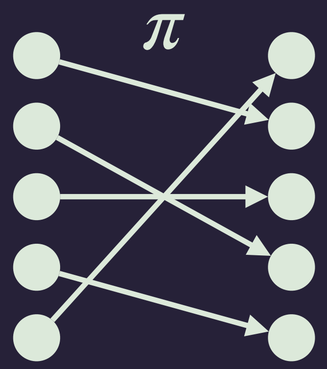
CRYPTO 2026 | AUGUST 19, 2026



Linear-time Shuffling With Malicious Security

begone selective failure attacks!

Rohit Nema (Stanford)

joint work with

Samuel Dittmer
(Stealth Software Technologies)

Rafail Ostrovsky
(UCLA)

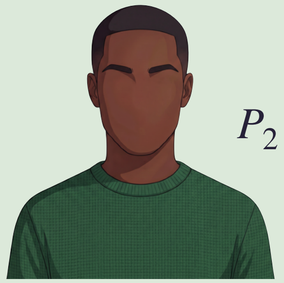
Paper: ia.cr/2025/2137

The Problem: Shuffling Secret-shared Values

P_1

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5



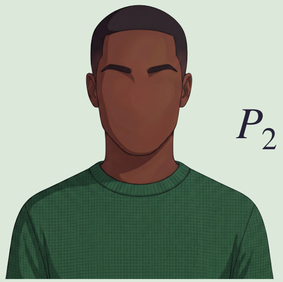
P_2

The Problem: Shuffling Secret-shared Values

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5

P_1



P_2

input

(additive) sharing of $\vec{v}$

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

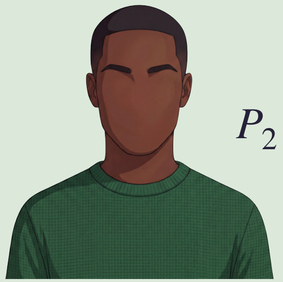
$$[v_i]_1 + [v_i]_2 = v_i$$

The Problem: Shuffling Secret-shared Values

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5

P_1



P_2

input

(additive) sharing of $\vec{v}$

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

$$[v_i]_1 + [v_i]_2 = v_i$$

Goal

Obtain fresh shares of

$$\vec{w} = \pi(\vec{v})$$

for *unknown* permutation π

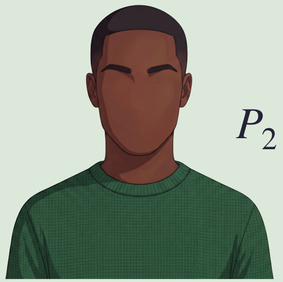
The Problem: Shuffling Secret-shared Values

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5

P_1

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$



P_2

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

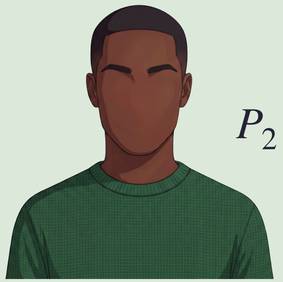
The Problem: Shuffling Secret-shared Values

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5

P_1

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$



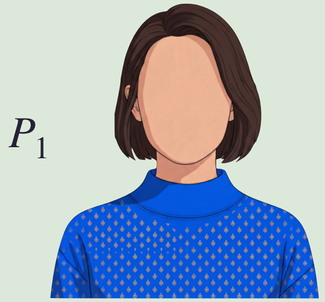
P_2

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

MPC protocol:

random permutation π

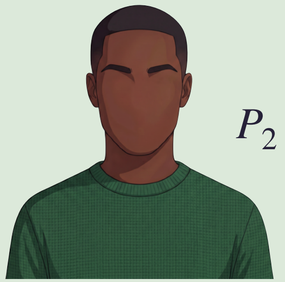
The Problem: Shuffling Secret-shared Values



P_1

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5



P_2

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

MPC protocol:

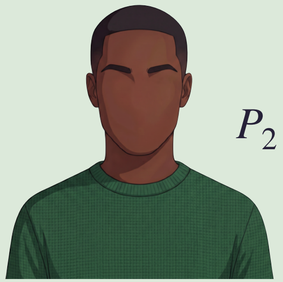
random permutation π

The Problem: Shuffling Secret-shared Values

Vector of values $\vec{v}$

P_1

v_1 v_2 v_3 v_4 v_5



P_2

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

MPC protocol:

random permutation π

$[w_1]_1$ $[w_2]_1$ $[w_3]_1$ $[w_4]_1$ $[w_5]_1$

$[w_1]_2$ $[w_2]_2$ $[w_3]_2$ $[w_4]_2$ $[w_5]_2$

output

sharing of $\vec{w} = \pi(\vec{v})$

$$[w_i]_1 + [w_i]_2 = \pi(\vec{v})_i$$

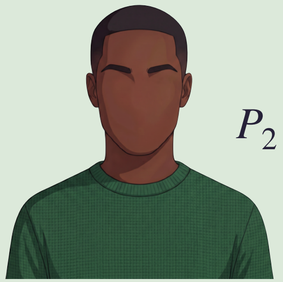
Relaxation: “Half” Shuffle

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5

P_1

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$



P_2

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

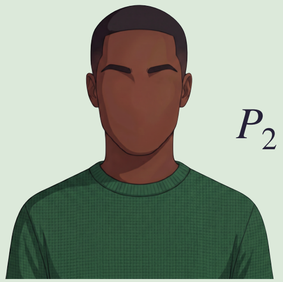
Relaxation: “Half” Shuffle

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5

P_1

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$



P_2

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

Picks a permutation π

Relaxation: “Half” Shuffle

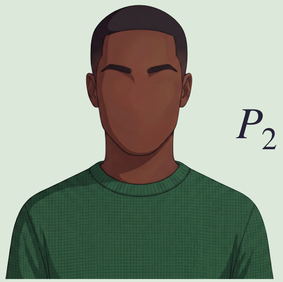
Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5

P_1

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

Picks a permutation π

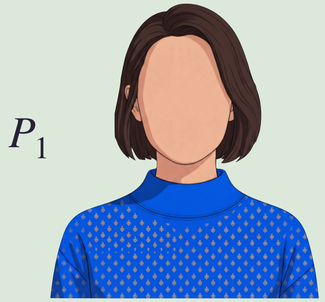


P_2

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

MPC protocol

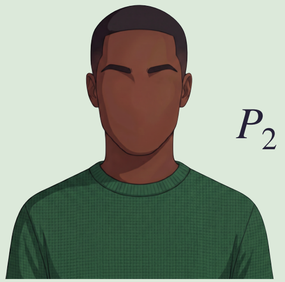
Relaxation: “Half” Shuffle



P_1

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5



P_2

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

Picks a permutation π

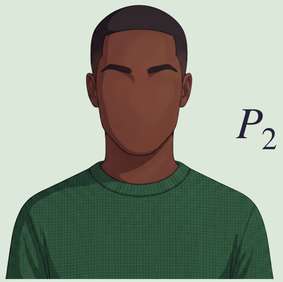
MPC protocol

Relaxation: “Half” Shuffle

P_1

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5



P_2

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

Picks a permutation π

MPC protocol

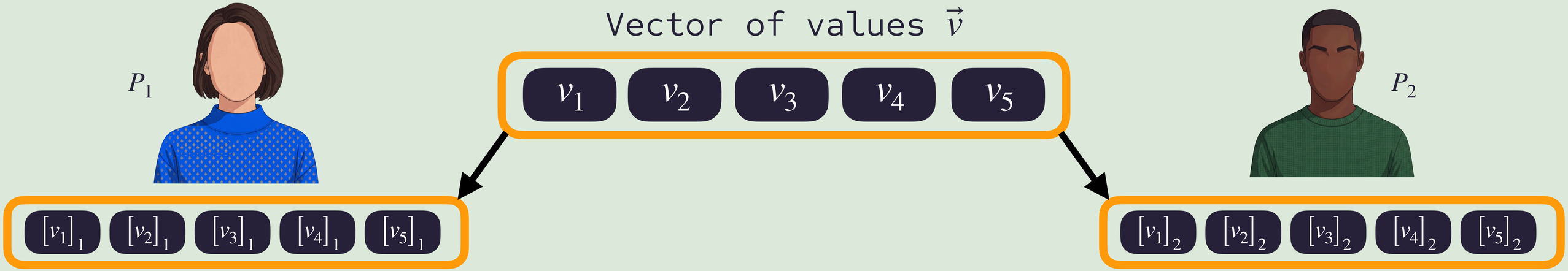
output

sharing of $\vec{w} = \pi(\vec{v})$

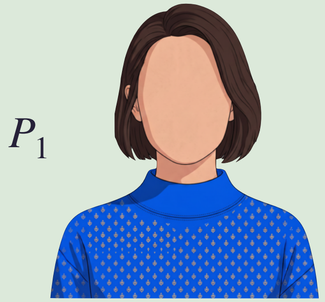
$[w_1]_1$ $[w_2]_1$ $[w_3]_1$ $[w_4]_1$ $[w_5]_1$

$[w_1]_2$ $[w_2]_2$ $[w_3]_2$ $[w_4]_2$ $[w_5]_2$

Shuffle = Two Half-Shuffles



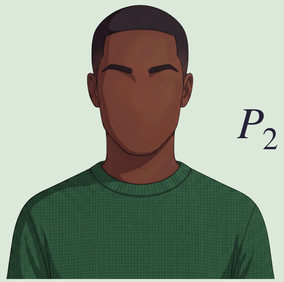
Shuffle = Two Half-Shuffles



P_1

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5



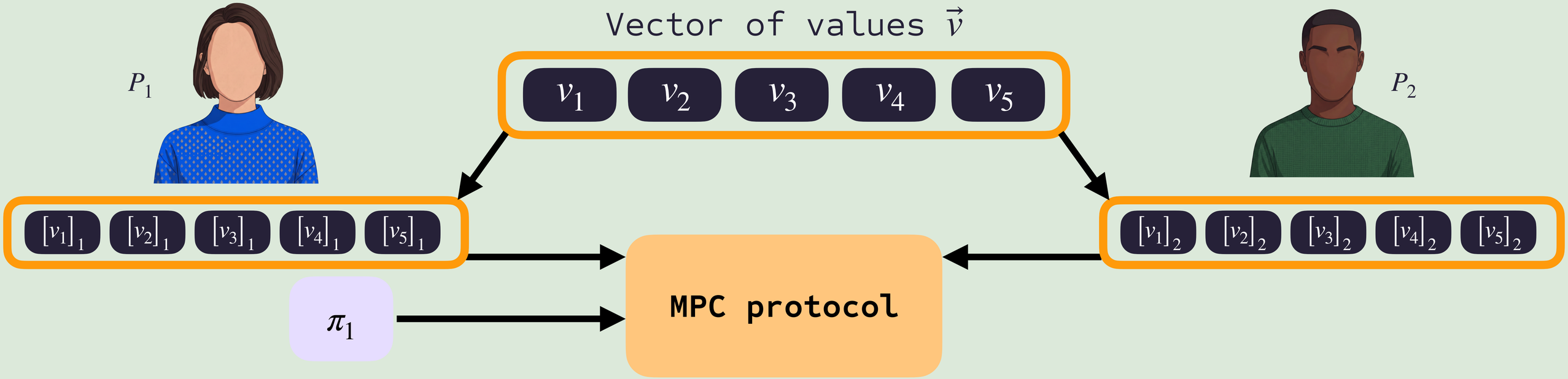
P_2

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

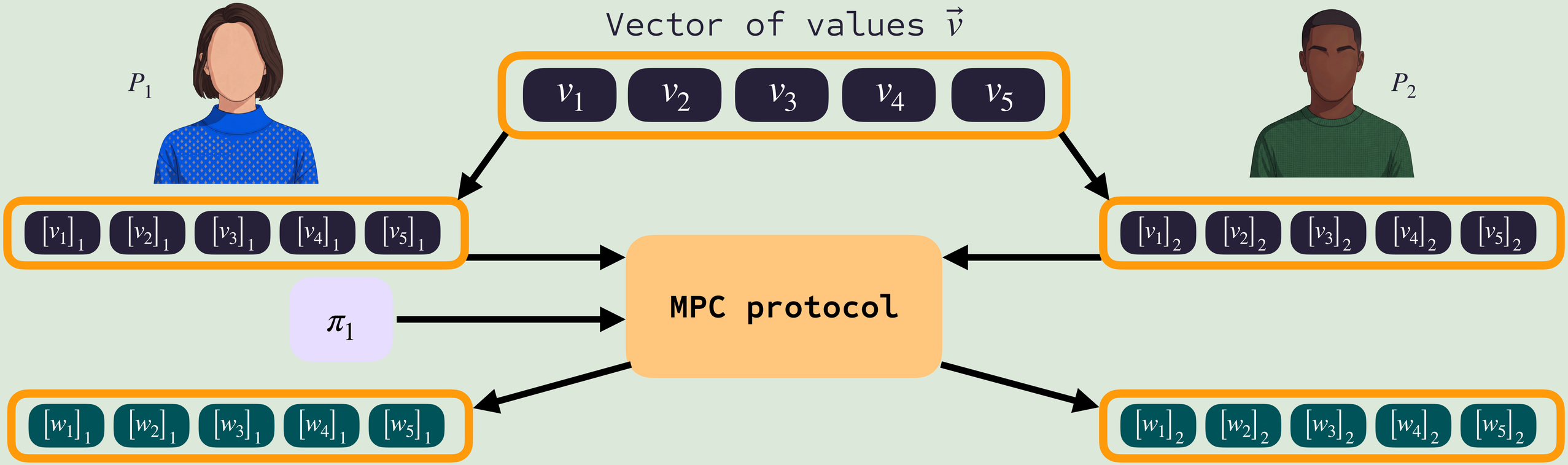
$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

π_1

Shuffle = Two Half-Shuffles



Shuffle = Two Half-Shuffles

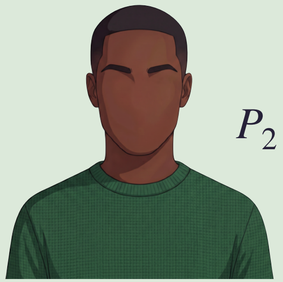


Shuffle = Two Half-Shuffles

P_1

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5



P_2

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

π_1

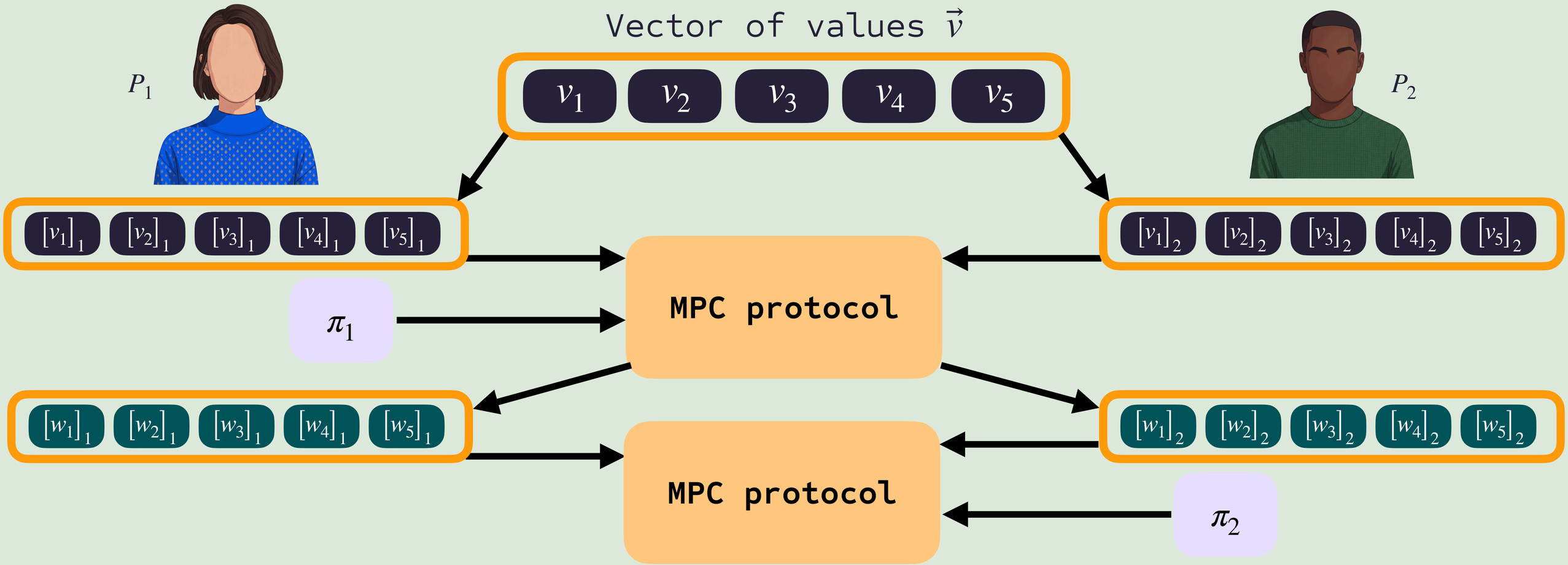
MPC protocol

$[w_1]_1$ $[w_2]_1$ $[w_3]_1$ $[w_4]_1$ $[w_5]_1$

$[w_1]_2$ $[w_2]_2$ $[w_3]_2$ $[w_4]_2$ $[w_5]_2$

π_2

Shuffle = Two Half-Shuffles

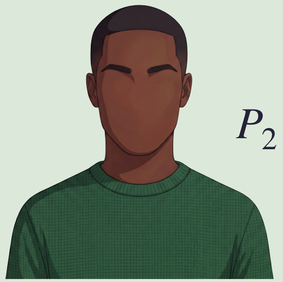


Shuffle = Two Half-Shuffles

P_1

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5



P_2

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

π_1

MPC protocol

$[w_1]_1$ $[w_2]_1$ $[w_3]_1$ $[w_4]_1$ $[w_5]_1$

$[w_1]_2$ $[w_2]_2$ $[w_3]_2$ $[w_4]_2$ $[w_5]_2$

MPC protocol

π_2

$[z_1]_1$ $[z_2]_1$ $[z_3]_1$ $[z_4]_1$ $[z_5]_1$

$[z_1]_2$ $[z_2]_2$ $[z_3]_2$ $[z_4]_2$ $[z_5]_2$

final output

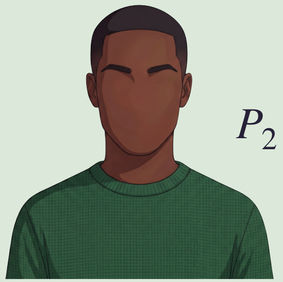
sharing of $\vec{z} = (\pi_2 \circ \pi_1)(\vec{v})$

Shuffle = Two Half-Shuffles

P_1

Vector of values $\vec{v}$

v_1 v_2 v_3 v_4 v_5



P_2

$[v_1]_1$ $[v_2]_1$ $[v_3]_1$ $[v_4]_1$ $[v_5]_1$

$[v_1]_2$ $[v_2]_2$ $[v_3]_2$ $[v_4]_2$ $[v_5]_2$

π_1

MPC protocol

$[w_1]_1$ $[w_2]_1$ $[w_3]_1$ $[w_4]_1$ $[w_5]_1$

$[w_1]_2$ $[w_2]_2$ $[w_3]_2$ $[w_4]_2$ $[w_5]_2$

MPC protocol

π_2

$[z_1]_1$ $[z_2]_1$ $[z_3]_1$ $[z_4]_1$ $[z_5]_1$

$[z_1]_2$ $[z_2]_2$ $[z_3]_2$ $[z_4]_2$ $[z_5]_2$

final output

sharing of $\vec{z} = (\pi_2 \circ \pi_1)(\vec{v})$

$\pi_2 \circ \pi_1$ appears random to both P_1 and P_2

Oblivious Shuffling

Oblivious Shuffling

Highly active area of research

Loads of work on this in various settings:

Permutation matrix: [CDI+24]

Homormorphic encryption-based (folklore): [HEK12,GHJ+15,FO21]

Offline-online: [CGP20,Lau21,EB22,SYB+24,CNO+26]

Verifiable computation: [CF13, GW13, LWZ18, FNP20, BCF+21, GNS23, KVM+24, ACG+25, CCC+25, ...]

Oblivious Shuffling

Highly active area of research

Loads of work on this in various settings:

Permutation matrix: [CDI+24]

Homormorphic encryption-based (folklore): [HEK12,GHJ+15,FO21]

Offline-online: [CGP20,Lau21,EB22,SYB+24,CNO+26]

Verifiable computation: [CF13, GW
LWZ18, FNP20, BCF+21, GNS23, KVM+
ACG+25, CCC+25, ...]

either **only semi-honest security** or **super-linear** in offline or online computation/communication

Oblivious Shuffling

Highly active area of research

Loads of work on this in various settings:

Permutation matrix: [CDI+24]

Homormorphic encryption-based (folklore): [HEK12,GHJ+15,FO21]

Offline-online: [CGP20,Lau21,EB22,SYB+24,CNO+26]

Verifiable computation: [CF13, GW13, LWZ18, FNP20, BCF+21, GNS23, KVM+24, ACG+25, CCC+25, ...]

Applications

Essential building block for privacy-preserving protocols e.g. in

(1) analytics

(2) messaging

(3) RAM computations

and many more...

