Stacking Sigma

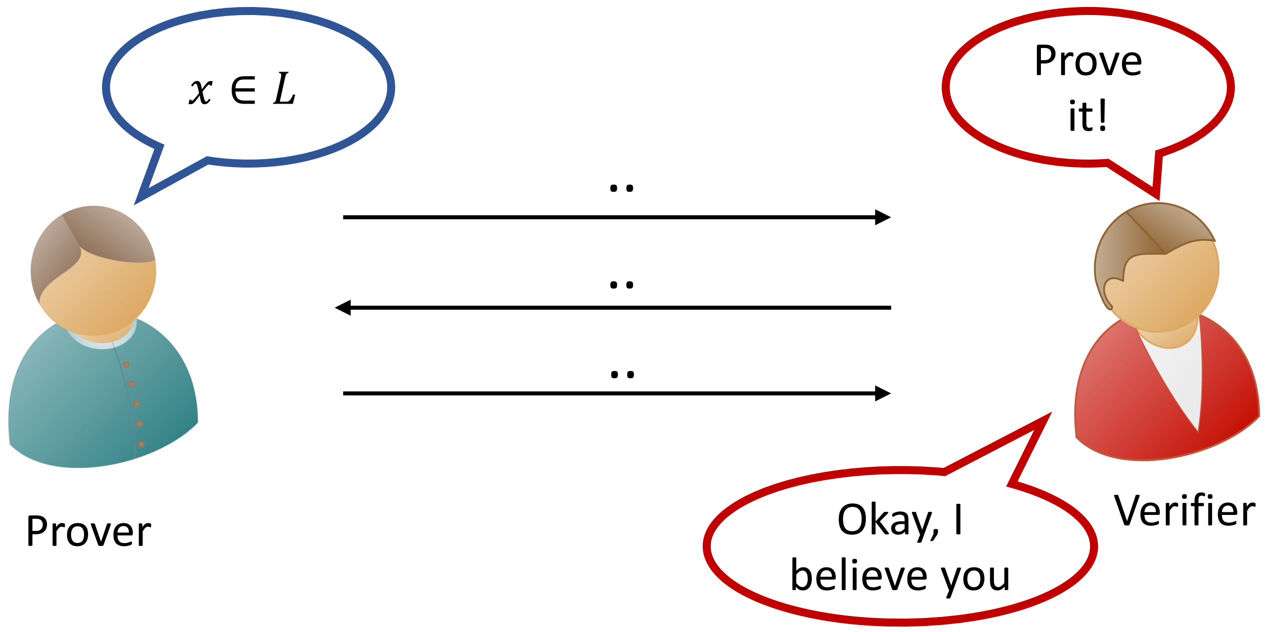
A Framework to Compose Σ–Protocols for Disjunctions

Aarushi Goel Matthew Green Mathias Hall-Andersen Gabriel Kaptchuk

JOHNS HOPKINS UNIVERSITY AARHUS UNIVERSITY BOSTON UNIVERSITY

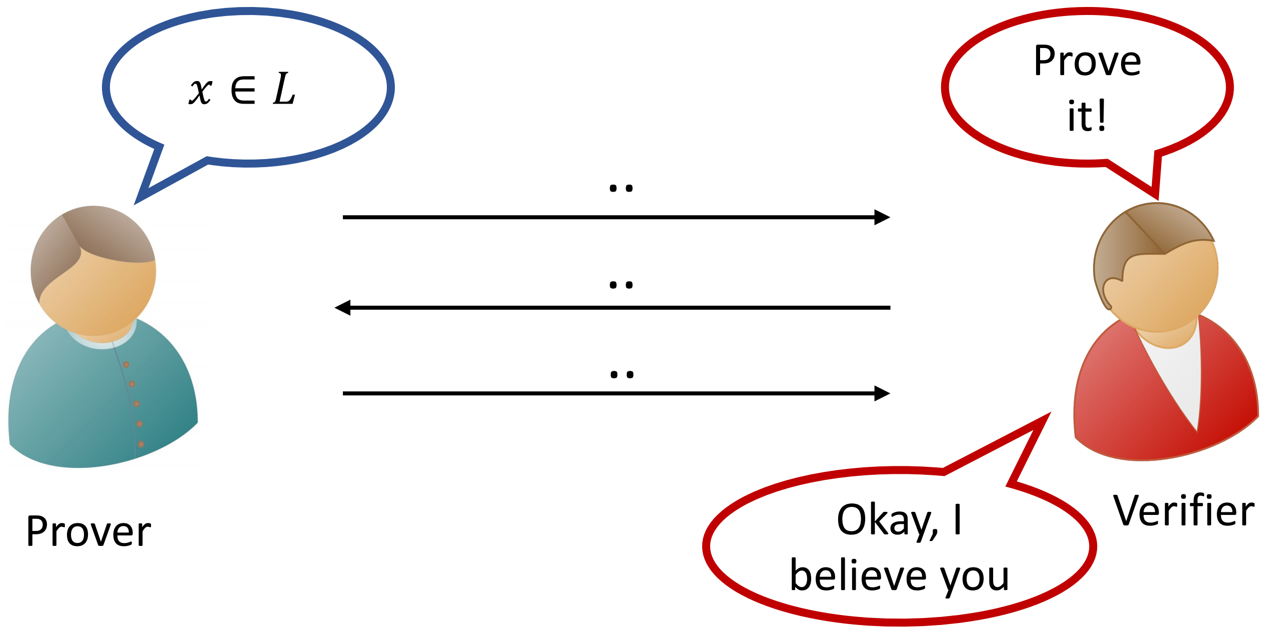
Zero Knowledge Proofs

$\boxed{L \in NP}$



Zero Knowledge Proofs

$\boxed{L \in NP}$

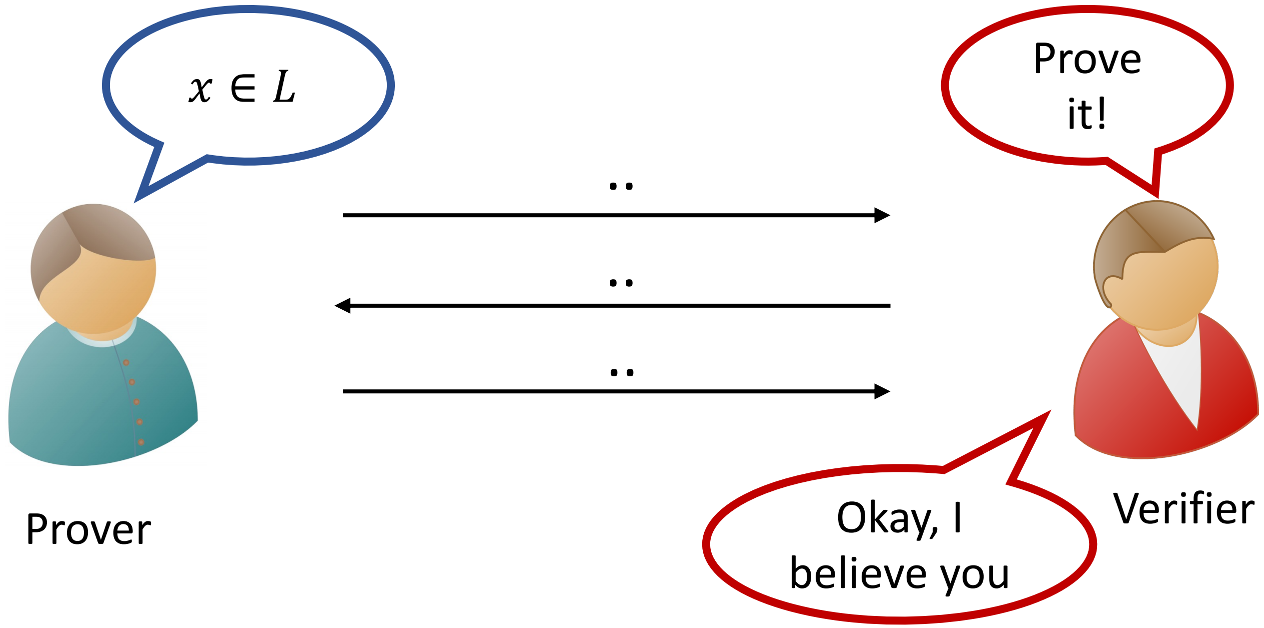


Soundness

Cheating prover should not be able to convince the verifier if $x \notin L$

Zero Knowledge Proofs

$\boxed{L \in NP}$



Soundness

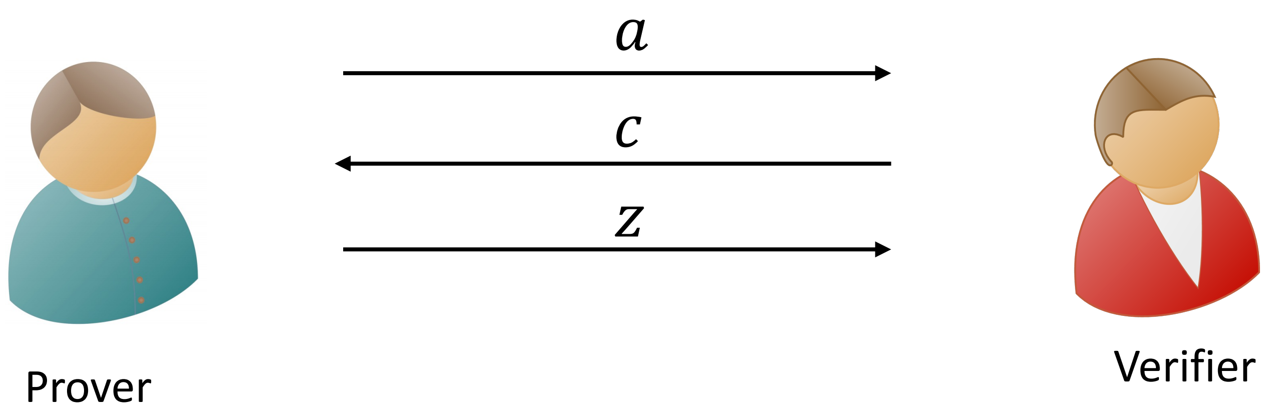
Cheating prover should not be able to convince the verifier if $x \notin L$

Zero knowledge

Verifier should not learn anything other than the validity of the statement

Sigma Protocols

$\boxed{L \in NP}$



Public coin proofs

Honest verifier zero-knowledge

Can be made non-interactive in the random oracle model

Disjunctive Statements: Interesting Class of Languages

$$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad \ldots\ldots \quad or \quad x_n \in L_n$$

Where each $L_i \in NP$

Disjunctive Statements: Interesting Class of Languages

$$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad ..... \quad or \quad x_n \in L_n$$

Where each $L_i \in NP$

Set-Membership Proofs – Ring signatures, confidential transactions

Applications:

Disjunctive Statements: Interesting Class of Languages

$$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad \ldots.. \quad or \quad x_n \in L_n$$

Where each $L_i \in NP$

Applications:

Set-Membership Proofs – Ring signatures, confidential transactions

Proving existence of bugs in codebase

Disjunctive Statements: Interesting Class of Languages

$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad \ldots\ldots \quad or \quad x_n \in L_n$

Where each $L_i \in NP$

Applications:

Set-Membership Proofs – Ring signatures, confidential transactions

Proving existence of bugs in codebase

Proving correct execution of a processor

....

Zero-Knowledge Proofs for Disjunctive Statements

Result	General Compiler	Languages	Proof Size	Prover Time	Non-interactive
Classical [CDS94, AOS02]	For Σ-Protocols	All	Linear in all the branches	Linear in all the branches	Random Oracle Model

Zero-Knowledge Proofs for Disjunctive Statements

Result	General Compiler	Languages	Proof Size	Prover Time	Non-interactive
Classical [CDS94, AOS02]	For Σ-Protocols	All	Linear in all the branches	Linear in all the branches	Random Oracle Model
Stacked Garbling [HK20]	NO	Circuit SAT	Linear in one branch	Linear in all the branches	NO

Zero-Knowledge Proofs for Disjunctive Statements

Result	General Compiler	Languages	Proof Size	Prover Time	Non-interactive
Classical [CDS94, AOS02]	For Σ-Protocols	All	Linear in all the branches	Linear in all the branches	Random Oracle Model
Stacked Garbling [HK20]	NO	Circuit SAT	Linear in one branch	Linear in all the branches	NO
Mac n' Cheese [BMRS20]	For special kind of interactive proofs	Circuit SAT	Linear in one branch	Linear in all the branch	NO

Zero-Knowledge Proofs for Disjunctive Statements

Result	General Compiler	Languages	Proof Size	Prover Time	Non-interactive
Classical [CDS94, AOS02]	For Σ-Protocols	All	Linear in all the branches	Linear in all the branches	Random Oracle Model
Stacked Garbling [HK20]	NO	Circuit SAT	Linear in one branch	Linear in all the branches	NO
Mac n' Cheese [BMRS20]	For special kind of interactive proofs	Circuit SAT	Linear in one branch	Linear in all the branch	NO
Bullet-Proofs/Compressed-Protocols [BCC+16,BBB+18, ACF20..]	NO	Restricted Class	Sublinear	Linear in all the branch	Random Oracle Model

Zero-Knowledge Proofs for Disjunctive Statements

Result	General Compiler	Languages	Proof Size	Prover Time	Non-interactive
Classical [CDS94, AOS02]	For Σ-Protocols	All	Linear in all the branches	Linear in all the branches	Random Oracle Model
Stacked Garbling [HK20]	NO	Circuit SAT	Linear in one branch	Linear in all the branches	NO
Mac n' Cheese [BMRS20]	For special kind of interactive proofs	Circuit SAT	Linear in one branch	Linear in all the branch	NO
Bullet-Proofs/Compressed-Protocols [BCC+16,BBB+18, ACF20..]	NO	Restricted Class	Sublinear	Linear in all the branch	Random Oracle Model
SNARKs		All	Sublinear	Super-linear in all the branches	CRS/Random Oracle Model

Zero-Knowledge Proofs for Disjunctive Statements

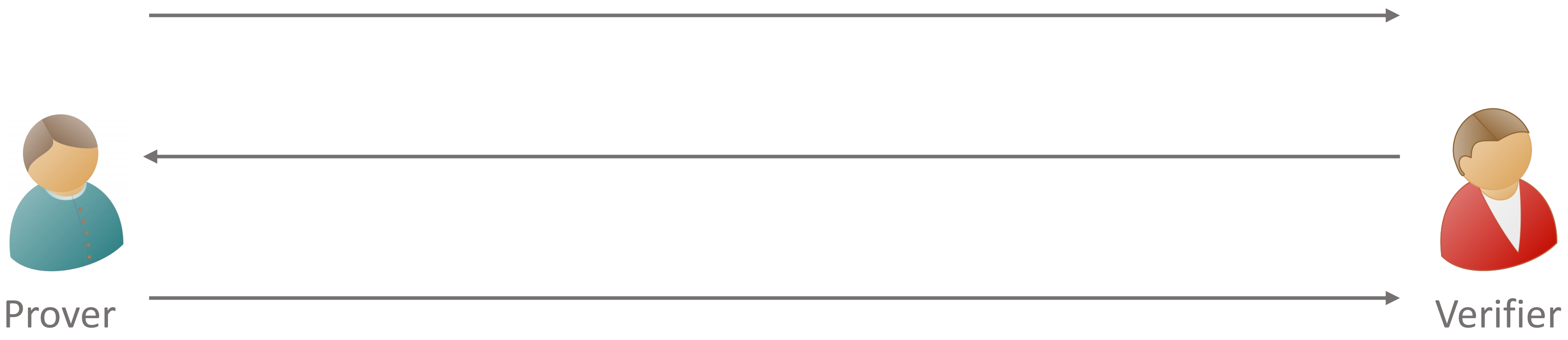
Result	General Compiler	Languages	Proof Size	Prover Time	Non-interactive
Classical [CDS94, AOS02]	For Σ-Protocols	All	Linear in all the branches	Linear in all the branches	Random Oracle Model
Stacked Garbling [HK20]	NO	Circuit SAT	Linear in one branch	Linear in all the branches	NO
Mac n' Cheese [BMRS20]	For special kind of interactive proofs	Circuit SAT	Linear in one branch	Linear in all the branch	NO
Bullet-Proofs/Compressed-Protocols [BCC+16,BBB+18, ACF20..]	NO	Restricted Class	Sublinear	Linear in all the branch	Random Oracle Model
SNARKs		All	Sublinear	Super-linear in all the branches	CRS/Random Oracle Model

