoto [Oka92]
- GQ [GQ88]

ingenious

but hand-crafted

Improve efficiency

- Batch-Schnorr [GLSY04]

Enrich functionality

- Commitments to bits [Bou00, GK15, BCC+15]
- k-out-of-n proofs [CDS94, GK15, AAB+21]
- Lattice-based problems [YAZ+19, BLS19, LNP22]

Whether there exists a common design principal of Sigma protocols?

Related Works

[Mau15] U. Maurer. Zero-knowledge proofs of knowledge for group homomorphisms.

$$x = f(w)$$

$(\mathbb{H}_1, +)$, $(\mathbb{H}_2, \cdot)$, homomorphism $f : \mathbb{H}_1 \to \mathbb{H}_2$, $f(w_1 + w_2) = f(w_1) \cdot f(w_2)$

Prover		Verifier
$t \xleftarrow{R} \mathbb{H}_1$ $a = f(t)$	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{R} C \subset \mathbb{Z}$
$z = t + e \times w$	$\xrightarrow{\quad z \quad}$	$f(z) ?= a \cdot x^e$

It unifies a substantial body of works, including classic Schnorr [Sch91], GQ [GQ88] and Okamoto [Oka92] protocols. ☺

Related Works

[Mau15] U. Maurer. Zero-knowledge proofs of knowledge for group homomorphisms.

Framework in [Mau15]

$$x = f(w)$$

Prover		Verifier
$t \xleftarrow{R} \mathbb{H}_1$ $a = f(t)$	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{R} C$
$z = t + e \times w$	$\xrightarrow{\quad z \quad}$	$f(z) ?= a \cdot x^e$

The pattern is fixed $\rightsquigarrow$ fail to explain some simple variants of classic protocols ☹

Related Works

[Mau15] U. Maurer. Zero-knowledge proofs of knowledge for group homomorphisms.

Framework in [Mau15]

$$x = f(w)$$

Prover		Verifier
$t \xleftarrow{R} \mathbb{H}_1$, $a = f(t)$	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{R} C$
$z = t + e \times w$	$\xrightarrow{\quad z \quad}$	$f(z) ?= a \cdot x^e$

A variant of [Sch91]

$$x = g^w$$

$$f(w) = g^w$$

Prover		Verifier
$t \xleftarrow{R} \mathbb{Z}_p$, $a = g^{-t}$ ($a = g^t$)	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{R} \mathbb{Z}_p$
$z = -t \times e + w$ ($z = t + w \times e$)	$\xrightarrow{\quad z \quad}$	$g^z ?= a^e \cdot x$

The pattern is fixed $\rightsquigarrow$ fail to explain some simple variants of classic protocols ☹

Related Works

[Mau15] U. Maurer. Zero-knowledge proofs of knowledge for group homomorphisms.

Framework in [Mau15]

$$x = f(w)$$

Prover		Verifier
$t \xleftarrow{R} \mathbb{H}_1$ $a = f(t)$	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{R} C$
$z = t + e \times w$	$\xrightarrow{\quad z \quad}$	$f(z) \stackrel{?}{=} a \cdot x^e$

A variant of [Sch91]

$$x = g^w$$

$$f(w) = g^w$$

Prover		Verifier
$t \xleftarrow{R} \mathbb{Z}_p$ $a = g^{-t}$ $\boxed{a = g^t}$	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{R} \mathbb{Z}_p$
$z = -t \times e + w$ $\boxed{z = t + w \times e}$	$\xrightarrow{\quad z \quad}$	$g^z \stackrel{?}{=} a^e \cdot x$

The pattern is fixed $\rightsquigarrow$ fail to explain some simple variants of classic protocols ☹
$\rightsquigarrow$ the machinery of Sigma protocols is still unclear.

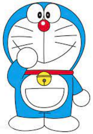
Motivation

Σ

Is there a more generic framework of Sigma protocols?

The Schnorr Protocol (JoC 1991: Schnorr)

$$x = g^w$$

Prover		Verifier
		
$t \xleftarrow{R} \mathbb{Z}_p$, $a = g^t$	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{R} \mathbb{Z}_p$
$z = t + e \times w$	$\xrightarrow{\quad z \quad}$	$g^z ?= a \cdot x^e$

- **Completeness:** $g^z = g^{t+e\times w} = g^t \cdot g^{w\times e} = a \cdot x^e$
- 2-**Special soundness:** $\mathsf{Ext}(x, (a, e_1, z_1), (a, e_2, z_2)) \rightarrow w = (z_1 - z_2)/(e_1 - e_2)$
- **SHVZK:** $\mathsf{Sim}(x, e) \rightarrow (a, e, z)$: pick $z \xleftarrow{R} \mathbb{Z}_p$ and set $a = g^z \cdot x^{-e}$

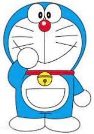
Outline

MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

$C(w) = y$ $\quad$ C: arithmetic or boolean circuit

MPC-in-the-Head

Prover



Verifier

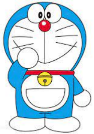
MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

$C(w) = y$ — C: arithmetic or boolean circuit

MPC-in-the-Head

1. Share w : $w = w_1 \oplus \cdots \oplus w_n$

Prover



Verifier

MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

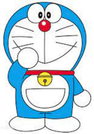
$C(w) = y$

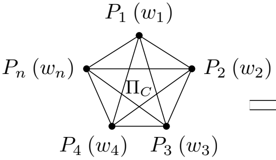
C: arithmetic or boolean circuit

MPC-in-the-Head

1. Share w : $w = w_1 \oplus \cdots \oplus w_n$
2. Run MPC protocol Π_C :

Prover





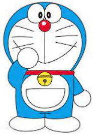
$\Longrightarrow P_i : w_i || view_i$

Verifier

MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

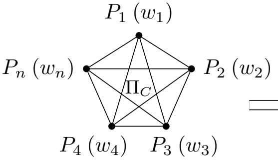
$C(w) = y$ — C: arithmetic or boolean circuit

Prover



MPC-in-the-Head

1. Share w : $w = w_1 \oplus \cdots \oplus w_n$
2. Run MPC protocol Π_C :



$\Longrightarrow P_i : w_i || view_i$

3. Commit to the views :

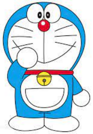
Verifier

MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

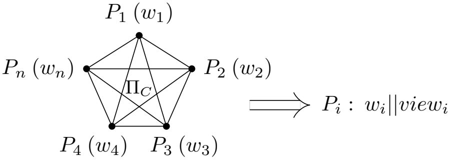
$C(w) = y$ C: arithmetic or boolean circuit

MPC-in-the-Head

Prover



1. Share w : $w = w_1 \oplus \cdots \oplus w_n$
2. Run MPC protocol Π_C :



3. Commit to the views :

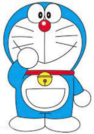
$c_1, \ldots, c_n$ $\longrightarrow$ Verifier

MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

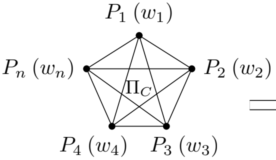
$C(w) = y$ — C: arithmetic or boolean circuit

MPC-in-the-Head

Prover



1. Share w : $w = w_1 \oplus \cdots \oplus w_n$
2. Run MPC protocol Π_C :



$\Longrightarrow$ P_i : $w_i || view_i$

3. Commit to the views :

Prover → Verifier: $c_1, \ldots, c_n$

Verifier → Prover: $I \subset [n]$

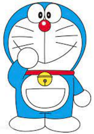
Verifier

MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

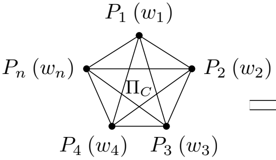
$C(w) = y$ — C: arithmetic or boolean circuit

MPC-in-the-Head

Prover



1. Share w : $w = w_1 \oplus \cdots \oplus w_n$
2. Run MPC protocol Π_C :



$\Longrightarrow P_i :\ w_i || view_i$

3. Commit to the views :

$w_1||view_1$: c_1 — $w_2||view_2$: c_2 — $\ldots$ — $w_n||view_n$: c_n

Verifier

Prover → Verifier: $c_1, \ldots, c_n$

Verifier → Prover: $I \subset [n]$

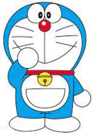
Prover → Verifier: $(w_i||view_i)_{i \in I}$

MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

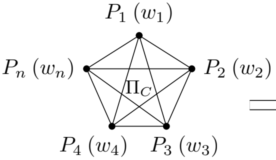
$C(w) = y$ C: arithmetic or boolean circuit

MPC-in-the-Head

Prover



1. Share w : $w = w_1 \oplus \cdots \oplus w_n$
2. Run MPC protocol Π_C :



$\Longrightarrow P_i : w_i || view_i$

3. Commit to the views :

$c_1, \ldots, c_n \longrightarrow$

$I \subset [n] \longleftarrow$

$(w_i||view_i)_{i \in I} \longrightarrow$

Verifier

✓Accept iff:

$(c_i)_{i \in I}$ opened successfully

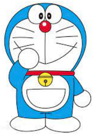
output=1 & consistent

MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

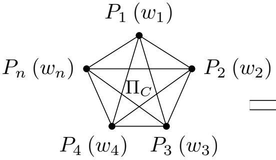
$C(w) = y$ C: arithmetic or boolean circuit

MPC-in-the-Head

Prover



1. Share w : $w = w_1 \oplus \cdots \oplus w_n$
2. Run MPC protocol Π_C :



$\Longrightarrow$ P_i : $w_i || view_i$

3. Commit to the views :

$c_1, \ldots, c_n$ $\longrightarrow$

$I \subset [n]$ $\longleftarrow$

$(w_i||view_i)_{i \in I}$ $\longrightarrow$

Verifier

✓Accept iff:
$(c_i)_{i \in I}$ opened successfully
output=1 & consistent

Fact: MPC-in-the-head is a Σ-pattern protocol for arithmetic statements!
Thinking: algebraic statements are arguably simpler than arithmetic statements. When scaling down to algebraic statements, we may start from a lite machinery than MPC.

