



# The Power of NAPs

Compressing OR-Proofs via Collision-Resistant Hashing

Katharina Boudgoust & Mark Simkin

# $\Sigma$ -Protocols

---



Prover



Verifier

# $\Sigma$ -Protocols

---



Prover

$x \in L?$

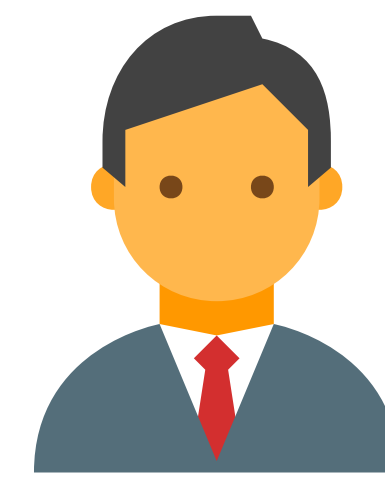
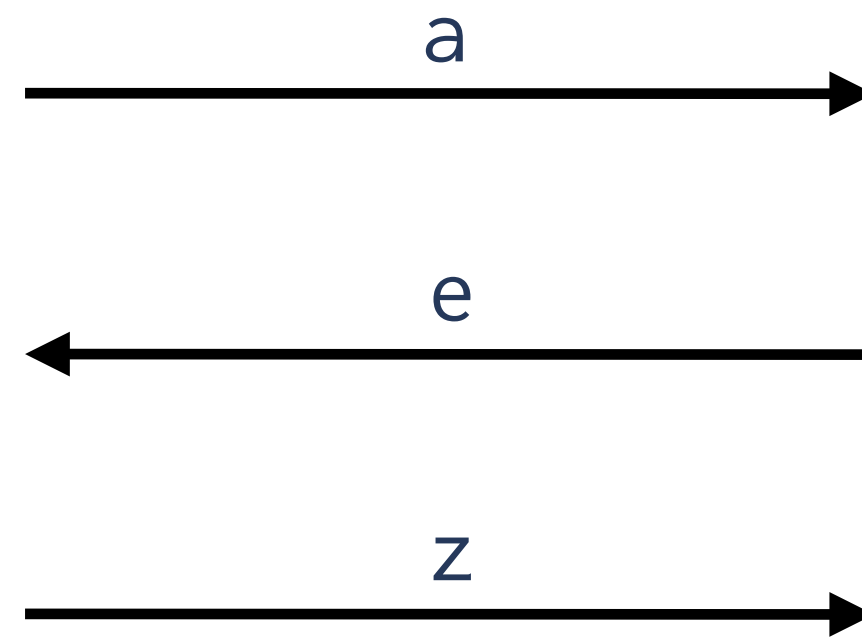


