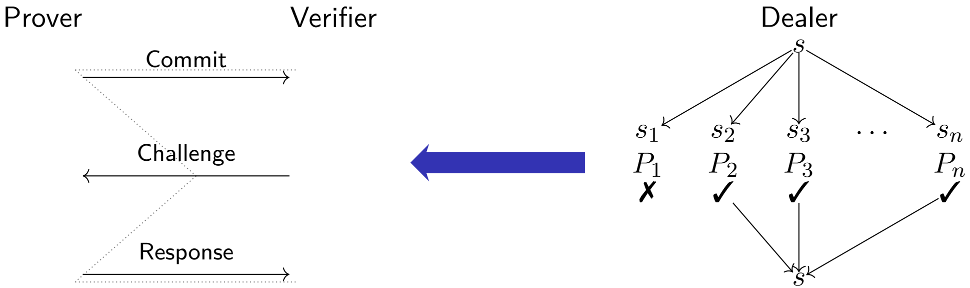
Sigma Protocols from Verifiable Secret Sharing and Their Applications



Yu Chen
Shandong University

Tutorial based on the following joint work

Min Zhang, **Yu Chen**, Chuanzhou Yao, Zhichao Wang
Sigma Protocols from Verifiable Secret Sharing and Their Applications
ASIACRYPT 2023

Outline

1. Background
2. Sigma Protocols from VSS-in-the-Head
3. Applications of VSS-in-the-Head
4. Summary

Outline

1. Background

2. Sigma Protocols from VSS-in-the-Head

3. Applications of VSS-in-the-Head

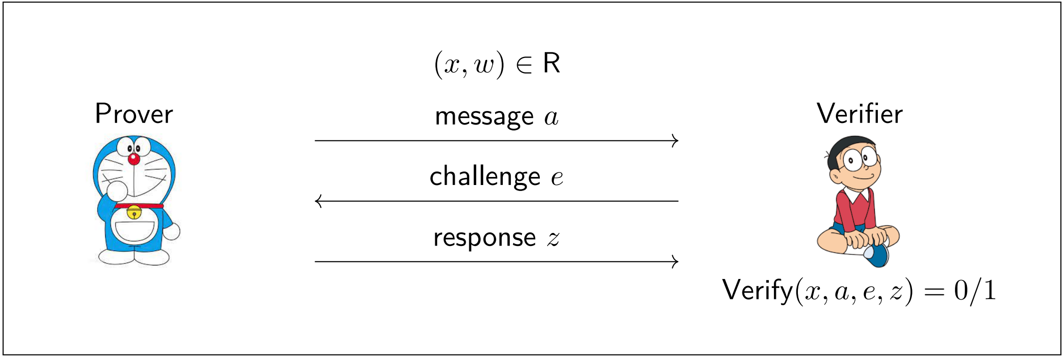
4. Summary

Sigma (Σ) Protocols (Cramer's PhD Thesis)

[Cra96]: Modular Design of Secure yet Practical Cryptographic Protocols

- initiate the formal study of Sigma protocols
- design the first practical CCA-secure PKE in the standard model from HPS
- design information-theoretic secure MPC

Sigma (Σ) Protocols



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

Prover

message a →

challenge e ←

response z →

Verifier

Verify$(x, a, e, z) = 0/1$

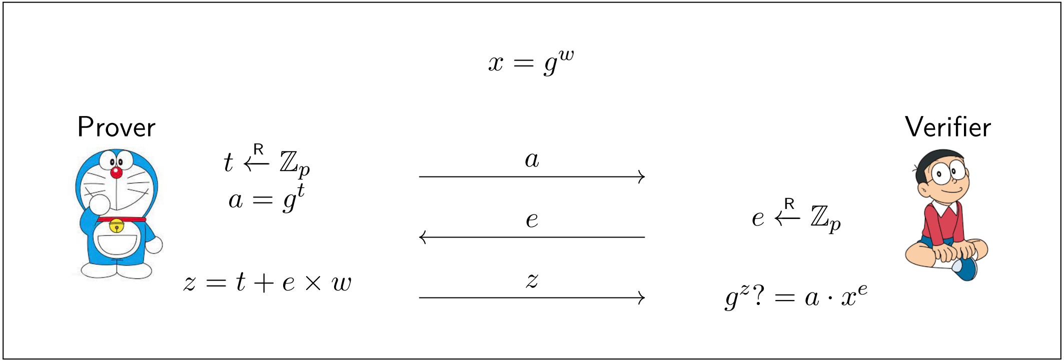
- **Completeness:** $\Pr[\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle = 1 \mid (x,w) \in \mathsf{R}] = 1$
- n**-Special soundness:** $\exists$ PPT Ext that given any x and any n accepting transcripts (a, e_i, z_i) with distinct e_i's can extract w s.t. $(x,w) \in \mathsf{R}$
- **Special honest verifier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

Perhaps the Simplest ZKP Prococol: Schnorr Protocol

[Sch91]: Efficient Signature Generation by Smart Cards

- Cryptography: Schnorr's identification protocol and signature (the tale of patent)
- Algorithmic information theory: and for creating an approach to the definition of an algorithmically random sequence

Perhaps the Simplest ZKP Prococol: Schnorr Protocol



$$x = g^w$$

Prover

$t \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_p$

$a = g^t$

$\xrightarrow{\quad a \quad}$

Verifier

$\xleftarrow{\quad e \quad}$

$e \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_p$

$z = t + e \times w$

$\xrightarrow{\quad z \quad}$

$g^z ? = a \cdot x^e$

- **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 $z \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_p$ and set $a = g^z \cdot x^{-e}$

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

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 ⇝ Full ZK
- Enable numerous real-world applications

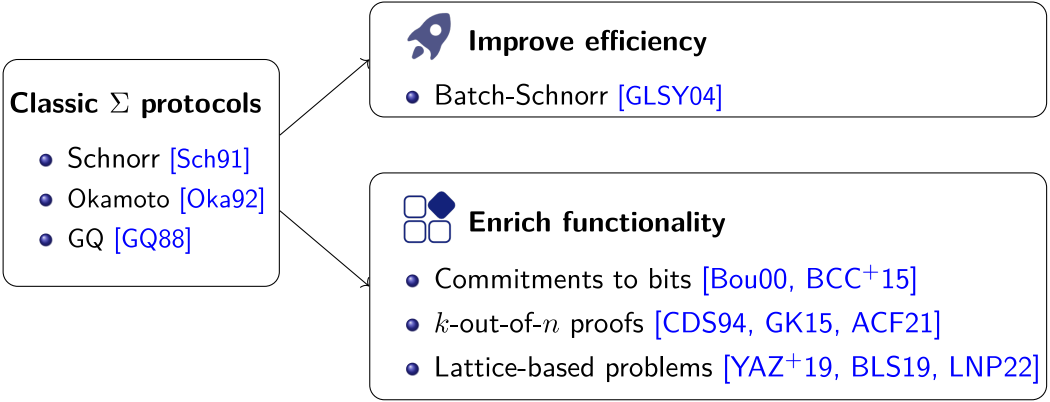
Identification protocols

(Ring) Signature schemes

Anonymous credentials

Privacy-preserving cryptocurrency

Research on Sigma Protocols



Classic Σ protocols

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

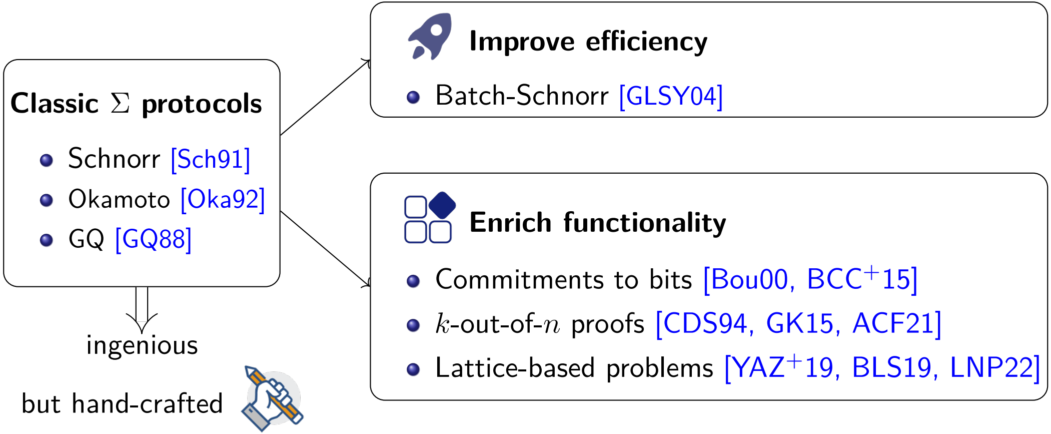
Improve efficiency

- Batch-Schnorr [GLSY04]

Enrich functionality

- Commitments to bits [Bou00, BCC+15]
- k-out-of-n proofs [CDS94, GK15, ACF21]
- Lattice-based problems [YAZ+19, BLS19, LNP22]