VSS: A Lite Machinery than MPC

A lite machinery than MPC: Verifiable Secret Sharing (VSS) [CGMA85]

Non-interactive VSS [Fel87]

Sharing Phase

- Setup$(1^\lambda) \to pp$
  - number of parties n
  - privacy threshold t_p
  - fault-tolerance threshold t_f
- Encrypt$(w) \to c$
- Share$(w) \to (w_1, \ldots, w_n)$

Reconstruction Phase

- Check$(i, w_i, c) \to 0/1$
- Recover$(I, (w_i)_{i \in I}) \to w$

Dealer

w

VSS: A Lite Machinery than MPC

A lite machinery than MPC: Verifiable Secret Sharing (VSS) [CGMA85]

Non-interactive VSS [Fel87]

Sharing Phase
- Setup$(1^\lambda) \to pp$
  - number of parties n
  - privacy threshold t_p
  - fault-tolerance threshold t_f
- Encrypt$(w) \to c$
- Share$(w) \to (w_1, \ldots, w_n)$

Reconstruction Phase
- Check$(i, w_i, c) \to 0/1$
- Recover$(I, (w_i)_{i \in I}) \to w$

Dealer

$c \leftarrow$ Encrypt — w

VSS: A Lite Machinery than MPC

A lite machinery than MPC: Verifiable Secret Sharing (VSS) [CGMA85]

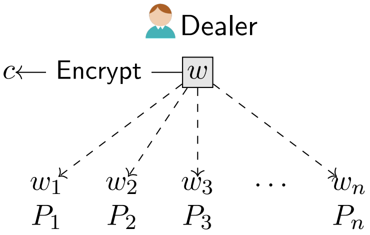
Non-interactive VSS [Fel87]

Sharing Phase

- Setup$(1^\lambda) \to pp$
  - number of parties n
  - privacy threshold t_p
  - fault-tolerance threshold t_f
- Encrypt$(w) \to c$
- Share$(w) \to (w_1, \ldots, w_n)$

Reconstruction Phase

- Check$(i, w_i, c) \to 0/1$
- Recover$(I, (w_i)_{i \in I}) \to w$



VSS: A Lite Machinery than MPC

A lite machinery than MPC: Verifiable Secret Sharing (VSS) [CGMA85]

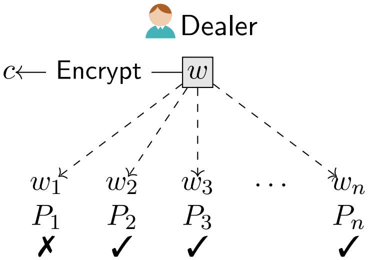
Non-interactive VSS [Fel87]

Sharing Phase

- Setup$(1^\lambda) \to pp$
  - number of parties n
  - privacy threshold t_p
  - fault-tolerance threshold t_f
- Encrypt$(w) \to c$
- Share$(w) \to (w_1, \ldots, w_n)$

Reconstruction Phase

- Check$(i, w_i, c) \to 0/1$
- Recover$(I, (w_i)_{i \in I}) \to w$



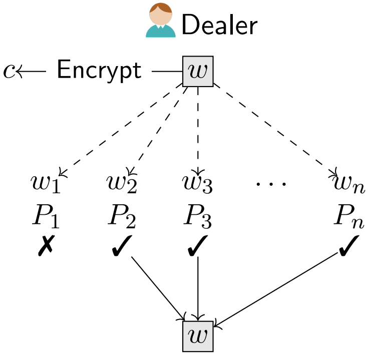
VSS: A Lite Machinery than MPC

A lite machinery than MPC: Verifiable Secret Sharing (VSS) [CGMA85]

Non-interactive VSS [Fel87]

Sharing Phase
- Setup$(1^\lambda) \to pp$
  - number of parties n
  - privacy threshold t_p
  - fault-tolerance threshold t_f
- Encrypt$(w) \to c$
- Share$(w) \to (w_1, \ldots, w_n)$

Reconstruction Phase
- Check$(i, w_i, c) \to 0/1$
- Recover$(I, (w_i)_{i \in I}) \to w$



VSS: A Lite Machinery than MPC

A lite machinery than MPC: Verifiable Secret Sharing (VSS) [CGMA85]

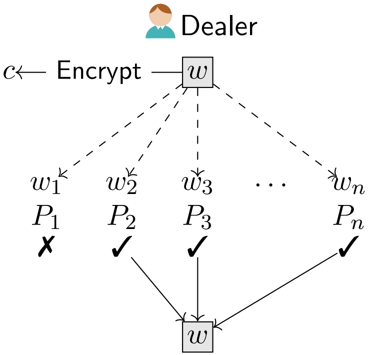
Non-interactive VSS [Fel87]

Sharing Phase

- $\mathsf{Setup}(1^\lambda) \rightarrow pp$
  - number of parties n
  - privacy threshold t_p
  - fault-tolerance threshold t_f
- $\mathsf{Encrypt}(w) \rightarrow c$
- $\mathsf{Share}(w) \rightarrow (w_1, \ldots, w_n)$

Reconstruction Phase

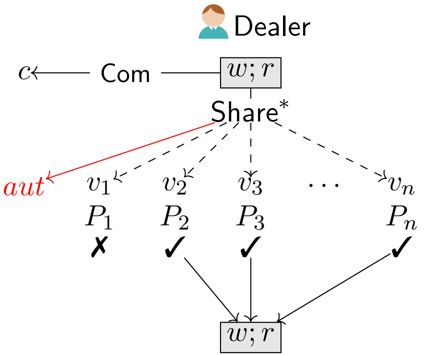
- $\mathsf{Check}(i, w_i, c) \rightarrow 0/1$
- $\mathsf{Recover}(I, (w_i)_{i \in I}) \rightarrow w$



- **Acceptance:** valid shares $w_i \Rightarrow \mathsf{Check}(i, w_i, c) = 1$
- t_p-**Privacy:** # [shares] $\leq t_p \Rightarrow$ leak nothing about w
- **Consistency:** # [valid shares] $\geq t_f \Rightarrow$ unique w and recover w

A Refined Definition of VSS

- Setup$(1^\lambda) \to pp$
  include n, t_p, t_f
- Share$(w) \to (c, (v_i)_{i\in[n]}, aut)$
  - Com$(w; r) \to c$
    r: could be empty
  - Share$^*(w, r) \to ((v_i)_{i\in[n]}, aut)$
    aut: authentication information
    (a commitment to the sharing procedure)
- Check$(i, v_i, c, aut) \to 0/1$
- Recover$(I, (v_i)_{i\in I}) \to (w, r)$



A Refined Definition of VSS

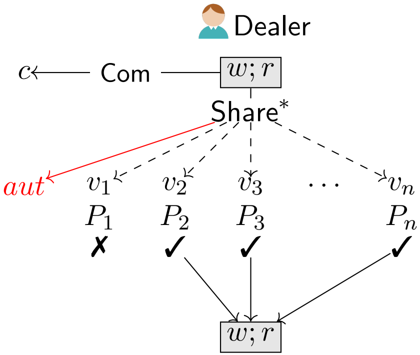
- Setup$(1^\lambda) \to pp$

  include n, t_p, t_f
- Share$(w) \to (c, (v_i)_{i\in[n]}, aut)$
  - Com$(w; r) \to c$

    r: could be empty
  - Share$^*(w, r) \to ((v_i)_{i\in[n]}, aut)$

    aut: authentication information

    (a commitment to the sharing procedure)
- Check$(i, v_i, c, aut) \to 0/1$
- Recover$(I, (v_i)_{i\in I}) \to (w, r)$



- **Acceptance:** valid shares $w_i \Rightarrow$ Check$(i, v_i, c, aut) = 1$
- t_p-**Privacy:** # [shares] $\leq t_p \Rightarrow$ leak nothing about w
- **Consistency:** # [valid shares] $\geq t_f \Rightarrow$ unique w and recover w (previous)

