

# Weak Fiat-Shamir Attacks on Modern Proof Systems

Quang Dao

[qvd@andrew.cmu.edu](mailto:qvd@andrew.cmu.edu)

Jim Miller

[james.miller@trailofbits.com](mailto:james.miller@trailofbits.com)

Opal Wright

[opal.wright@trailofbits.com](mailto:opal.wright@trailofbits.com)

Paul Grubbs

[paulgrub@umich.edu](mailto:paulgrub@umich.edu)

Carnegie  
Mellon  
University

TRAIL  
OF  
BITS



*Workshop on Attacks in Cryptography, 2023*

# **Proof Systems and Blockchain Applications**

# Proof Systems and Blockchain Applications

Zcash is cash for the new age.

## Monero Means Money

Private, decentralized cryptocurrency that keeps your finances confidential and secure.

 polygon 2.0

The Value Layer of the Internet



Filecoin is a decentralized storage network designed to store humanity's most important information.

Regulated And Decentralized Finance

## STARK Proof Pioneers

Bringing scalability, security, and privacy to a blockchain near you

Espresso helps rollups:

Ethereum,

AN INTENT-CENTRIC PROTOCOL FOR COMPOSABLE PRIVACY, DECENTRALIZED COUNTERPARTY DISCOVERY, SOLVING, AND ATOMIC MULTI-CHAIN SETTLEMENT

Ethereum's First zkRollup Layer 2

encrypted



ztec is a first-of-its-kind hybrid zkRollup supporting both public and private smart contract execution.

The Native zkEVM Scalability Solution for Ethereum

Mina is building the security layer for web knowledge proofs.

The ZK Coprocessor for Ethereum

Scroll is a zkEVM-based zkRollup on Ethereum that enables native compatibility for existing Ethereum applications and

# Proof Systems and Blockchain Applications

Zcash is cash for the new age.

Monero Means Money

Private, decentralized cryptocurrency that keeps your finances confidential and secure.

 polygon 2.0

The Value Layer of the Internet



Filecoin is a decentralized storage network designed to store humanity's most important information.

Regulated And Decentralized

How do they work?

STARK Proof Pioneers

Bringing scalability, security, and privacy to a blockchain near you

AN INTENT-CENTRIC PROTOCOL FOR COMPOSABLE PRIVACY, DECENTRALIZED COUNTERPARTY DISCOVERY, SOLVING, AND ATOMIC MULTI-CHAIN SETTLEMENT



erc20 is a first-of-its-kind hybrid zkRollup supporting both public and private smart contract execution.

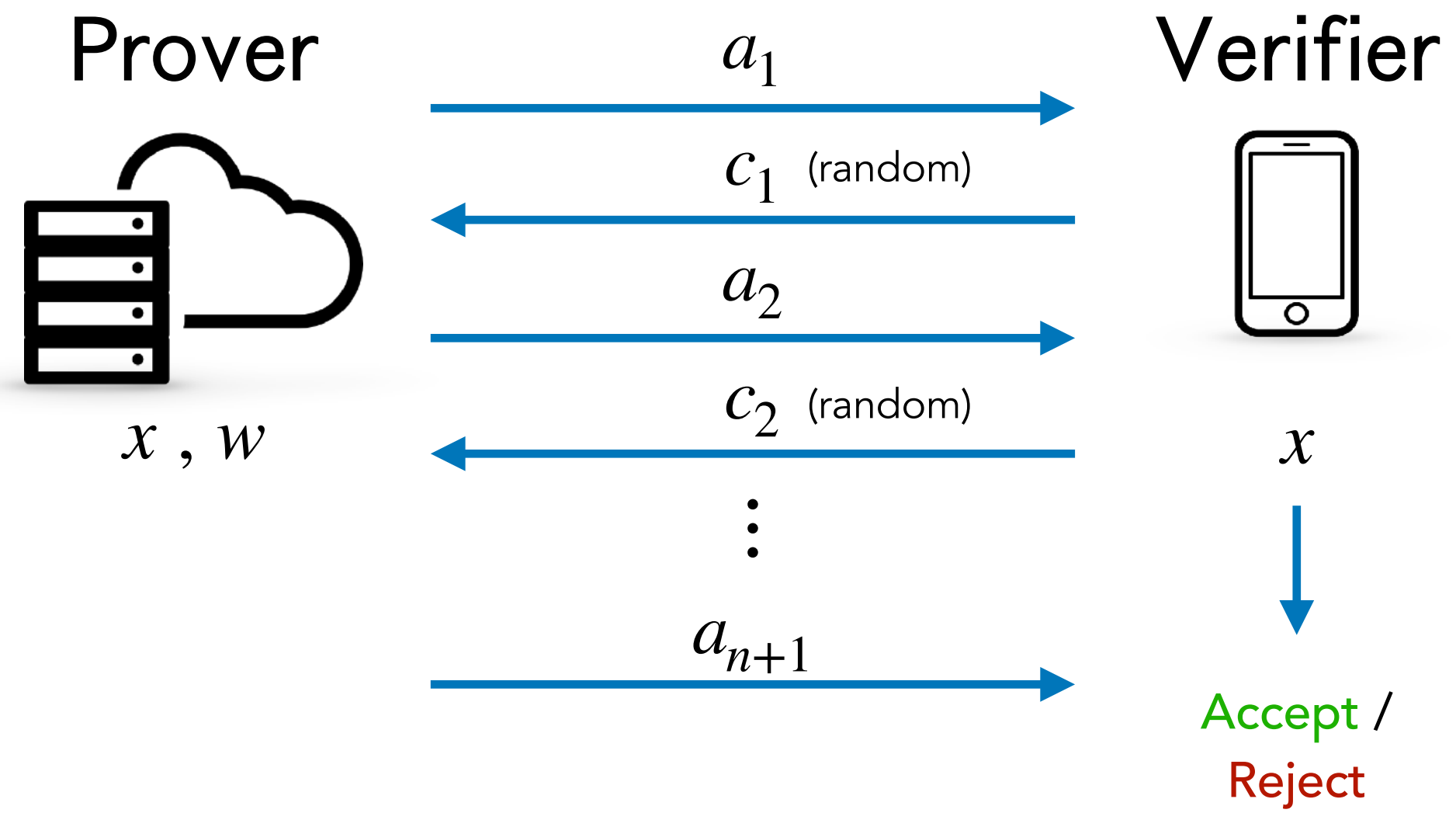
The Native zkEVM Scalability Solution for Ethereum  
Mina is building the privacy security layer for web3 knowledge proofs.

Scroll is a zkEVM-based zkRollup on Ethereum that enables native compatibility for existing Ethereum applications and

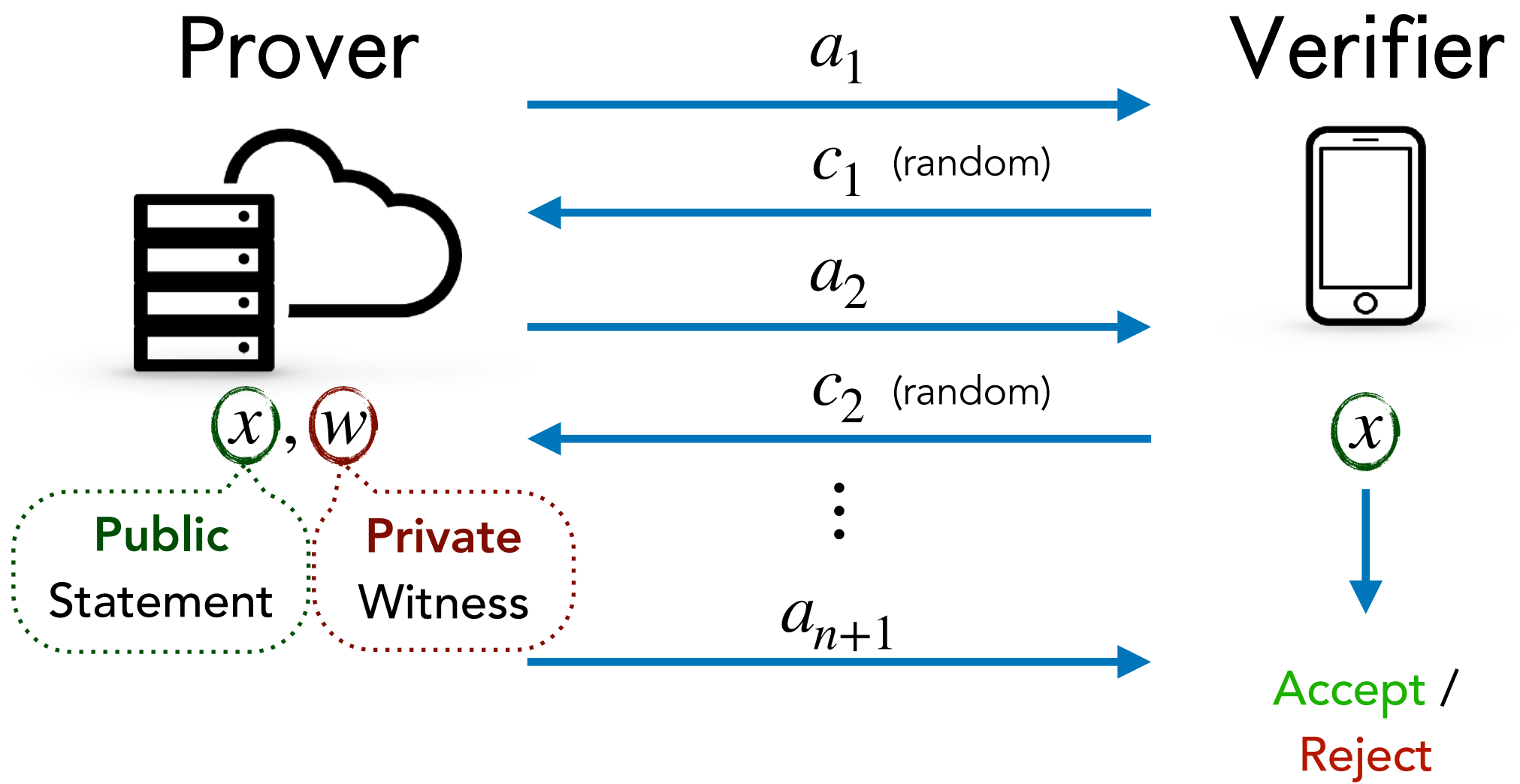
The ZK Coprocessor for Ethereum

# Proof Systems from Fiat-Shamir

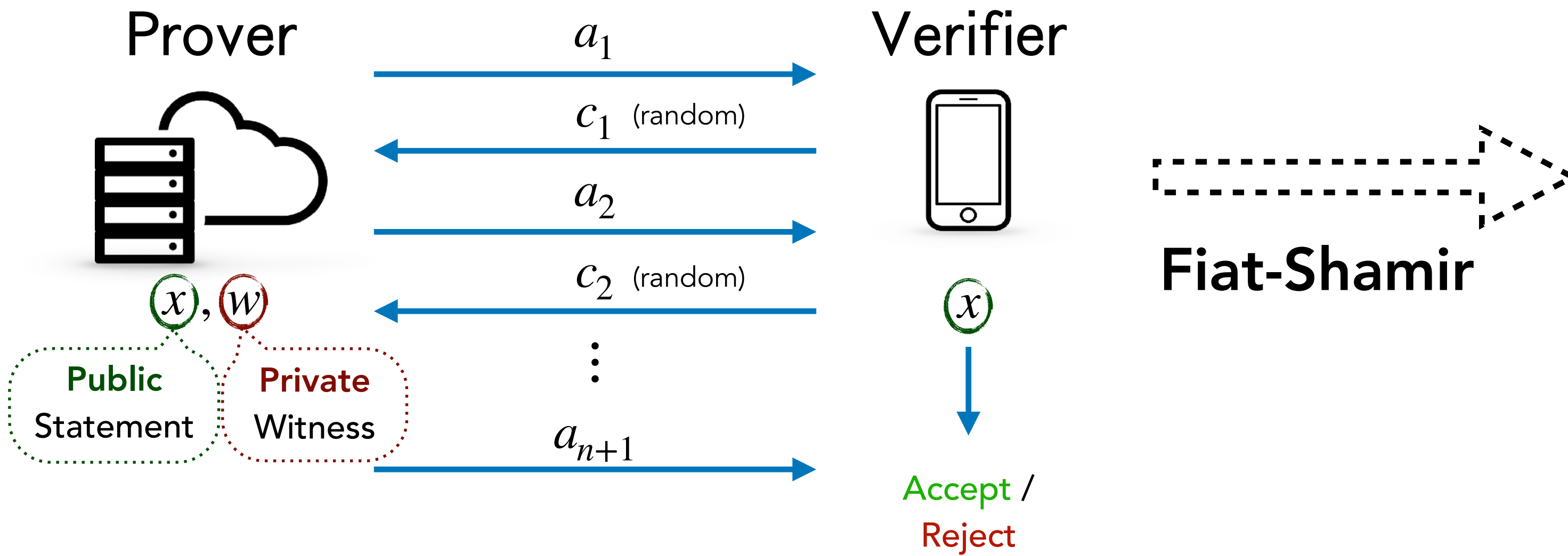
# Proof Systems from Fiat-Shamir



# Proof Systems from Fiat-Shamir

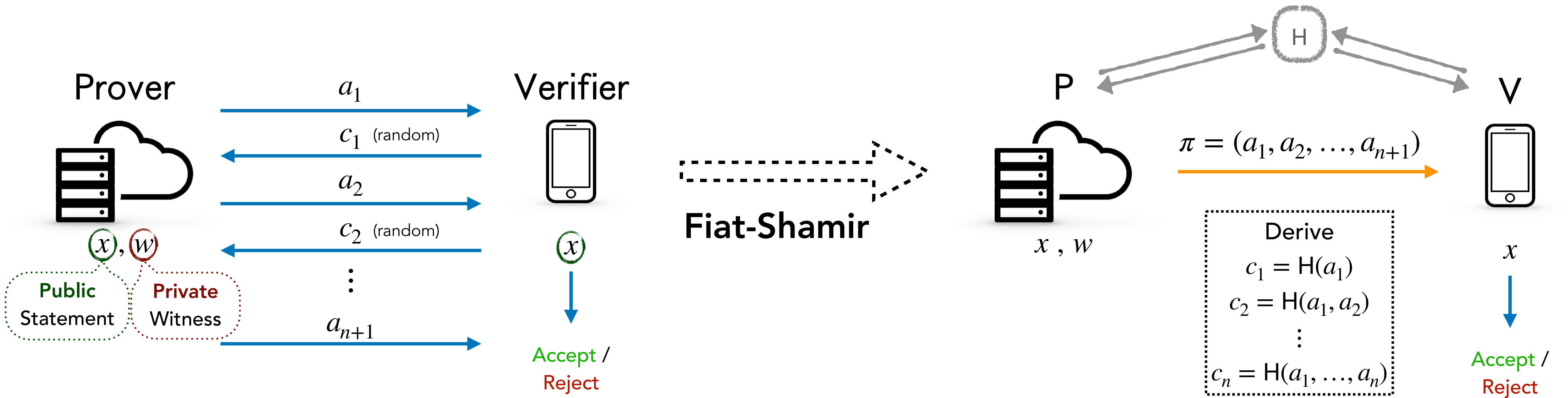


# Proof Systems from Fiat-Shamir

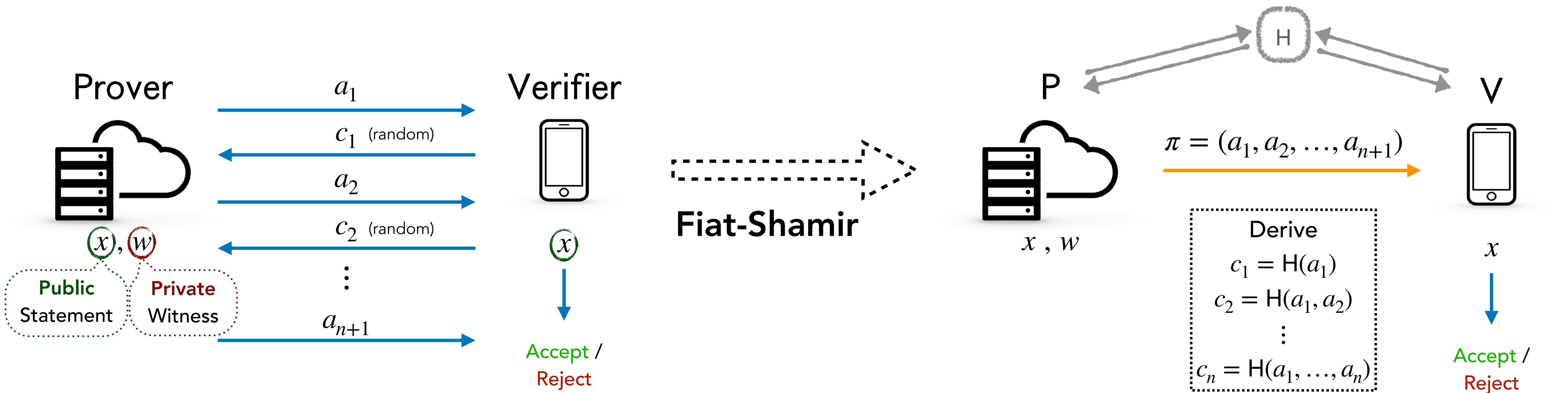




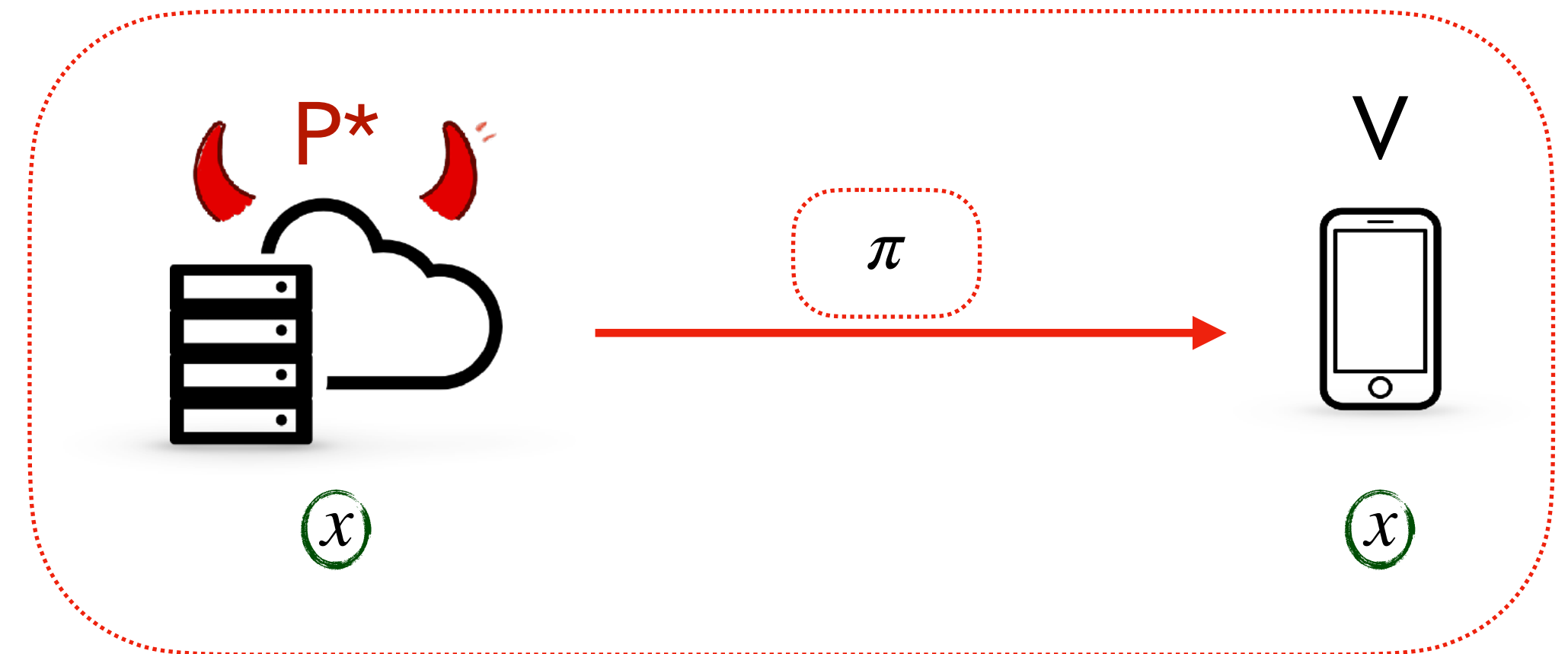
# Proof Systems from Fiat-Shamir



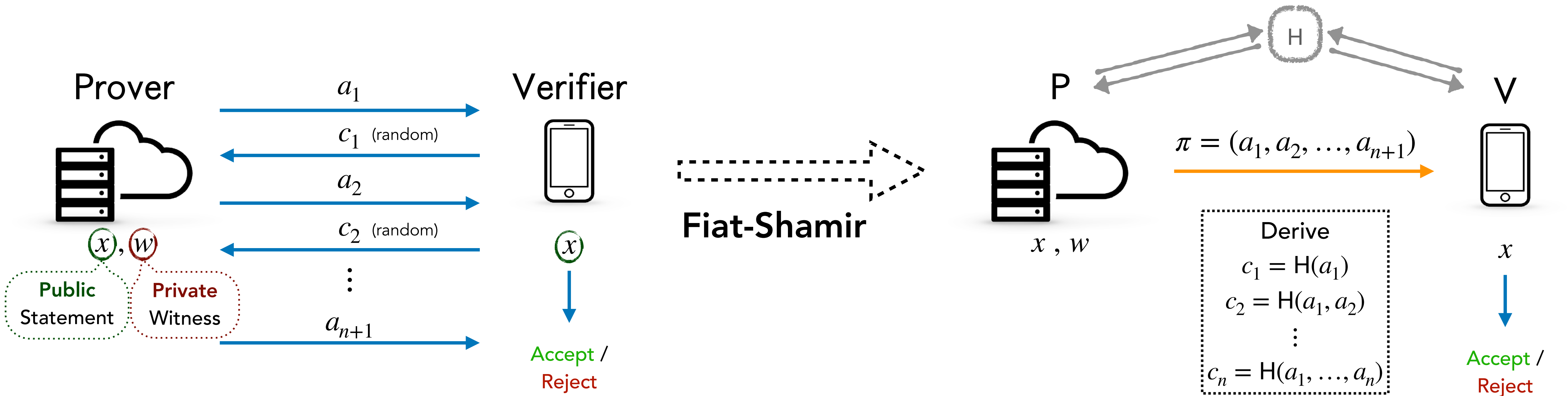
# Proof Systems from Fiat-Shamir



## Security:

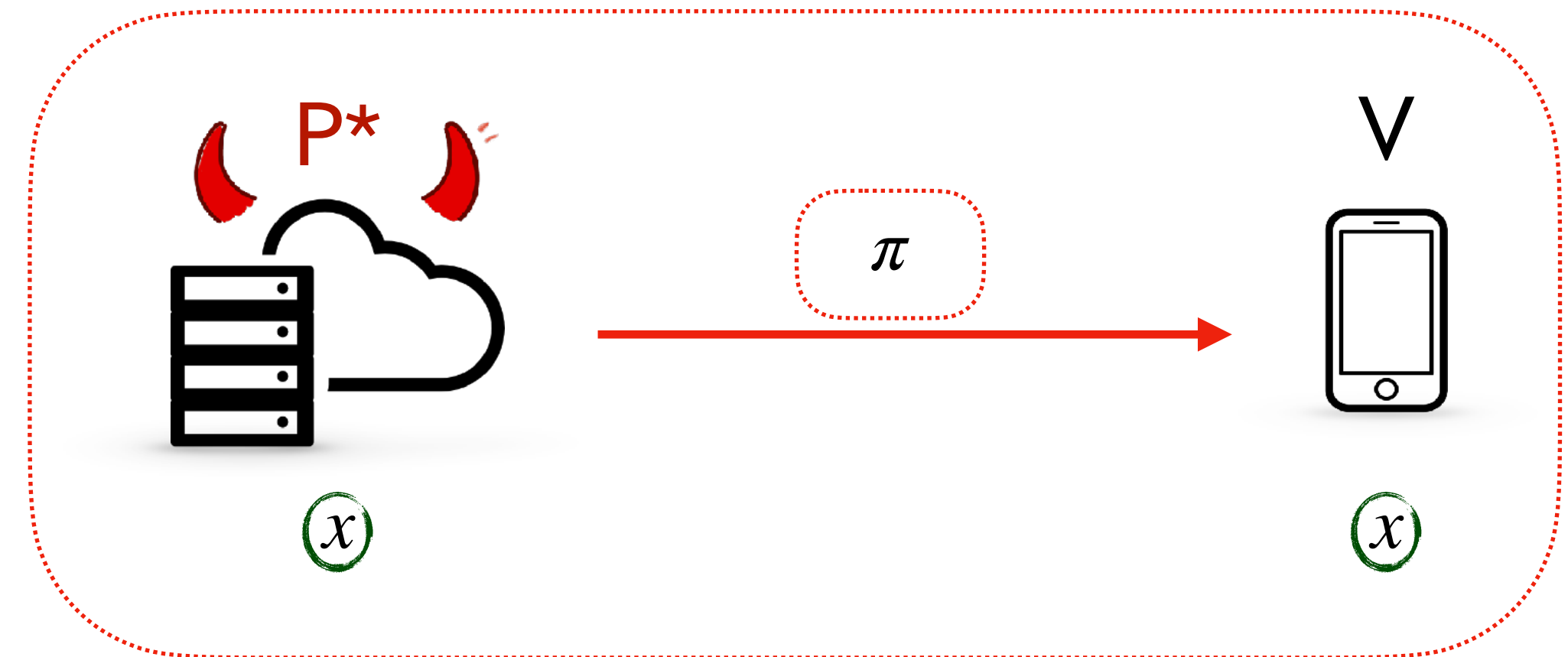


# Proof Systems from Fiat-Shamir

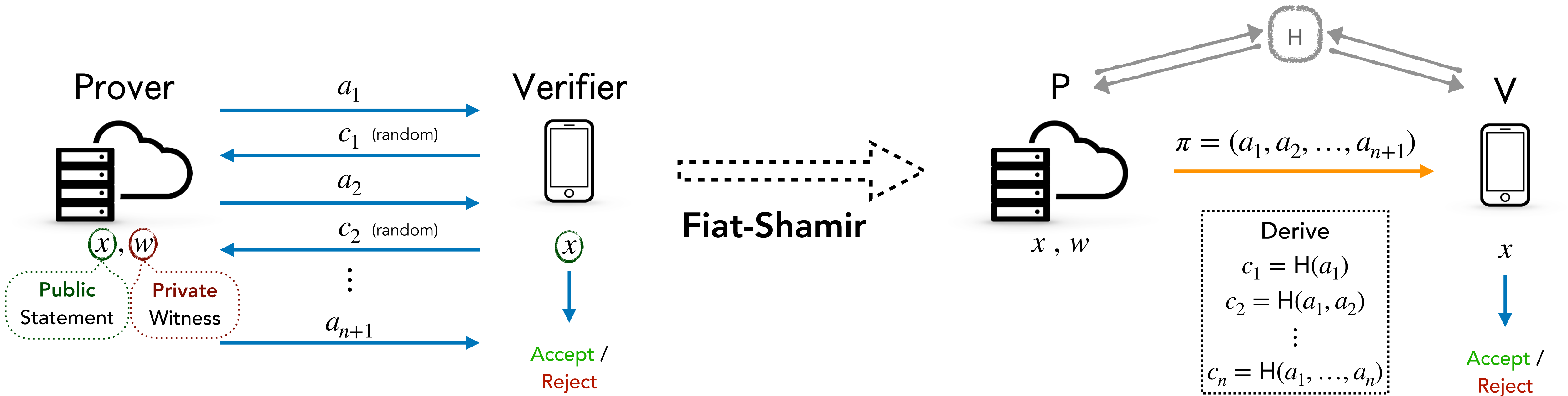


## Security:

Soundness: If  $x$  has no  $w$ , then  $V$  rejects.