Our Result

Our Result

Asymptotically linear

Linear-time and space protocol by dealing with the RAM computation directly instead of compiling a circuit

Our Result

Asymptotically linear

Linear-time and space protocol by dealing with the RAM computation directly instead of compiling a circuit

Malicious security

We obtain **security *with abort*** against a **malicious** adversary

Our Result

Asymptotically linear

Linear-time and space protocol by dealing with the RAM computation directly instead of compiling a circuit

Malicious security

We obtain **security *with abort*** against a **malicious** adversary

A new assumption

We introduce a **novel weakening of the “linear-only” assumption** on encryption schemes that limits operations on ciphertexts to that of “weak predicates”

Our Construction: Roadmap

Our Construction: Roadmap

Folklore semi-honest protocol using Linearly Homomorphic Encryption (LHE) and authenticated secret sharing

Our Construction: Roadmap

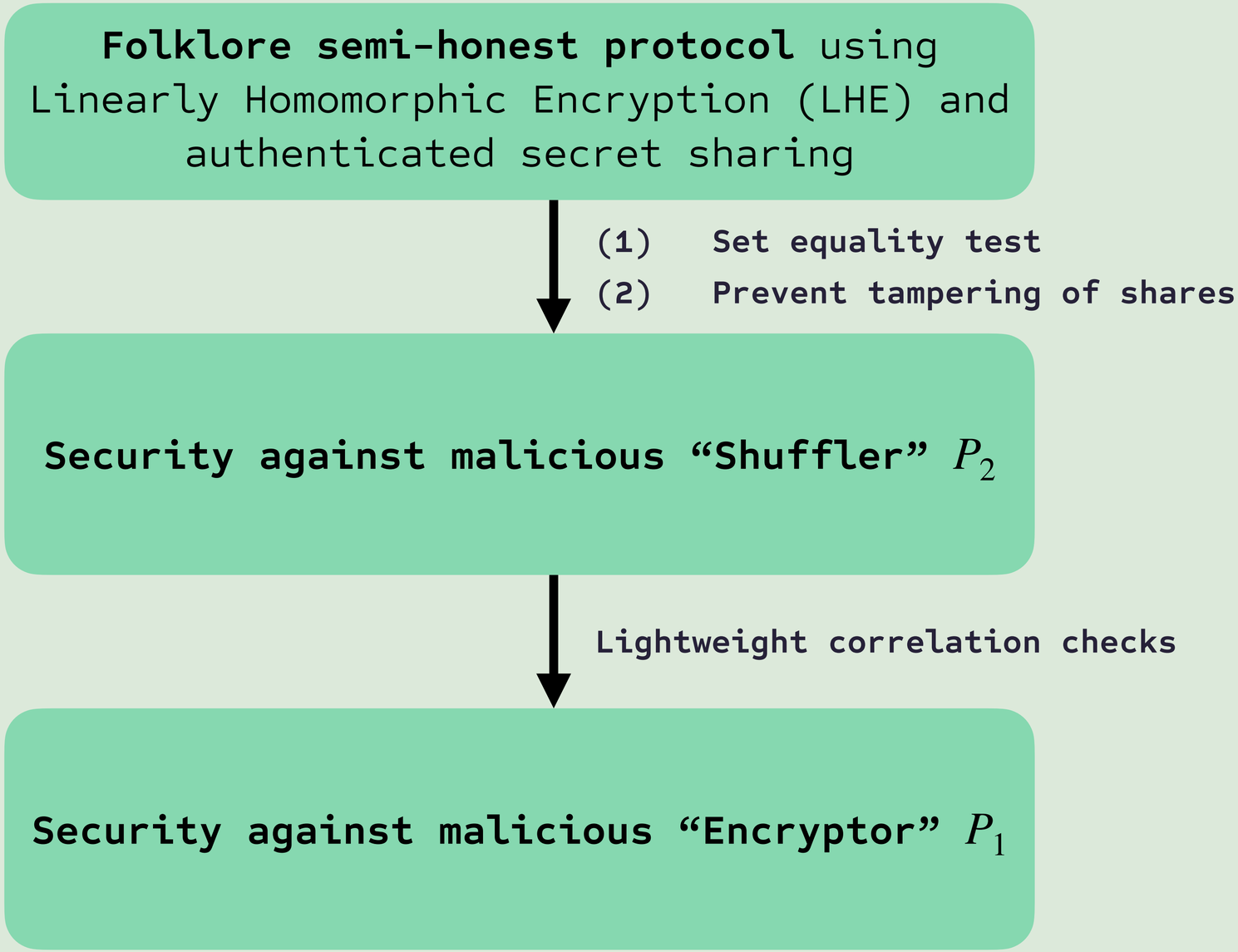
Folklore semi-honest protocol using Linearly Homomorphic Encryption (LHE) and authenticated secret sharing

(1) Set equality test

(2) Prevent tampering of shares

Security against malicious “Shuffler” P_2

Our Construction: Roadmap



Folklore Approach Using LHE

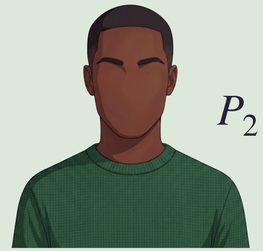
$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

$[\vec{v}]_1$

Encryptor

$[\vec{v}]_2$ π_2



P_2

Shuffler

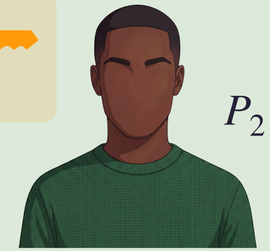
Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

$[\vec{v}]_1$

Encryptor



P_2

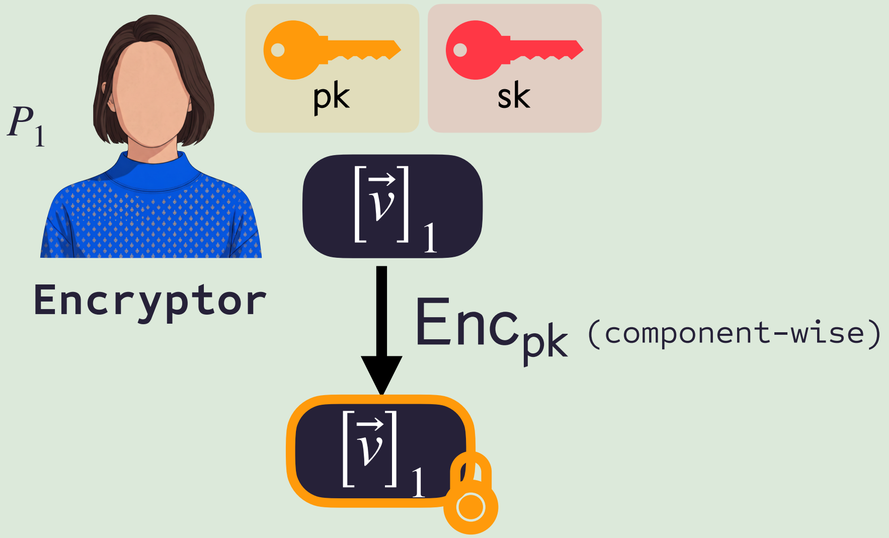
$[\vec{v}]_2$

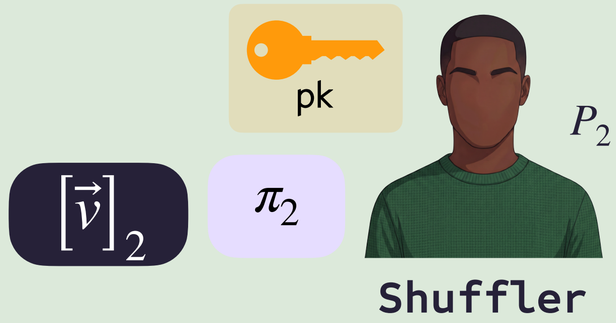
π_2

Shuffler

Folklore Approach Using LHE

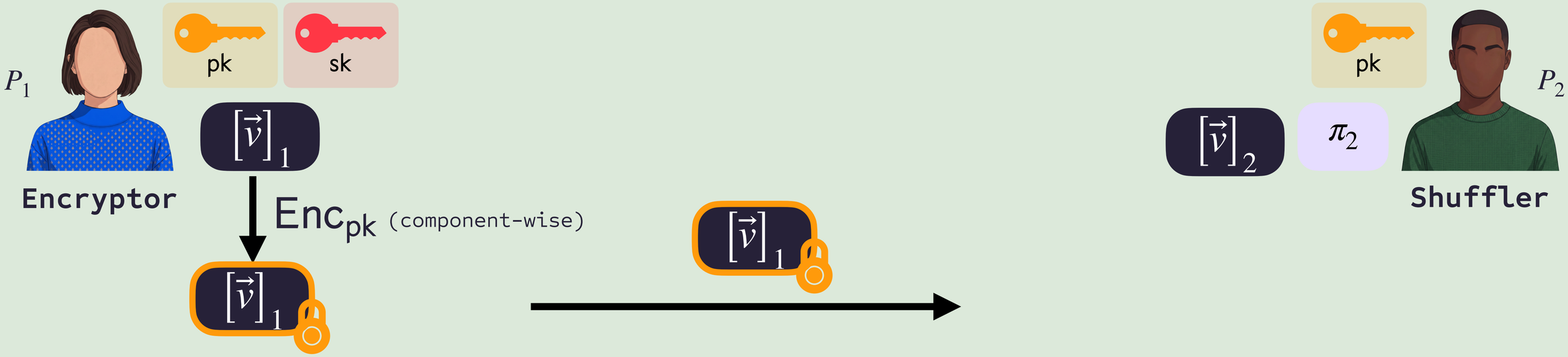
$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$





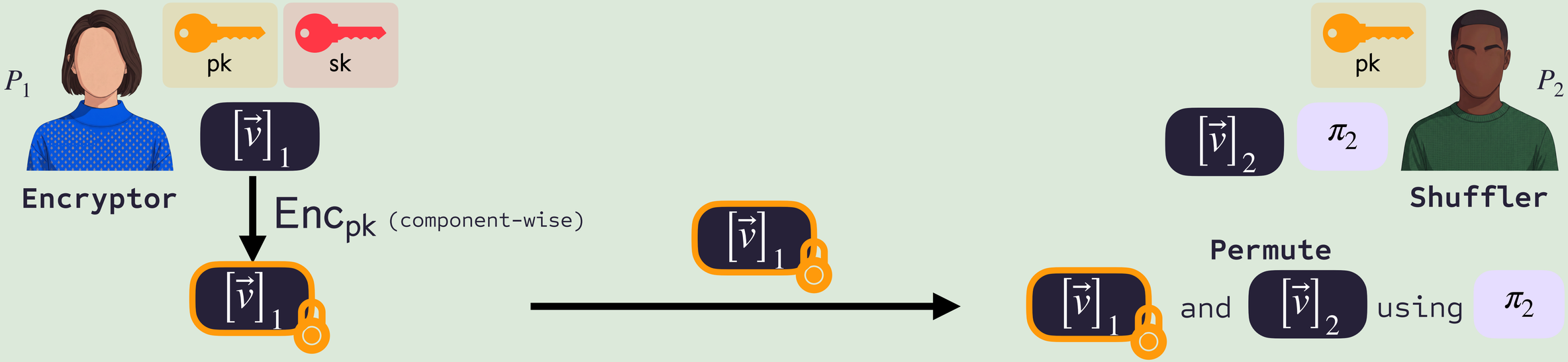
Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$



Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$



Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

$[\vec{v}]_1$

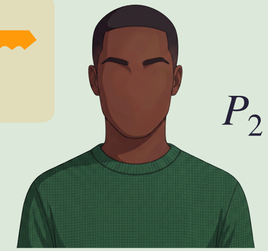
Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

$[\vec{v}]_1$

pk



P_2

$[\vec{v}]_2$

π_2

Shuffler

Permute

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

Linearly re-randomize

$\pi([\vec{v}]_1) + \vec{r}$, $\pi([\vec{v}]_2) - \vec{r}$

Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

$[\vec{v}]_1$

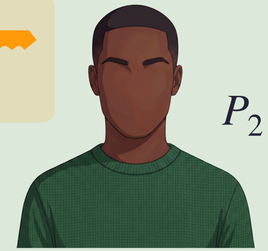
Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

$[\vec{v}]_1$

pk



P_2

$[\vec{v}]_2$

π_2

Shuffler

Permute

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

Linearly re-randomize

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_2) - \vec{r}$

Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

$[\vec{v}]_1$

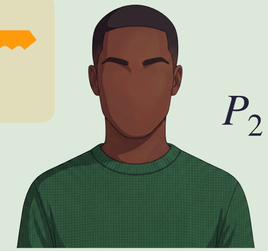
Encryptor

Enc_{pk} (component-wise)

$[\vec{v}]_1$

$[\vec{v}]_1$

pk



P_2

$[\vec{v}]_2$

π_2

Shuffler

Permute

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

Linearly re-randomize

$\pi([\vec{v}]_1) + \vec{r}$, $\pi([\vec{v}]_2) - \vec{r}$

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_1) + \vec{r} = \pi([\vec{v}]_1) + \vec{r}$

Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

$[\vec{v}]_1$

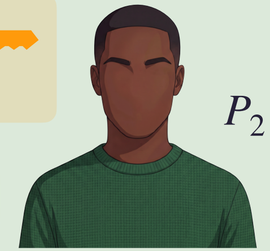
Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

$[\vec{v}]_1$

pk



P_2

$[\vec{v}]_2$

π_2

Shuffler

Permute

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

Linearly re-randomize

$\pi([\vec{v}]_1) + \vec{r}$, $\pi([\vec{v}]_2) - \vec{r}$

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_1) + \vec{r} = \pi([\vec{v}]_1) + \vec{r}$

(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

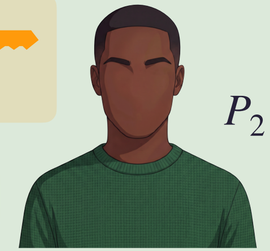
$[\vec{v}]_1$

Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

$[\vec{v}]_1$



P_2

pk

$[\vec{v}]_2$

π_2

Shuffler

Permute

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

Linearly re-randomize

$\pi([\vec{v}]_1) + \vec{r}$, $\pi([\vec{v}]_2) - \vec{r}$

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_1) + \vec{r} = \pi([\vec{v}]_1) + \vec{r}$

(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

$\pi([\vec{v}]_1) + \vec{r}$

Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

$[\vec{v}]_1$

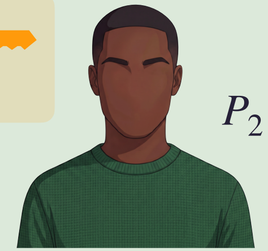
Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

$[\vec{v}]_1$

pk



P_2

$[\vec{v}]_2$

π_2

Shuffler

Permute

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

Linearly re-randomize

$\pi([\vec{v}]_1) + \vec{r}$, $\pi([\vec{v}]_2) - \vec{r}$

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_1) + \vec{r} = \pi([\vec{v}]_1) + \vec{r}$

(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_2) - \vec{r}$

Folklore Approach Using LHE

$[\vec{v}]_i = [[v_1]_i, [v_2]_i, [v_3]_i, [v_4]_i, [v_5]_i]$

P_1

pk

sk

Encryptor

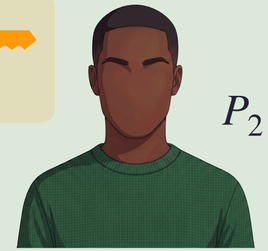
$[\vec{v}]_1$

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

pk

P_2



$[\vec{v}]_2$

π_2

Shuffler

$[\vec{v}]_1$

Permute

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

Linearly re-randomize

$\pi([\vec{v}]_1) + \vec{r}$, $\pi([\vec{v}]_2) - \vec{r}$

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_1) + \vec{r} = \pi([\vec{v}]_1) + \vec{r}$

(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

$[\vec{w}]_1 = \pi([\vec{v}]_1) + \vec{r}$

$[\vec{w}]_2 = \pi([\vec{v}]_2) - \vec{r}$

set output shares

Folklore Approach Using LHE

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

Encryptor

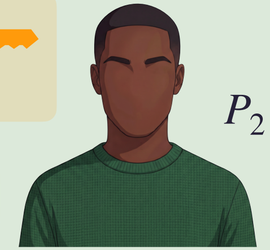
$[\vec{v}]_1$

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

$[\vec{v}]_1$

pk



P_2

$[\vec{v}]_2$

π_2

Shuffler

Permute

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

Linearly re-randomize

$\pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_1) + \vec{r} = \pi([\vec{v}]_1) + \vec{r}$

$\pi([\vec{v}]_1) + \vec{r}$, $\pi([\vec{v}]_2) - \vec{r}$

(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

$[\vec{w}]_1 = \pi([\vec{v}]_1) + \vec{r}$

$[\vec{w}]_2 = \pi([\vec{v}]_2) - \vec{r}$

concretely expensive due to public key operations but asymptomatically linear nonetheless...

Not maliciously secure...

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

$[\vec{v}]_1$

Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

pk

P_2

$[\vec{v}]_2$

π_2

Shuffler

$[\vec{v}]_1$

$[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

can modify shares arbitrarily

$\pi\left([\vec{v}]_1\right) + \vec{r}$

$\pi\left([\vec{v}]_1\right) + \vec{r} = \pi\left([\vec{v}]_1 + \vec{r}\right)$

$\left([\vec{v}]_2\right) - \vec{r}$

(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

$[\vec{w}]_1 = \pi\left([\vec{v}]_1\right) + \vec{r}$

set output shares

$[\vec{w}]_2 = \pi\left([\vec{v}]_2\right) - \vec{r}$

Not maliciously secure...

$[\vec{v}]_i = [v_1]_i \; [v_2]_i \; [v_3]_i \; [v_4]_i \; [v_5]_i$

P_1

pk

sk

$[\vec{v}]_1$

Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$[\vec{v}]_1$

pk

P_2

$[\vec{v}]_2$

π_2

Shuffler

$[\vec{v}]_1$

Permute $[\vec{v}]_1$ and $[\vec{v}]_2$ using π_2

$\pi([\vec{v}]_1) + \vec{r}$

can modify shares arbitrarily

$\pi([\vec{v}]_1) + \vec{r} = \pi([\vec{v}]_1) + \vec{r}$

(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

$[\vec{w}]_1 = \pi([\vec{v}]_1) + \vec{r}$

set output sh

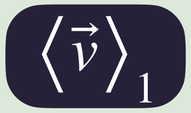
Reconstruct $([\vec{v}]_2) - \vec{r}$

Solution?

use *authenticated* secret sharing

With Authentication

P_1

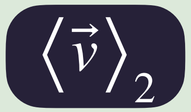


Encryptor

e.g. SPDZ: $\langle x \rangle = ([x], [\alpha \cdot x])$

for $\alpha \xleftarrow{\$} \mathbb{F}$

P_2



Shuffler

With Authentication

P_1

pk

sk

Encryptor

e.g. SPDZ: $\langle x \rangle = ([x], [\alpha \cdot x])$ for $\alpha \xleftarrow{\$} \mathbb{F}$

pk

P_2

Shuffler

$\langle \vec{v} \rangle_2$ π_2

$\langle \vec{v} \rangle_1$

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$\langle \vec{v} \rangle_1$

$\langle \vec{v} \rangle_1$

Permute $\langle \vec{v} \rangle_1$ and $\langle \vec{v} \rangle_2$ using π_2

Linearly re-randomize

$\pi(\langle \vec{v} \rangle_1) + \vec{r}$, $\pi(\langle \vec{v} \rangle_2) - \vec{r}$

$\pi(\langle \vec{v} \rangle_1) + \vec{r}$

$\pi(\langle \vec{v} \rangle_1) + \vec{r} = \pi(\langle \vec{v} \rangle_1) + \vec{r}$

(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

$\langle \vec{w} \rangle_1 = \pi(\langle \vec{v} \rangle_1) + \vec{r}$

set output shares

$\langle \vec{w} \rangle_2 = \pi(\langle \vec{v} \rangle_2) - \vec{r}$

With Authentication

P_1

pk

sk

$\langle \vec{v} \rangle_1$

Encryptor

e.g. SPDZ: $\langle x \rangle = ([x], [\alpha \cdot x])$ for $\alpha \xleftarrow{\$} \mathbb{F}$

pk

P_2

$\langle \vec{v} \rangle_2$

π_2

Shuffler

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$\langle \vec{v} \rangle_1$

$\langle \vec{v} \rangle_1$

$\langle \vec{v} \rangle_1$ and $\langle \vec{v} \rangle_2$ using π_2

can no longer modify arbitrarily without being caught by authentication check

$\pi(\langle \vec{v} \rangle_1) + \vec{r}$

$\pi(\langle \vec{v} \rangle_1) + \vec{r} = \pi(\langle \vec{v} \rangle_1 + \vec{r})$

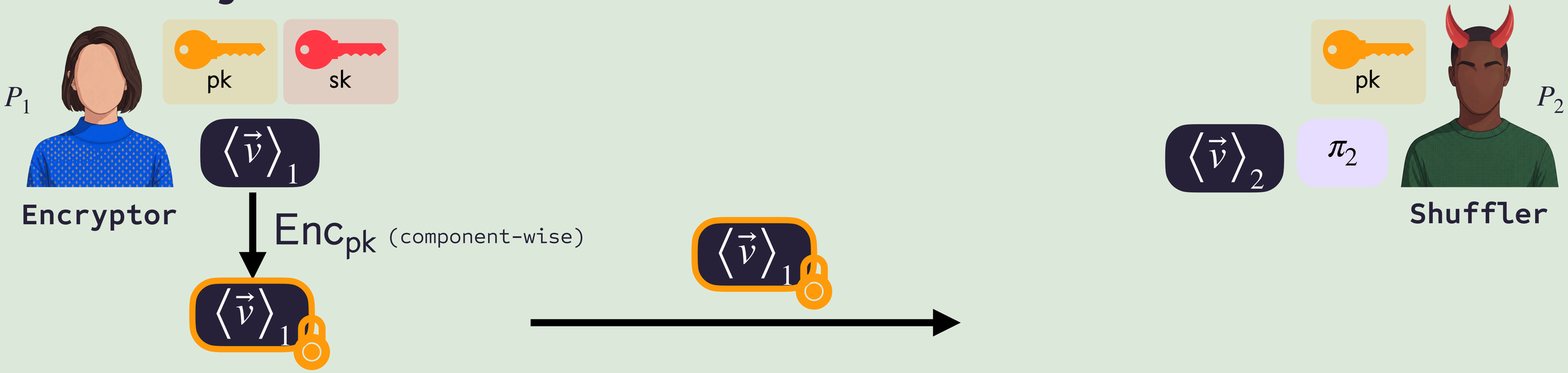
(component-wise) $\mathsf{Dec}_{\mathsf{sk}}$