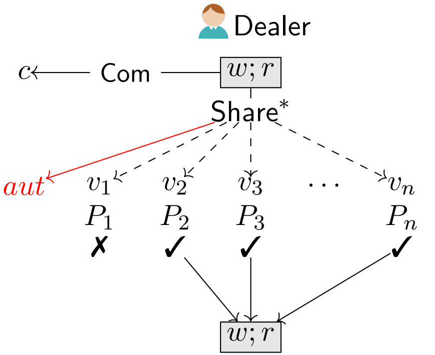
A Refined Definition of VSS

- $\mathsf{Setup}(1^\lambda) \to pp$

  include n, t_p, t_f
- $\mathsf{Share}(w) \to (c, (v_i)_{i\in[n]}, aut)$
  - $\mathsf{Com}(w;r) \to c$

    r: could be empty
  - $\mathsf{Share}^*(w,r) \to ((v_i)_{i\in[n]}, aut)$

    aut: authentication information
    (a commitment to the sharing procedure)
- $\mathsf{Check}(i, v_i, c, aut) \to 0/1$
- $\mathsf{Recover}(I, (v_i)_{i\in I}) \to (w, r)$

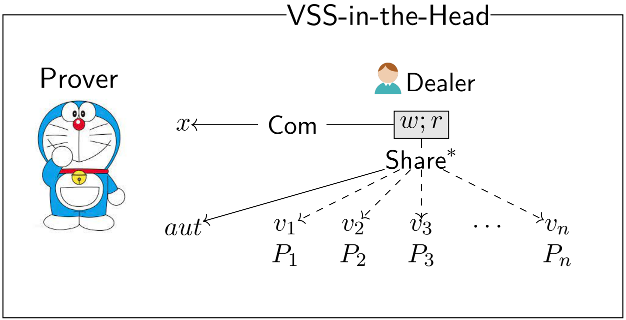


- **Acceptance:** valid shares $w_i \Rightarrow \mathsf{Check}(i, v_i, c, aut) = 1$
- t_p-**Privacy:** # [shares] $\leq t_p \Rightarrow$ leak nothing about w
- t_f-**Correctness:** # [valid shares] $\geq t_f \Rightarrow$ recover $(w,r) \wedge \mathsf{Com}(w;r) = c$

Sigma Protocols from VSS

$\mathsf{Com}(w; r) = x$

Com: an algebraic committing algorithm

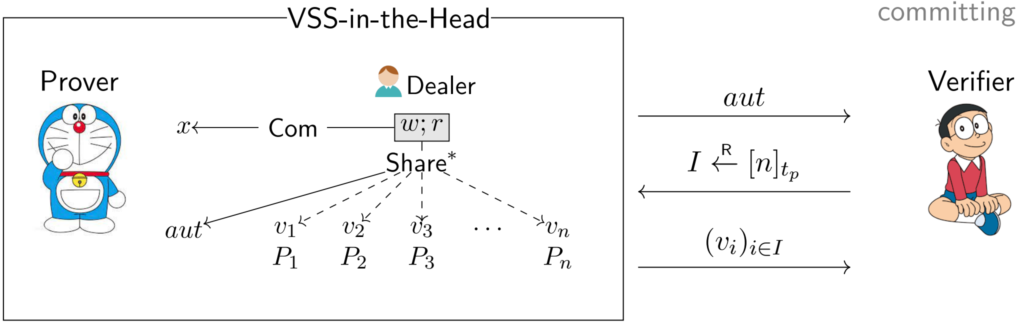


Verifier

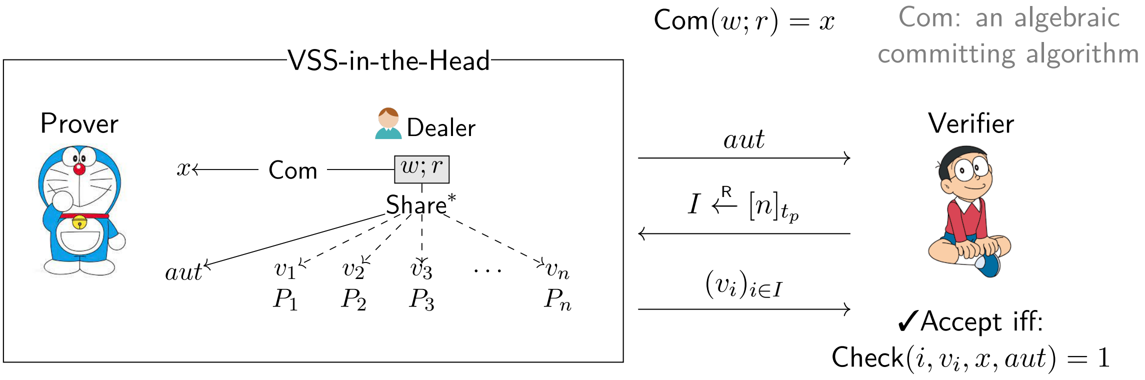
Sigma Protocols from VSS

$\mathsf{Com}(w; r) = x$

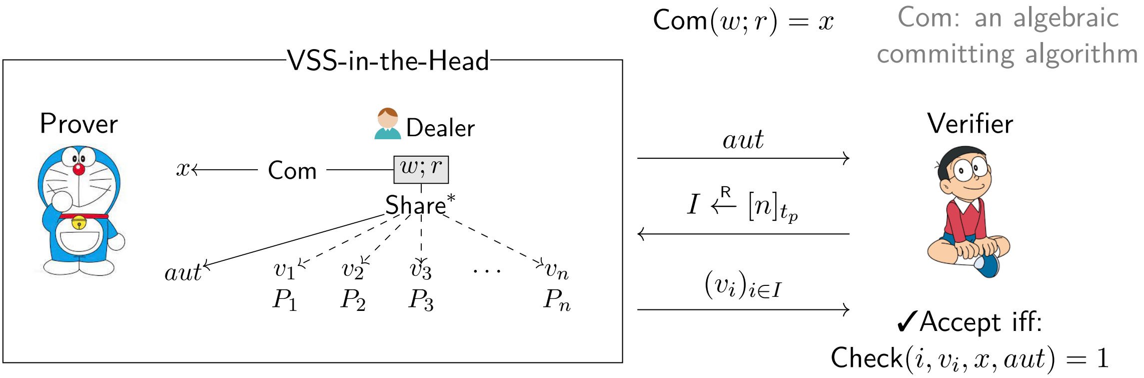
Com: an algebraic committing algorithm



Sigma Protocols from VSS

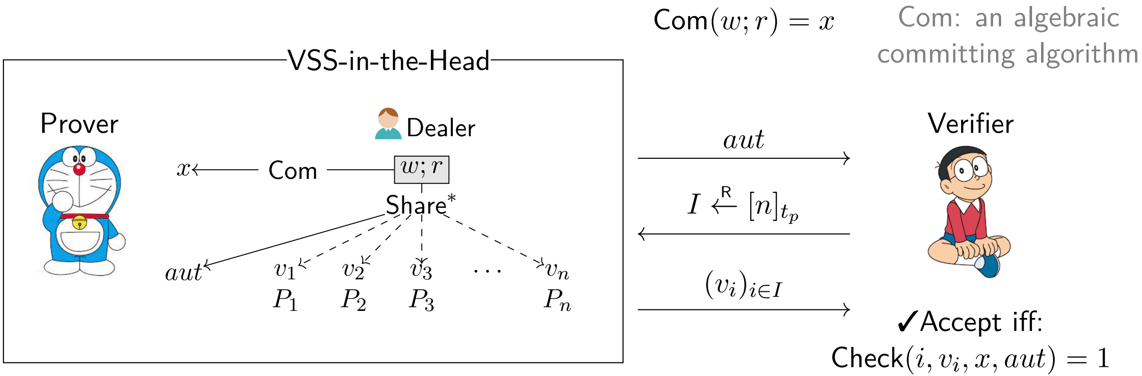


Sigma Protocols from VSS



- Completeness $\Leftarrow$ VSS Acceptance
- Special soundness $\Leftarrow$ VSS t_f-Correctness
- SHVZK $\Leftarrow$ VSS t_p-Privacy

Sigma Protocols from VSS



☺

- Neatly explain classic Sigma protocols [Sch91, GQ88, Oka92].
- Give a generic way to construct Sigma protocols.

Instantiation I: the Schnorr Protocol

Feldman's VSS scheme [Fel87]:
[parties] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$.