# Proof Systems from Fiat-Shamir

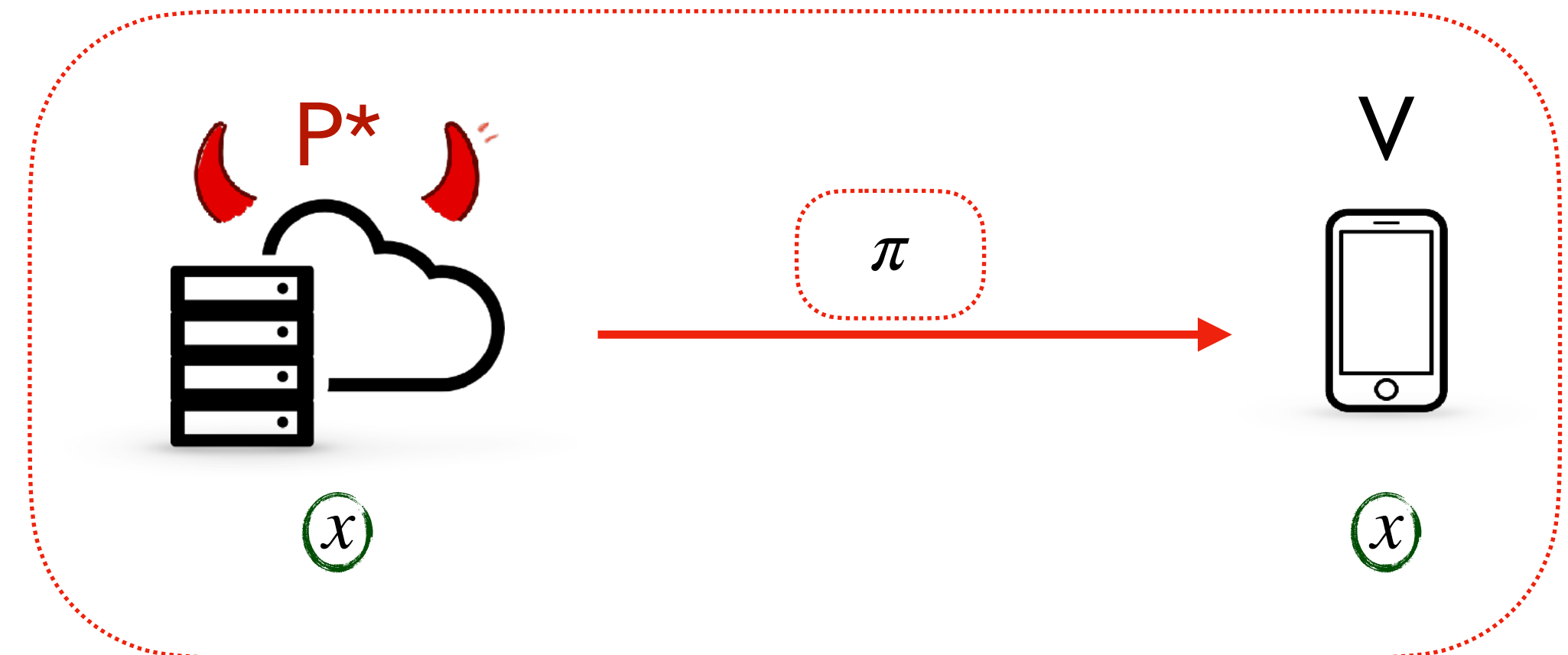


## Security:

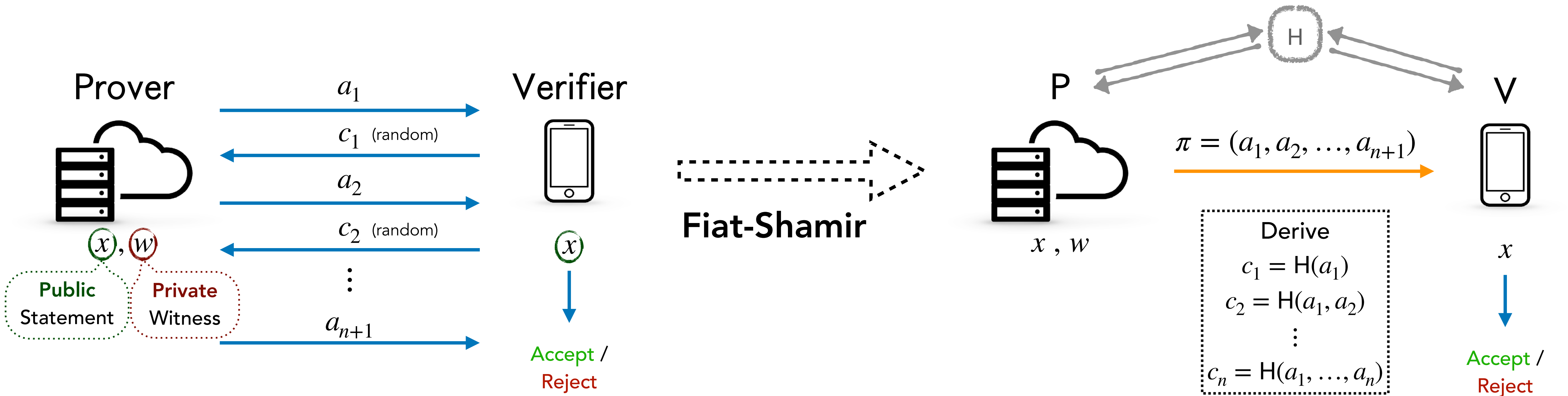
Soundness: If  $x$  has no  $w$ , then  $V$  rejects.

Knowledge Soundness: If  $V$  accepts, then

$P^*$  must "know"  $w$ .



# Proof Systems from Fiat-Shamir

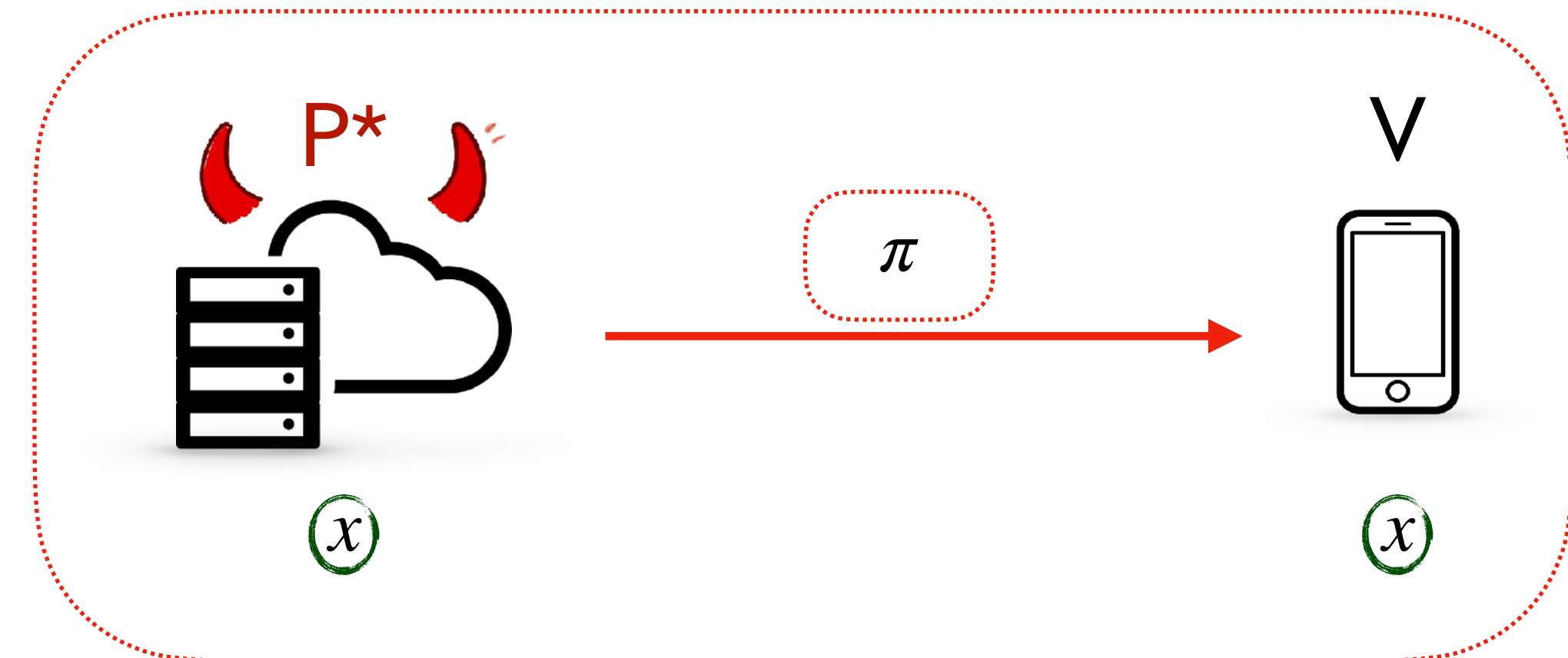


## Security:

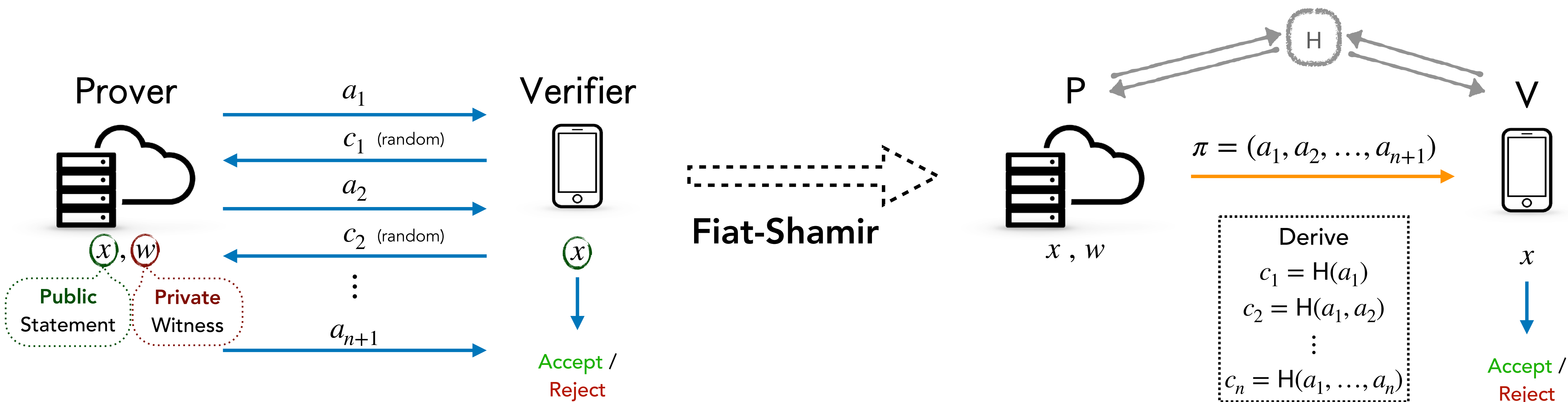
Soundness: If  $x$  has no  $w$ , then  $V$  rejects. ✓

Knowledge Soundness: If  $V$  accepts, then ✓

$P^*$  must "know"  $w$ .



# Proof Systems from Fiat-Shamir

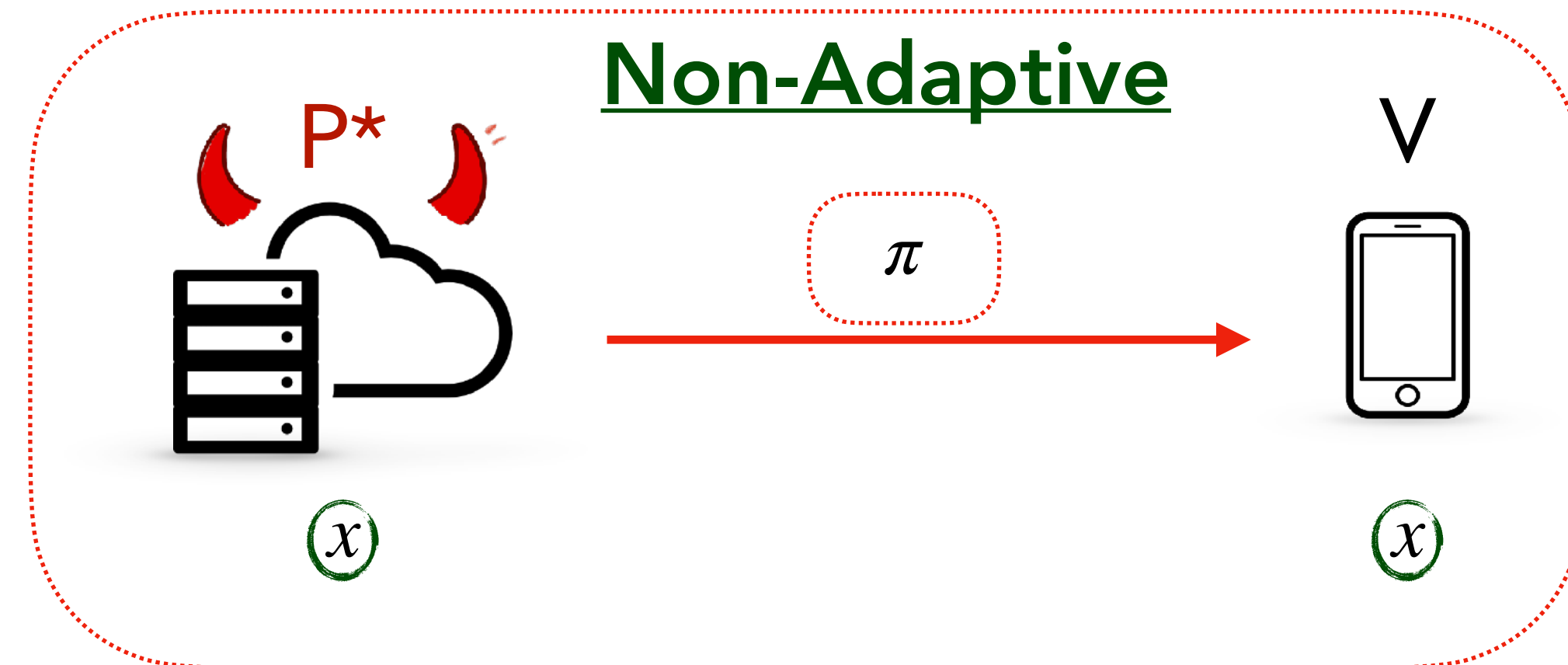


## Security:

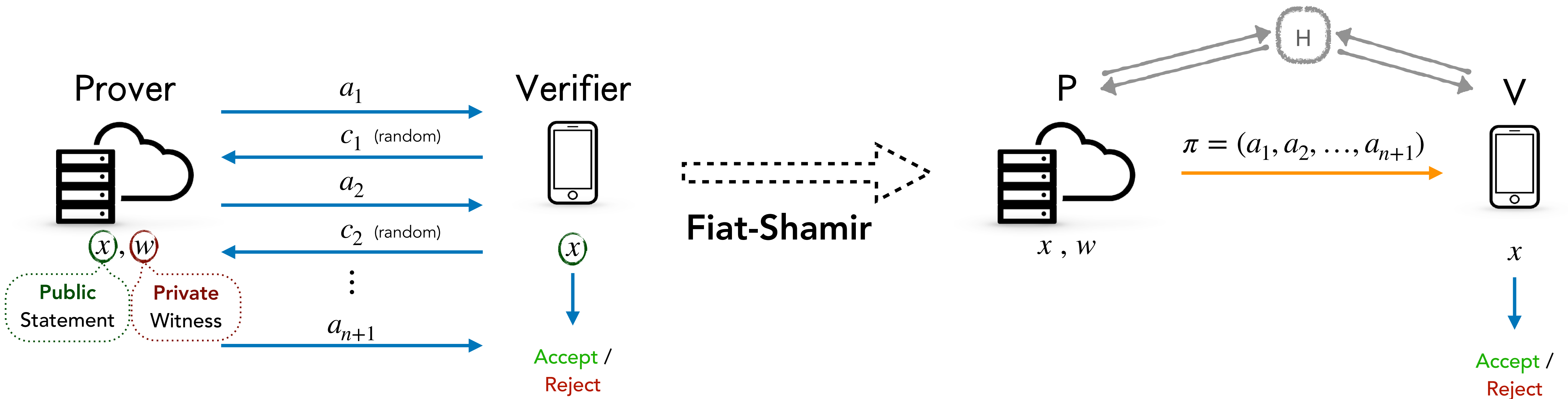
Soundness: If  $x$  has no  $w$ , then  $V$  rejects. ✓

Knowledge Soundness: If  $V$  accepts, then ✓

$P^*$  must "know"  $w$ .



# Strong Fiat-Shamir for Adaptive Security



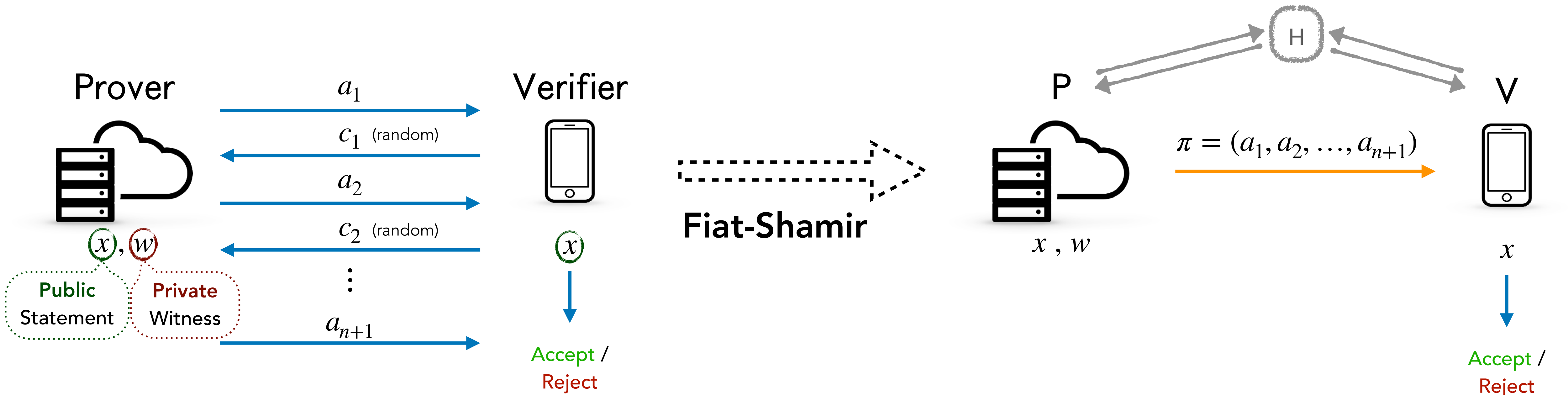
## Security:

Soundness: If  $x$  has no  $w$ , then  $V$  rejects.

Knowledge Soundness: If  $V$  accepts, then

$P^*$  must "know"  $w$ .

# Strong Fiat-Shamir for Adaptive Security

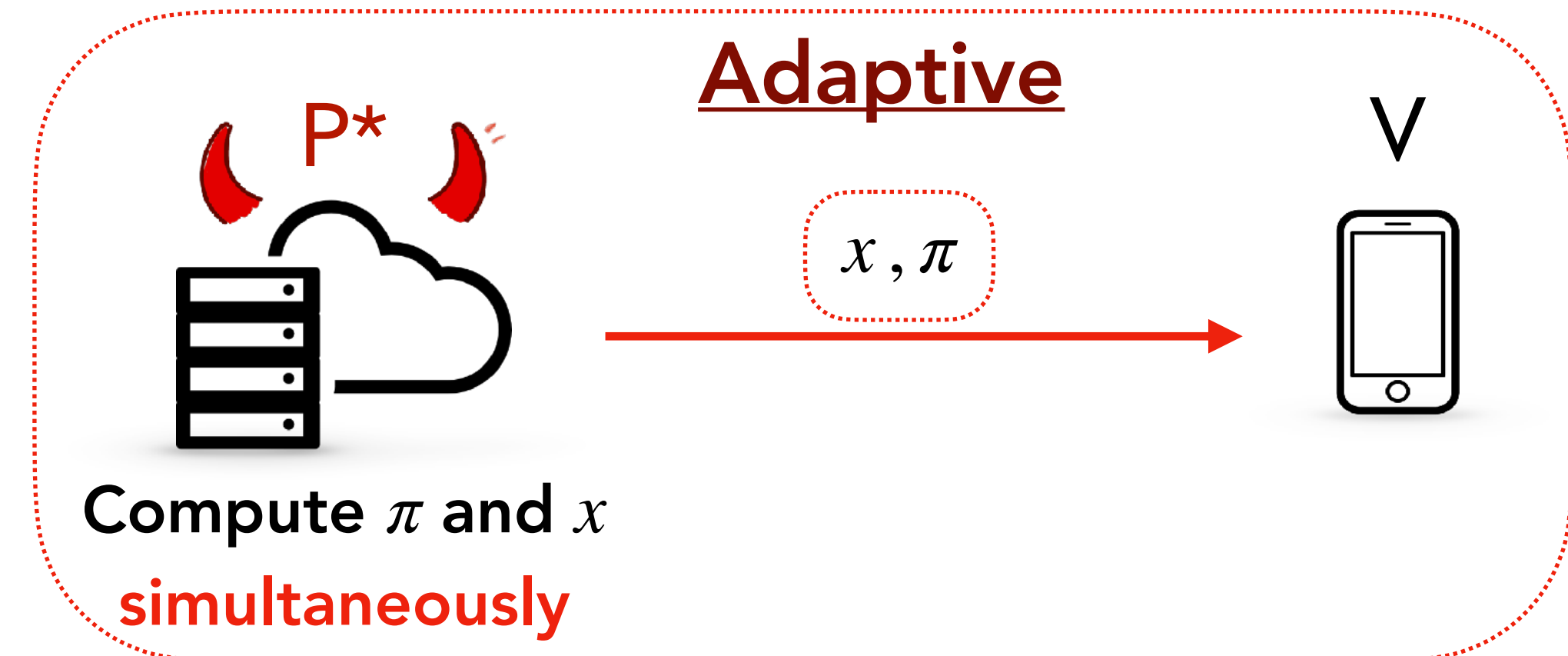


## Security:

Soundness: If  $x$  has no  $w$ , then  $V$  rejects.

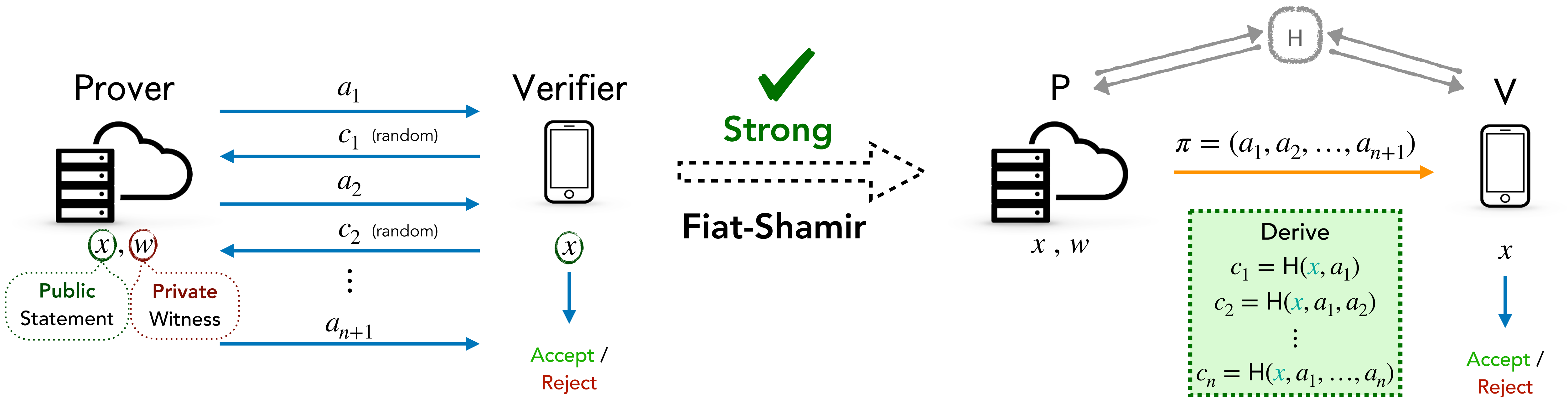
Knowledge Soundness: If  $V$  accepts, then

$P^*$  must "know"  $w$ .





# Strong Fiat-Shamir for Adaptive Security

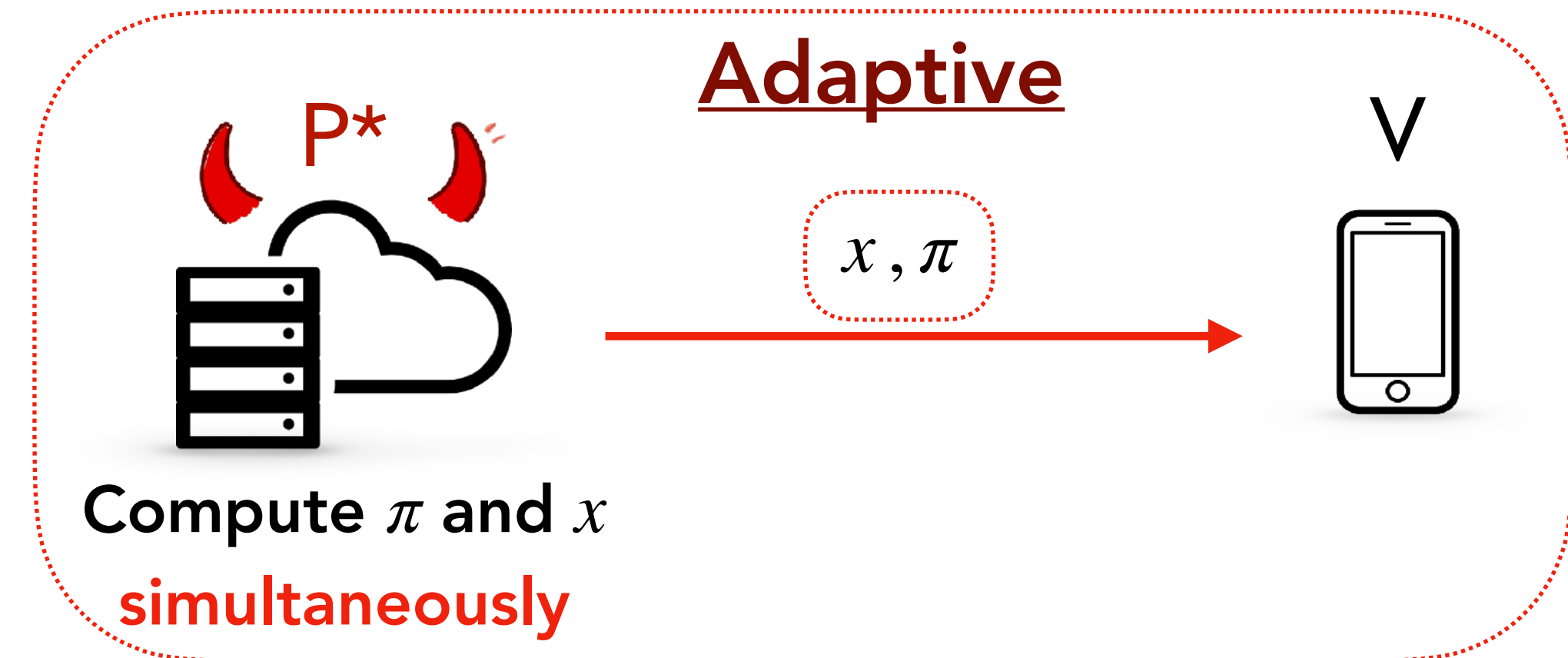


## Security:

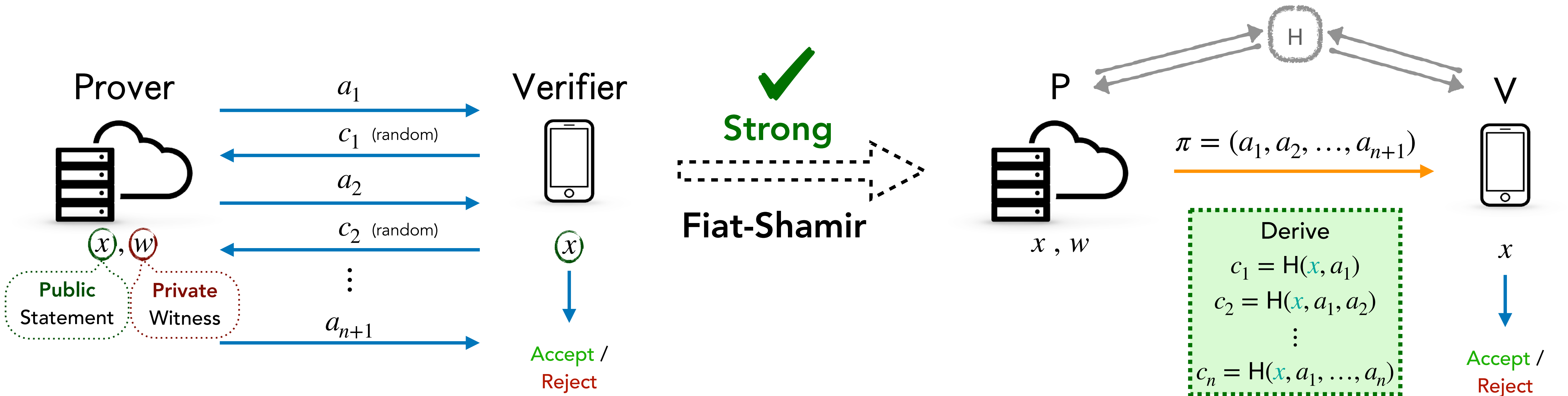
Soundness: If  $x$  has no  $w$ , then  $V$  rejects.