Verifier

# $\Sigma$ -Protocols



Prover

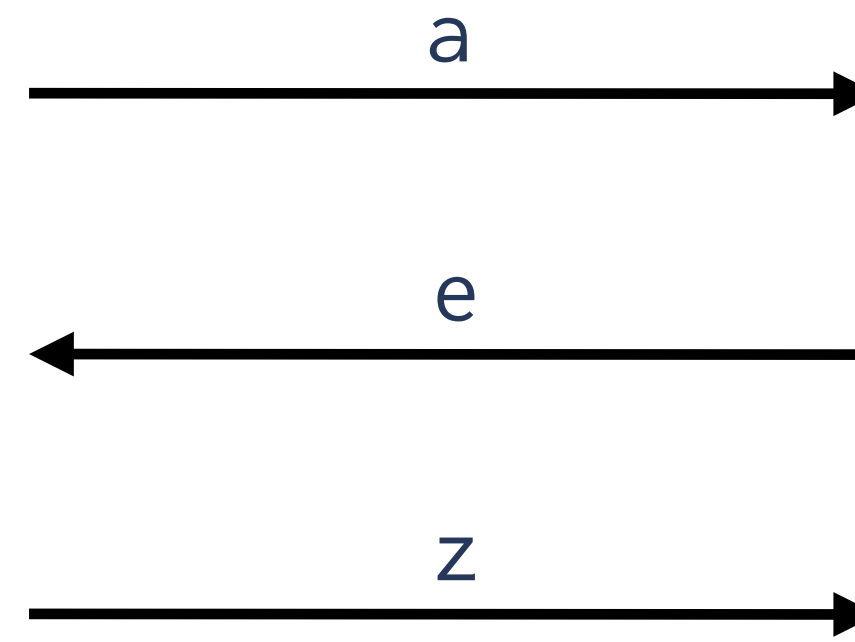


Verifier

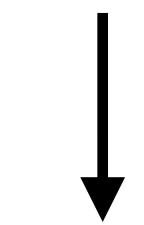
# $\Sigma$ -Protocols



Prover



Verifier

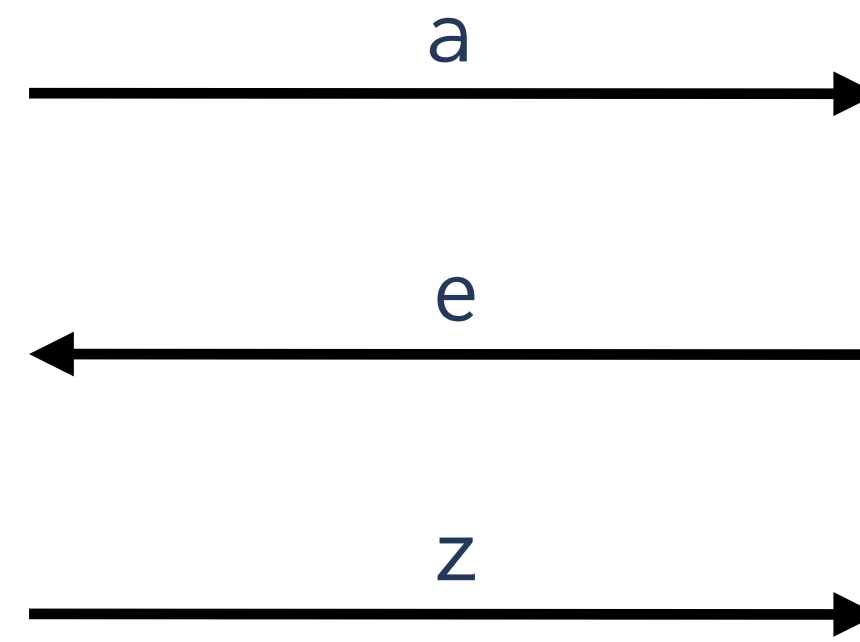


0/1

# $\Sigma$ -Protocols



Prover

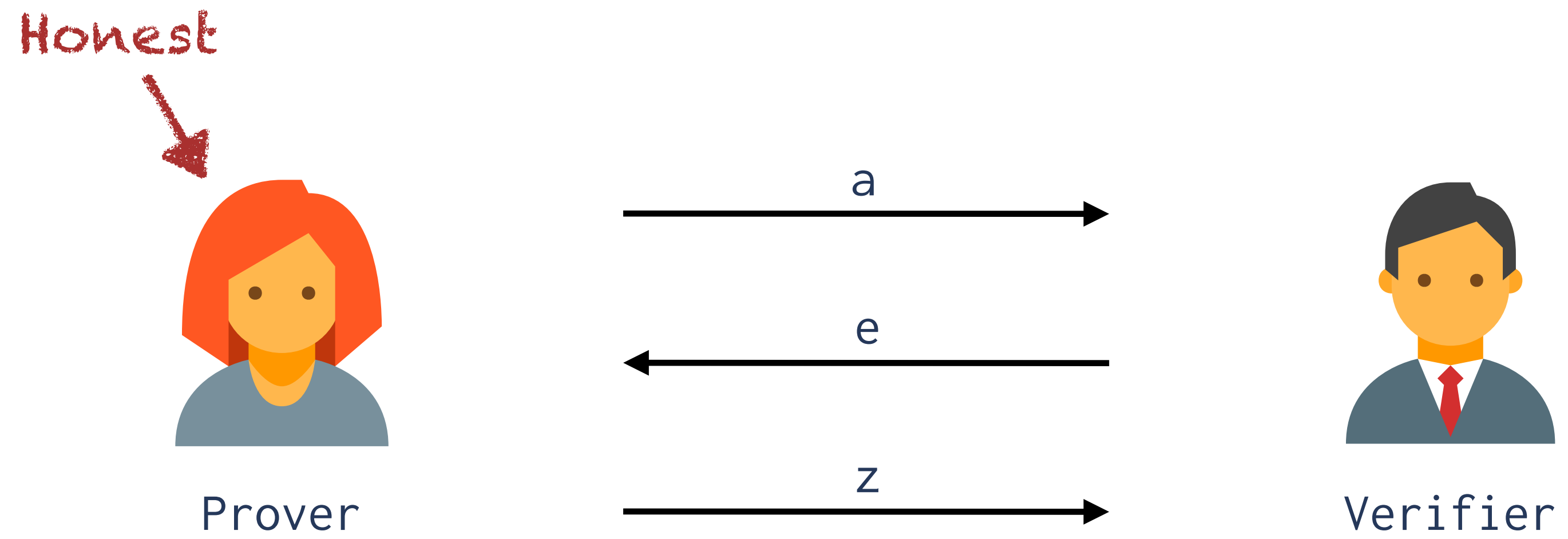


Verifier



Completeness

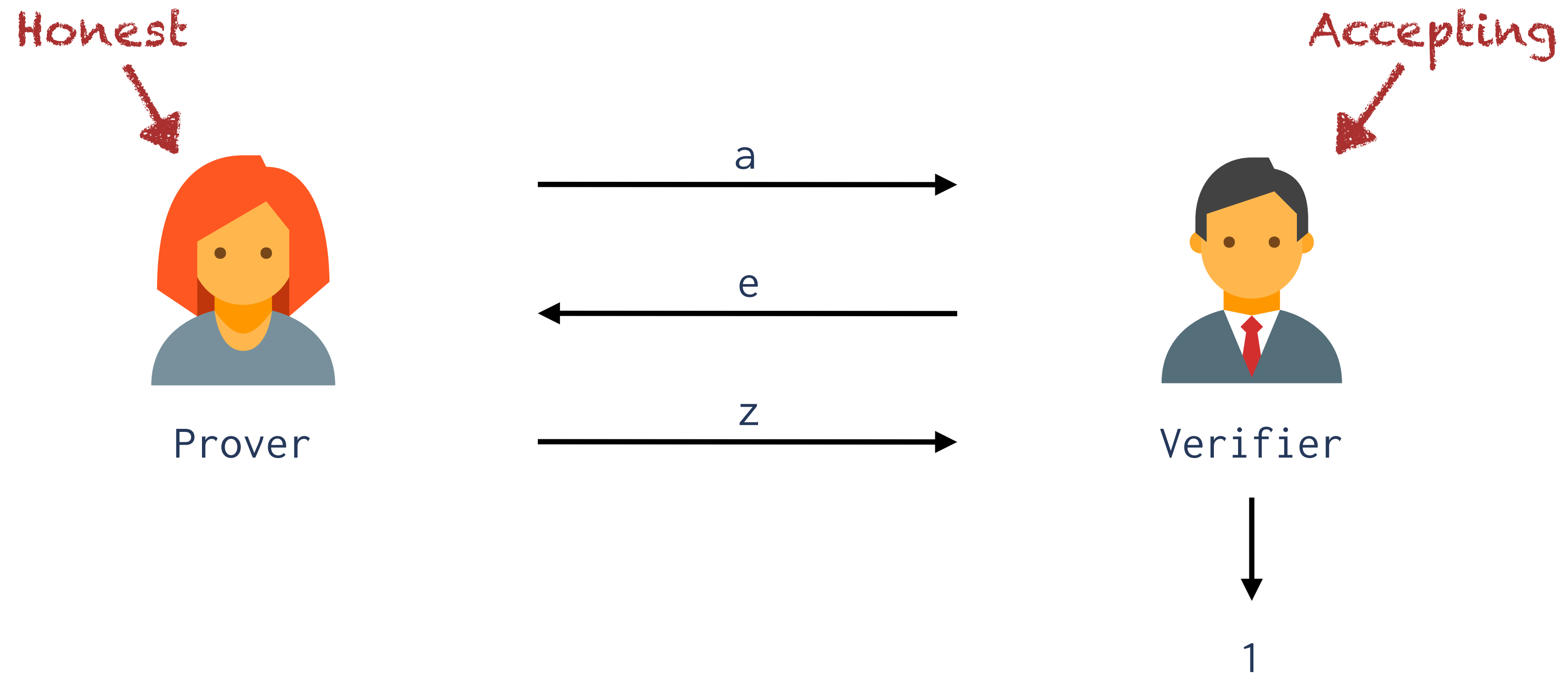
# $\Sigma$ -Protocols



 Completeness



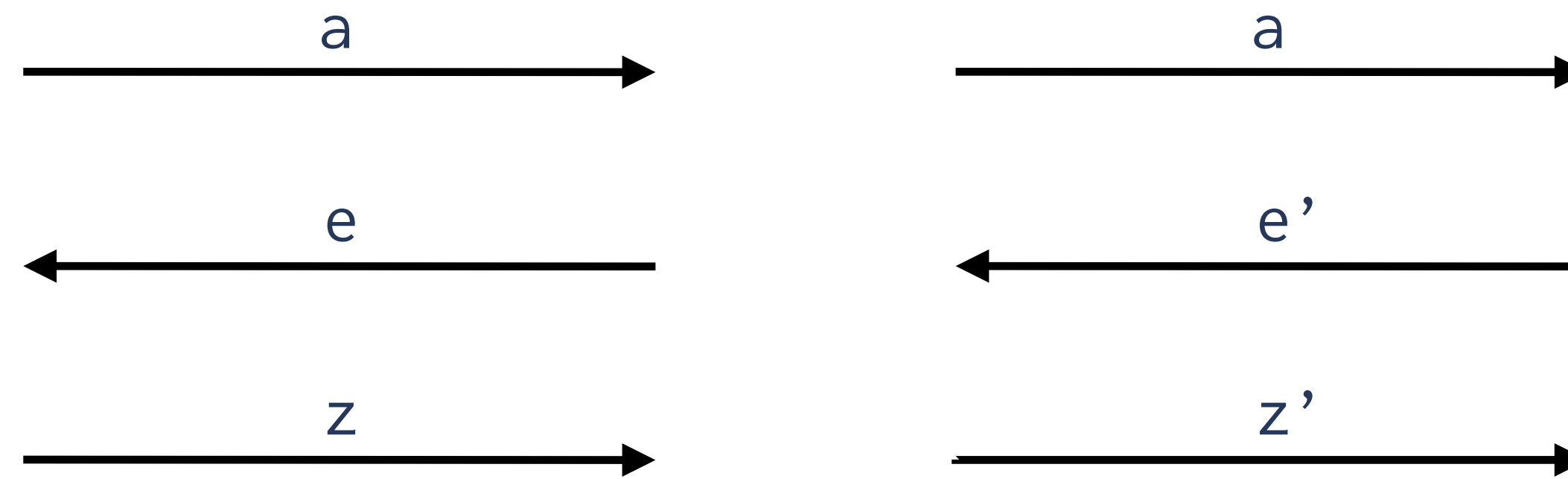
# $\Sigma$ -Protocols



 Completeness



# $\Sigma$ -Protocols

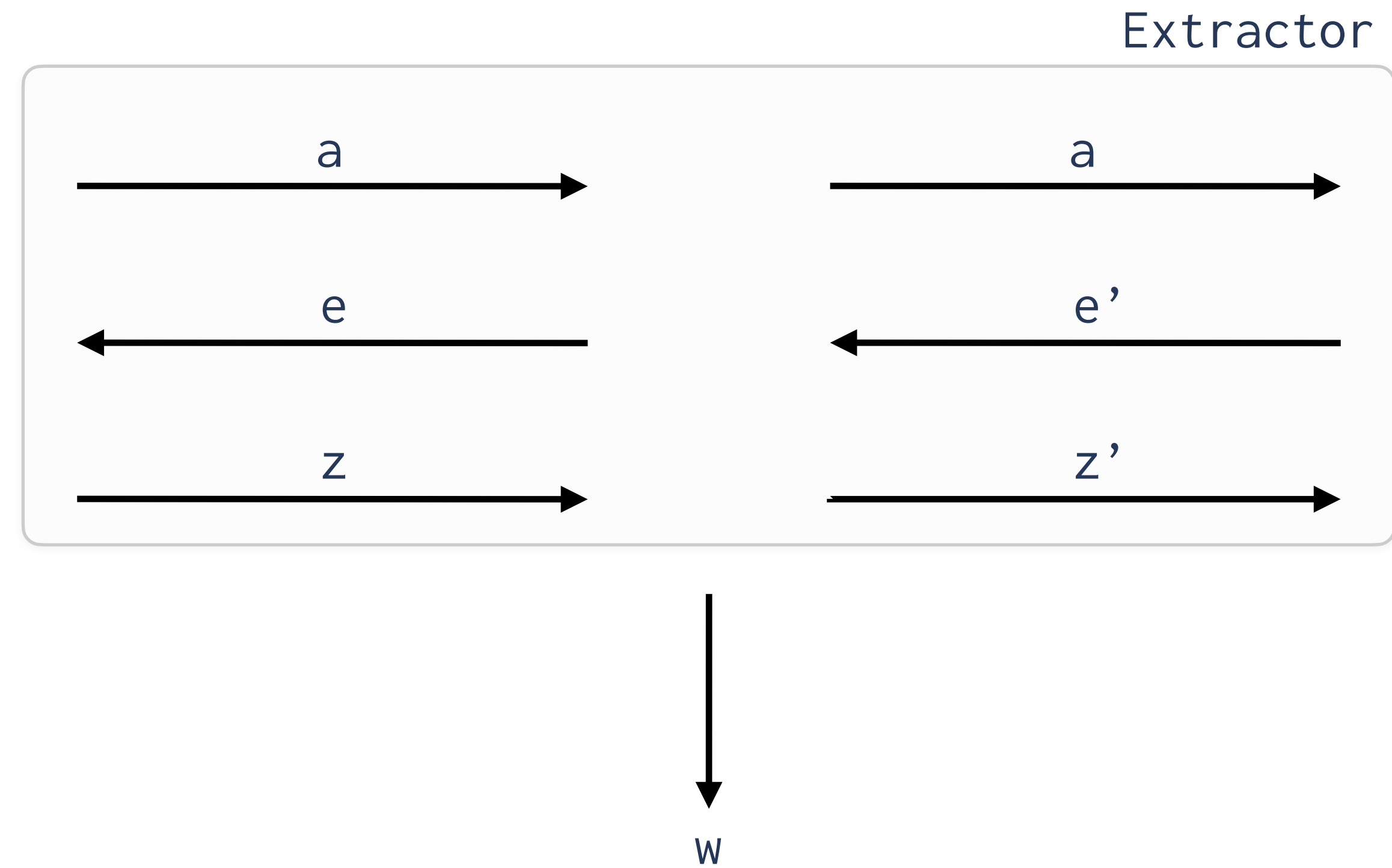


Completeness



Special soundness

# $\Sigma$ -Protocols



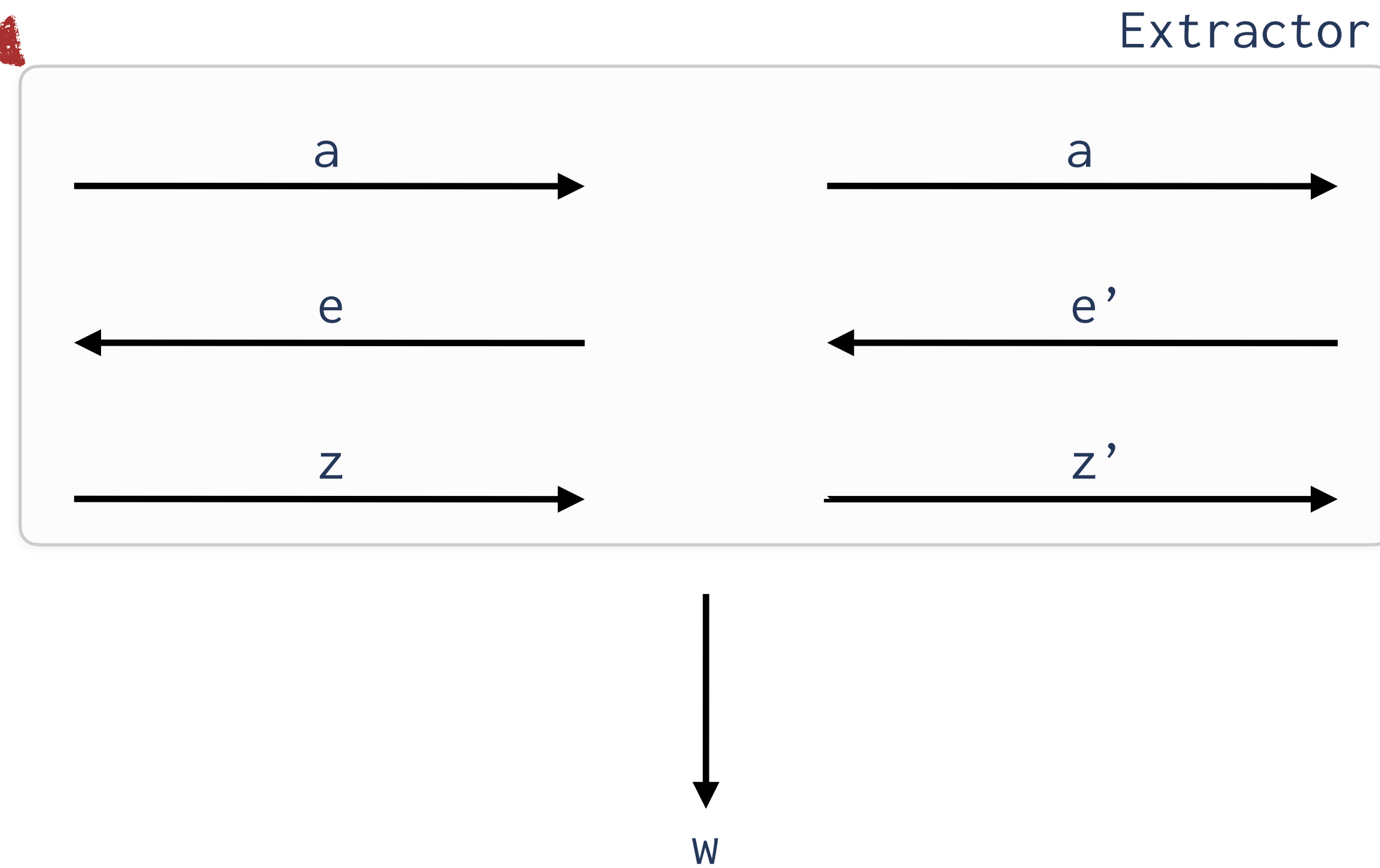
Completeness



Special soundness

# $\Sigma$ -Protocols

Accepting



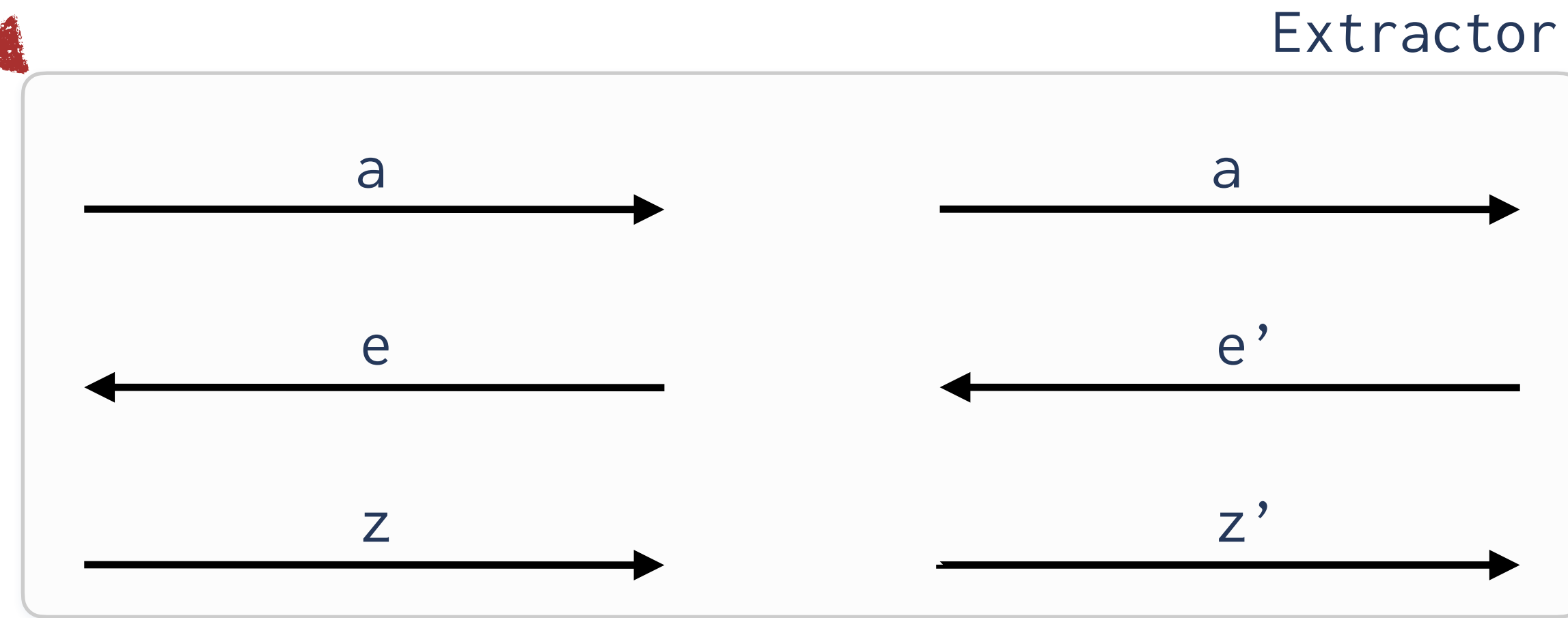
Completeness



Special soundness

# $\Sigma$ -Protocols

Accepting