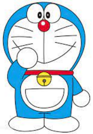
VSS-in-the-Head

$g^w = x \ (r = \perp)$

Dealer

$x \leftarrow g^w \leftarrow \boxed{w, \perp}$

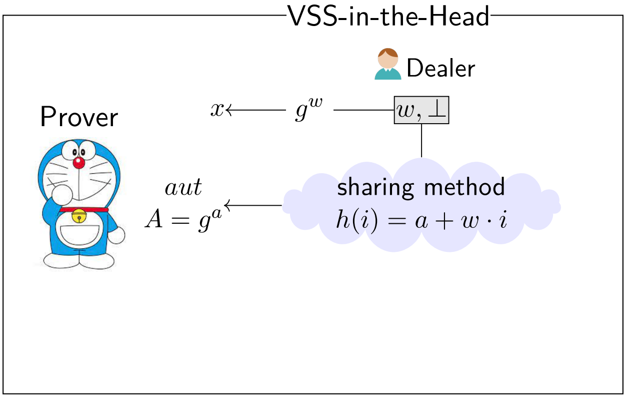
Prover



Verifier

Instantiation I: the Schnorr Protocol

Feldman's VSS scheme [Fel87]:
[parties] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$.

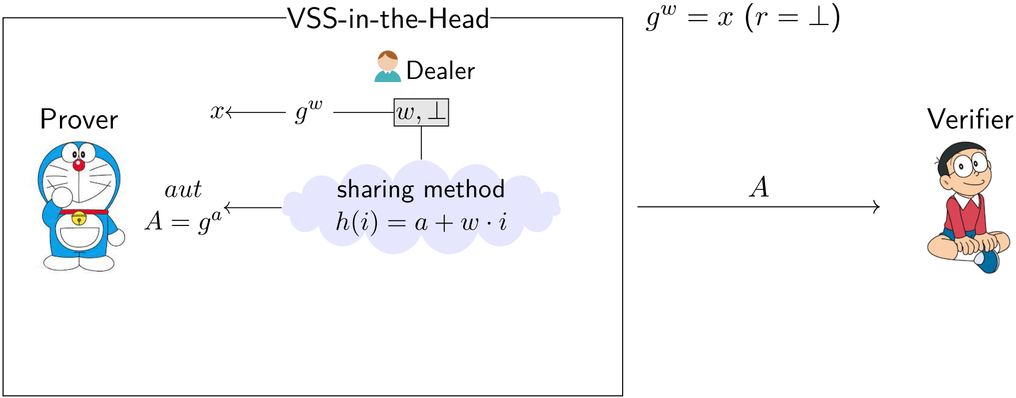


$g^w = x \ (r = \perp)$

Instantiation I: the Schnorr Protocol

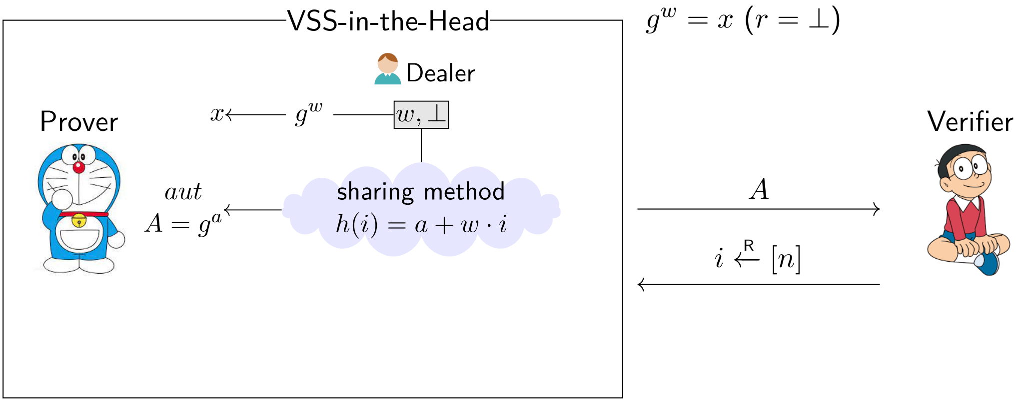
Feldman's VSS scheme [Fel87]:
[parties] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$.

$g^w = x \ (r = \perp)$



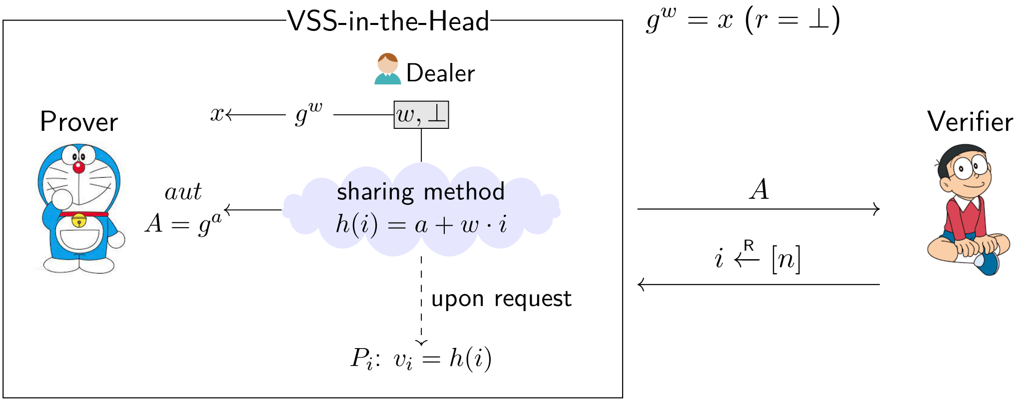
Instantiation I: the Schnorr Protocol

Feldman's VSS scheme [Fel87]:
[parties] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$.



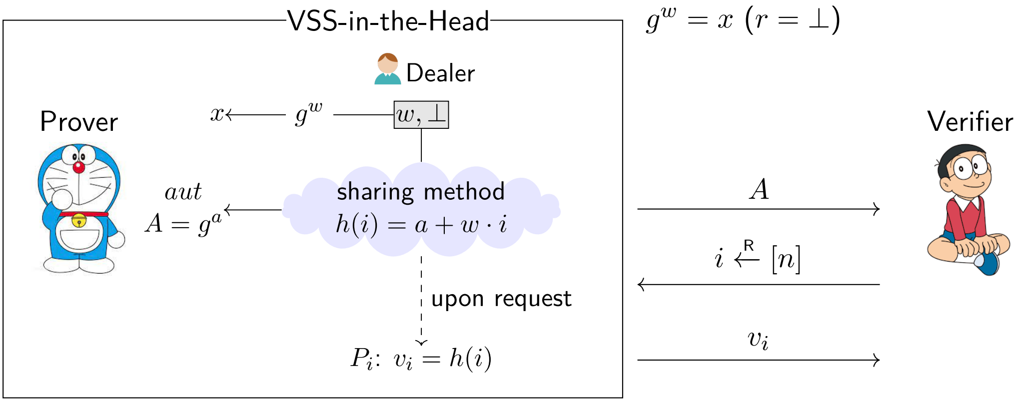
Instantiation I: the Schnorr Protocol

Feldman's VSS scheme [Fel87]:
[parties] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$.



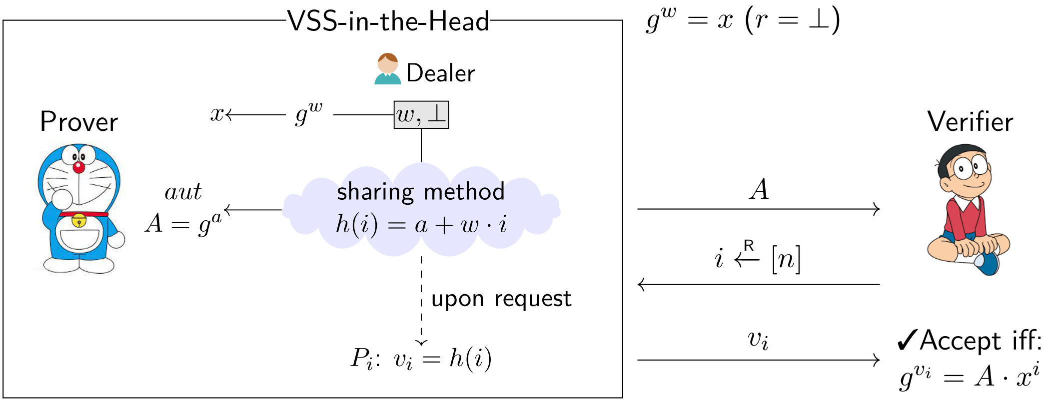
Instantiation I: the Schnorr Protocol

Feldman's VSS scheme [Fel87]:
\# [parties] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$.