$\langle \vec{w} \rangle_1 = \pi(\langle \vec{v} \rangle_1) + \vec{r}$

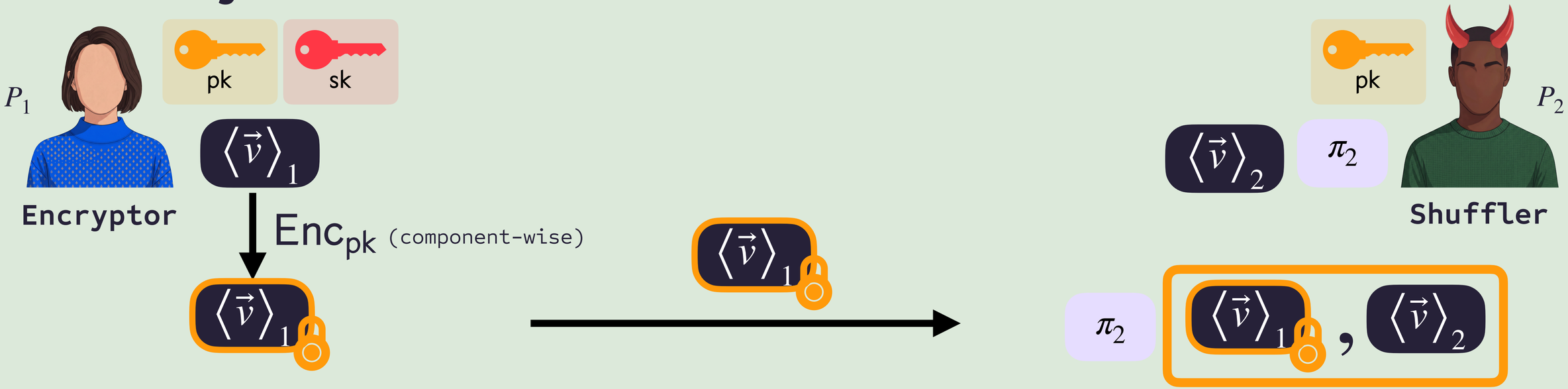
set output shares

$\langle \vec{w} \rangle_2 = \pi(\langle \vec{v} \rangle_2) - \vec{r}$

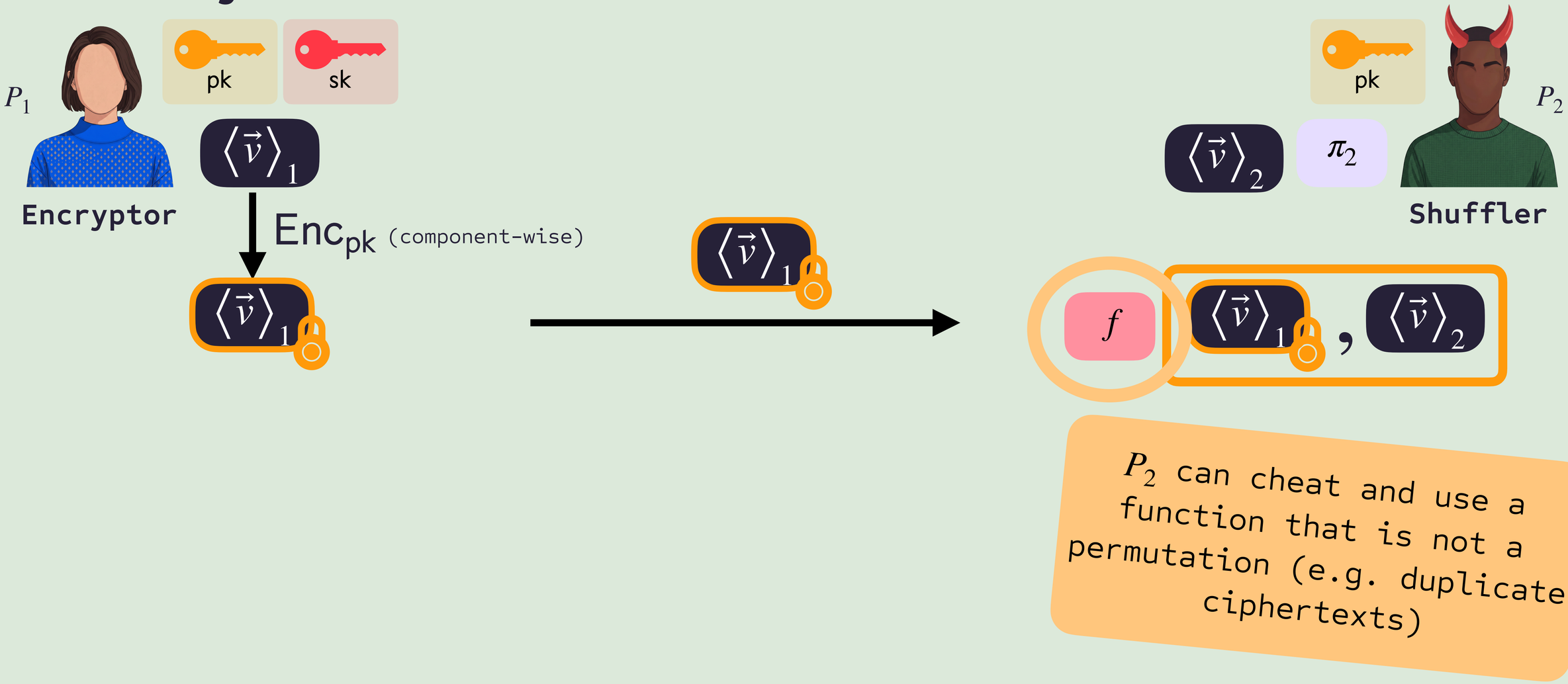
Validity of Permutation



Validity of Permutation

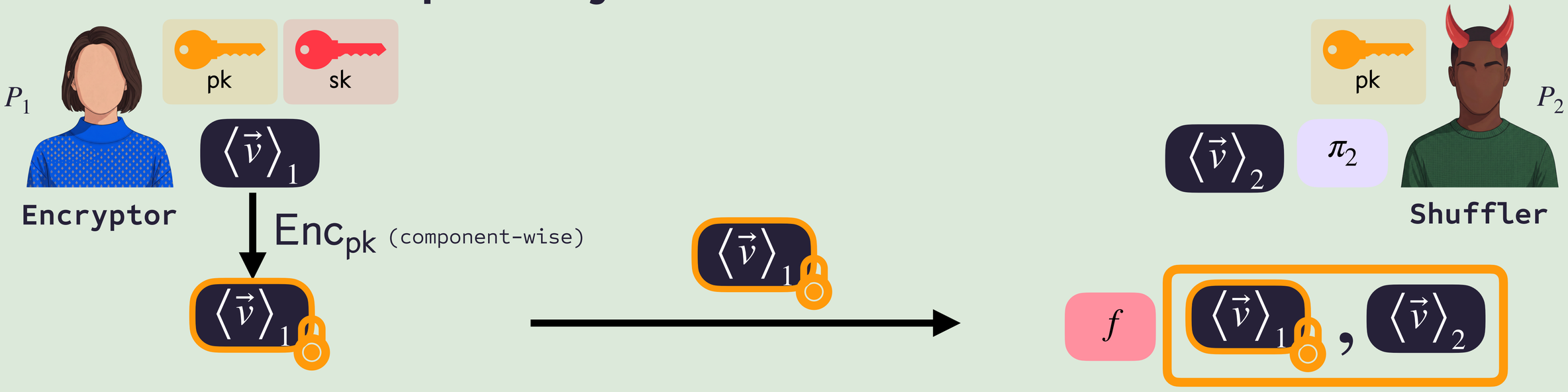


Validity of Permutation



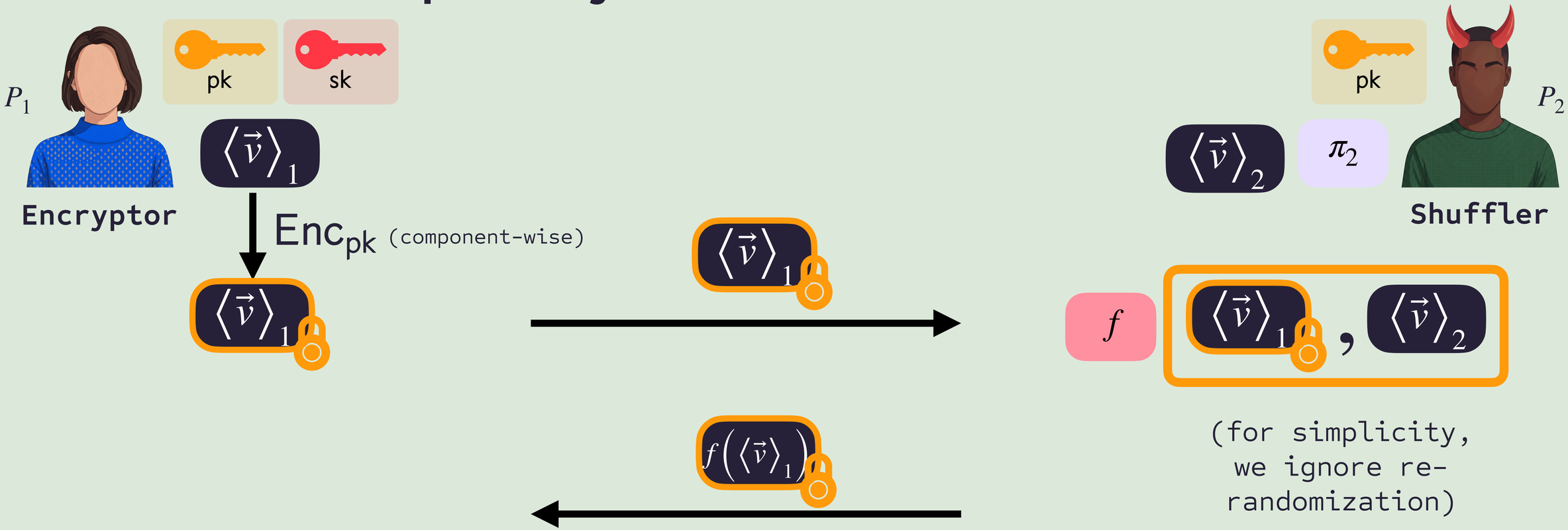
P_2 can cheat and use a function that is not a permutation (e.g. duplicate ciphertexts)

Idea 1: Set Equality Test



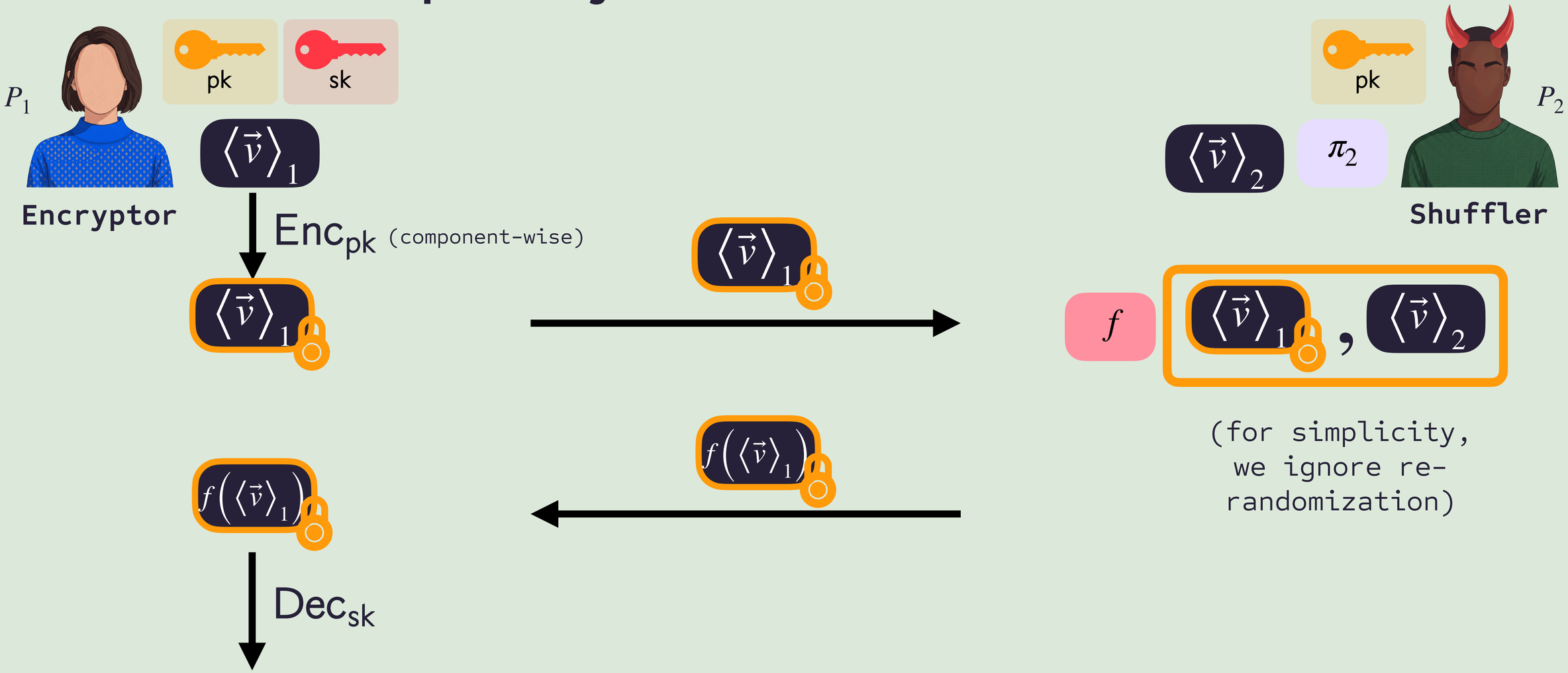
(for simplicity, we ignore re-randomization)

Idea 1: Set Equality Test



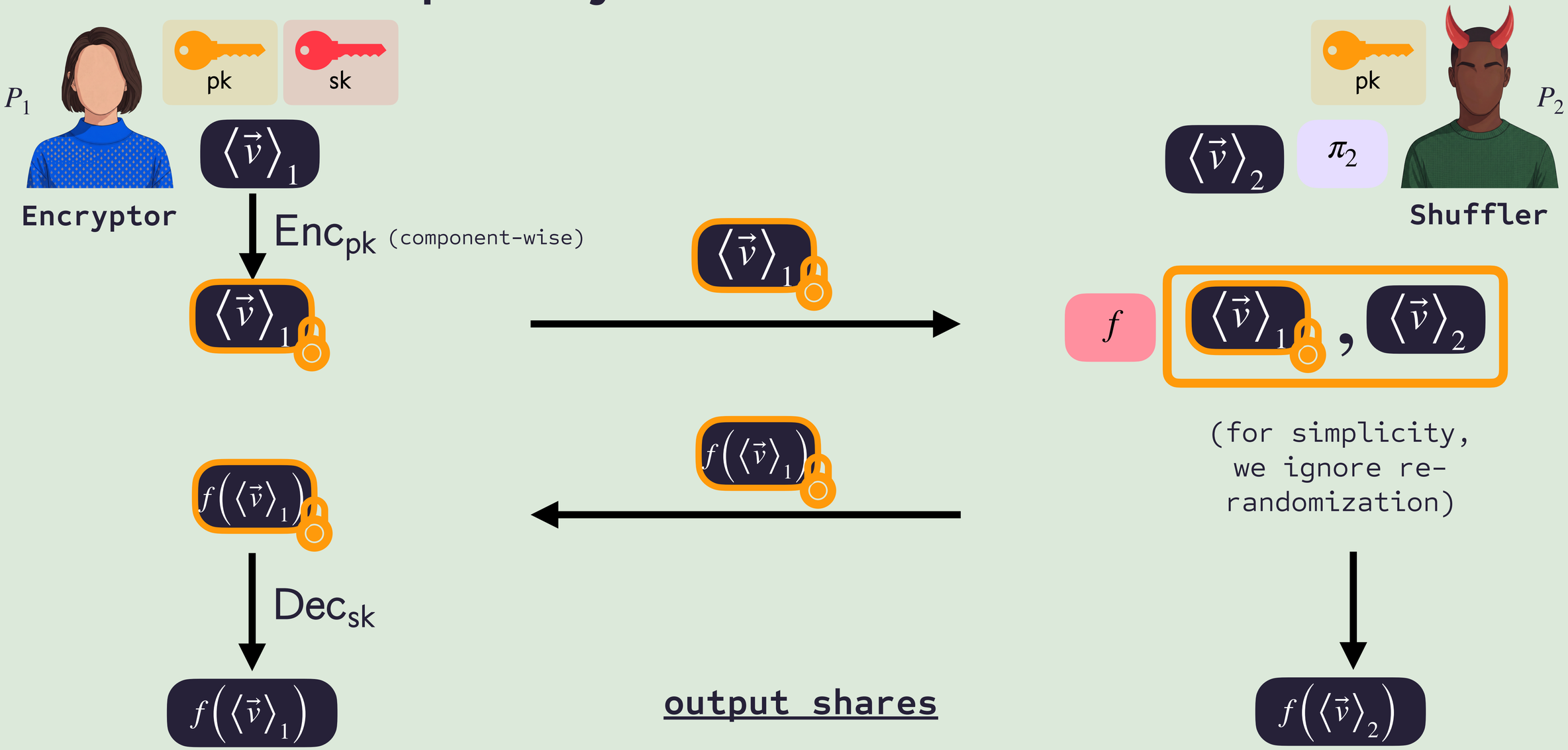
(for simplicity, we ignore re-randomization)

Idea 1: Set Equality Test



(for simplicity, we ignore re-randomization)

Idea 1: Set Equality Test



Idea 1: Set Equality Test

P_1

pk

sk

$\langle \vec{v} \rangle_1$

Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (component-wise)

$\langle \vec{v} \rangle_1$

$\langle \vec{v} \rangle_1$

$\langle \vec{v} \rangle_2$

π_2

pk

P_2

Shuffler

f

$\langle \vec{v} \rangle_1$, $\langle \vec{v} \rangle_2$

(for simplicity, we ignore re-randomization)

Use a set equality test to check the **output shares** represent the **same set** as the **input shares** $\Longrightarrow$ **valid permutation**

$f\left(\langle \vec{v} \rangle_1\right)$

$\mathsf{Dec}_{\mathsf{sk}}$

$f\left(\langle \vec{v} \rangle_1\right)$

output shares

$f\left(\langle \vec{v} \rangle_2\right)$

Idea 1: Set Equality Test

P_1

pk

sk

$\langle \vec{v} \rangle_1$

Encryptor

$\mathsf{Enc}_{\mathsf{pk}}$ (comp

$\langle \vec{v} \rangle_1$

$\langle \vec{v} \rangle_2$

π_2

pk

P_2

Shuffler

BUT... P_2 can now learn information about $\vec{v}$ based on whether the test fails!

$\langle \vec{v} \rangle_1$, $\langle \vec{v} \rangle_2$

Use a set equality test to check the **output shares** represent the **same set** as the **input shares** $\Longrightarrow$ **valid permutation**

(for simplicity, we ignore re-randomization)

$f(\langle \vec{v} \rangle_1)$

$\mathsf{Dec}_{\mathsf{sk}}$

$f(\langle \vec{v} \rangle_1)$

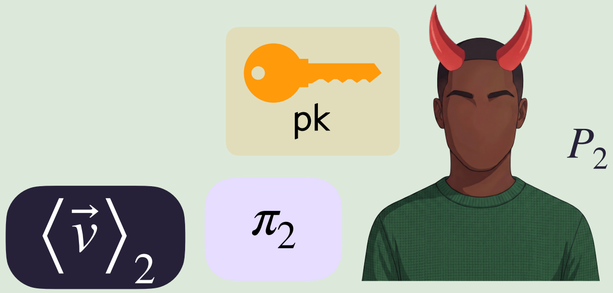
output shares

$f(\langle \vec{v} \rangle_2)$

Selective Failure Attack

Problem:

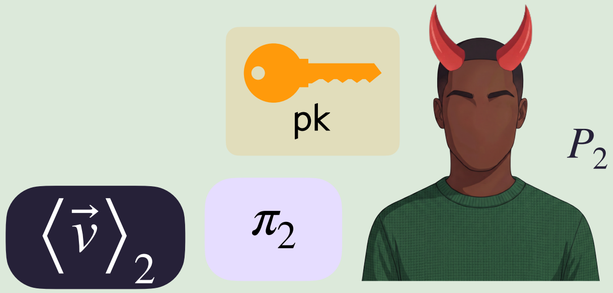
P_2 can exploit linearity to test linear relations over $\vec{v}$



Selective Failure Attack

Problem:

P_2 can exploit linearity to test linear relations over $\vec{v}$



e.g.

test $v_1 + v_2 \stackrel{?}{=} v_3$

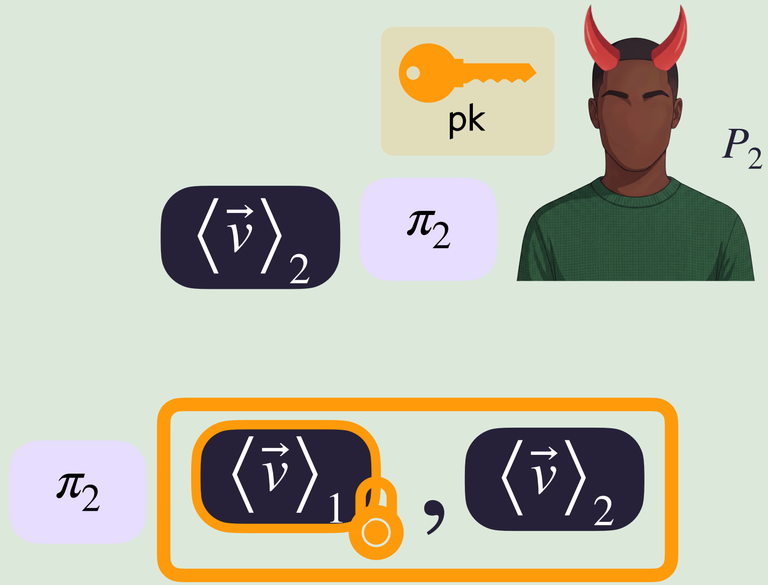
Selective Failure Attack

Problem:

P_2 can exploit linearity to test linear relations over $\vec{v}$

e.g.