Research on Sigma Protocols



Classic Σ protocols

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

ingenious

but hand-crafted

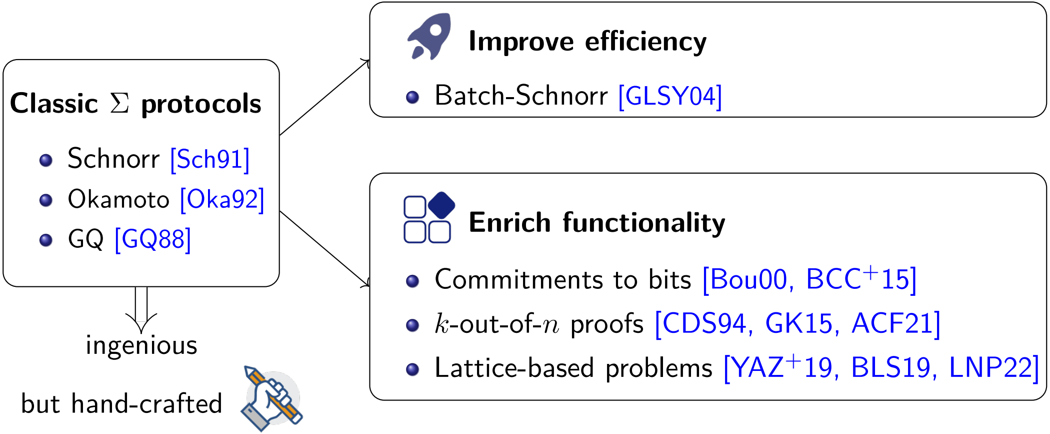
Improve efficiency

- Batch-Schnorr [GLSY04]

Enrich functionality

- Commitments to bits [Bou00, BCC$^+$15]
- k-out-of-n proofs [CDS94, GK15, ACF21]
- Lattice-based problems [YAZ$^+$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]

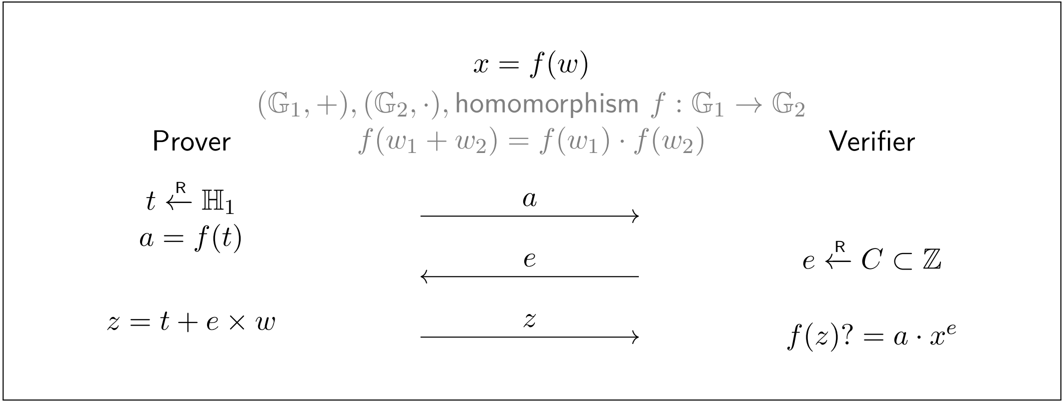
Enrich functionality

- Commitments to bits [Bou00, BCC+15]
- k-out-of-n proofs [CDS94, GK15, ACF21]
- Lattice-based problems [YAZ+19, BLS19, LNP22]

Schnorr's protocol is simple? But, how did Schnorr figure it out?
Whether there exists a common design principal of Sigma protocols?

Maurer's Framework

[Mau15]: Zero-knowledge proofs of knowledge for group homomorphisms



$$x = f(w)$$

$(\mathbb{G}_1, +), (\mathbb{G}_2, \cdot)$, homomorphism $f : \mathbb{G}_1 \to \mathbb{G}_2$

$f(w_1 + w_2) = f(w_1) \cdot f(w_2)$

Prover

Verifier

$t \xleftarrow{\mathsf{R}} \mathbb{H}_1$

$a = f(t)$

$\xrightarrow{a}$

$e \xleftarrow{\mathsf{R}} C \subset \mathbb{Z}$

$\xleftarrow{e}$

$z = t + e \times w$

$\xrightarrow{z}$

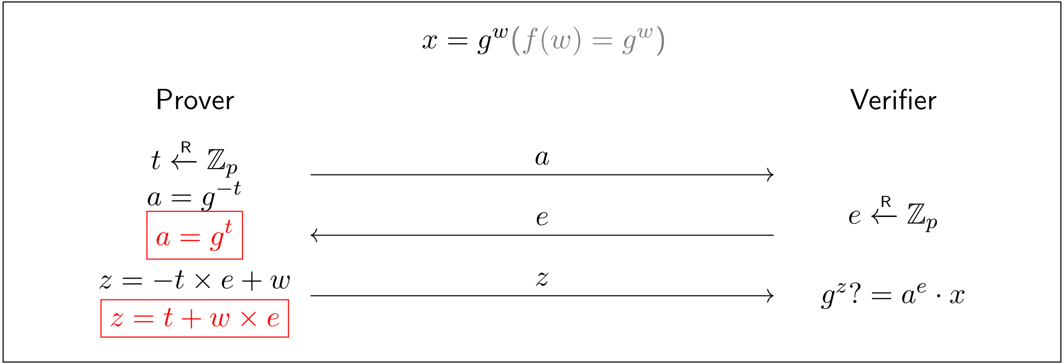
$f(z) ? = a \cdot x^e$

Maurer's Framework

Pros: unifies many protocols, including Schnorr [Sch91], GQ [GQ88], Okamoto [Oka92]

Cons: pattern is fixed ⇝ cannot to explain some simple variants of classic protocols

The framework is superficial and fails to capture the essence



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

Prover

Verifier

$t \xleftarrow{\mathsf{R}} \mathbb{Z}_p$

$a = g^{-t}$

$\boxed{a = g^t}$

$\xrightarrow{a}$

$\xleftarrow{e}$

$e \xleftarrow{\mathsf{R}} \mathbb{Z}_p$

$z = -t \times e + w$

$\boxed{z = t + w \times e}$

$\xrightarrow{z}$

$g^z ?= a^e \cdot x$

Figure: A variant of [Sch91]

Motivation

$$\Sigma$$

The machinery of Sigma protocols is still unclear.
Is there a more generic framework of Sigma protocols?

Outline

1. Background

2. Sigma Protocols from VSS-in-the-Head

3. Applications of VSS-in-the-Head

4. Summary

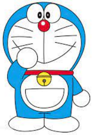
MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

[IKOS07]: Zero-knowledge from secure multiparty computation

$$C(w) = y$$

MPC-in-the-Head

Prover



Verifier

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

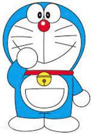
[IKOS07]: Zero-knowledge from secure multiparty computation

$C(w) = y$

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)

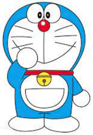
[IKOS07]: Zero-knowledge from secure multiparty computation

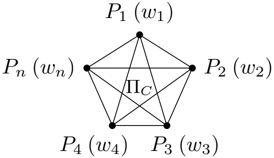
$$C(w) = y$$

MPC-in-the-Head

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

Prover





P_i : $w_i || view_i$

Verifier

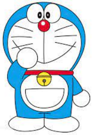
MPC-in-the-head Revisit (STOC 2007: Ishai-Kushilevitz-Ostrovsky-Sahai)

[IKOS07]: Zero-knowledge from secure multiparty computation

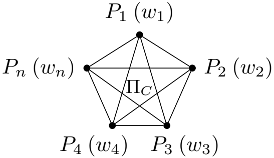
$$C(w) = y$$

MPC-in-the-Head

Prover



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



$P_i\ :\ w_i||view_i$

3. Commit to the views:

$w_1||view_1$ c_1 $w_2||view_2$ c_2 $\cdots$ $w_n||view_n$ c_n

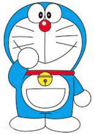
Verifier

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

[IKOS07]: Zero-knowledge from secure multiparty computation

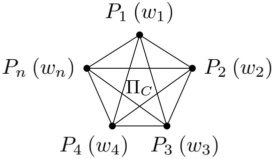
$C(w) = y$

Prover



MPC-in-the-Head

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



$P_i :\ w_i || view_i$

3. Commit to the views:

$w_1||view_1$ c_1 $\quad$ $w_2||view_2$ c_2 $\quad \cdots \quad$ $w_n||view_n$ c_n

$\xrightarrow{c_1, \ldots, c_n}$

Verifier

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

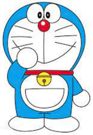
[IKOS07]: Zero-knowledge from secure multiparty computation

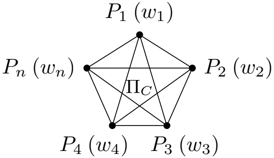
$$C(w) = y$$

MPC-in-the-Head

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

Prover





$P_i :\ w_i||view_i$

3. Commit to the views:

$w_1\|\|view_1$	$w_2\|\|view_2$		$w_n\|\|view_n$
c_1	c_2	$\cdots$	c_n

$$\xrightarrow{c_1, \ldots, c_n}$$

$$\xleftarrow{I \subset [n]}$$

Verifier

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

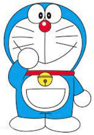
[IKOS07]: Zero-knowledge from secure multiparty computation

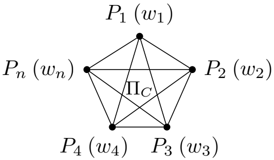
$C(w) = y$

MPC-in-the-Head

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

Prover





$P_i : w_i||view_i$

3. Commit to the views:

$w_1||view_1$	$w_2||view_2$		$w_n||view_n$
c_1	c_2	$\cdots$	c_n

$$\xrightarrow{c_1, \ldots, c_n}$$

$$\xleftarrow{I \subset [n]}$$

$$\xrightarrow{\{w_i||view_i\}_{i\in I}}$$

Verifier

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

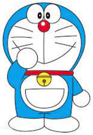
[IKOS07]: Zero-knowledge from secure multiparty computation

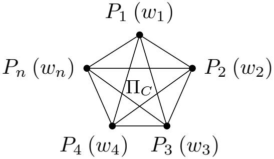
$$C(w) = y$$

MPC-in-the-Head

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

Prover





$P_i : w_i || view_i$

3. Commit to the views:

$w_1 \vert\vert view_1$	$w_2 \vert\vert view_2$		$w_n \vert\vert view_n$
c_1	c_2	$\cdots$	c_n

Verifier

$$\xrightarrow{c_1, \ldots, c_n}$$

$$\xleftarrow{I \subset [n]}$$

$$\xrightarrow{\{w_i || view_i\}_{i \in I}}$$

✓accept iff:

$\{c_i\}_{i \in I}$ opened correctly

output=1 & consistent

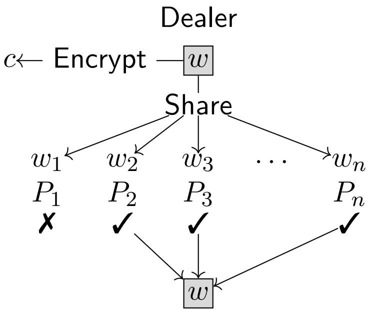
MPC-in-the-head Revisit

MPC-in-the-head is a Σ-pattern protocol for arithmetic statements!