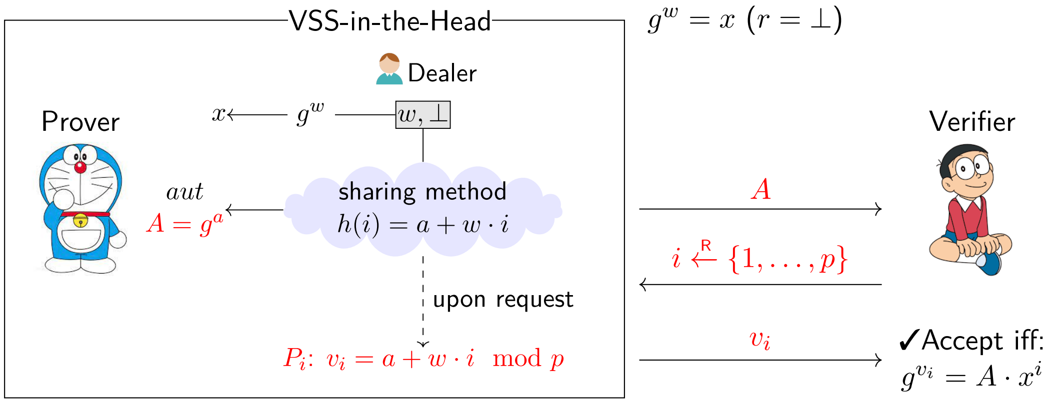
Instantiation I: the Schnorr Protocol

Feldman's VSS scheme [Fel87]:
[parties] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$.



Instantiation I: the Schnorr Protocol

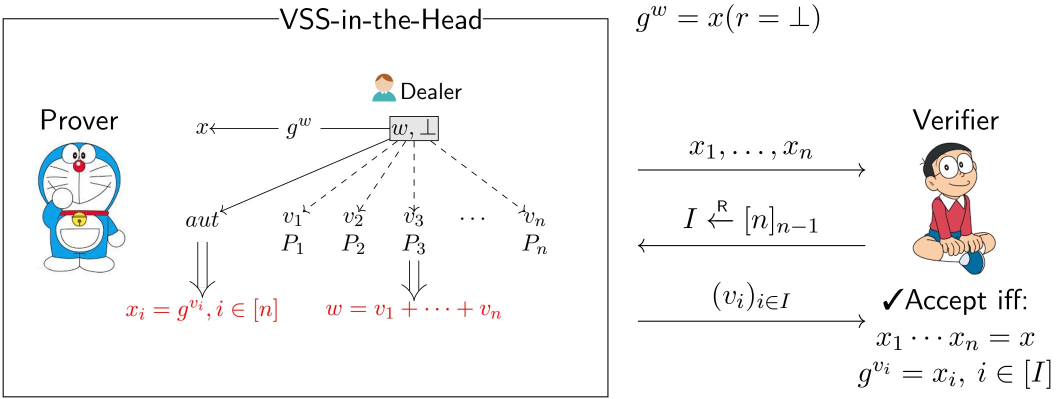
Feldman's VSS scheme [Fel87]:
[parties] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$.



Set $n = |\mathbb{Z}_p| \Rightarrow$ Schnorr protocol [Sch91].

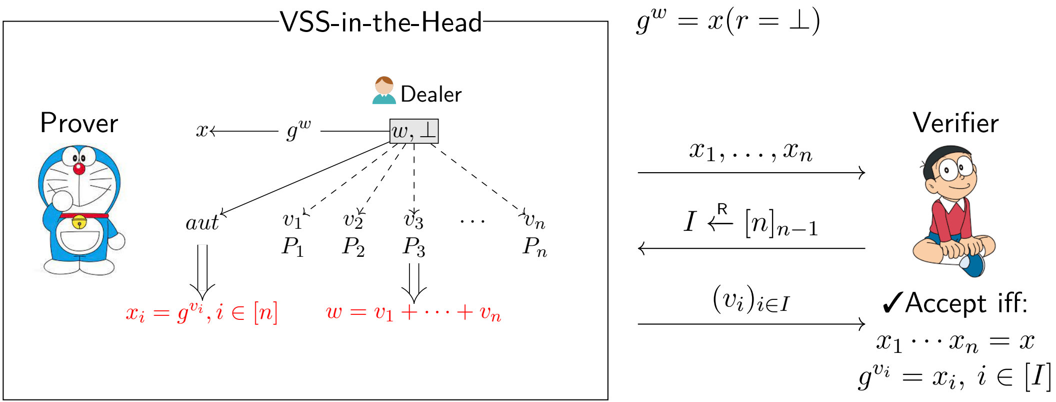
Instantiation II: A New Sigma Protocol for DL

Additive VSS scheme:
[parties] $= n$, privacy threshold $t_p = n - 1$, fault-tolerance threshold $t_f = n$.



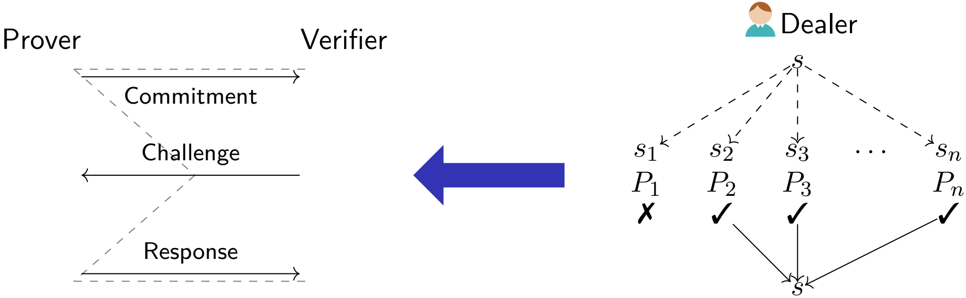
Instantiation II: A New Sigma Protocol for DL

Additive VSS scheme:
[parties] $= n$, privacy threshold $t_p = n - 1$, fault-tolerance threshold $t_f = n$.



A Sigma protocol for DL with 2-special soundness.

Outline



Is there any other application of VSS-in-the-Head?

Forms of Statements in Zero-knowledge Proofs (ZKPs)

Algebraic Statements

functions over some groups

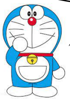
$\Uparrow$

Sigma (Σ) protocols

- Schnorr [Sch91]
- Okamoto [Oka92]
- GQ [GQ88]

I know w such that

$g^w = x$

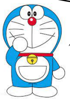


Forms of Statements in Zero-knowledge Proofs (ZKPs)

Algebraic Statements	Non-Algebraic Statements
functions over some groups	boolean/arithmetic circuits
$\Uparrow$	$\Uparrow$
Sigma (Σ) protocols	General-purpose ZKPs

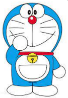
- Schnorr [Sch91]
- Okamoto [Oka92]
- GQ [GQ88]

I know w such that
$g^w = x$



- PCP, IPCP, IOP [Kil92]
- Linear PCP [IKO07]
- Garbled circuit [JKO13]

I know w such that
$\mathsf{SHA}(w) = x$



Composite Statements

Algebraic Statements + Non-Algebraic Statements

e.g. $g^{w_1} = x$ e.g. $\mathsf{SHA}(w_2) = y$

$\Downarrow$

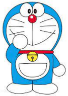
combine in arbitrary ways

e.g. $w_1 = w_2$

$\Downarrow$

Composite Statements

I know w such that
$g^w = x \wedge \mathsf{SHA}(w) = y$



Composite Statements

Algebraic Statements + Non-Algebraic Statements

e.g. $g^{w_1} = x$ || e.g. $\mathsf{SHA}(w_2) = y$

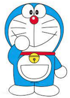
combine in arbitrary ways

e.g. $w_1 = w_2$

$\Downarrow$

Composite Statements

I know w such that
$g^w = x \wedge \mathsf{SHA}(w) = y$



Commit-and-Prove Type:

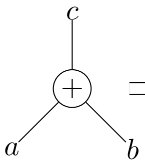
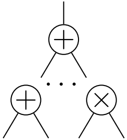
I know w such that
$\mathsf{Com}(w) = x \wedge C(w) = y$

Com: algebraic commitment

C: arithmetic or boolean circuit

ZKPs for Commit-and-Prove Type Composite Statements

Naïve method: Homogenize the form then use only Σ protocols or general-purpose ZKPs.

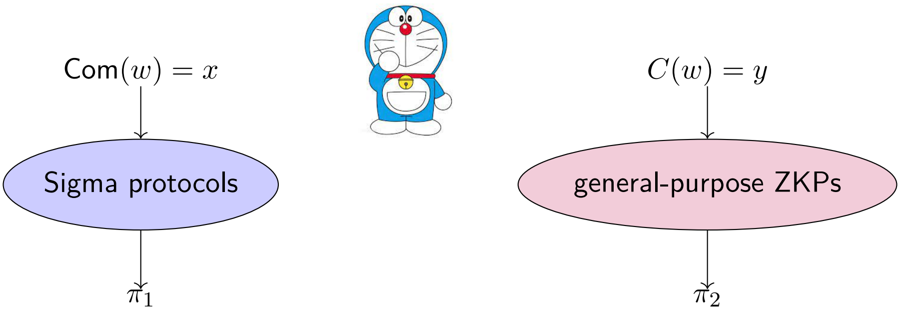
circuits $\Rightarrow$ algebraic constraints	algebraic constraints $\Rightarrow$ circuits
 $\Longrightarrow$ $g^a \cdot g^b = g^c$	$g^w = x$ $\Longrightarrow$ 
# [public-key ops] and # [group elements] linear to the circuit size	size of the statements dramatically increases [2]

☹ Both directions introduce significant overhead.

[2] As noted by [AGM18], the circuit for computing a single exponentiation could be of thousands or millions of gates depending on the group size.