test $v_1 + v_2 \stackrel{?}{=} v_3$



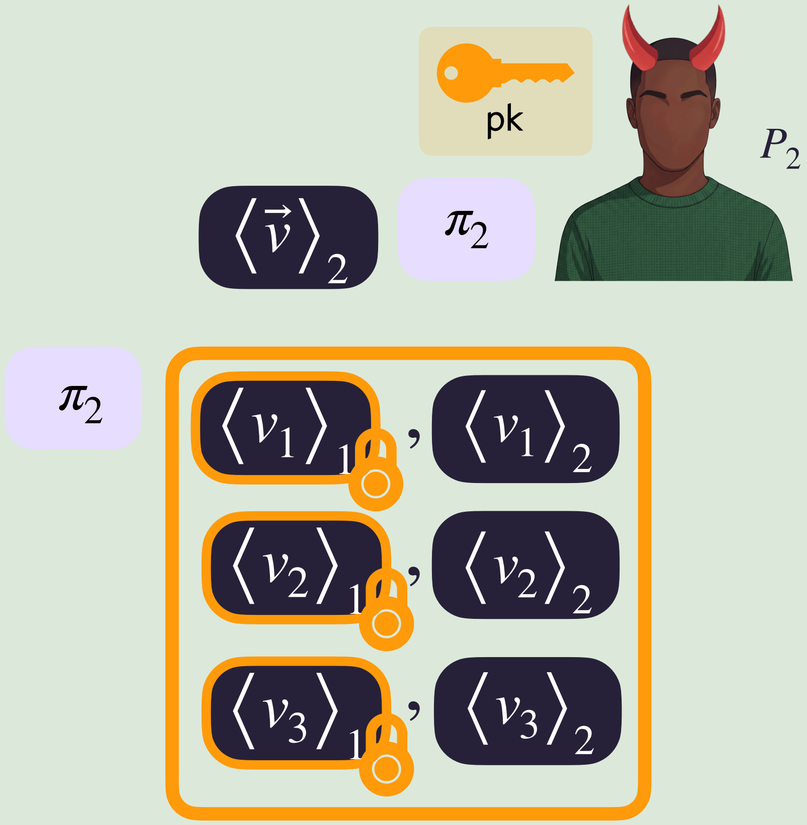
Selective Failure Attack

Problem:

P_2 can exploit linearity to test linear relations over $\vec{v}$

e.g.

test $v_1 + v_2 \stackrel{?}{=} v_3$



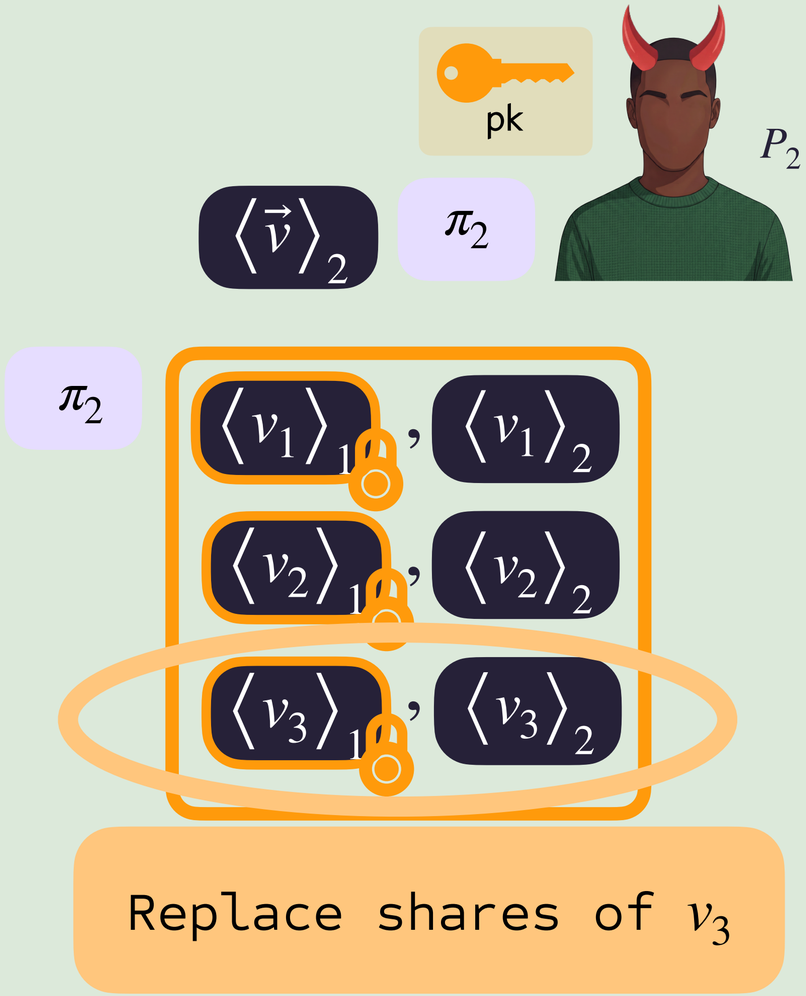
Selective Failure Attack

Problem:

P_2 can exploit linearity to test linear relations over $\vec{v}$

e.g.

test $v_1 + v_2 \stackrel{?}{=} v_3$



Replace shares of v_3

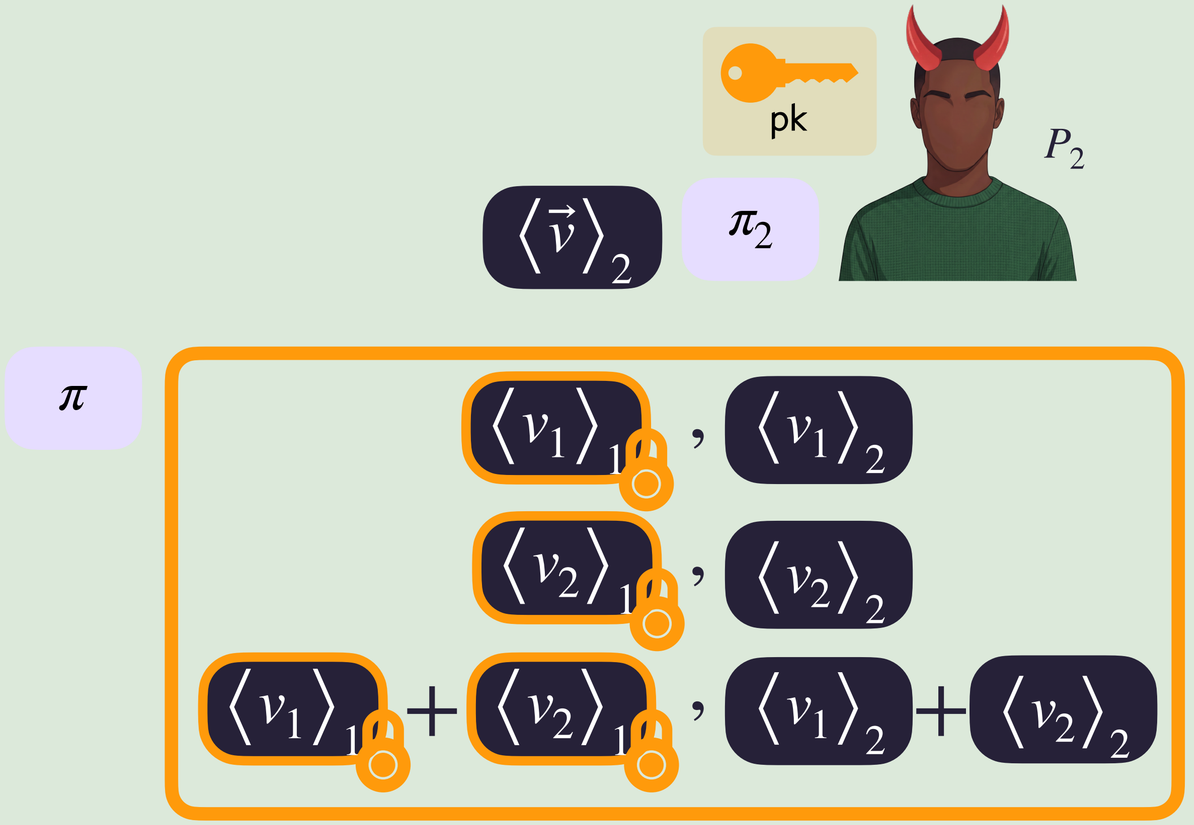
Selective Failure Attack

Problem:

P_2 can exploit linearity to test linear relations over $\vec{v}$

e.g.

test $v_1 + v_2 \stackrel{?}{=} v_3$



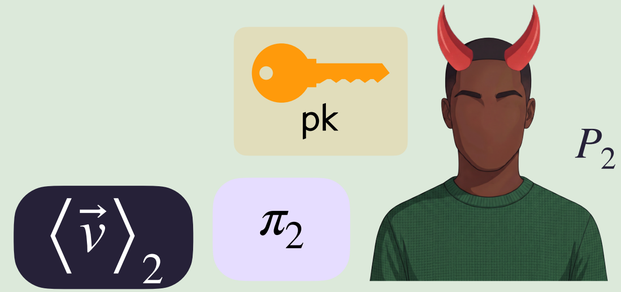
Selective Failure Attack

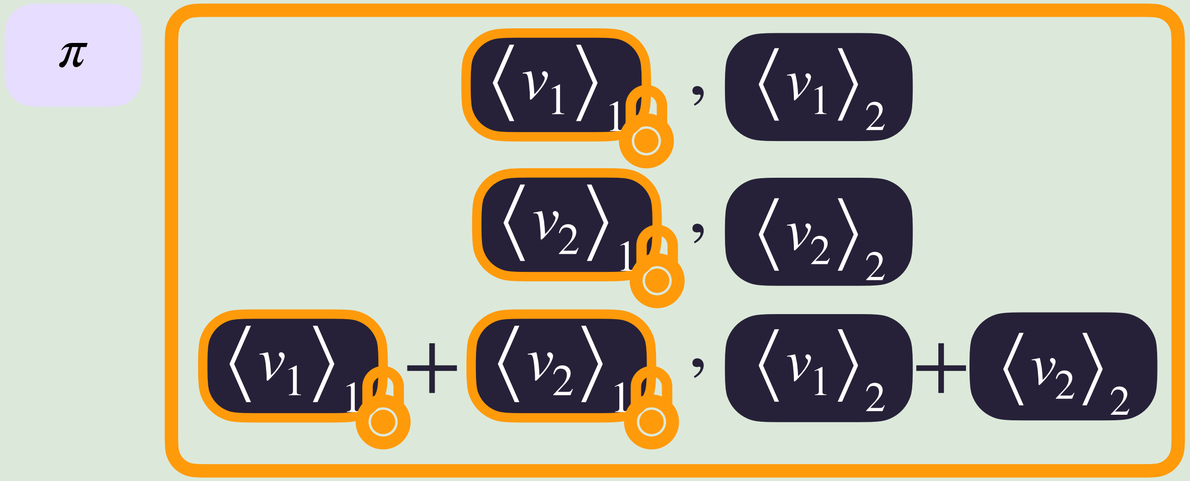
Problem:

P_2 can exploit linearity to test linear relations over $\vec{v}$

e.g.

test $v_1 + v_2 \stackrel{?}{=} v_3$

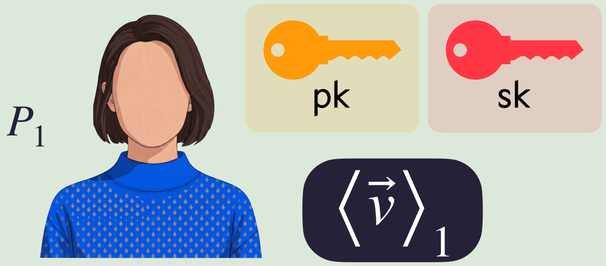


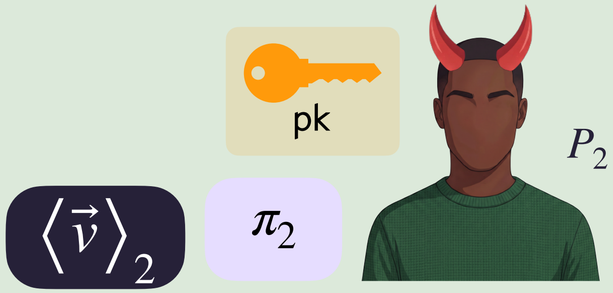


P_1 **aborts set equality test** $\implies v_1 + v_2 \neq v_3$

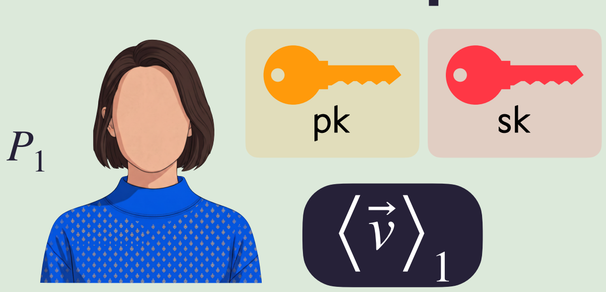
P_1 **does not abort** $\implies v_1 + v_2 = v_3$

Idea 2: Split Shares Randomly



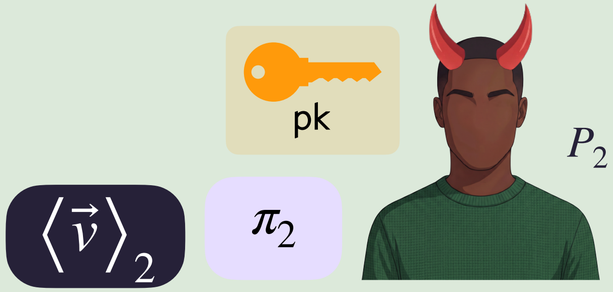


Idea 2: Split Shares Randomly

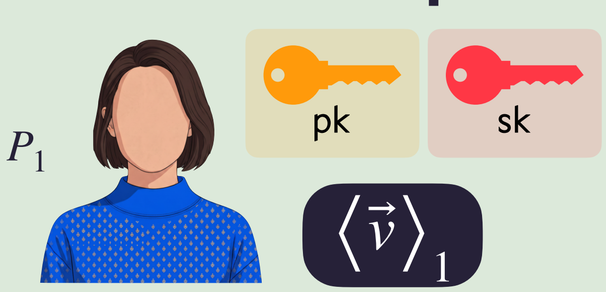


Sample shares of a **random masking vector** $\langle \vec{m} \rangle$

e.g. using SPDZ preprocessing

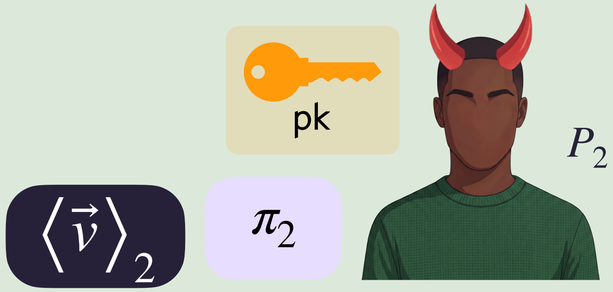


Idea 2: Split Shares Randomly



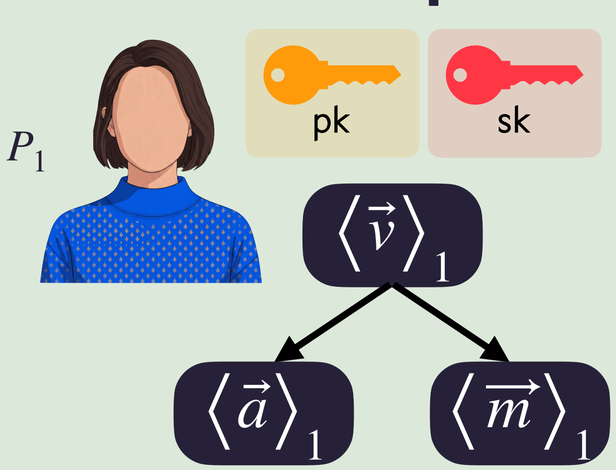
Sample shares of a **random masking vector** $\langle \vec{m} \rangle$

e.g. using SPDZ preprocessing



Compute $\langle \vec{a} \rangle \leftarrow \langle \vec{v} \rangle + \langle \vec{m} \rangle$

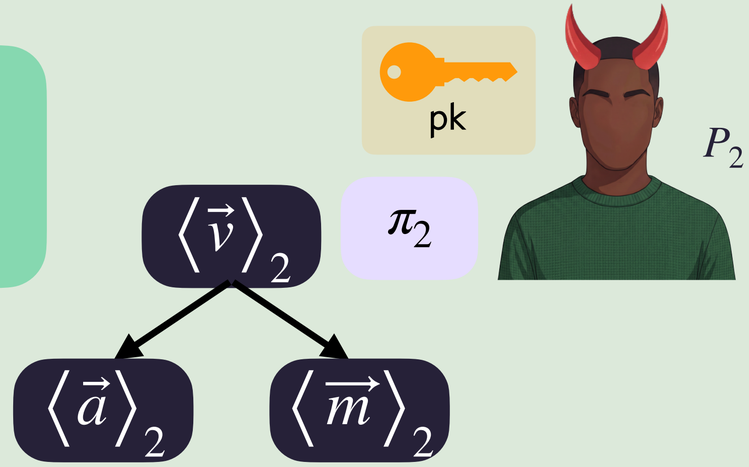
Idea 2: Split Shares Randomly



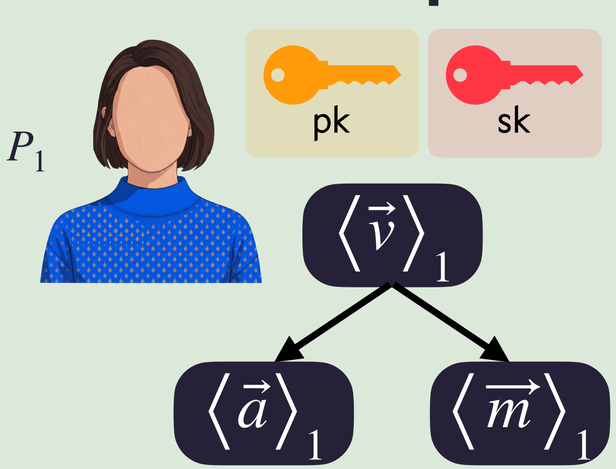
Sample shares of a **random masking vector** $\langle \vec{m} \rangle$

e.g. using SPDZ preprocessing

Compute $\langle \vec{a} \rangle \leftarrow \langle \vec{v} \rangle + \langle \vec{m} \rangle$



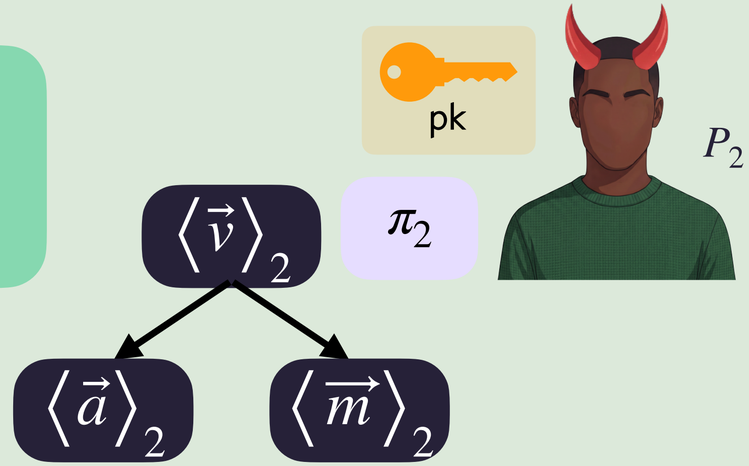
Idea 2: Split Shares Randomly



Sample shares of a **random masking vector** $\langle \vec{m} \rangle$

e.g. using SPDZ preprocessing

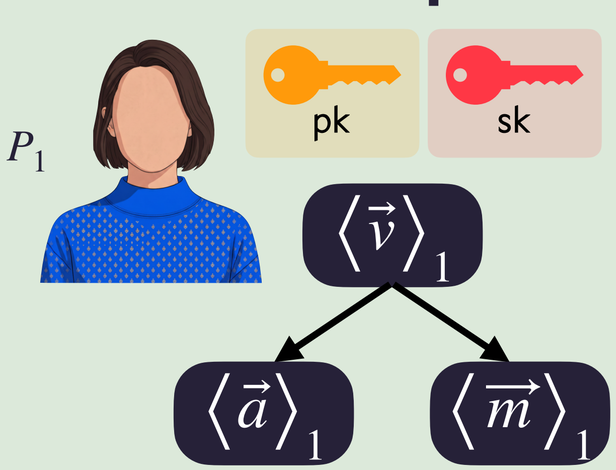
Compute $\langle \vec{a} \rangle \leftarrow \langle \vec{v} \rangle + \langle \vec{m} \rangle$



Do the protocol on $\vec{a}$ and $\vec{m}$ simultaneously instead of directly on $\vec{v}$;

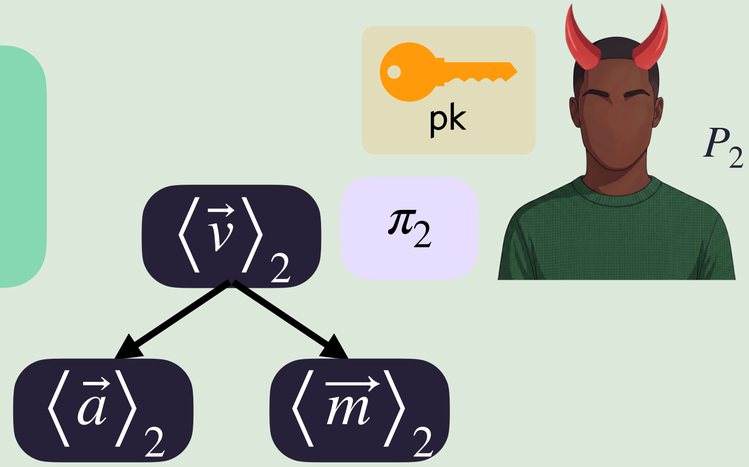
in the end, reconstruct $\vec{w} = \pi_2(\vec{v})$