Zero-Knowledge Proofs for Disjunctive Statements

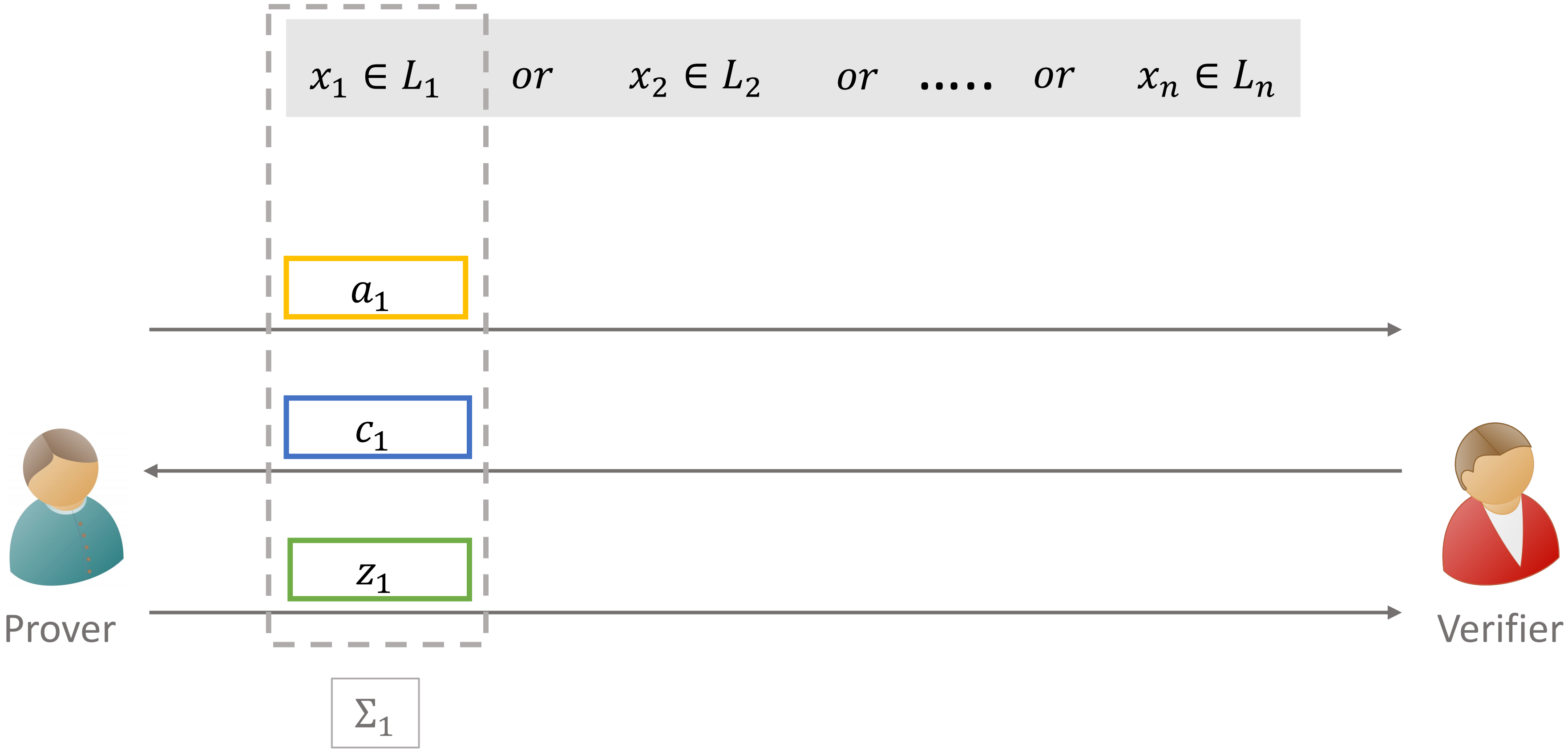
Result	General Compiler	Languages	Proof Size	Prover Time	Non-interactive
Classical [CDS94, AOS02]	For Σ-Protocols	All	Linear in all the branches	Linear in all the branches	Random Oracle Model
Stacked Garbling [HK20]	NO	Circuit SAT	Linear in one branch	Linear in all the branches	NO
Mac n' Cheese [BMRS20]	For special kind of interactive proofs	Circuit SAT	Linear in one branch	Linear in all the branch	NO
Bullet-Proofs/Compressed-Protocols [BCC+16,BBB+18, ACF20..]	NO	Restricted Class	Sublinear	Linear in all the branch	Random Oracle Model
SNARKs		All	Sublinear	Super-linear in all the branches	CRS/Random Oracle Model
Our Work	For Σ-Protocols	All	Linear in one branch	Linear in all the branches	Random Oracle Model

Stacking Σ-Protocols for Disjunctions

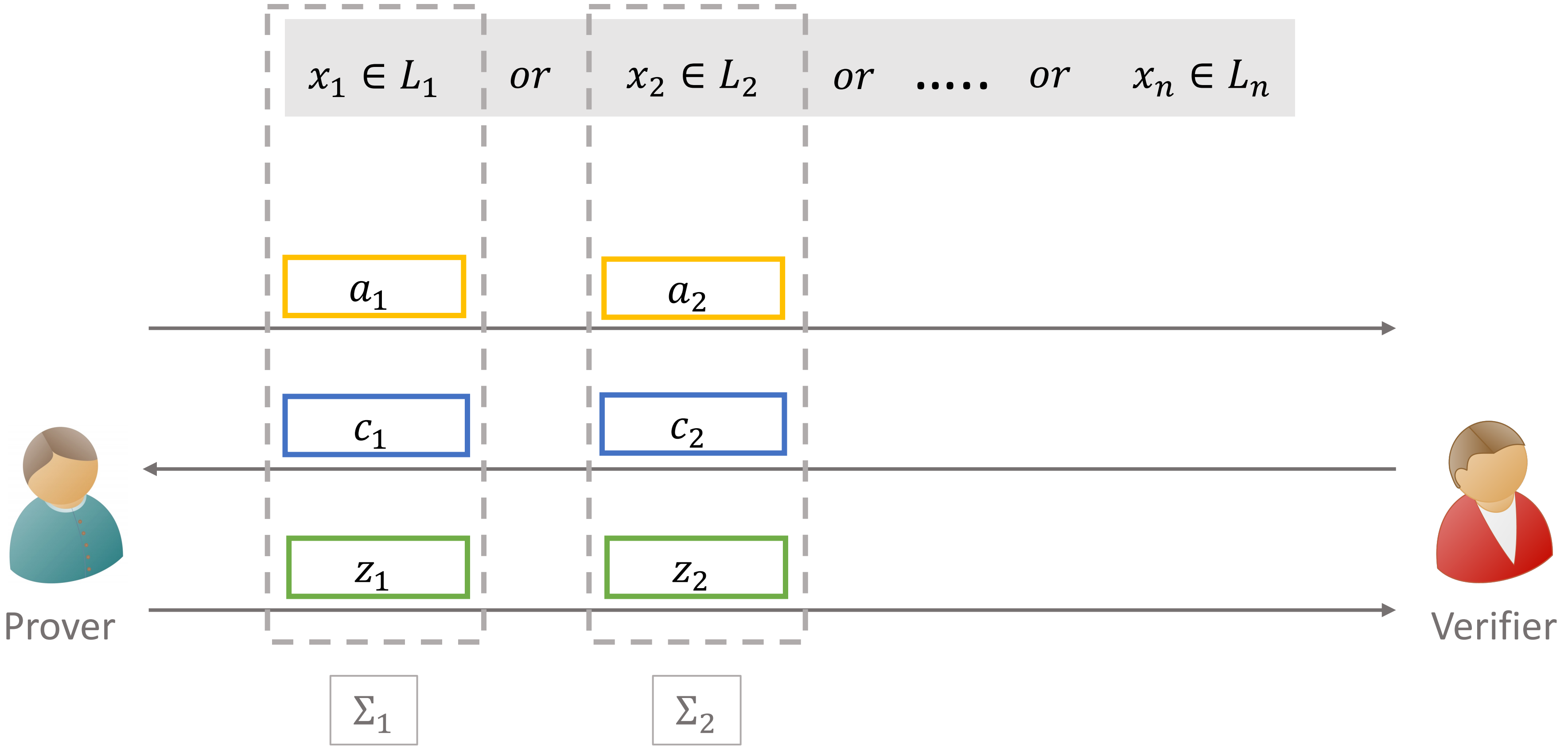
$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad ..... \quad or \quad x_n \in L_n$