ZKPs for Commit-and-Prove Type Composite Statements

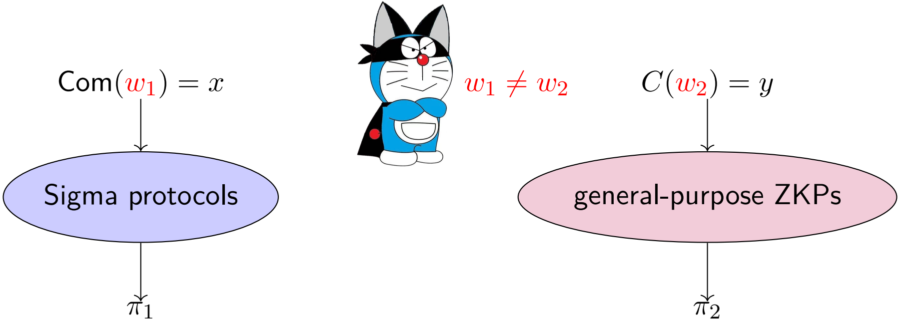
- A better method:



Take advantages of both Sigma protocols and general-purpose ZKPs. ☺

ZKPs for Commit-and-Prove Type Composite Statements

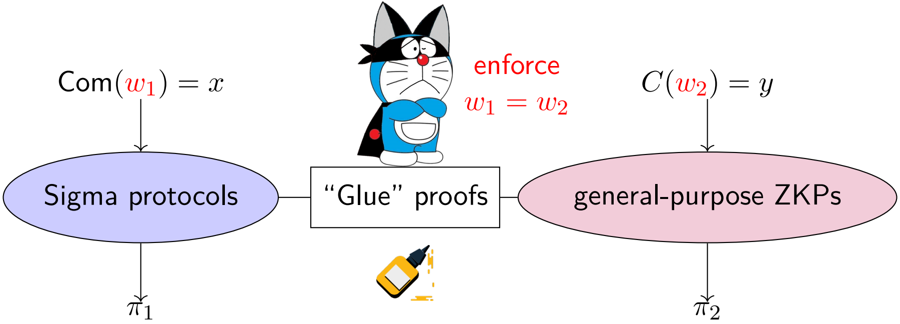
- A better method:



A malicious prover could generate π_1 and π_2 using $w_1 \neq w_2$. ☹

ZKPs for Commit-and-Prove Type Composite Statements

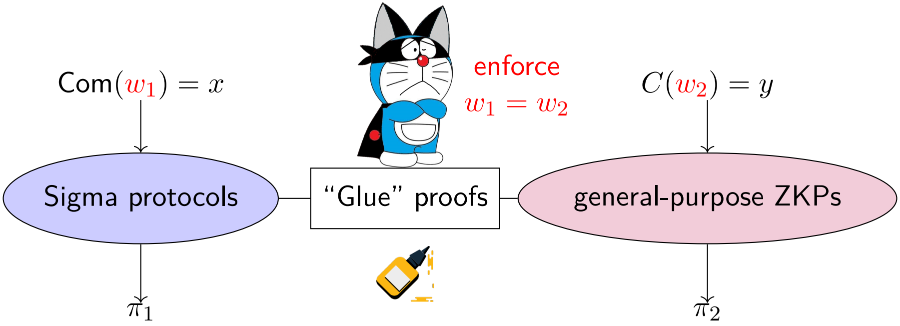
- A better method: [CGM16, AGM18, CFQ19, ABC+22, BHH+19]



The prover is enforced to generate π_1 and π_2 using $w_1 = w_2$. ☺

ZKPs for Commit-and-Prove Type Composite Statements

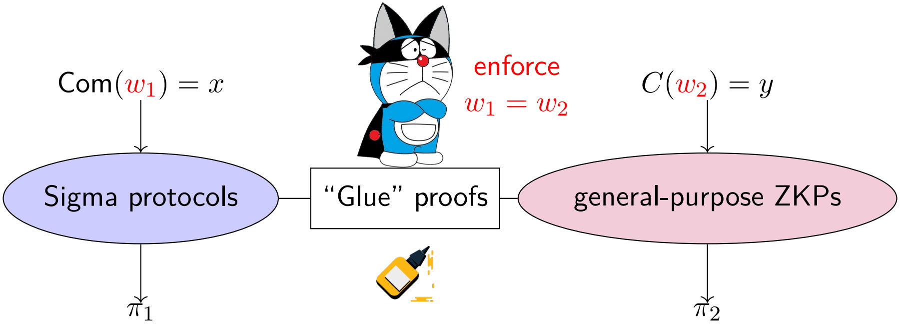
- A better method: [CGM16, AGM18, CFQ19, ABC+22, BHH+19]



1. Inevitably incur additional overheads in computation cost and proof size ☹
2. Must be tailored in a specific way to align with the general-purpose ZKPs
   $\rightsquigarrow$ Require extra design efforts ☹

ZKPs for Commit-and-Prove Type Composite Statements

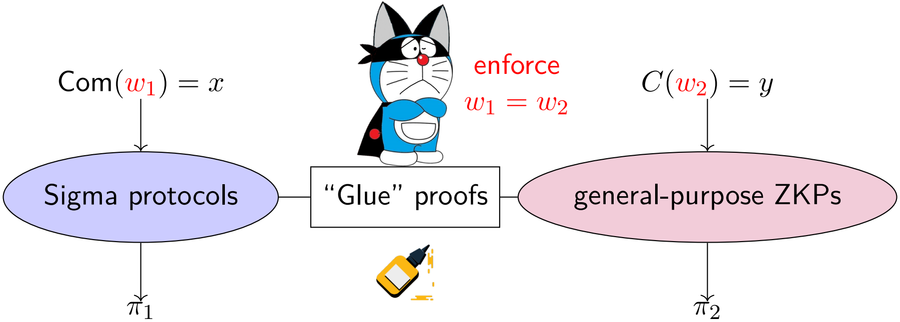
- A better method: [CGM16, AGM18, CFQ19, ABC+22, BHH+19]



Whether the seemingly indispensable "glue" proofs are necessary?

ZKPs for Commit-and-Prove Type Composite Statements

- A better method: [CGM16, AGM18, CFQ19, ABC$^+$22, BHH$^+$19]



Whether the seemingly indispensable "glue" proofs are necessary?

VSS-in-the-head paradigm gives rise to
a generic construction of ZKPs for composite statements without "glue" proofs

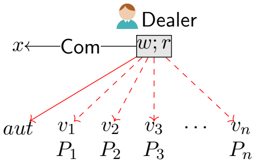
Main Observation

$\mathsf{Com}(w;r) = x$

Prover — Verifier

VSS-in-the-Head

1. Share w :



Prover → Verifier: aut

Verifier → Prover: $I \subset_R [n]$

Prover → Verifier: $(v_i)_{i \in I}$

✓ or ✗

$C(w) = y$

Prover — Verifier

MPC-in-the-Head

1. Share w: $w = w_1 \oplus \cdots \oplus w_n$
2. Run MPC protocol Π_C: $\Rightarrow P_i : w_i || view_i$
3. Commit to the views: c_1 c_2 $\ldots$ c_n

Prover → Verifier: $c_1, \ldots, c_n$

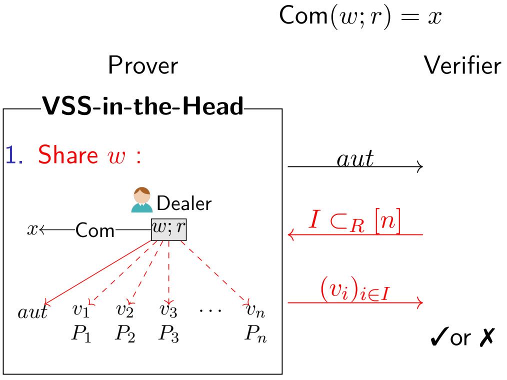
Verifier → Prover: $I \subset_R [n]$

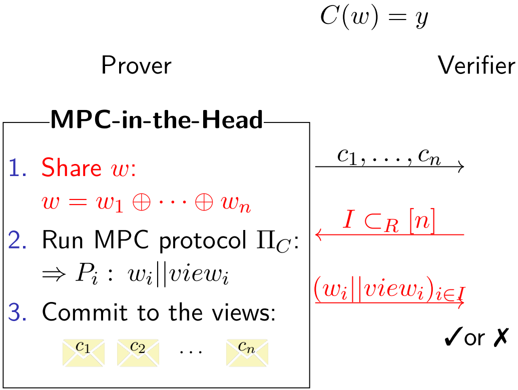
Prover → Verifier: $(w_i || view_i)_{i \in I}$

✓ or ✗

Share the same Σ pattern & same secret sharing procedure!