Idea 2: Split Shares Randomly



Sample shares of a **random masking vector** $\langle \vec{m} \rangle$

e.g. using SPDZ preprocessing



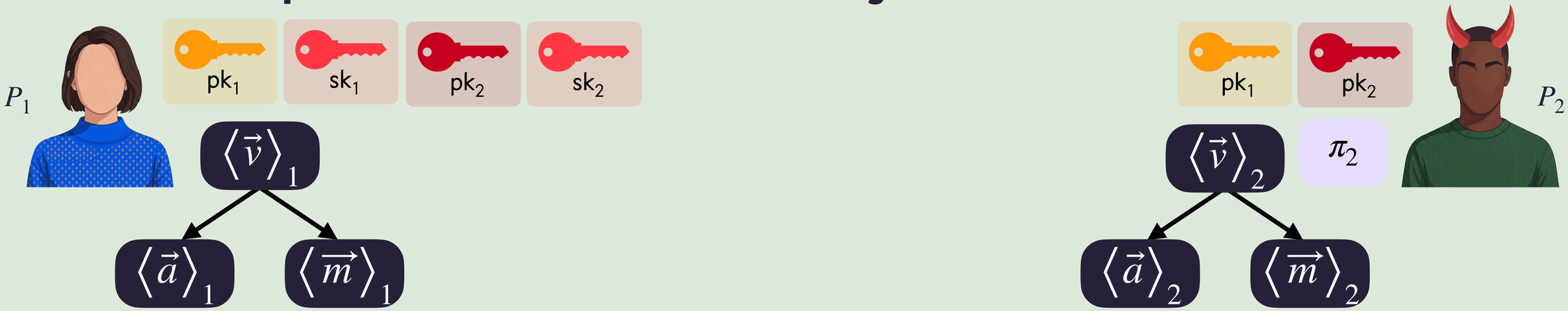
Compute $\langle \vec{a} \rangle \leftarrow \langle \vec{v} \rangle + \langle \vec{m} \rangle$

Do the protocol on $\vec{a}$ and $\vec{m}$ simultaneously instead of directly on $\vec{v}$;

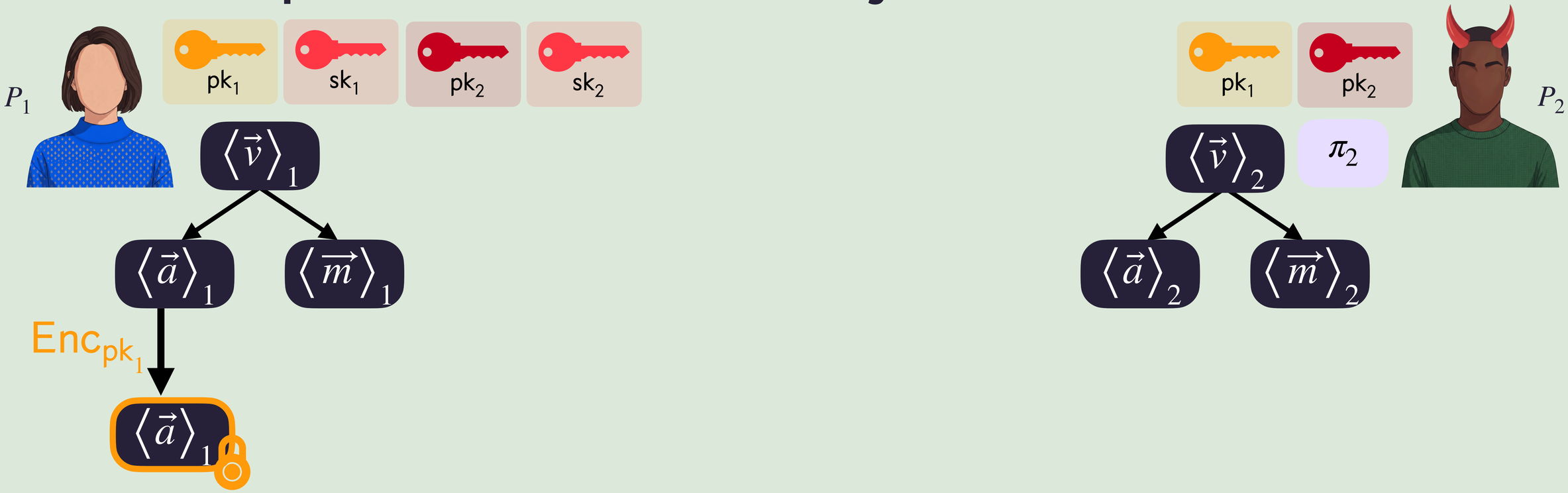
in the end, reconstruct $\vec{w} = \pi_2(\vec{v})$

To prevent linear attacks, encrypt using ***different*** **public keys**

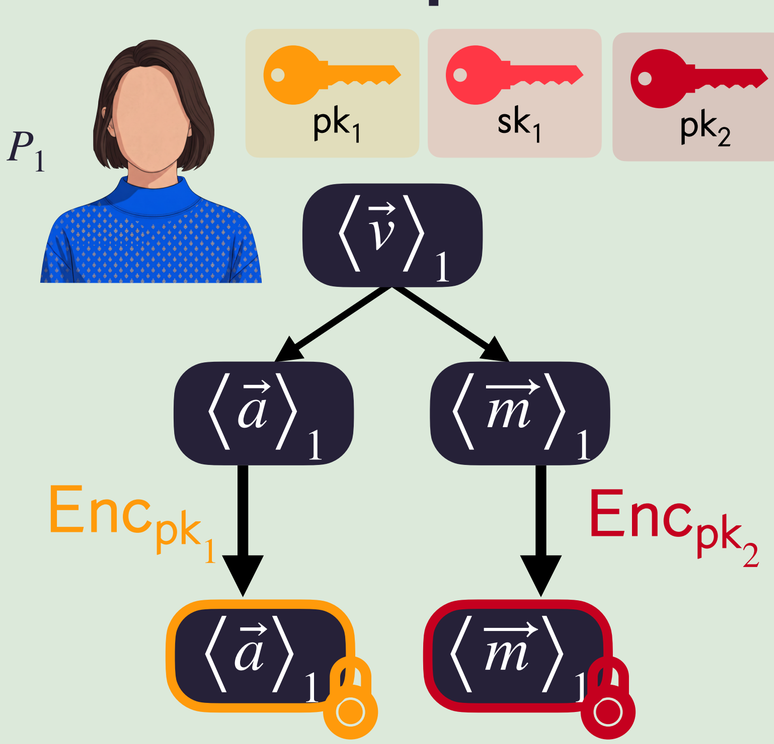
Idea 2: Split Shares Randomly

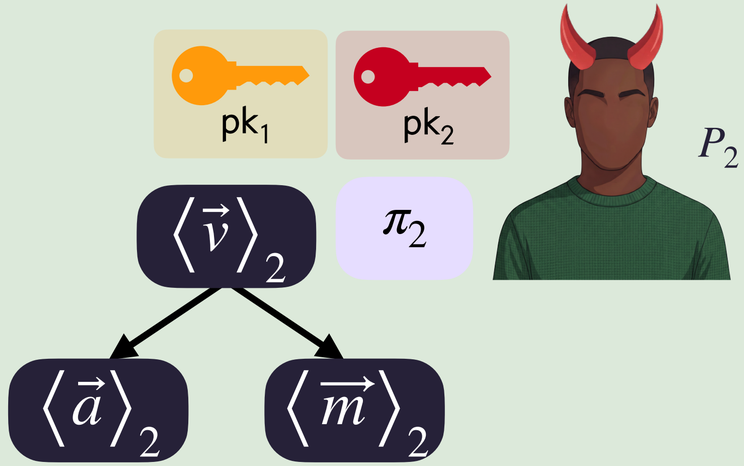


Idea 2: Split Shares Randomly

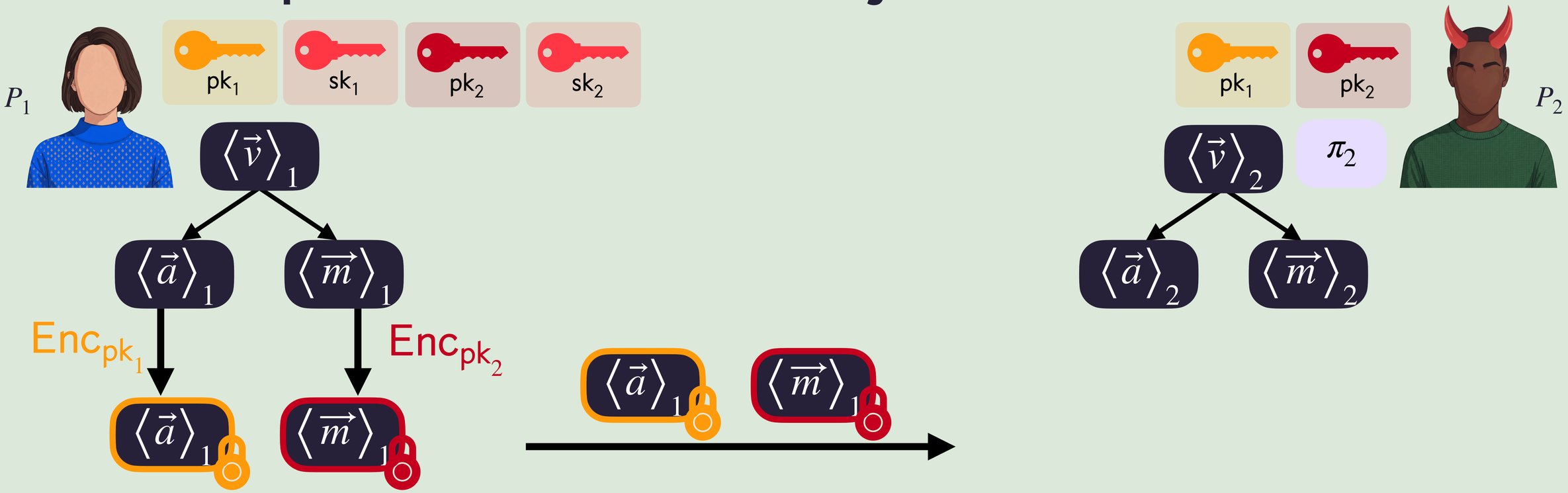


Idea 2: Split Shares Randomly

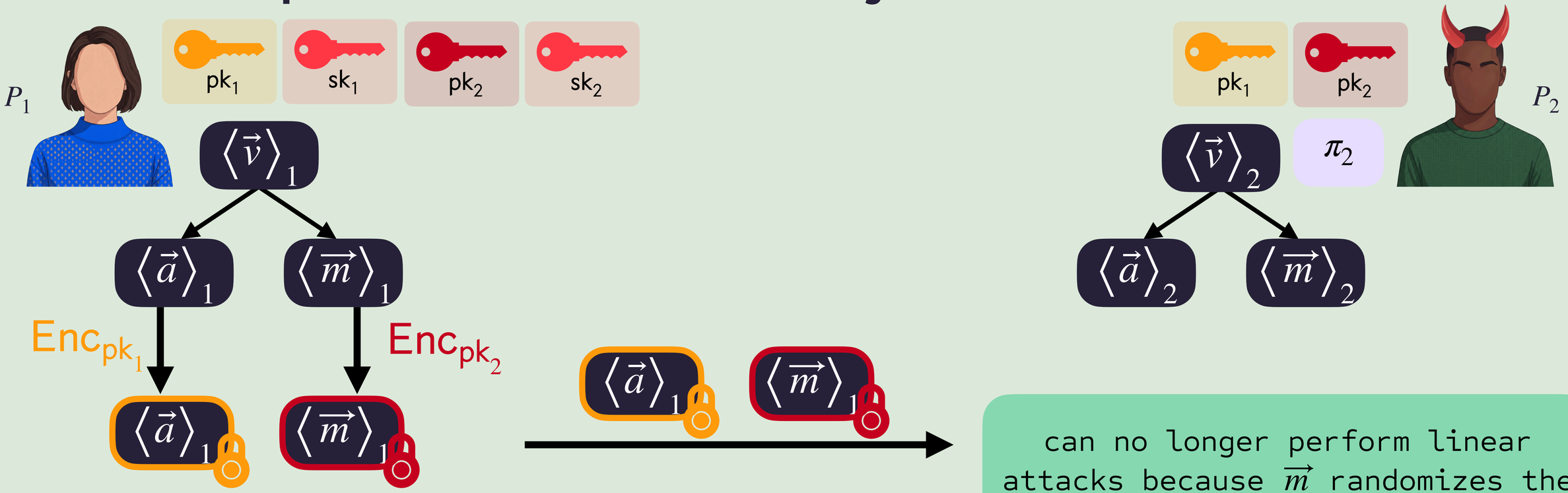




Idea 2: Split Shares Randomly

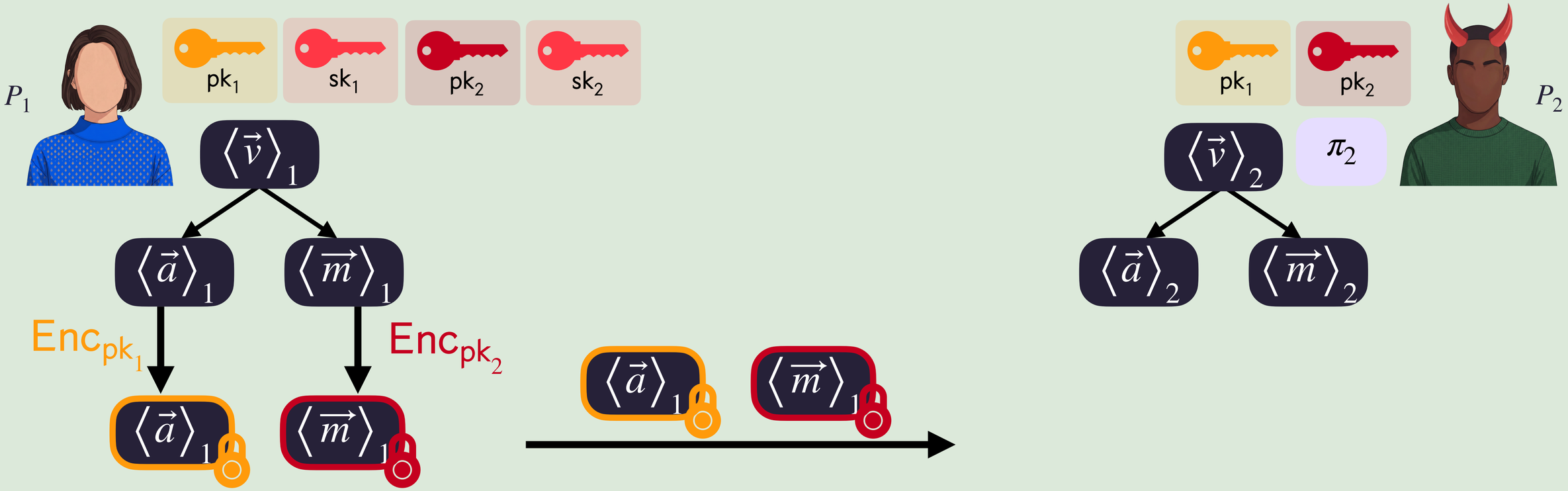


Idea 2: Split Shares Randomly

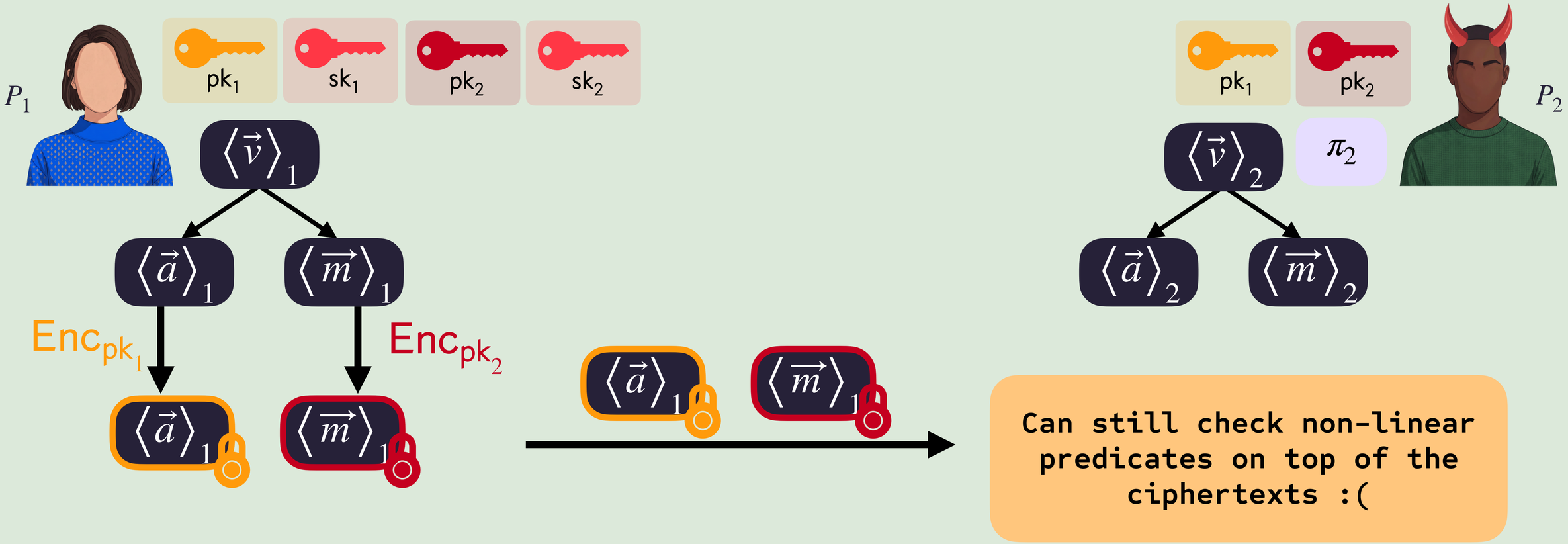


can no longer perform linear attacks because $\vec{m}$ randomizes the input $\vec{v}$, and ciphertexts of $\vec{a}$ and $\vec{m}$ are under different keys

What about non-linear attacks?

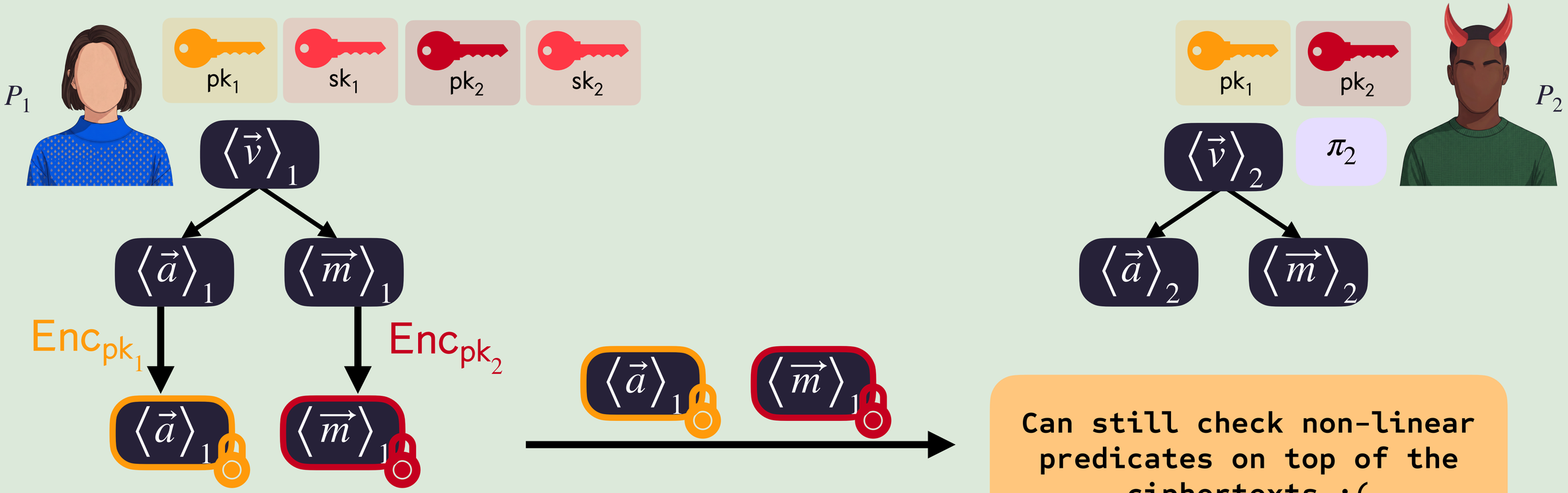


What about non-linear attacks?



Can still check non-linear predicates on top of the ciphertexts :(

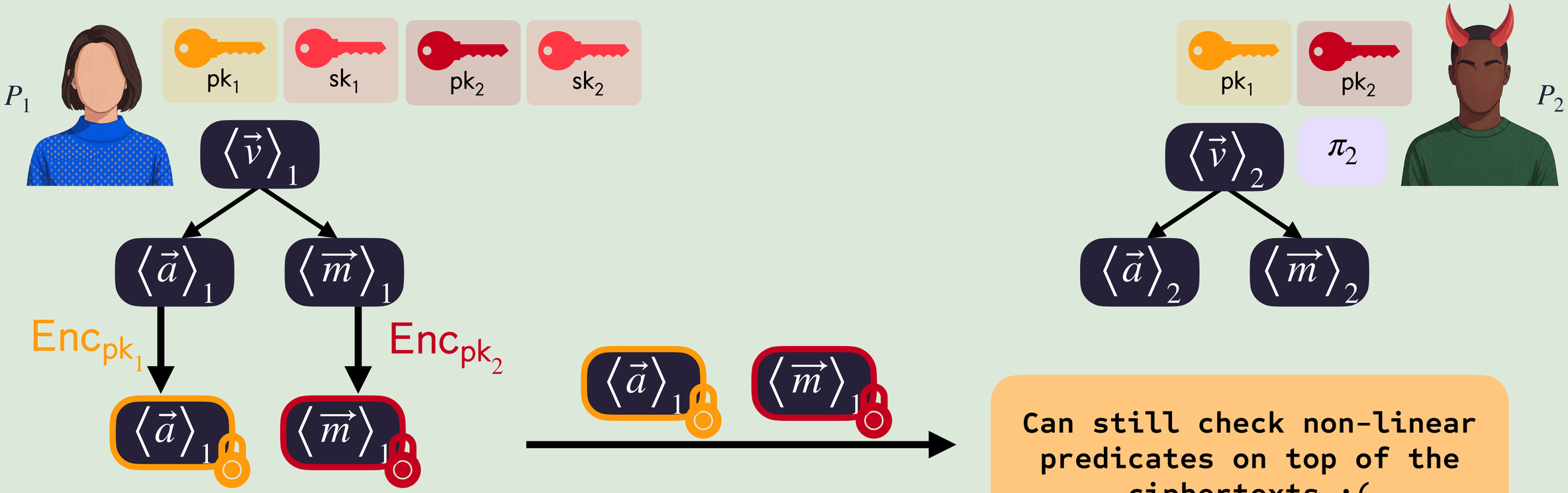
What about non-linear attacks?



