Does Fiat-Shamir Require a Cryptographic Hash Function?

Yilei Chen (Tsinghua University)

Alex Lombardi (MIT)

Fermi Ma (Princeton → Simons Institute + UC Berkeley)

Willy Quach (Northeastern)

(Public-Coin) Interactive Protocols

[GMR85, B85]

P

V

(Public-Coin) Interactive Protocols

[GMR85, B85]

$x \in 3\mathrm{SAT}$

(Public-Coin) Interactive Protocols

[GMR85, B85]

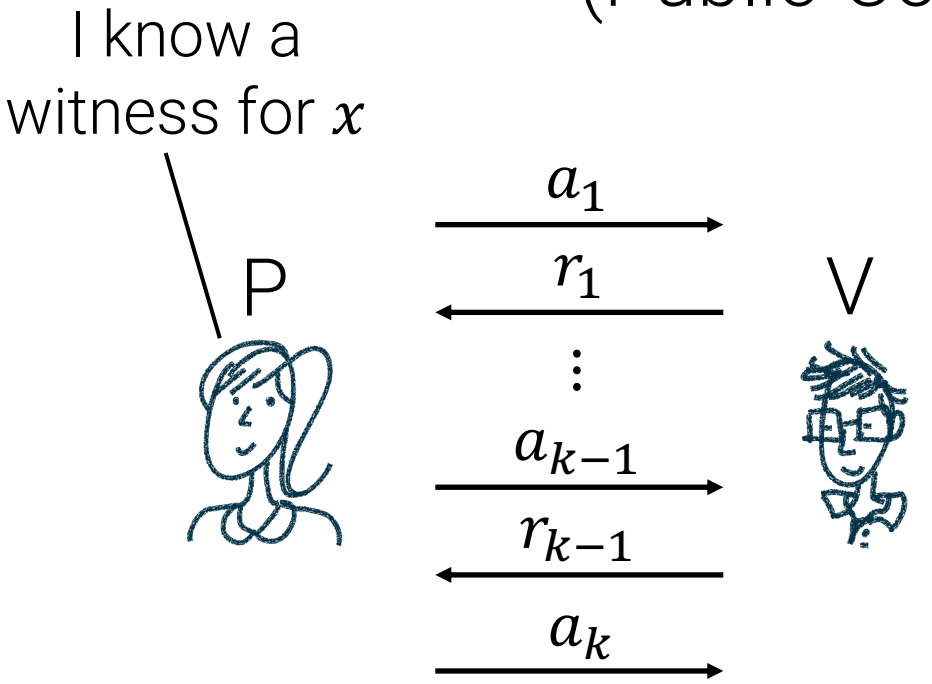
I know a witness for x

P

V

(Public-Coin) Interactive Protocols

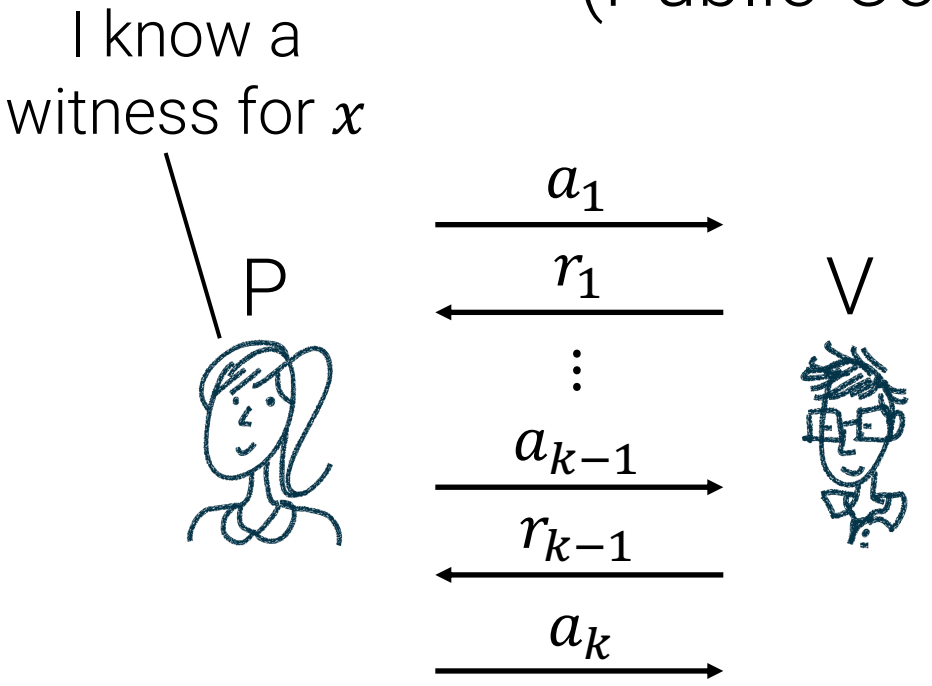
[GMR85, B85]



Public coin: r_i uniformly random

(Public-Coin) Interactive Protocols

[GMR85, B85]



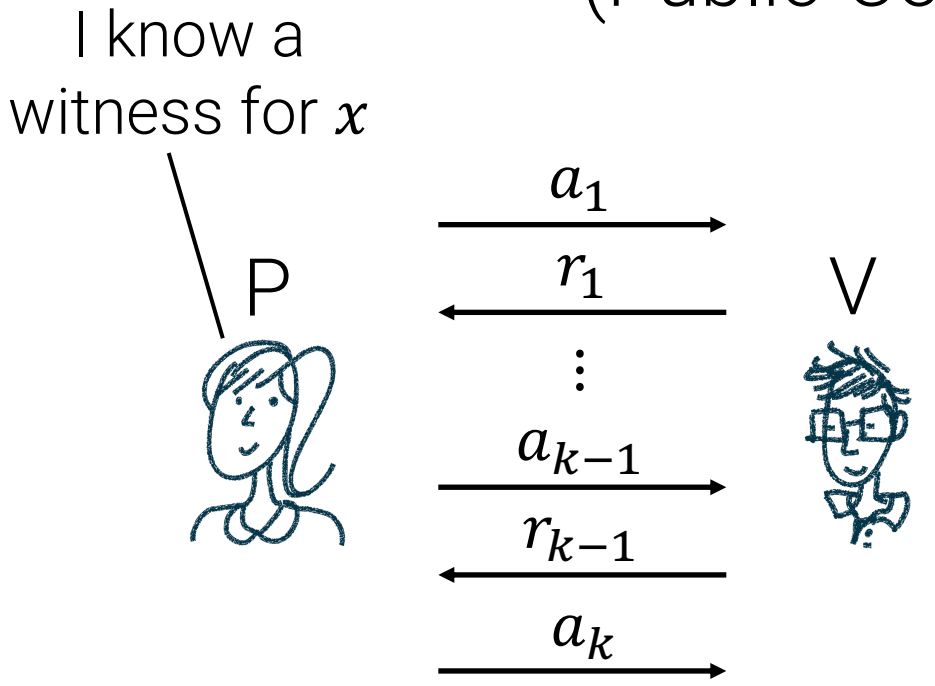
Public coin: r_i uniformly random

Completeness: If statement is true, verifier accepts w/ probability 1.

Soundness: If statement is false, verifier rejects w/ high probability, no matter what prover does.

(Public-Coin) Interactive Protocols

[GMR85, B85]



Completeness: If statement is true, verifier accepts w/ probability 1.

Soundness: If statement is false, verifier rejects w/ high probability, no matter what prover does.

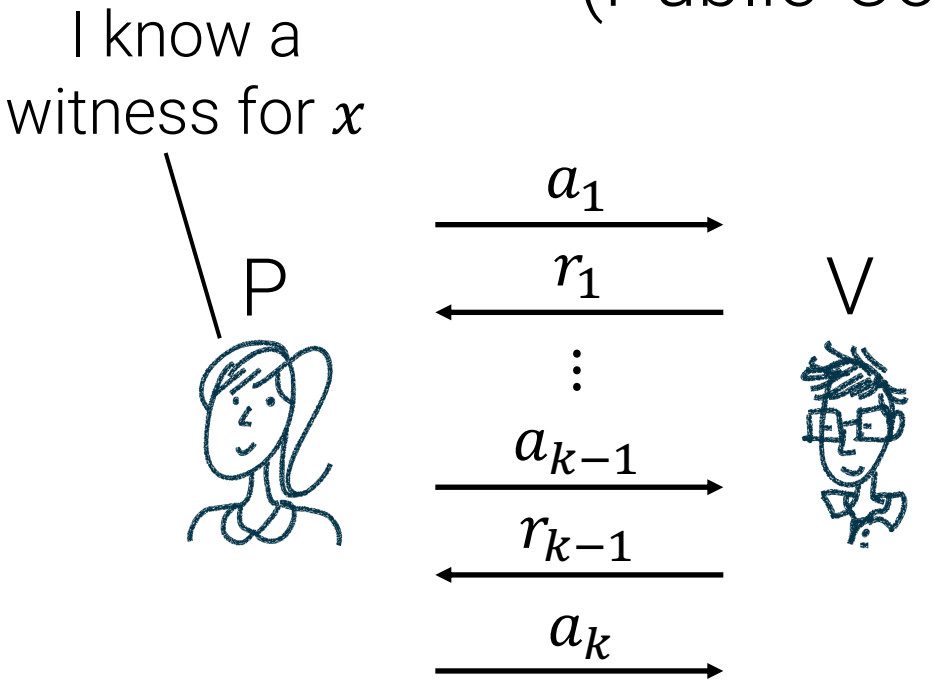
Public coin: r_i uniformly random

Interaction is powerful [GS86, GMR89, GMW91, S92, K92, …]

IP = PSPACE, zero-knowledge, succinct arguments, etc.

(Public-Coin) Interactive Protocols

[GMR85, B85]



Completeness: If statement is true, verifier accepts w/ probability 1.

Soundness: If statement is false, verifier rejects w/ high probability, no matter what prover does.

Public coin: r_i uniformly random

Interaction is powerful [GS86, GMR89, GMW91, S92, K92, …]

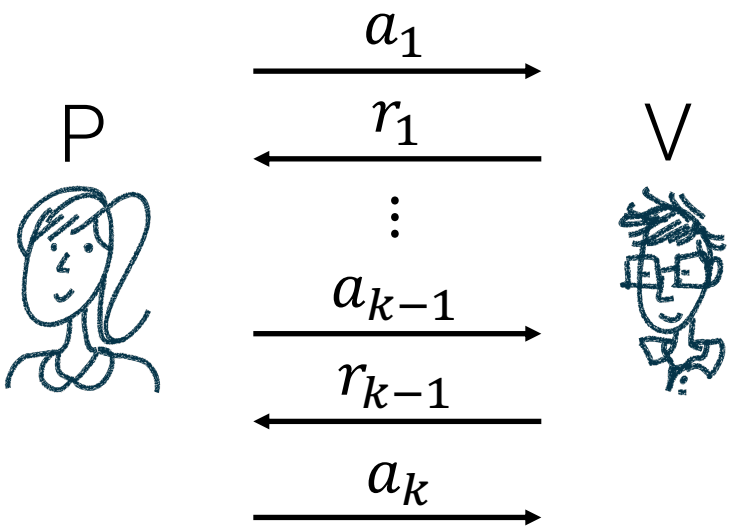
… but often impractical

Fiat-Shamir Heuristic [FS86]

Magical compiler that removes interaction from public-coin interactive protocols.