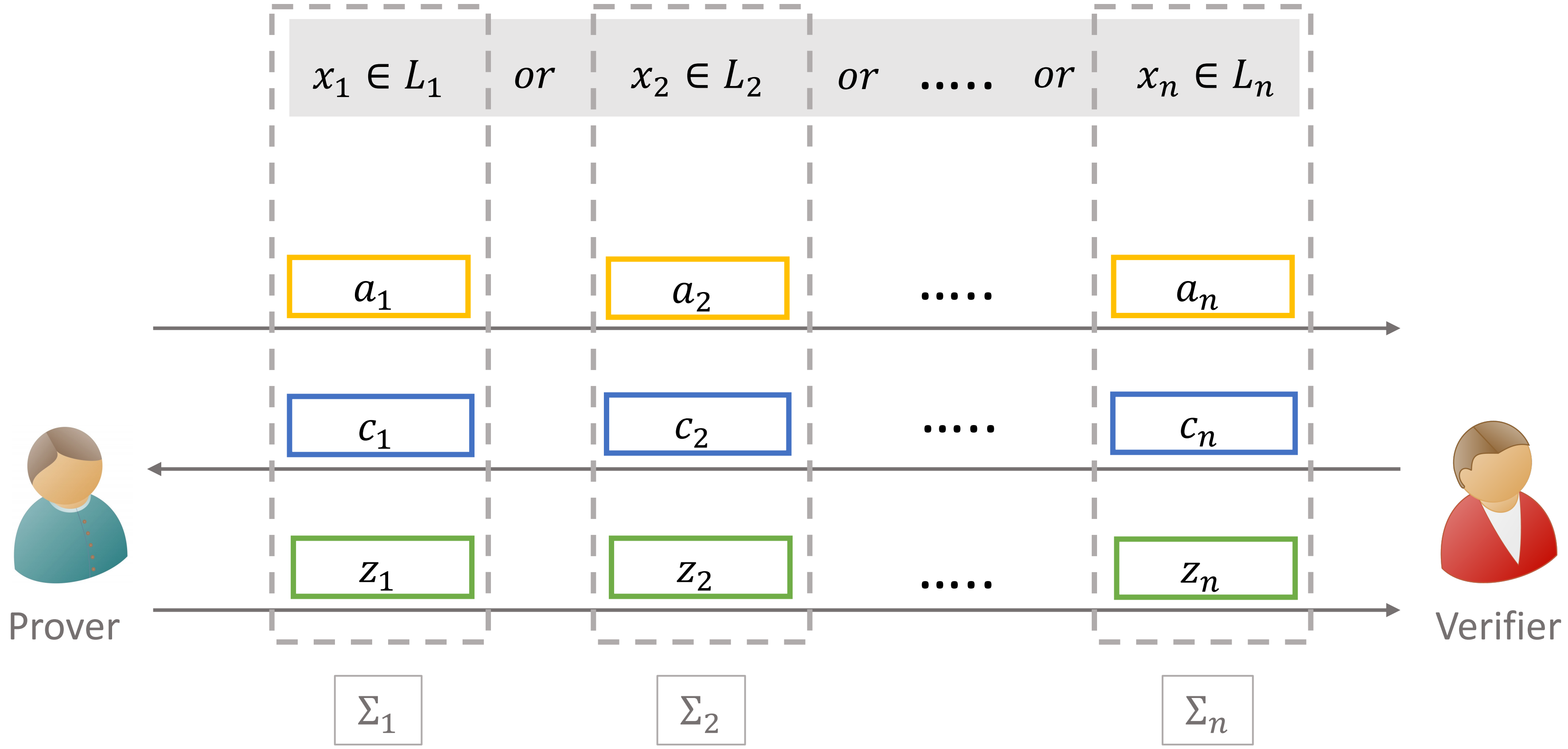
Stacking Σ-Protocols for Disjunctions



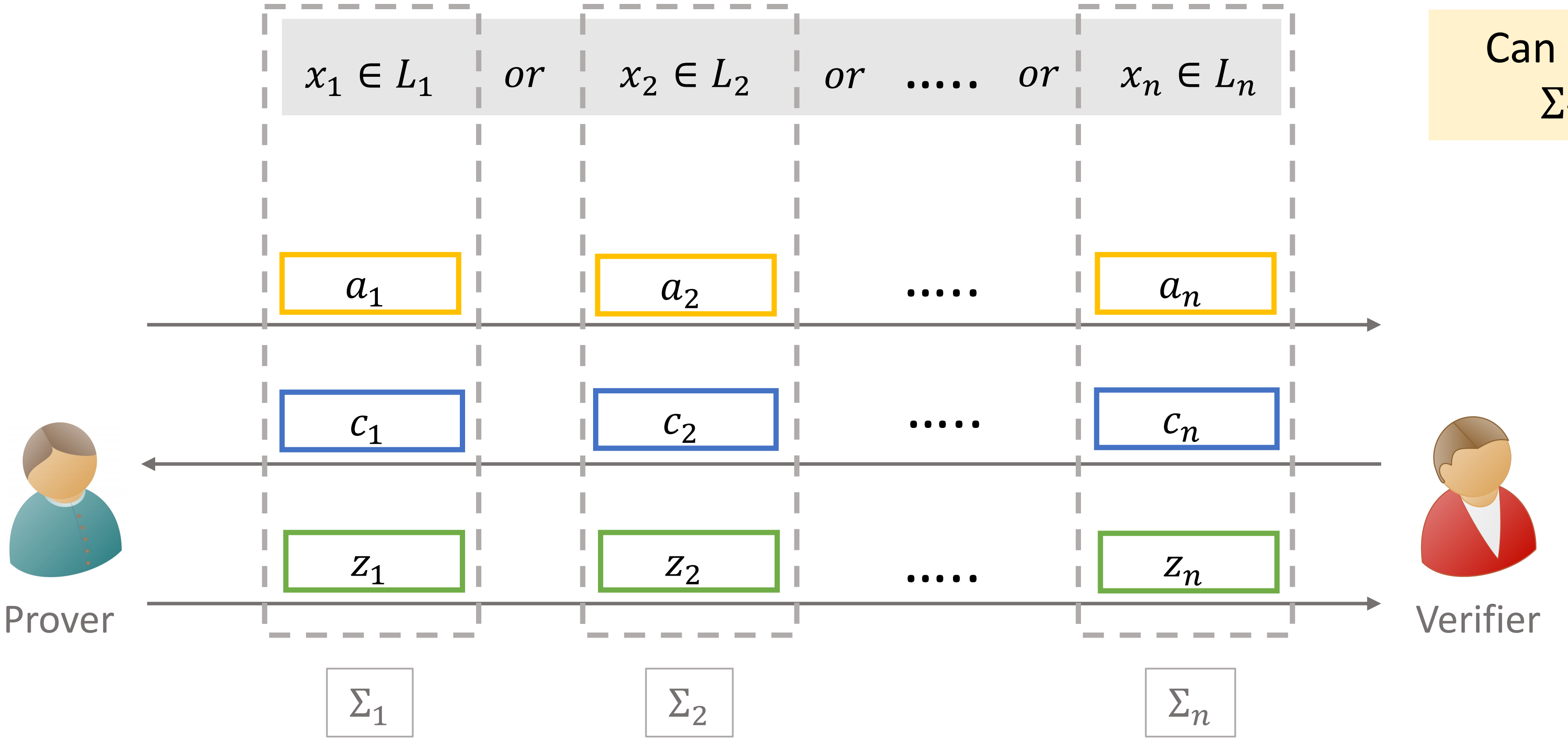
Stacking Σ-Protocols for Disjunctions



Stacking Σ-Protocols for Disjunctions

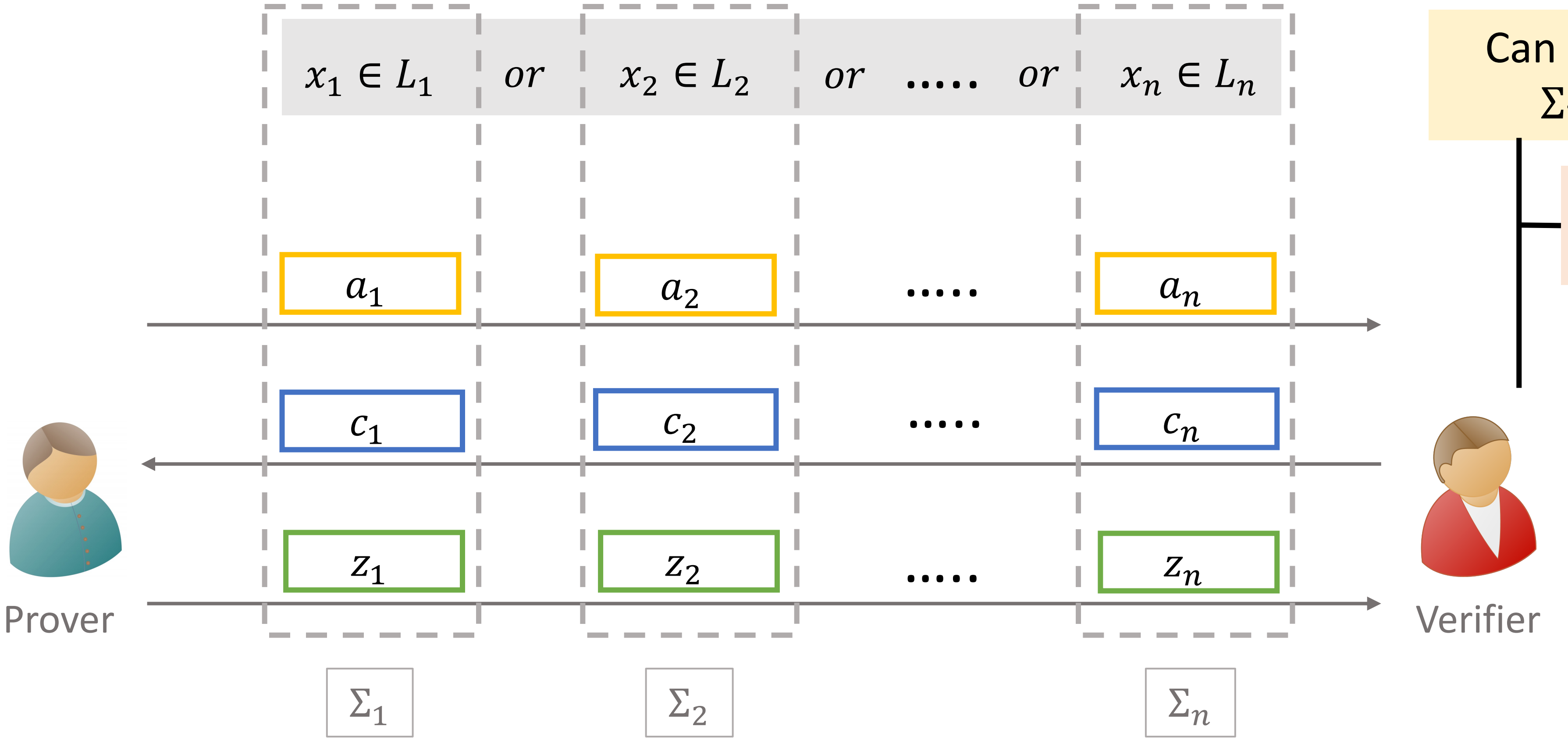


Stacking Σ-Protocols for Disjunctions



Can we generically compose these Σ-Protocols for disjunctions?

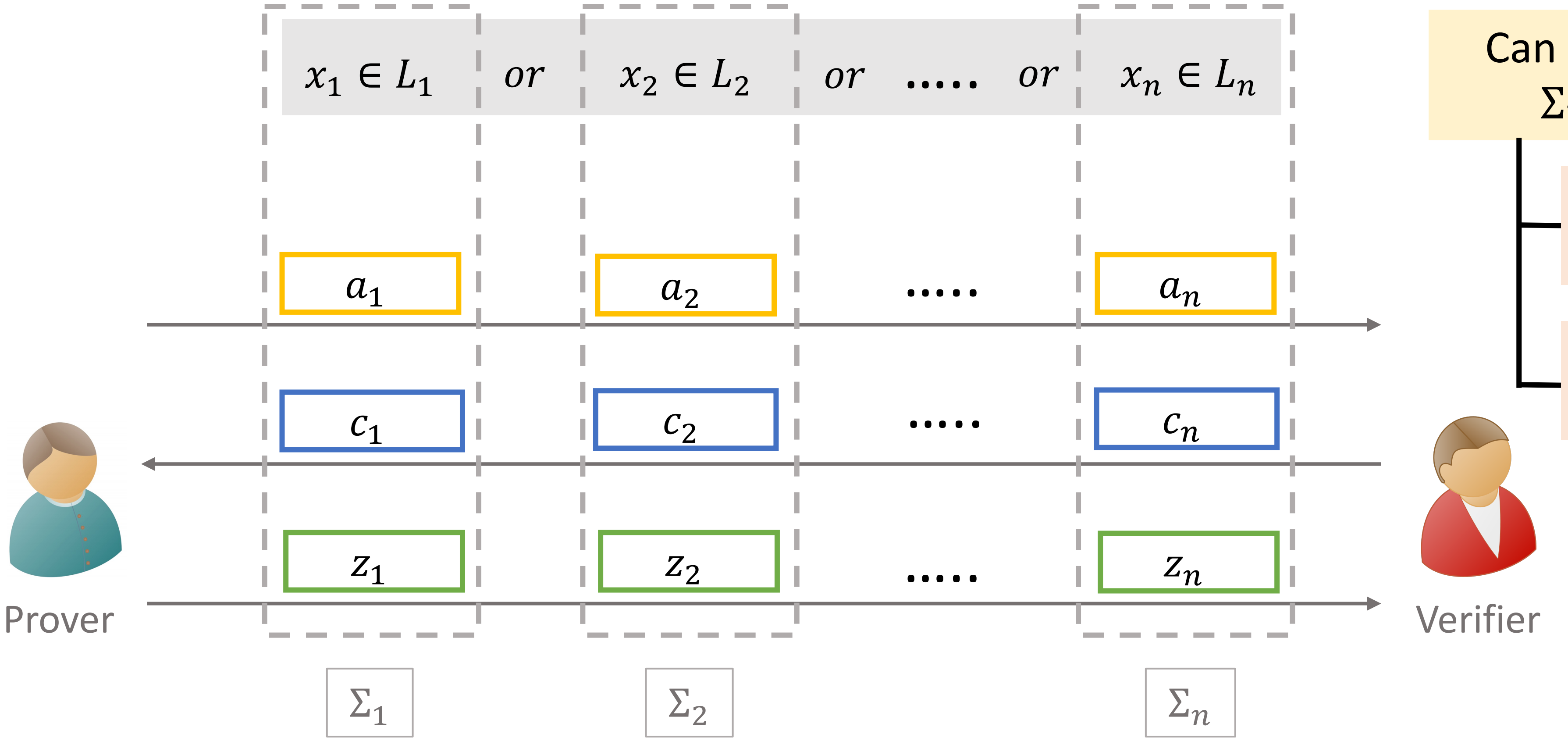
Stacking Σ-Protocols for Disjunctions



Can we generically compose these Σ-Protocols for disjunctions?

Without modifying the underlying Σ-Protocols

Stacking Σ-Protocols for Disjunctions



Can we generically compose these Σ-Protocols for disjunctions?

- Without modifying the underlying Σ-Protocols
- Communication is that same as that of proving a single branch

Applications of such Stacking Compilers

Reduces manual effort of modifying existing techniques

Applications of such Stacking Compilers

Reduces manual effort of modifying existing techniques

Newly developed Σ-protocols can also be used to produce stacked proofs immediately

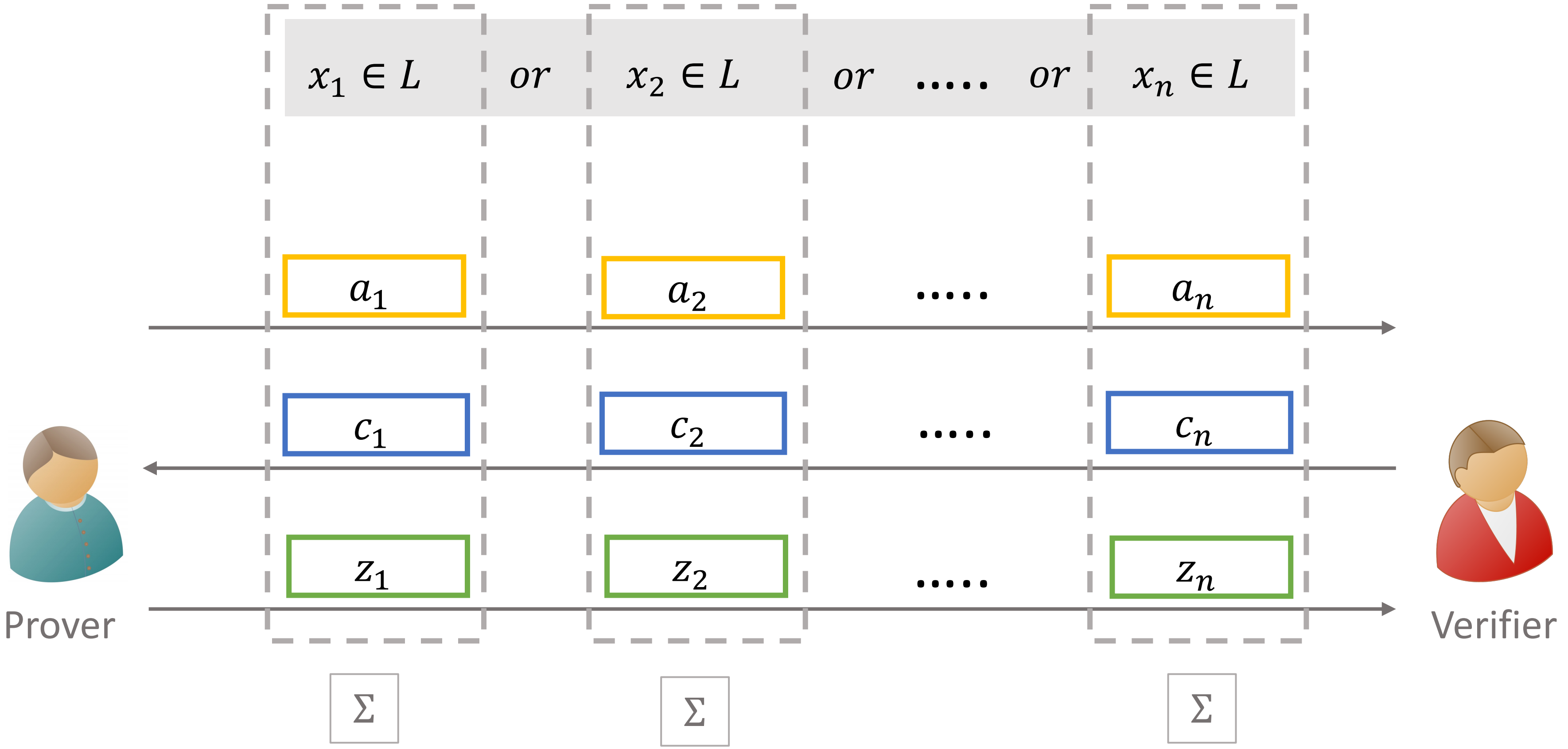
Applications of such Stacking Compilers

Reduces manual effort of modifying existing techniques

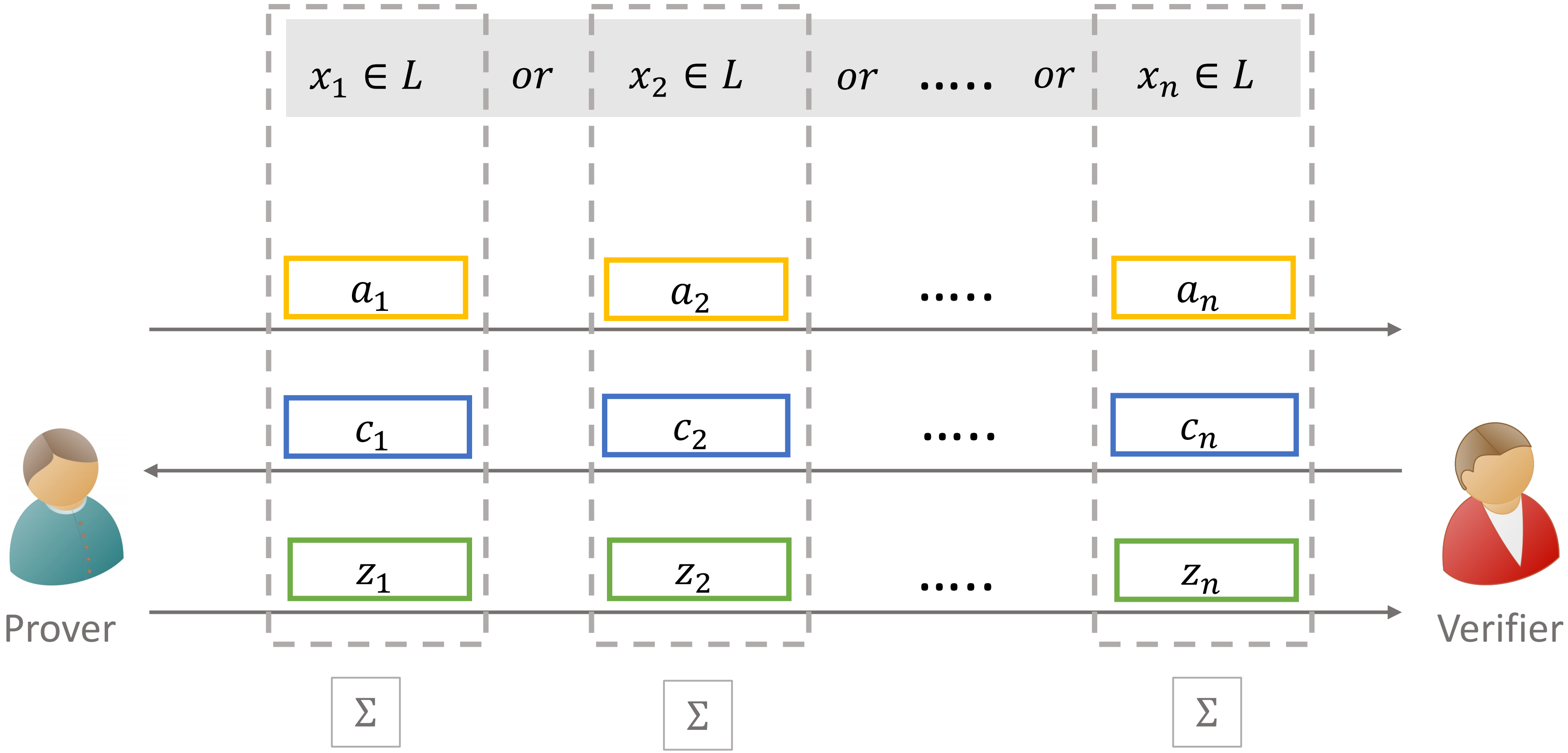
Newly developed Σ-protocols can also be used to produce stacked proofs immediately

Empowering protocol designers to choose appropriate Σ-protocols based on their application

Stacking Σ-Protocols for Disjunctions

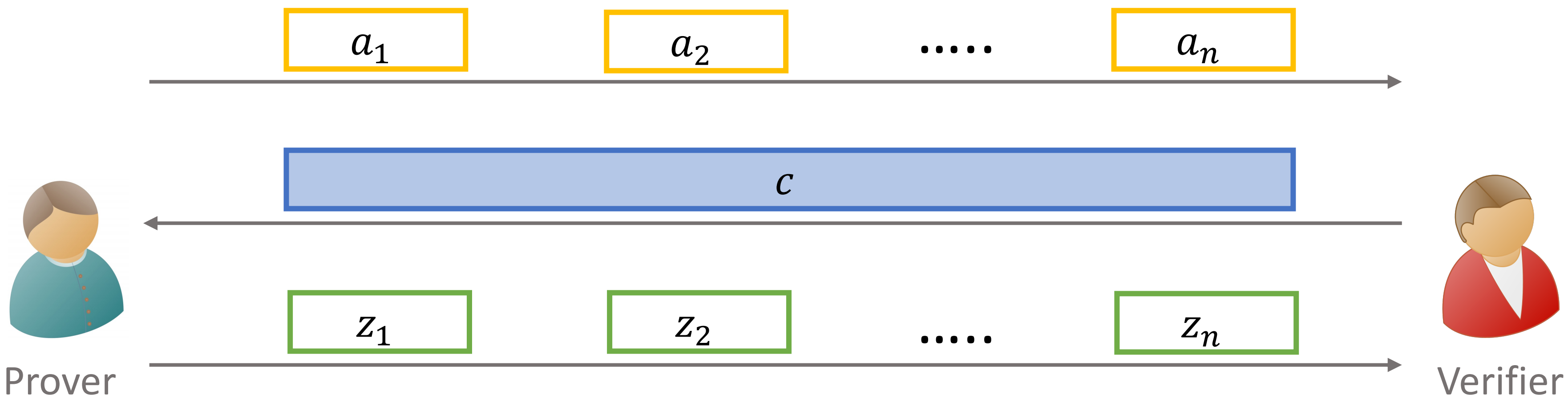


Stacking Σ-Protocols for Disjunctions



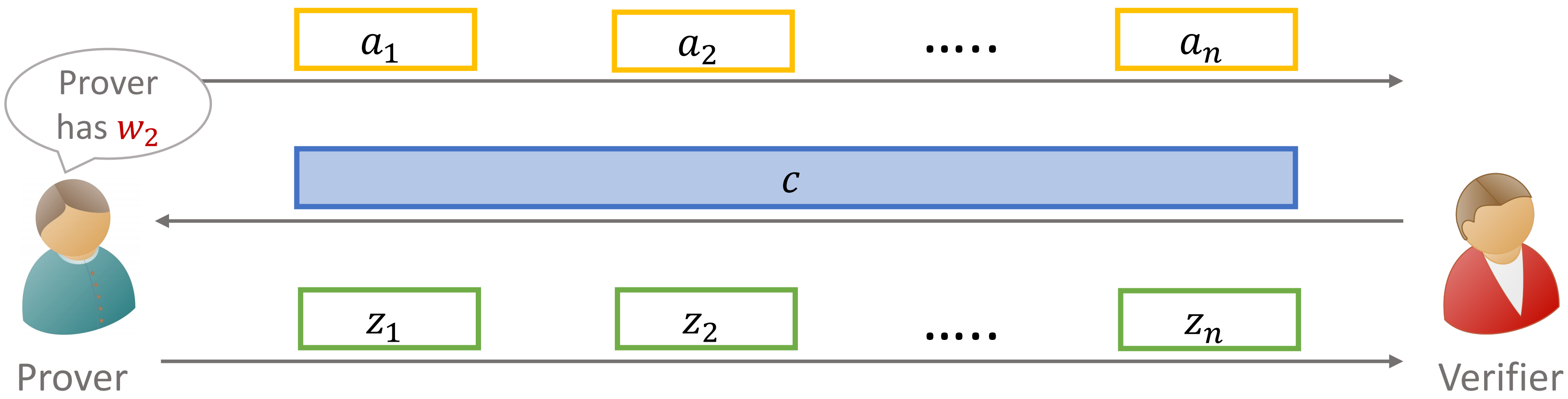
Stacking Σ-Protocols for Disjunctions

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad ..... \quad or \quad x_n \in L$