Can still check non-linear predicates on top of the ciphertexts :(

To prevent non-linear predicates, we assume our LHE scheme is “**linear-only**”: a well-known (non-falsifiable) assumption in proof systems

What about non-linear attacks?

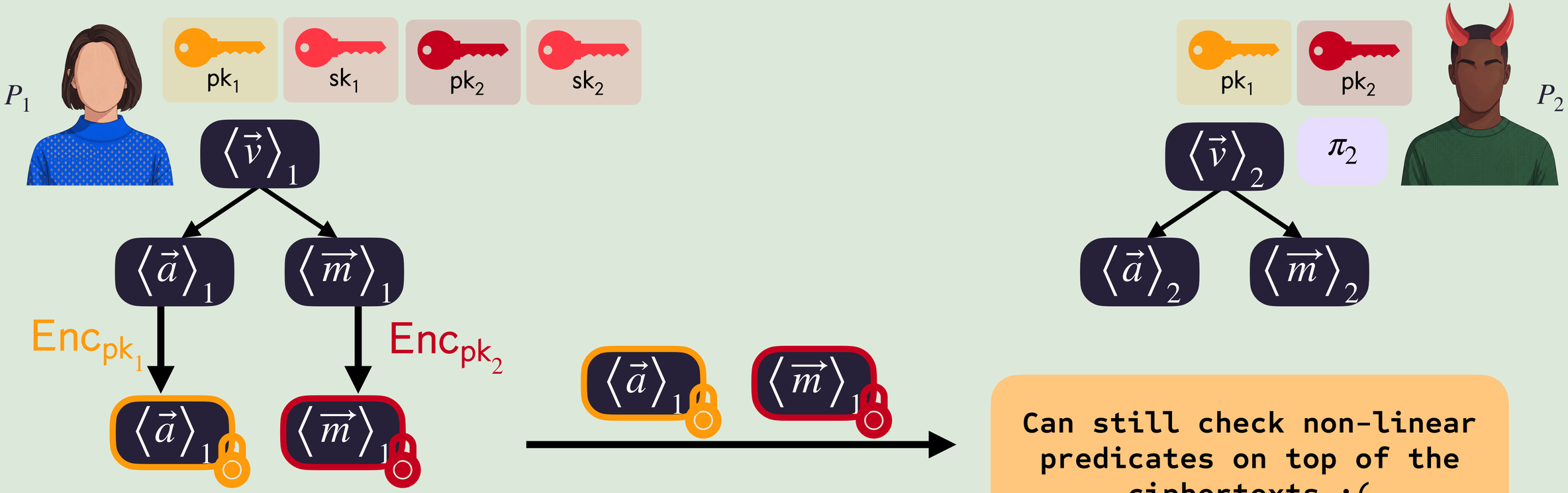


Can still check non-linear predicates on top of the ciphertexts :(

To prevent non-linear predicates, we assume our LHE scheme is “**linear-only**”: a well-known (non-falsifiable) assumption in proof systems

linear targeted malleability

What about non-linear attacks?

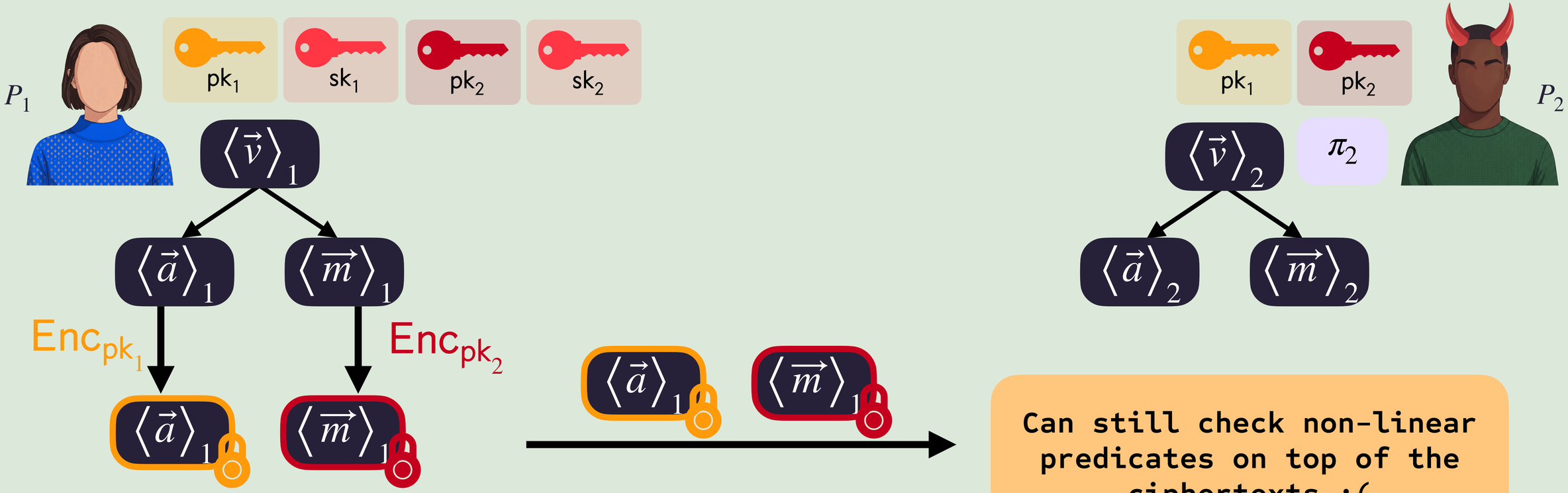


Can still check non-linear predicates on top of the ciphertexts :(

To prevent non-linear predicates, we assume our LHE scheme is “**linear-only**”: a well-known (non-falsifiable) assumption in proof systems

- linear targeted malleability
- new: “weakly predicable”

What about non-linear attacks?



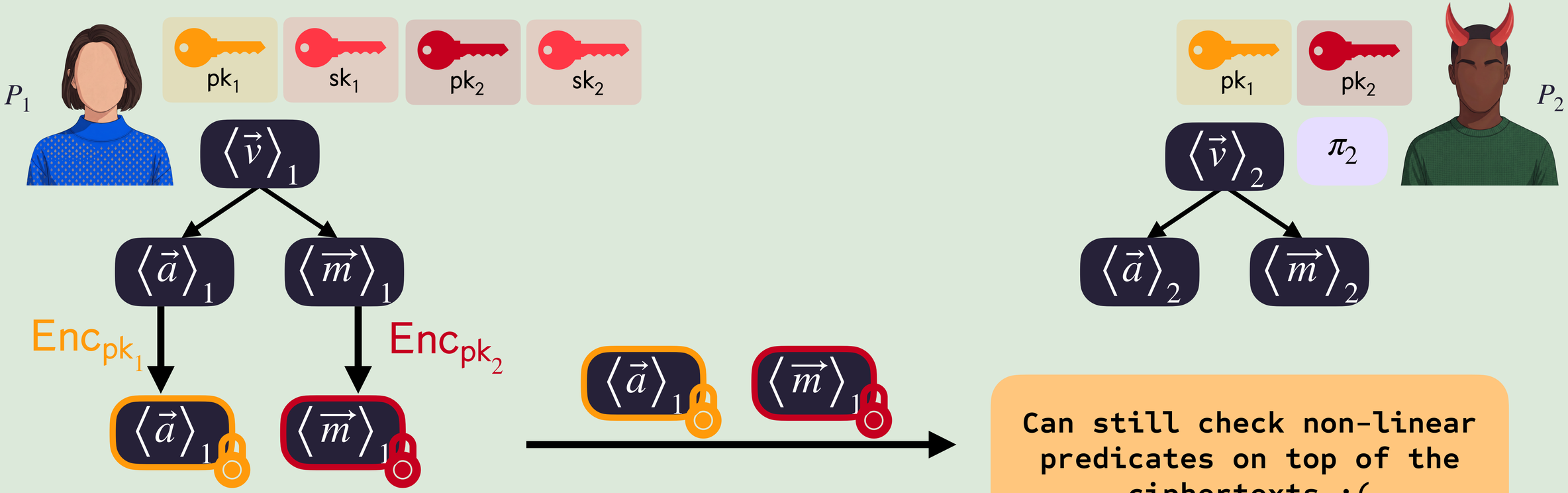
Can still check non-linear predicates on top of the ciphertexts :(

To prevent non-linear predicates, we assume our LHE scheme is “**linear-only**”: a well-known (non-falsifiable) assumption in proof systems

- linear targeted malleability
- new: “weakly predicable”

weaker assumptions

What about non-linear attacks?



Can still check non-linear predicates on top of the ciphertexts :(

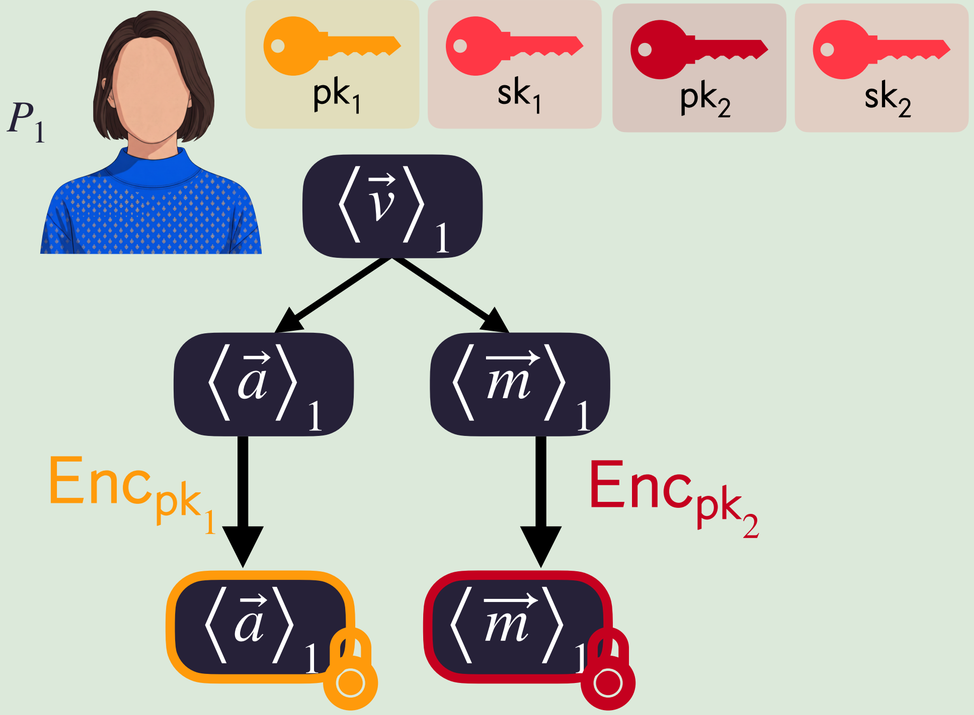
To prevent non-linear predicates, we assume our LHE scheme is “**linear-only**”: a well-known (non-falsifiable) assumption in proof systems

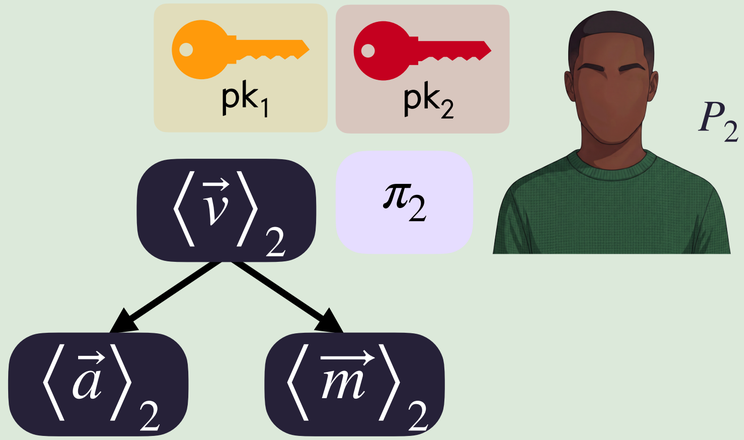
- linear targeted malleabil
- new: “weakly predicable

see paper for details!

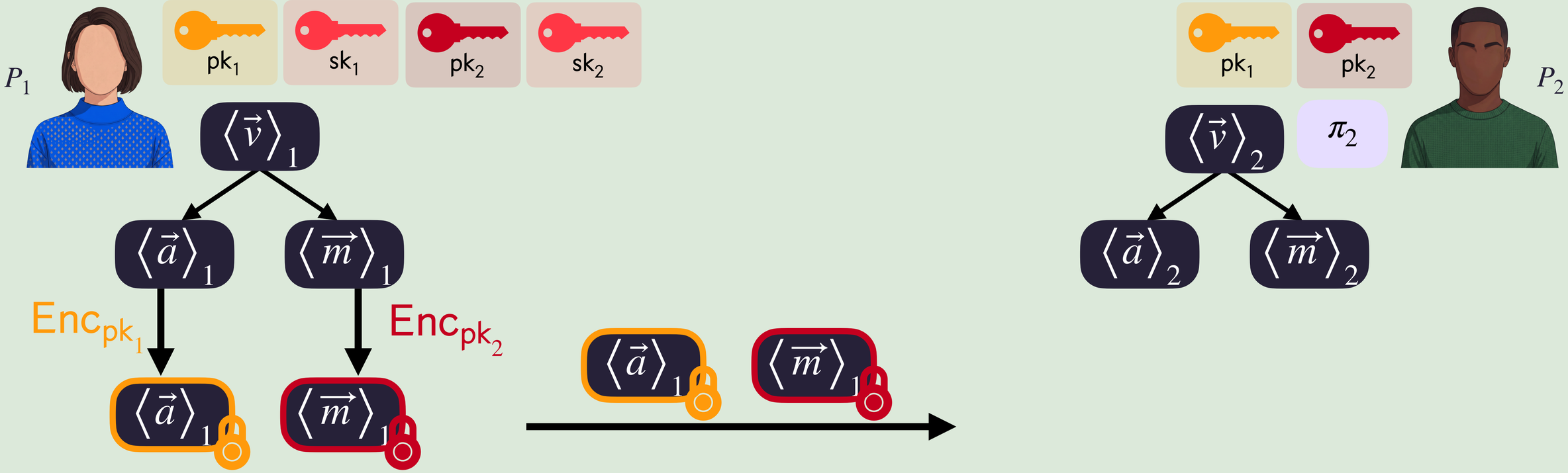
weaker assumptions

Protocol so far...

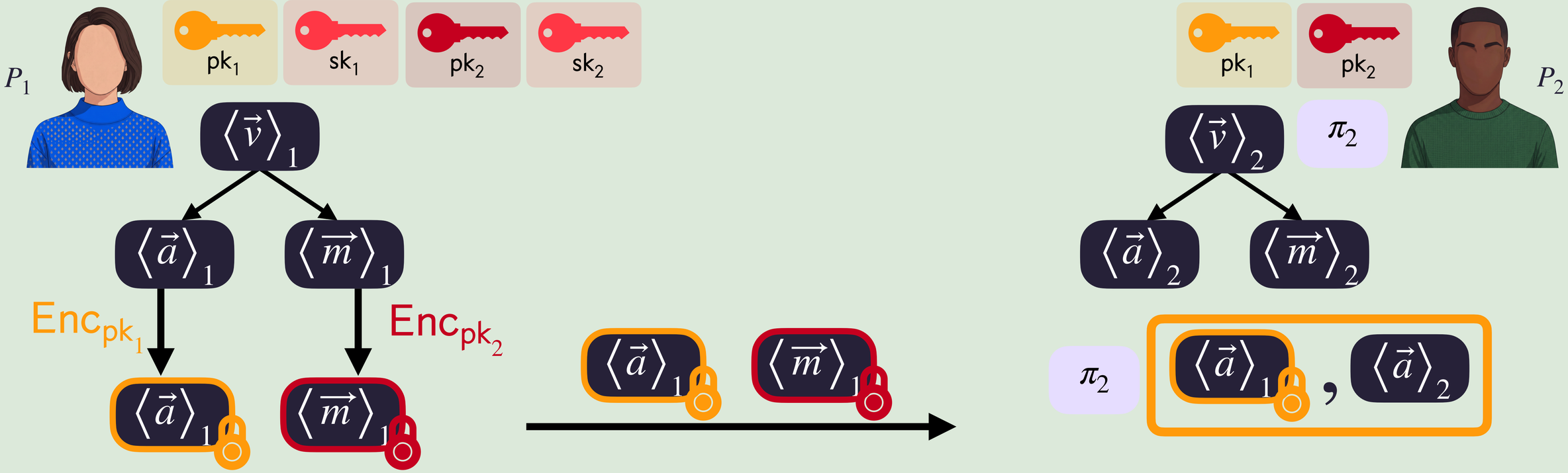




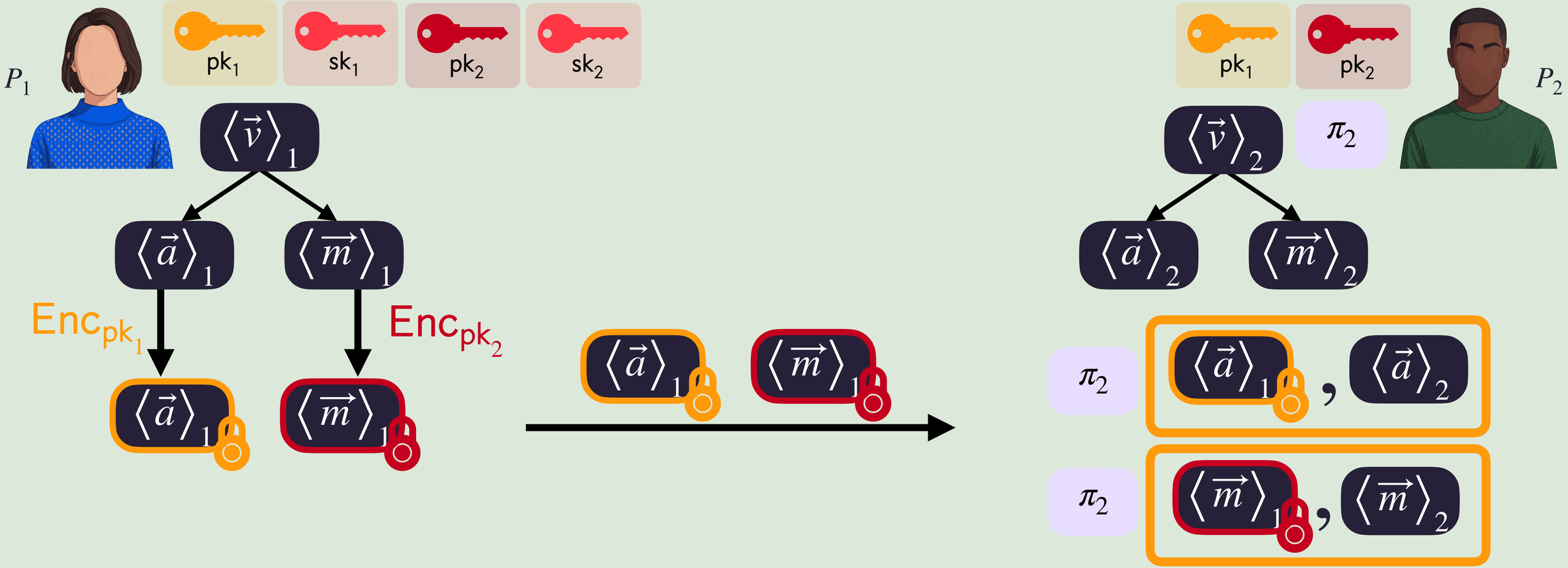
Protocol so far...



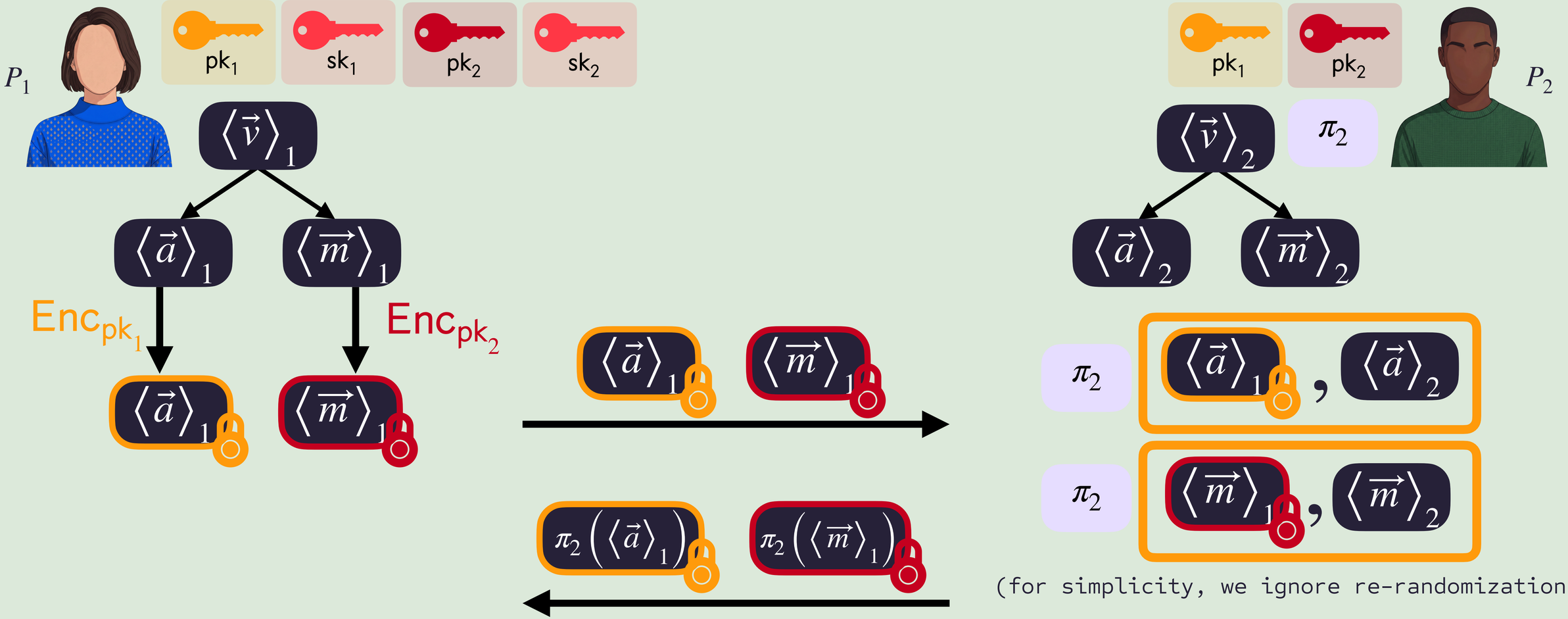
Protocol so far...