Fiat-Shamir Heuristic [FS86]

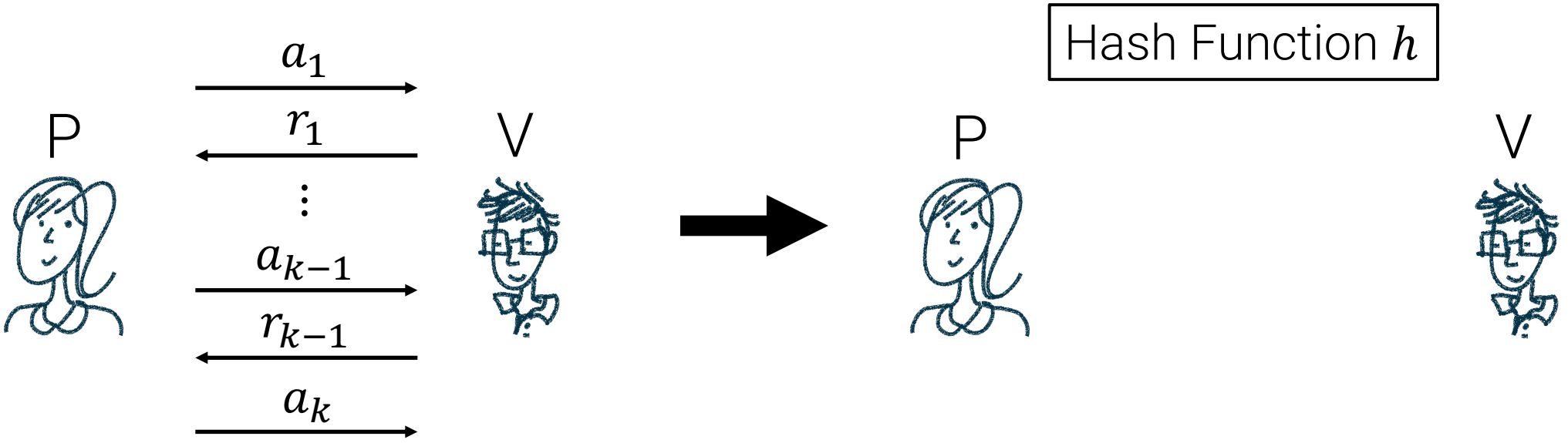
Magical compiler that removes interaction from public-coin interactive protocols.



Public-Coin Interactive Protocol Π

Fiat-Shamir Heuristic [FS86]

Magical compiler that removes interaction from public-coin interactive protocols.

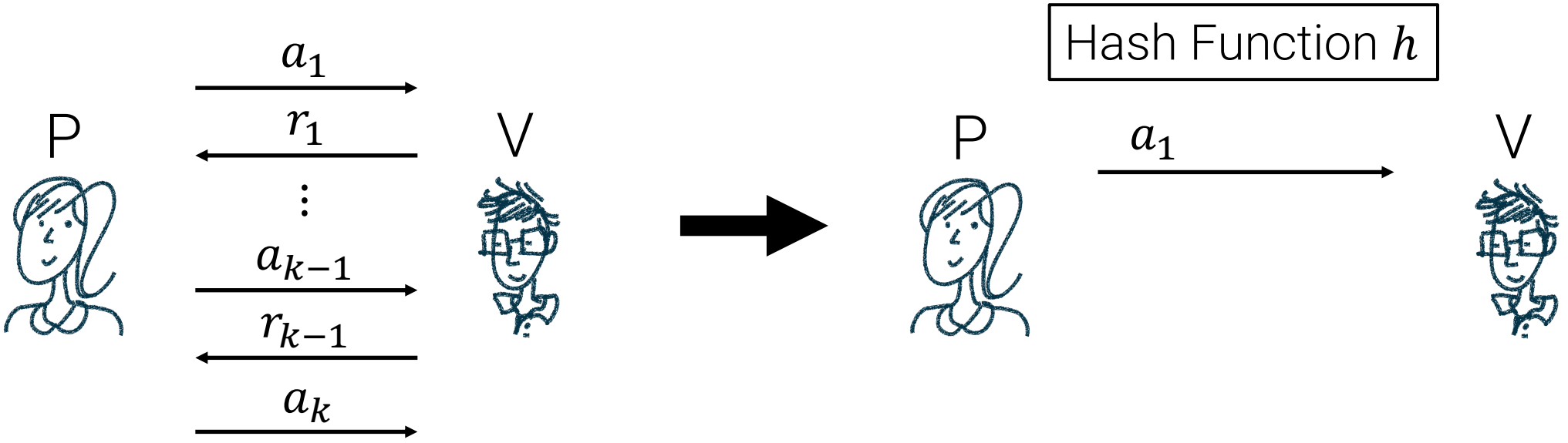


Public-Coin Interactive Protocol Π

Non-Interactive Argument $\mathbf{FS}[\Pi, h]$

Fiat-Shamir Heuristic [FS86]

Magical compiler that removes interaction from public-coin interactive protocols.

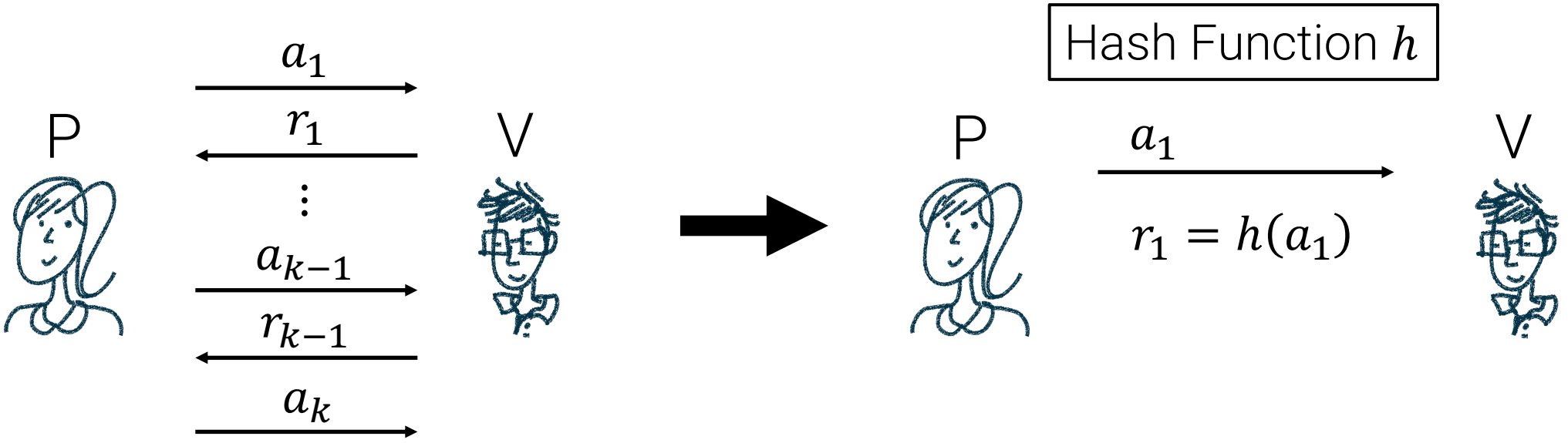


Public-Coin Interactive Protocol Π

Non-Interactive Argument $\mathbf{FS}[\Pi, h]$

Fiat-Shamir Heuristic [FS86]

Magical compiler that removes interaction from public-coin interactive protocols.

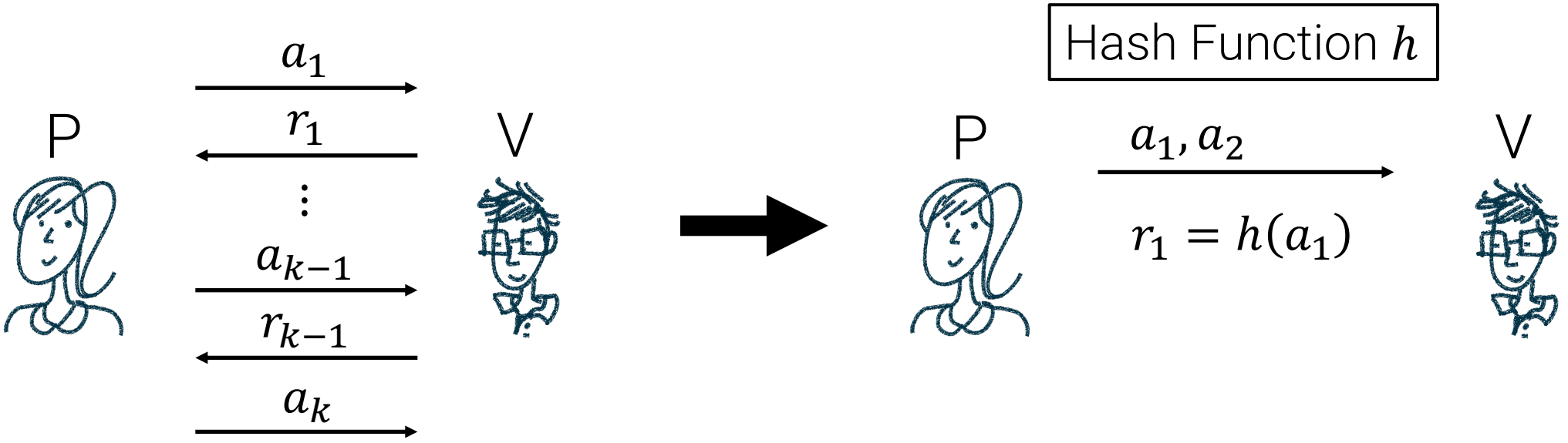


Public-Coin Interactive Protocol Π

Non-Interactive Argument $\mathbf{FS}[\Pi, h]$

Fiat-Shamir Heuristic [FS86]

Magical compiler that removes interaction from public-coin interactive protocols.

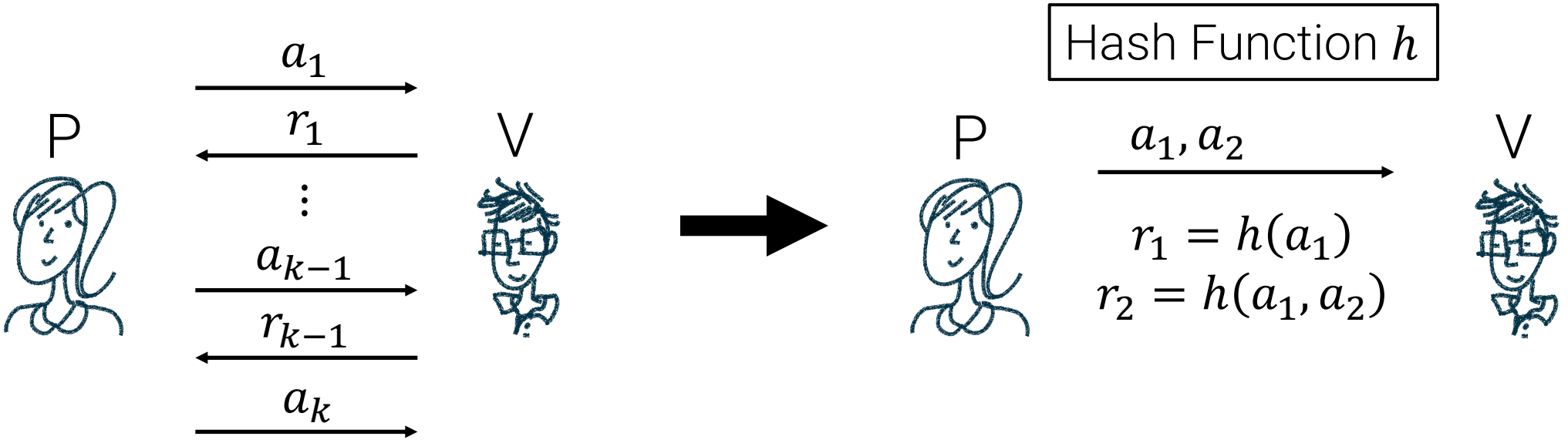


Public-Coin Interactive Protocol Π

Non-Interactive Argument $\mathbf{FS}[\Pi, h]$

Fiat-Shamir Heuristic [FS86]

Magical compiler that removes interaction from public-coin interactive protocols.

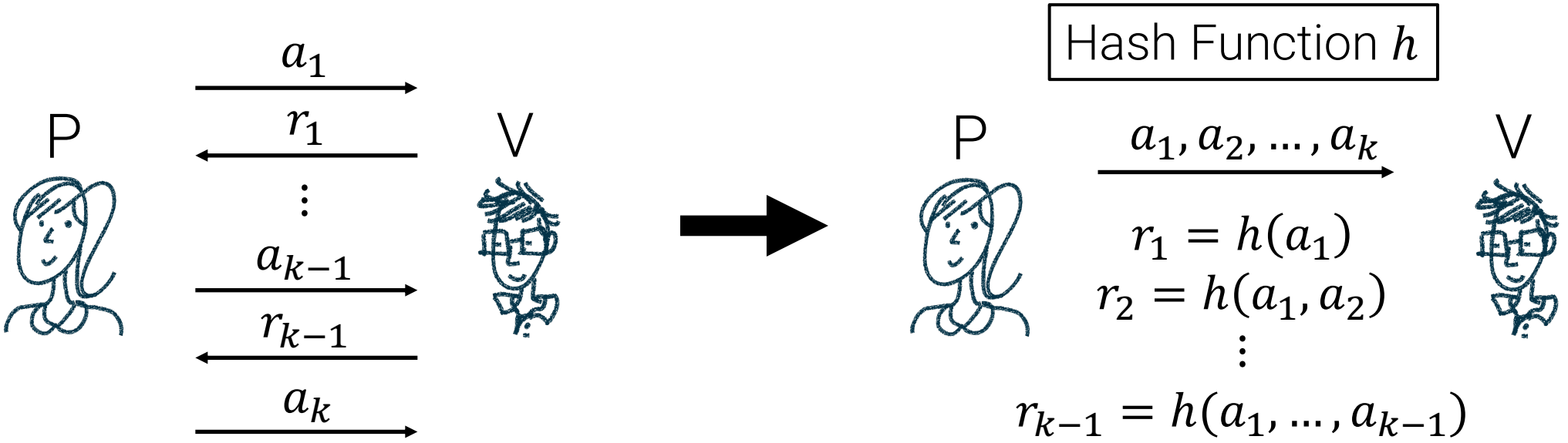


Public-Coin Interactive Protocol Π

Non-Interactive Argument $\mathbf{FS}[\Pi, h]$

Fiat-Shamir Heuristic [FS86]

Magical compiler that removes interaction from public-coin interactive protocols.