- Algebraic statements are arguably simpler than arithmetic statements.
- When scaling down to algebraic statements, we may start from a lite machinery than MPC — Verifiable Secret Sharing (VSS)

Non-Interactive VSS

[Fel87]: A Practical Scheme for Non-interactive Verifiable Secret Sharing



Sharing Phase

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

  number of participants n

  privacy threshold t_p

  fault-tolerance threshold t_f
- Encrypt$(w) \to c$
- Share$(w) \to (w_1, \ldots, w_n)$

Reconstruct Phase

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

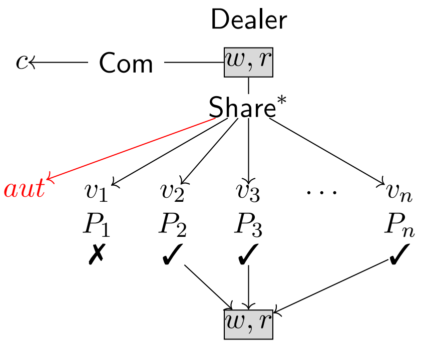
- **Acceptance:** valid shares $w_i \Rightarrow$ Check$(i, w_i, c) = 1$
- t_p**-Privacy:** # [shares] $\leq t_p \Rightarrow$ leak nothing about w
- **Consistency:** # [valid shares] $\geq t_f \Rightarrow$ unique w and recover w

Blind Spot: The Darkest Place Is Under The Candlestick

Over roughly 40 years, there is no well-established yet handy-to-use definition for non-interactive VSS.

In cryptography, definition is of uttermost importance.

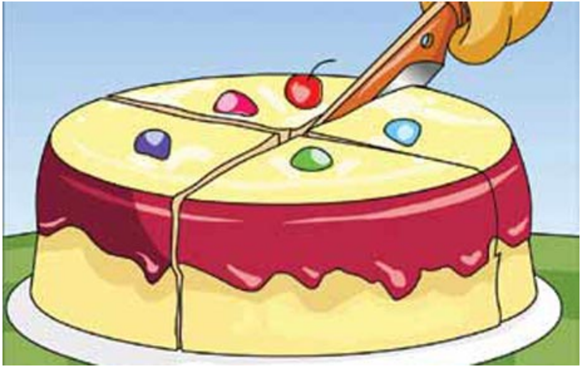
A Refined Definition of Non-Interactive VSS



- 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 method)
- 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 other than c
- t_f-**Correctness:** #[valid shares] $\geq t_f \Rightarrow$ recover $(w, r) \wedge$ Com$(w; r) = c$

A Metaphor of Authenticated Information



aut could right be interpreted as a commitment of sharing method

Dissection of Share*

Conventionally, sharing algorithm outputs all shares $(v_1, \ldots, v_n)$ in one shot, where n is the maximum number of possible participants.

Such syntax is fine when n is poly in λ. But, it is problematic when n is super-poly in λ. To fix this issue, we further dissect Share*

- ShareinMind(s, r): output compact description of sharing method sharedesc and the associated authentication information, both sizes are poly-bounded.
- Distribute$(s, r,$ sharedesc$, i)$: generate v_i for P_i on-the-fly.

Example 1

Shamir's Secret-Sharing via Polynomial Interpolation

- long sharedesc: $v_1, \ldots, v_n$
- compact sharedesc: $a_1, \ldots, a_t$

Differences in Definition: Syntax

1. In our definition the secret s is committed rather than being encrypted
   - Make our definition more general

2. In Feldman's definition Share only outputs the shares, while in our definition Share additionally outputs authentication information aut
   - aut is crucial for participants to check the validity of their shares

3. In Feldman's definition Recover only outputs the secret s, while in our definition Recover output the opening of a commitment, i.e., the secret s and the randomness r (if there is any).
   - This modification is crucial for our Sigma's framework.

Differences in Definition: Security

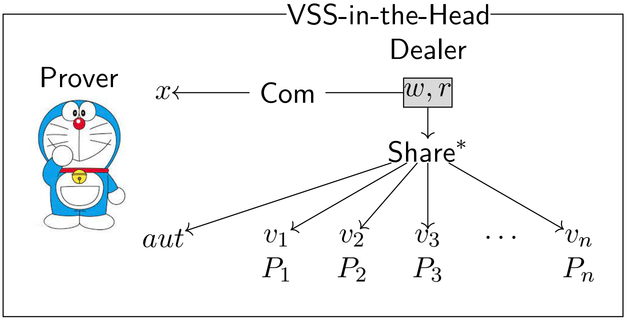
1. For correctness, our definition does not stipulate that the secrets recovered by different groups of participants are consistent as in Feldman's definition. Instead, it requires that the recovered secrets and randomness (if there is any) must be valid opening of c.
   - This requirement is in fact has been met by many existing VSS schemes (such as the Feldman's [Fel87] and Pedersen's VSS schemes [Ped91]), but it has never been formally defined.

2. For privacy, our definition is simulation-based rather than a game-based one as in Feldman's definition.
   - Such adoption aligns our definition with ZKP and MPC. In particular, the simulator Sim is given c as an auxiliary input, allowing the use of commitment schemes satisfying merely one-way hiding property.

Sigma Protocols from VSS

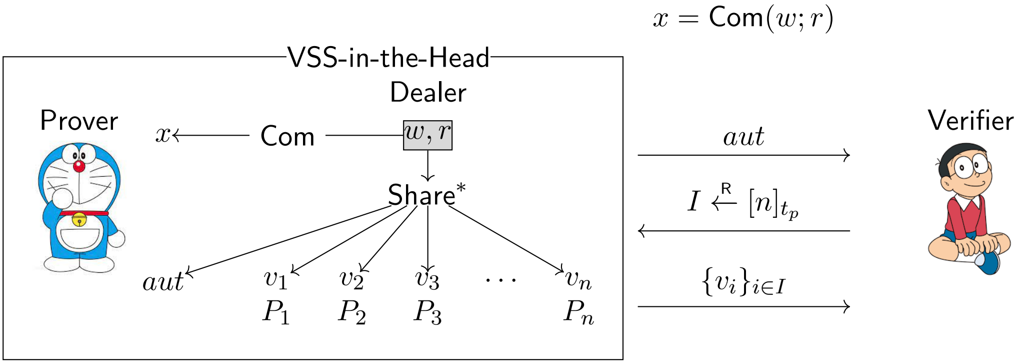
To prove knowledge of opening $x = \mathsf{Com}(w; r)$, we start from a (n, t_p, t_f)-VSS w.r.t. the same Com

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



Sigma Protocols from VSS

To prove knowledge of opening $x = \mathsf{Com}(w; r)$, we start from a (n, t_p, t_f)-VSS w.r.t. the same Com



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

VSS-in-the-Head

Prover

Dealer

$x \leftarrow \mathsf{Com} \leftarrow \boxed{w, r}$

Share^*

aut v_1 v_2 v_3 $\cdots$ v_n

P_1 P_2 P_3 P_n

Verifier

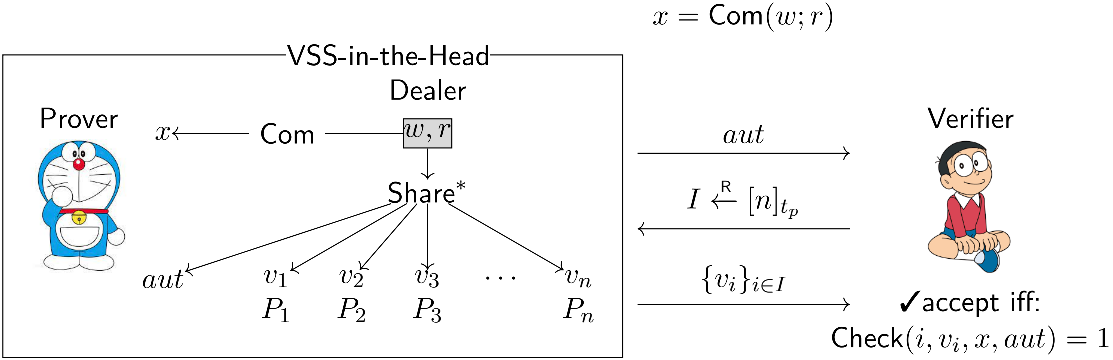
$\xrightarrow{aut}$

$\xleftarrow{I \overset{\mathsf{R}}{\leftarrow} [n]_{t_p}}$

$\xrightarrow{\{v_i\}_{i \in I}}$

Sigma Protocols from VSS

To prove knowledge of opening $x = \mathsf{Com}(w; r)$, we start from a (n, t_p, t_f)-VSS w.r.t. the same Com