Main Observation





Share the same Σ pattern & same secret sharing procedure!

reuse witness sharing procedure

$\Rightarrow$ Enforce the prover to use consistent witness without "glue" proofs

Two Main Technical Obstacles

1. The secret sharing mechanism in the MPC-in-the-head [IKOS07] sticks to $w = w_1 \oplus \cdots \oplus w_n$ (a special case of $(n, n-1, n)$-SS scheme).

   $\rightsquigarrow$ Make it hard to interact with general (n, t_p, t_f)-VSS schemes.

2. The relationship between VSS and SS is unclear.

   $\rightsquigarrow$ Make it difficult to reuse the common part of witness sharing procedure.

Two Main Technical Obstacles

1. The secret sharing mechanism in the MPC-in-the-head [IKOS07] sticks to $w = w_1 \oplus \cdots \oplus w_n$ (a special case of $(n, n-1, n)$-SS scheme).

   $\rightsquigarrow$ Make it hard to interact with general (n, t_p, t_f)-VSS schemes.

2. The relationship between VSS and SS is unclear.

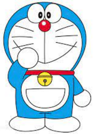
   $\rightsquigarrow$ Make it difficult to reuse the common part of witness sharing procedure.

A Generalized Version of MPC-in-the-Head

$$C(w) = y$$

MPC-in-the-Head

Prover



1. Share w :

   $w = w_1 \oplus \cdots \oplus w_n$

   $(n, n-1, n)$-secret sharing scheme

   $(w_1, \ldots, w_n) \leftarrow \mathsf{SS.Share}(w)$

   (n, t_p, t_f)-secret sharing scheme

2. Run MPC protocol Π_C :

   $\Rightarrow P_i : \ w_i || view_i$

3. Commit to the views :

   c_1 c_2 $\cdots$ c_n

$c_1, \ldots, c_n$ $\longrightarrow$

$\longleftarrow$ $I \subset_R [n]$

$(w_i || view_i)_{i \in I}$ $\longrightarrow$

Verifier

✓Accept iff:

$(c_i)_{i \in I}$ opened successfully

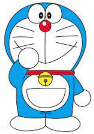
output=1 & consistent

A Generalized Version of MPC-in-the-Head

$C(w) = y$

MPC-in-the-Head

Prover



1. Share w :
   $w = w_1 \oplus \cdots \oplus w_n$
   $(n, n-1, n)$-secret sharing scheme
   $(w_1, \ldots, w_n) \leftarrow \text{SS.Share}(w)$
   (n, t_p, t_f)-secret sharing scheme
2. Run MPC protocol Π_C :
   $\Rightarrow P_i : \; w_i || view_i$
3. Commit to the views :
   c_1 c_2 $\cdots$ c_n

$c_1, \ldots, c_n \longrightarrow$

$\longleftarrow I \subset_R [n]$

$(w_i || view_i)_{i \in I} \longrightarrow$

Verifier

✓Accept iff:
$(c_i)_{i \in I}$ opened successfully
output=1 & consistent

- Completeness $\Leftarrow$ SS $+ \Pi_C +$Commit correctness
- Special soundness $\Leftarrow$ Π_C consistency+SS correctness
- SHVZK $\Leftarrow$ SS $+ \Pi_C$ privacy

Two Main Technical Obstacles

1. **The secret sharing mechanism in the MPC-in-the-head [IKOS07] sticks to $w = w_1 \oplus \cdots \oplus w_n$ (a special case of $(n, n-1, n)$-SS scheme).**

   $\rightsquigarrow$ Make it hard to interact with (n, t_p, t_f)-VSS schemes.

2. **The relationship between VSS and SS is unclear.**

   $\rightsquigarrow$ Make it difficult to reuse the common part of witness sharing procedure.

Definition 1 (Separability)

The algorithms $\mathsf{VSS.Share}^*(w, r) \to ((v_i)_{i\in[n]}, aut)$ can be dissected as below:

$$(w_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(w)$$
$$(r_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(r)$$
$$aut \leftarrow \mathsf{AutGen}((w_i, r_i)_{i\in[n]})$$

Separable VSS: A Relationship between VSS and SS

Definition 1 (Separability)

The algorithms $\mathsf{VSS.Share}^*(w, r) \to ((v_i)_{i\in[n]}, aut)$ can be dissected as below:

$$(w_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(w)$$
$$(r_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(r)$$
$$aut \leftarrow \mathsf{AutGen}((w_i, r_i)_{i\in[n]})$$

$\mathsf{VSS.Share}^*(w, r)$

Separable VSS: A Relationship between VSS and SS

Definition 1 (Separability)

The algorithms $\mathsf{VSS.Share}^*(w, r) \to ((v_i)_{i\in[n]}, aut)$ can be dissected as below:

$$(w_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(w)$$
$$(r_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(r)$$
$$aut \leftarrow \mathsf{AutGen}((w_i, r_i)_{i\in[n]})$$

$$\mathsf{VSS.Share}^*(w, r) \begin{cases} \text{Generate shares } v_i \\ \text{Generate } aut \end{cases}$$

Separable VSS: A Relationship between VSS and SS

Definition 1 (Separability)

The algorithms $\mathsf{VSS.Share}^*(w, r) \to ((v_i)_{i\in[n]}, aut)$ can be dissected as below:

$$(w_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(w)$$
$$(r_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(r)$$
$$aut \leftarrow \mathsf{AutGen}((w_i, r_i)_{i\in[n]})$$

$$\mathsf{VSS.Share}^*(w, r) \begin{cases} \text{Generate shares } v_i \begin{cases} w_i \\ r_i \end{cases} \\ \text{Generate } aut \end{cases}$$

Separable VSS: A Relationship between VSS and SS

Definition 1 (Separability)

The algorithms $\mathsf{VSS.Share}^*(w, r) \rightarrow ((v_i)_{i\in[n]}, aut)$ can be dissected as below:

$$(w_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(w)$$
$$(r_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(r)$$
$$aut \leftarrow \mathsf{AutGen}((w_i, r_i)_{i\in[n]})$$

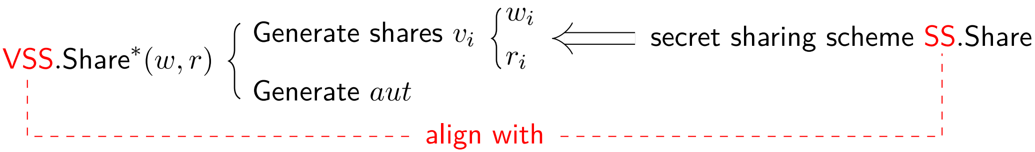
$$\mathsf{VSS.Share}^*(w, r) \begin{cases} \text{Generate shares } v_i \begin{cases} w_i \\ r_i \end{cases} \Longleftarrow \text{secret sharing scheme SS.Share} \\ \text{Generate } aut \end{cases}$$

Separable VSS: A Relationship between VSS and SS

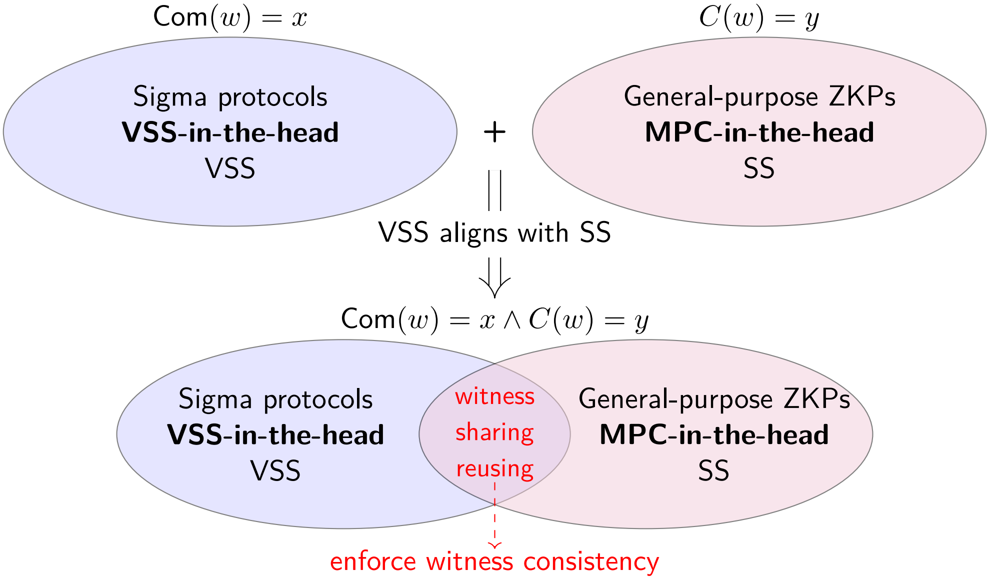
Definition 1 (Separability)

The algorithms $\mathsf{VSS.Share}^*(w, r) \to ((v_i)_{i\in[n]}, aut)$ can be dissected as below:

$$(w_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(w)$$
$$(r_i)_{i\in[n]} \leftarrow \mathsf{SS.Share}(r)$$
$$aut \leftarrow \mathsf{AutGen}((w_i, r_i)_{i\in[n]})$$