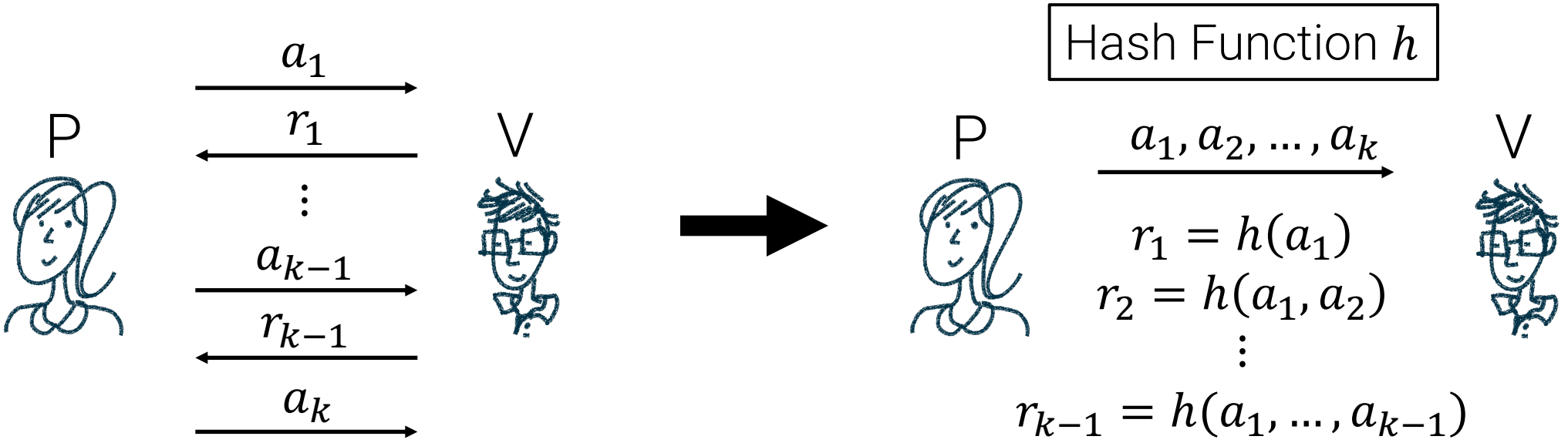


Public-Coin Interactive Protocol Π

Non-Interactive Argument $\mathbf{FS}[\Pi, h]$

Fruitful approach:

Construct ***interactive*** protocol for desired functionality (e.g., identification, verifiable computation) and apply Fiat-Shamir.



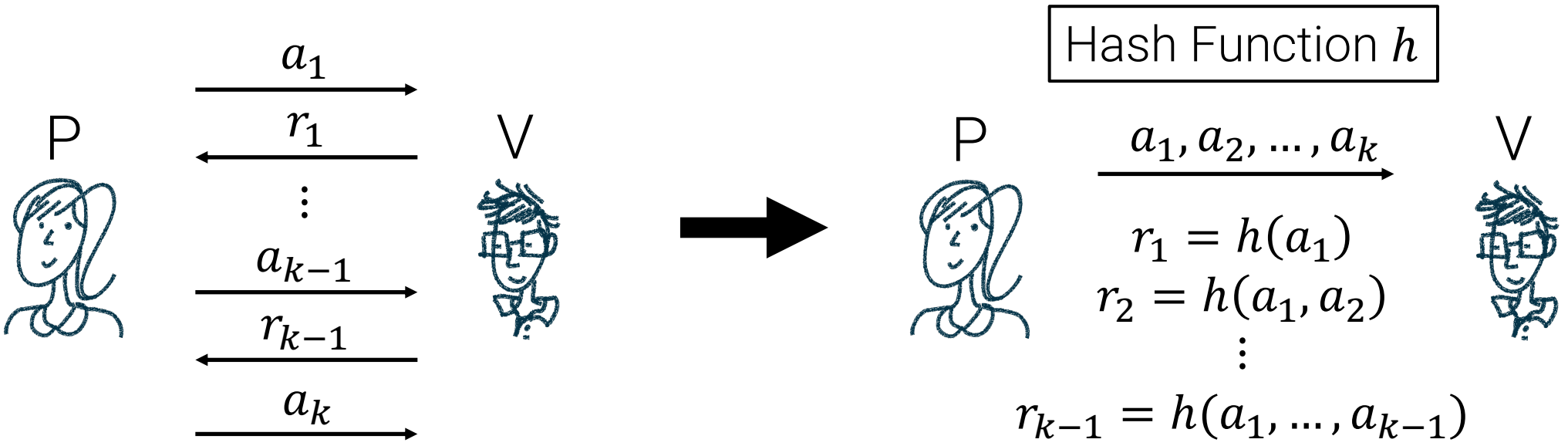
Public-Coin Interactive Protocol Π

Non-Interactive Argument $\mathbf{FS}[\Pi, h]$

When does Fiat-Shamir preserve soundness?

Intuition: Interactive protocol sound when r_i chosen randomly.

FS hash function h shouldn't let prover obtain "favorable" r_i.



Public-Coin Interactive Protocol Π

Non-Interactive Argument $\mathbf{FS}[\Pi, h]$

When does Fiat-Shamir preserve soundness?

Intuition: Interactive protocol sound when r_i chosen randomly.

FS hash function h shouldn't let prover obtain "favorable" r_i.

Two known approaches:

1) h is a random oracle [FS86, BR93, PS96]

2) h is correlation intractable [CGH04, CCHLRRW19, PS19, …]

When does Fiat-Shamir preserve soundness?

Intuition: Interactive protocol sound when r_i chosen randomly.

FS hash function h shouldn't let prover obtain "favorable" r_i.

Two known approaches:

1) h is a random oracle [FS86, BR93, PS96]

2) h is correlation intractable [CGH04, CCHLRRW19, PS19, …]

In all cases, h is a **cryptographic hash function.**

This work: Is this necessary?

In all cases, h is a cryptographic hash function.

Our Results, Part 1

We can compile some protocols with simple FS hash functions.

Our Results, Part 1

We can compile some protocols with simple FS hash functions.

What is a simple FS hash function?
Examples:

- $h(x) = \mathrm{BitDecomp}(x)$
- $h(x) = ax + b \ (mod \ p)$

Our Results, Part 1

We can compile some protocols with simple FS hash functions.

What is a simple FS hash function?
Examples:

- $h(x) = \mathbf{BitDecomp}(x)$
- $h(x) = ax + b \ (mod \ p)$

Which protocols?

- Lyubashevsky's ID protocol
- Schnorr's ID protocol
- Chaum-Pedersen protocol

Our Results, Part 1

We can compile some protocols with simple FS hash functions.

What is a simple FS hash function?
Examples:

- $h(x) = \mathbf{BitDecomp}(x)$
- $h(x) = ax + b \ (mod\ p)$

Which protocols?

- Lyubashevsky's ID protocol
- Schnorr's ID protocol
- Chaum-Pedersen protocol

plain model

generic group model

Our Results, Part 2

Cryptographic FS hash functions are ***necessary*** to compile many 3-message HVZK arguments.

Our Results, Part 2

Cryptographic FS hash functions are ***necessary*** to compile many 3-message HVZK arguments.

In particular, any FS hash function that compiles these protocols must be “mix-and-match resistant”.

Our Results, Part 2

Cryptographic FS hash functions are ***necessary*** to compile many 3-message HVZK arguments.

In particular, any FS hash function that compiles these protocols must be “mix-and-match resistant”.

Example protocols:

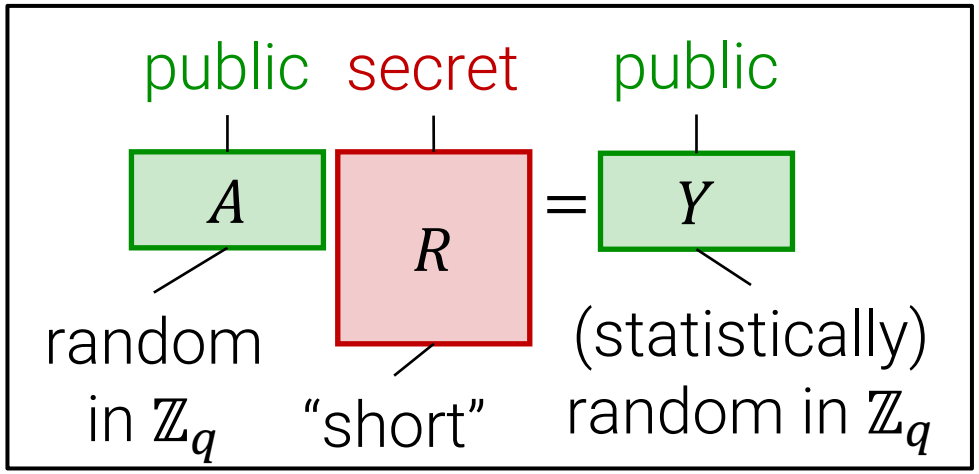
- Blum’s Hamiltonicity protocol (with parallel repetition)
- GMW86 3-Coloring protocol (with parallel repetition)
- 1-bit challenge Schnorr (with parallel repetition)

This talk

- Simple FS compiler for Lyubashevsky's protocol
- Why some protocols require a cryptographic FS hash function

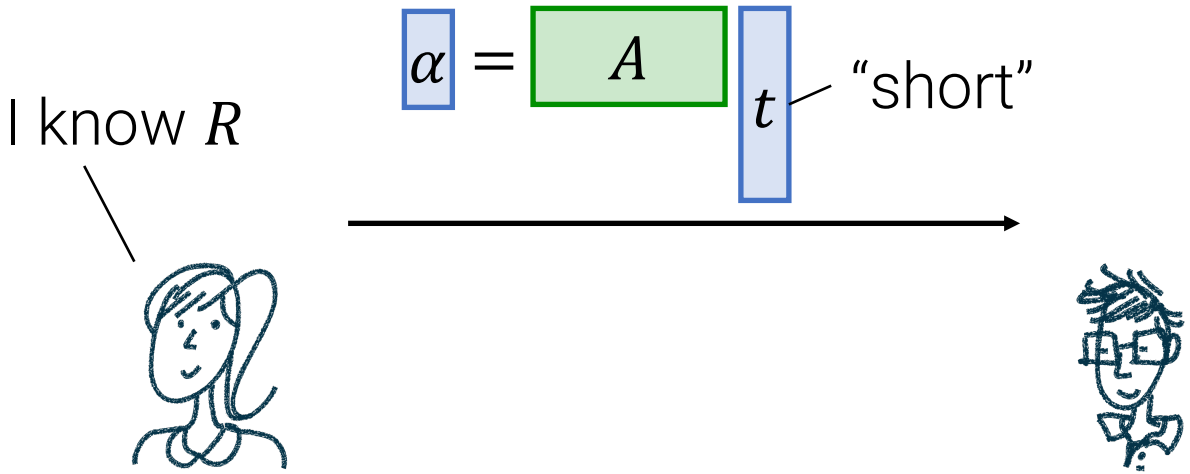
(see paper for positive results on Schnorr signatures)