Protocol so far...

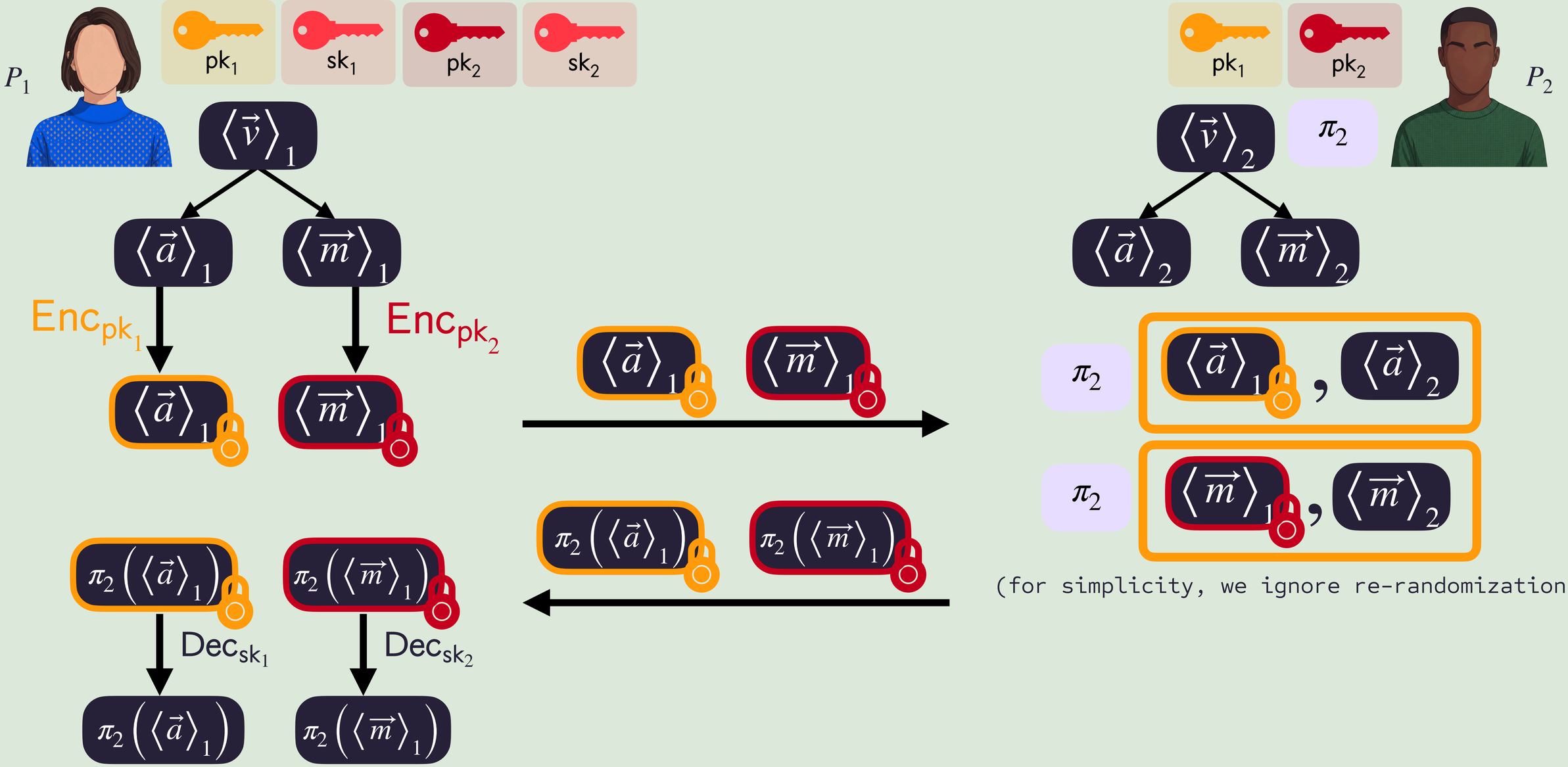


Protocol so far...



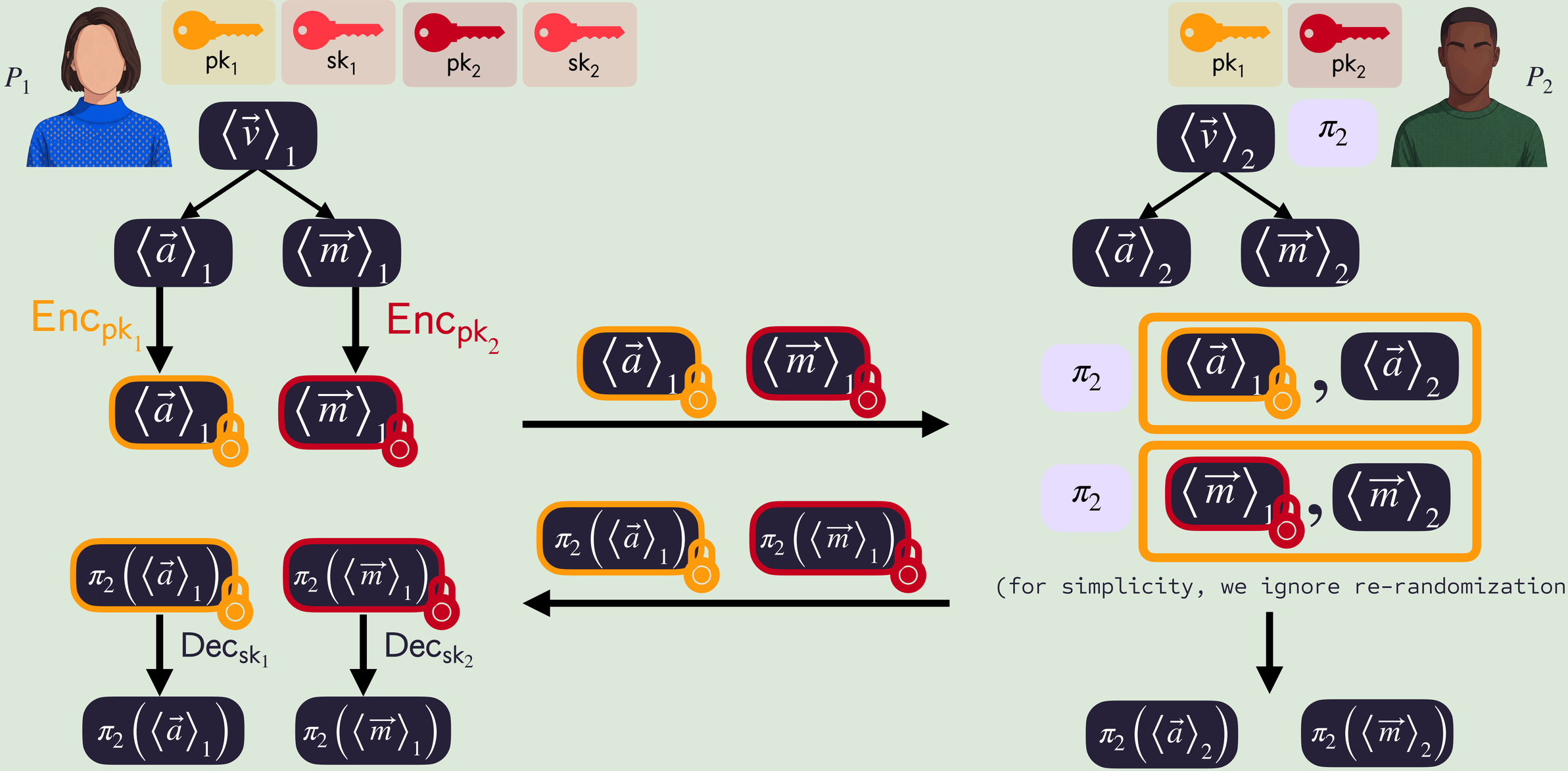
(for simplicity, we ignore re-randomization)

Protocol so far...

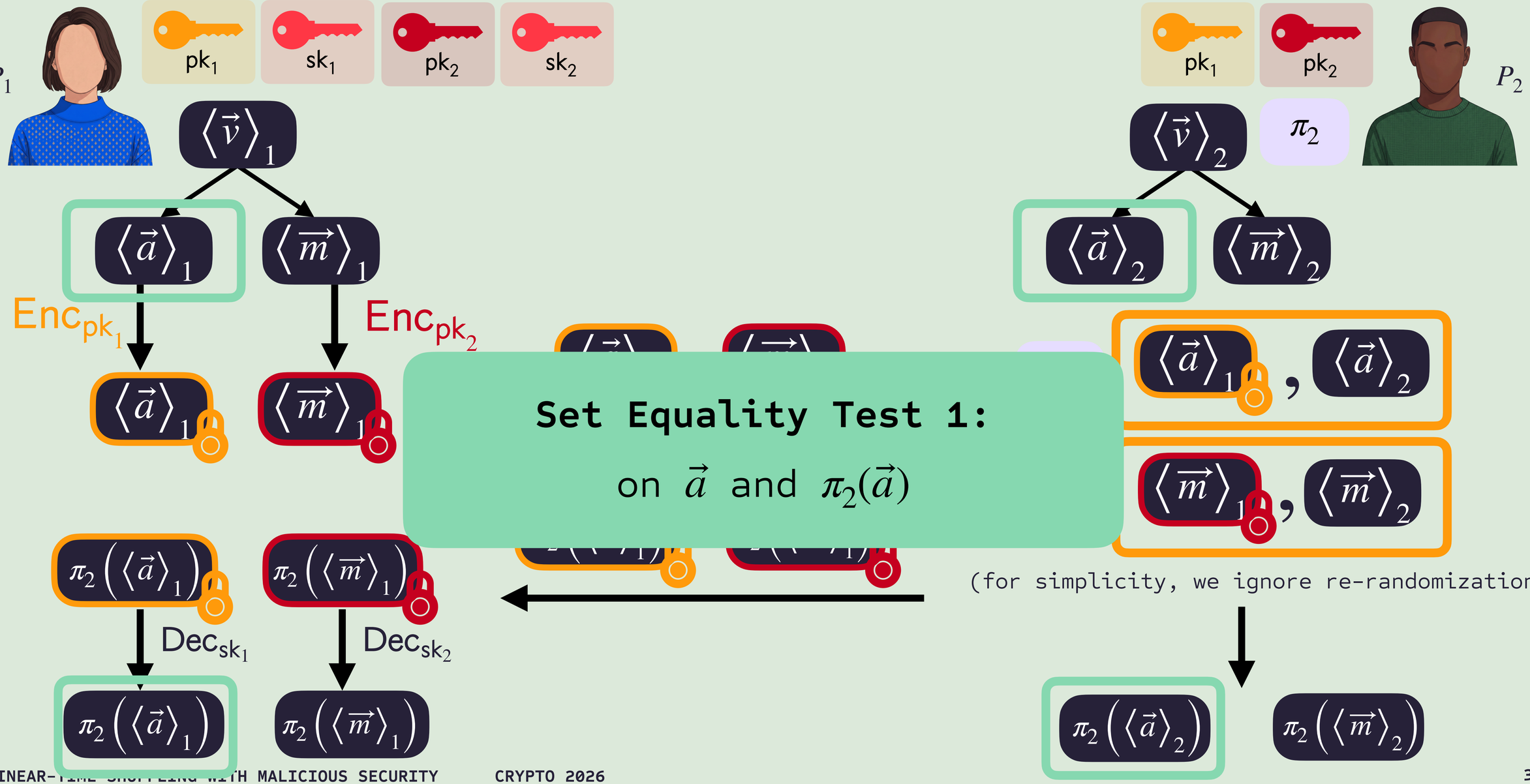


(for simplicity, we ignore re-randomization)

Protocol so far...



Protocol so far...

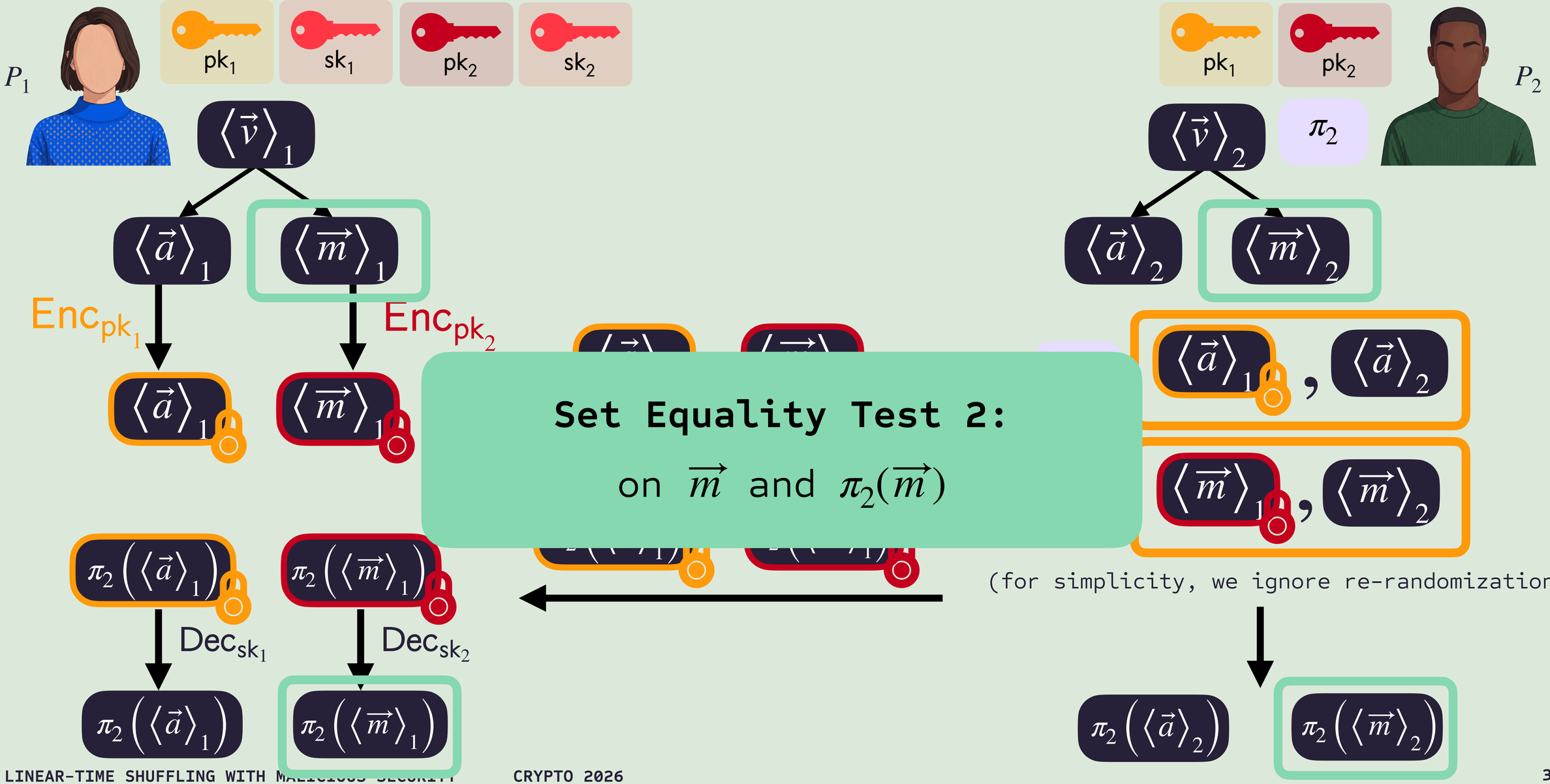


Set Equality Test 1:

on $\vec{a}$ and $\pi_2(\vec{a})$

(for simplicity, we ignore re-randomization)

Protocol so far...

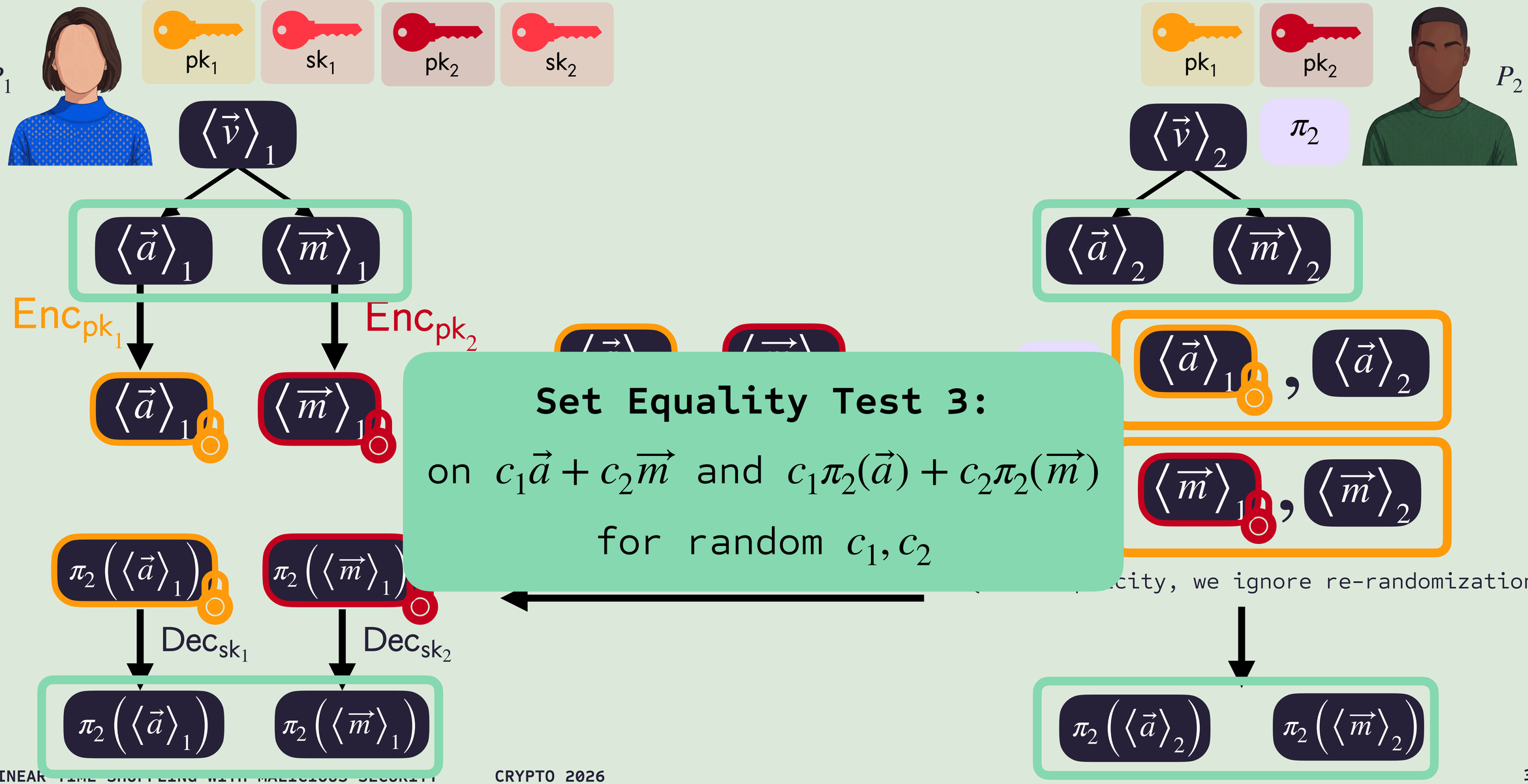


Set Equality Test 2:

on $\vec{m}$ and $\pi_2(\vec{m})$

(for simplicity, we ignore re-randomization)

Protocol so far...



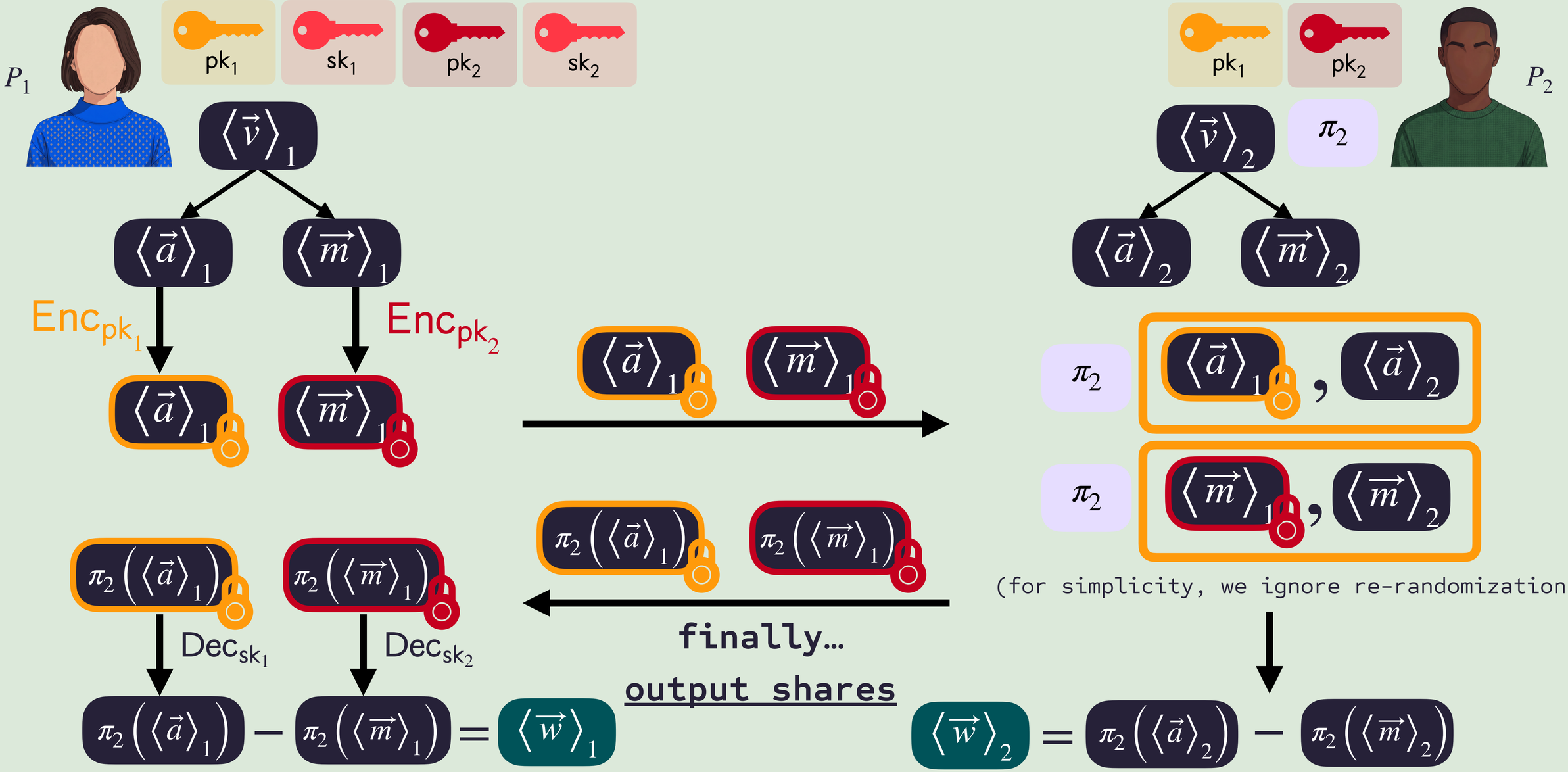
Set Equality Test 3:

on $c_1\vec{a} + c_2\vec{m}$ and $c_1\pi_2(\vec{a}) + c_2\pi_2(\vec{m})$

for random c_1, c_2

city, we ignore re-randomization)

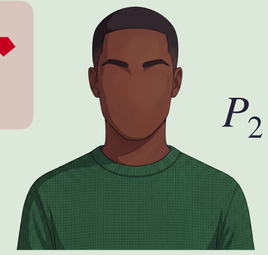
Protocol so far...