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

VSS-in-the-Head

Prover

Dealer

$x \leftarrow \mathsf{Com} \leftarrow \boxed{w, r}$

Share^*

aut v_1 v_2 v_3 $\cdots$ v_n

P_1 P_2 P_3 P_n

$\xrightarrow{aut}$

$\xleftarrow{I \overset{\mathsf{R}}{\leftarrow} [n]_{t_p}}$

$\xrightarrow{\{v_i\}_{i \in I}}$

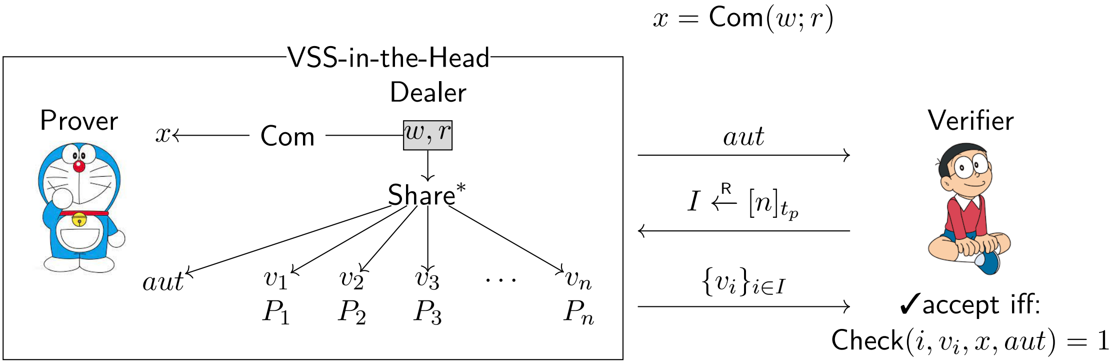
Verifier

✓accept iff:

$\mathsf{Check}(i, v_i, x, aut) = 1$

Sigma Protocols from VSS

To prove knowledge of opening $x = \mathsf{Com}(w; r)$, we start from a (n, t_p, t_f)-VSS w.r.t. the same Com

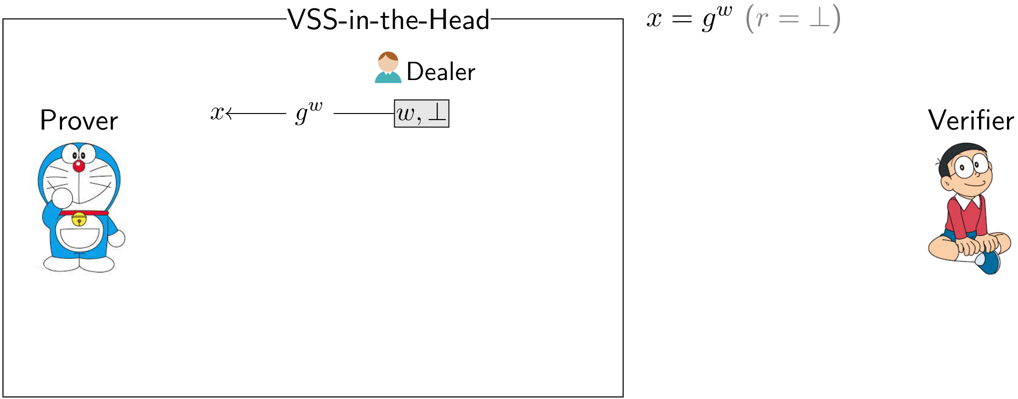


- Completeness $\Leftarrow$ VSS Acceptance
- Special soundness $\Leftarrow$ VSS t_f-Correctness
- SHVZK $\Leftarrow$ VSS t_p-Privacy

Instantiation I: The Schnorr Protocol

Feldman's VSS scheme [Fel87]

- #[participants] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$



VSS-in-the-Head

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

Dealer

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

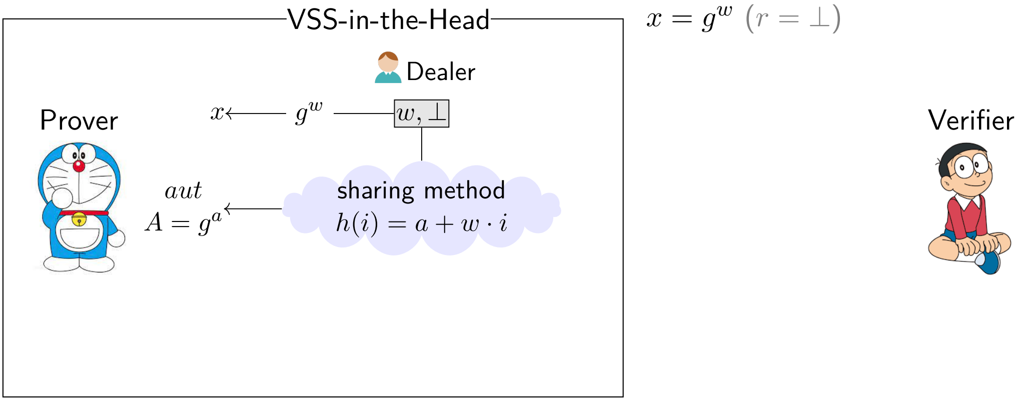
Prover

Verifier

Instantiation I: The Schnorr Protocol

Feldman's VSS scheme [Fel87]

- #[participants] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$



VSS-in-the-Head

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

Dealer

Prover

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

sharing method

$h(i) = a + w \cdot i$

aut

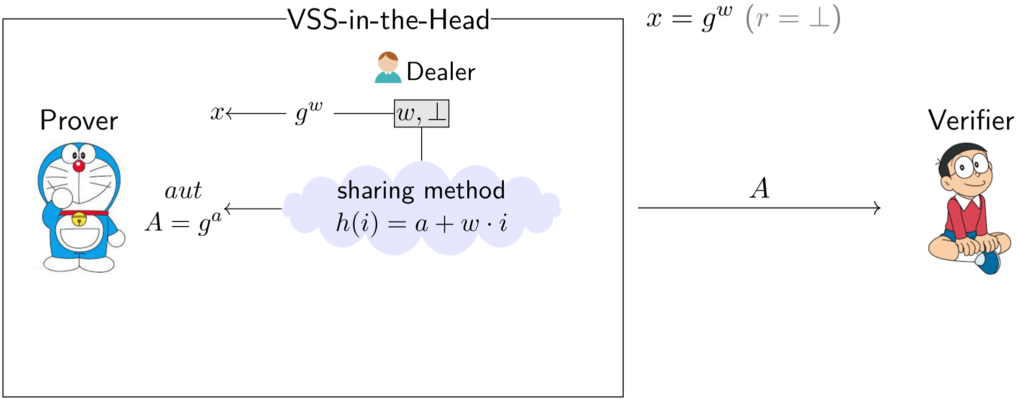
$A = g^a \leftarrow$

Verifier

Instantiation I: The Schnorr Protocol

Feldman's VSS scheme [Fel87]

- #[participants] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$



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

VSS-in-the-Head

Dealer

Prover

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

sharing method

$h(i) = a + w \cdot i$

aut

$A = g^a$

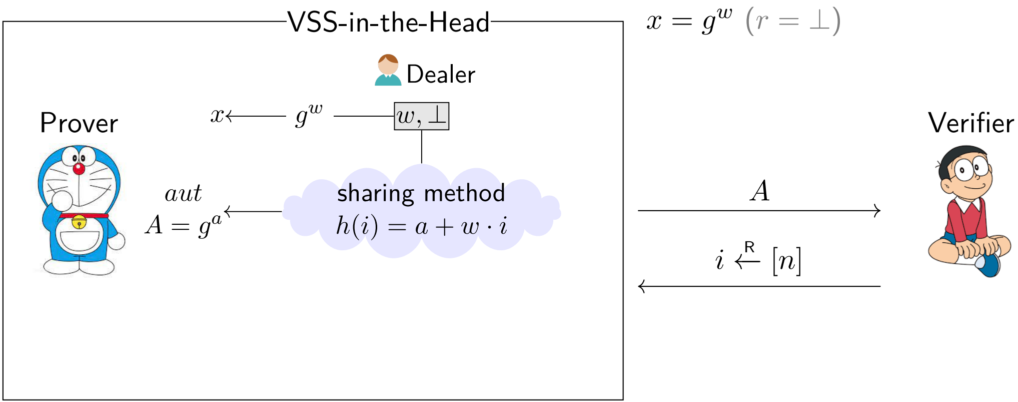
$\xrightarrow{A}$

Verifier

Instantiation I: The Schnorr Protocol

Feldman's VSS scheme [Fel87]

- $\#[\text{participants}] = n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$



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

VSS-in-the-Head

Dealer

Prover

Verifier

$x \leftarrow g^w$ — $\boxed{w, \bot}$

sharing method

$h(i) = a + w \cdot i$

aut

$A = g^a$

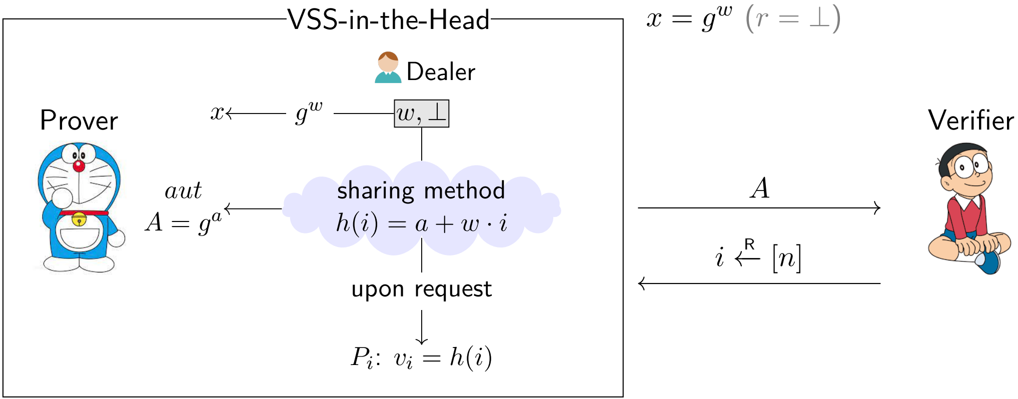
A

$i \overset{R}{\leftarrow} [n]$

Instantiation I: The Schnorr Protocol

Feldman's VSS scheme [Fel87]