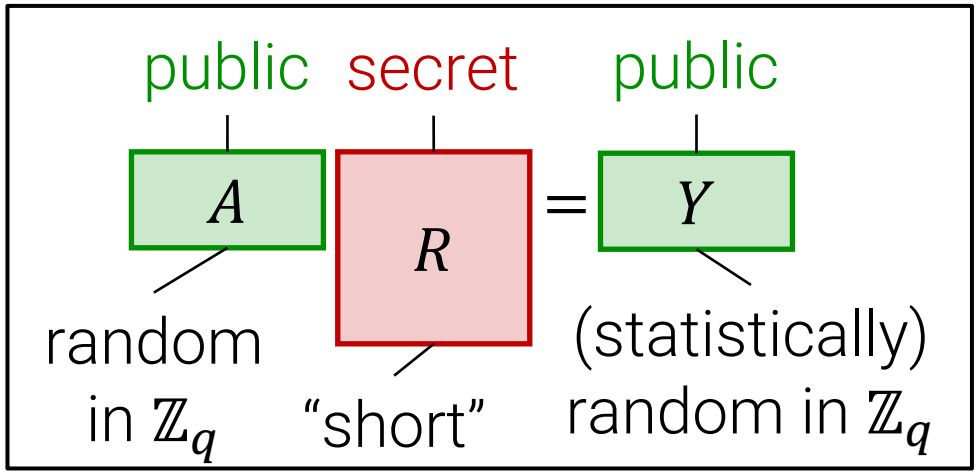
Lyubashevsky's Protocol (recap)



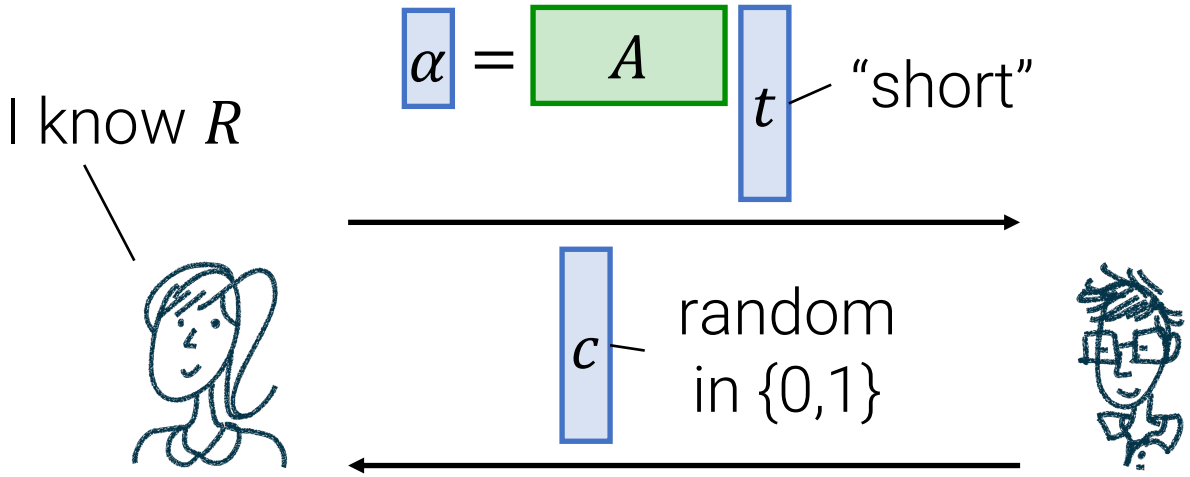
I know R

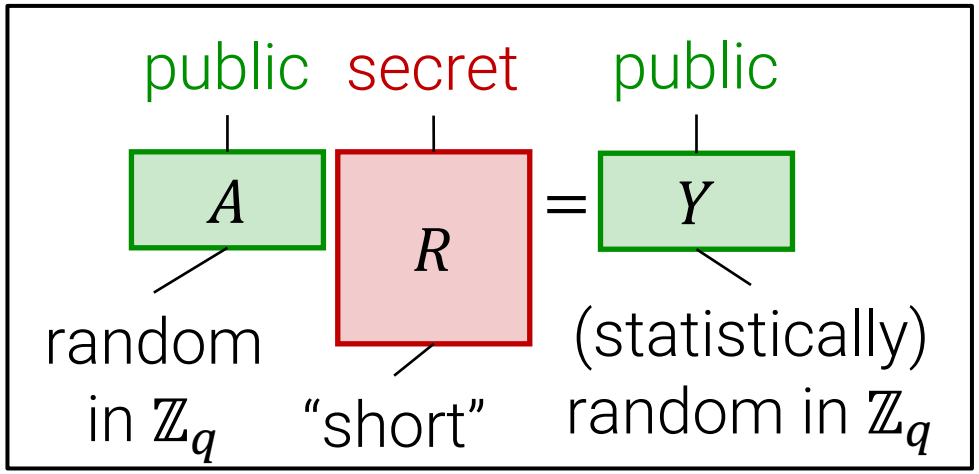
Lyubashevsky's Protocol (recap)



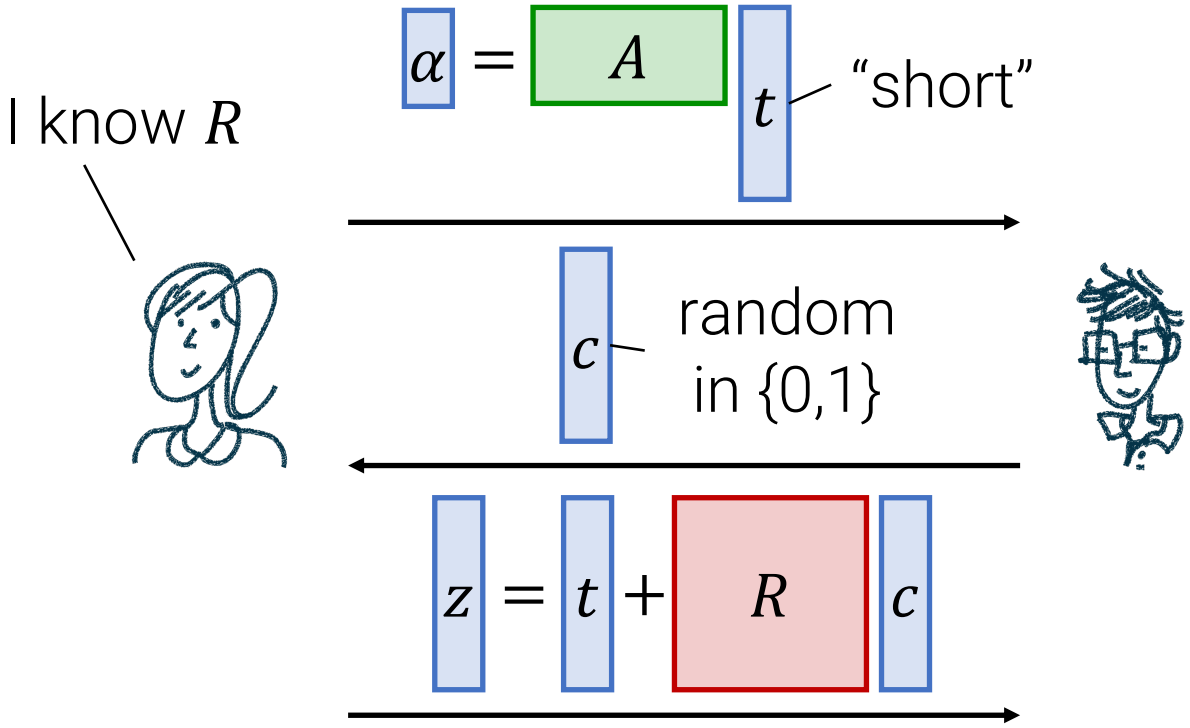


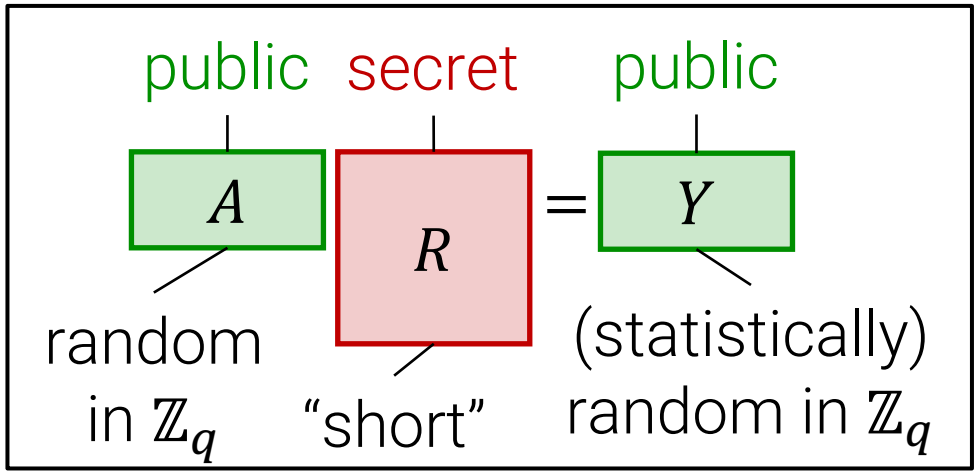
Lyubashevsky's Protocol (recap)



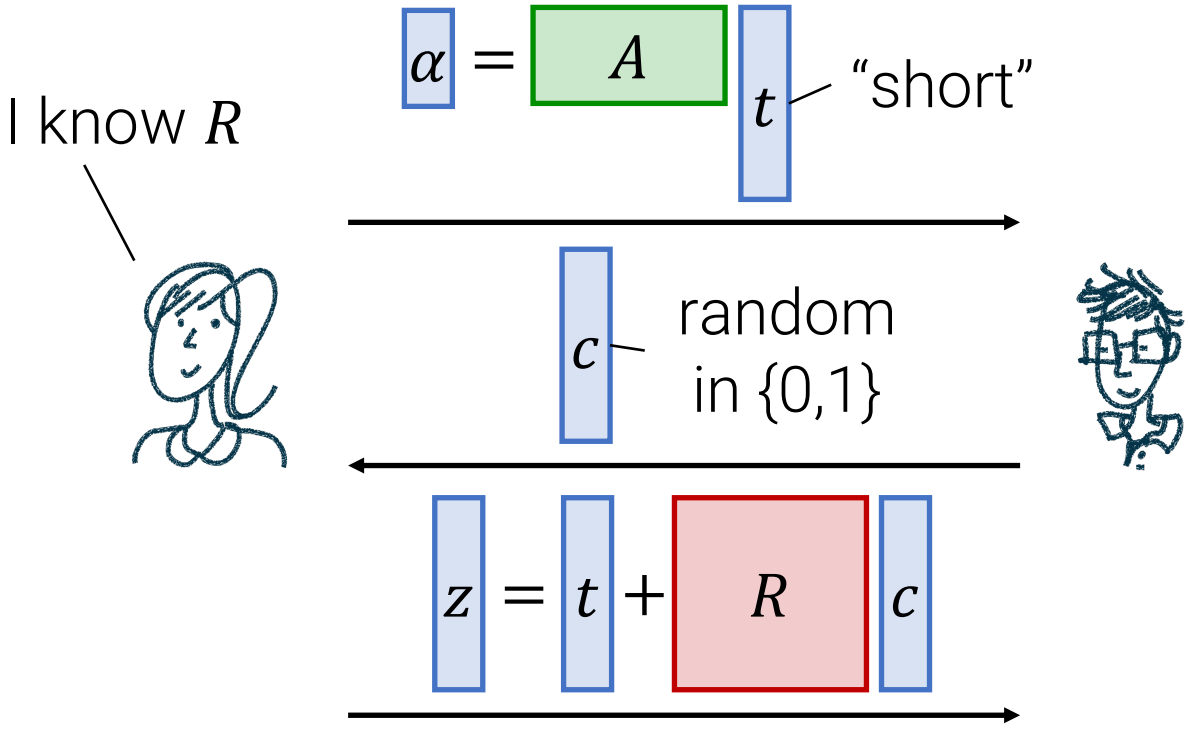


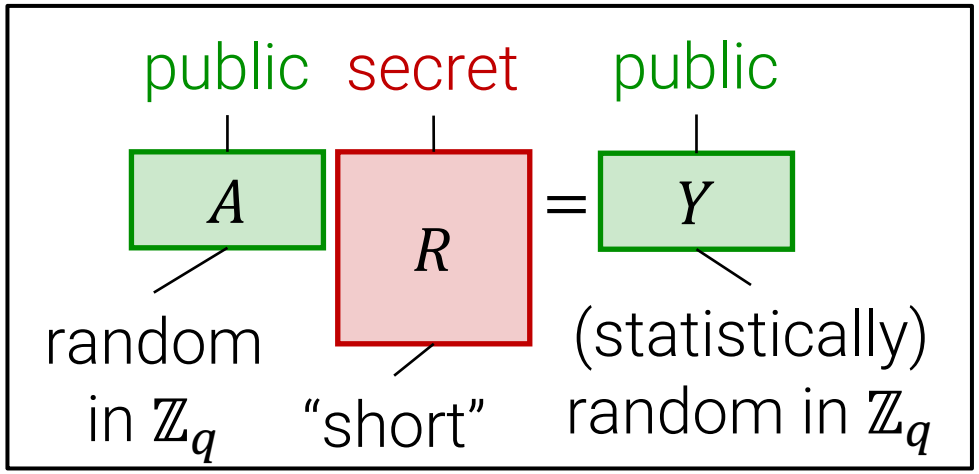
Lyubashevsky's Protocol (recap)