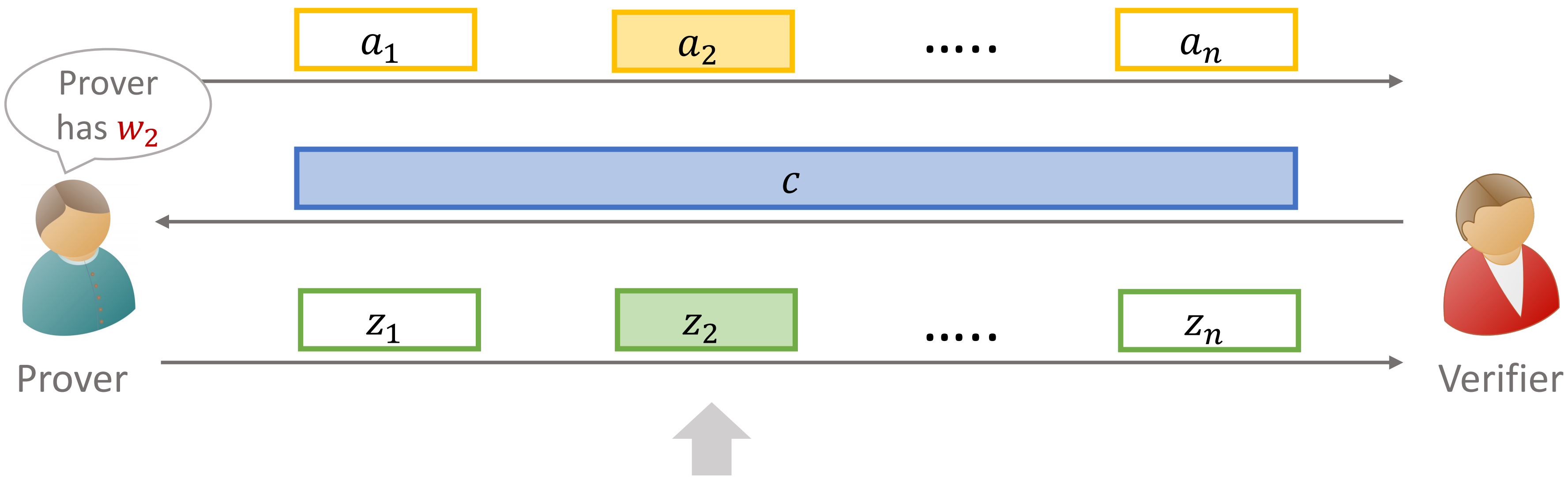
Stacking Σ-Protocols for Disjunctions

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad ..... \quad or \quad x_n \in L$



Stacking Σ-Protocols for Disjunctions

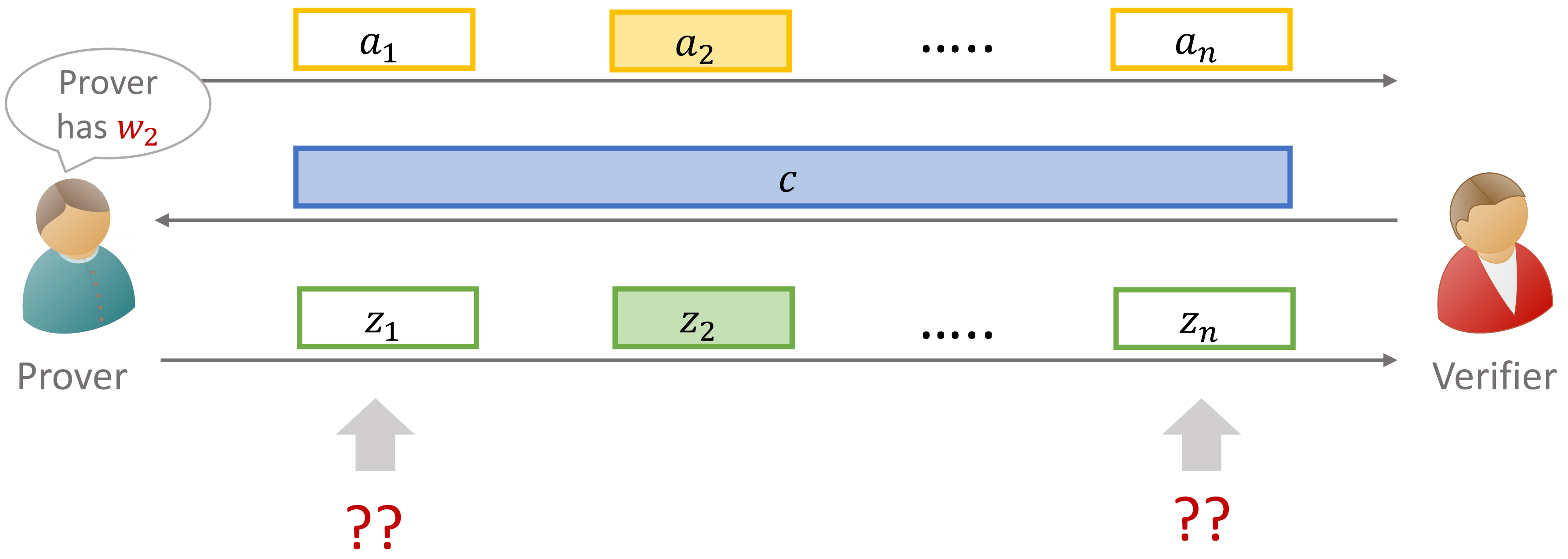
$x_1 \in L$ $\quad or \quad$ $x_2 \in L$ $\quad or \quad$ ….. $\quad or \quad$ $x_n \in L$



Prover can compute these messages

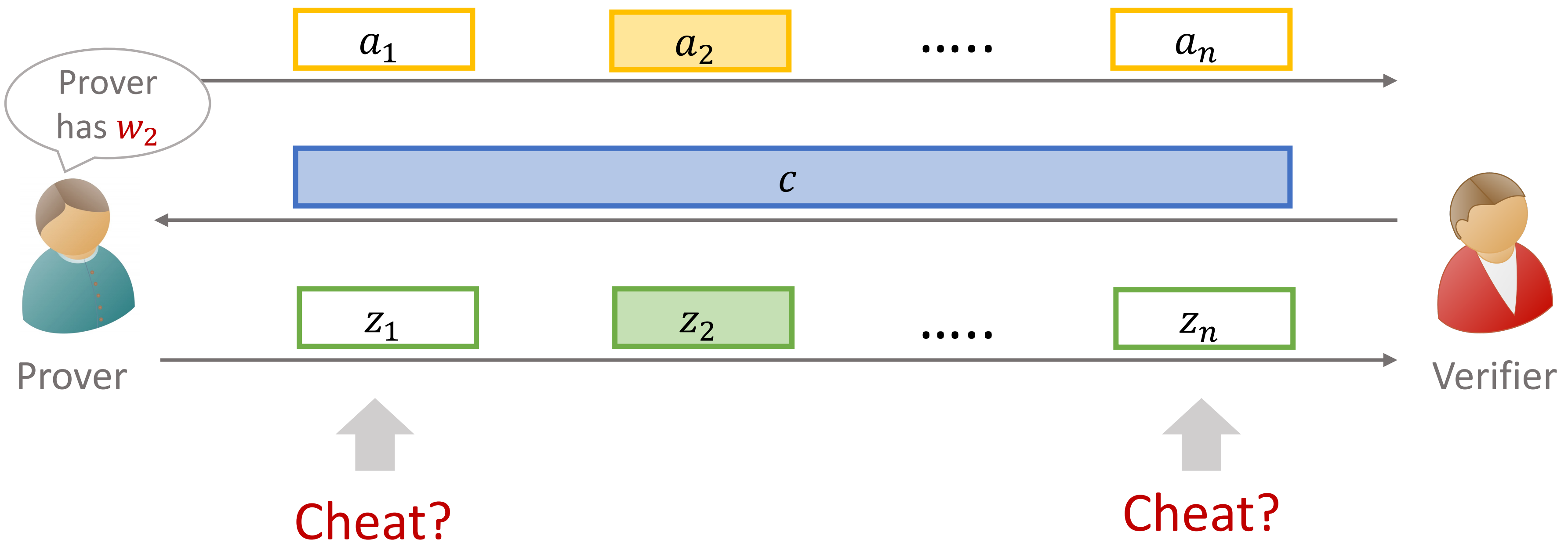
Stacking Σ-Protocols for Disjunctions

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad ..... \quad or \quad x_n \in L$



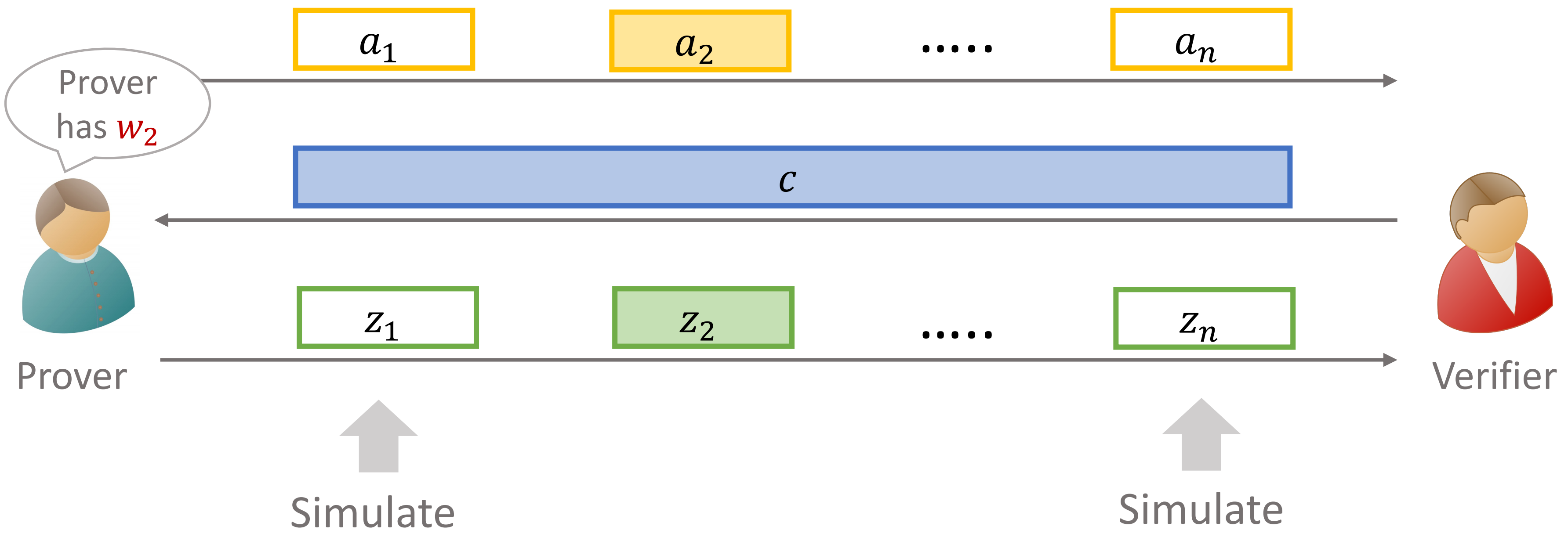
Stacking Σ-Protocols for Disjunctions

$x_1 \in L$ or $x_2 \in L$ or ….. or $x_n \in L$

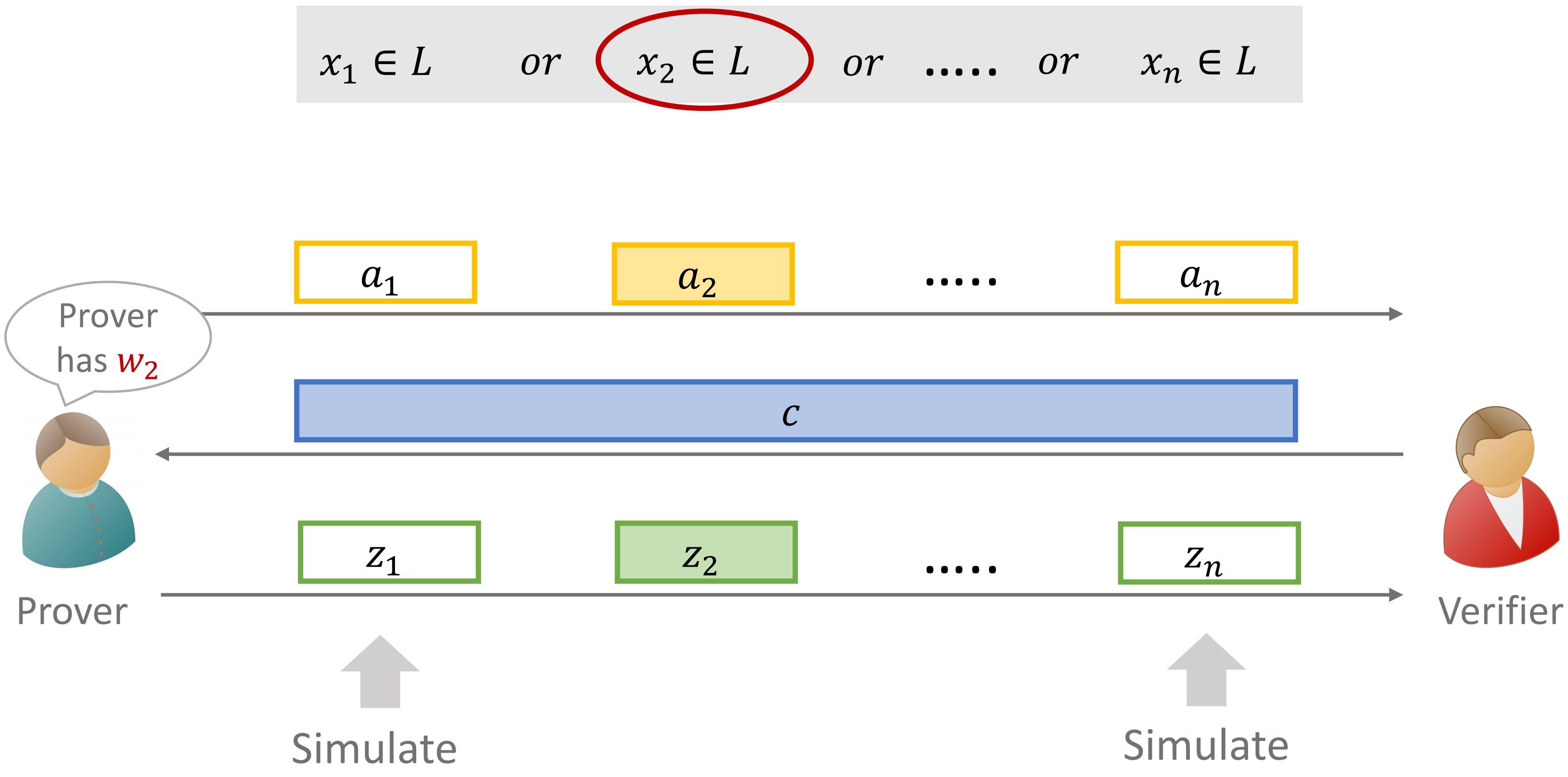


Stacking Σ-Protocols for Disjunctions

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad \ldots.. \quad or \quad x_n \in L$