- #[participants] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$



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

VSS-in-the-Head

Dealer

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

Prover

aut

$A = g^a \leftarrow$

sharing method

$h(i) = a + w \cdot i$

upon request

$P_i\colon\ v_i = h(i)$

Verifier

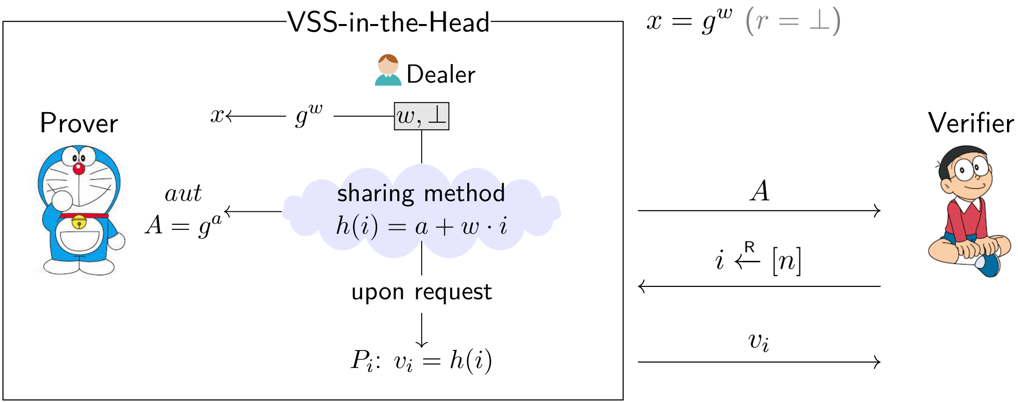
A

$i \xleftarrow{R} [n]$

Instantiation I: The Schnorr Protocol

Feldman's VSS scheme [Fel87]

- $\#[\text{participants}] = n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$



VSS-in-the-Head

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

Dealer

Prover

Verifier

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

aut

$A = g^a \leftarrow$

sharing method

$h(i) = a + w \cdot i$

upon request

P_i: $v_i = h(i)$

$\xrightarrow{A}$

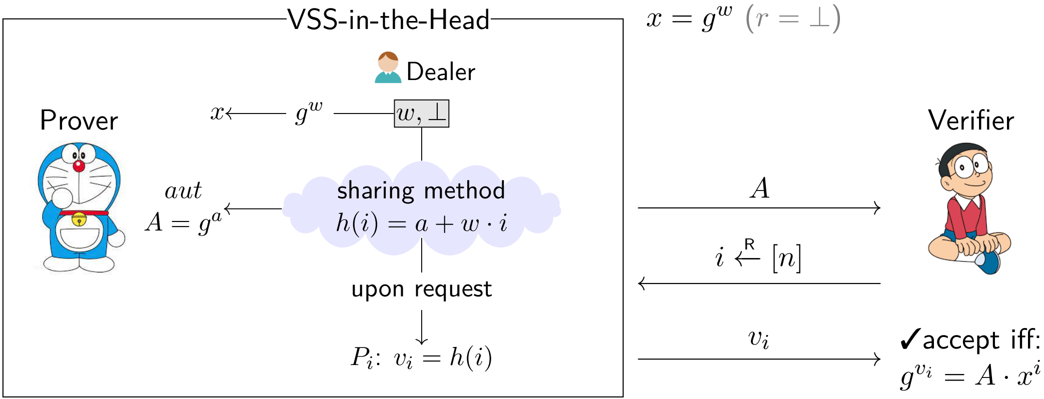
$i \xleftarrow{R} [n]$

$\xrightarrow{v_i}$

Instantiation I: The Schnorr Protocol

Feldman's VSS scheme [Fel87]

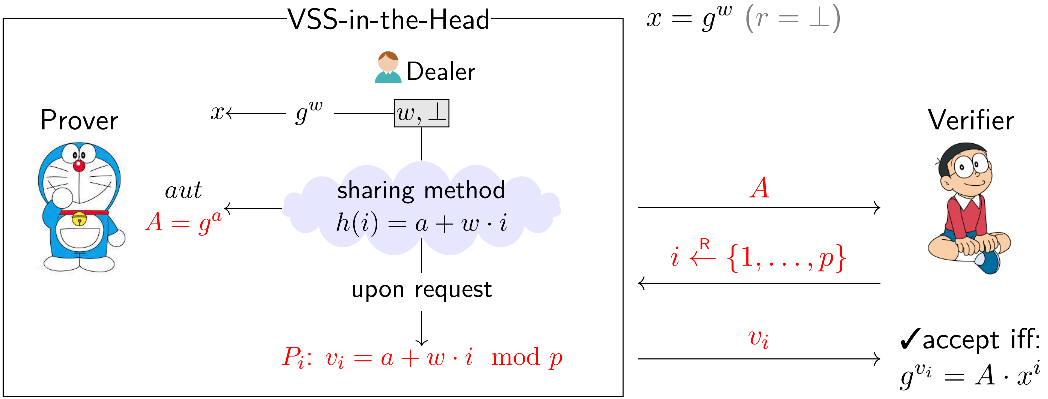
- $\#[\text{participants}] = n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$



Instantiation I: The Schnorr Protocol

Feldman's VSS scheme [Fel87]

- #[participants] $= n$, privacy threshold $t_p = 1$, fault-tolerance threshold $t_f = 2$



VSS-in-the-Head

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

Dealer

Prover

Verifier

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

sharing method

$h(i) = a + w \cdot i$

aut

$A = g^a$

upon request

P_i: $v_i = a + w \cdot i \mod p$

$\xrightarrow{A}$

$i \overset{R}{\leftarrow} \{1, \ldots, p\}$

$\xrightarrow{v_i}$

✓accept iff:

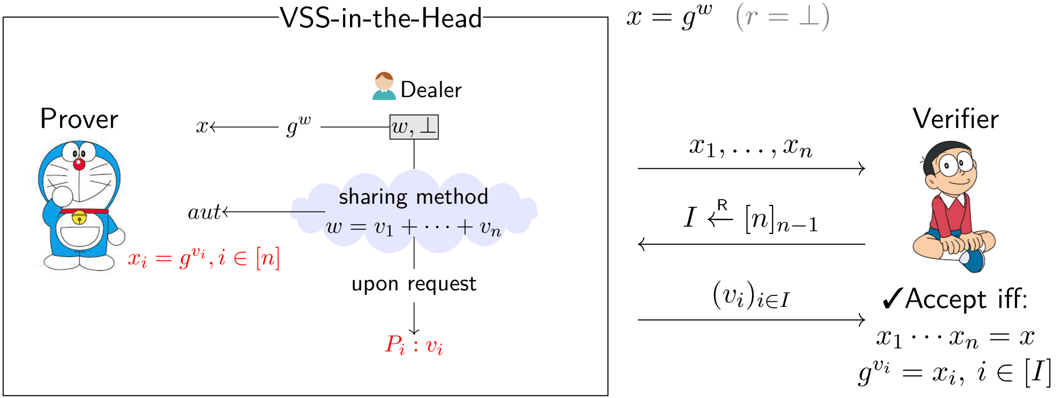
$g^{v_i} = A \cdot x^i$

Set $n = p \Rightarrow$ Schnorr protocol [Sch91]

Instantiation II: A New Sigma Protocol for DL

Additive VSS scheme

- $\#$[participants] $= n$, privacy threshold $t_p = n - 1$, fault-tolerance threshold $t_f = n$



VSS-in-the-Head

Dealer

Prover

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

sharing method

$w = v_1 + \cdots + v_n$

$aut \leftarrow$

$x_i = g^{v_i}, i \in [n]$

upon request

$P_i : v_i$

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

Verifier

$x_1, \ldots, x_n \longrightarrow$

$I \overset{\text{R}}{\leftarrow} [n]_{n-1}$

$(v_i)_{i \in I} \longrightarrow$

✓Accept iff:

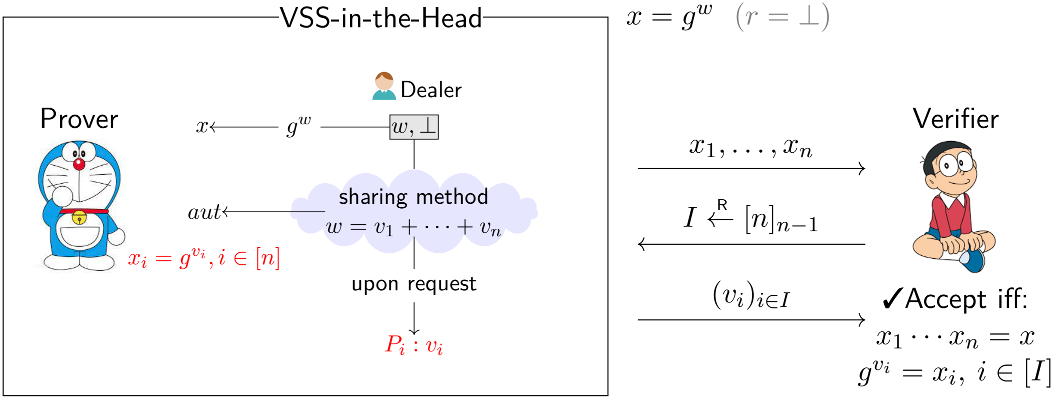
$x_1 \cdots x_n = x$

$g^{v_i} = x_i, \; i \in [I]$

Instantiation II: A New Sigma Protocol for DL

Additive VSS scheme

- $\#[\text{participants}] = n$, privacy threshold $t_p = n - 1$, fault-tolerance threshold $t_f = n$



VSS-in-the-Head

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

Dealer

Prover

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

sharing method

$w = v_1 + \cdots + v_n$

$aut \leftarrow$

$x_i = g^{v_i}, i \in [n]$

upon request

$P_i : v_i$

Verifier

$\xrightarrow{x_1, \ldots, x_n}$

$\xleftarrow{I \overset{R}{\leftarrow} [n]_{n-1}}$

$\xrightarrow{(v_i)_{i \in I}}$

✓Accept iff:

$x_1 \cdots x_n = x$

$g^{v_i} = x_i, \; i \in [I]$

Yield a new Sigma protocol for DL with 2-special soundness.