Knowledge Soundness: If  $V$  accepts, then

$P^*$  must "know"  $w$ .



# Strong Fiat-Shamir for Adaptive Security

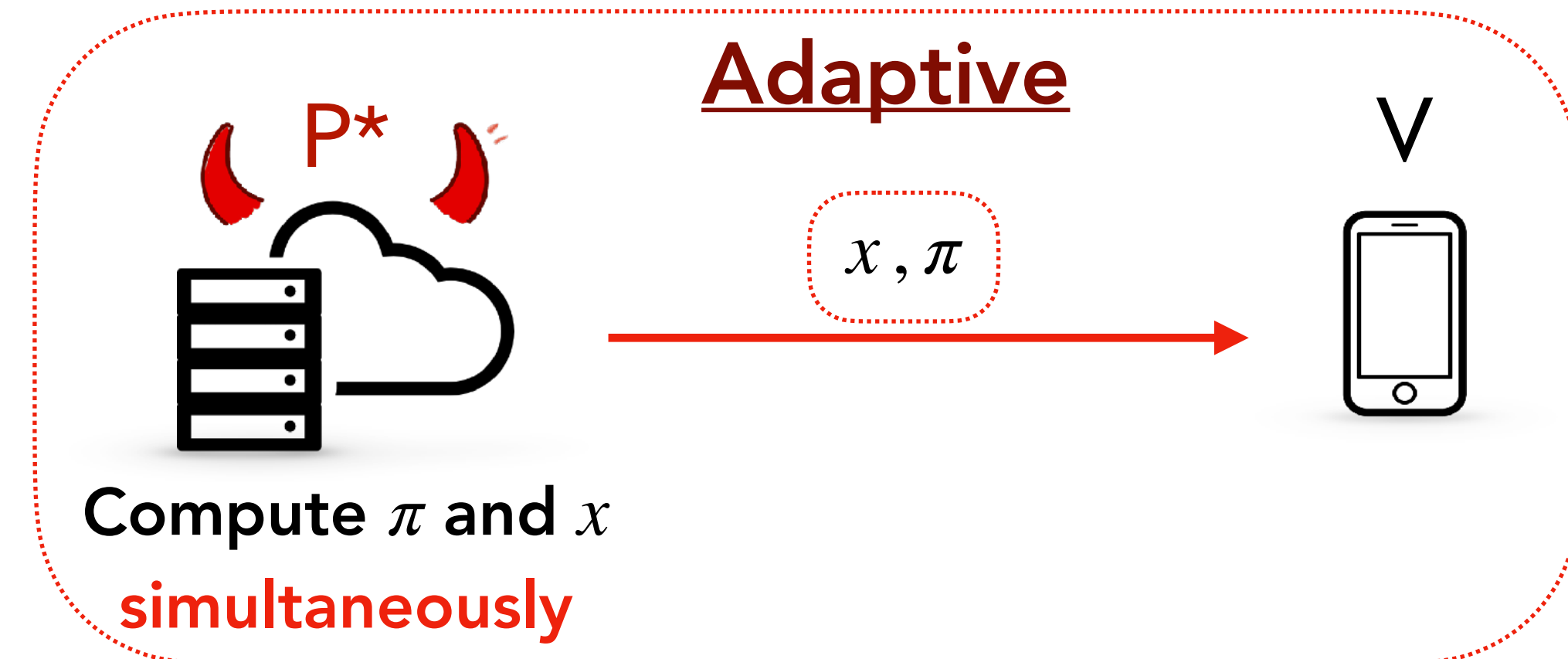


## Security:

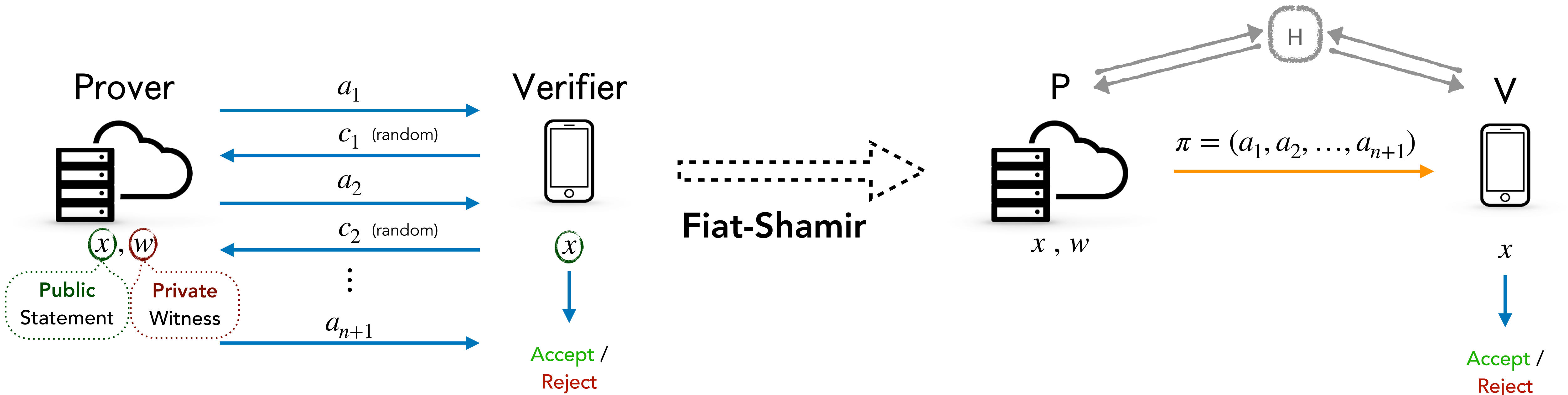
Soundness: If  $x$  has no  $w$ , then  $V$  rejects. ✓

Knowledge Soundness: If  $V$  accepts, then ✓

$P^*$  must "know"  $w$ .



# Weak Fiat-Shamir and Attacks

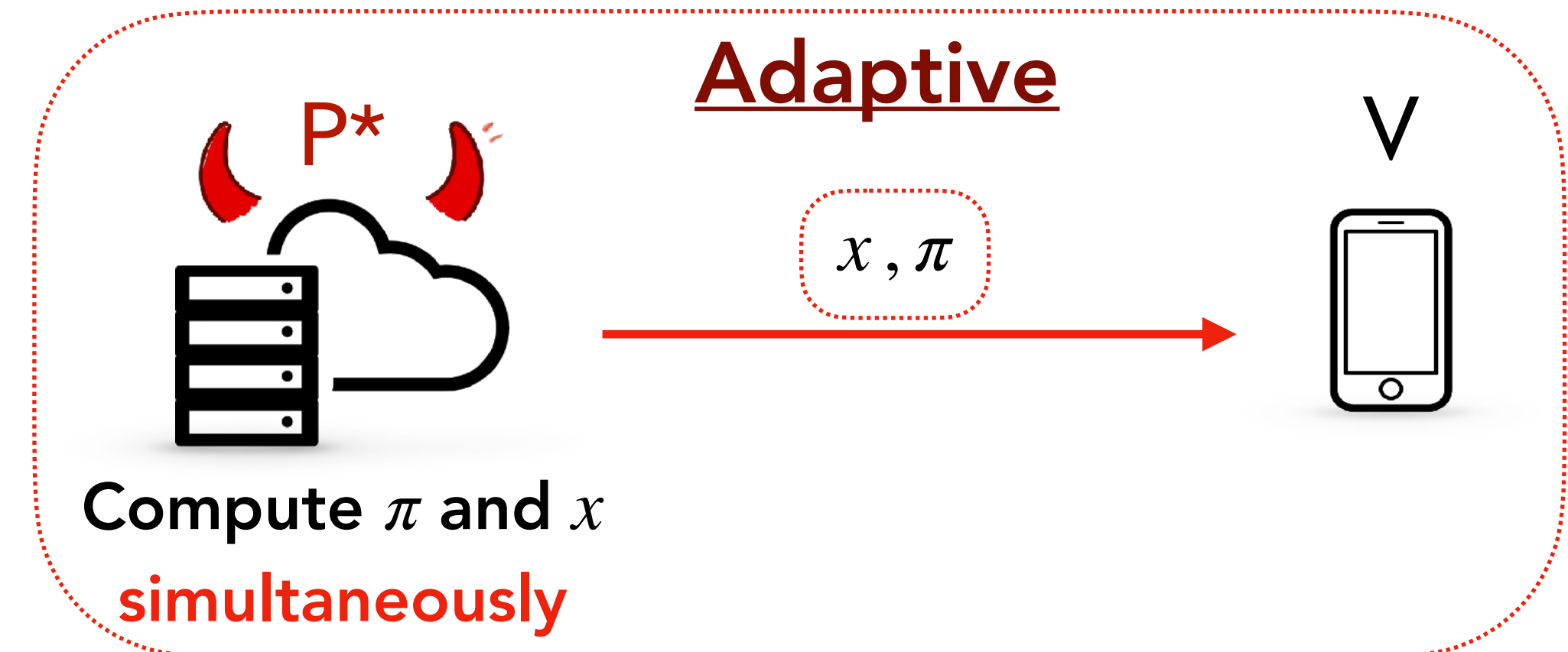


## Security:

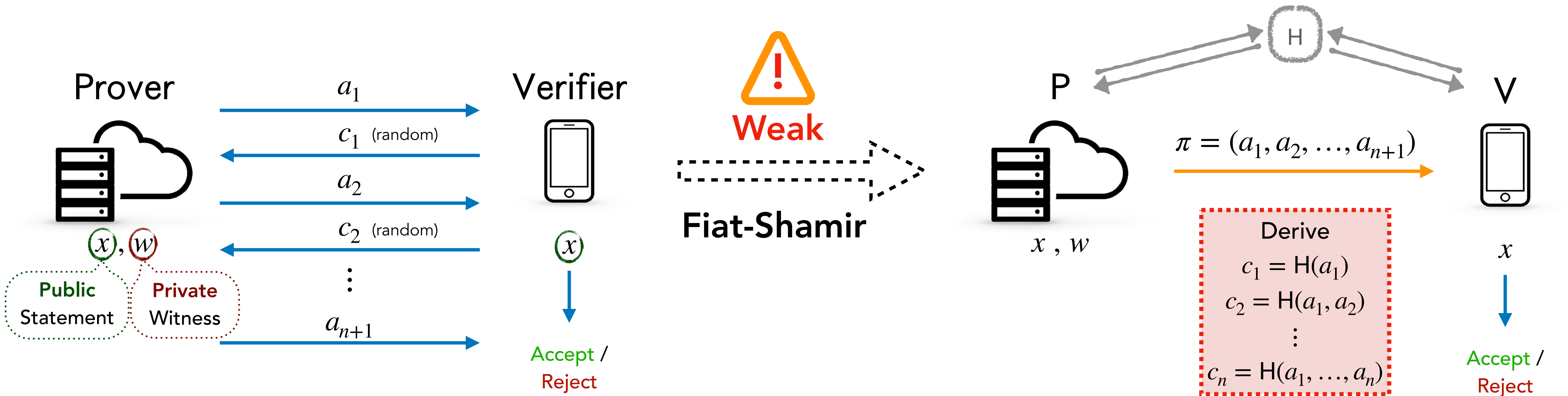
Soundness: If  $x$  has no  $w$ , then  $V$  rejects.

Knowledge Soundness: If  $V$  accepts, then

$P^*$  must "know"  $w$ .



# Weak Fiat-Shamir and Attacks

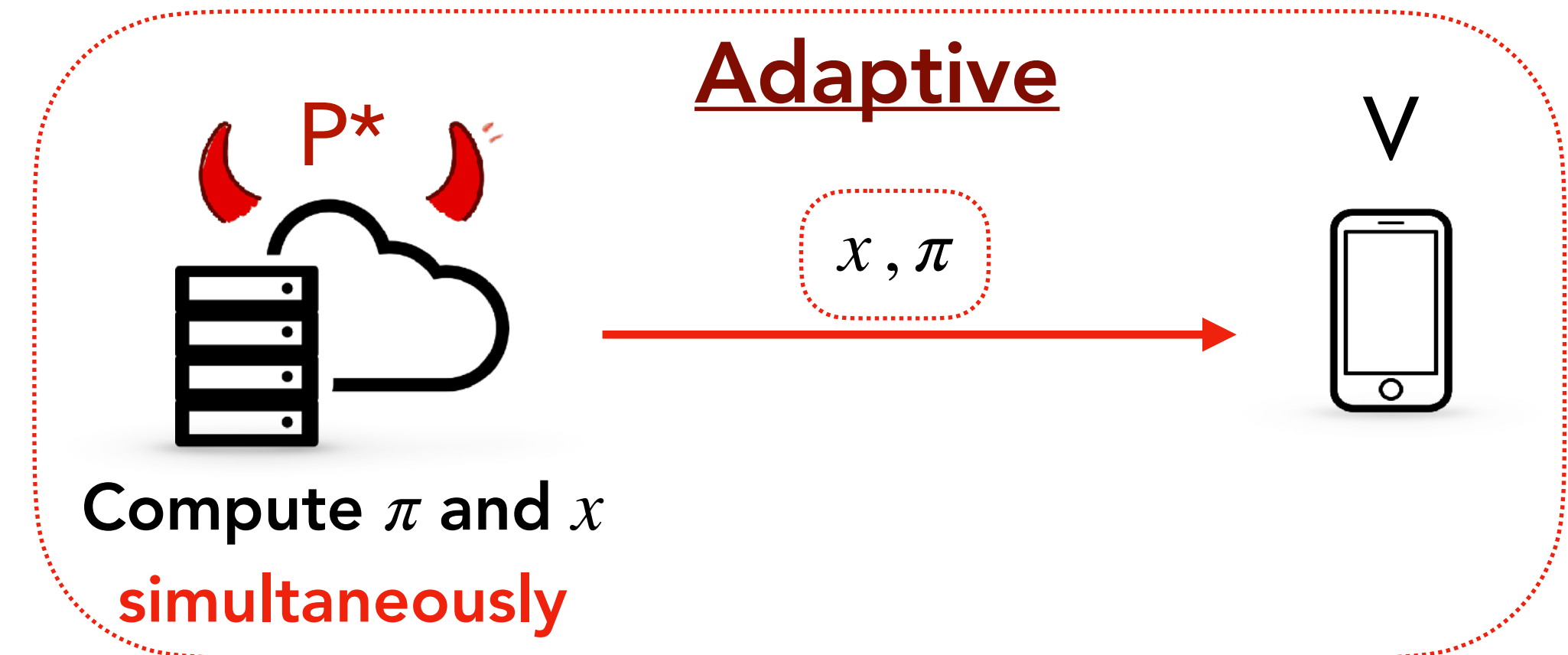


## Security:

Soundness: If  $x$  has no  $w$ , then  $V$  rejects.

Knowledge Soundness: If  $V$  accepts, then

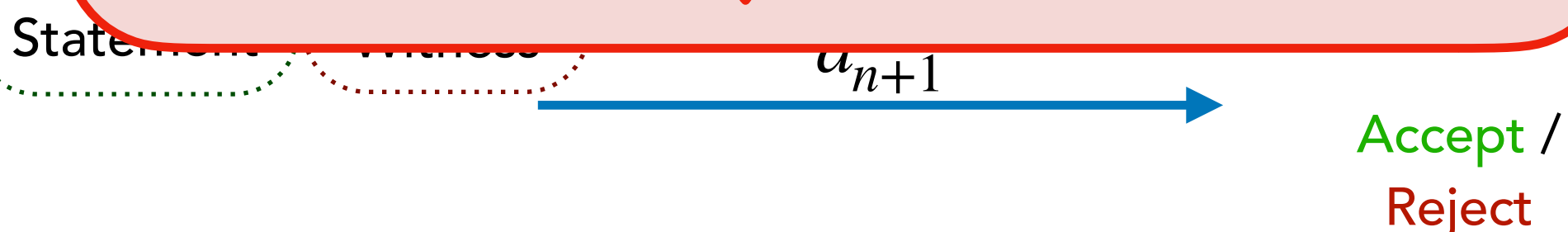
$P^*$  must "know"  $w$ .



# Weak Fiat-Shamir and Attacks

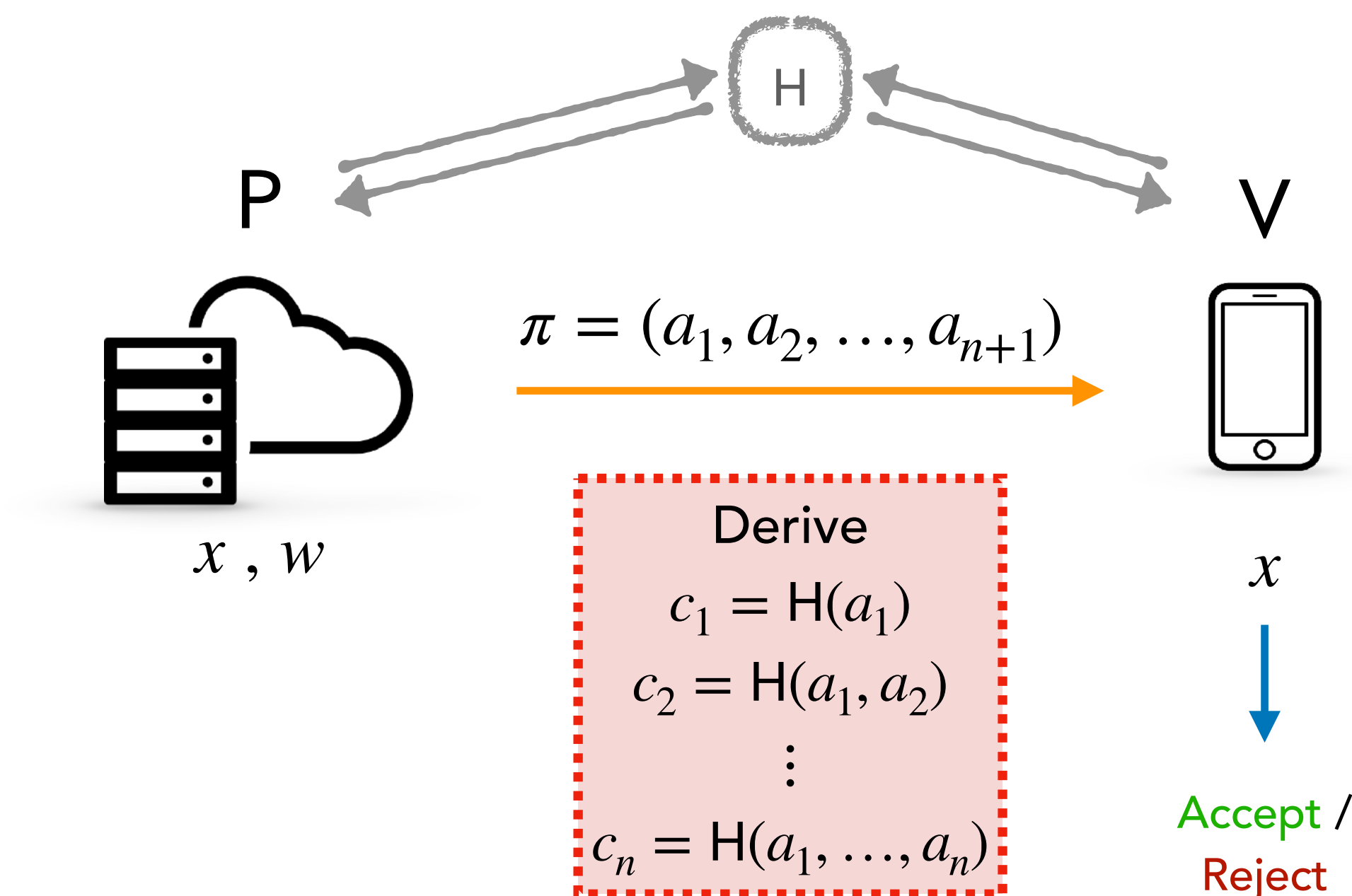
## Simple $\Sigma$ -Protocols

(e.g. Schnorr, Girault, Chaum-Pedersen)



Weak

Fiat-Shamir

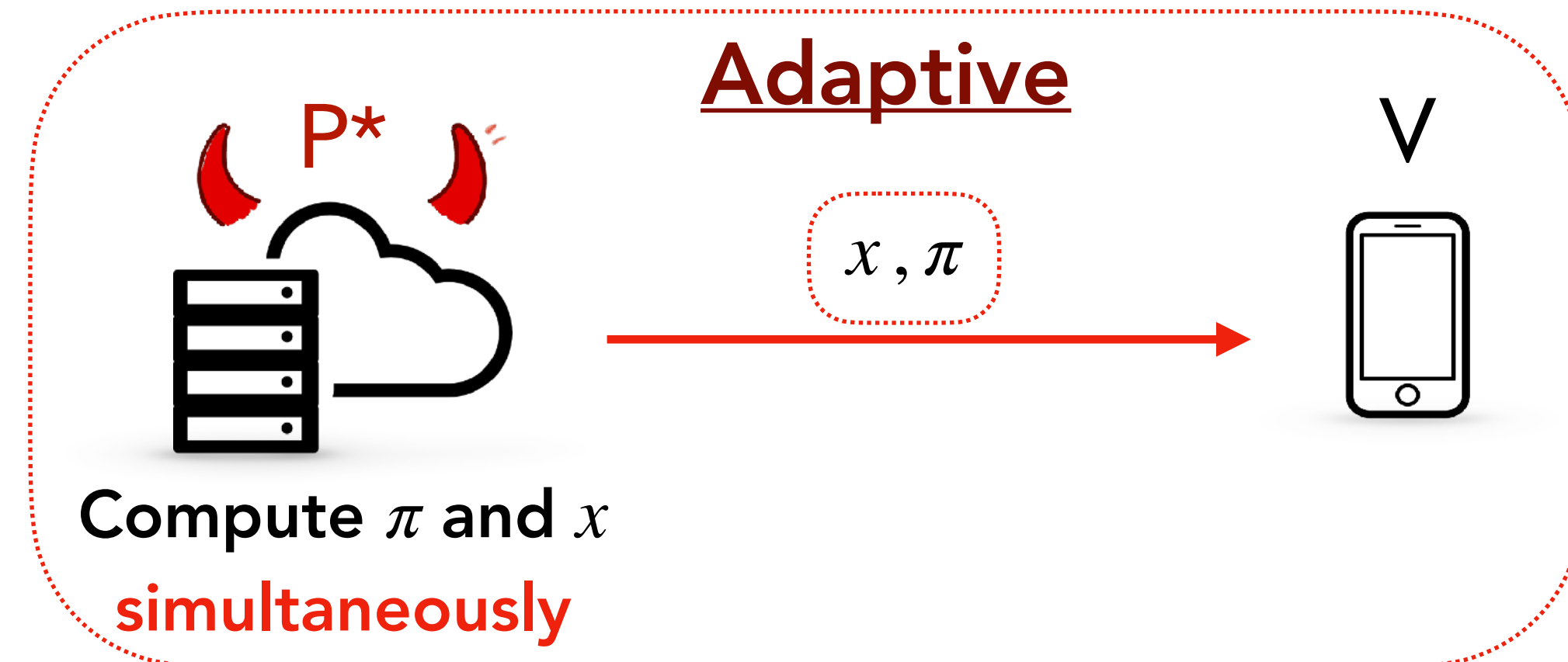


## Security:

Soundness: If  $x$  has no  $w$ , then  $V$  rejects. 

Knowledge Soundness: If  $V$  accepts, then 


$P^*$  must "know"  $w$ .

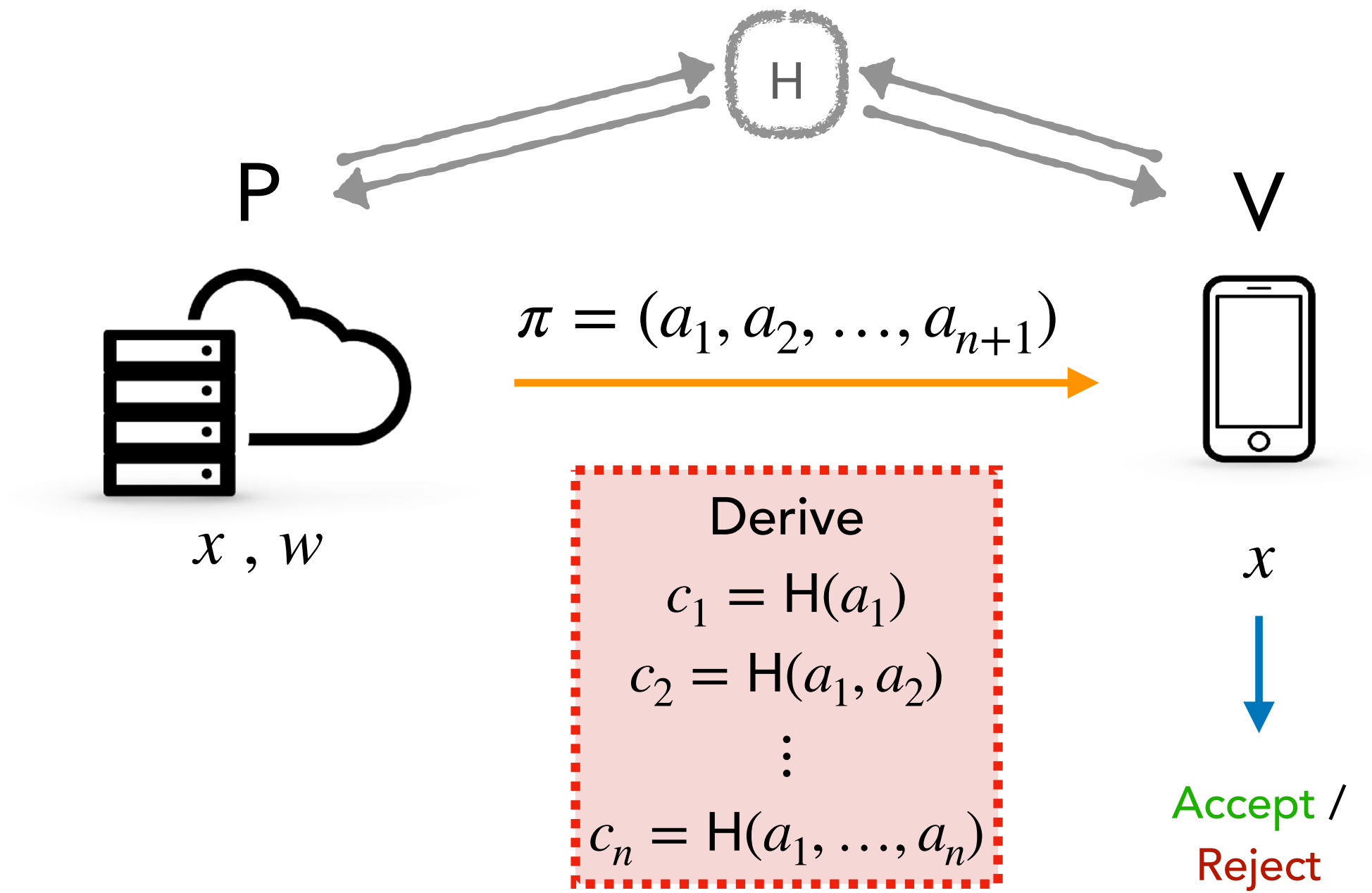


# Weak Fiat-Shamir and Attacks

Simple  $\Sigma$ -Protocols  
 (e.g. Schnorr, Girault, Chaum-Pedersen)