Lyubashevsky's Protocol (recap)





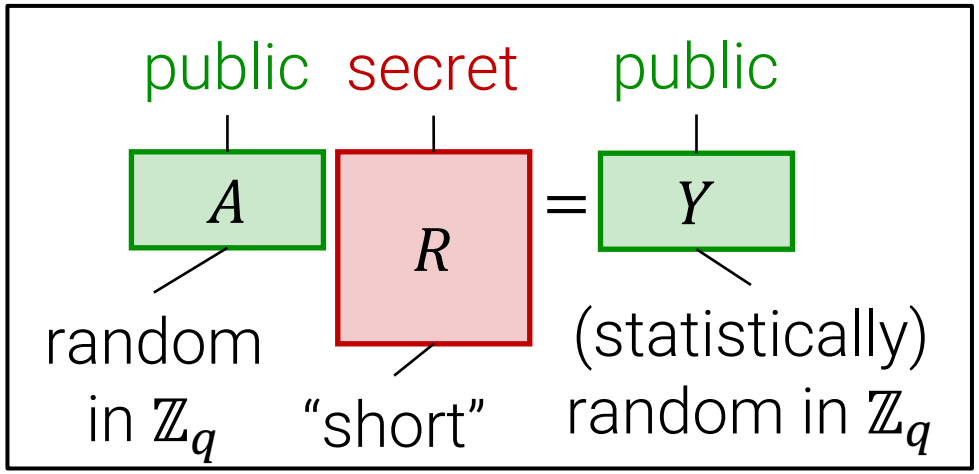
Accept if

$$A\,z = \alpha + Y\,c$$

and z is "short".

Non-Interactive Lyubashevsky

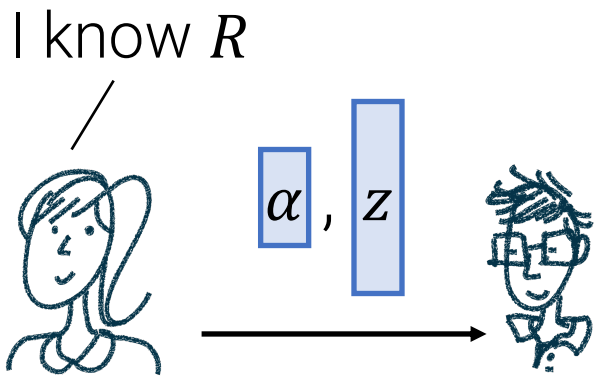
Fiat-Shamir hash function h outputs binary (short) vectors



1) Sample "short" t

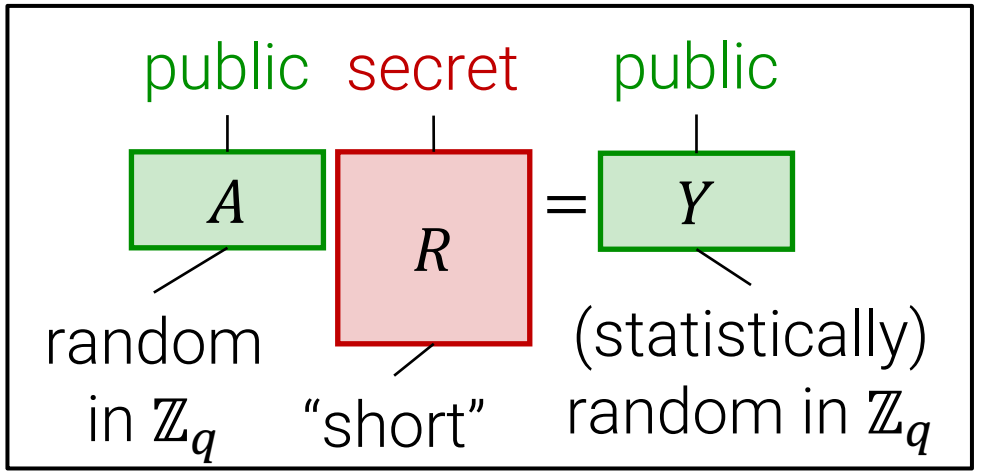
2) $\alpha = A\,t$

3) $z = t + R\,h(\alpha)$



Non-Interactive Lyubashevsky

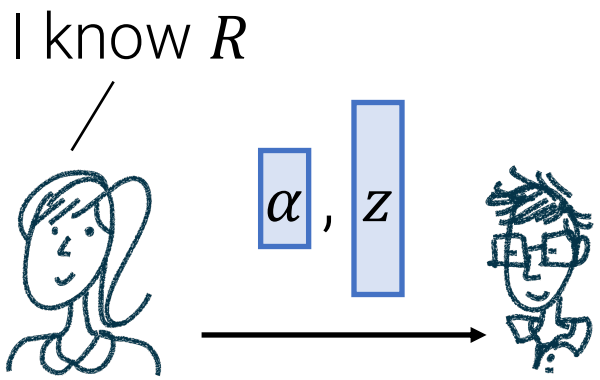
Fiat-Shamir hash function h outputs binary (short) vectors



1) Sample "short" t

2) $\alpha = A\,t$

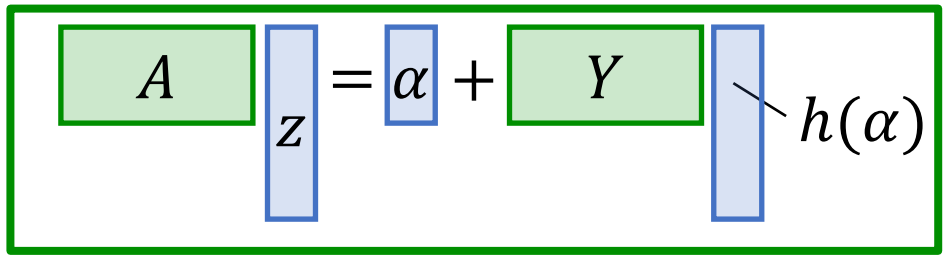
3) $z = t + R\,h(\alpha)$



Accept if z is "short" and:

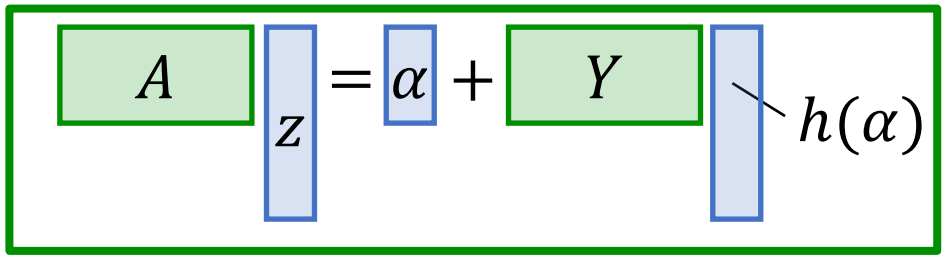
$A\,z = \alpha + Y\,h(\alpha)$

For soundness, want h such that it's hard to find α and short z satisfying:



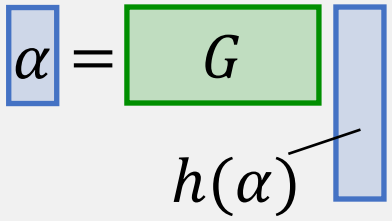
(verifier accepts)

For soundness, want h such that it's hard to find α and short z satisfying:



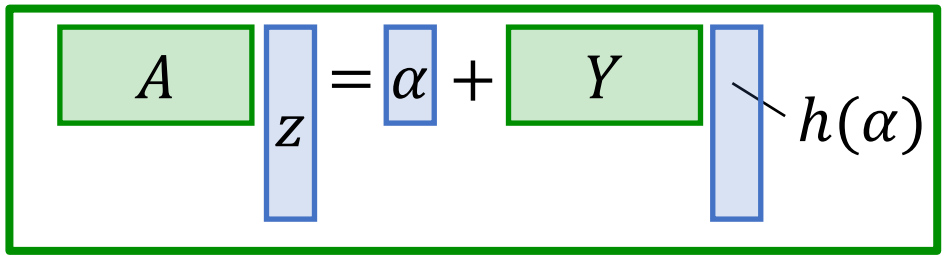
(verifier accepts)

Key Idea: Pick h such that



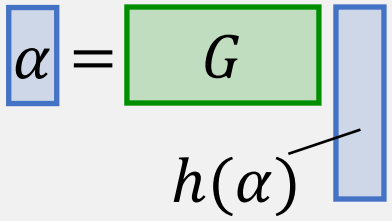
for some matrix G.

For soundness, want h such that it's hard to find α and short z satisfying:



(verifier accepts)

Key Idea: Pick h such that

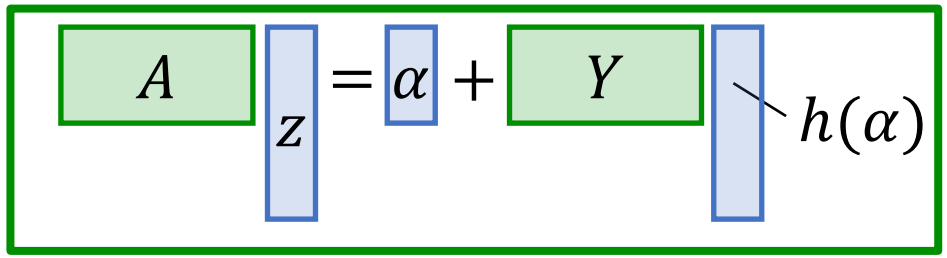


for some matrix G.

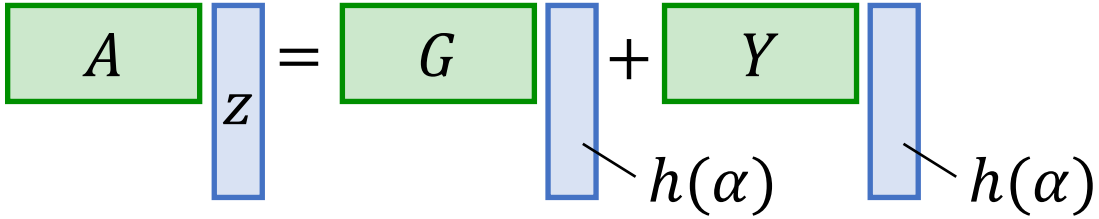
Example: $h(\alpha) = BitDecomp(\alpha)$

$$G = \begin{bmatrix} 1,2,4,\ldots & & \\ & 1,2,4,\ldots. & \\ & & \ddots \\ & & 1,2,4,\ldots \end{bmatrix}.$$

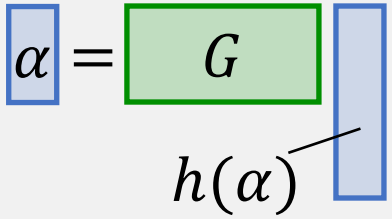
For soundness, want h such that it's hard to find α and short z satisfying:



plug in α

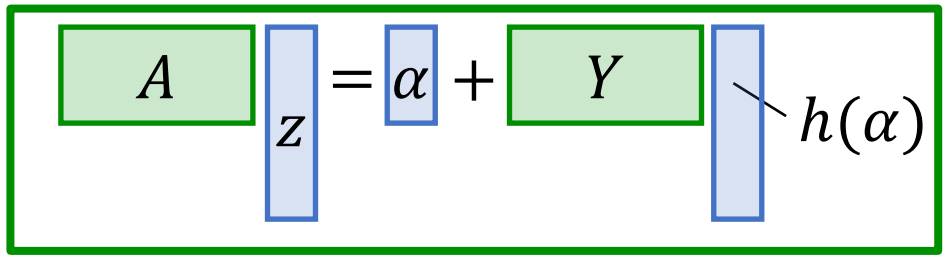


Key Idea: Pick h such that

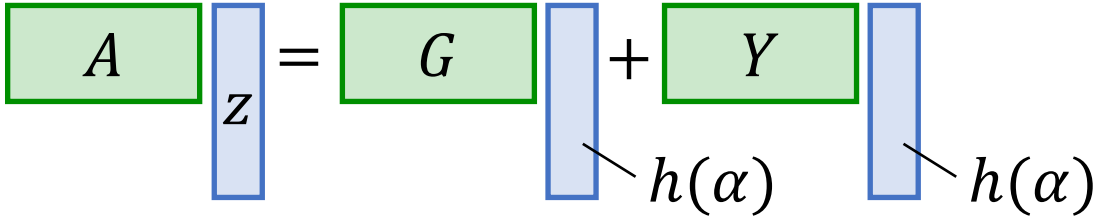


for some matrix G.