What about Malicious Encryptor P_1?

P_1

pk_1 sk_1 pk_2 sk_2



P_2

pk_1 pk_2

$\langle \vec{v} \rangle_1$ → $\langle \vec{a} \rangle_1$, $\langle \vec{m} \rangle_1$

$\mathsf{Enc}_{\mathsf{pk}_1}$ $\langle \vec{a} \rangle_1$

$\mathsf{Enc}_{\mathsf{pk}_2}$ $\langle \vec{m} \rangle_1$

$\langle \vec{a} \rangle_1$ $\langle \vec{m} \rangle_1$ →

$\langle \vec{v} \rangle_2$ → $\langle \vec{a} \rangle_2$, $\langle \vec{m} \rangle_2$

π_2

π_2 $\langle \vec{a} \rangle_1$, $\langle \vec{a} \rangle_2$

π_2 $\langle \vec{m} \rangle_1$, $\langle \vec{m} \rangle_2$

(for simplicity, we ignore re-randomization)

$\pi_2(\langle \vec{a} \rangle_1)$ $\pi_2(\langle \vec{m} \rangle_1)$ ←

finally…

output shares

$\mathsf{Dec}_{\mathsf{sk}_1}$ $\pi_2(\langle \vec{a} \rangle_1)$

$\mathsf{Dec}_{\mathsf{sk}_2}$ $\pi_2(\langle \vec{m} \rangle_1)$

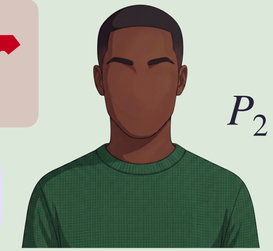
$\pi_2(\langle \vec{a} \rangle_1) - \pi_2(\langle \vec{m} \rangle_1) = \langle \vec{w} \rangle_1$

$\langle \vec{w} \rangle_2 = \pi_2(\langle \vec{a} \rangle_2) - \pi_2(\langle \vec{m} \rangle_2)$

What about Malicious Encryptor P_1?

P_1

pk_1 sk_1 pk_2 sk_2



P_2

pk_1 pk_2

$\langle \vec{v} \rangle_1$

$\langle \vec{v} \rangle_2$ π_2

$\langle \vec{a} \rangle_1$ $\langle \vec{m} \rangle_1$

$\mathsf{Enc}_{\mathsf{pk}_1}$ $\mathsf{Enc}_{\mathsf{pk}_2}$

$\langle \vec{a} \rangle_1$ $\langle \vec{m} \rangle_1$

$\langle \vec{a} \rangle_2$ $\langle \vec{m} \rangle_2$

P_1 can encrypt to incorrect shares

$\langle \vec{a} \rangle_1$ $\langle \vec{m} \rangle_1$

π_2 $\langle \vec{a} \rangle_1$, $\langle \vec{a} \rangle_2$

π_2 $\langle \vec{m} \rangle_1$, $\langle \vec{m} \rangle_2$

$\pi_2(\langle \vec{a} \rangle_1)$ $\pi_2(\langle \vec{m} \rangle_1)$

(for simplicity, we ignore re-randomization)

$\pi_2(\langle \vec{a} \rangle_1)$ $\pi_2(\langle \vec{m} \rangle_1)$

$\mathsf{Dec}_{\mathsf{sk}_1}$ $\mathsf{Dec}_{\mathsf{sk}_2}$

finally…

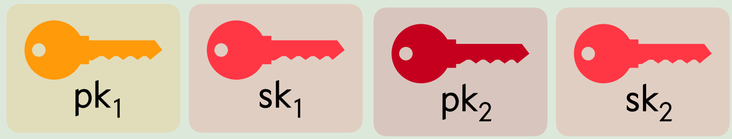
output shares

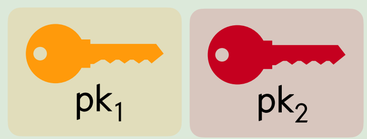
$\pi_2(\langle \vec{a} \rangle_1) - \pi_2(\langle \vec{m} \rangle_1) = \langle \vec{w} \rangle_1$

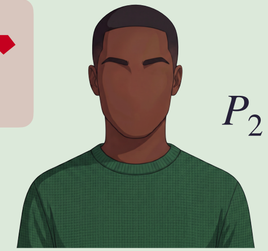
$\langle \vec{w} \rangle_2 = \pi_2(\langle \vec{a} \rangle_2) - \pi_2(\langle \vec{m} \rangle_2)$

What about Malicious Encryptor P_1?

P_1







P_2

$\langle \vec{v} \rangle_1$

$\langle \vec{a} \rangle_1$ $\langle \vec{m} \rangle_1$

$\mathsf{Enc}_{\mathsf{pk}_1}$ $\mathsf{Enc}_{\mathsf{pk}_2}$

$\langle \vec{a} \rangle_1$ $\langle \vec{m} \rangle_1$

$\langle \vec{v} \rangle_2$ π_2

$\langle \vec{a} \rangle_2$ $\langle \vec{m} \rangle_2$

P_1 can encrypt to incorrect shares

concrete attack: P_1 can learn information about the permutation with non-negligible probability

π_2 $\langle \vec{a} \rangle_1, \langle \vec{a} \rangle_2$

π_2 $\langle \vec{m} \rangle_1, \langle \vec{m} \rangle_2$

$\pi_2(\langle \vec{a} \rangle_1)$ $\pi_2(\langle \vec{m} \rangle_1)$

$\mathsf{Dec}_{\mathsf{sk}_1}$ $\mathsf{Dec}_{\mathsf{sk}_2}$

$\pi_2(\langle \vec{a} \rangle_1) - \pi_2(\langle \vec{m} \rangle_1) = \langle \vec{w} \rangle_1$

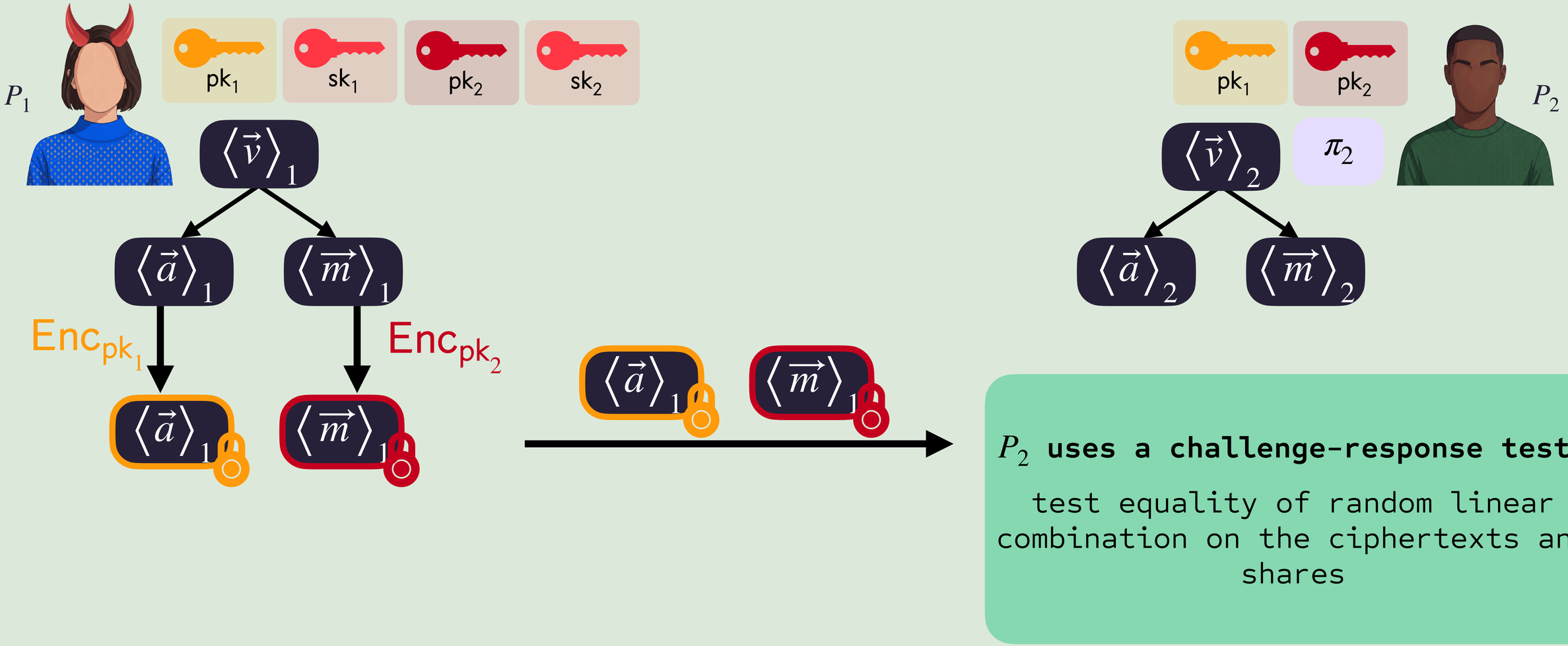
finally…

output shares

(for simplicity, we ignore re-randomization)

$\langle \vec{w} \rangle_2 = \pi_2(\langle \vec{a} \rangle_2) - \pi_2(\langle \vec{m} \rangle_2)$

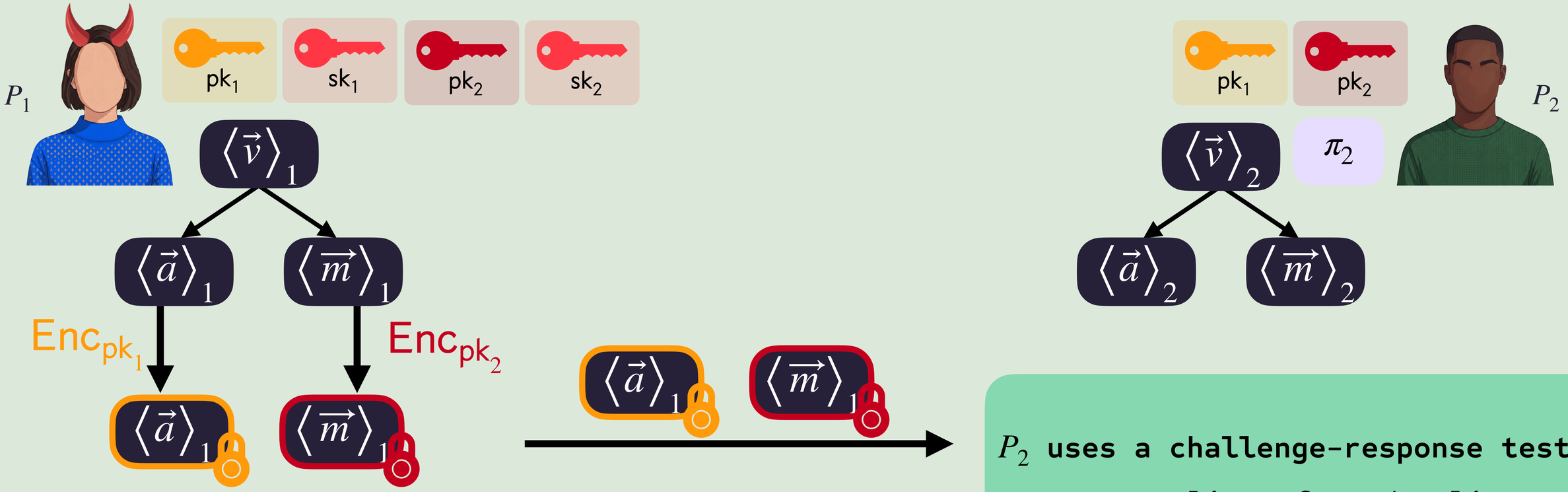
Idea 3: Correlation Checks



P_2 uses a challenge-response test:

test equality of random linear combination on the ciphertexts and shares

Idea 3: Correlation Checks



P_2 uses a challenge-response test:

test equality of random linear combination on the ciphertexts and shares

see paper for details

Recap

Recap

Folklore protocol using Linearly Homomorphic Encryption (LHE) and authenticated secret sharing

Recap

Folklore protocol using Linearly Homomorphic Encryption (LHE) and authenticated secret sharing

(1) Set equality test

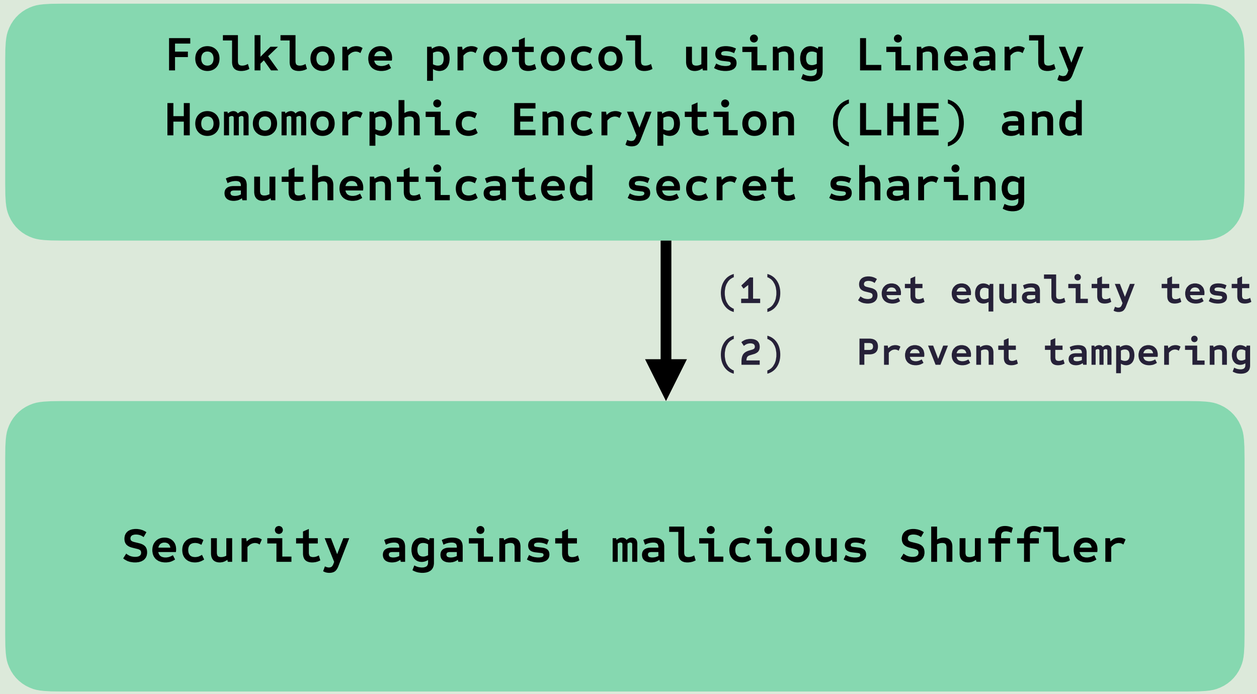
(2) Prevent tampering of shares

(a) using authenticated secret sharing,

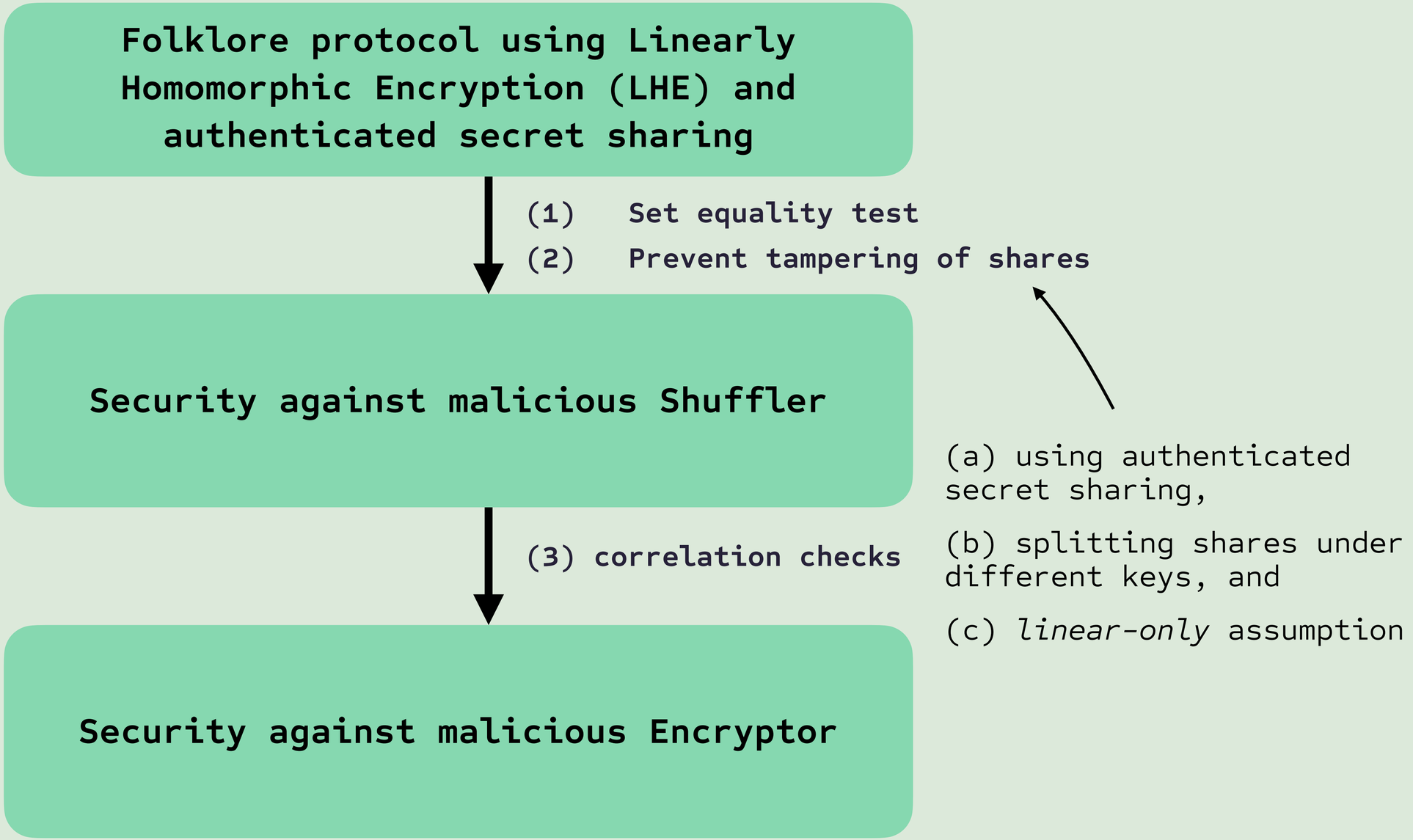
(b) splitting shares under different keys, and

(c) *linear-only* assumption

Recap


Folklore protocol using Linearly Homomorphic Encryption (LHE) and authenticated secret sharing
(1) Set equality test
(2) Prevent tampering of shares
Security against malicious Shuffler
(a) using authenticated secret sharing,
(b) splitting shares under different keys, and
(c) linear-only assumption

Recap


Folklore protocol using Linearly Homomorphic Encryption (LHE) and authenticated secret sharing
(1) Set equality test
(2) Prevent tampering of shares
Security against malicious Shuffler
(3) correlation checks
Security against malicious Encryptor
(a) using authenticated secret sharing,
(b) splitting shares under different keys, and
(c) linear-only assumption

Additional Contributions

Additional Contributions

3-party IT protocol

Present a maliciously secure 3-party information-theoretic protocol also using authenticated secret sharing

Additional Contributions

3-party IT protocol

Present a maliciously secure 3-party information-theoretic protocol also using authenticated secret sharing

Communication-efficient variant

Also present a more efficient protocol in which shares can be delegated in unauthenticated way with additional checks

Additional Contributions

3-party IT protocol

Present a maliciously secure 3-party information-theoretic protocol also using authenticated secret sharing

Communication-efficient variant

Also present a more efficient protocol in which shares can be delegated in unauthenticated way with additional checks

Constant-round set equality test

Construct a constant-round set equality test using logarithmic derivative of the standard polynomial root test

(pending paper update)

Thank you!

Folklore protocol using Linearly Homomorphic Encryption (LHE) and authenticated secret sharing

(1) Set equality test

(2) Prevent tampering of shares

Security against malicious Shuffler

(3) correlation checks

Security against malicious Encryptor

(a) using authenticated secret sharing,

(b) splitting shares under different keys, and

(c) *linear-only* assumption

Linear Secret-shared Shuffle with Malicious Security (ePrint 2025/2137)