✘

  
**Weak**  
 ----->  
**Fiat-Shamir**



Statement  $\xrightarrow{a_{n+1}}$  Accept /

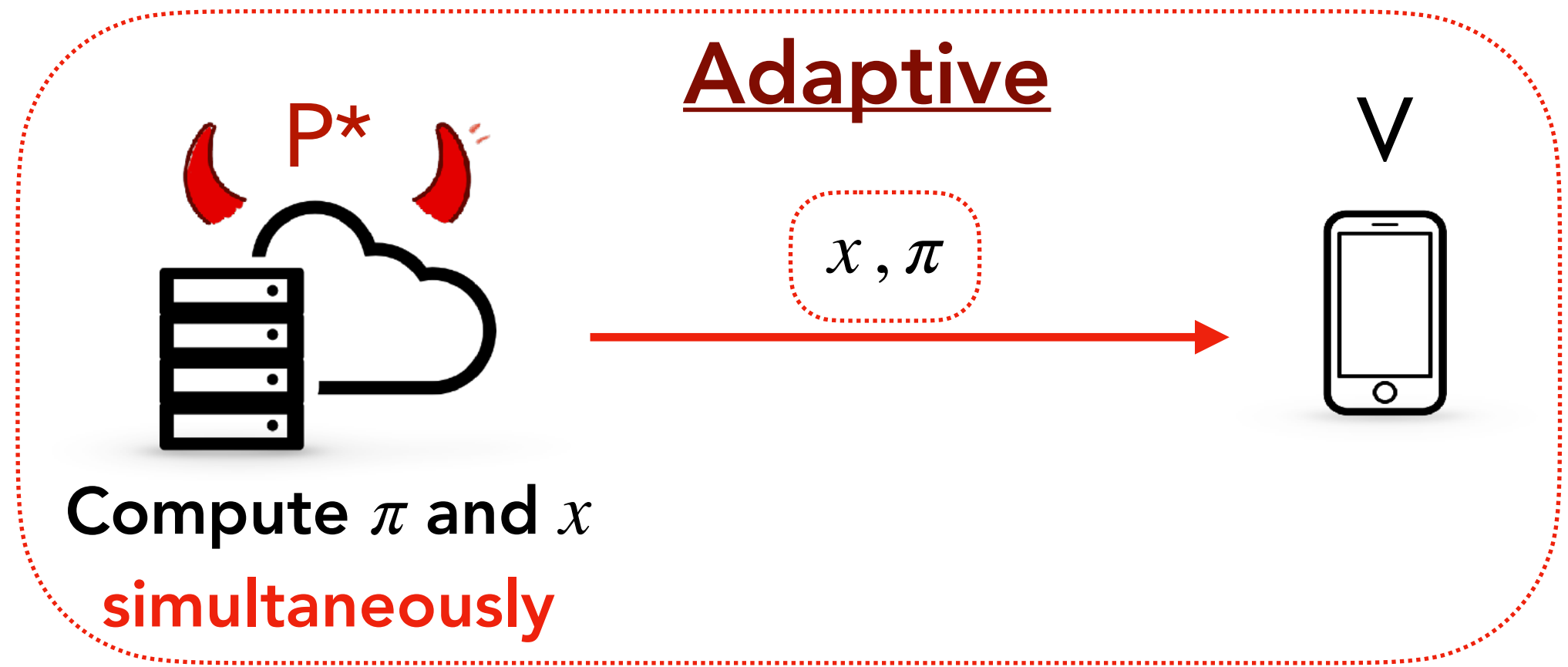
**How not to Prove Yourself:  
 Pitfalls of the Fiat-Shamir Heuristic and  
 Applications to Helios**

David Bernhard<sup>1</sup>, Olivier Pereira<sup>2</sup>, and Bogdan Warinschi<sup>1</sup>

✘  
✘

**How not to prove your election outcome**

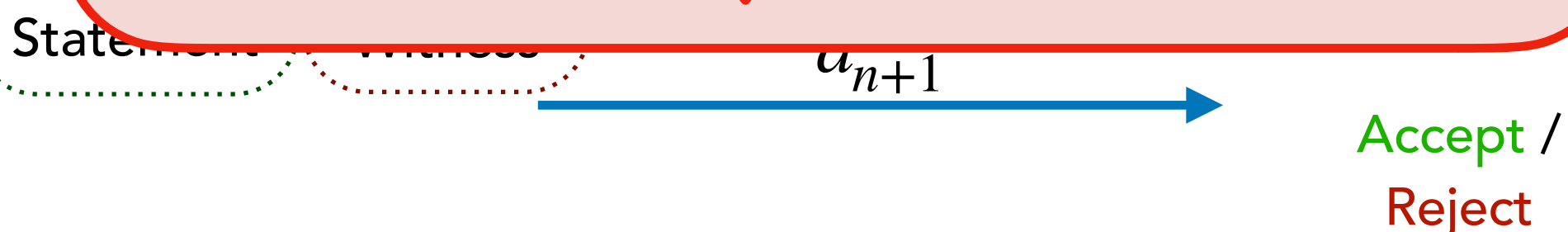
Thomas Haines<sup>\*</sup>, Sarah Jamie Lewis<sup>†</sup>, Olivier Pereira<sup>‡</sup>, and Vanessa Teague<sup>§</sup>



# Weak Fiat-Shamir and Attacks

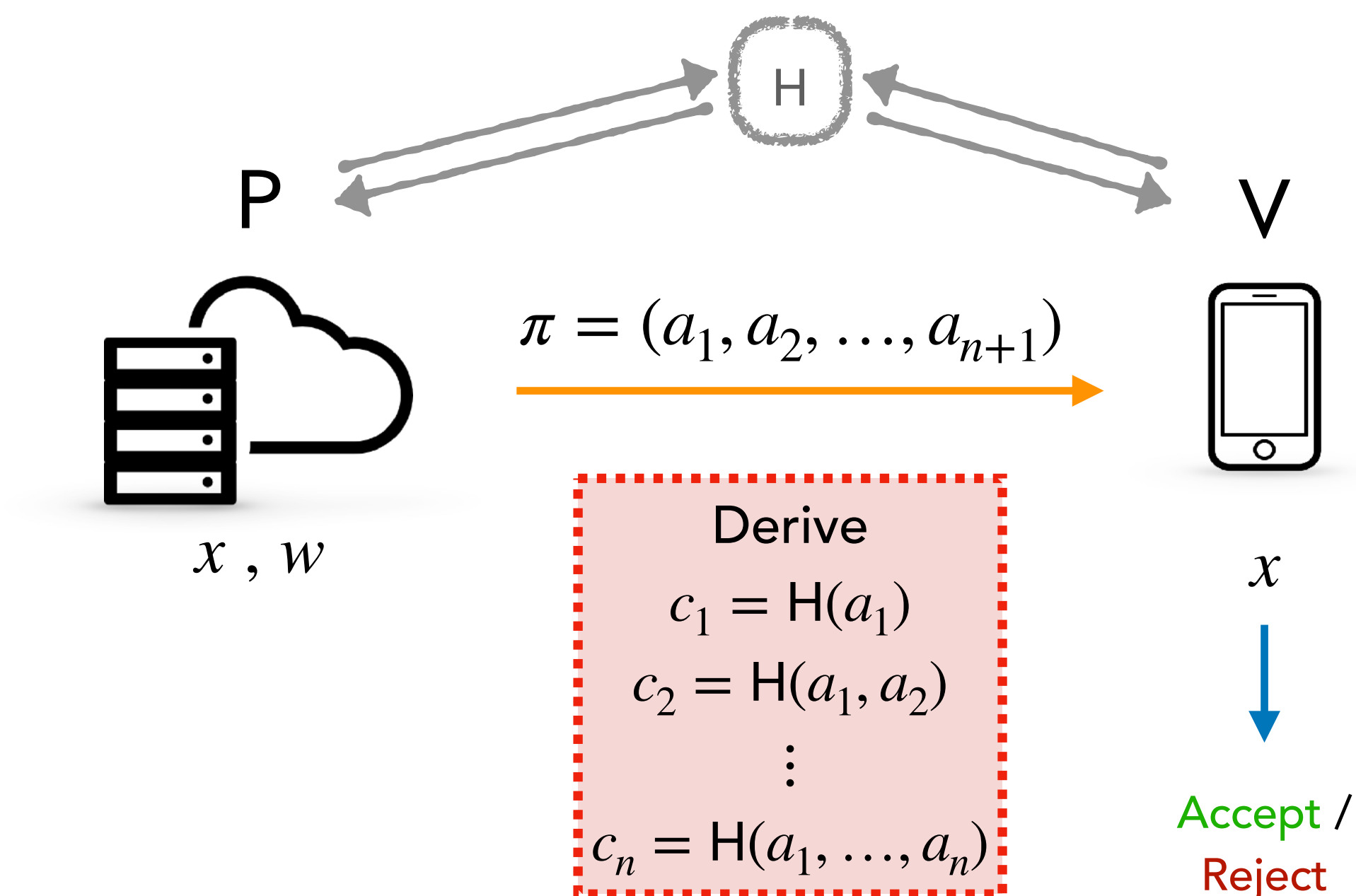
## Simple $\Sigma$ -Protocols

(e.g. Schnorr, Girault, Chaum-Pedersen)



Weak

Fiat-Shamir

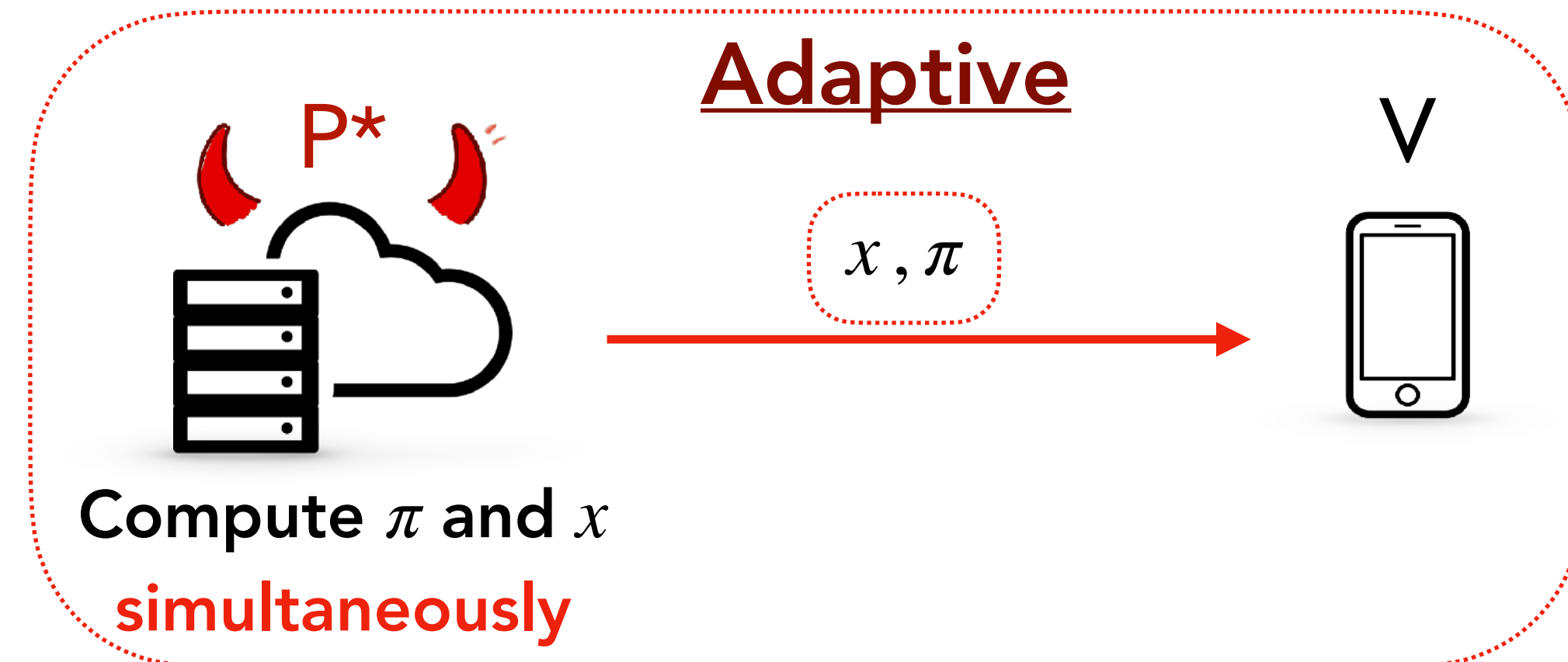


## Security:

Soundness: If  $x$  has no  $w$ , then  $V$  rejects. 

Knowledge Soundness: If  $V$  accepts, then 

$P^*$  must "know"  $w$ .



# Weak Fiat-Shamir and Attacks

## Simple $\Sigma$ -Protocols

(e.g. Schnorr, Girault, Chaum-Pedersen)

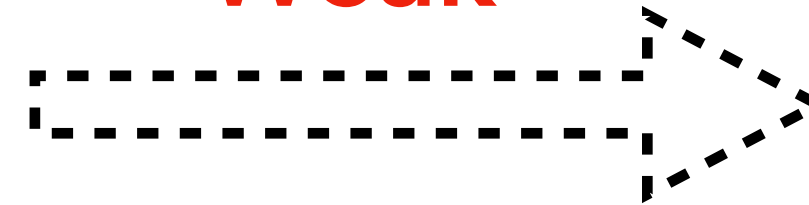


$a_{n+1}$

Accept /  
Reject



Weak



Fiat-Shamir

## Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)

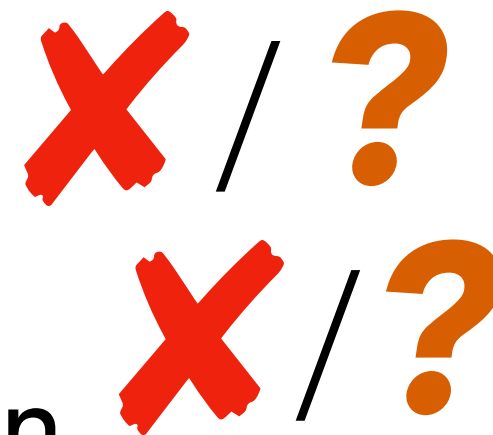


$$c_n = H(a_1, \dots, a_n)$$

Accept /  
Reject

## Security:

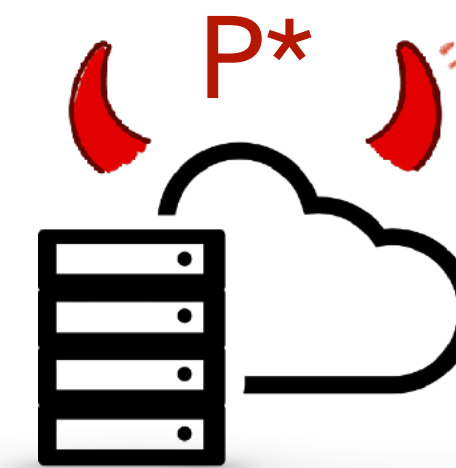
Soundness: If  $x$  has no  $w$ , then  $V$  rejects.



Knowledge Soundness: If  $V$  accepts, then

$P^*$  must "know"  $w$ .

## Adaptive



Compute  $\pi$  and  $x$   
simultaneously

$x, \pi$





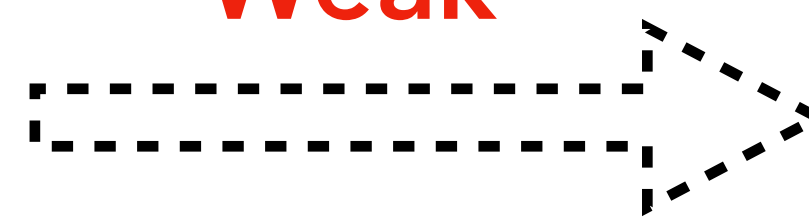
# Weak Fiat-Shamir and Attacks

## Simple $\Sigma$ -Protocols

(e.g. Schnorr, Girault, Chaum-Pedersen)



Weak



Fiat-Shamir

## Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)



1. Are there *Weak Fiat-Shamir Attacks* against *Modern Proof Systems*?

$P^*$  must "know"  $w$ .

Compute  $\pi$  and  $x$   
**simultaneously**

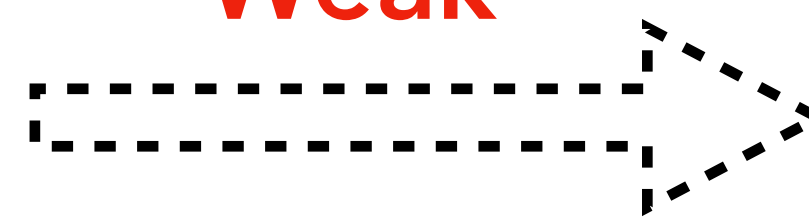
# Weak Fiat-Shamir and Attacks

## Simple $\Sigma$ -Protocols

(e.g. Schnorr, Girault, Chaum-Pedersen)



Weak



Fiat-Shamir

## Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)



1. *Are there Weak Fiat-Shamir Attacks against Modern Proof Systems?*
2. *Do Modern-Day Systems Implement Weak Fiat-Shamir?*

$P^*$  must "know"  $w$ .

Compute  $\pi$  and  $x$   
**simultaneously**

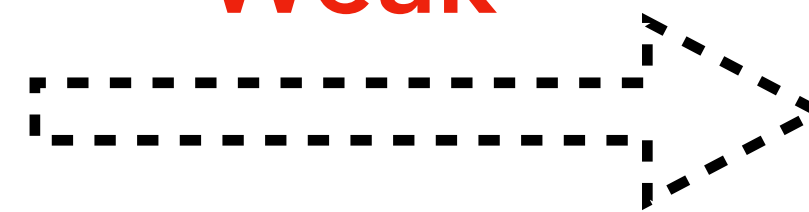
# Weak Fiat-Shamir and Attacks

## Simple $\Sigma$ -Protocols

(e.g. Schnorr, Girault, Chaum-Pedersen)



Weak



Fiat-Shamir

## Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)



1. *Are there Weak Fiat-Shamir Attacks against Modern Proof Systems?*
2. *Do Modern-Day Systems Implement Weak Fiat-Shamir?*
3. *How Severe are Weak Fiat-Shamir Vulnerabilities?*

$P^*$  must "know"  $w$ .

Compute  $\pi$  and  $x$   
**simultaneously**

# Our Contributions

# Our Contributions

1. Comprehensive Survey of 75+ open-source implementations:

⇒ 36 weak F-S vulnerabilities across 12 different proof systems.

# Our Contributions

1. Comprehensive Survey of 75+ open-source implementations:

⇒ 36 weak F-S vulnerabilities across 12 different proof systems.

Proof System	Codebase	Weak F-S?
Bulletproofs [22]	bp-go [87]	✓
	bulletproof-js [2]	✓
	simple-bulletproof-js [83]	✓
	BulletproofSwift [20]	✓
	python-bulletproofs [78]	✓
	adjoint-bulletproofs [3]	✓
	zkSen [98]	✓
	incognito-chain [51]	✓♦
	encoins-bulletproofs [33]	✓♦
	ZenGo-X [96]	✓♦
	zkp [52]	✓♦
	ckb-zkp [81]	✓♦
	bulletproofsrb [21]	✓♦
	monero [68]	✗
	dalek-bulletproofs [29]	✗
	secp256k1-zkp [75]	✗
	bulletproofs-ocaml [74]	✗
tari-project [85]	✗	
Litecoin [59]	✗	
Grin [44]	✗	
Bulletproofs variant [40]	dalek-bulletproofs [29]	✓♦
	cpp-lwevss [60]	✗
Sonic [61]	ebfull-sonic [18]	✓
	lx-sonic [58]	✓
	iohk-sonic [53]	✗
	adjoint-sonic [4]	✗
Schnorr [79]	noknow-python [7]	✓

Proof System	Codebase	Weak F-S?
Plonk [37]	anoma-plonkup [6]	✓
	gnark [17]	✓♦
	dusk-network [31]	✓♦
	snarkjs [50]	✓♦
	ZK-Garage [97]	✓♦
	plonky [67]	✗
	ckb-zkp [81]	✗
	halo2 [93]	✗
	o1-labs [71]	✗
	jellyfish [34]	✗
	matter-labs [62]	✗
	aztec-connect [8]	✗
Wesolowski's VDF [90]	0xProject [1]	✓
	Chia [69]	✓
	Harmony [47]	✓
	POA Network [70]	✓
	IOTA Ledger [54]	✓
	master-thesis-ELTE [48]	✓
Hyrax [89]	ckb-zkp [81]	✓♦
	hyraxZK [49]	✗
Spartan [82]	Spartan [64]	✓♦
	ckb-zkp [81]	✓♦
Libra [91]	ckb-zkp [81]	✓♦
Brakedown [43]	Brakedown [19]	✓
Nova [57]	Nova [63]	✓♦
Gemini [16]	arkworks-gemini [38]	✓♦
Girault [42]	zk-paillier [95]	✓♦

# Our Contributions

1. Comprehensive Survey of 75+ open-source implementations:

⇒ 36 weak F-S vulnerabilities across 12 different proof systems.

# Our Contributions

1. Comprehensive Survey of 75+ open-source implementations:

⇒ 36 weak F-S vulnerabilities across 12 different proof systems.

2. Explicit Attacks against Bulletproofs, Plonk, Spartan, and Wesolowski's VDF:

⇒ Provably break adaptive (knowledge) soundness.



# Our Contributions

1. Comprehensive Survey of 75+ open-source implementations:

⇒ 36 weak F-S vulnerabilities across 12 different proof systems.

2. Explicit Attacks against Bulletproofs, Plonk, Spartan, and Wesolowski's VDF:

⇒ Provably break adaptive (knowledge) soundness.

3. Case Studies of Practical Impacts:

⇒ Unlimited currency minting in two separate blockchain protocols

# Our Contributions

1. Comprehensive Survey of 75+ open-source implementations:
  - ⇒ 36 weak F-S vulnerabilities across 12 different proof systems.
2. Explicit Attacks against Bulletproofs, Plonk, Spartan, and Wesolowski's VDF:
  - ⇒ Provably break adaptive (knowledge) soundness.
3. Case Studies of Practical Impacts:
  - ⇒ Unlimited currency minting in two separate blockchain protocols
4. Discussion & Takeaways for Academics & Practitioners.

# Our Contributions

1. Comprehensive Survey of 75+ open-source implementations:  
⇒ 36 weak F-S vulnerabilities across 12 different proof systems.
2. Explicit Attacks against Bulletproofs, Plonk, Spartan, and Wesolowski's VDF:  
⇒ Provably break adaptive (knowledge) soundness.
3. Case Studies of Practical Impacts:  
⇒ Unlimited currency minting in two separate blockchain protocols
4. Discussion & Takeaways for Academics & Practitioners.

# **Weak Fiat-Shamir Attacks**

**(as easy as solving a linear equation)**

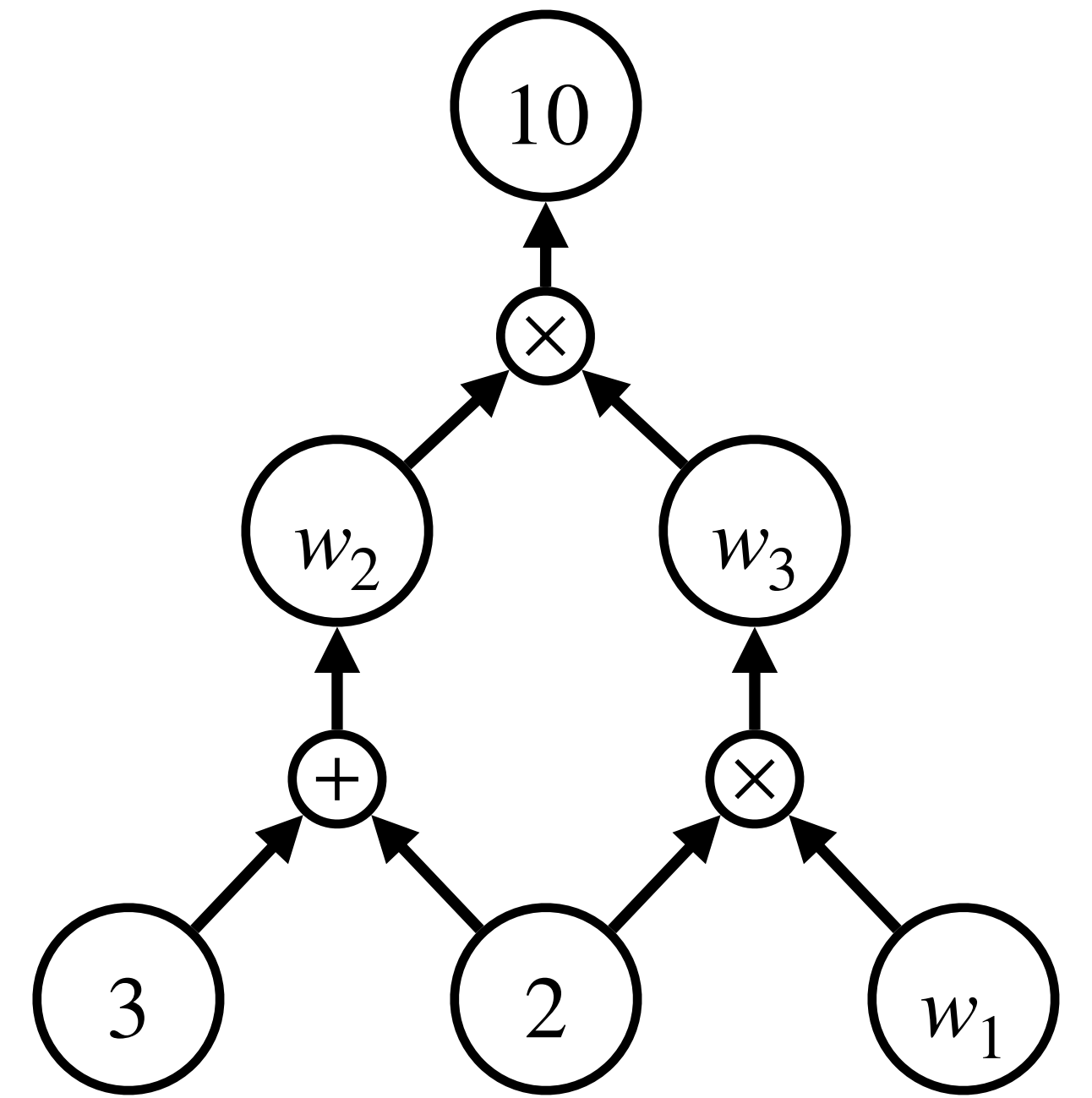
# Plonk - Protocol Description

# Plonk - Protocol Description

Constraint System:

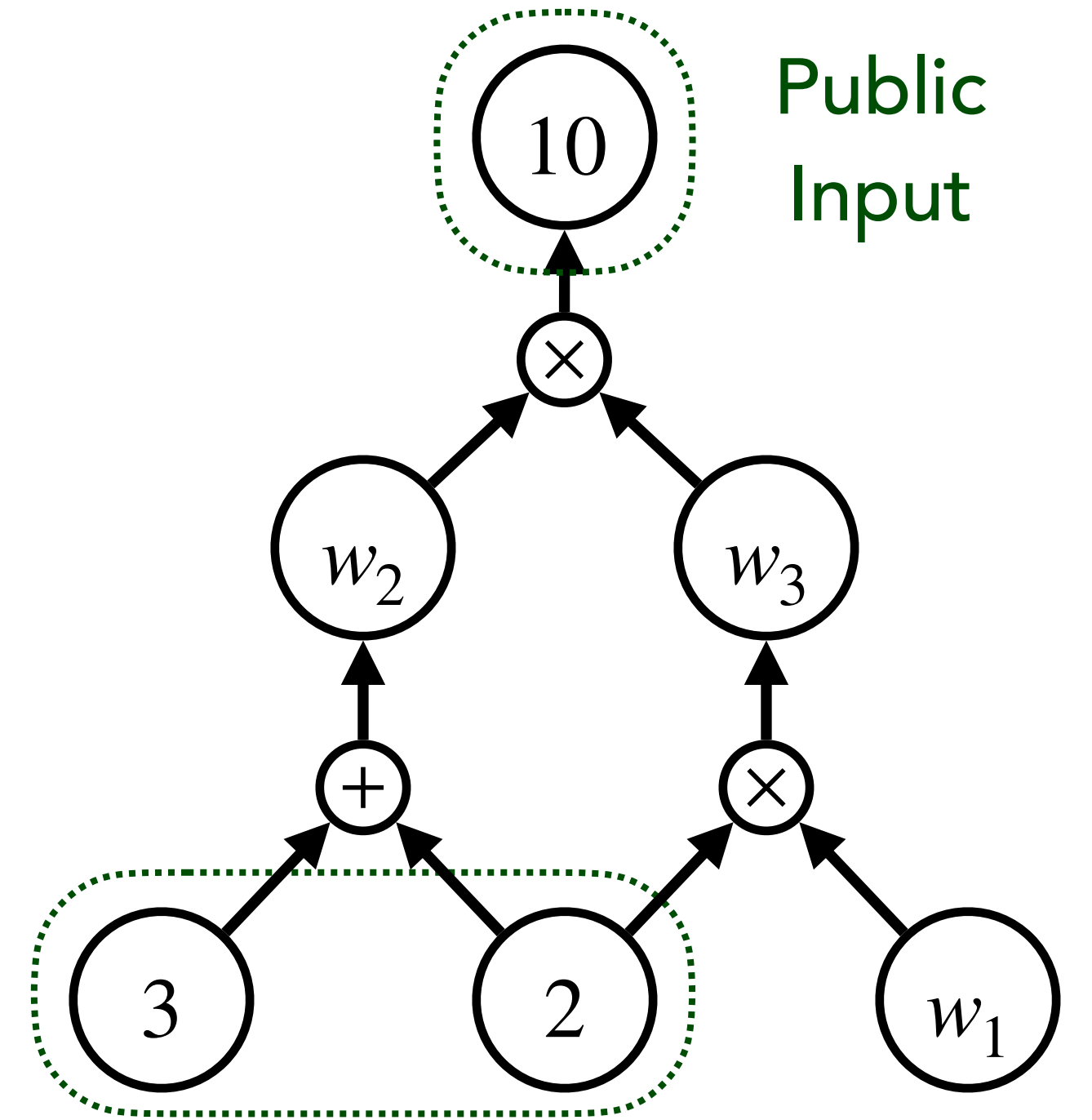
# Plonk - Protocol Description

Constraint System:



# Plonk - Protocol Description

Constraint System:

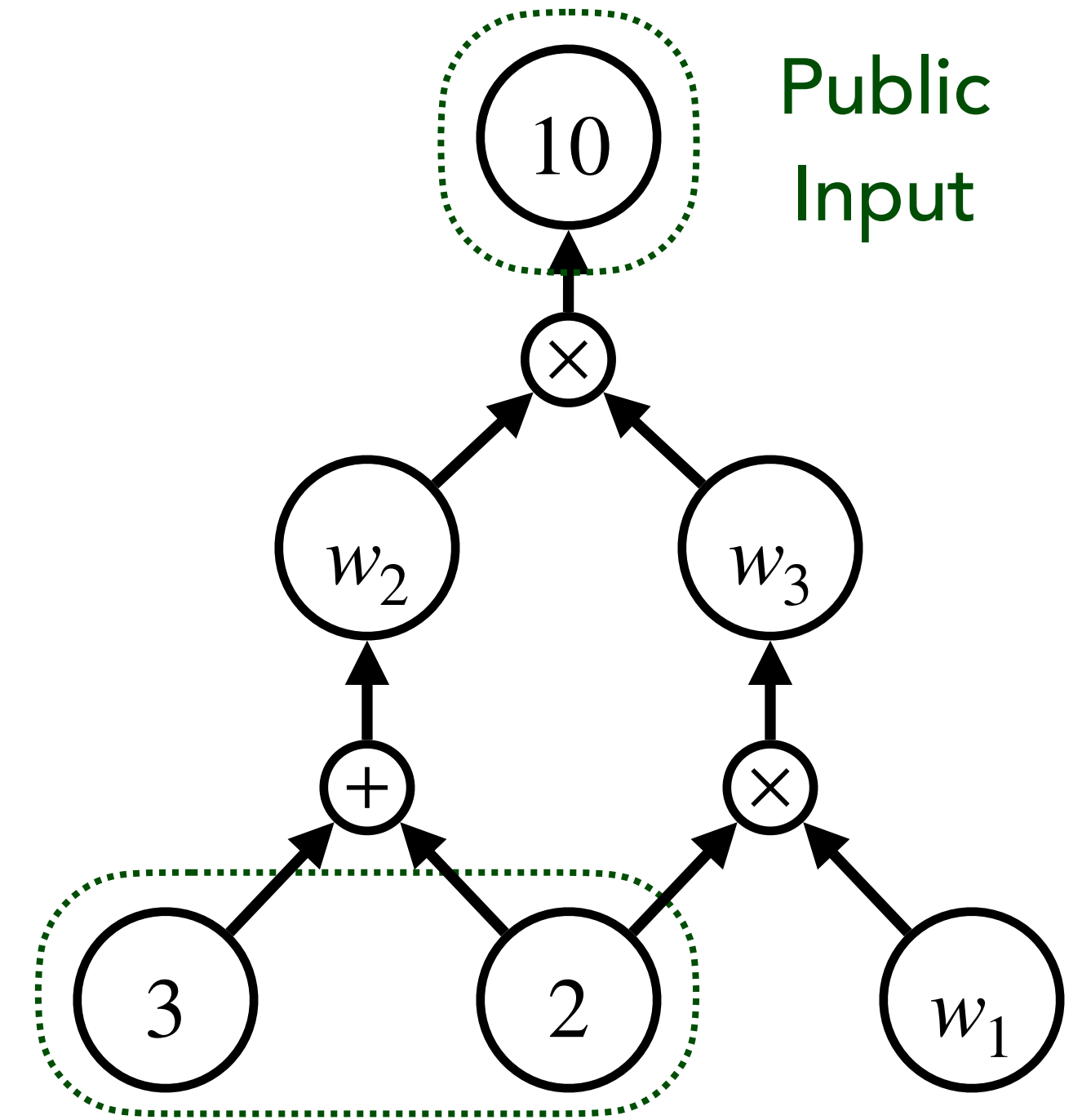




# Plonk - Protocol Description

## Constraint System:

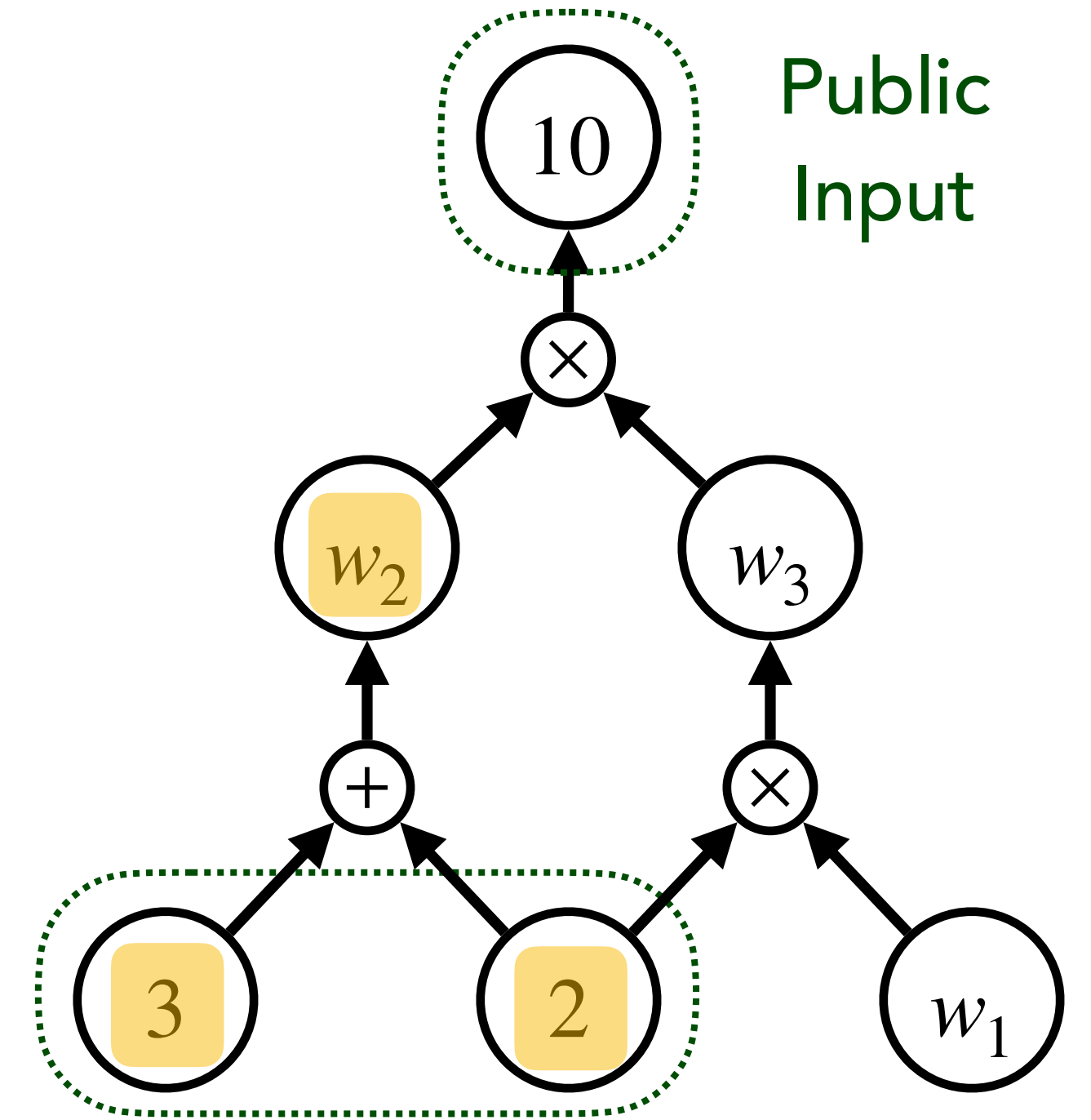
- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$



# Plonk - Protocol Description

## Constraint System:

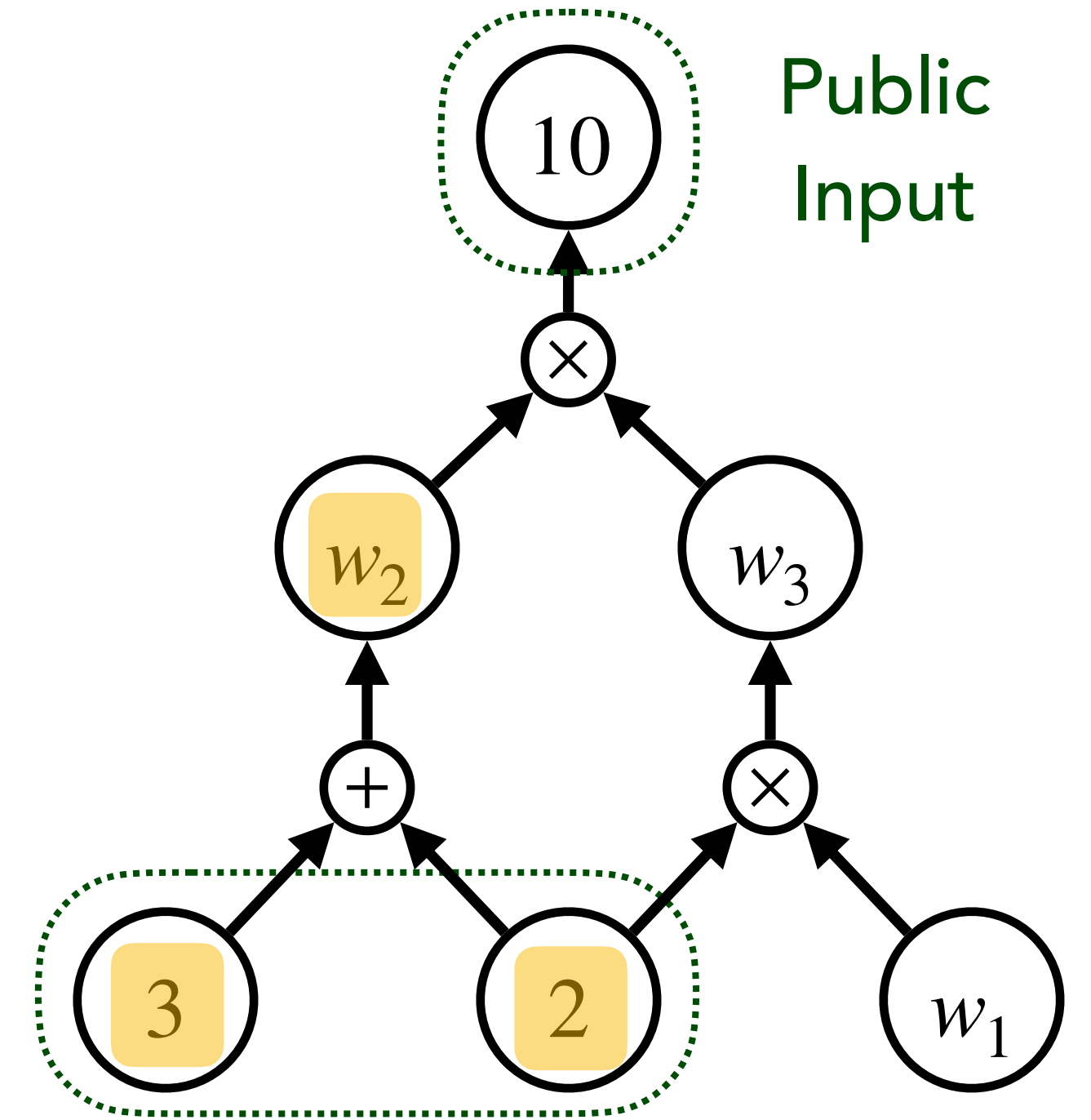
- Gate Vectors:  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$



# Plonk - Protocol Description

## Constraint System:

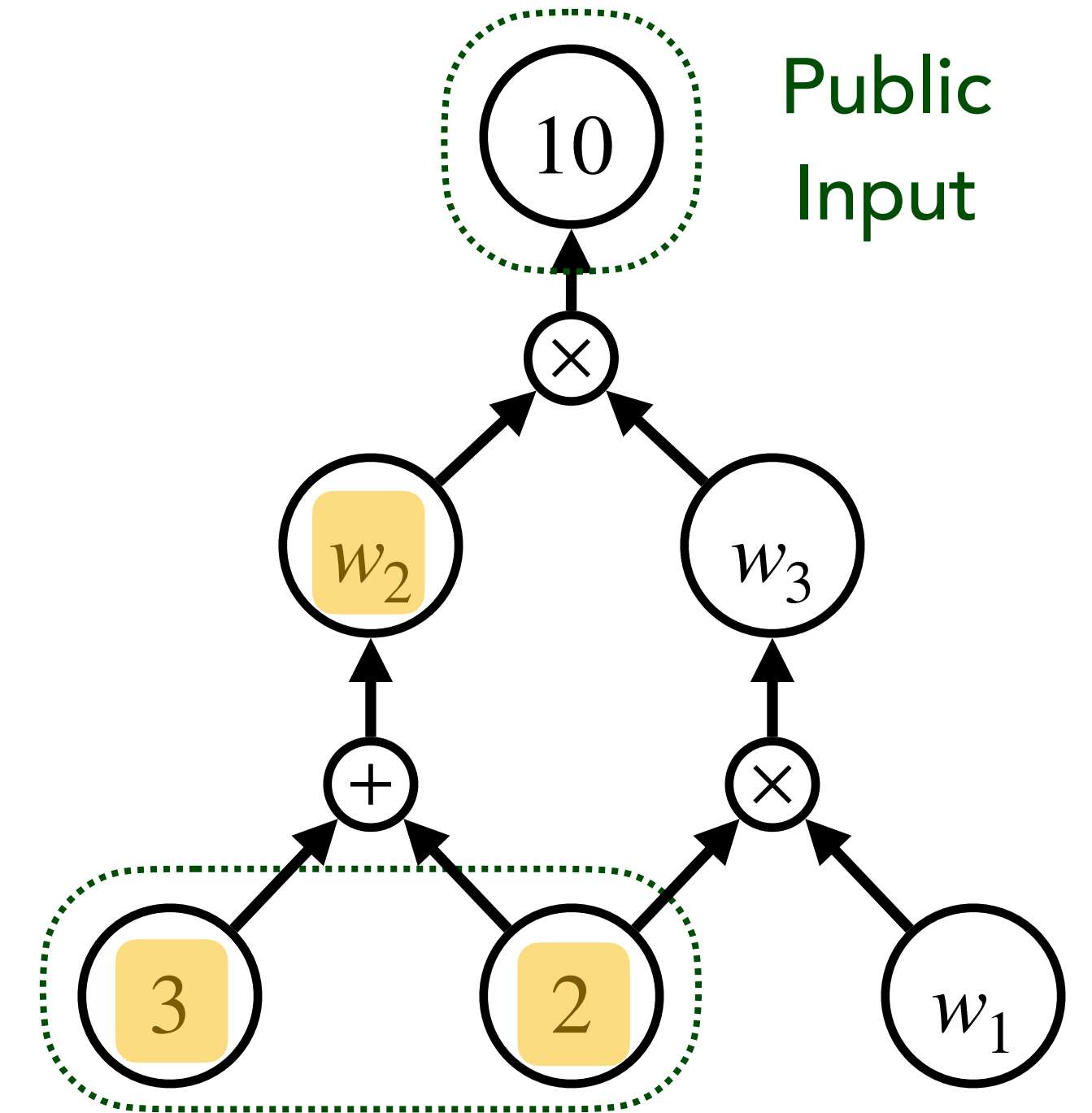
- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$



# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$

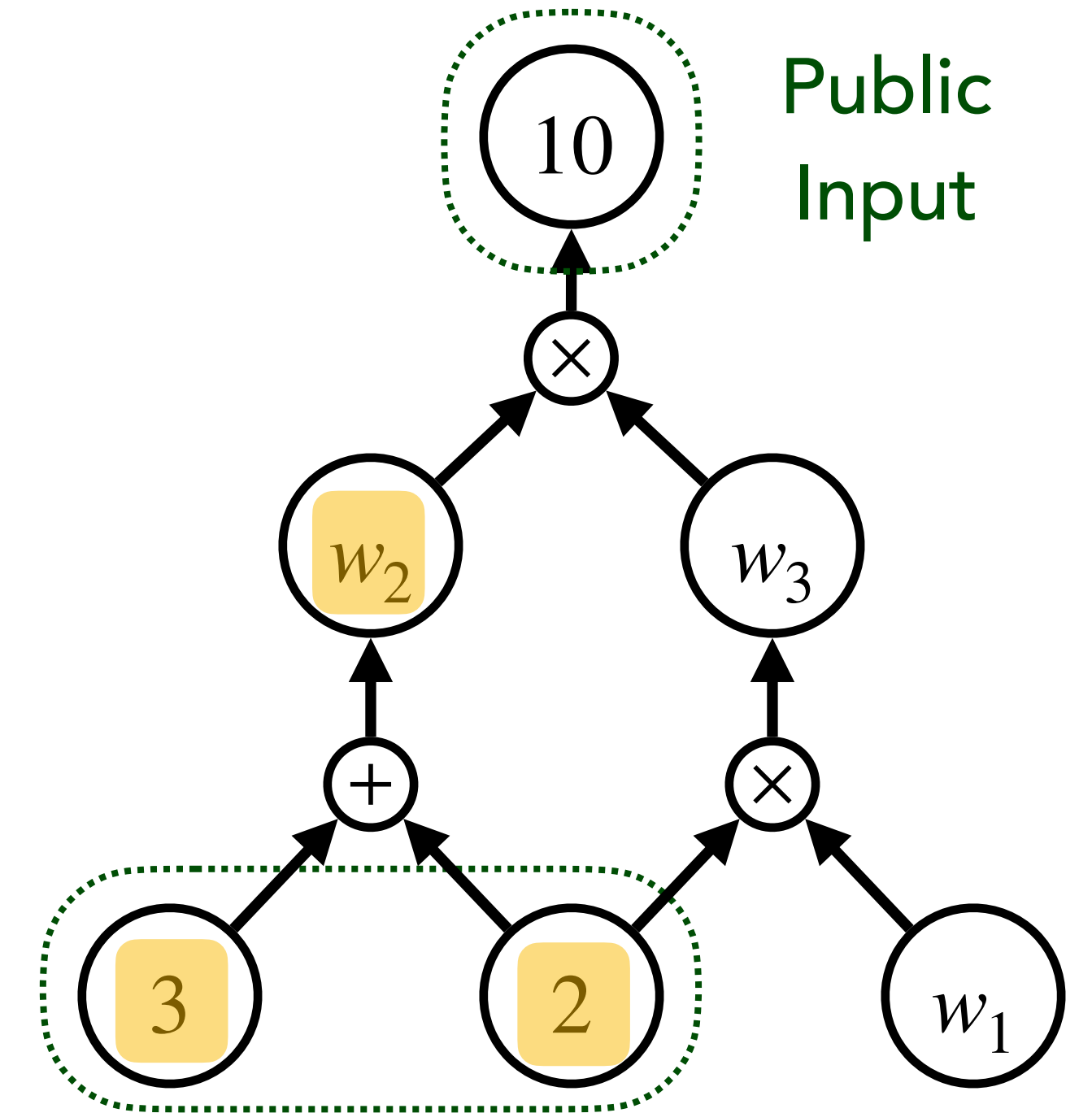


# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$

## Verification Equation:



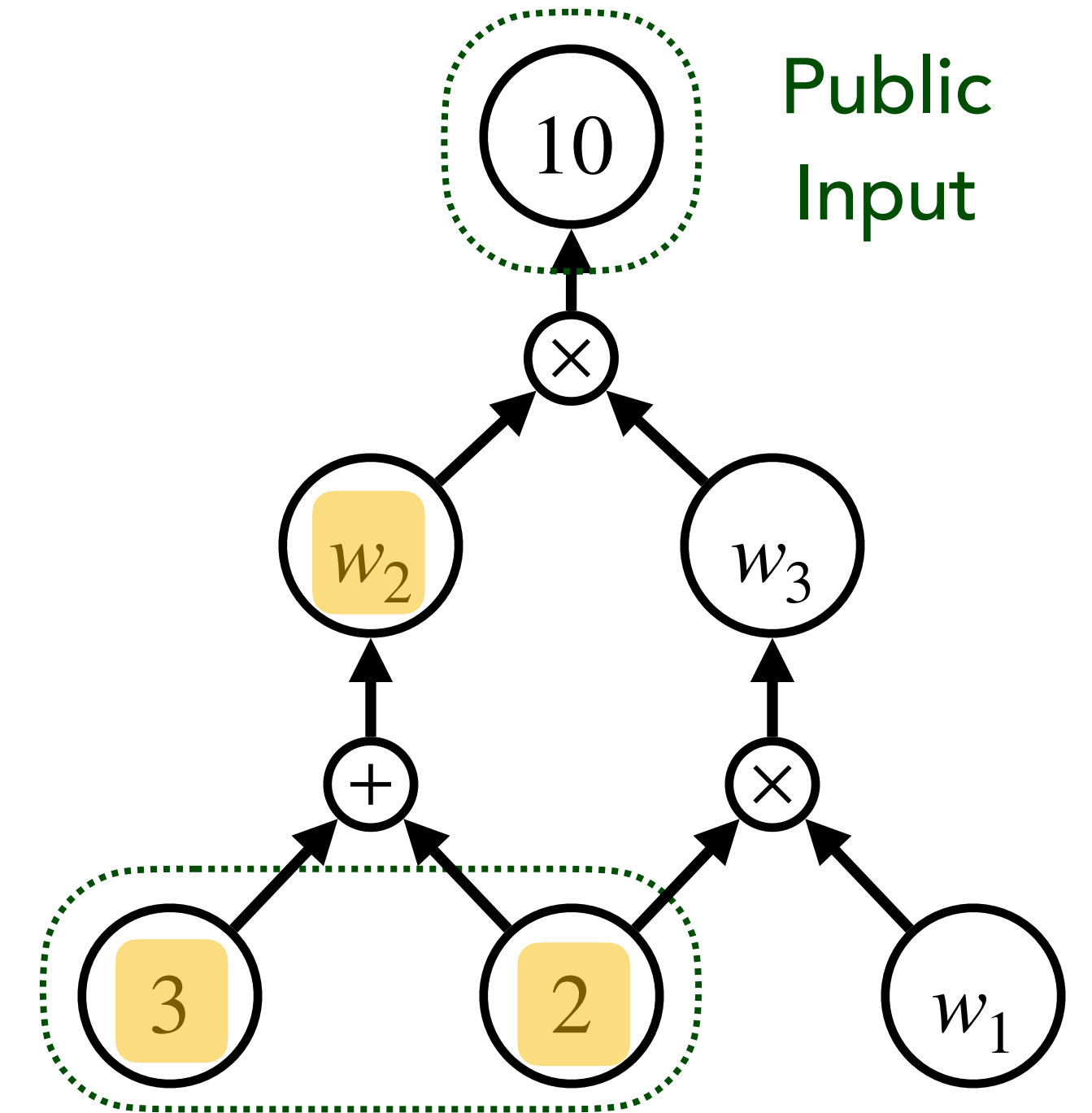
# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$

## Verification Equation:

$$\text{PI}(\zeta) + \text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta) = Z_H(\zeta) \cdot t(\zeta)$$



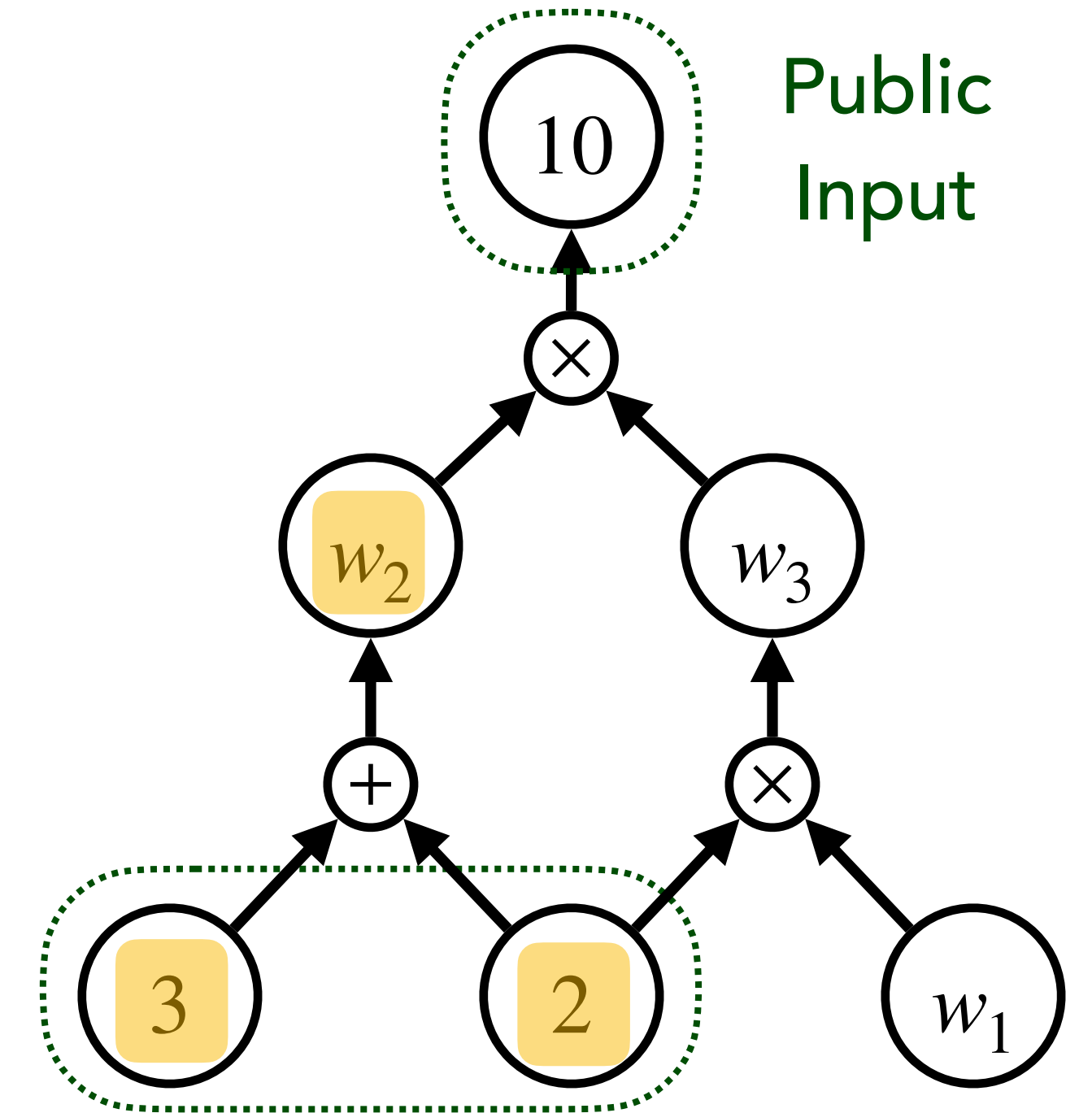
# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$

## Verification Equation:

$$\underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta) = Z_H(\zeta) \cdot t(\zeta)$$