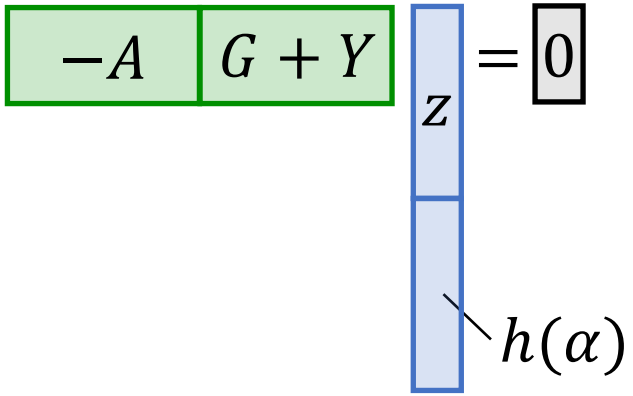
For soundness, want h such that it's hard to find α and short z satisfying:



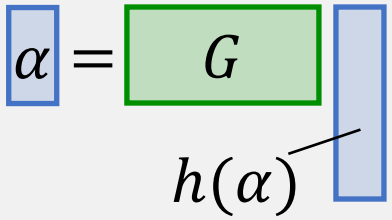
plug in α



rearrange

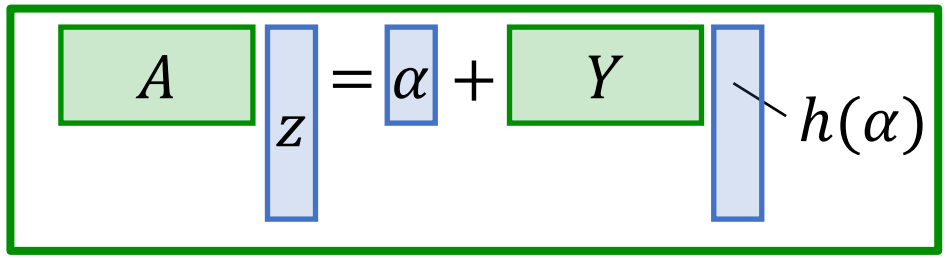


Key Idea: Pick h such that

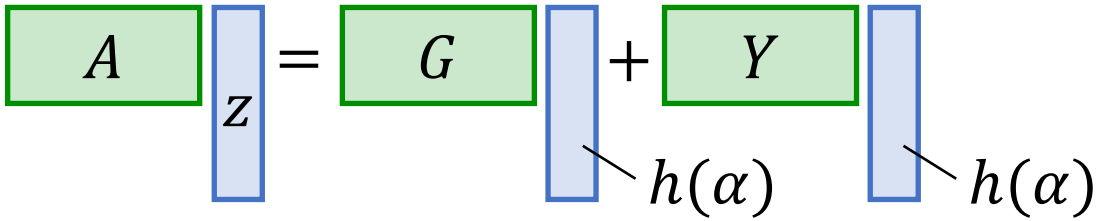


for some matrix G.

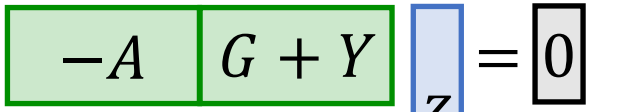
For soundness, want h such that it's hard to find α and short z satisfying:

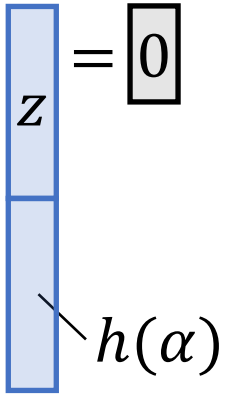


plug in α

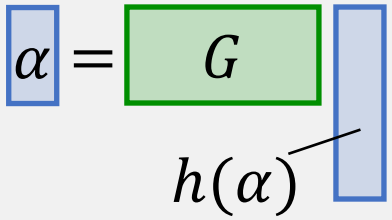


rearrange





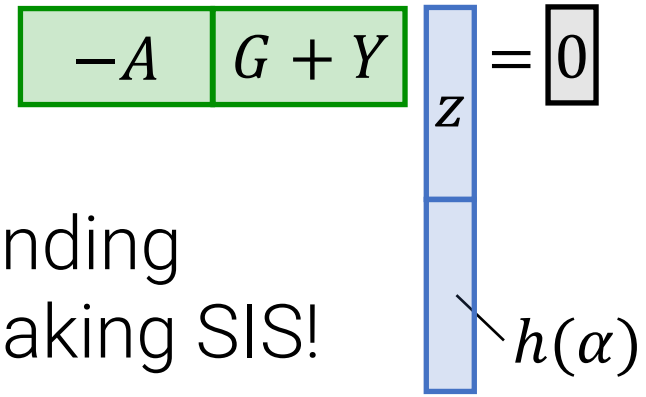
Key Idea: Pick h such that



for some matrix G.

Since A and Y are random, finding short $z, h(\alpha)$ amounts to breaking SIS!

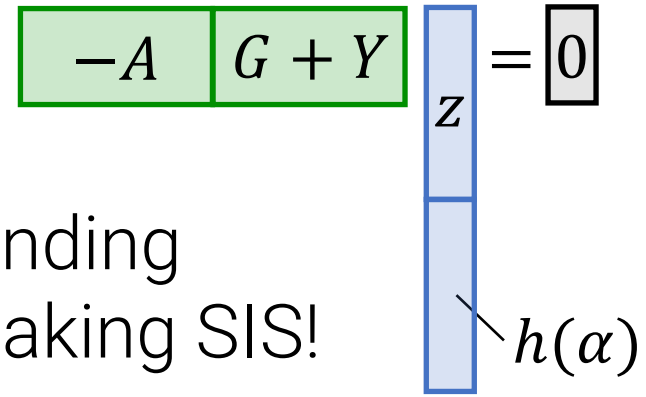
Note: the honest prover in non-interactive Lyubashevsky uses short R satisfying $AR = Y$ to compute this SIS solution.



Since A and Y are random, finding short $z, h(\alpha)$ amounts to breaking SIS!

Note: the honest prover in non-interactive Lyubashevsky uses short R satisfying $AR = Y$ to compute this SIS solution.

Fun fact: Lattice trapdoors of [MP12, LW15] arise from applying Fiat-Shamir with the bit-decomposition function to Lyubashevky.



Since A and Y are random, finding short $z, h(\alpha)$ amounts to breaking SIS!

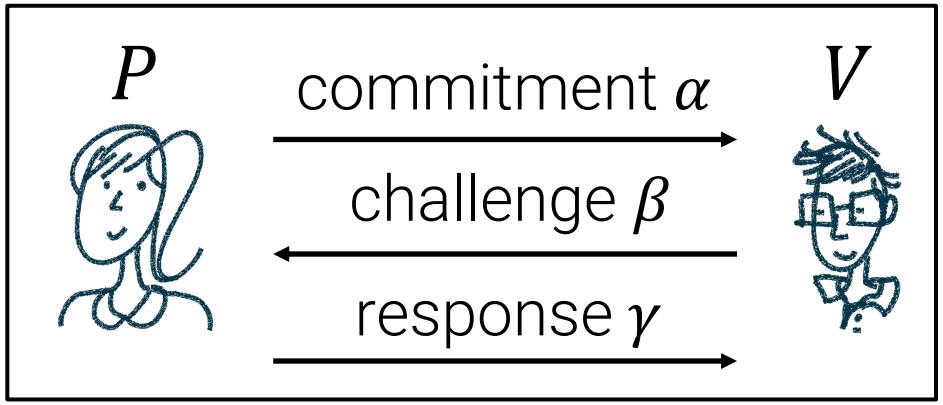
Two seemingly unrelated approaches to lattice signatures:

(1) [GPV08]: To sign m, apply a random oracle h, then use lattice trapdoors to compute a "pre-image" of $h(m)$.

(2) Fiat-Shamir + Lyubashevky.

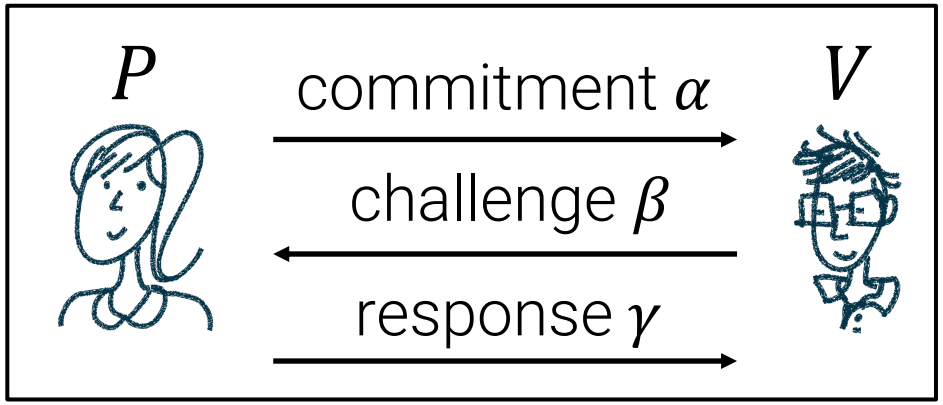
We show: these are equivalent if we instantiate (2) with a bit-decomposition-based FS hash function and use Hash-and-Sign.

Next up: why some protocols ***require*** a cryptographic FS hash function



generic sigma protocol Π

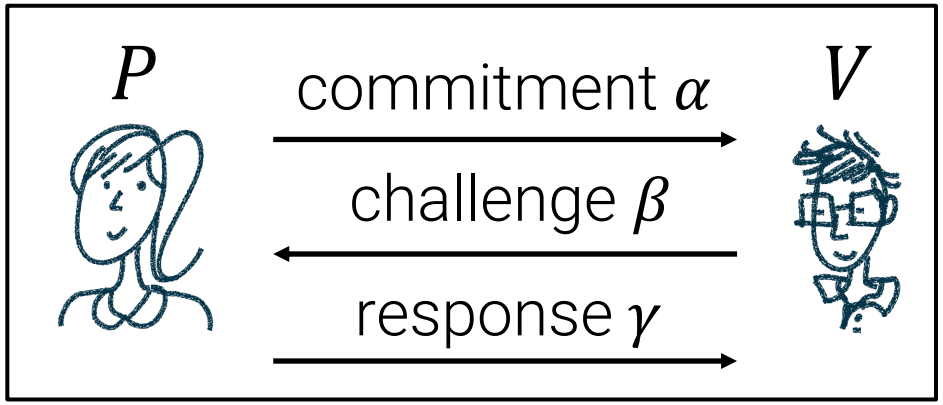
In our positive results, $\beta \leftarrow R$ for super-poly size $|R|$.



generic sigma protocol Π

In our positive results, $\beta \leftarrow R$ for super-poly size $|R|$.

But in many important protocols [GMR85, Blum86, GMW86, …], $|R|$ is poly and soundness requires parallel repetition.

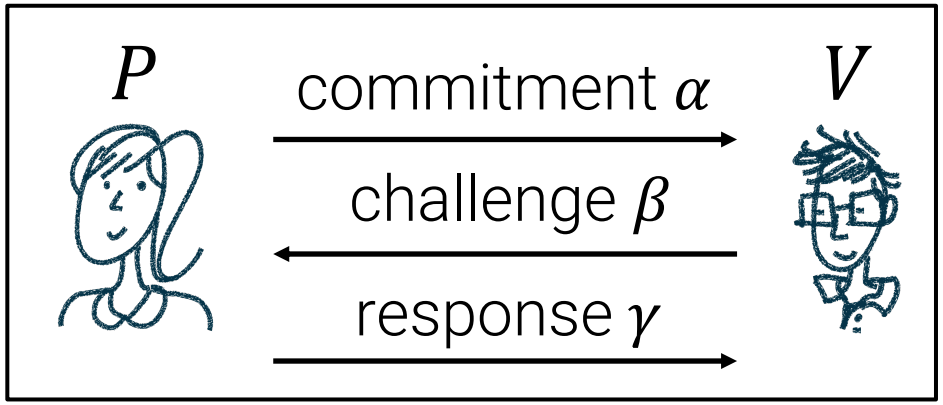


generic sigma protocol Π

In our positive results, $\beta \leftarrow R$ for super-poly size $|R|$.