w



$(x, w) \in R$



Completeness



Special soundness

# $\Sigma$ -Protocols

---



Simulator



Completeness



Special soundness



Honest verifier zero-knowledge

# $\Sigma$ -Protocols



Completeness



Special soundness



Honest verifier zero-knowledge

# $\Sigma$ -Protocols



Indistinguishable from real execution



Completeness



Special soundness






Honest verifier zero-knowledge

# $\Sigma$ -Protocols

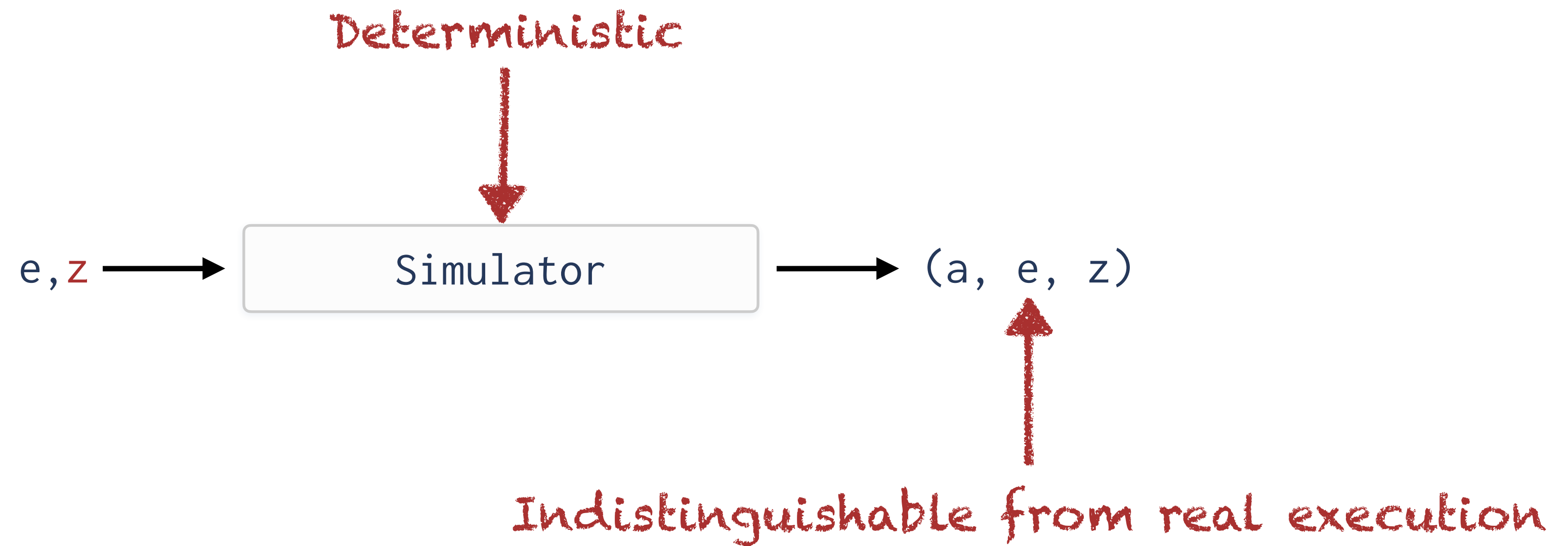


*Indistinguishable from real execution*

-  Completeness
-  Special soundness
-  Strong honest verifier zero-knowledge



# $\Sigma$ -Protocols



Completeness



Special soundness



Strong honest verifier zero-knowledge

# Disjunctive Statements

---



Prover

Is  $x_1 \in L$  or  $x_2 \in L$ ?



Verifier

# Disjunctive Statements



Prover