Stacking Σ-Protocols for Disjunctions

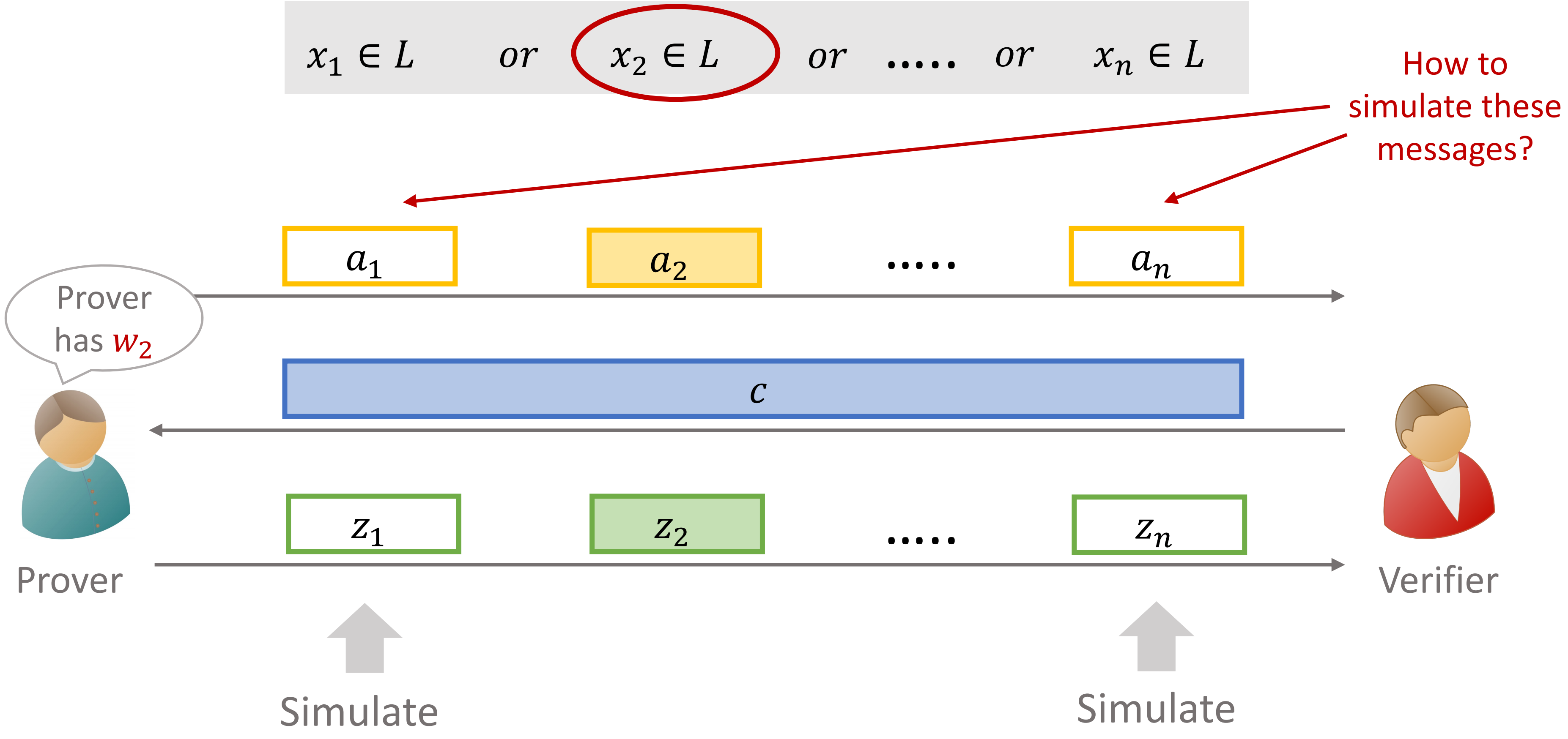


Common simulation strategy in Σ-protocols:

Step 1: Sample c

Step 2: Compute a, z

Stacking Σ-Protocols for Disjunctions



Common simulation strategy in Σ-protocols:

Step 1: Sample c

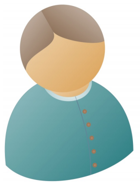
Step 2: Compute a, z

Stacking Σ-Protocols for Disjunctions

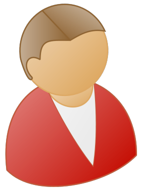
$x_1 \in L$ $\quad or \quad$ $x_2 \in L$ $\quad or \quad$ ..... $\quad or \quad$ $x_n \in L$

$Com($ garbage $\quad a_2 \quad$ ..... $\quad$ garbage $)$

Prover has w_2



Prover



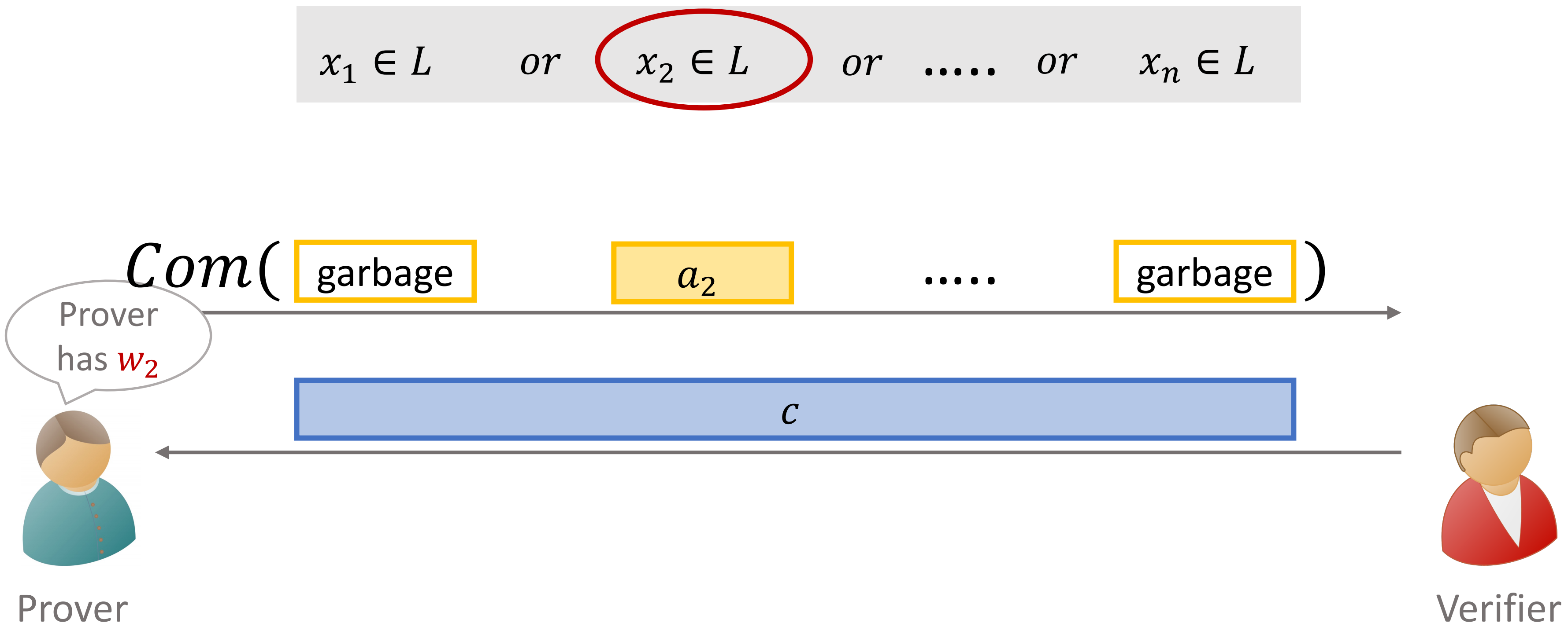
Verifier

<u>Common simulation strategy in Σ-protocols:</u>

Step 1: Sample c

Step 2: Compute a, z

Stacking Σ-Protocols for Disjunctions



Common simulation strategy in Σ-protocols:

Step 1: Sample c

Step 2: Compute a, z

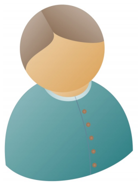
Stacking Σ-Protocols for Disjunctions

$x_1 \in L$ $\quad or \quad$ $x_2 \in L$ $\quad or \quad$ ..... $\quad or \quad$ $x_n \in L$

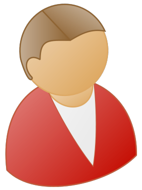
$Com($ garbage $\quad a_2 \quad$ ..... $\quad$ garbage $)$

Prover has w_2

c



Prover



Verifier

Simulate $(a_1, z_1), (a_3, z_3) \ldots, (a_n, z_n)$

$op =$ Equivocate com to $[a_1, a_2, \ldots, a_n]$

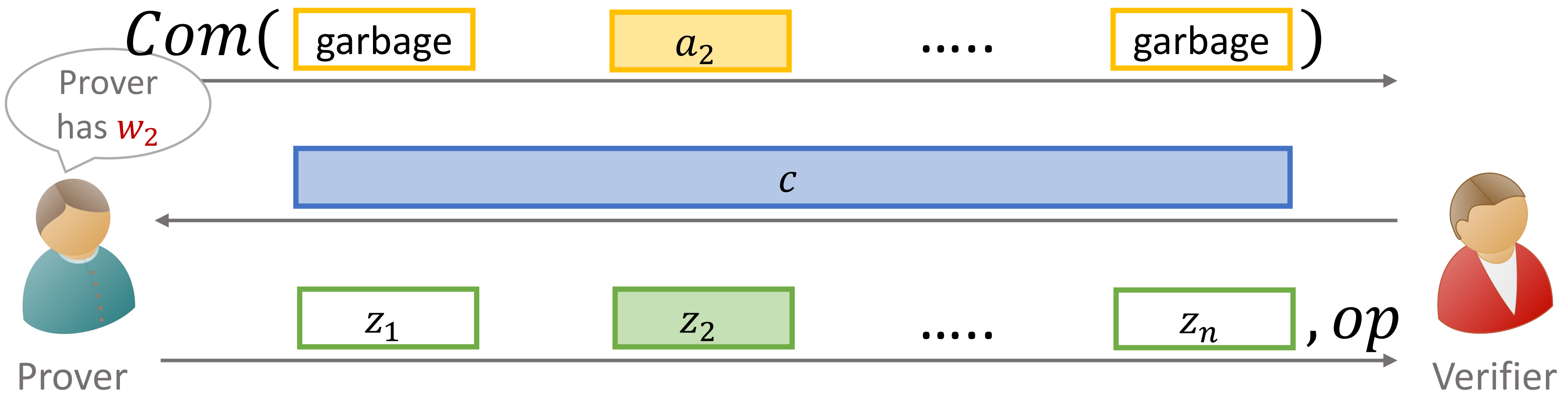
Common simulation strategy in Σ-protocols:

Step 1: Sample c

Step 2: Compute $a,$ z

Stacking Σ-Protocols for Disjunctions

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad ..... \quad or \quad x_n \in L$



Simulate $(a_1, z_1), (a_3, z_3) \dots, (a_n, z_n)$

$op =$ Equivocate com to $[a_1, a_2, \dots, a_n]$

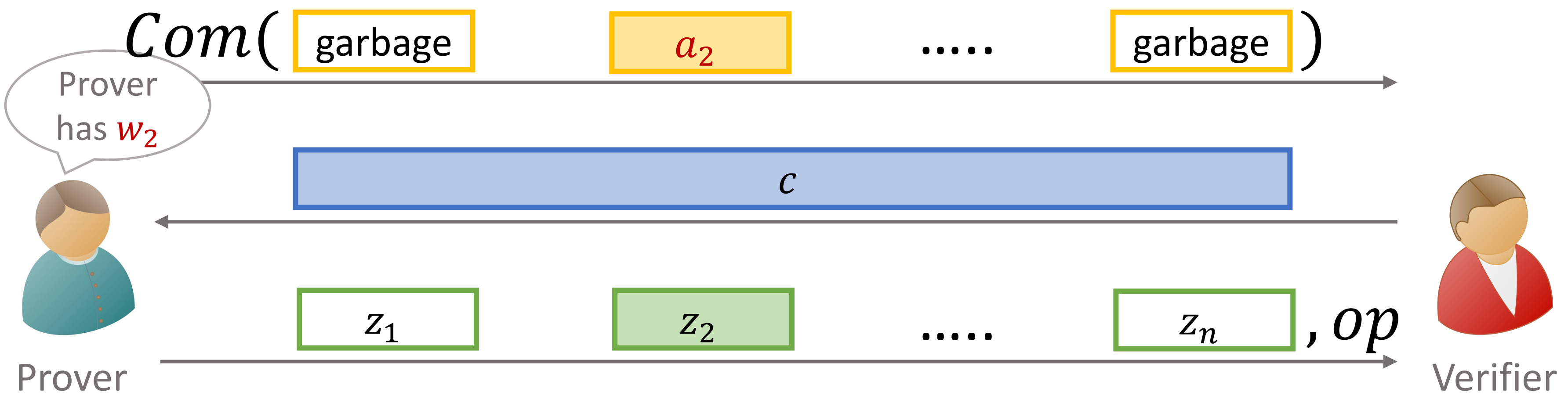
Common simulation strategy in Σ-protocols:

Step 1: Sample c

Step 2: Compute a, z

Stacking Σ-Protocols for Disjunctions

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad ..... \quad or \quad x_n \in L$



Common simulation strategy in Σ-protocols:

Step 1: Sample c

Step 2: Compute $a,$ z

Properties of these commitment schemes?

Partially Binding Vector Commitments

t-out-of-n positions are binding. Rest can be equivocated.

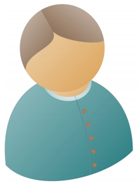
Binding positions are fixed at the time of commitment.

Binding positions remain hidden from the receiver.

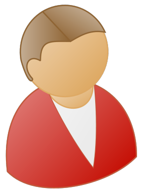
We propose a construction using Discrete Log

Stacking Σ-Protocol for Disjunctions

Prover has w_2



Prover



Verifier

$x_1 \in L$ or $x_2 \in L$ or ..... or $x_n \in L$

$Com($ garbage a_2 ..... garbage $)$

c

Compute z_2 honestly

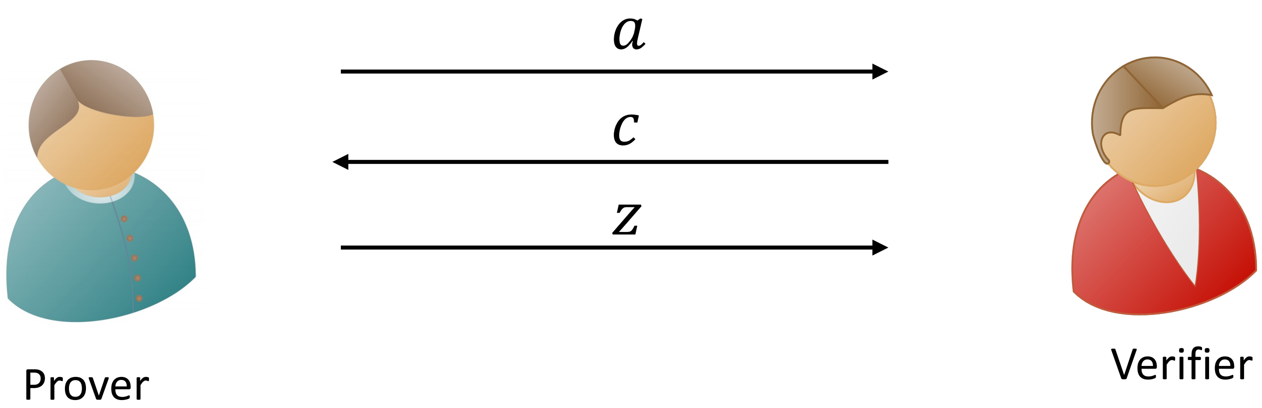
Simulate $(a_1, z_1), (a_3, z_3) \dots, (a_n, z_n)$ using c

$op =$ Equivocate com to $[a_1, a_2, \dots, a_n]$

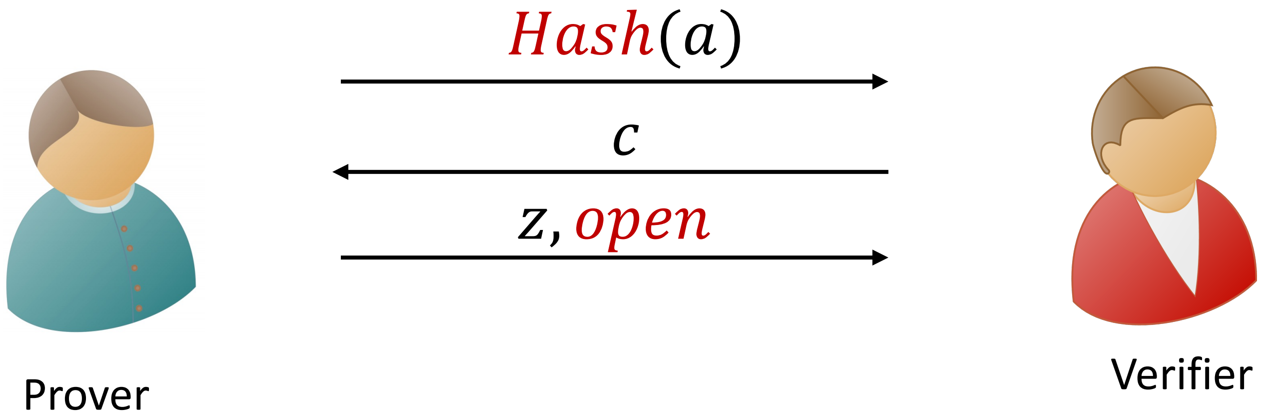
z_1 z_2 ..... z_n $, op$

This is a valid Σ-protocol for disjunctions. But we haven't really saved any communication?