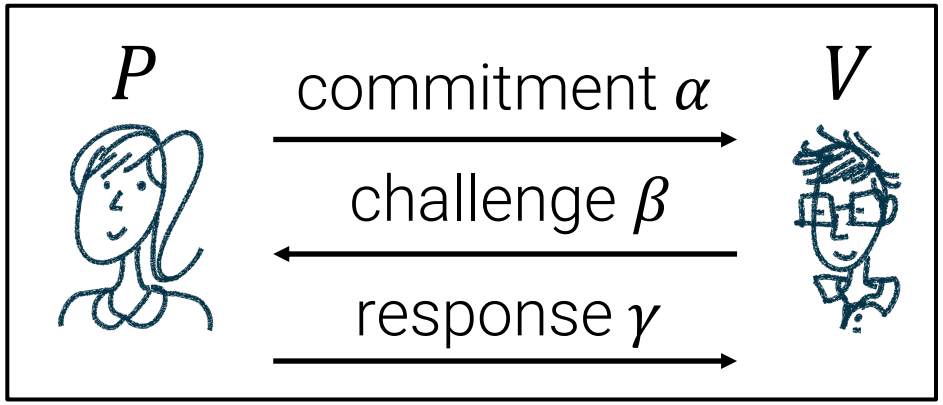
But in many important protocols [GMR85, Blum86, GMW86, …], $|R|$ is poly and soundness requires parallel repetition.

Bad news: our positive FS results *do not apply* to these protocols!



generic sigma protocol Π

Theorem: For many common sigma protocols Π, FS hash function h for Π^t must be cryptographic.

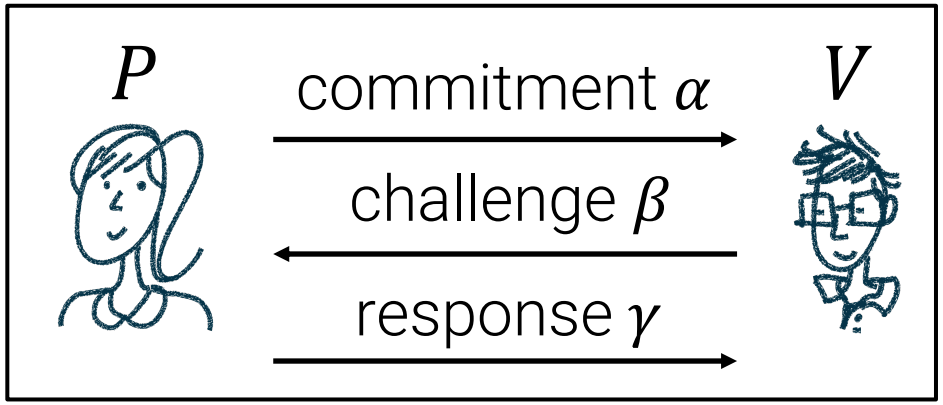


generic sigma protocol Π

Assume:

- β sampled from poly-size R
- α pseudorandom
- Honest-Verifier ZK (HVZK)

Theorem: For many common sigma protocols Π, FS hash function h for Π^t must be cryptographic.



generic sigma protocol Π

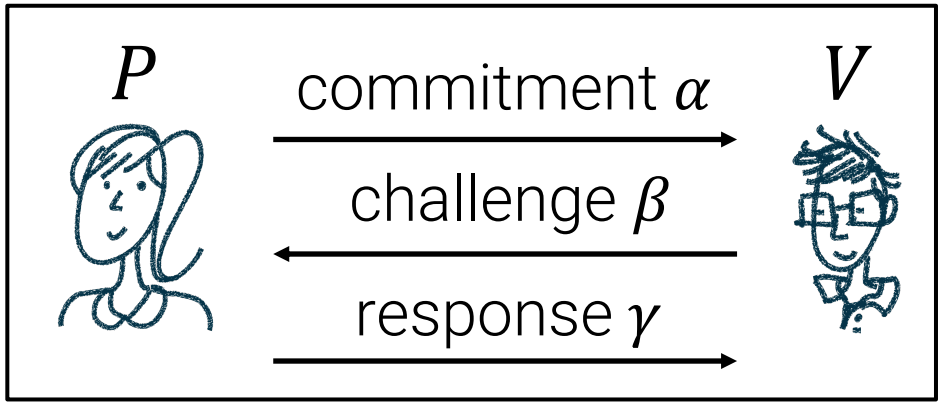
Assume:

- β sampled from poly-size R
- α pseudorandom
- Honest-Verifier ZK (HVZK)

Theorem: For many common sigma protocols Π, FS hash function h for Π^t must be cryptographic.

What does it mean for h to be "cryptographic"?

Informally: can use h to define a challenger-adversary game with a computational/statistical gap.



generic sigma protocol Π

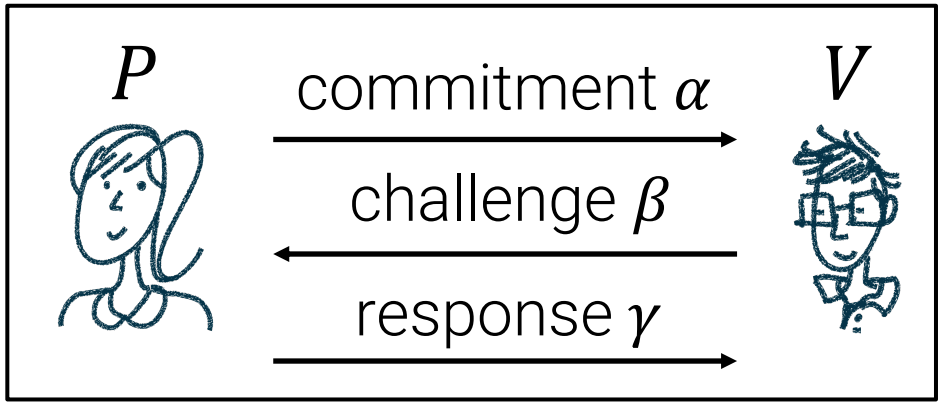
Assume:

- β sampled from poly-size R
- α pseudorandom
- Honest-Verifier ZK (HVZK)

Theorem: For many common sigma protocols Π, FS hash function h for Π^t must be cryptographic.

Proof Idea:

- For any h, define a "mix-and-match" game that an *inefficient* adversary can almost always win.



generic sigma protocol Π

Assume:
- β sampled from poly-size R
- α pseudorandom
- Honest-Verifier ZK (HVZK)

Theorem: For many common sigma protocols Π, FS hash function h for Π^t must be cryptographic.

Proof Idea:
- For any h, define a "mix-and-match" game that an *inefficient* adversary can almost always win.
- We show: if an *efficient* adversary can win the "mix-and-match" game, it can break $\mathrm{FS}[\Pi^t, h]$.

Mix-and-match game for h.

(1) Challenger samples random $\{\alpha_{i,j}, \beta_{i,j}\}$ for $i \in [t], j \in [k]$.

Mix-and-match game for h.

(1) Challenger samples random $\{\alpha_{i,j}, \beta_{i,j}\}$ for $i \in [t], j \in [k]$.

t columns

$\alpha_{1,1}$ $\beta_{1,1}$	$\alpha_{2,1}$ $\beta_{2,1}$	$\alpha_{3,1}$ $\beta_{3,1}$	$\alpha_{4,1}$ $\beta_{4,1}$
$\alpha_{1,2}$ $\beta_{1,2}$	$\alpha_{2,2}$ $\beta_{2,2}$	$\alpha_{3,2}$ $\beta_{3,2}$	$\alpha_{4,2}$ $\beta_{4,2}$
$\alpha_{1,3}$ $\beta_{1,3}$	$\alpha_{2,3}$ $\beta_{2,3}$	$\alpha_{3,3}$ $\beta_{3,3}$	$\alpha_{4,3}$ $\beta_{4,3}$
$\alpha_{1,4}$ $\beta_{1,4}$	$\alpha_{2,4}$ $\beta_{2,4}$	$\alpha_{3,4}$ $\beta_{3,4}$	$\alpha_{4,4}$ $\beta_{4,4}$
$\alpha_{1,5}$ $\beta_{1,5}$	$\alpha_{2,5}$ $\beta_{2,5}$	$\alpha_{3,5}$ $\beta_{3,5}$	$\alpha_{4,5}$ $\beta_{4,5}$

k rows

Mix-and-match game for h.

(1) Challenger samples random $\{\alpha_{i,j}, \beta_{i,j}\}$ for $i \in [t], j \in [k]$.

(2) Given $\{\alpha_{i,j}, \beta_{i,j}\}$, adversary wins if it finds $j_1, \ldots, j_t$ such that $h(\alpha_{1,j_1}, \ldots, \alpha_{t,j_t}) = \beta_{1,j_1}, \ldots, \beta_{t,j_t}$

t columns

$\alpha_{1,1}$ $\beta_{1,1}$	$\alpha_{2,1}$ $\beta_{2,1}$	$\alpha_{3,1}$ $\beta_{3,1}$	$\alpha_{4,1}$ $\beta_{4,1}$
$\alpha_{1,2}$ $\beta_{1,2}$	$\alpha_{2,2}$ $\beta_{2,2}$	$\alpha_{3,2}$ $\beta_{3,2}$	$\alpha_{4,2}$ $\beta_{4,2}$
$\alpha_{1,3}$ $\beta_{1,3}$	$\alpha_{2,3}$ $\beta_{2,3}$	$\alpha_{3,3}$ $\beta_{3,3}$	$\alpha_{4,3}$ $\beta_{4,3}$
$\alpha_{1,4}$ $\beta_{1,4}$	$\alpha_{2,4}$ $\beta_{2,4}$	$\alpha_{3,4}$ $\beta_{3,4}$	$\alpha_{4,4}$ $\beta_{4,4}$
$\alpha_{1,5}$ $\beta_{1,5}$	$\alpha_{2,5}$ $\beta_{2,5}$	$\alpha_{3,5}$ $\beta_{3,5}$	$\alpha_{4,5}$ $\beta_{4,5}$

k rows

Mix-and-match game for h.

(1) Challenger samples random $\{\alpha_{i,j}, \beta_{i,j}\}$ for $i \in [t], j \in [k]$.

(2) Given $\{\alpha_{i,j}, \beta_{i,j}\}$, adversary wins if it finds $j_1, \ldots, j_t$ such that

$$h(\alpha_{1,j_1}, \ldots, \alpha_{t,j_t}) = \beta_{1,j_1}, \ldots, \beta_{t,j_t}$$

t columns

$\alpha_{1,1}$ $\beta_{1,1}$	$\alpha_{2,1}$ $\beta_{2,1}$	$\alpha_{3,1}$ $\beta_{3,1}$	$\alpha_{4,1}$ $\beta_{4,1}$
$\alpha_{1,2}$ $\beta_{1,2}$	$\alpha_{2,2}$ $\beta_{2,2}$	$\alpha_{3,2}$ $\beta_{3,2}$	$\alpha_{4,2}$ $\beta_{4,2}$
$\alpha_{1,3}$ $\beta_{1,3}$	$\alpha_{2,3}$ $\beta_{2,3}$	$\alpha_{3,3}$ $\beta_{3,3}$	$\alpha_{4,3}$ $\beta_{4,3}$
$\alpha_{1,4}$ $\beta_{1,4}$	$\alpha_{2,4}$ $\beta_{2,4}$	$\alpha_{3,4}$ $\beta_{3,4}$	$\alpha_{4,4}$ $\beta_{4,4}$
$\alpha_{1,5}$ $\beta_{1,5}$	$\alpha_{2,5}$ $\beta_{2,5}$	$\alpha_{3,5}$ $\beta_{3,5}$	$\alpha_{4,5}$ $\beta_{4,5}$

k rows

Mix-and-match game for h.