Outline

1. Background

2. Sigma Protocols from VSS-in-the-Head

3. Applications of VSS-in-the-Head

4. Summary

Forms of Statements in Zero-knowledge Proofs (ZKPs)

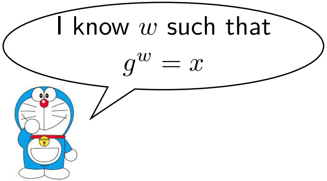
Algebraic Statements

functions over some groups

$\Uparrow$

Sigma (Σ) protocols

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



Forms of Statements in Zero-knowledge Proofs (ZKPs)

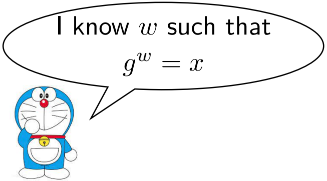
Algebraic Statements

functions over some groups

⇑

Sigma (Σ) protocols

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



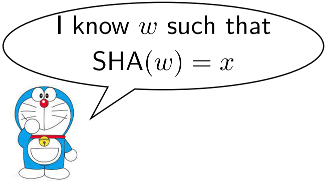
Non-Algebraic Statements

boolean/arithmetic circuits

⇑

General-purpose ZKPs

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



Composite Statements

Algebraic Statements + Non-Algebraic Statements

e.g. $g^{w_1} = x$

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

$\|$

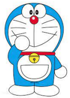
combine in arbitrary ways

e.g. $w_1 = w_2$

$\Downarrow$

Composite Statements

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



Composite Statements

Algebraic Statements + Non-Algebraic Statements

e.g. $g^{w_1} = x$

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

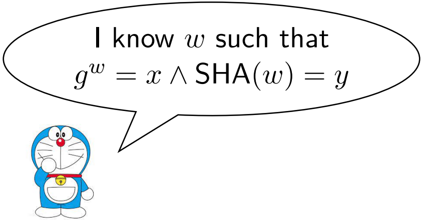
$||$

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:

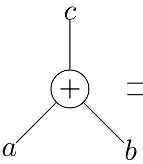
I know w such that

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

ZKPs for Composite Statements

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

circuits $\Rightarrow$ algebraic constraints

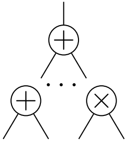


$$\Longrightarrow \quad g^a \cdot g^b = g^c$$

# [public-key ops] and # [group elements] linear to the circuit size

algebraic constraints $\Rightarrow$ circuits

$$g^w = x \quad \Longrightarrow$$



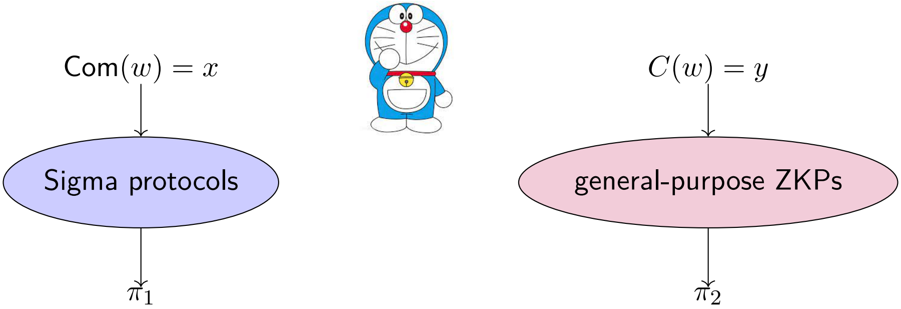
size of the statements dramatically increases [1]

☹ Both directions incur significant overhead.

[1] 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:



$\mathrm{Com}(w) = x$

Sigma protocols

π_1

$C(w) = y$

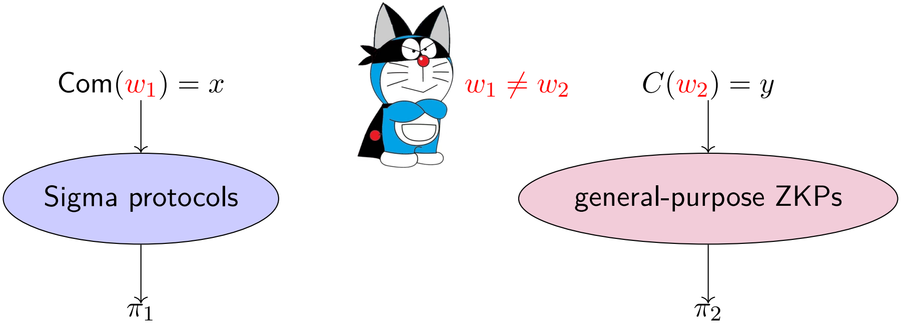
general-purpose ZKPs

π_2

Take advantages of both Sigma protocols and general-purpose ZKPs

ZKPs for Commit-and-Prove Type Composite Statements

- A better method:



$\mathsf{Com}(w_1) = x$

Sigma protocols

π_1

$w_1 \neq w_2$

$C(w_2) = y$

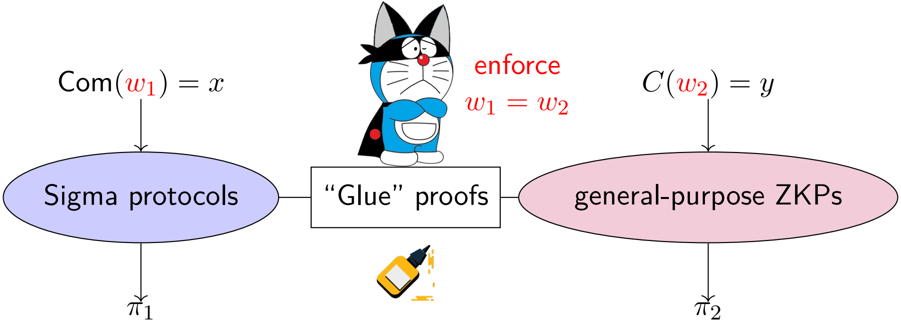
general-purpose ZKPs

π_2

But, a malicious prover can 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]



$\mathsf{Com}(w_1) = x$

enforce $w_1 = w_2$

$C(w_2) = y$

Sigma protocols

“Glue” proofs

general-purpose ZKPs

π_1

π_2

Solution: Enforce the prover to generate π_1 and π_2 using $w_1 = w_2$ via glue proof.

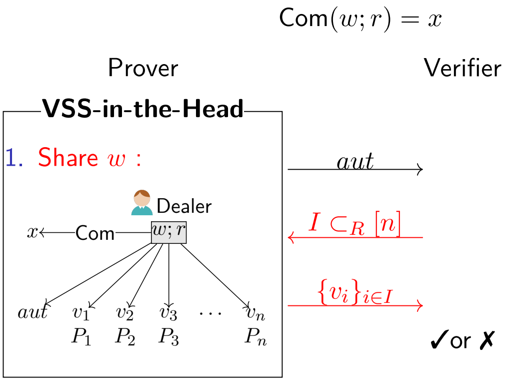
- glue two different worlds $\rightsquigarrow$ additional overheads in computation and proof size
- must be tailored to align with general-purpose ZKPs $\rightsquigarrow$ require extra design efforts

Whether the seemingly indispensable “glue” proofs are necessary?

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

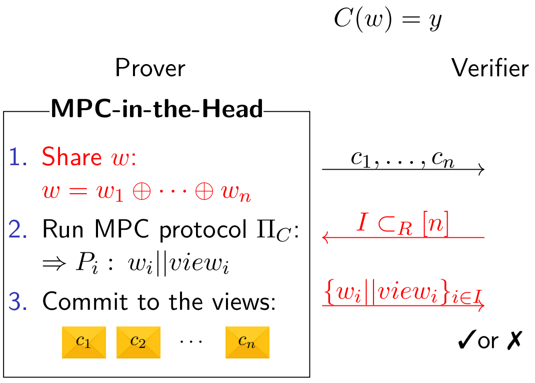
1. Share w :

$\xrightarrow{aut}$

$\xleftarrow{I \subset_R [n]}$

$\xrightarrow{\{v_i\}_{i \in I}}$

✓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:

$\xrightarrow{c_1, \ldots, c_n}$

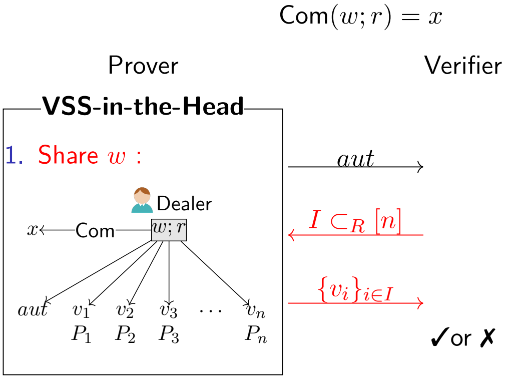
$\xleftarrow{I \subset_R [n]}$

$\xrightarrow{\{w_i || view_i\}_{i \in I}}$

✓or ✗

Share the same Σ pattern & same secret sharing procedure!