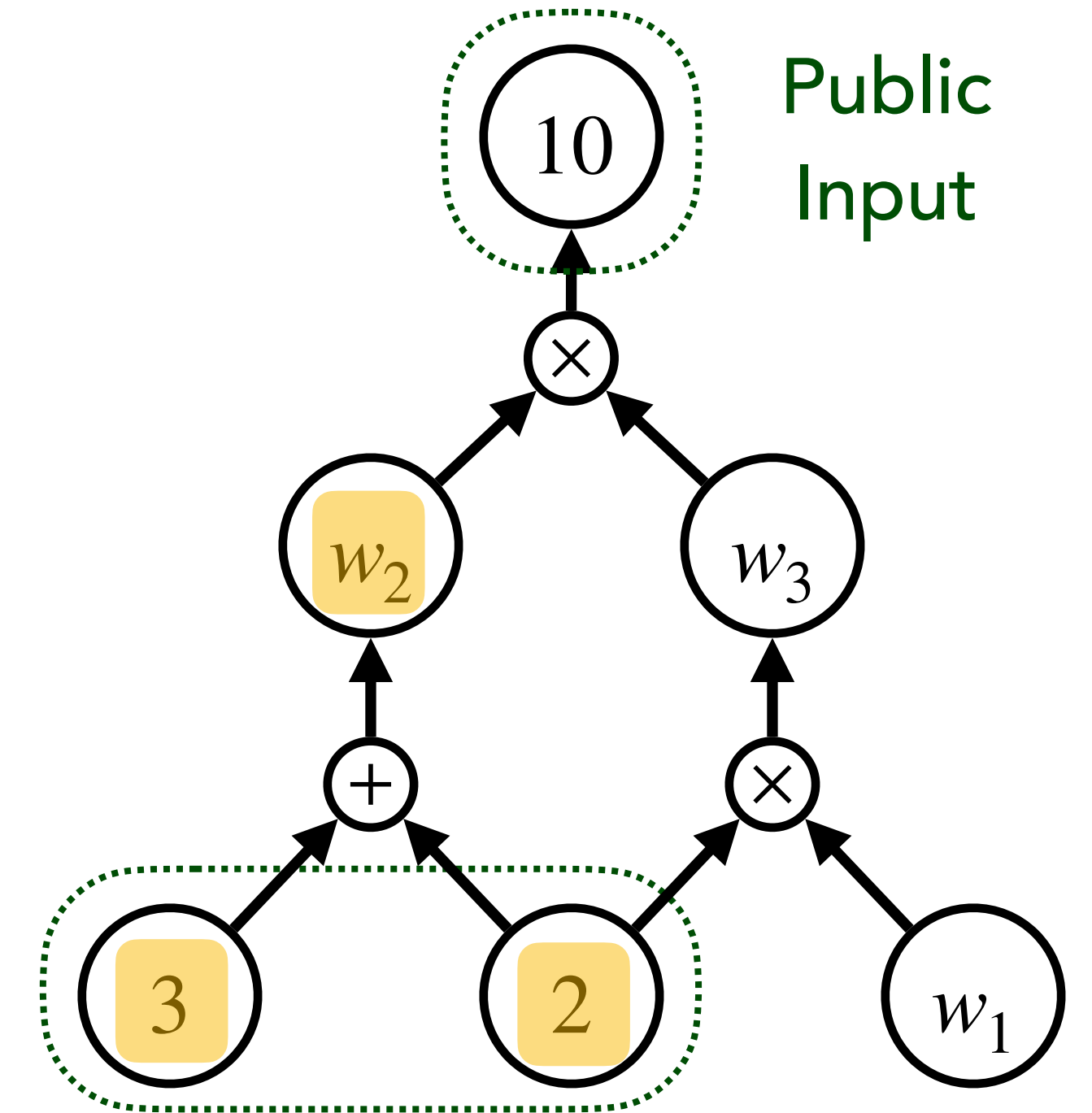
# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$

## Verification Equation:

$$\underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = Z_H(\zeta) \cdot t(\zeta)$$





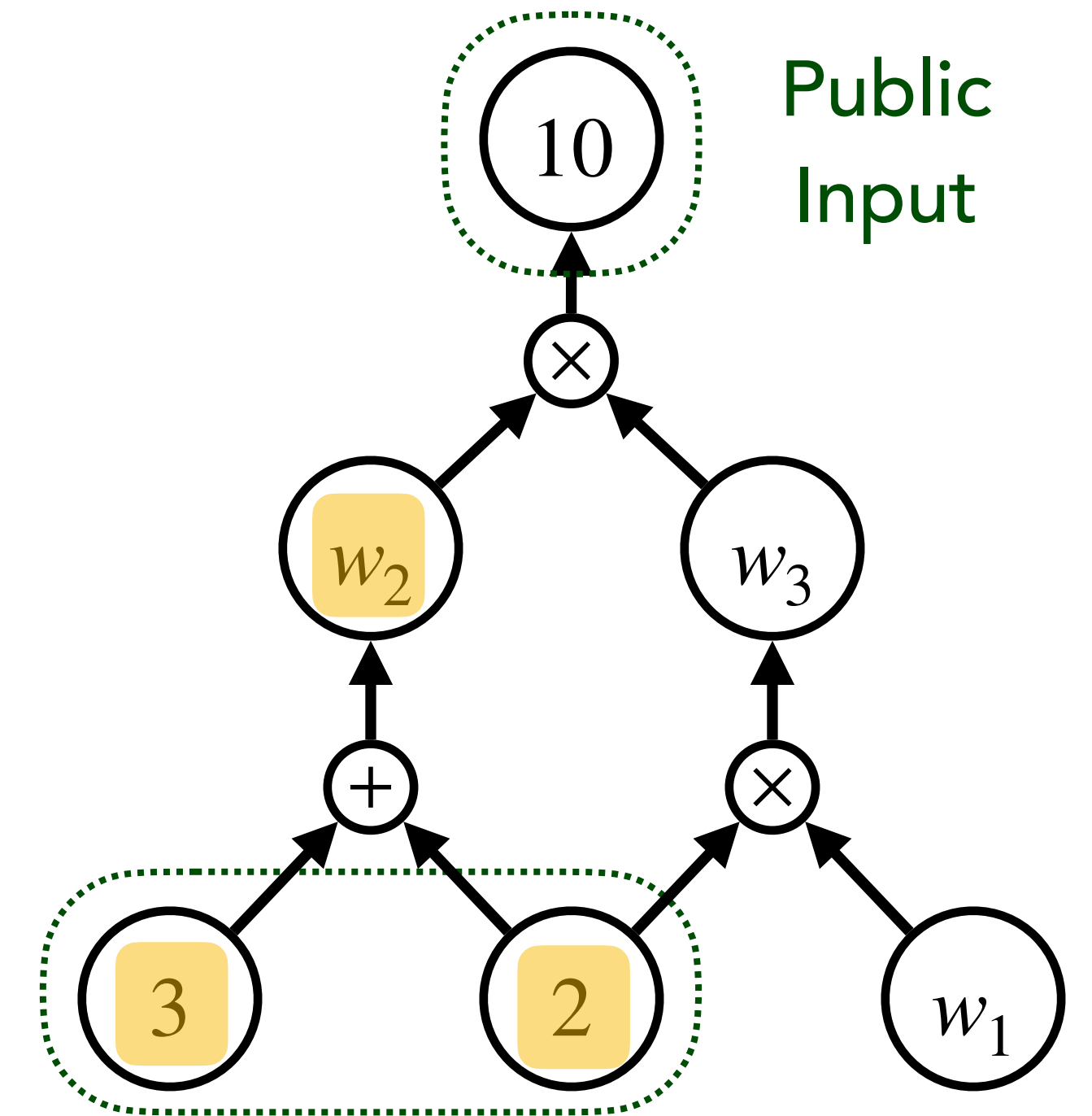
# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$

## Verification Equation:

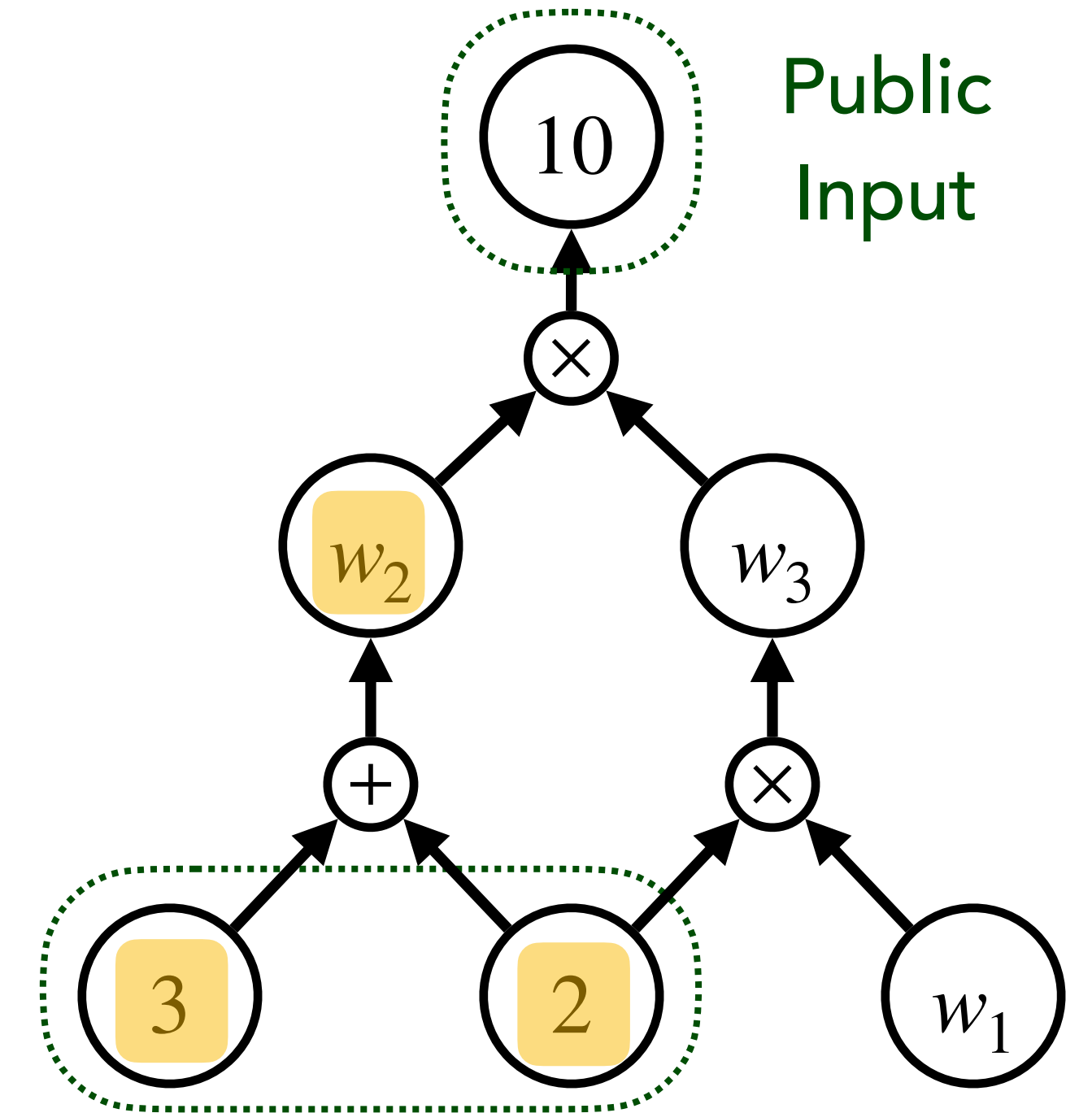
$$\underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$



# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$



## Verification Equation:

$$\underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

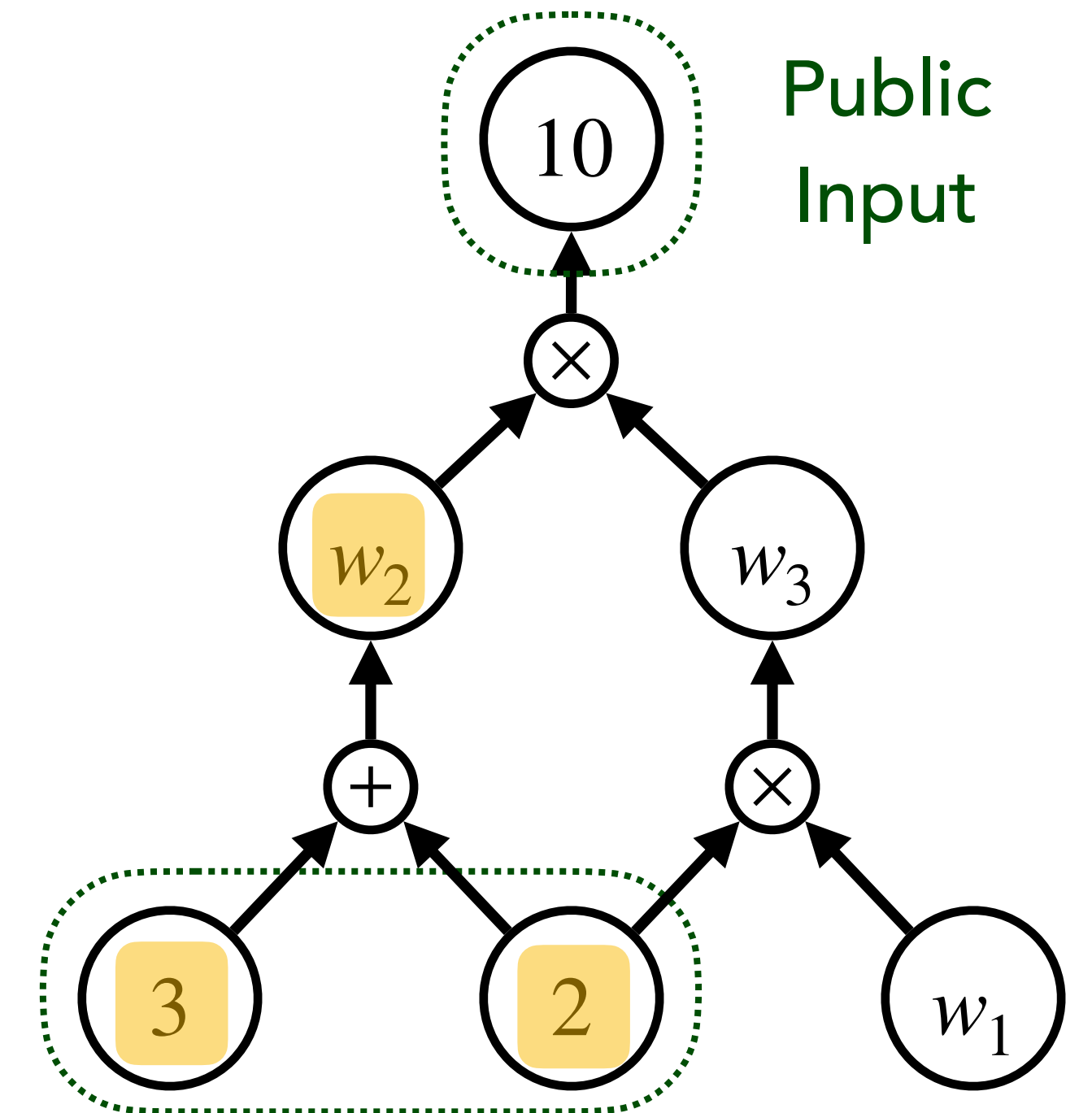
Batching Challenge

Evaluation Point

# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$



## Verification Equation:

$$\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta) + \underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

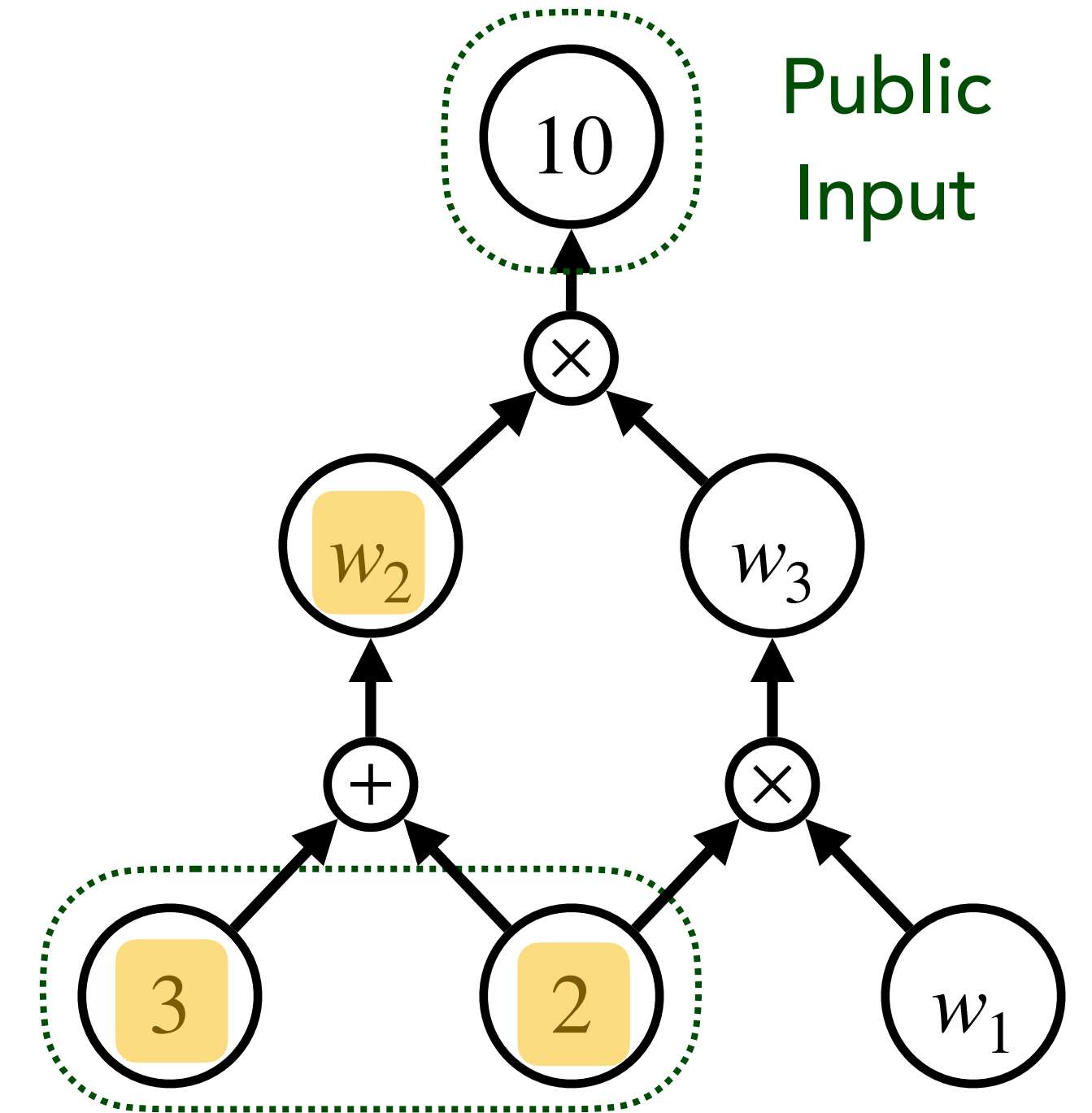
The verification equation is annotated with arrows and labels:
 

- A blue arrow points from the summation term  $\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta)$  to the  $\text{PI}(\zeta)$  term in the Gate Check.
- An orange arrow labeled "Batching Challenge" points to the  $\alpha$  term in the Consistency Check.
- An orange arrow labeled "Evaluation Point" points to the  $\zeta$  term in the Vanishing Domain.

# Plonk - Protocol Description

## Constraint System:

- **Gate Vectors:**  $\vec{a} = (3, 2, w_2)$ ,  $\vec{b} = (2, w_1, w_3)$ ,  $\vec{c} = (w_2, w_3, 10)$
- **Gate Constraints:**  $\vec{a}_1 + \vec{b}_1 = \vec{c}_1$ ,  $\vec{a}_2 \times \vec{b}_2 = \vec{c}_2$ ,  $\vec{a}_3 \times \vec{b}_3 = \vec{c}_3$
- **Consistency Constraints:**  $\vec{a}_2 = \vec{b}_1$ ,  $\vec{a}_3 = \vec{c}_1$ ,  $\vec{b}_3 = \vec{c}_2$



## Verification Equation:

$$\underbrace{\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta)}_{\text{(Fixed) Scalars}} + \underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

The diagram includes several annotations:
 

- An orange arrow points from the label "(Fixed) Scalars" to the sum  $\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta)$ .
- A blue arrow points from the label "Gate Check" to the term  $\text{PI}(\zeta) + \text{Eq}(\zeta)$ .
- An orange arrow points from the label "Batching Challenge" to the term  $\alpha \cdot \text{Per}(\zeta)$ .
- An orange arrow points from the label "Evaluation Point" to the term  $t(\zeta)$ .

# Plonk - Weak Fiat-Shamir Attack

Verification Equation:

$$\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta) + \underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars

# Plonk - Weak Fiat-Shamir Attack

Verification Equation:

$$\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta) + \underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars      Gate Check      Consistency Check      Vanishing Domain

Weak F-S Attack: When PI is not part of hash computation (for deriving  $\alpha, \zeta$ )

# Plonk - Weak Fiat-Shamir Attack

Verification Equation:

$$\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta) + \underbrace{\text{PI}(\zeta) + \text{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars      Gate Check      Consistency Check      Vanishing Domain

**Weak F-S Attack:** When PI is not part of hash computation (for deriving  $\alpha, \zeta$ )

1. Select arbitrary polynomials for the proof  $\implies$  compute all evaluations except  $\text{PI}(\zeta)$ .

# Plonk - Weak Fiat-Shamir Attack

Verification Equation:

$$\sum_{i=1}^k PI_i \cdot L_i(\zeta) + \underbrace{PI(\zeta)}_{\text{Gate Check}} + \underbrace{Eq(\zeta) + \alpha \cdot Per(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars

**Weak F-S Attack:** When PI is not part of hash computation (for deriving  $\alpha, \zeta$ )

1. Select arbitrary polynomials for the proof  $\implies$  compute all evaluations except  $PI(\zeta)$ .



# Plonk - Weak Fiat-Shamir Attack

Verification Equation:

The diagram illustrates the verification equation for Plonk, showing the relationship between various components. On the left, a dotted box contains the sum  $\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta)$ . An orange arrow points from the label "(Fixed) Scalars" below to this sum. A blue arrow points from the sum to the  $\text{PI}(\zeta)$  term in the main equation. The main equation is  $\text{PI}(\zeta) + \text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta) = Z_H(\zeta) \cdot t(\zeta)$ . Brackets below the equation identify the terms: "Gate Check" for  $\text{PI}(\zeta)$ , "Consistency Check" for  $\text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)$ , and "Vanishing Domain" for  $Z_H(\zeta) \cdot t(\zeta)$ . A yellow dotted box highlights the right-hand side of the equation.

$$\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta) = \underbrace{\text{PI}(\zeta)}_{\text{Gate Check}} + \underbrace{\text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars

**Weak F-S Attack:** When PI is not part of hash computation (for deriving  $\alpha, \zeta$ )

1. Select *arbitrary* polynomials for the proof  $\implies$  compute *all* evaluations except  $\text{PI}(\zeta)$ .
2. Solve for the public values  $\text{PI} = (\text{PI}_1, \dots, \text{PI}_k)$  that will pass verification.

# Plonk - Weak Fiat-Shamir Attack

## Verification Equation:

The diagram illustrates the verification equation for Plonk, showing the relationship between various components. On the left, the sum  $\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta)$  is enclosed in a dotted box, with an orange arrow pointing to it from the label "(Fixed) Scalars". A blue arrow labeled "Linear Equation" points from this sum to the right-hand side of the equation. The right-hand side is  $\text{PI}(\zeta) + \text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta) = Z_H(\zeta) \cdot t(\zeta)$ . Brackets below the equation identify the terms: "Gate Check" under  $\text{PI}(\zeta)$ , "Consistency Check" under the entire left-hand side, and "Vanishing Domain" under  $Z_H(\zeta) \cdot t(\zeta)$ . A yellow dashed box highlights the terms  $\text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)$ .

$$\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta) = \underbrace{\text{PI}(\zeta)}_{\text{Gate Check}} + \underbrace{\text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars

**Weak F-S Attack:** When PI is not part of hash computation (for deriving  $\alpha, \zeta$ )

1. Select *arbitrary* polynomials for the proof  $\implies$  compute *all* evaluations except  $\text{PI}(\zeta)$ .
2. Solve for the public values  $\text{PI} = (\text{PI}_1, \dots, \text{PI}_k)$  that will pass verification.

# Plonk - Weak Fiat-Shamir Attack

## Verification Equation:

The diagram illustrates the verification equation for Plonk, showing the relationship between various components. On the left, a sum of products  $\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta)$  is enclosed in a dotted box, with an orange arrow pointing to it from the label "(Fixed) Scalars". A blue arrow labeled "Linear Equation" points from this sum to the right-hand side of the equation. The right-hand side is a large yellow rounded rectangle containing the equation:  $\text{PI}(\zeta) + \text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta) = Z_H(\zeta) \cdot t(\zeta)$ . Brackets below the equation identify the terms: "Gate Check" under  $\text{PI}(\zeta)$ , "Consistency Check" under  $\text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)$ , and "Vanishing Domain" under  $Z_H(\zeta) \cdot t(\zeta)$ .

$$\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta) = \underbrace{\text{PI}(\zeta)}_{\text{Gate Check}} + \underbrace{\text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars      Gate Check      Consistency Check      Vanishing Domain

**Weak F-S Attack:** When PI is not part of hash computation (for deriving  $\alpha, \zeta$ )

1. Select *arbitrary* polynomials for the proof  $\implies$  compute *all* evaluations except  $\text{PI}(\zeta)$ .
2. Solve for the public values  $\text{PI} = (\text{PI}_1, \dots, \text{PI}_k)$  that will pass verification.

**In Contrast:** For **strong** Fiat-Shamir, changing PI will also change  $\alpha, \zeta$ .

# Plonk - Weak Fiat-Shamir Attack

## Verification Equation:

The diagram illustrates the verification equation for Plonk, showing the relationship between various components. On the left, a sum of products  $\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta)$  is enclosed in a dotted box, with an orange arrow pointing to it from the label "(Fixed) Scalars". A blue arrow labeled "Linear Equation" points from this sum to the right-hand side of the equation. The right-hand side is a large yellow rounded rectangle containing the equation:  $\text{PI}(\zeta) + \text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta) = Z_H(\zeta) \cdot t(\zeta)$ . Brackets below the equation identify the terms: "Gate Check" under  $\text{PI}(\zeta)$ , "Consistency Check" under the entire left-hand side, and "Vanishing Domain" under  $Z_H(\zeta) \cdot t(\zeta)$ .

$$\sum_{i=1}^k \text{PI}_i \cdot L_i(\zeta) = \underbrace{\text{PI}(\zeta)}_{\text{Gate Check}} + \underbrace{\text{Eq}(\zeta) + \alpha \cdot \text{Per}(\zeta) + \alpha^2 \cdot (z(\zeta) - 1)L_1(\zeta)}_{\text{Consistency Check}} = \underbrace{Z_H(\zeta) \cdot t(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars      Gate Check      Consistency Check      Vanishing Domain

**Weak F-S Attack:** When PI is not part of hash computation (for deriving  $\alpha, \zeta$ )

1. Select *arbitrary* polynomials for the proof  $\implies$  compute *all* evaluations except  $\text{PI}(\zeta)$ .
2. Solve for the public values  $\text{PI} = (\text{PI}_1, \dots, \text{PI}_k)$  that will pass verification.