Is  $x_1 \in L$  or  $x_2 \in L$ ?



Verifier

- Prove that one out of several statements is true

# Disjunctive Statements



Prover

Is  $x_1 \in L$  or  $x_2 \in L$ ?



Verifier

- Prove that one out of several statements is true
- Specific enough to be solved more efficiently

# Disjunctive Statements



Prover

Is  $x_1 \in L$  or  $x_2 \in L$ ?

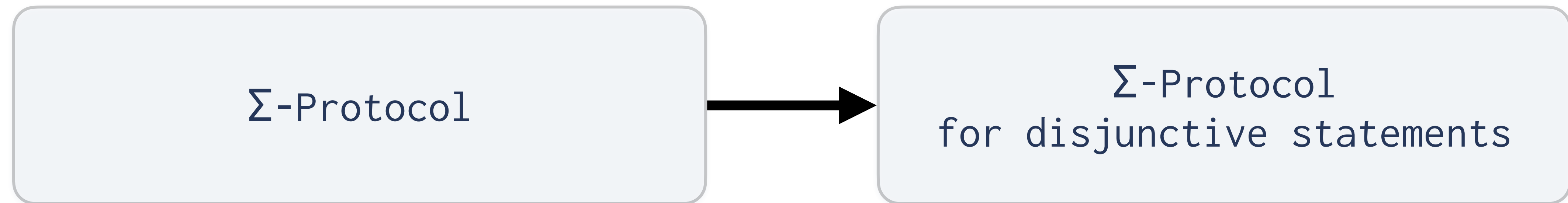


Verifier

- Prove that one out of several statements is true
- Specific enough to be solved more efficiently
- General enough to be useful (ring signatures, electronic voting, ...)

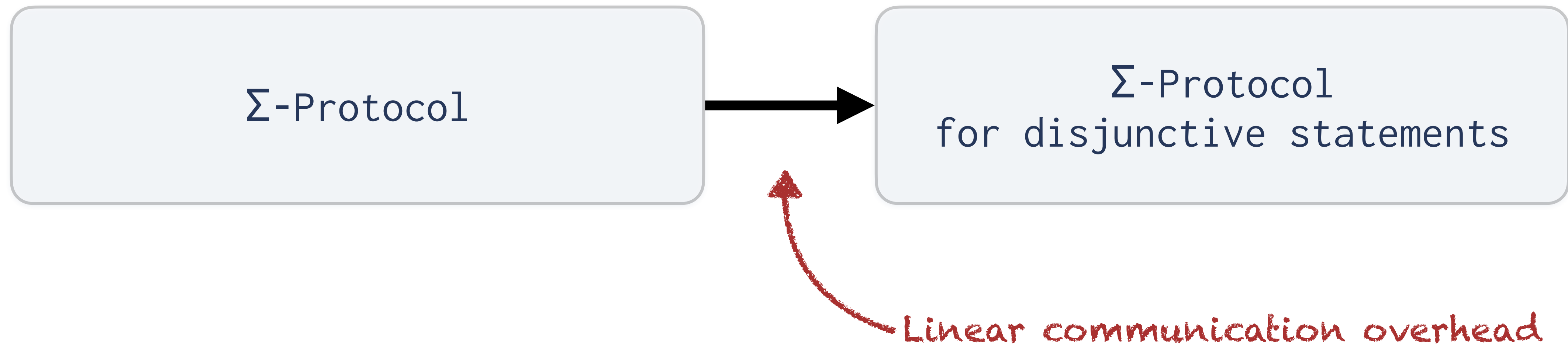
# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]

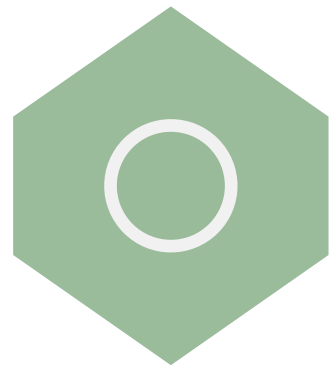


# How Much Communication Needed?





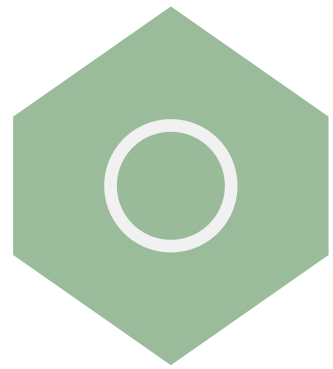
# How Much Communication Needed?