Combination of Two Worlds

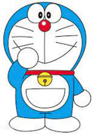


A Generic Construction of ZKPs for Commit-and-Prove Type Composite Statements

$\mathsf{Com}(w; r) = x \wedge C(w) = y$

(VSS+MPC)-in-the-Head

Prover



1. Share w, r using VSS.Share*:
   $(w_1, \ldots, w_n) \leftarrow \mathsf{SS.Share}(w)$
   $(r_1, \ldots, r_n) \leftarrow \mathsf{SS.Share}(r)$
   $aut \leftarrow \mathsf{AutGen}((w_i, r_i)_{i \in [n]})$
2. Run MPC protocol Π_C :
   $\Rightarrow P_i :\ w_i || view_i$
3. Commit to the views :
   c_1 c_2 $\ldots$ c_n

$c_1, \ldots, c_n, aut$ $\longrightarrow$

$\longleftarrow$ $I \subset_R [n]$

$(w_i || view_i,\ r_i)_{i \in I}$ $\longrightarrow$

Verifier

Accept iff:
MPC-in-the-head check ✓
VSS-in-the-head check ✓

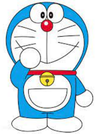
- Completeness $\Leftarrow$ VSS separability+(VSS/MPC)-in-the-head completeness
- Special soundness $\Leftarrow$ witness sharing reusing+(VSS/MPC)-in-the-head special soundness
- SHVZK $\Leftarrow$ (VSS/MPC)-in-the-head SHVZK

A Generic Construction of ZKPs for Commit-and-Prove Type Composite Statements

$\mathsf{Com}(w; r) = x \wedge C(w) = y$

(VSS+MPC)-in-the-Head

Prover



1. Share w, r using VSS.Share*:
   $(w_1, \ldots, w_n) \leftarrow \mathsf{SS.Share}(w)$
   $(r_1, \ldots, r_n) \leftarrow \mathsf{SS.Share}(r)$
   $aut \leftarrow \mathsf{AutGen}((w_i, r_i)_{i \in [n]})$
2. Run MPC protocol Π_C :
   $\Rightarrow P_i : w_i || view_i$
3. Commit to the views :

$c_1, \ldots, c_n, aut \longrightarrow$

$\longleftarrow I \subset_R [n]$

$(w_i || view_i, r_i)_{i \in I} \longrightarrow$

Verifier

Accept iff:

MPC-in-the-head check ✓

VSS-in-the-head check ✓

no "glue" proofs | public-coin | transparent

An Instantiation from Ligero++ (CCS 2020: Bhadauria et al.)

Step 1: Identify the SS scheme used in Ligero++

Randomized Reed-Solomon code	$\Rightarrow$	Packed Shamir's SS scheme
length of the code n		number of participants n
length of the message k		fault-tolerance threshold $t_f = k$
number of the randomness $\hat{t}$		privacy threshold $t_p = \hat{t}$

An Instantiation from Ligero++ (CCS 2020: Bhadauria et al.)

Step 1: Identify the SS scheme used in Ligero++

Step 2: Construct a VSS scheme that aligns with this SS

Randomized Reed-Solomon code

length of the code n

length of the message k

number of the randomness $\hat{t}$

$\Longrightarrow$

Packed Shamir's SS scheme

number of participants n

fault-tolerance threshold $t_f = k$

privacy threshold $t_p = \hat{t}$

$\Longrightarrow$

VSS scheme

number of participants n

fault-tolerance threshold $t_f = k$

privacy threshold $t_p = \hat{t}$

An Instantiation from Ligero++ (CCS 2020: Bhadauria et al.)

Step 1: Identify the SS scheme used in Ligero++

Step 2: Construct a VSS scheme that aligns with this SS

Randomized Reed-Solomon code
length of the code n
length of the message k
number of the randomness $\hat{t}$

$\Longrightarrow$

Packed Shamir's SS scheme
number of participants n
fault-tolerance threshold $t_f = k$
privacy threshold $t_p = \hat{t}$

$\Longrightarrow$

VSS scheme
number of participants n
fault-tolerance threshold $t_f = k$
privacy threshold $t_p = \hat{t}$

Solve the open problem left in [BHH$^+$19] ☺:

> the prover's running time is critical. As future work, it would be interesting to explore whether the approach by Ames et al. [4] can be used to achieve yet more efficient and compact NIZK proofs in cross-domains.

Protocols	Prover time	Verifier time	Proof size
[BHH$^+$19]	$O((\lvert w\rvert + \lambda)$ pub $O(\lvert C\rvert \cdot \lambda)$ sym	$O((\lvert w\rvert + \lambda)$ pub $O(\lvert C\rvert \cdot \lambda)$ sym	$O(\lvert C\rvert\lambda + \lvert w\rvert)$
This work	$O(\lambda)$ pub $O(\lvert C\rvert \log(\lvert C\rvert))$ sym	$O(\frac{(\lvert w\rvert+\lambda)^2}{\log(\lvert w\rvert+\lambda)})$ pub $O(\lvert C\rvert)$ sym	$O(\text{polylog}(\lvert C\rvert) + \lambda)$

Outline

Summary

- **A framework of Sigma protocols for algebraic statements**
  - A refined definition of VSS
  - VSS-in-the-head paradigm

☺
- Neatly explain classic Sigma protocols [Sch91, GQ88, Oka92].
- Give a generic way to construct Sigma protocols.

- **A generic construction of ZKPs for commit-and-prove type composite statements**
  - Technique: witness sharing reusing
  - A Generalization of MPC-in-the-head paradigm
  - Separability of VSS scheme: define the relationship between VSS and SS
  - An instantiation from Ligero++

☺ no "glue" proofs public-coin transparent

Thanks for Your Attention!

Any Questions?

Reference I

Masayuki Abe, Miguel Ambrona, Andrej Bogdanov, Miyako Ohkubo, and Alon Rosen.
Acyclicity programming for sigma-protocols.
In *TCC*, 2021.

Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi.
ECLIPSE: enhanced compiling method for pedersen-committed zksnark engines.
In *PKC*, 2022.

Shashank Agrawal, Chaya Ganesh, and Payman Mohassel.
Non-interactive zero-knowledge proofs for composite statements.
In *CRYPTO*, 2018.

Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit.
Short accountable ring signatures based on DDH.
In *ESORICS*, 2015.

Michael Backes, Lucjan Hanzlik, Amir Herzberg, Aniket Kate, and Ivan Pryvalov.
Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup.
In *PKC*, 2019.

Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler.
Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs.
In *CRYPTO*, 2019.

Reference II

Fabrice Boudot.
Efficient proofs that a committed number lies in an interval.
In *EUROCRYPT*, 2000.

Ronald Cramer, Ivan Damgård, and Berry Schoenmakers.
Proofs of partial knowledge and simplified design of witness hiding protocols.
In *CRYPTO*, 1994.

Matteo Campanelli, Dario Fiore, and Anaïs Querol.
Legosnark: Modular design and composition of succinct zero-knowledge proofs.
In *ACM CCS*, 2019.

Melissa Chase, Chaya Ganesh, and Payman Mohassel.
Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials.
In *CRYPTO*, 2016.

Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch.
Verifiable secret sharing and achieving simultaneity in the presence of faults.
In *FOCS*, 1985.

David Chaum and Torben P. Pedersen.
Wallet databases with observers.
In *CRYPTO*, 1992.

Reference III

Paul Feldman.
A practical scheme for non-interactive verifiable secret sharing.
In *FOCS*, 1987.

Amos Fiat and Adi Shamir.
How to prove yourself: practical solutions to identification and signature problems.
In *CRYPTO*, 1986.

Jens Groth and Markulf Kohlweiss.
One-out-of-many proofs: Or how to leak a secret and spend a coin.
In *EUROCRYPT*, 2015.

Rosario Gennaro, Darren Leigh, Ravi Sundaram, and William S. Yerazunis.
Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices.
In *ASIACRYPT*, 2004.

Louis C. Guillou and Jean-Jacques Quisquater.
A "paradoxical" indentity-based signature scheme resulting from zero-knowledge.
In *CRYPTO*, 1988.

Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky.
Efficient arguments without short pcps.
In *IEEE CCC*, 2007.

Reference IV

Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai.
Zero-knowledge from secure multiparty computation.
In *STOC*, 2007.

Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi.
Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently.
In *ACM CCS*, 2013.

Joe Kilian.
A note on efficient zero-knowledge proofs and arguments (extended abstract).
In *STOC*, 1992.

Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon.
Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general.
In *CRYPTO*, 2022.

Ueli Maurer.
Zero-knowledge proofs of knowledge for group homomorphisms.
DCC, 2015.

Tatsuaki Okamoto.
Provably secure and practical identification schemes and corresponding signature schemes.
In *CRYPTO*, 1992.

Reference V

Claus-Peter Schnorr.
Efficient signature generation by smart cards.
Journal of Cryptology, 1991.

Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, Zuoxia Yu, and William Whyte.
Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications.
In *CRYPTO*, 2019.