Main Observation



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

Prover | Verifier

VSS-in-the-Head

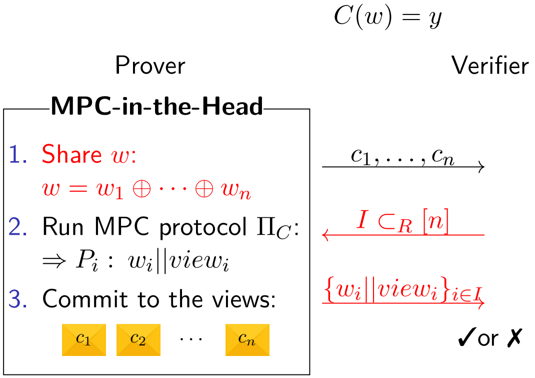
1. Share w :

$\xrightarrow{aut}$

$\xleftarrow{I \subset_R [n]}$

$\xrightarrow{\{v_i\}_{i \in I}}$

✓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:

$\xrightarrow{c_1, \ldots, c_n}$

$\xleftarrow{I \subset_R [n]}$

$\xrightarrow{\{w_i || view_i\}_{i \in I}}$

✓or ✗

💡 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 $w = w_1 \oplus \cdots \oplus w_n$, which is a special case of $(n, n-1, n)$-SS scheme) $\rightsquigarrow$ make it incompatible with general (n, t_p, t_f)-VSS schemes

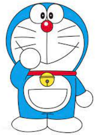
2. The relationship between VSS and SS is unclear $\rightsquigarrow$ make it difficult to reuse the common part of witness sharing procedure

A Generalization of MPC-in-the-Head

$C(w) = y$

MPC-in-the-Head

Prover



1. Share w :

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

   $(n, n-1, n)$-SS scheme

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

   (n, t_p, t_f)-SS 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

$$\xrightarrow{c_1, \ldots, c_n}$$

$$\xleftarrow{I \subset_R [n]}$$

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

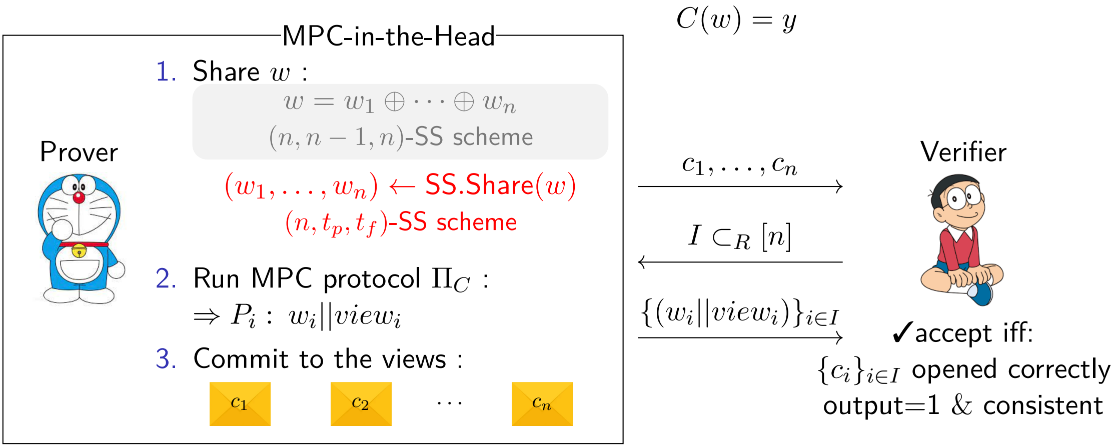
Verifier

✓accept iff:

$\{c_i\}_{i \in I}$ opened correctly

output=1 & consistent

A Generalization of MPC-in-the-Head



$C(w) = y$

MPC-in-the-Head

Prover

1. Share w :

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

   $(n, n-1, n)$-SS scheme

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

   (n, t_p, t_f)-SS scheme

2. Run MPC protocol Π_C :

   $\Rightarrow P_i : \ w_i || view_i$

3. Commit to the views :

   $c_1 \quad c_2 \quad \cdots \quad c_n$

$\xrightarrow{c_1, \ldots, c_n}$

$\xleftarrow{I \subset_R [n]}$

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

Verifier

✓accept iff:

$\{c_i\}_{i \in I}$ opened correctly

output$=1$ & consistent

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

Separable VSS: Clear Relationship between VSS and SS

Definition 2 (Separability)

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

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

Separable VSS: Clear Relationship between VSS and SS

Definition 2 (Separability)

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

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

VSS.Share$^*(w, r)$

Separable VSS: Clear Relationship between VSS and SS

Definition 2 (Separability)

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

$$\begin{aligned}
&\{w_i\}_{i\in[n]} \leftarrow \text{SS.Share}(w)\\
&\{r_i\}_{i\in[n]} \leftarrow \text{SS.Share}(r)\\
&aut \leftarrow \text{AutGen}(\{(w_i, r_i)\}_{i\in[n]})
\end{aligned}$$

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

Separable VSS: Clear Relationship between VSS and SS

Definition 2 (Separability)

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

$$\begin{aligned}
\{w_i\}_{i\in[n]} &\leftarrow \textsf{SS.Share}(w) \\
\{r_i\}_{i\in[n]} &\leftarrow \textsf{SS.Share}(r) \\
aut &\leftarrow \textsf{AutGen}(\{(w_i, r_i)\}_{i\in[n]})
\end{aligned}$$

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

Separable VSS: Clear Relationship between VSS and SS

Definition 2 (Separability)

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

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

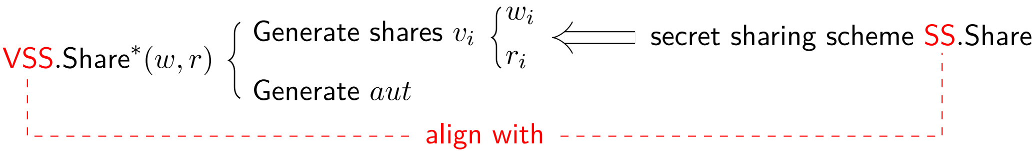
$$\mathsf{VSS.Share}^*(w, r) \begin{cases} \text{Generate shares } v_i \begin{cases} w_i \\ r_i \end{cases} \Longleftarrow \text{secret sharing scheme } \mathsf{SS.Share} \\ \text{Generate } aut \end{cases}$$

Separable VSS: Clear Relationship between VSS and SS

Definition 2 (Separability)

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

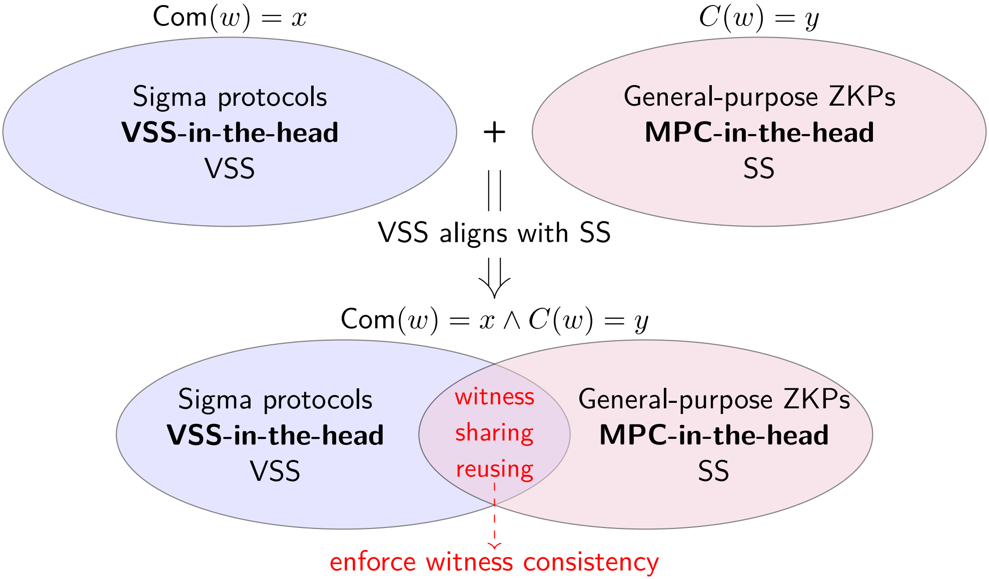
$$\begin{aligned}\{w_i\}_{i\in[n]} &\leftarrow \mathsf{SS.Share}(w)\\ \{r_i\}_{i\in[n]} &\leftarrow \mathsf{SS.Share}(r)\\ aut &\leftarrow \mathsf{AutGen}(\{(w_i, r_i)\}_{i\in[n]})\end{aligned}$$



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

align with

Combination of Two Worlds



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

Sigma protocols
VSS-in-the-head
VSS

$+$

$C(w) = y$

General-purpose ZKPs
MPC-in-the-head
SS

$\|$

VSS aligns with SS

$\Downarrow$

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

Sigma protocols
VSS-in-the-head
VSS

witness
sharing
reusing

General-purpose ZKPs
MPC-in-the-head
SS

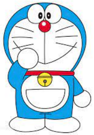
enforce witness consistency

A Generic Construction of ZKPs for Composite Statements (commit-and-prove type)

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

(VSS+MPC)-in-the-Head

Prover



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

$$\xrightarrow{c_1, \dots, c_n, aut}$$

$$\xleftarrow{\boxed{I \subset_R [n]}}$$

$$\xrightarrow{\{\boxed{w_i} || view_i, r_i\}_{i \in I}}$$

Verifier

Accept iff:

MPC-in-the-head check ✓

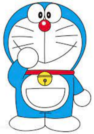
VSS-in-the-head check ✓

A Generic Construction of ZKPs for Composite Statements (commit-and-prove type)