From structured hardness assumptions

Small communication complexity, but stronger assumptions

# How Much Communication Needed?



From structured hardness assumptions

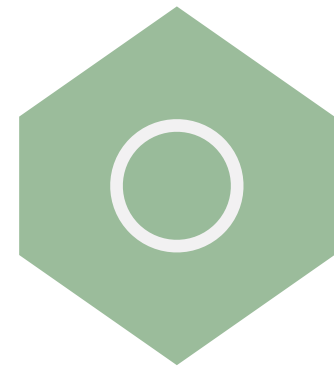
Small communication complexity, but stronger assumptions



Via MPC-in-the-head

Unstructured assumptions, but large communication complexity

# How Much Communication Needed?



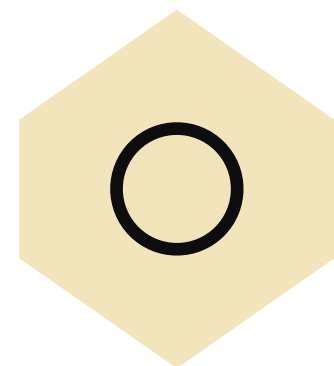
From structured hardness assumptions

Small communication complexity, but stronger assumptions



Via MPC-in-the-head

Unstructured assumptions, but large communication complexity

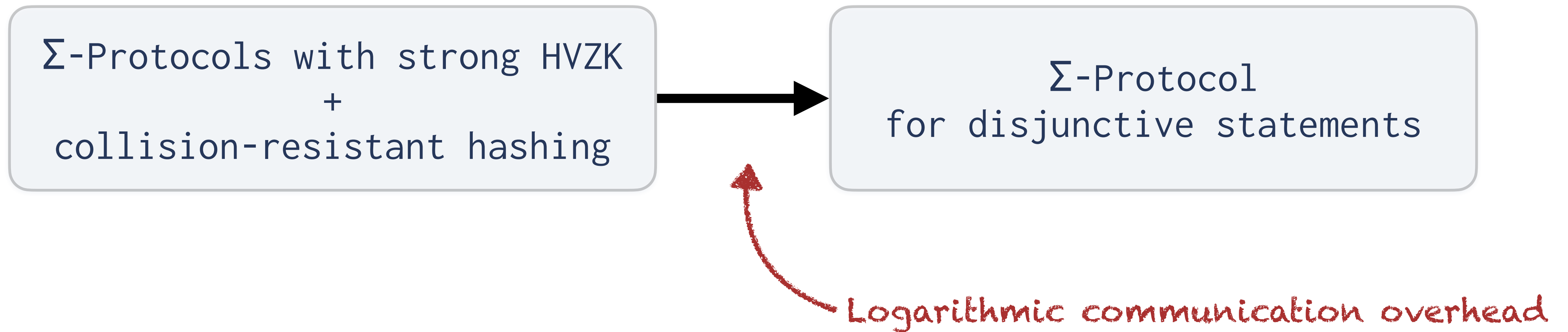


Via PCPs/IOPs and collision-resistant hashing

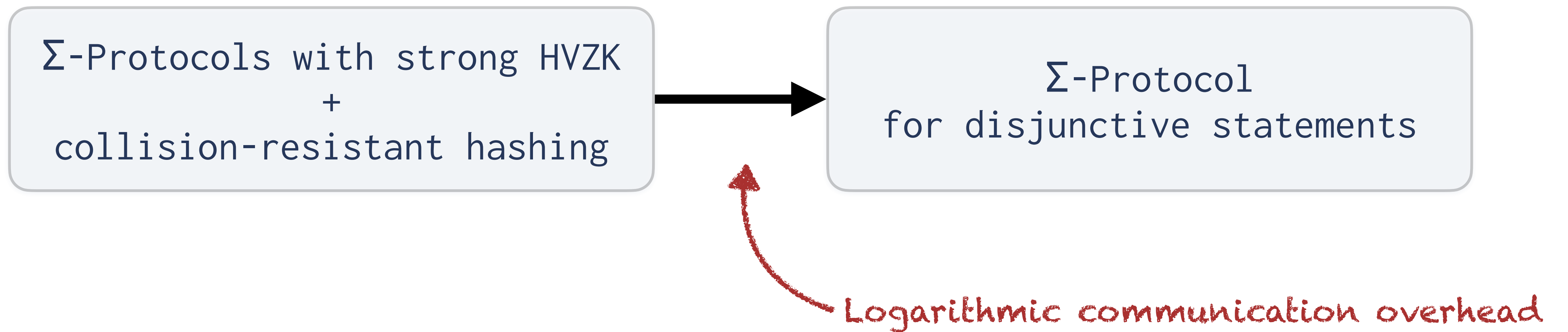
Complex, heavy machinery with polylog communication

# Our Contribution

---

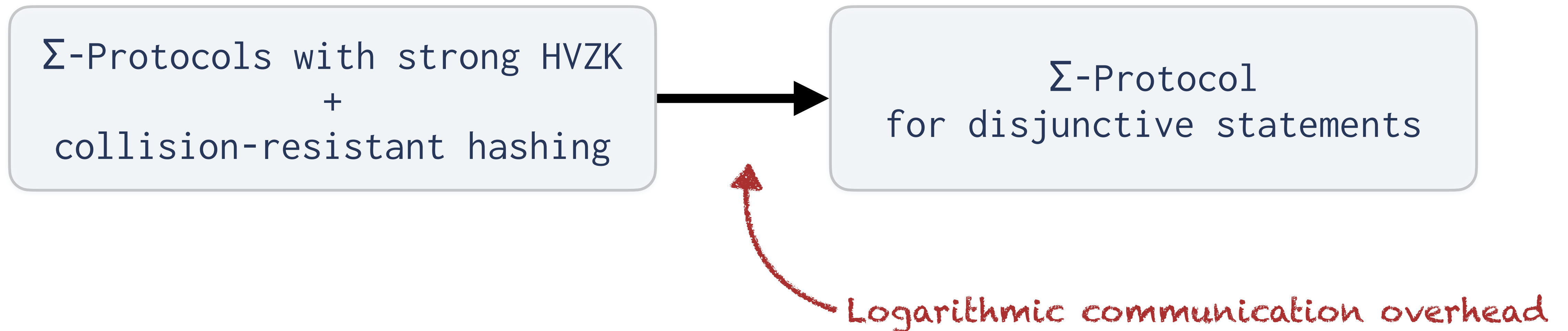


# Our Contribution



- ◆ New Notion of non-adaptively programmable functions (NAPs)

# Our Contribution



- ◆ New Notion of non-adaptively programmable functions (NAPs)
- ◆ Rejection sampling can be explained

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover



Verifier

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover



Verifier

honest   $a_1$



# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover



Verifier

*honest* →  $a_1$

*simulated* →

$(a_2, e_2, z_2)$   
⋮  
 $(a_n, e_n, z_n)$

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover

$a_1, \dots, a_n$



Verifier

*honest* →  $a_1$

*simulated* →

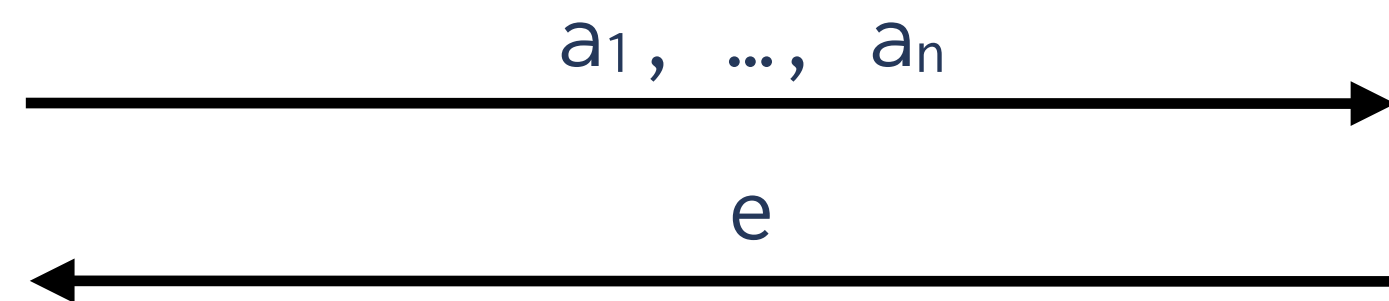
$(a_2, e_2, z_2)$   
⋮  
 $(a_n, e_n, z_n)$