Bulkiest Part of a Σ-Protocol

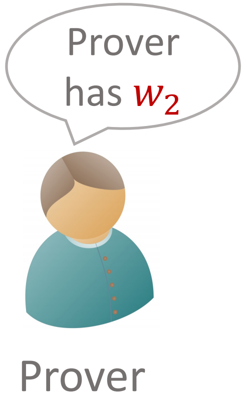


Bulkiest Part of a Σ-Protocol



W.l.o.g., Third round messages are the longest!

Stacking Σ-Protocol for Disjunctions



$x_1 \in L$ or $x_2 \in L$ or ..... or $x_n \in L$

$Com($ garbage a_2 ..... garbage $)$

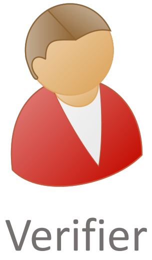
c

Compute z_2 honestly

Simulate $(a_1, z_1), (a_3, z_3) \dots, (a_n, z_n)$ using c

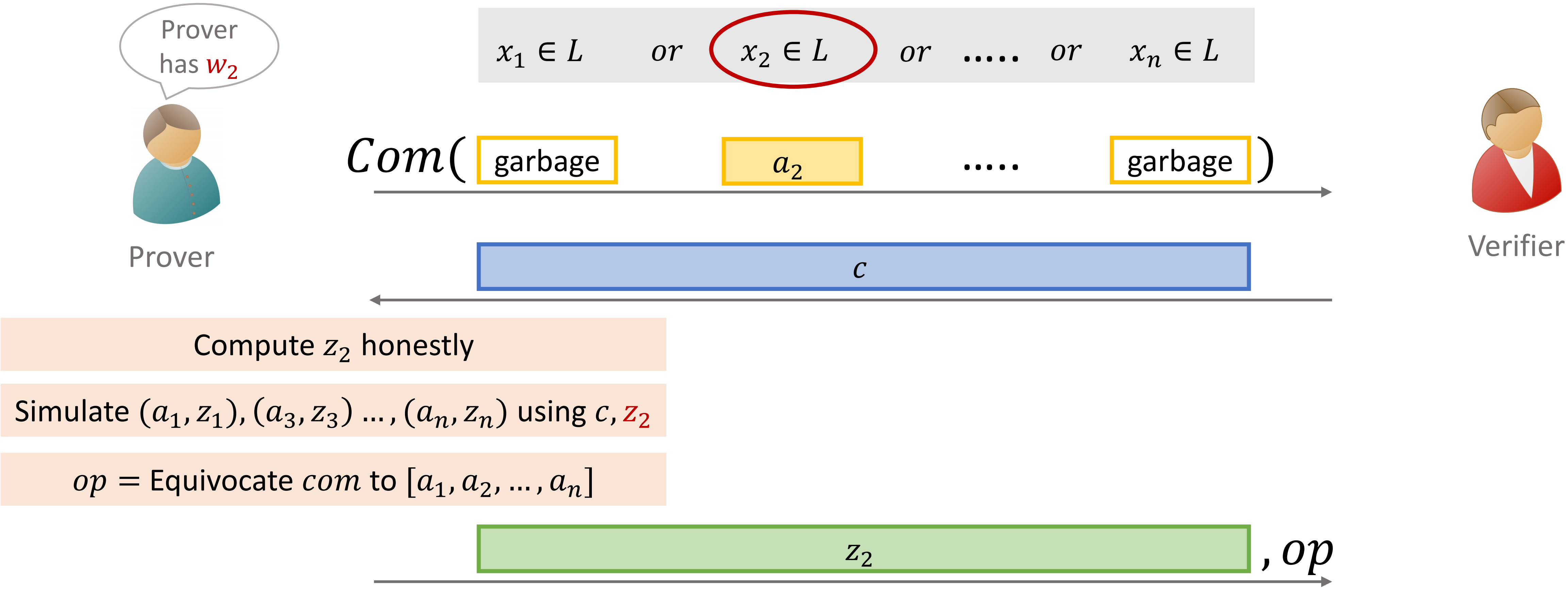
$op =$ Equivocate com to $[a_1, a_2, \dots, a_n]$

z_1 z_2 ..... z_n $, op$

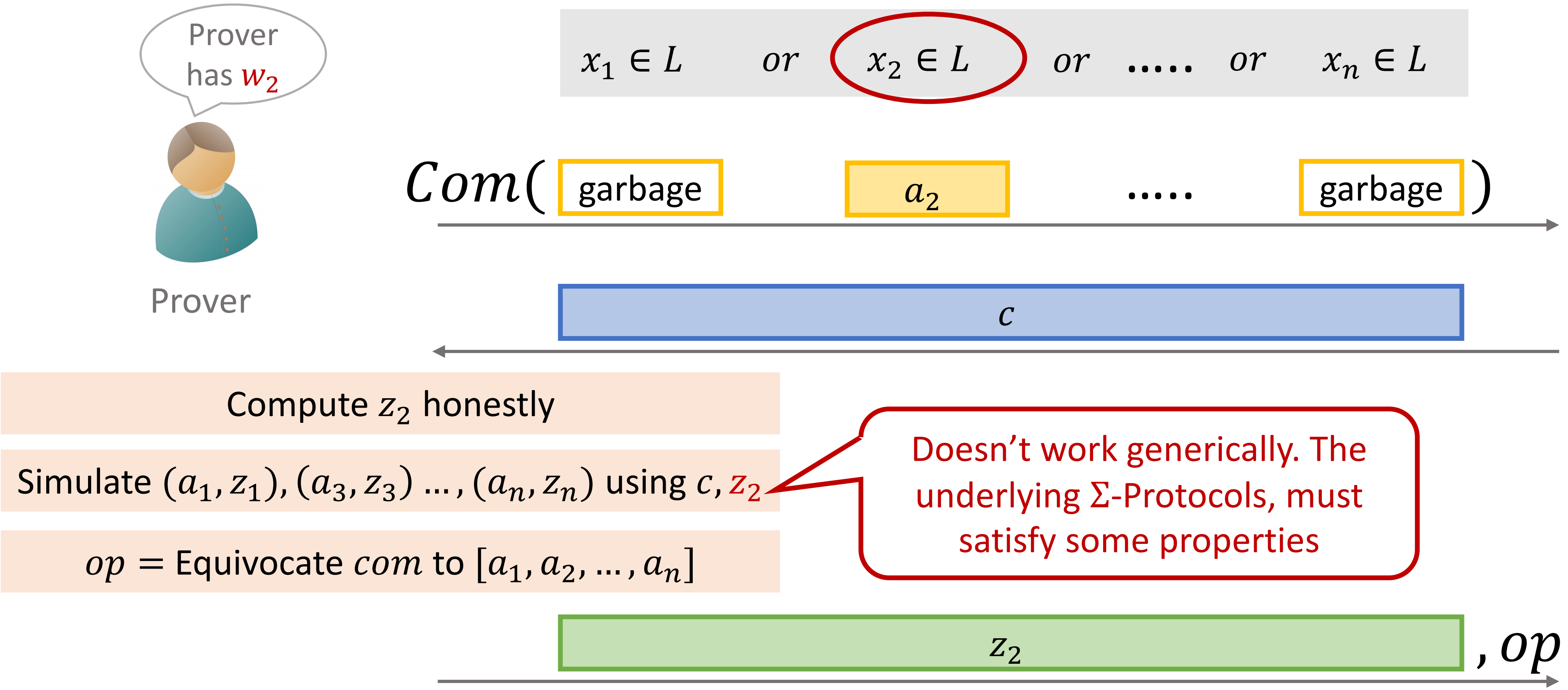


Can we re-use the third-round message of the active branch?

Stacking Σ-Protocol for Disjunctions



Stacking Σ-Protocol for Disjunctions



Stackable Properties

Property 1: Extended Honest Verifier Zero-knowledge

Stackable Properties

Property 1: Extended Honest Verifier Zero-knowledge

Simulation: For any instance x and challenge c, first compute a third-round message, then simulate the corresponding first round message.

Stackable Properties

Property 1: Extended Honest Verifier Zero-knowledge

Simulation: For any instance x and challenge c, first compute a third-round message, then simulate the corresponding first round message.

$$\left\{(a,c,z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\} \approx \left\{(a,c,z) \mid z \xleftarrow{\$} \mathcal{D}^{(z)}_{x,c}; a \leftarrow \mathcal{S}^{\mathrm{EHVZK}}(1^\lambda,x,c,z)\right\}$$

Stackable Properties

Property 1: Extended Honest Verifier Zero-knowledge

Simulation: For any instance x and challenge c, first compute a third-round message, then simulate the corresponding first round message.

$$\left\{(a,c,z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\} \approx \left\{(a,c,z) \mid z \xleftarrow{\$} \mathcal{D}^{(z)}_{x,c}; a \leftarrow \mathcal{S}^{\mathrm{EHVZK}}(1^\lambda,x,c,z)\right\}$$

Property 2: Recyclable Third Round Messages

Stackable Properties

Property 1: Extended Honest Verifier Zero-knowledge

Simulation: For any instance x and challenge c, first compute a third-round message, then simulate the corresponding first round message.

$$\left\{(a, c, z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x, w; r^p); z \leftarrow Z(x, w, c; r^p)\right\} \approx \left\{(a, c, z) \mid z \xleftarrow{\$} \mathcal{D}_{x,c}^{(z)}; a \leftarrow \mathcal{S}^{\mathrm{EHVZK}}(1^\lambda, x, c, z)\right\}$$

Property 2: Recyclable Third Round Messages

Given a fixed challenge, the distribution of possible third round messages for any pair of statements in the language are indistinguishable from each other.

Stackable Properties

Property 1: Extended Honest Verifier Zero-knowledge

Simulation: For any instance x and challenge c, first compute a third-round message, then simulate the corresponding first round message.

$$\left\{(a,c,z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\} \approx \left\{(a,c,z) \mid z \xleftarrow{\$} \mathcal{D}_{x,c}^{(z)}; a \leftarrow \mathcal{S}^{\text{EHVZK}}(1^\lambda,x,c,z)\right\}$$

Property 2: Recyclable Third Round Messages

Given a fixed challenge, the distribution of possible third round messages for any pair of statements in the language are indistinguishable from each other.

$$\mathcal{D}_c^{(z)} \approx \left\{z \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\}$$

Stackable Properties

Property 1: Extended Honest Verifier Zero-knowledge

$$\left\{(a,c,z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\} \approx \left\{(a,c,z) \mid z \xleftarrow{\$} \mathcal{D}_{x,c}^{(z)}; a \leftarrow \mathcal{S}^{\text{EHVZK}}(1^\lambda,x,c,z)\right\}$$

+

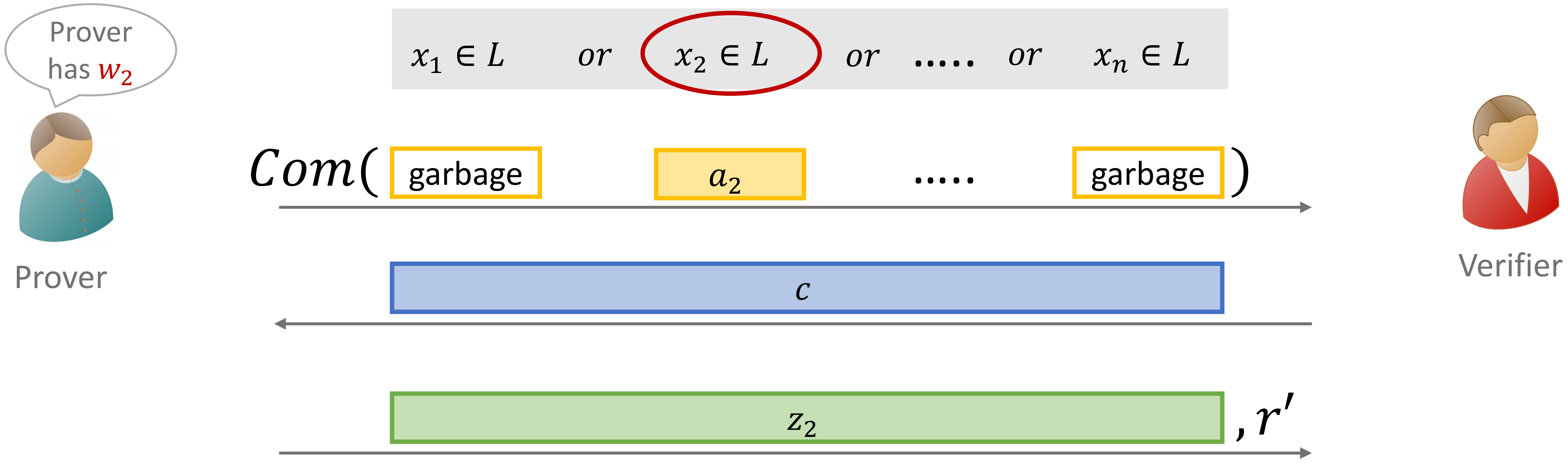
Property 2: Recyclable Third Round Messages

$$\mathcal{D}_c^{(z)} \approx \left\{z \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\}$$

=

$$\left\{(a,z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\} \approx \left\{(a,z) \mid z \xleftarrow{\$} \mathcal{D}_c^{(z)}; a \leftarrow \mathcal{S}^{\text{EHVZK}}(1^\lambda,x,c,z)\right\}$$

Stacked Σ-Protocol for Disjunctions



Communication = proof size for proving a single branch + size of commitment + size of opening

Can be short

At least linear in the length of the vector

Recursive Stacking

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad \ldots.. \quad or \quad x_n \in L$

1 out of 2 disjunction

$\Sigma_2 =$ Stack Σ and Σ

Communication = | Σ | + Commitment + 1

Recursive Stacking

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad ..... \quad or \quad x_n \in L$

1 out of 2 disjunction	Σ_2 = Stack Σ and Σ	Communication = | Σ | + Commitment + 1
1 out of 4 disjunction	Σ_4 = Stack Σ_2 and Σ_2	Communication = | Σ | + 2 $\times$ Commitment + 1 + 1

Recursive Stacking

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad ..... \quad or \quad x_n \in L$

1 out of 2 disjunction	Σ_2 = Stack Σ and Σ	Communication = | Σ | + Commitment + 1
1 out of 4 disjunction	Σ_4 = Stack Σ_2 and Σ_2	Communication = | Σ | + 2 $\times$ Commitment + 1 + 1
1 out of 8 disjunction	Σ_8 = Stack Σ_4 and Σ_4	Communication = | Σ | + 3 $\times$ Commitment + 1 +1 + 1

Recursive Stacking

$x_1 \in L \quad or \quad x_2 \in L \quad or \quad ..... \quad or \quad x_n \in L$

1 out of 2 disjunction	$\Sigma_2 =$ Stack Σ and Σ	Communication = | Σ | + Commitment + 1
1 out of 4 disjunction	$\Sigma_4 =$ Stack Σ_2 and Σ_2	Communication = | Σ | + 2 $\times$ Commitment + 1 + 1
1 out of 8 disjunction	$\Sigma_8 =$ Stack Σ_4 and Σ_4	Communication = | Σ | + 3 $\times$ Commitment + 1 +1 + 1

.....