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

(VSS+MPC)-in-the-Head

Prover



1. Share w, r using VSS.Share*:

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

   $(r_1, \dots, r_n) \leftarrow \mathsf{SS.Share}(r)$

   $aut \leftarrow \mathsf{AutGen}(\{w_i, r_i\}_{i \in [n]})$

2. Run MPC protocol Π_C :

   $\Rightarrow P_i : \ w_i || view_i$

3. Commit to the views :

   c_1 c_2 $\cdots$ c_n

$$\xrightarrow{c_1, \dots, c_n, aut}$$

$$\xleftarrow{\boxed{I \subset_R [n]}}$$

$$\xrightarrow{\{\boxed{w_i}||view_i, r_i\}_{i \in I}}$$

Verifier

Accept iff:

MPC-in-the-head check ✓

VSS-in-the-head check ✓

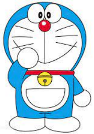
- Completeness ⇐ VSS separability+(VSS/MPC)-in-the-head completeness
- Special soundness ⇐ witness sharing reusing+(VSS/MPC)-in-the-head special soundness
- SHVZK ⇐ (VSS/MPC)-in-the-head SHVZK

A Generic Construction of ZKPs for Composite Statements (commit-and-prove type)

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

(VSS+MPC)-in-the-Head

Prover



1. Share w, r using VSS.Share*:

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

   $(r_1, \ldots, r_n) \leftarrow \mathsf{SS.Share}(r)$

   $aut \leftarrow \mathsf{AutGen}(\{w_i, r_i\}_{i\in[n]})$

2. Run MPC protocol Π_C :

   $\Rightarrow P_i : \; w_i || view_i$

3. Commit to the views :

   c_1 c_2 $\cdots$ c_n

$\xrightarrow{c_1, \ldots, c_n, aut}$

$\xleftarrow{\boxed{I \subset_R [n]}}$

$\xrightarrow{\{\boxed{w_i}||view_i, r_i\}_{i\in I}}$

Verifier

Accept iff:

MPC-in-the-head check ✓

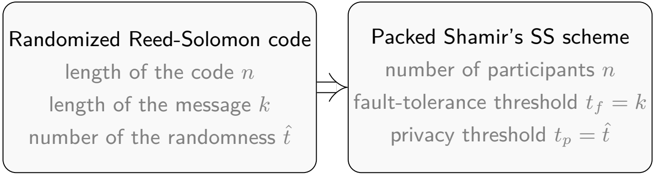
VSS-in-the-head check ✓

☺

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

length of the code n

length of the message k

number of the randomness $\hat{t}$

$\Rightarrow$

Packed Shamir's SS scheme

number of participants n

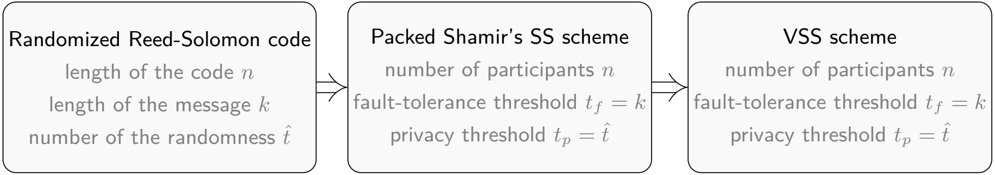
fault-tolerance threshold $t_f = k$

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 $\hat{t}$

Packed Shamir's SS scheme

number of participants n

fault-tolerance threshold $t_f = k$

privacy threshold $t_p = \hat{t}$

VSS scheme

number of participants n

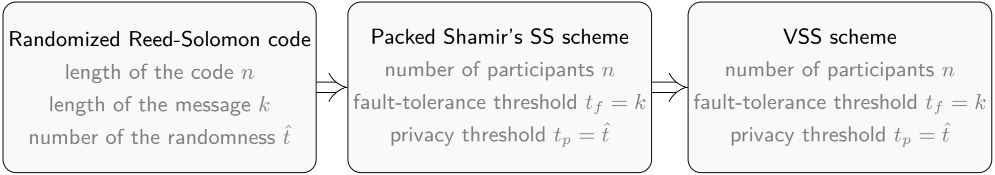
fault-tolerance threshold $t_f = k$

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

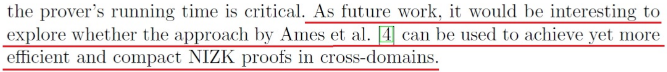


Randomized Reed-Solomon code
length of the code n
length of the message k
number of the randomness $\hat{t}$

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

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

1. Background

2. Sigma Protocols from VSS-in-the-Head

3. Applications of VSS-in-the-Head

4. Summary

Summary

A framework of Sigma protocols for algebraic statements: VSS-in-the-head paradigm

Establish an unexpceted connection between VSS and Sigma protocols

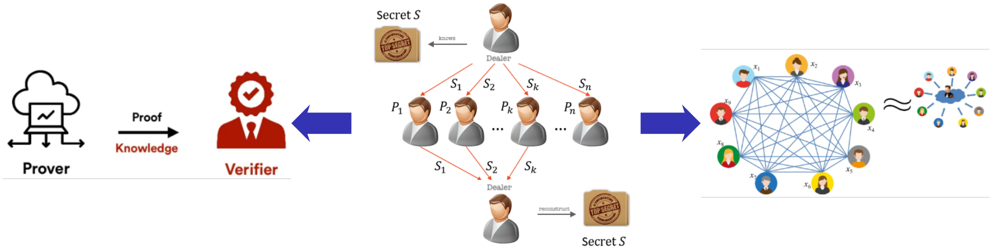
- Give a vivid and refined definition of VSS
- Capture the essence of Sigma protocols
- Neatly explain classic Sigma protocols [Sch91, GQ88, Oka92]
- Give an automatic way to construct Sigma protocols

A generic ZKP construction for composite statements (commit-and-prove type)

- Combine the best of two worlds without glue proofs
- Give an efficient instantiation from Ligero++

Take Away

Secret Sharing is the common theme underlying both ZKP and MPC



Thanks for Your Attention!

Any Questions?

Reference I

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 *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography*, volume 13177 of *Lecture Notes in Computer Science*, pages 584–614. Springer, 2022.

Thomas Attema, Ronald Cramer, and Serge Fehr.
Compressing proofs of k-out-of-n partial knowledge.
In *Advances in Cryptology - CRYPTO 2021*, volume 12828 of *Lecture Notes in Computer Science*, pages 65–91. Springer, 2021.

Shashank Agrawal, Chaya Ganesh, and Payman Mohassel.
Non-interactive zero-knowledge proofs for composite statements.
In *Advances in Cryptology - CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*, pages 643–673. Springer, 2018.

Reference II

- Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit.
  Short accountable ring signatures based on DDH.
  In *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security*, volume 9326 of *Lecture Notes in Computer Science*, pages 243–265. Springer, 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 *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography*, volume 11442 of *Lecture Notes in Computer Science*, pages 286–313. Springer, 2019.

Reference III

Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler.
Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs.
In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11692 of *Lecture Notes in Computer Science*, pages 176–202. Springer, 2019.

Fabrice Boudot.
Efficient proofs that a committed number lies in an interval.
In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer, 2000.

Ronald Cramer, Ivan Damgård, and Berry Schoenmakers.
Proofs of partial knowledge and simplified design of witness hiding protocols.
In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

Reference IV

- Matteo Campanelli, Dario Fiore, and Anaïs Querol.
  Legosnark: Modular design and composition of succinct zero-knowledge proofs.
  In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*, pages 2075–2092. ACM, 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 *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference*, volume 9816 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2016.

- Ronald Cramer.
  Modular design of secure yet practical cryptographic protocols.
  PhD thesis, 1996.
  CWI and University of Amsterdam.

Reference V

Paul Feldman.
A practical scheme for non-interactive verifiable secret sharing.
In *28th Annual Symposium on Foundations of Computer Science*, pages 427–437. IEEE Computer Society, 1987.

Amos Fiat and Adi Shamir.
How to prove yourself: practical solutions to identification and signature problems.
In *Advances in Cryptology - CRYPTO 1986*, pages 186–194, 1986.

Jens Groth and Markulf Kohlweiss.
One-out-of-many proofs: Or how to leak a secret and spend a coin.
In *Advances in Cryptology - EUROCRYPT 2015*, pages 253–280, 2015.

Reference VI

- 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 *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security*, volume 3329 of *Lecture Notes in Computer Science*, pages 276–292. Springer, 2004.

- Louis C. Guillou and Jean-Jacques Quisquater.
  A "paradoxical" indentity-based signature scheme resulting from zero-knowledge.
  In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference*, pages 216–231, 1988.

- Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky.
  Efficient arguments without short pcps.
  In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007)*, pages 278–291, 2007.

Reference VII

Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai.
Zero-knowledge from secure multiparty computation.
In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, STOC 2007*, pages 21–30. ACM, 2007.

Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi.
Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently.
In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*, pages 955–966. ACM, 2013.

Joe Kilian.
A note on efficient zero-knowledge proofs and arguments (extended abstract).
In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, STOC 1992*, pages 723–732, 1992.

Reference VIII

Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon.
Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general.
In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference*, volume 13508 of *Lecture Notes in Computer Science*, pages 71–101. Springer, 2022.

Ueli Maurer.
Zero-knowledge proofs of knowledge for group homomorphisms.
Des. Codes Cryptogr., 77(2-3):663–676, 2015.

Tatsuaki Okamoto.
Provably secure and practical identification schemes and corresponding signature schemes.
In *Advances in Cryptology - CRYPTO 1992*, volume 740, pages 31–53. Springer, 1992.

Reference IX

Torben P. Pedersen.
Non-interactive and information-theoretic secure verifiable secret sharing.
In *Advances in Cryptology - CRYPTO 1991*, pages 129–140, 1991.

Claus-Peter Schnorr.
Efficient signature generation by smart cards.
Journal of Cryptology, 4(3):161–174, 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 *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11692 of *Lecture Notes in Computer Science*, pages 147–175. Springer, 2019.