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover



Verifier

*honest* →  $a_1$

*simulated* →

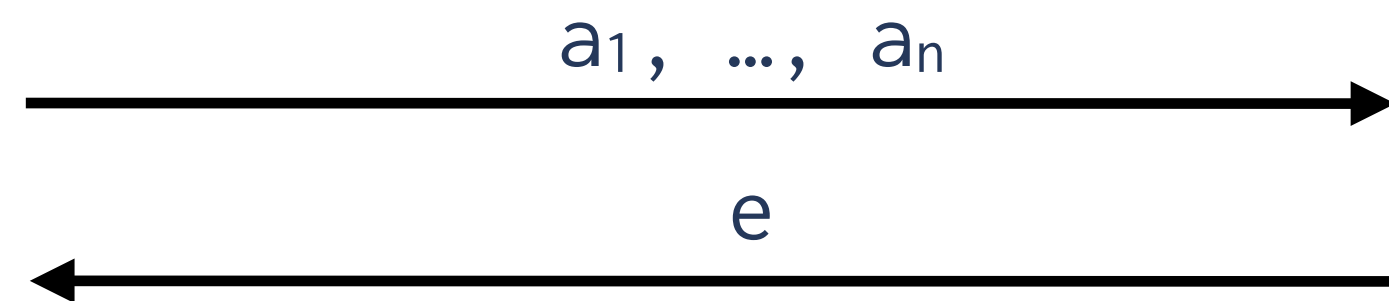
$(a_2, e_2, z_2)$   
 $\vdots$   
 $(a_n, e_n, z_n)$