1 out of n disjunction	$\Sigma_n =$ Stack $\Sigma_{n/2}$ and $\Sigma_{n/2}$	Communication = | Σ | + log(n) $\times$ Commitment + log(n)

Examples of Stackable Σ-Protocols

Many natural sigma protocols are stackable

Examples of Stackable Σ-Protocols

Many natural sigma protocols are stackable

Example 1: Schnorr's Σ-Protocol

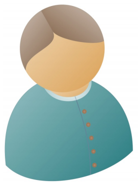
$R(x, w): x \stackrel{?}{=} g^x$

Examples of Stackable Σ-Protocols

Many natural sigma protocols are stackable

Example 1: Schnorr's Σ-Protocol

$R(x, w): x =^{?}= g^x$

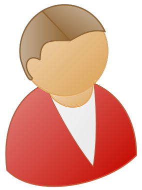


Prover

$$\xrightarrow{a = g^r}$$

$$\xleftarrow{c}$$

$$\xrightarrow{z = cw + r}$$



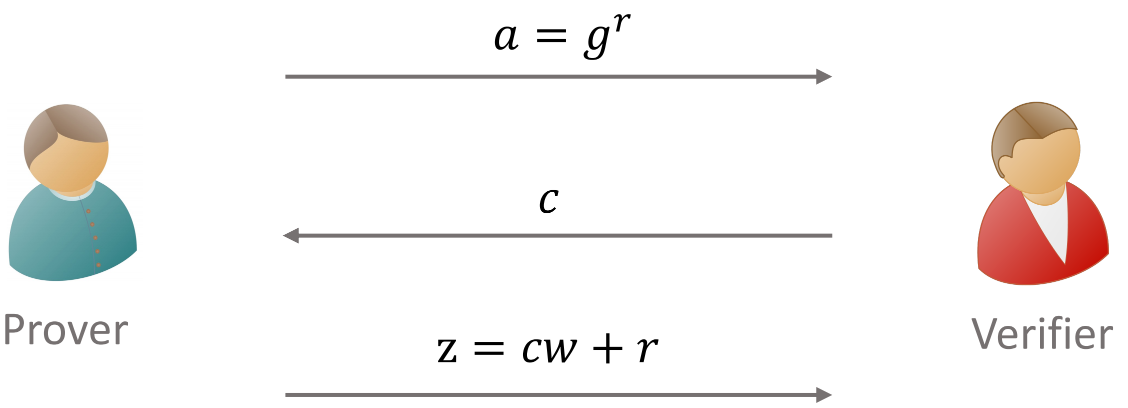
Verifier

Examples of Stackable Σ-Protocols

Many natural sigma protocols are stackable

Example 1: Schnorr's Σ-Protocol

$R(x, w): x \stackrel{?}{=} g^x$



Simulation Strategy: Sample random z. Compute $a = g^z x^{-c}$

Independent of instance

Examples of Stackable Σ-Protocols

Many natural sigma protocols are stackable

Example 1:	Schnorr's Σ-Protocol
Example 2:	Graph 3-coloring

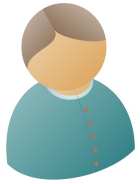
Is a graph $G = (V, E)$, 3-colorable?

Examples of Stackable Σ-Protocols

Many natural sigma protocols are stackable

Example 1:	Schnorr's Σ-Protocol
Example 2:	Graph 3-coloring

Is a graph $G = (V, E)$, 3-colorable?

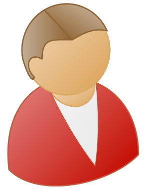


Prover

$a =$ commitment of permuted 3-coloring

c = random edge in graph

z =open colors of the edge



Verifier

Examples of Stackable Σ-Protocols

Many natural sigma protocols are stackable

Example 1:	Schnorr's Σ-Protocol
Example 2:	Graph 3-coloring

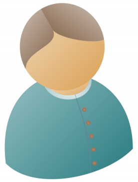
Is a graph $G = (V, E)$, 3-colorable?

$a =$ commitment of permuted 3-coloring →

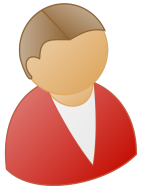
c = random edge in graph ←

z =open colors of the edge →

Independent of instance



Prover



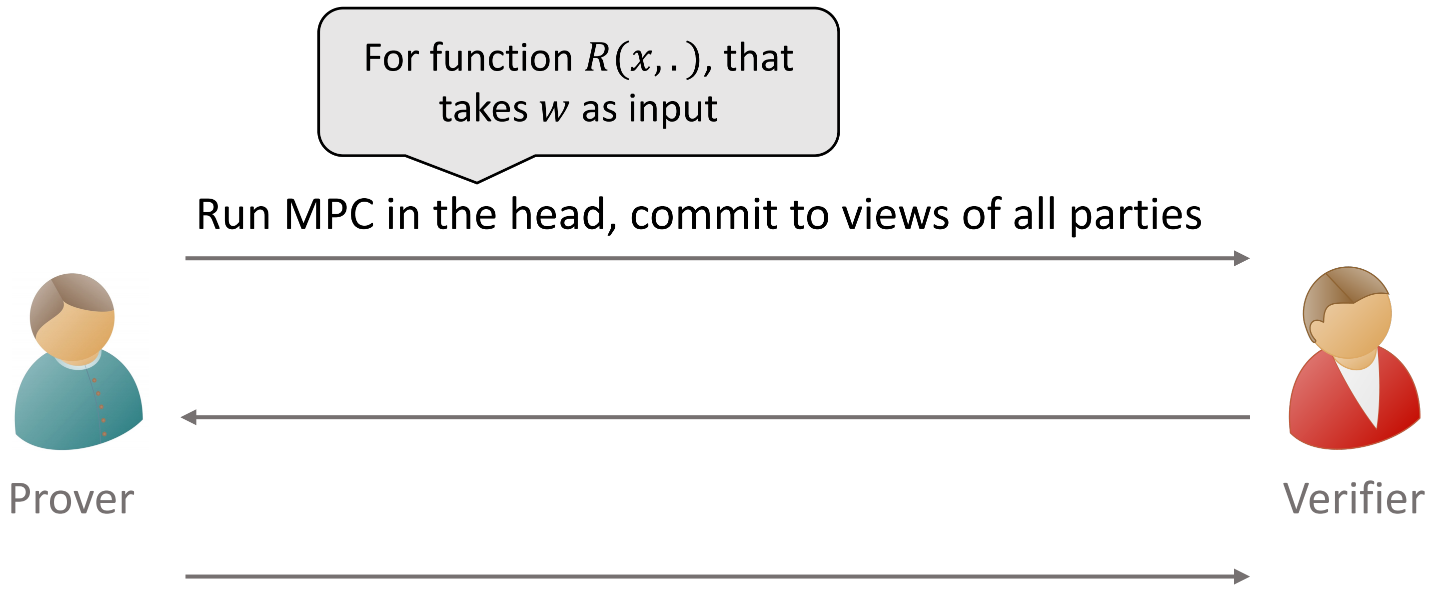
Verifier

Examples of Stackable Σ-Protocols

Many natural sigma protocols are stackable

Example 1:	Schnorr’s Σ-Protocol
Example 2:	Graph 3-coloring
Example 2:	MPC-in-the-head [IKOS]

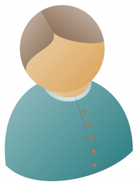
[IKOS07] is Stackable?



[IKOS07] is Stackable?

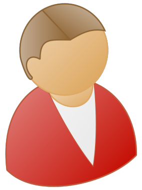
For function $R(x,.)$, that takes w as input

Run MPC in the head, commit to views of all parties



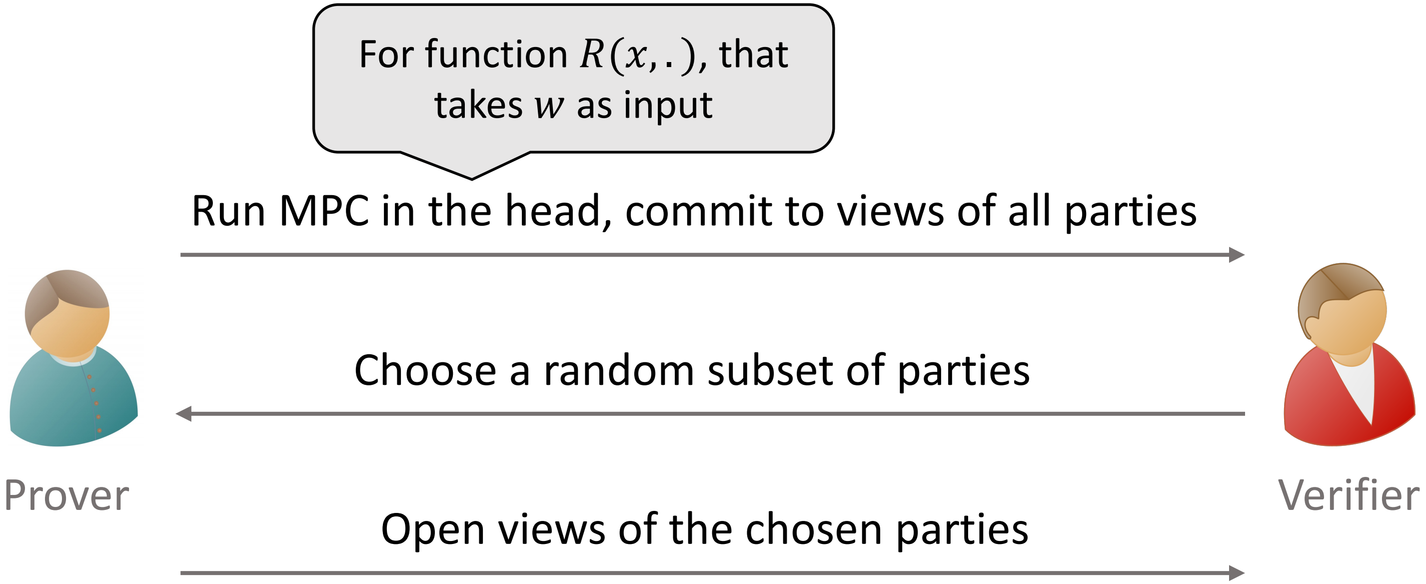
Prover

Choose a random subset of parties



Verifier

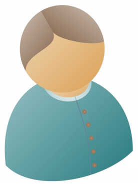
[IKOS07] is Stackable?



[IKOS07] is Stackable?

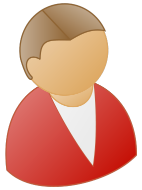
For function $R(x, .)$, that takes w as input

Run MPC in the head, commit to views of all parties →



Prover

Choose a random subset of parties ←



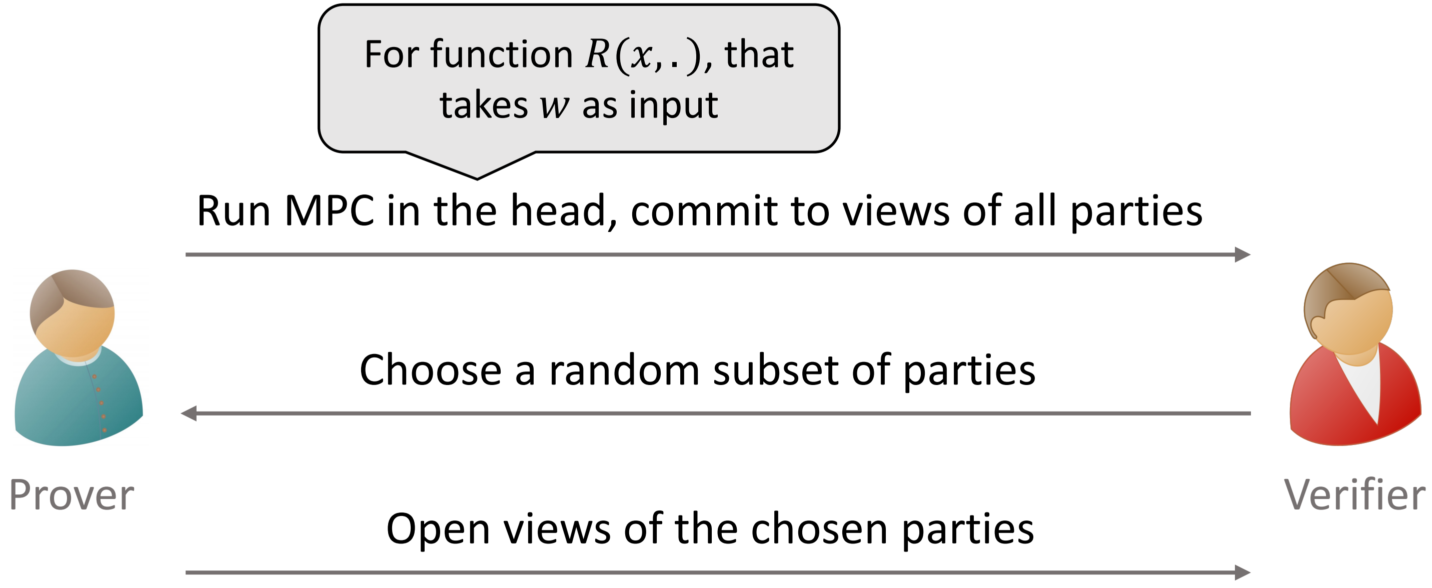
Verifier

Open views of the chosen parties →

Simulator

Choose a random subset of parties

[IKOS07] is Stackable?

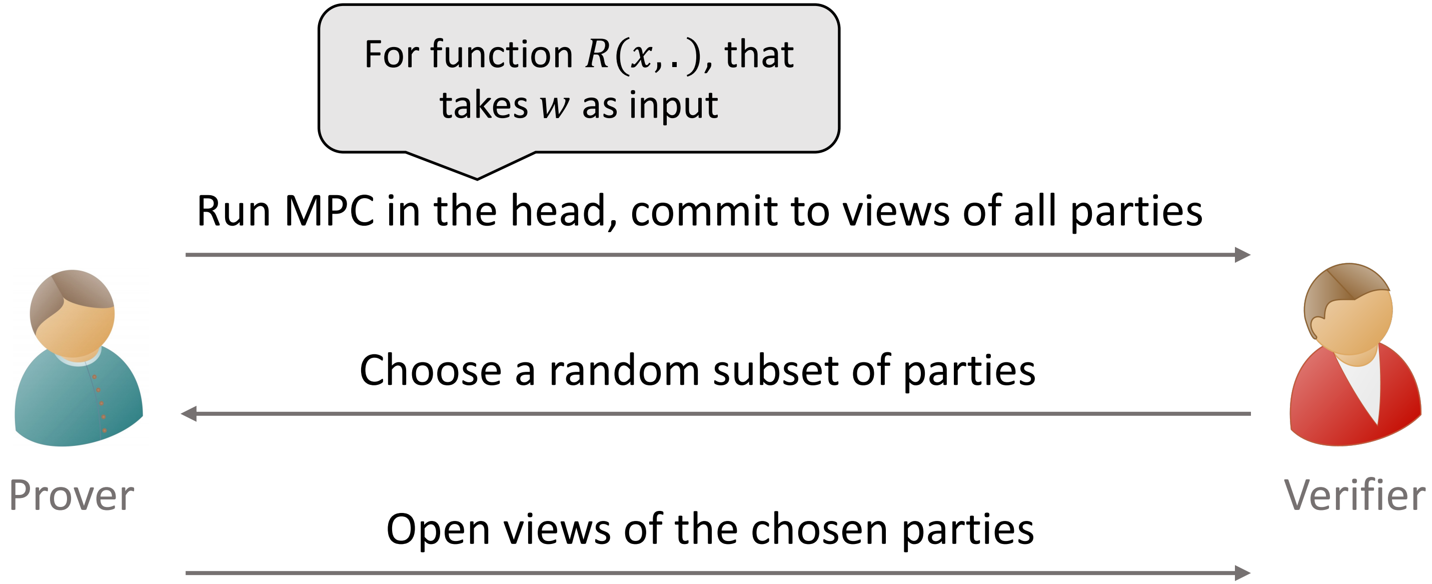


Simulator

Choose a random subset of parties

Simulate the views of these parties' using simulator of the underlying MPC protocol

[IKOS07] is Stackable?