(1) Challenger samples random $\{\alpha_{i,j}, \beta_{i,j}\}$ for $i \in [t], j \in [k]$.

(2) Given $\{\alpha_{i,j}, \beta_{i,j}\}$, adversary wins if it finds $j_1, \ldots, j_t$ such that $h(\alpha_{1,j_1}, \ldots, \alpha_{t,j_t}) = \beta_{1,j_1}, \ldots, \beta_{t,j_t}$

Lemma: If $k = \omega(t)$, solution exists with $1 - \mathbf{negl}(t)$ probability

t columns

$\alpha_{1,1}$ $\beta_{1,1}$	$\alpha_{2,1}$ $\beta_{2,1}$	$\alpha_{3,1}$ $\beta_{3,1}$	$\alpha_{4,1}$ $\beta_{4,1}$
$\alpha_{1,2}$ $\beta_{1,2}$	$\alpha_{2,2}$ $\beta_{2,2}$	$\alpha_{3,2}$ $\beta_{3,2}$	$\alpha_{4,2}$ $\beta_{4,2}$
$\alpha_{1,3}$ $\beta_{1,3}$	$\alpha_{2,3}$ $\beta_{2,3}$	$\alpha_{3,3}$ $\beta_{3,3}$	$\alpha_{4,3}$ $\beta_{4,3}$
$\alpha_{1,4}$ $\beta_{1,4}$	$\alpha_{2,4}$ $\beta_{2,4}$	$\alpha_{3,4}$ $\beta_{3,4}$	$\alpha_{4,4}$ $\beta_{4,4}$
$\alpha_{1,5}$ $\beta_{1,5}$	$\alpha_{2,5}$ $\beta_{2,5}$	$\alpha_{3,5}$ $\beta_{3,5}$	$\alpha_{4,5}$ $\beta_{4,5}$

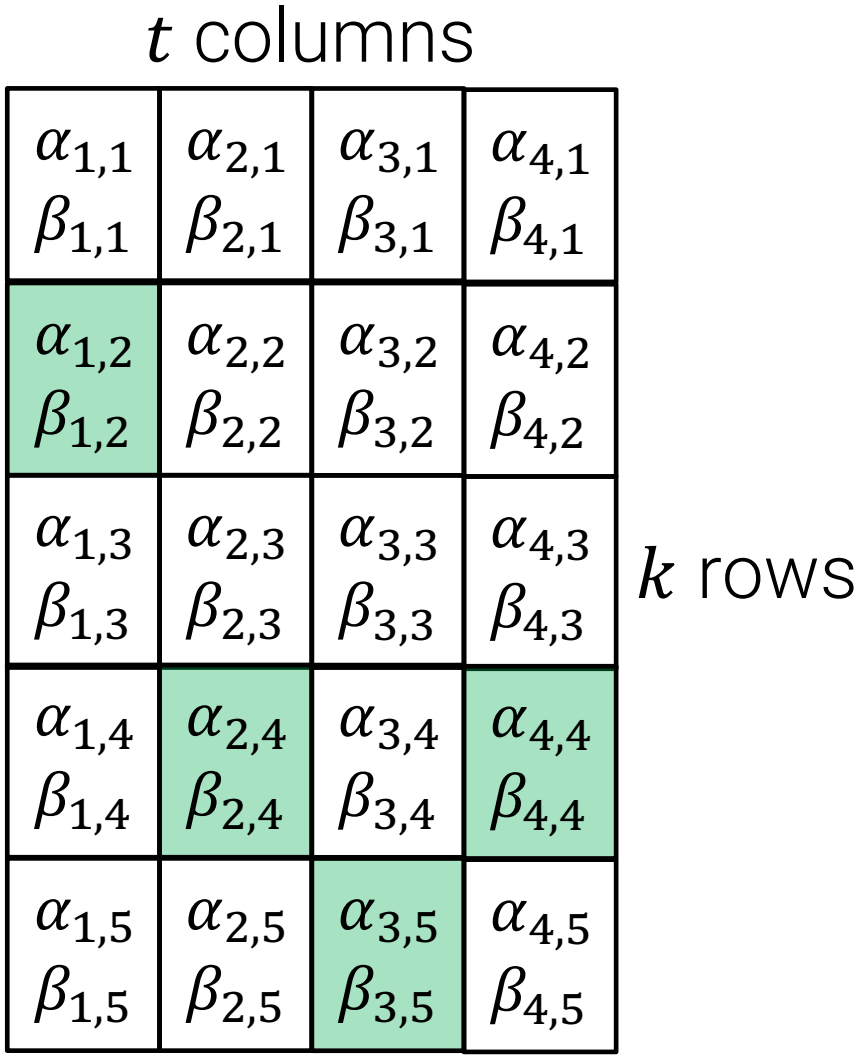
k rows

Mix-and-match game for h.

(1) Challenger samples random $\{\alpha_{i,j}, \beta_{i,j}\}$ for $i \in [t], j \in [k]$.

(2) Given $\{\alpha_{i,j}, \beta_{i,j}\}$, adversary wins if it finds $j_1, \ldots, j_t$ such that $h(\alpha_{1,j_1}, \ldots, \alpha_{t,j_t}) = \beta_{1,j_1}, \ldots, \beta_{t,j_t}$

Want to show: given *efficient* attacker in the mix-and-match game for h, can break soundness of $\mathbf{FS}[\Pi^t, h]$:



Attack on $\mathbf{FS}[\Pi^t, h]$

(1) Run HVZK simulator to generate accepting transcripts $\{\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}\}$

Attack on $\mathbf{FS}[\Pi^t, h]$

(1) Run HVZK simulator to generate accepting transcripts $\{\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}\}$

(2) Run Mix-and-Match attacker on $\{\alpha_{i,j}, \beta_{i,j}\}$ to obtain $j_1, \ldots, j_t$

t columns

$\alpha_{1,1}$ $\beta_{1,1}$	$\alpha_{2,1}$ $\beta_{2,1}$	$\alpha_{3,1}$ $\beta_{3,1}$	$\alpha_{4,1}$ $\beta_{4,1}$
$\alpha_{1,2}$ $\beta_{1,2}$	$\alpha_{2,2}$ $\beta_{2,2}$	$\alpha_{3,2}$ $\beta_{3,2}$	$\alpha_{4,2}$ $\beta_{4,2}$
$\alpha_{1,3}$ $\beta_{1,3}$	$\alpha_{2,3}$ $\beta_{2,3}$	$\alpha_{3,3}$ $\beta_{3,3}$	$\alpha_{4,3}$ $\beta_{4,3}$
$\alpha_{1,4}$ $\beta_{1,4}$	$\alpha_{2,4}$ $\beta_{2,4}$	$\alpha_{3,4}$ $\beta_{3,4}$	$\alpha_{4,4}$ $\beta_{4,4}$
$\alpha_{1,5}$ $\beta_{1,5}$	$\alpha_{2,5}$ $\beta_{2,5}$	$\alpha_{3,5}$ $\beta_{3,5}$	$\alpha_{4,5}$ $\beta_{4,5}$

k rows

Attack on $\mathbf{FS}[\Pi^t, h]$

(1) Run HVZK simulator to generate accepting transcripts $\{\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}\}$

(2) Run Mix-and-Match attacker on $\{\alpha_{i,j}, \beta_{i,j}\}$ to obtain $j_1, \ldots, j_t$

t columns

$\alpha_{1,1}$ $\beta_{1,1}$	$\alpha_{2,1}$ $\beta_{2,1}$	$\alpha_{3,1}$ $\beta_{3,1}$	$\alpha_{4,1}$ $\beta_{4,1}$
$\alpha_{1,2}$ $\beta_{1,2}$	$\alpha_{2,2}$ $\beta_{2,2}$	$\alpha_{3,2}$ $\beta_{3,2}$	$\alpha_{4,2}$ $\beta_{4,2}$
$\alpha_{1,3}$ $\beta_{1,3}$	$\alpha_{2,3}$ $\beta_{2,3}$	$\alpha_{3,3}$ $\beta_{3,3}$	$\alpha_{4,3}$ $\beta_{4,3}$
$\alpha_{1,4}$ $\beta_{1,4}$	$\alpha_{2,4}$ $\beta_{2,4}$	$\alpha_{3,4}$ $\beta_{3,4}$	$\alpha_{4,4}$ $\beta_{4,4}$
$\alpha_{1,5}$ $\beta_{1,5}$	$\alpha_{2,5}$ $\beta_{2,5}$	$\alpha_{3,5}$ $\beta_{3,5}$	$\alpha_{4,5}$ $\beta_{4,5}$

k rows

Attack on $\mathbf{FS}[\Pi^t, h]$

(1) Run HVZK simulator to generate accepting transcripts $\{\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}\}$
(2) Run Mix-and-Match attacker on $\{\alpha_{i,j}, \beta_{i,j}\}$ to obtain $j_1, \ldots, j_t$
(3) Output $\{\alpha_{i,j_i}, \gamma_{i,j_i}\}$ for $i \in [t]$

t columns

$\alpha_{1,1}$ $\beta_{1,1}$	$\alpha_{2,1}$ $\beta_{2,1}$	$\alpha_{3,1}$ $\beta_{3,1}$	$\alpha_{4,1}$ $\beta_{4,1}$
$\alpha_{1,2}$ $\beta_{1,2}$	$\alpha_{2,2}$ $\beta_{2,2}$	$\alpha_{3,2}$ $\beta_{3,2}$	$\alpha_{4,2}$ $\beta_{4,2}$
$\alpha_{1,3}$ $\beta_{1,3}$	$\alpha_{2,3}$ $\beta_{2,3}$	$\alpha_{3,3}$ $\beta_{3,3}$	$\alpha_{4,3}$ $\beta_{4,3}$
$\alpha_{1,4}$ $\beta_{1,4}$	$\alpha_{2,4}$ $\beta_{2,4}$	$\alpha_{3,4}$ $\beta_{3,4}$	$\alpha_{4,4}$ $\beta_{4,4}$
$\alpha_{1,5}$ $\beta_{1,5}$	$\alpha_{2,5}$ $\beta_{2,5}$	$\alpha_{3,5}$ $\beta_{3,5}$	$\alpha_{4,5}$ $\beta_{4,5}$

k rows

Attack on $\mathbf{FS}[\Pi^t, h]$

(1) Run HVZK simulator to generate accepting transcripts $\{\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}\}$
(2) Run Mix-and-Match attacker on $\{\alpha_{i,j}, \beta_{i,j}\}$ to obtain $j_1, \ldots, j_t$
(3) Output $\{\alpha_{i,j_i}, \gamma_{i,j_i}\}$ for $i \in [t]$

Why does this work?

- $\{\alpha_{i,j}, \beta_{i,j}\}$ is pseudorandom, so Mix-and-Match attacker succeeds with good probability
- If attacker succeeds, h outputs "correct" $\beta_{1,j_1}, \ldots, \beta_{t,j_t}$.

t columns

$\alpha_{1,1}$ $\beta_{1,1}$	$\alpha_{2,1}$ $\beta_{2,1}$	$\alpha_{3,1}$ $\beta_{3,1}$	$\alpha_{4,1}$ $\beta_{4,1}$
$\alpha_{1,2}$ $\beta_{1,2}$	$\alpha_{2,2}$ $\beta_{2,2}$	$\alpha_{3,2}$ $\beta_{3,2}$	$\alpha_{4,2}$ $\beta_{4,2}$
$\alpha_{1,3}$ $\beta_{1,3}$	$\alpha_{2,3}$ $\beta_{2,3}$	$\alpha_{3,3}$ $\beta_{3,3}$	$\alpha_{4,3}$ $\beta_{4,3}$
$\alpha_{1,4}$ $\beta_{1,4}$	$\alpha_{2,4}$ $\beta_{2,4}$	$\alpha_{3,4}$ $\beta_{3,4}$	$\alpha_{4,4}$ $\beta_{4,4}$
$\alpha_{1,5}$ $\beta_{1,5}$	$\alpha_{2,5}$ $\beta_{2,5}$	$\alpha_{3,5}$ $\beta_{3,5}$	$\alpha_{4,5}$ $\beta_{4,5}$

k rows

In summary, Fiat-Shamir *sometimes* requires a cryptographic FS hash function.

- For certain protocols (Lyubashevsky, Schnorr, Chaum-Pedersen), simple FS hash functions can be enough.
- But for many classic protocols such as [GMR85,Blum86,GMW86], cryptographic FS hash functions are necessary.

eprint: 2020/915

drawings by Eysa Lee