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover



Verifier

*honest* →  $a_1$

*simulated* →

$(a_2, e_2, z_2)$   
 $\vdots$   
 $(a_n, e_n, z_n)$

$$e = e_1 + \dots + e_n$$

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover

$a_1, \dots, a_n$



$e$



Verifier

*honest* →  $a_1$

*simulated* →

$(a_2, e_2, z_2)$   
 $\vdots$   
 $(a_n, e_n, z_n)$

$$e = e_1 + \dots + e_n$$

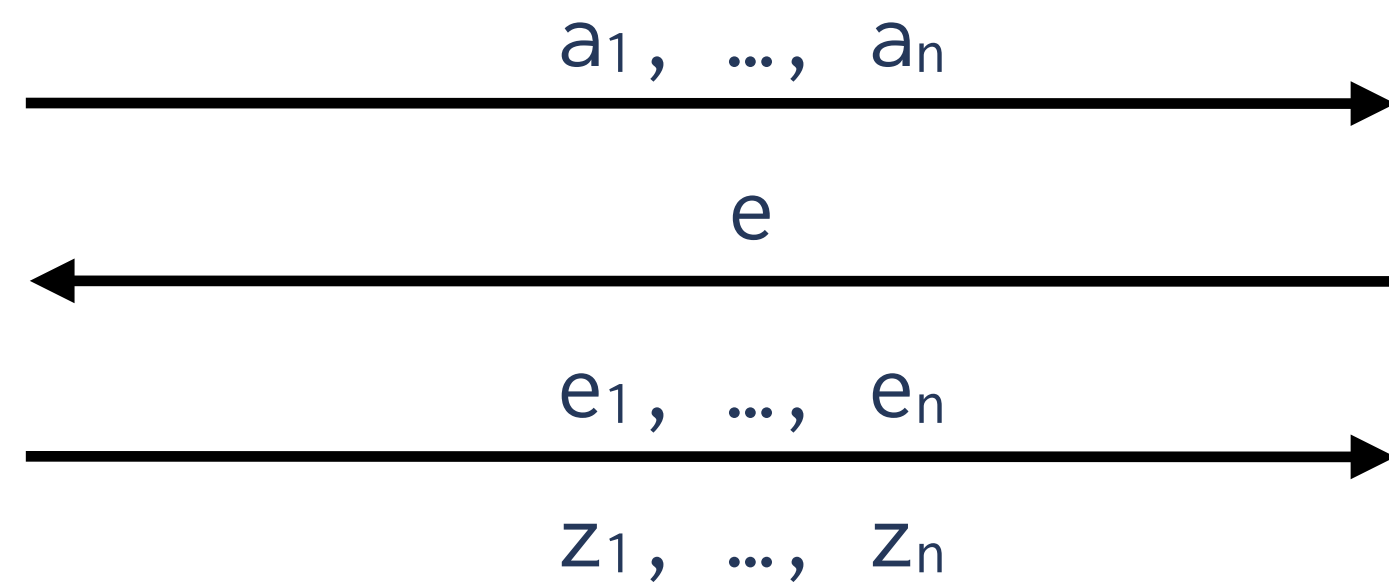
*honest* →  $z_1$

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover



Verifier

*honest* →  $a_1$

*simulated* →

$(a_2, e_2, z_2)$   
⋮  
 $(a_n, e_n, z_n)$

$$e = e_1 + \dots + e_n$$

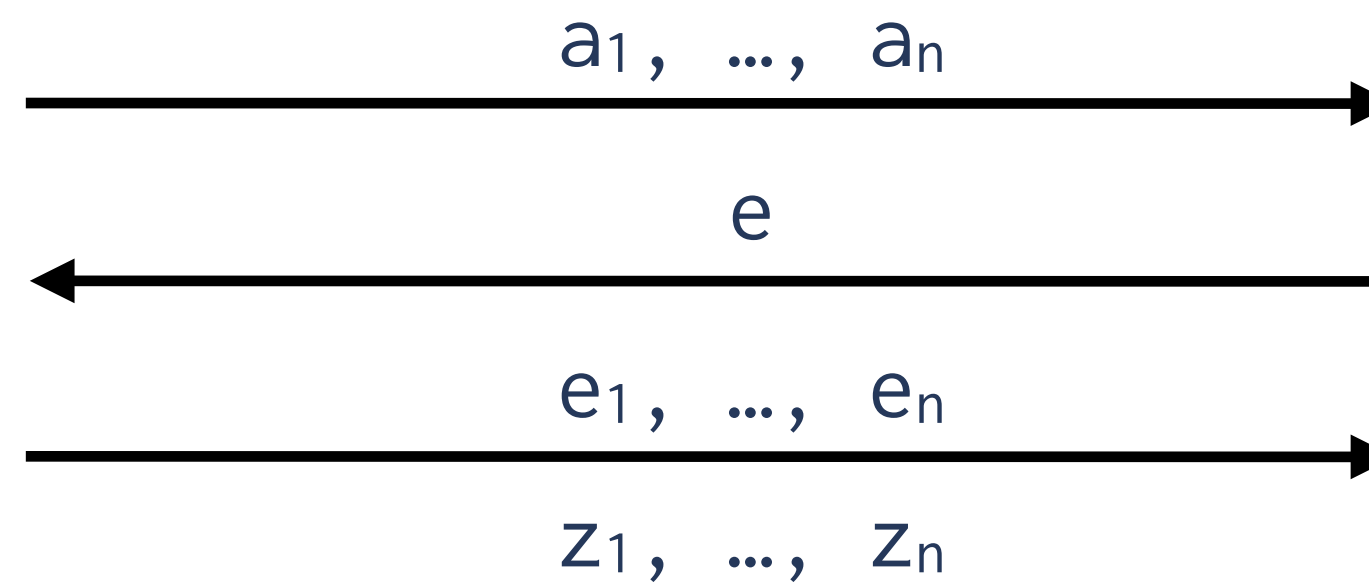
*honest* →  $z_1$

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover



Verifier

*honest* →  $a_1$

*simulated* →

$(a_2, e_2, z_2)$   
 $\vdots$   
 $(a_n, e_n, z_n)$

$$e = e_1 + \dots + e_n$$

*honest* →  $z_1$

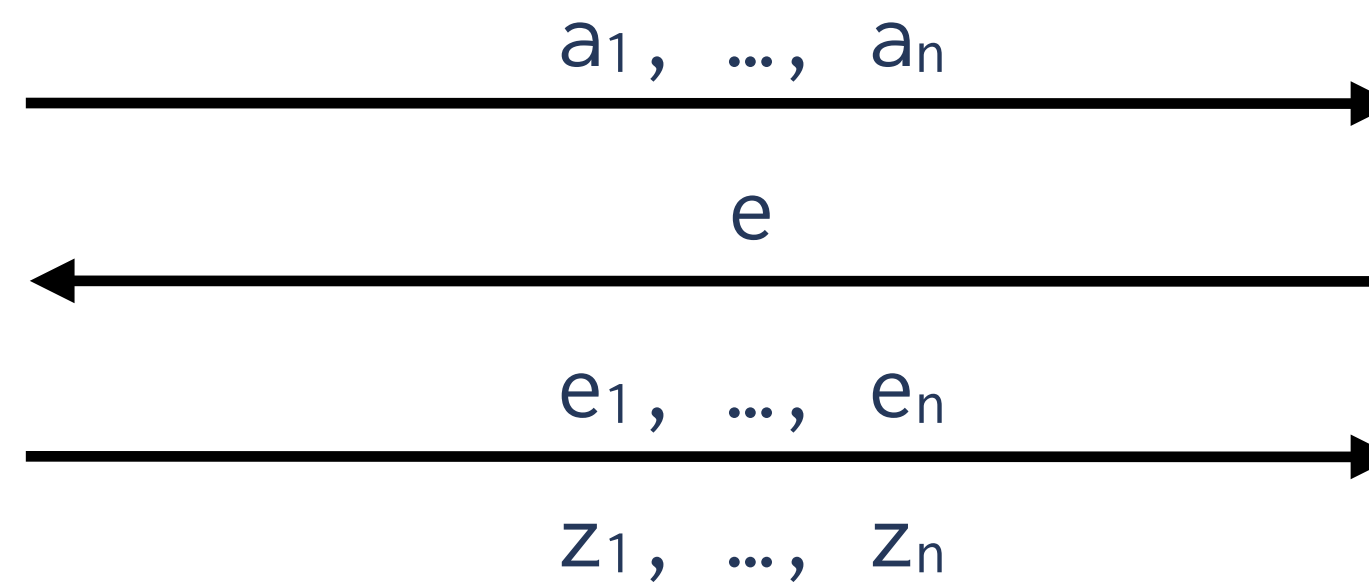
$(a_1, e_1, z_1)$   
 $\vdots$   
 $(a_n, e_n, z_n)$

# Constructing OR-Proofs

[Cramer, Damgård, Schoenmakers 1994]



Prover



Verifier

*honest* →  $a_1$

*simulated* →

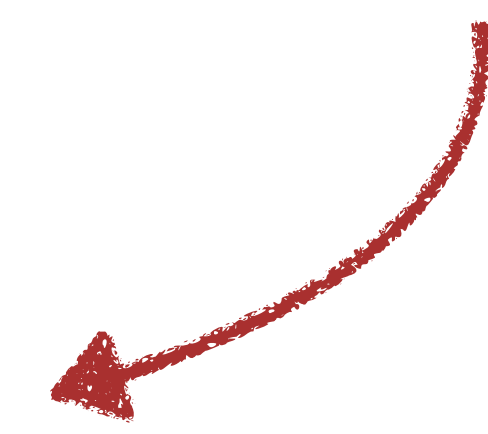
$(a_2, e_2, z_2)$   
⋮  
 $(a_n, e_n, z_n)$