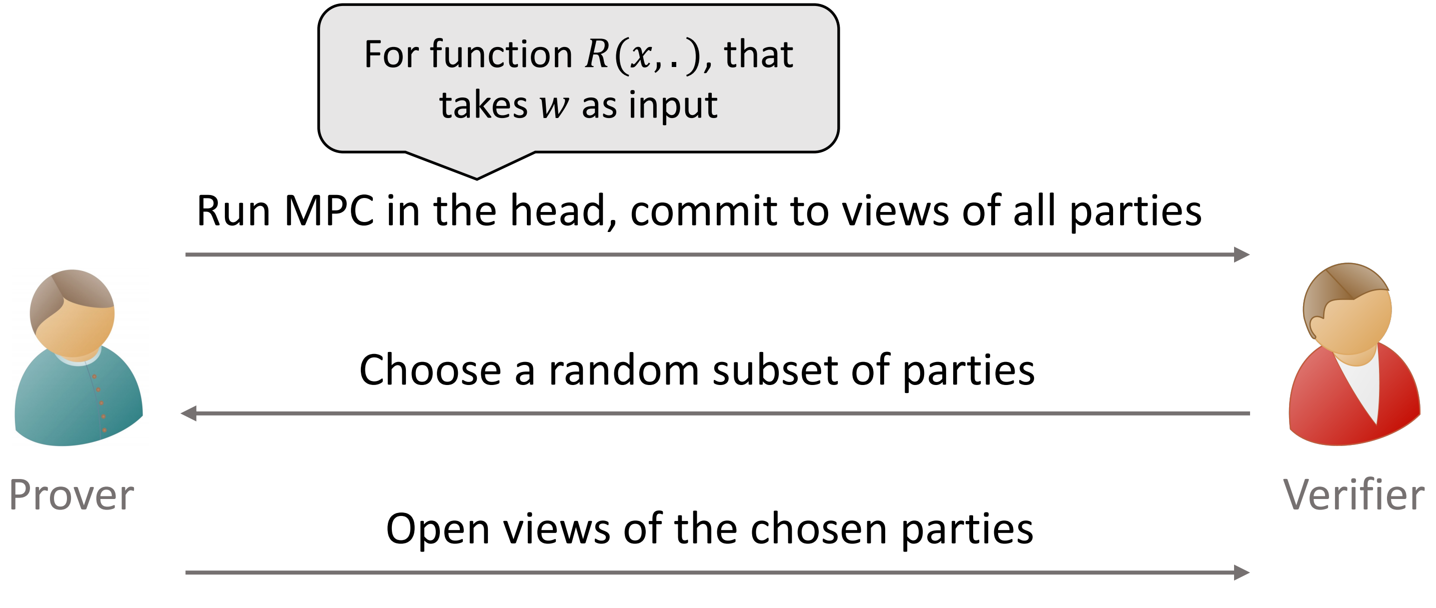
Simulator

Choose a random subset of parties

Simulate the views of these parties' using simulator of the underlying MPC protocol

Honestly commit to these views and garbage for the remaining parties' views

[IKOS07] is Stackable?



Simulator

Choose a random subset of parties

Simulate the views of these parties' using simulator of the underlying MPC protocol

Honestly commit to these views and garbage for the remaining parties' views

It is naturally EHVZK. What about recyclable third round messages?

F-Universally Simulatable MPC

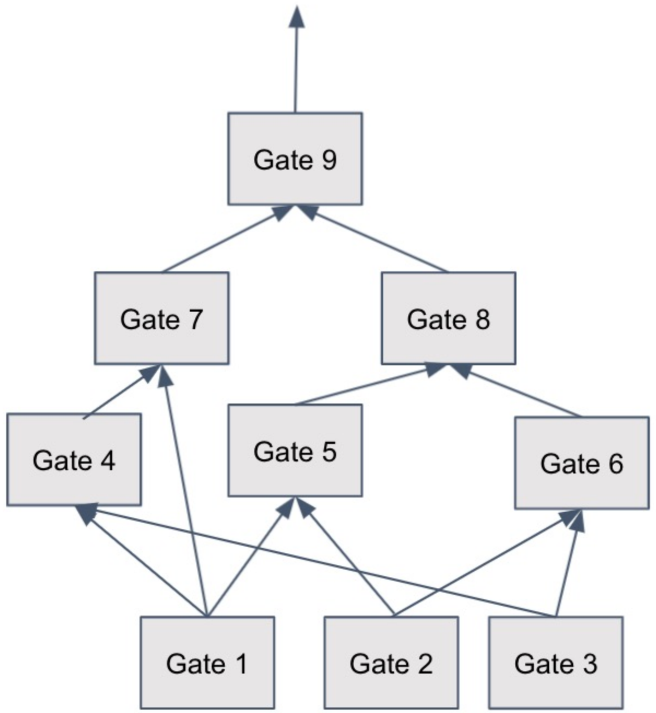
F-Universally Simulatable MPC

Adversary's view in many MPC protocols can be condensed and decoupled from the structure of the functionality being evaluated

F-Universally Simulatable MPC

Adversary's view in many MPC protocols can be condensed and decoupled from the structure of the functionality being evaluated

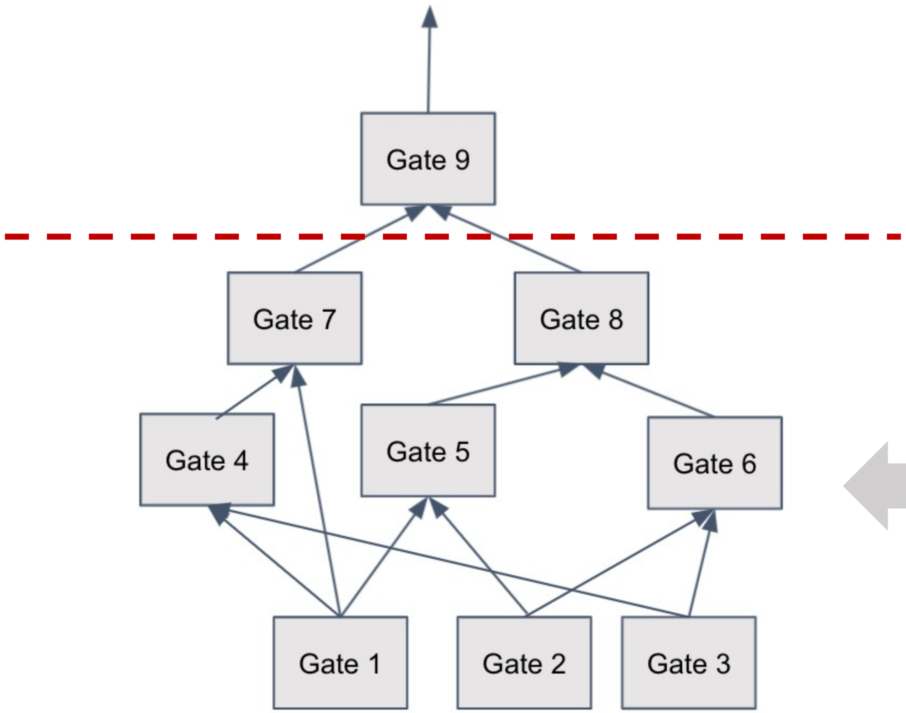
Example: Many secret sharing-based MPC (e.g. [BGW88])



F-Universally Simulatable MPC

Adversary's view in many MPC protocols can be condensed and decoupled from the structure of the functionality being evaluated

Example: Many secret sharing-based MPC (e.g. [BGW88])

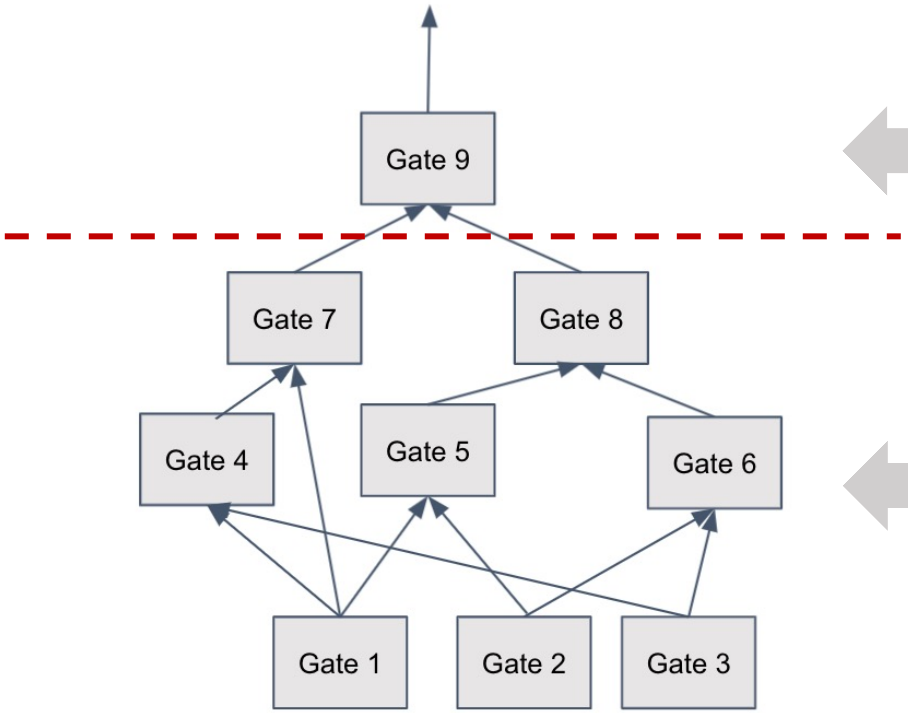


Simulator simulates random shares for the adversary for each of these gates

F-Universally Simulatable MPC

Adversary's view in many MPC protocols can be condensed and decoupled from the structure of the functionality being evaluated

Example: Many secret sharing-based MPC (e.g. [BGW88])



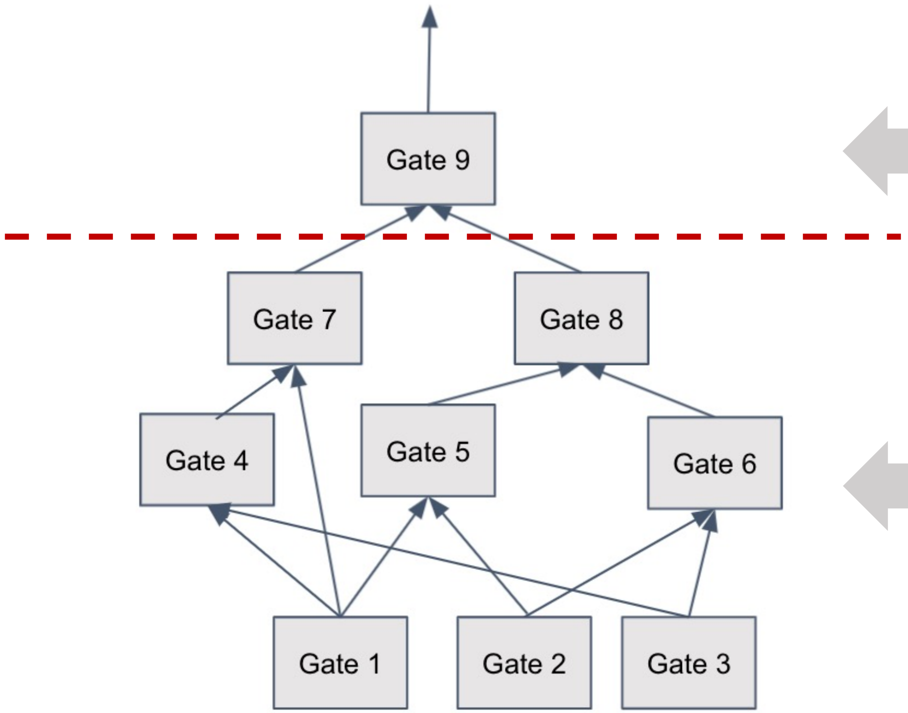
Given previously simulated shares and the output, simulate the final message

Simulator simulates random shares for the adversary for each of these gates

F-Universally Simulatable MPC

Adversary's view in many MPC protocols can be condensed and decoupled from the structure of the functionality being evaluated

Example: Many secret sharing-based MPC (e.g. [BGW88])



Given previously simulated shares and the output, simulate the final message

Deterministic computation

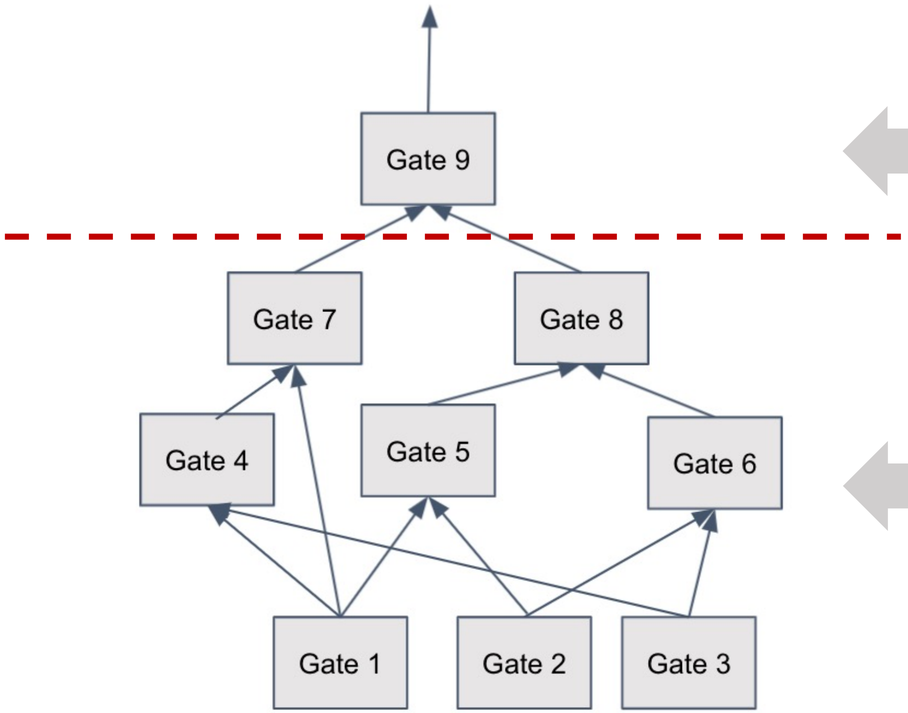
Simulator simulates random shares for the adversary for each of these gates

Independent of the function/circuit!

F-Universally Simulatable MPC

Adversary's view in many MPC protocols can be condensed and decoupled from the structure of the functionality being evaluated

Example: Many secret sharing-based MPC (e.g. [BGW88])



Given previously simulated shares and the output, simulate the final message

Expanded Views

Deterministic computation

Simulator simulates random shares for the adversary for each of these gates

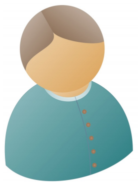
Condensed Views

Independent of the function/circuit!

Modified [IKOS07] for F-Universally Simulatable MPC

For function $R(x,.)$, that takes w as input

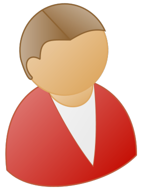
Run MPC in the head, commit to views of all parties



Prover

Choose a random subset of parties

Condensed views of the chosen parties and randomness used in corresponding commitments

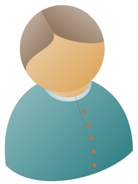


Verifier

Modified [IKOS07] for F-Universally Simulatable MPC

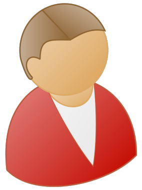
For function $R(x,.)$, that takes w as input

Run MPC in the head, commit to views of all parties



Prover

Choose a random subset of parties



Verifier

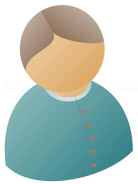
Condensed views of the chosen parties and randomness used in corresponding commitments

Verifier can expand condensed views assuming output is 1, check if commitments are valid and perform all other consistency checks

Modified [IKOS07] for F-Universally Simulatable MPC

For function $R(x, .)$, that takes w as input

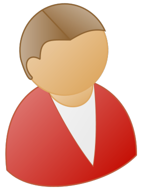
Run MPC in the head, commit to views of all parties



Prover

Choose a random subset of parties

Condensed views of the chosen parties and randomness used in corresponding commitments



Verifier

Since condensed views are independent of the functionality, this protocol now has recyclable third-round message

Verifier can expand condensed views assuming output is 1, check if commitments are valid and perform all other consistency checks

Disjunctions with Different Languages

$$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad \textbf{.....} \quad or \quad x_n \in L_n$$

Disjunctions with Different Languages

$$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad ..... \quad or \quad x_n \in L_n$$

$\boxed{\Sigma_1}$ $\boxed{\Sigma_2}$ $\boxed{\Sigma_n}$

Disjunctions with Different Languages

$$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad \ldots\ldots \quad or \quad x_n \in L_n$$

$\boxed{\Sigma_1}$ $\boxed{\Sigma_2}$ $\boxed{\Sigma_n}$

Sometimes same protocol works for different languages

Disjunctions with Different Languages

$x_1 \in L_1 \quad or \quad x_2 \in L_2 \quad or \quad \ldots\ldots \quad or \quad x_n \in L_n$

$\boxed{\Sigma_1}$ $\boxed{\Sigma_2}$ $\boxed{\Sigma_n}$

Sometimes same protocol works for different languages

If third round messages are over different fields/rings – represent as bits and see what parts can be re-used

Thank You!