$$e = e_1 + \dots + e_n$$

*honest* →  $z_1$

*Check each transcript*

$(a_1, e_1, z_1)$   
⋮  
 $(a_n, e_n, z_n)$





# Constructing OR-Proofs

## Our Construction



Prover

$a_1, \dots, a_n$

$e$

$e_1, \dots, e_n$

$z_1, \dots, z_n$



Verifier

$a_1$

$(a_2, e_2, z_2)$   
 $\vdots$   
 $(a_n, e_n, z_n)$

$(a_1, e_1, z_1)$

$\vdots$

$(a_n, e_n, z_n)$

$$e = e_1 + \dots + e_n$$

$z_1$

# Constructing OR-Proofs

## Our Construction



Prover

$a_1, \dots, a_n$

$e$

$e_1, \dots, e_n$

$z_1, \dots, z_n$



Verifier

Just send hash

$a_1$

$(a_2, e_2, z_2)$   
 $\vdots$   
 $(a_n, e_n, z_n)$

$(a_1, e_1, z_1)$   
 $\vdots$   
 $(a_n, e_n, z_n)$

$$e = e_1 + \dots + e_n$$

$z_1$

# Constructing OR-Proofs

Our Construction



Prover

$a_1, \dots, a_n$

$e$

$e_1, \dots, e_n$

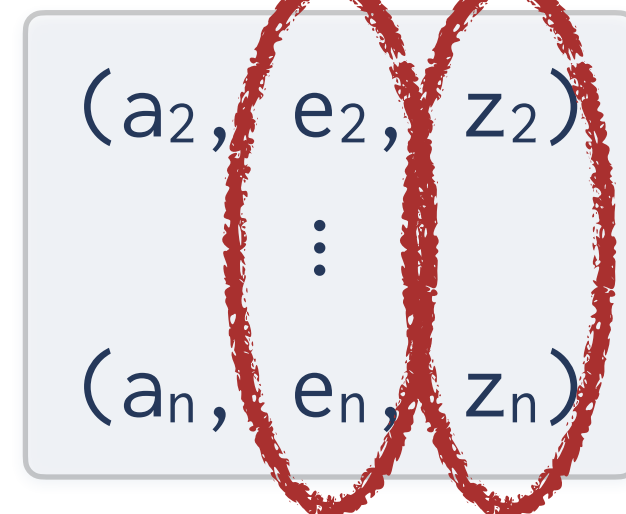
$z_1, \dots, z_n$



Verifier

Just send hash

$a_1$



Pick pseudorandom

$(a_1, e_1, z_1)$

$\vdots$

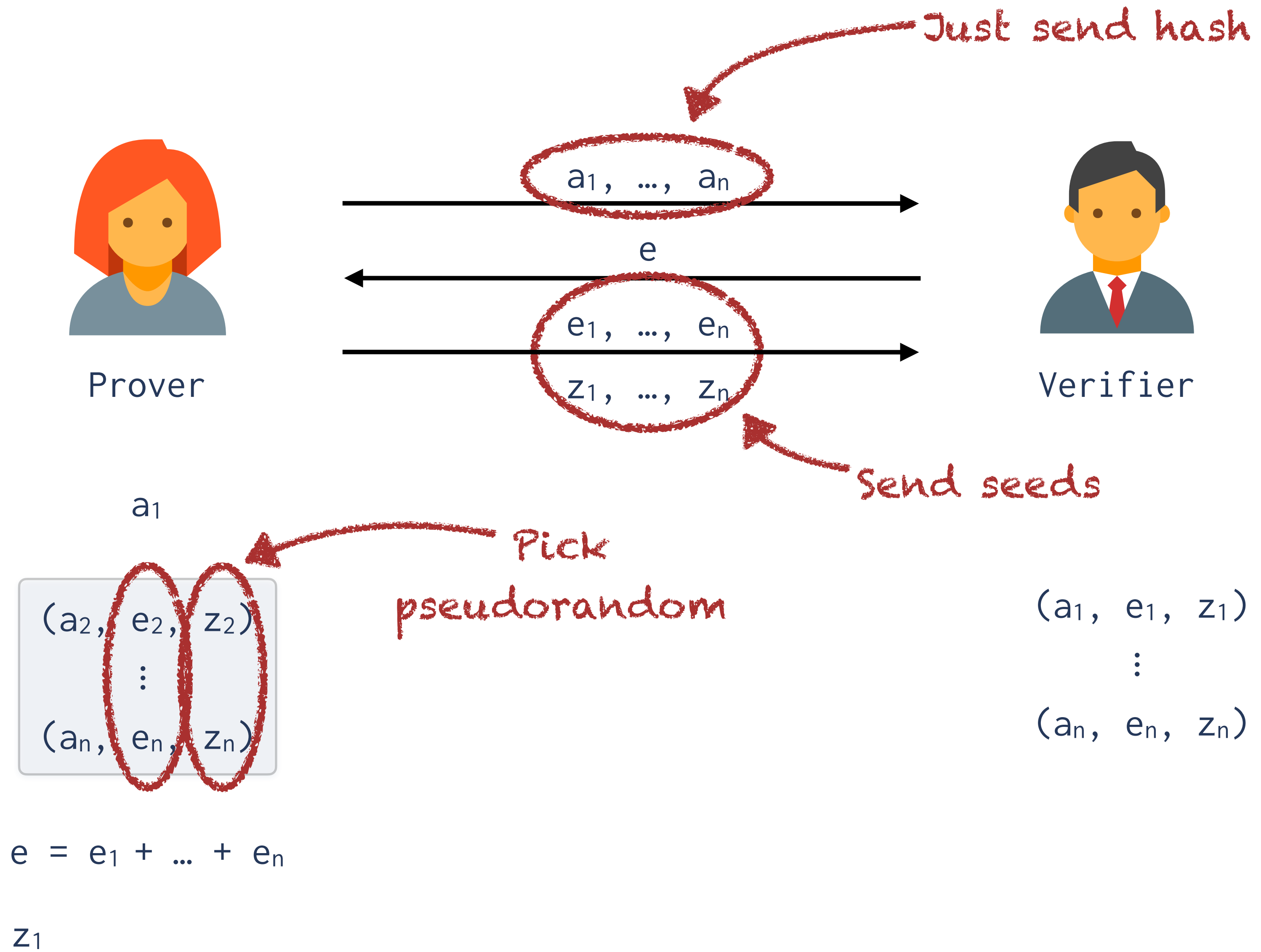
$(a_n, e_n, z_n)$

$$e = e_1 + \dots + e_n$$

$z_1$

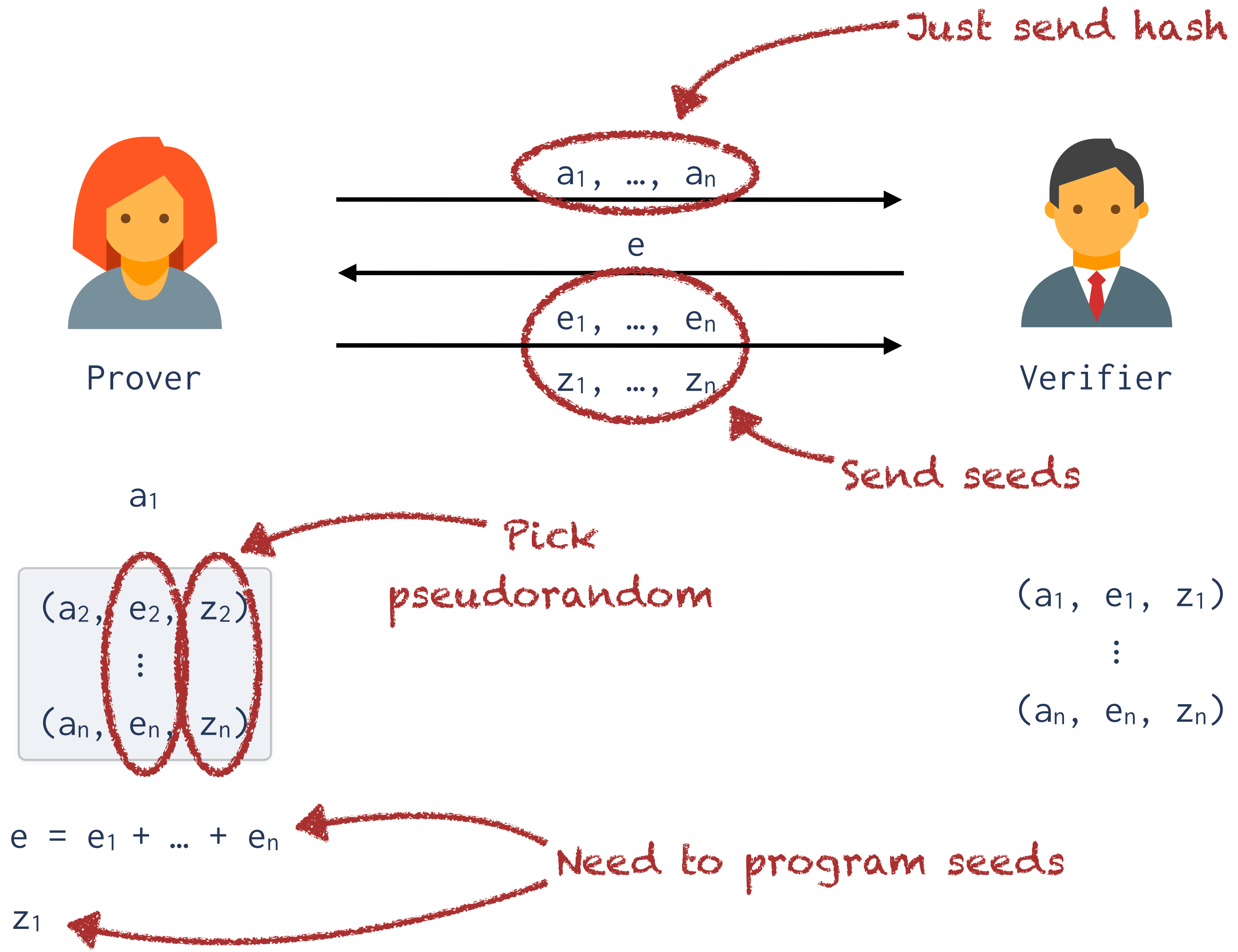
# Constructing OR-Proofs

## Our Construction



# Constructing OR-Proofs

## Our Construction



# Non-Adaptively Programmable Functions

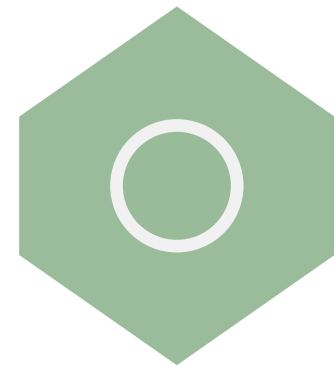
---

NAPs

# Non-Adaptively Programmable Functions

---

NAPs

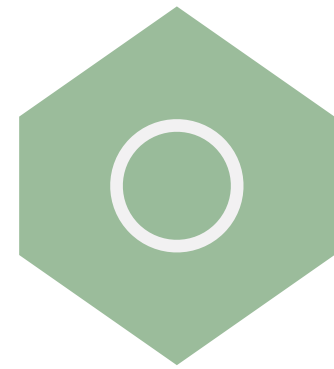


Privately programmable PRFs are hard

Require iO or FHE

# Non-Adaptively Programmable Functions

NAPs



Privately programmable PRFs are hard

Require IO or FHE



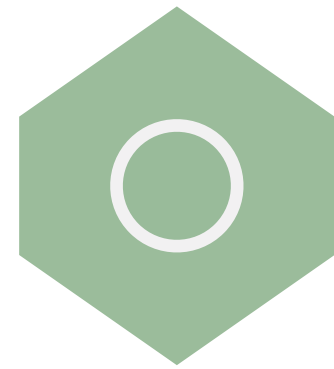
Know programming location during key generation

Massively simplifies the problem



# Non-Adaptively Programmable Functions

NAPs



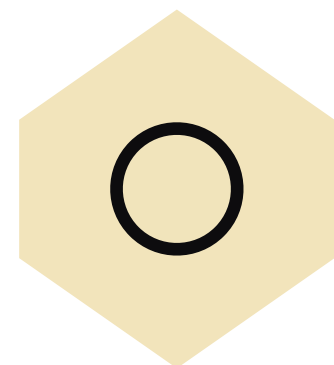
Privately programmable PRFs are hard

Require IO or FHE



Know programming location during key generation

Massively simplifies the problem



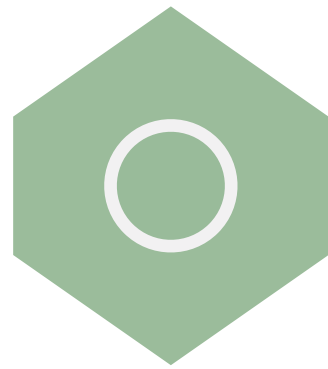
Can be constructed from one-way functions

Via distributed point functions

# Non-Adaptively Programmable Functions

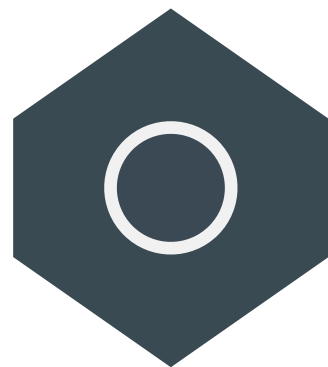
NAPs

31



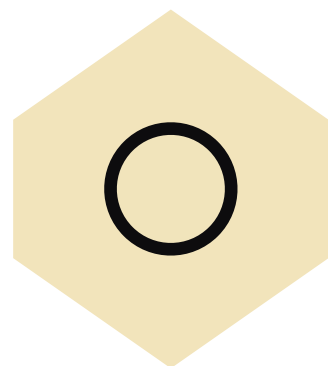
Privately programmable PRFs are hard

Require iO or FHE



Know programming location during key generation

Massively simplifies the problem



Can be constructed from one-way functions

Via distributed point functions (and explainable samplers)



Questions?