**Degrees of freedom:** can set *all but one*  $\text{PI}_i$  to be arbitrary.

**In Contrast:** For **strong** Fiat-Shamir, changing PI will also change  $\alpha, \zeta$ .

# Bulletproofs - Protocol Description

# Bulletproofs - Protocol Description

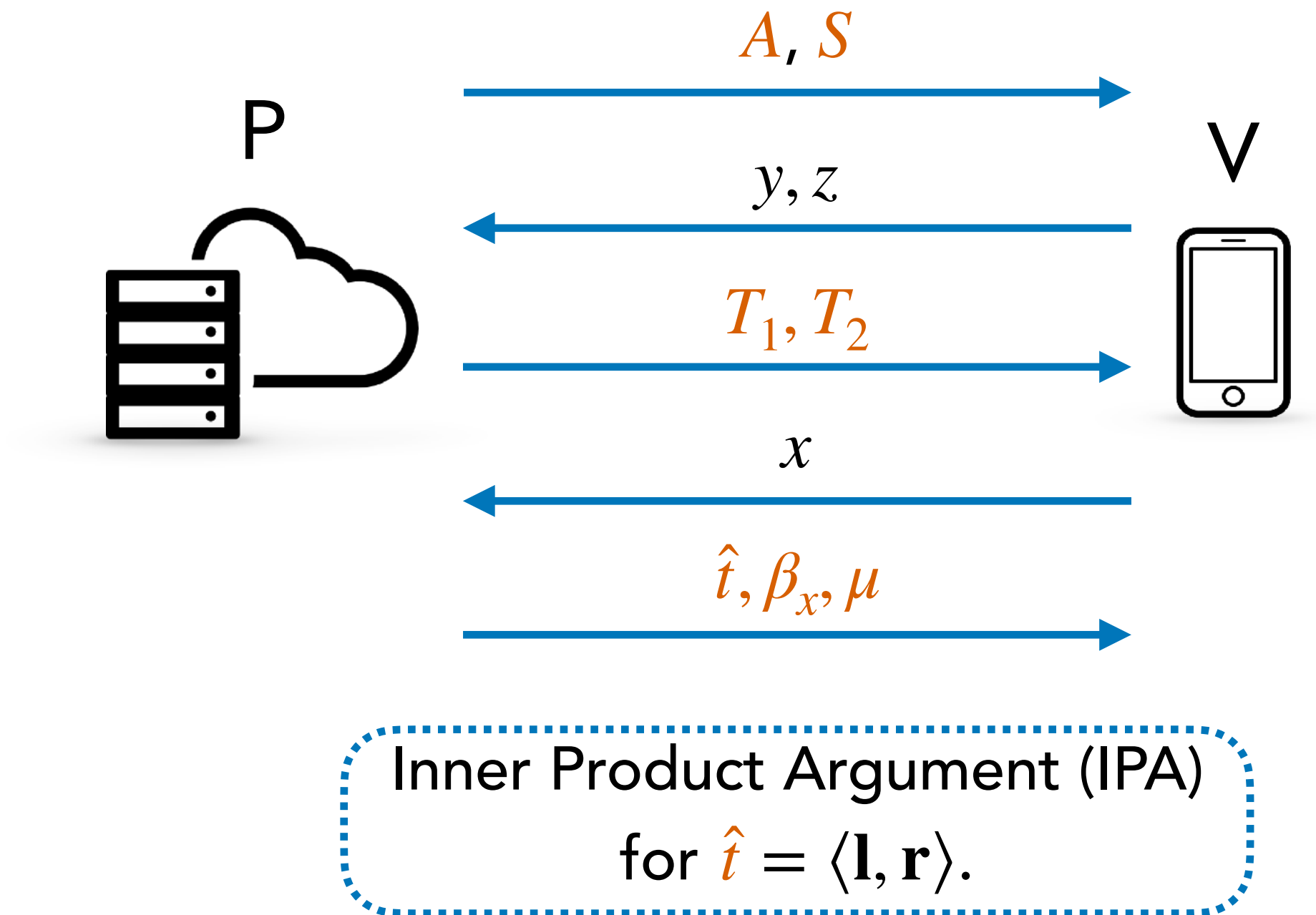
## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

# Bulletproofs - Protocol Description

## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$



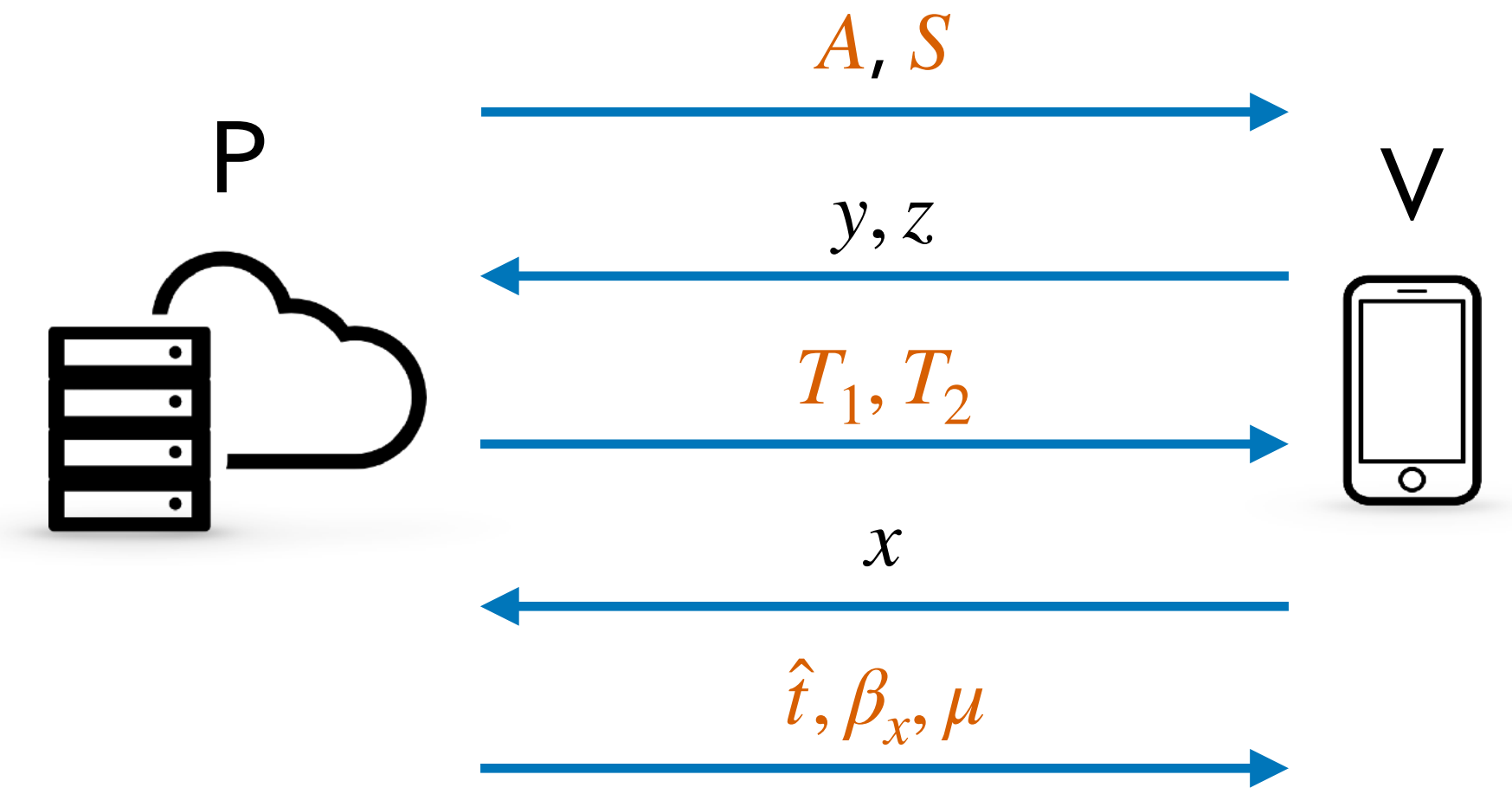
# Bulletproofs - Protocol Description

## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

**Note:**  $T_1 = g^{t_1} h^{\beta_1}, T_2 = g^{t_2} h^{\beta_2}$  in an honest proof

(with  $t_1, \beta_1, t_2, \beta_2$  known by P)



Inner Product Argument (IPA)  
for  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .



# Bulletproofs - Protocol Description

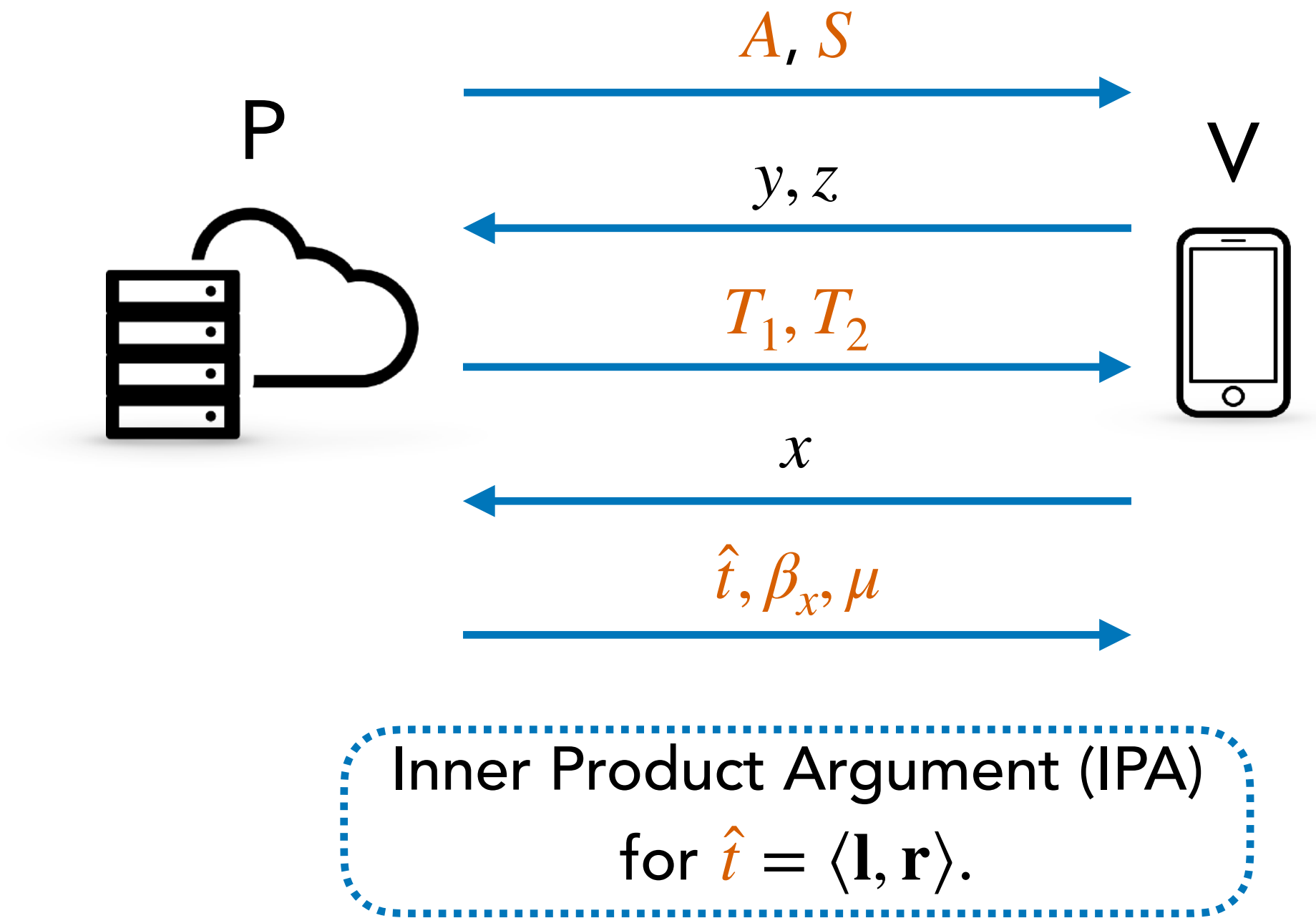
## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

**Note:**  $T_1 = g^{t_1} h^{\beta_1}, T_2 = g^{t_2} h^{\beta_2}$  in an honest proof

(with  $t_1, \beta_1, t_2, \beta_2$  known by P)

## Verification:



# Bulletproofs - Protocol Description

## Aggregate Range Proof Relation:

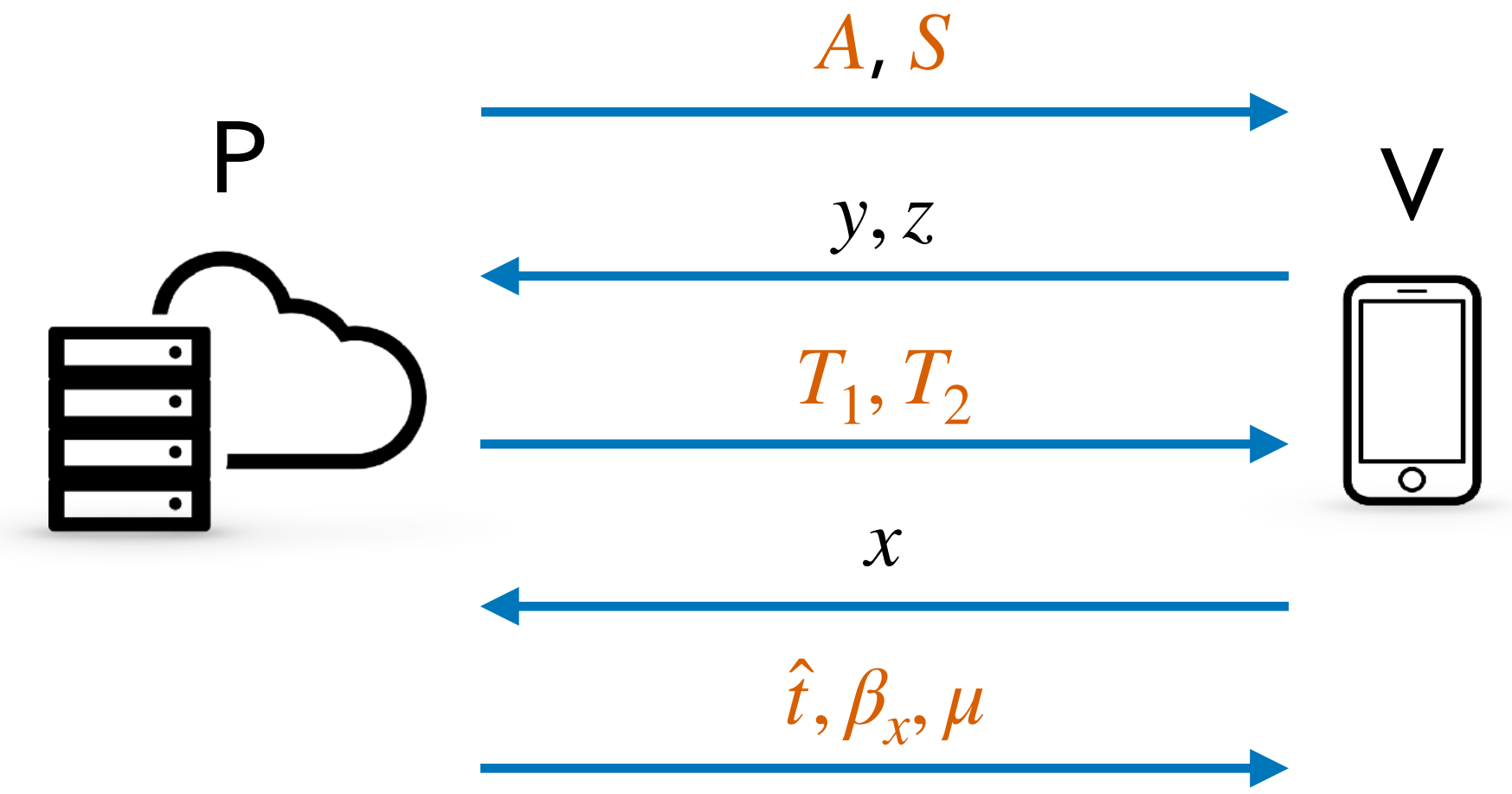
- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

**Note:**  $T_1 = g^{t_1} h^{\beta_1}, T_2 = g^{t_2} h^{\beta_2}$  in an honest proof

(with  $t_1, \beta_1, t_2, \beta_2$  known by P)

## Verification:

$$V_1^{z^2} \cdot \dots \cdot V_m^{z^{m+1}} = g^{\hat{t}} \cdot h^{\beta_x} \cdot g^{-\delta(y,z)} \cdot T_1^{-x} \cdot T_2^{-x^2} \quad \text{(along with IPA check)}$$



Inner Product Argument (IPA)  
for  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .

# Bulletproofs - Protocol Description

## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

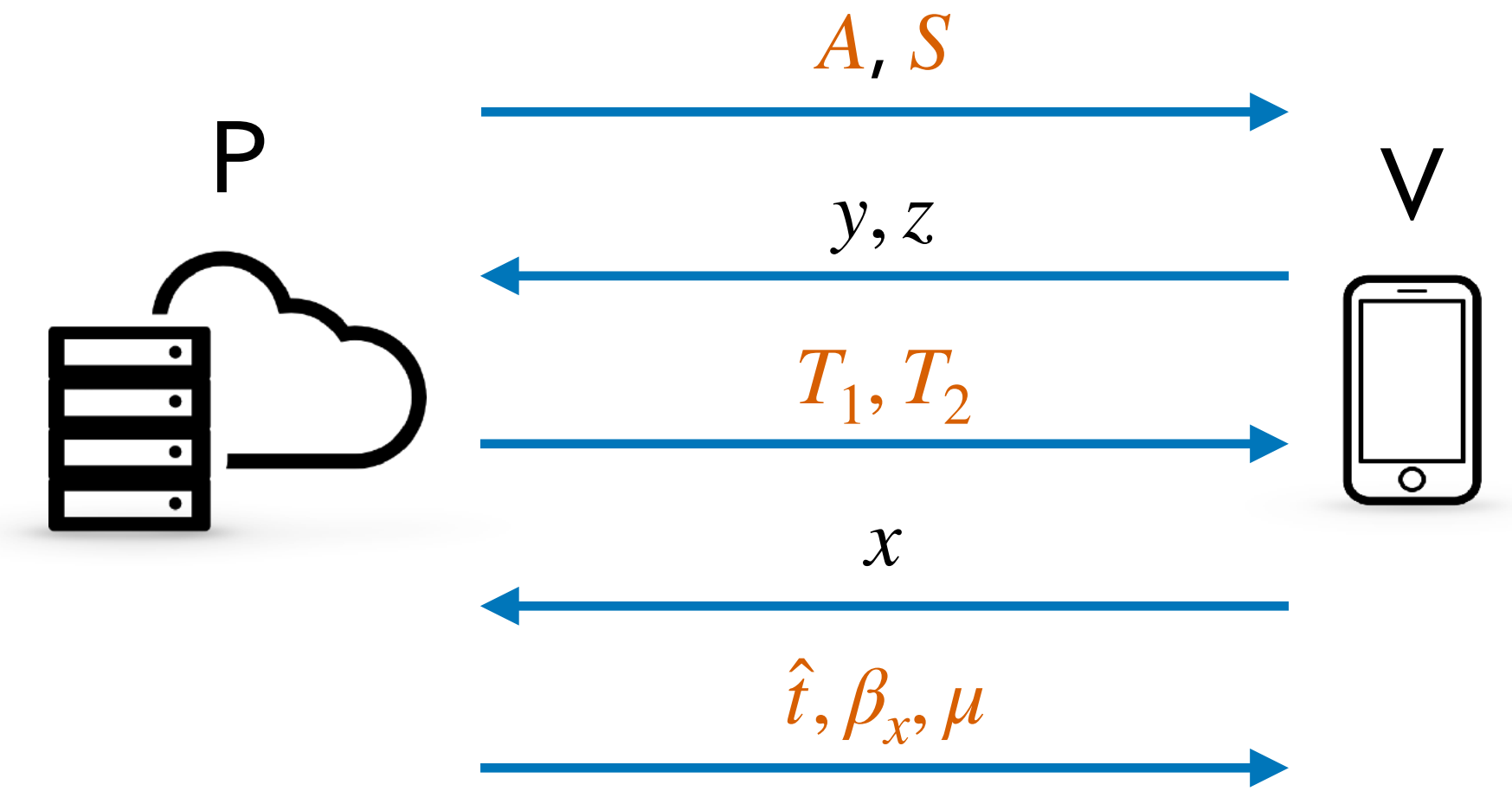
**Note:**  $T_1 = g^{t_1} h^{\beta_1}, T_2 = g^{t_2} h^{\beta_2}$  in an honest proof

(with  $t_1, \beta_1, t_2, \beta_2$  known by P)

## Verification:

$$V_1^{z^2} \cdot \dots \cdot V_m^{z^{m+1}} = g^{\hat{t}} \cdot h^{\beta_x} \cdot g^{-\delta(y,z)} \cdot T_1^{-x} \cdot T_2^{-x^2} \quad \text{(along with IPA check)}$$

↑↑



Inner Product Argument (IPA)  
for  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .

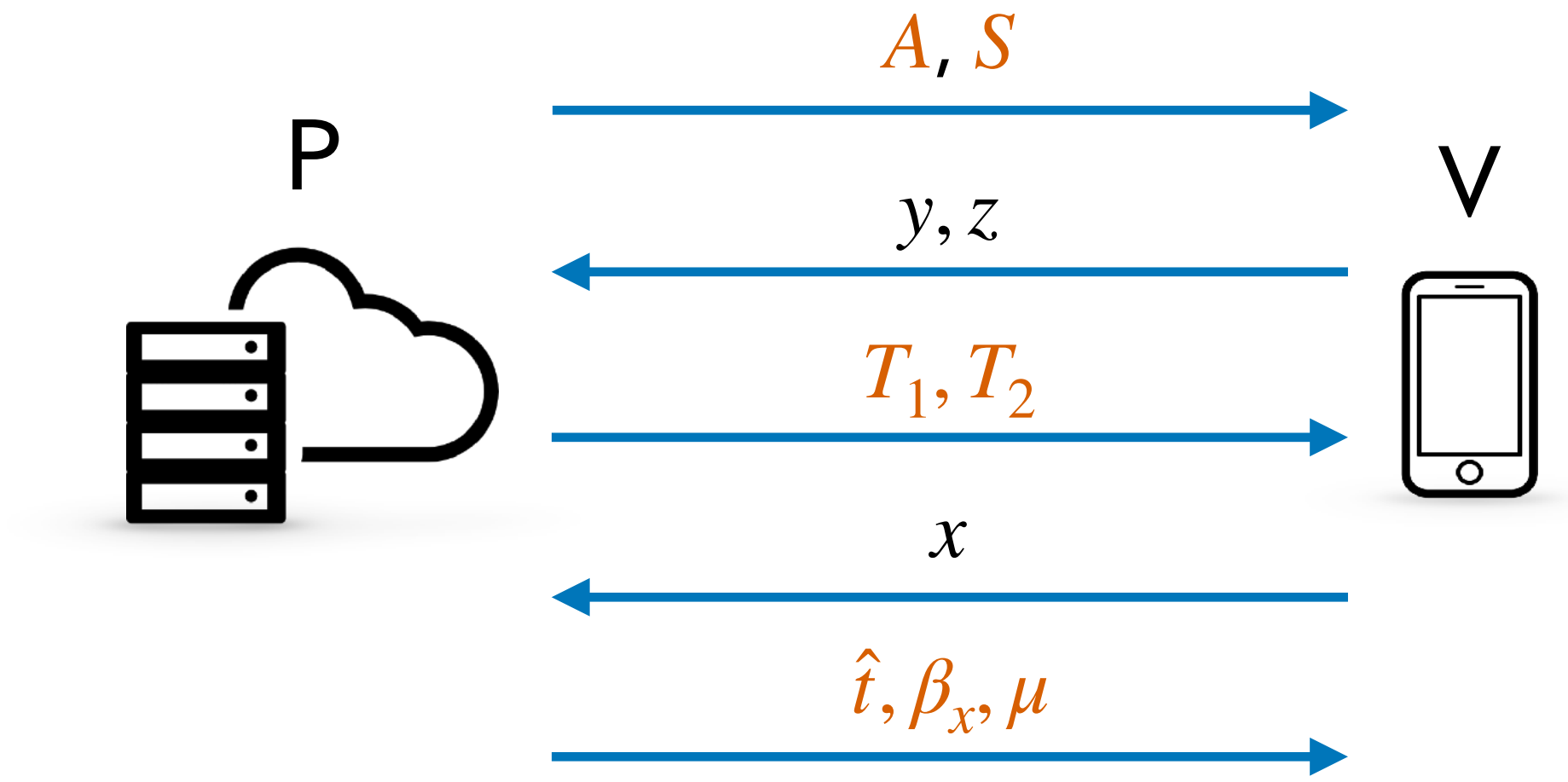
# Bulletproofs - Protocol Description

## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

**Note:**  $T_1 = g^{t_1} h^{\beta_1}, T_2 = g^{t_2} h^{\beta_2}$  in an honest proof

(with  $t_1, \beta_1, t_2, \beta_2$  known by P)



Inner Product Argument (IPA)  
for  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .

**Verification:**  $V_1 z^2 \cdot \dots \cdot V_m z^{m+1} = g^{\hat{t}} \cdot h^{\beta_x} \cdot g^{-\delta(y,z)} \cdot T_1^{-x} \cdot T_2^{-x^2}$  (along with IPA check)

$$\text{Exponents of } g, h : \begin{cases} v_1 z^2 + \dots + v_m z^{m+1} = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \dots + \gamma_m z^{m+1} = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

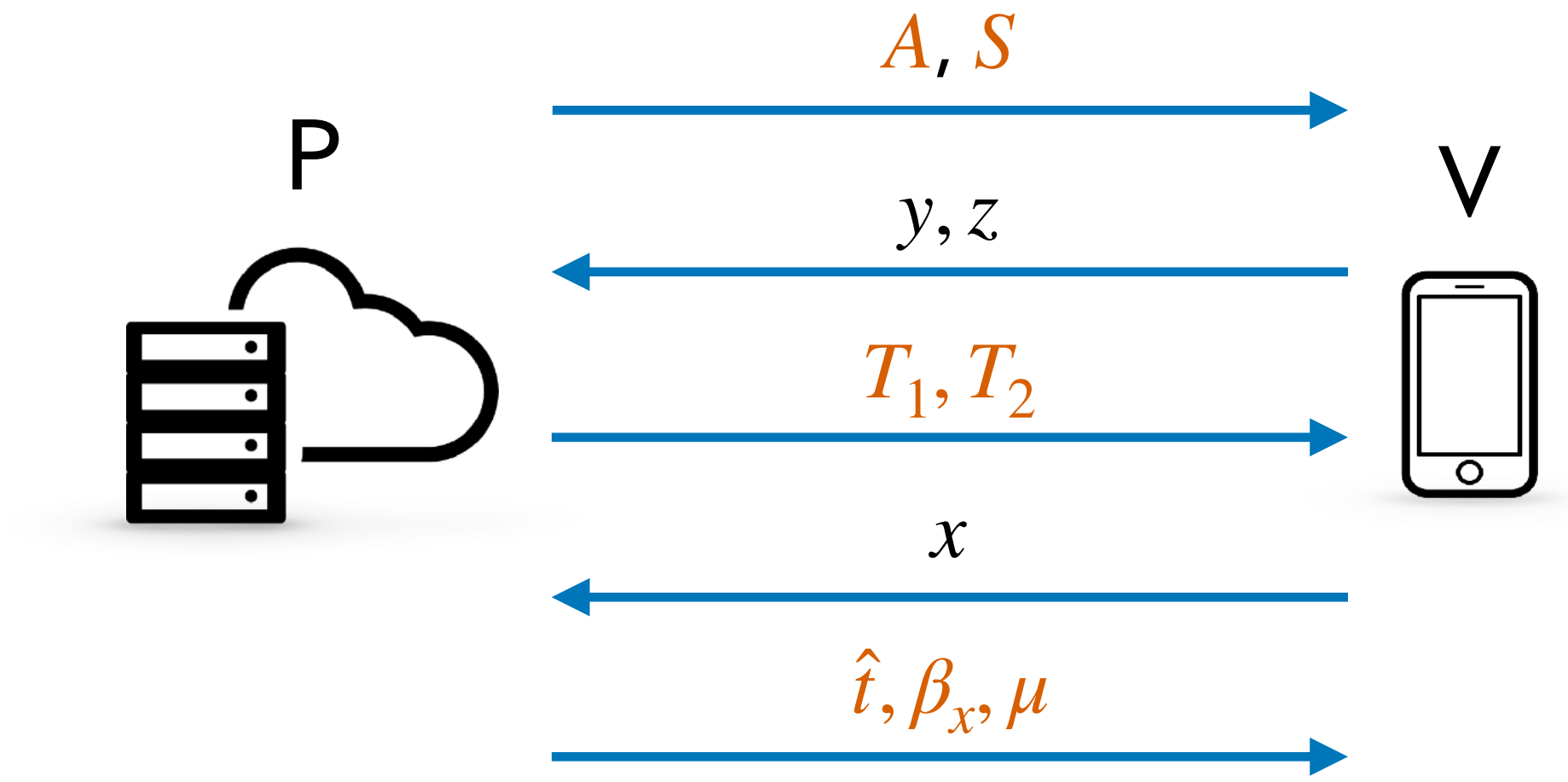
# Bulletproofs - Protocol Description

## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

**Note:**  $T_1 = g^{t_1} h^{\beta_1}, T_2 = g^{t_2} h^{\beta_2}$  in an honest proof

(with  $t_1, \beta_1, t_2, \beta_2$  known by P)



Inner Product Argument (IPA)  
for  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .

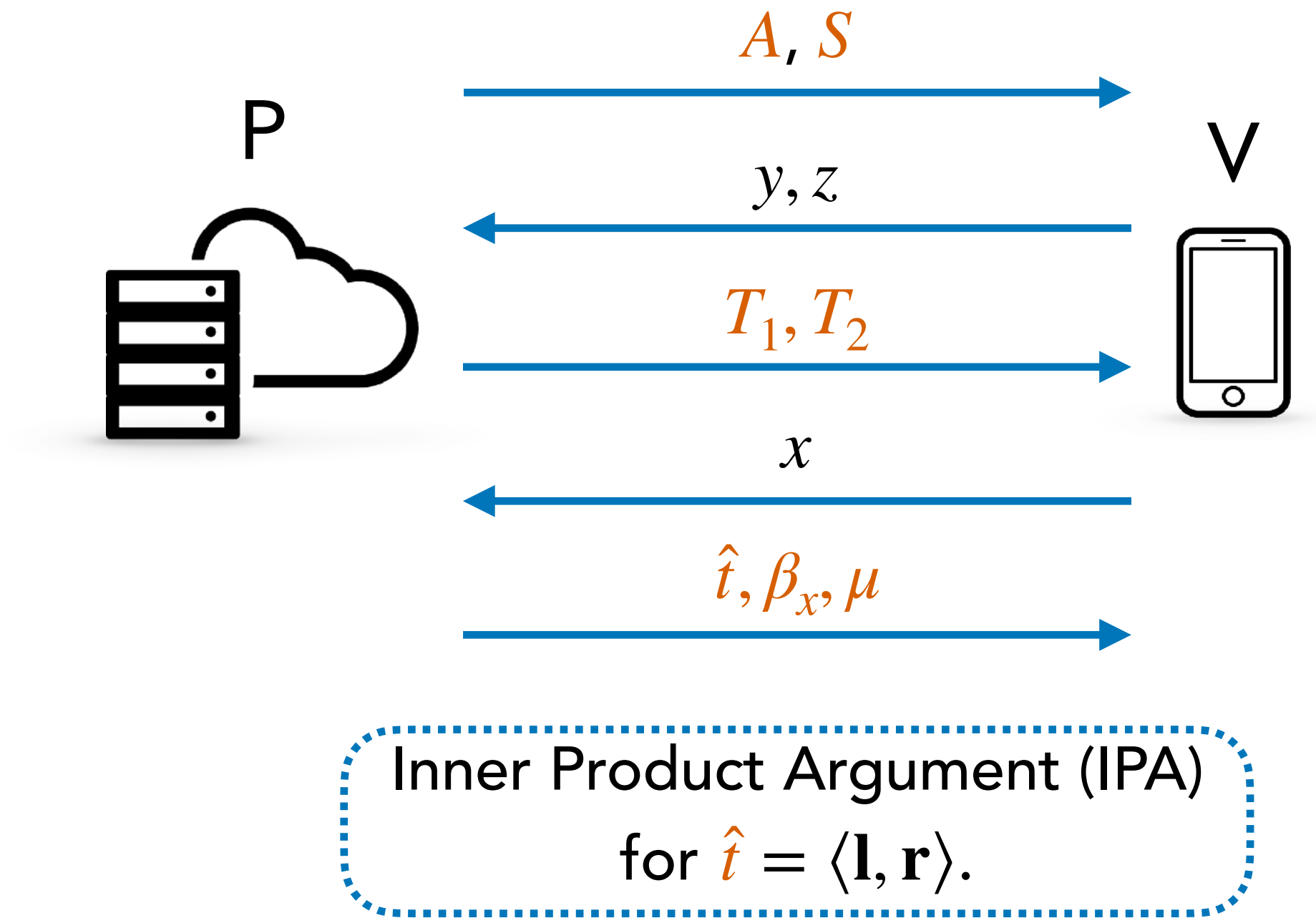
**Verification:**  $V_1 z^2 \cdot \dots \cdot V_m z^{m+1} = g^{\hat{t}} \cdot h^{\beta_x} \cdot g^{-\delta(y,z)} \cdot T_1^{-x} \cdot T_2^{-x^2}$  (along with IPA check)

$$\text{Exponents of } g, h : \begin{cases} v_1 z^2 + \dots + v_m z^{m+1} = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \dots + \gamma_m z^{m+1} = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

# Bulletproofs - Weak Fiat-Shamir Attack

## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$



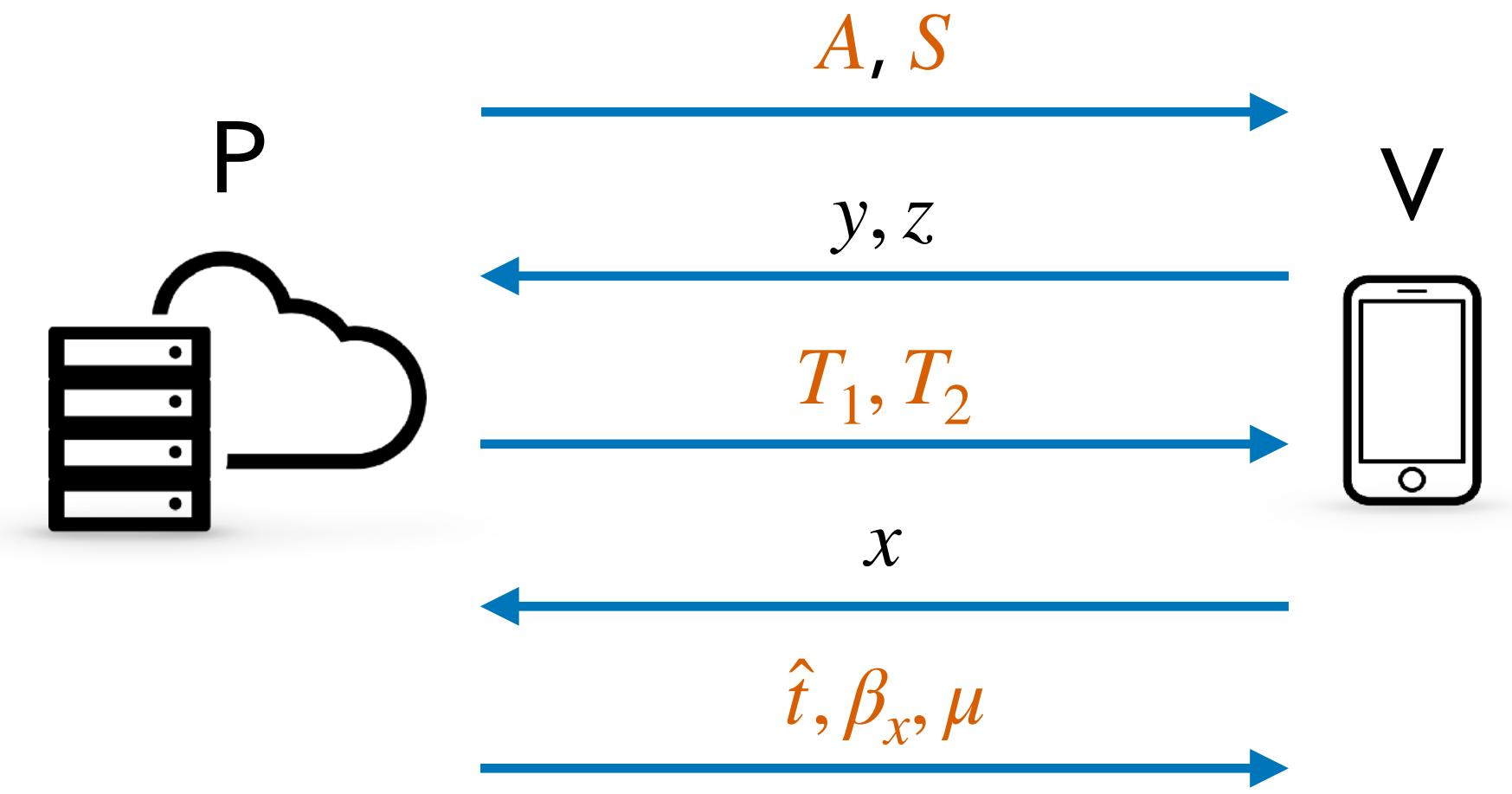
$$\begin{cases} v_1 z^2 + \dots + v_m z^{m+1} = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \dots + \gamma_m z^{m+1} = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

# Bulletproofs - Weak Fiat-Shamir Attack

## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

**Weak F-S Attack:** When  $V_1, \dots, V_m$  are not hashed



Inner Product Argument (IPA)  
for  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .

$$\begin{cases} v_1 z^2 + \dots + v_m z^{m+1} = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \dots + \gamma_m z^{m+1} = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

# Bulletproofs - Weak Fiat-Shamir Attack

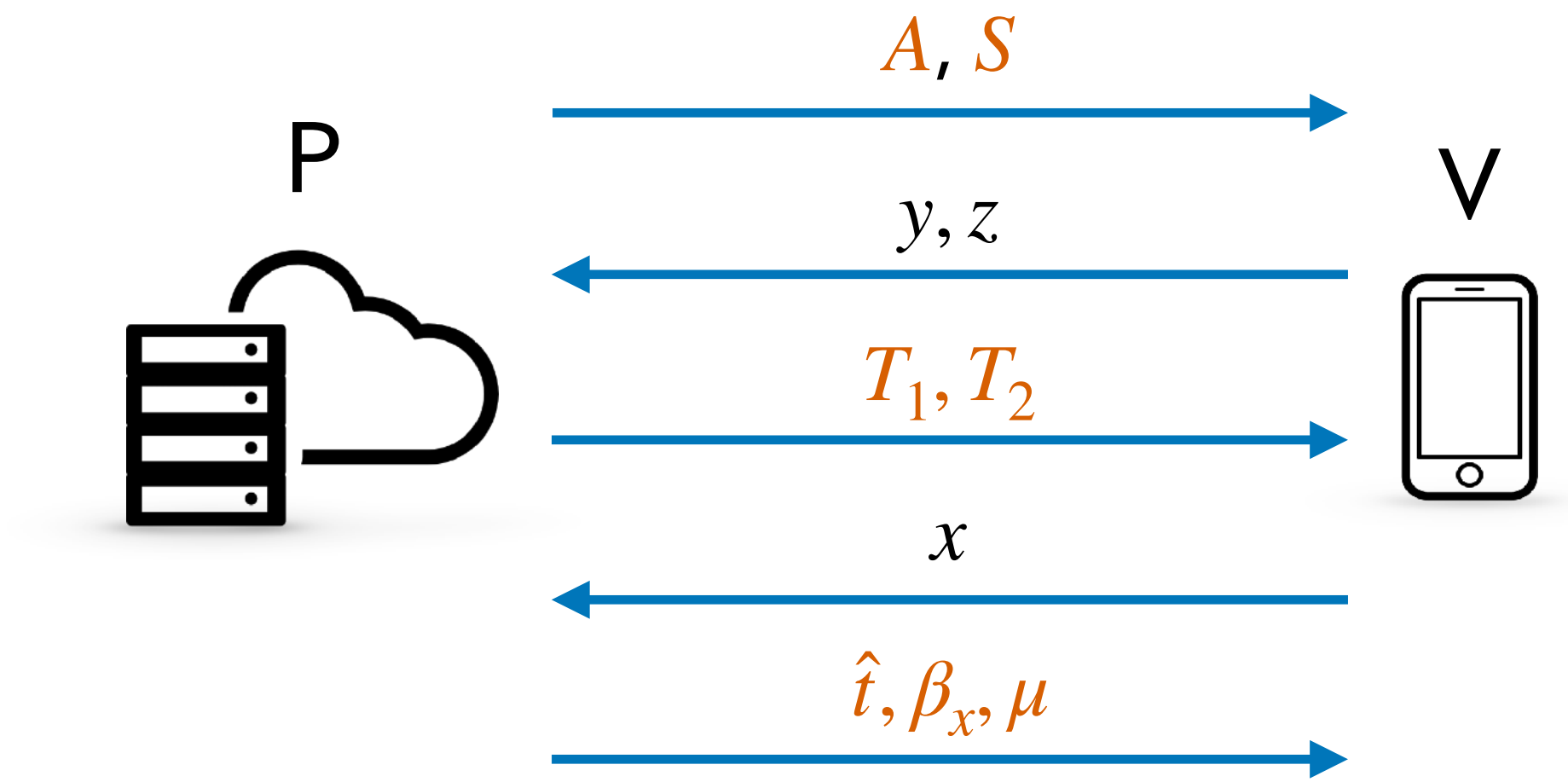
## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

**Weak F-S Attack:** When  $V_1, \dots, V_m$  are not hashed

1. Compute P's messages using an arbitrary witness:

- Set  $T_1 = g^{t_1} h^{\beta_1}, T_2 = g^{t_2} h^{\beta_2}$  for arbitrary  $t_1, t_2, \beta_1, \beta_2$ .



Inner Product Argument (IPA)  
for  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .

$$\begin{cases} v_1 z^2 + \dots + v_m z^{m+1} = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \dots + \gamma_m z^{m+1} = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$



# Bulletproofs - Weak Fiat-Shamir Attack

## Aggregate Range Proof Relation:

- $V_1 = g^{v_1} h^{\gamma_1}, \dots, V_m = g^{v_m} h^{\gamma_m}$
- $v_1, \dots, v_m \in [0, 2^n - 1]$

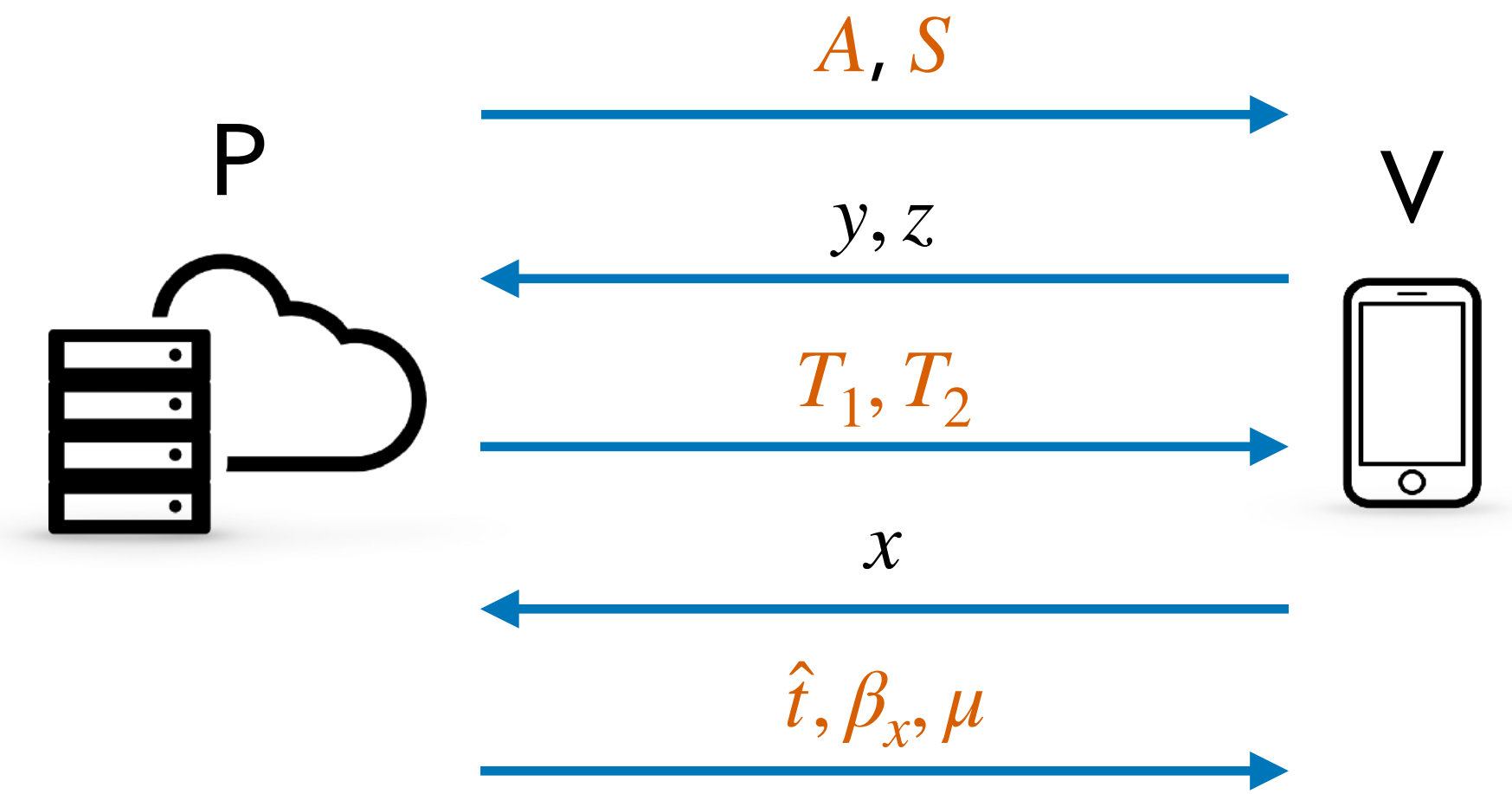
**Weak F-S Attack:** When  $V_1, \dots, V_m$  are not hashed

1. Compute P's messages using an arbitrary witness:

- Set  $T_1 = g^{t_1} h^{\beta_1}, T_2 = g^{t_2} h^{\beta_2}$  for arbitrary  $t_1, t_2, \beta_1, \beta_2$ .

2. Solve for  $v_1, \dots, v_m, \gamma_1, \dots, \gamma_m$  that satisfy (1).

$$\begin{cases} v_1 z^2 + \dots + v_m z^{m+1} = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \dots + \gamma_m z^{m+1} = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases} \quad (1)$$



Inner Product Argument (IPA)  
for  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .

# Weak Fiat-Shamir Attacks

## **Practical Impacts**

**(printing money on blockchains for fun and profit)**

# Weak Fiat-Shamir Attacks

## Practical Impacts

(printing money on blockchains for fun and profit)

We didn't do this!



# Weak Fiat-Shamir Attacks

## Practical Impacts

(printing money on blockchains for fun and profit)


We didn't do this!  
(but others might have...)



# Weak Fiat-Shamir Attacks

## Practical Impacts

(printing money on blockchains for fun and profit)



We didn't do this!  
(but others might have...  
...and we wouldn't know!)

# Case Study: Dusk Network

# Case Study: Dusk Network



Regulated **And** Decentralized Finance.

Market cap ⓘ

\$42,646,372

#375

(as of August 18, 2023)

# Case Study: Dusk Network

Transaction Model (simplified):



Regulated **And** Decentralized Finance.

Market cap ⓘ

\$42,646,372

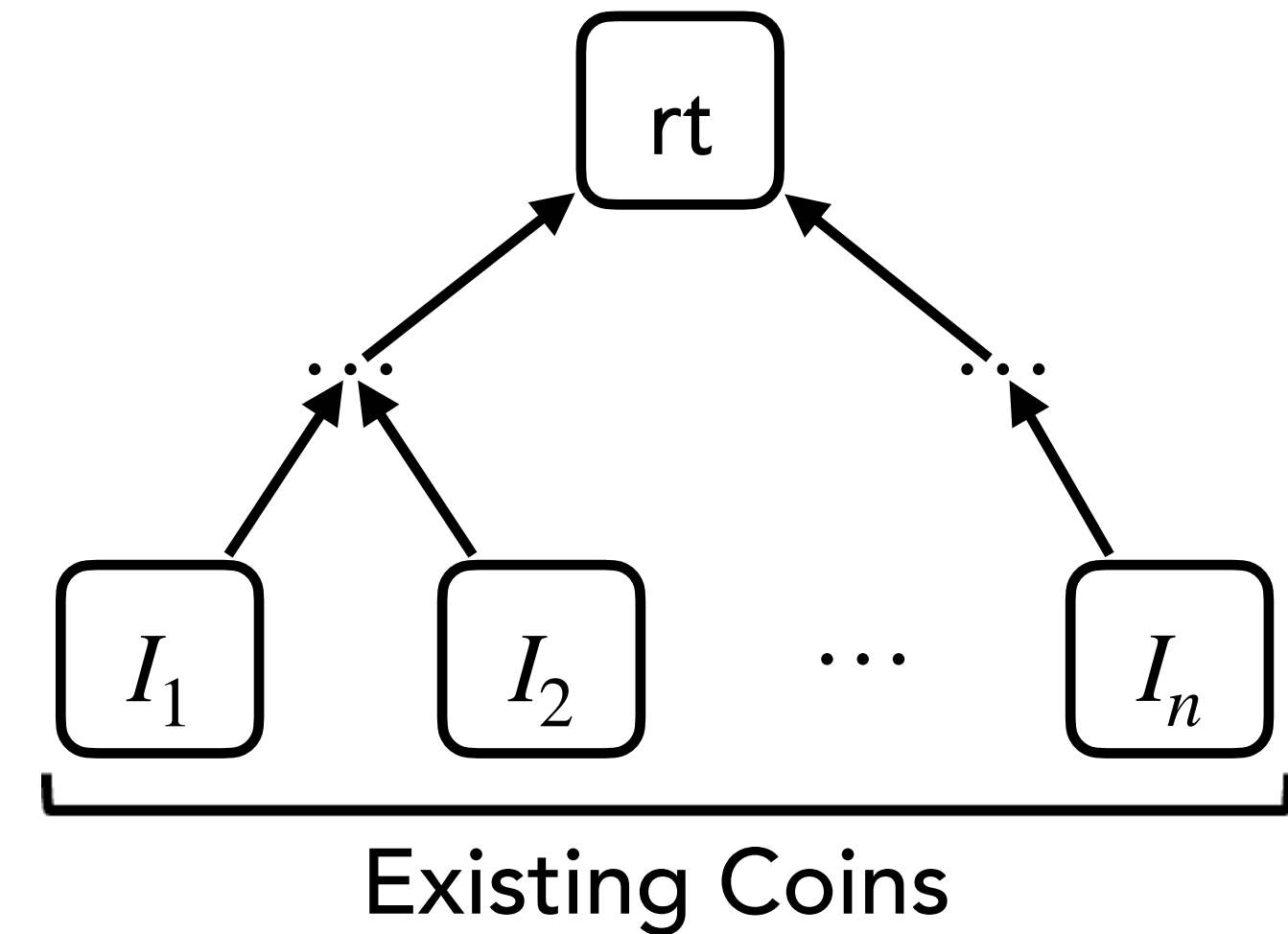
#375

(as of August 18, 2023)



# Case Study: Dusk Network

Transaction Model (simplified):



Market cap ⓘ

\$42,646,372

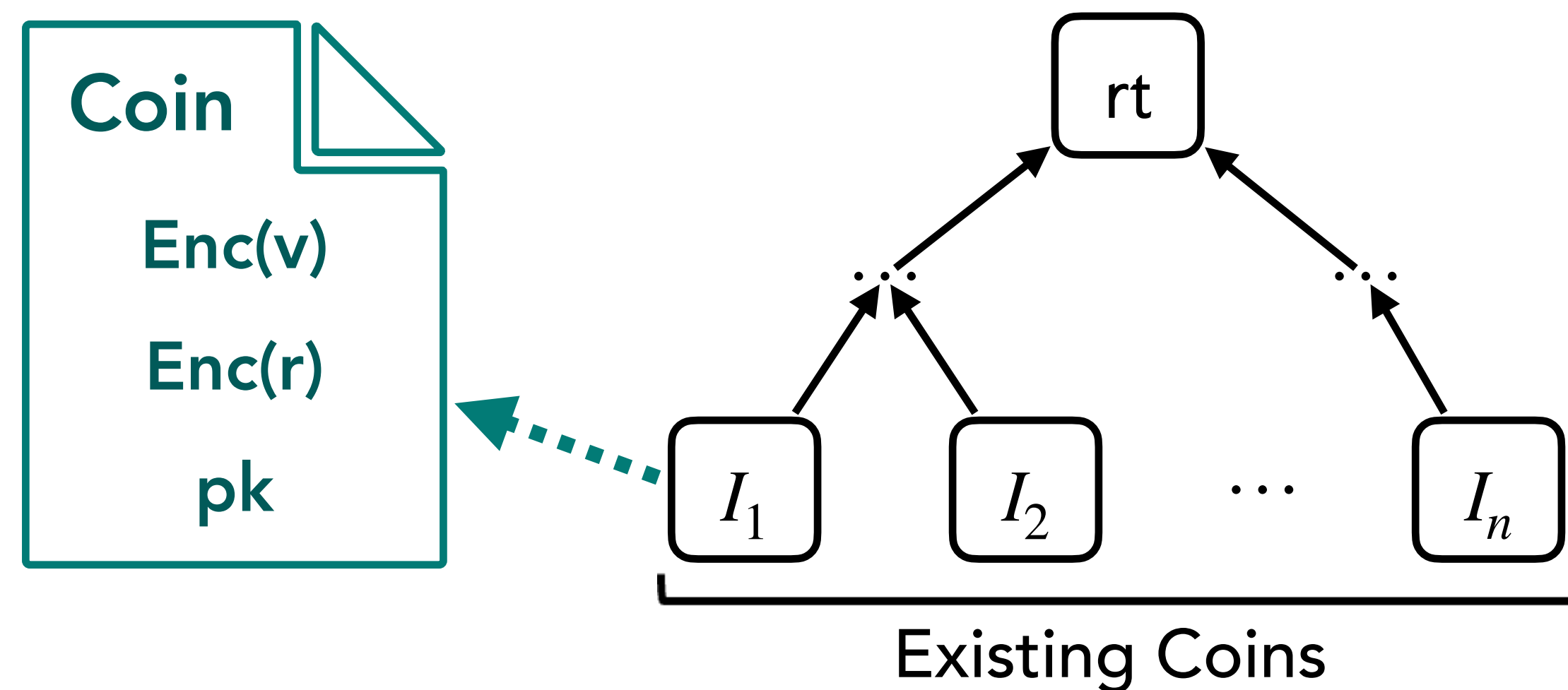
#375

Regulated **And** Decentralized Finance.

(as of August 18, 2023)

# Case Study: Dusk Network

Transaction Model (simplified):



Market cap ⓘ

\$42,646,372

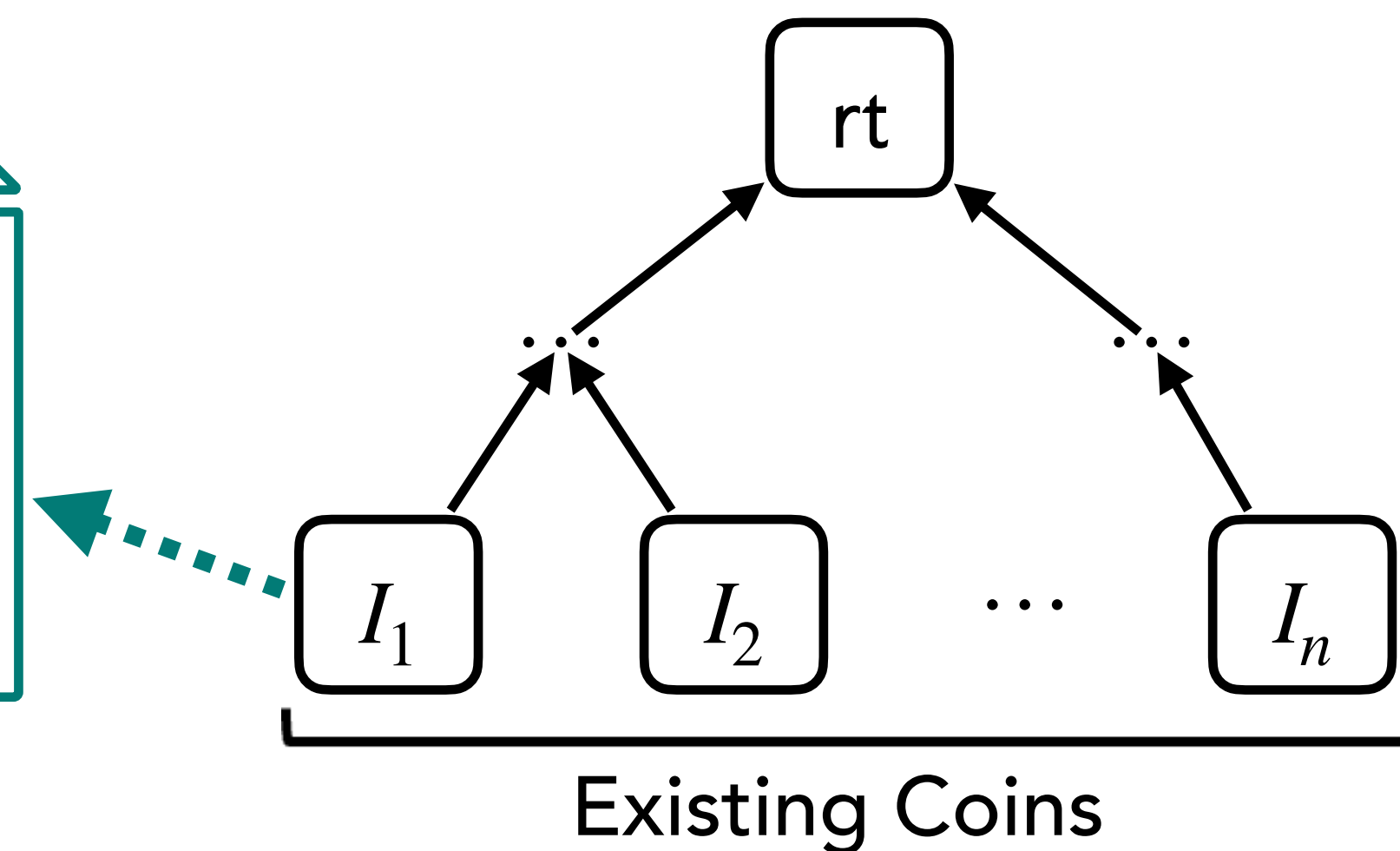
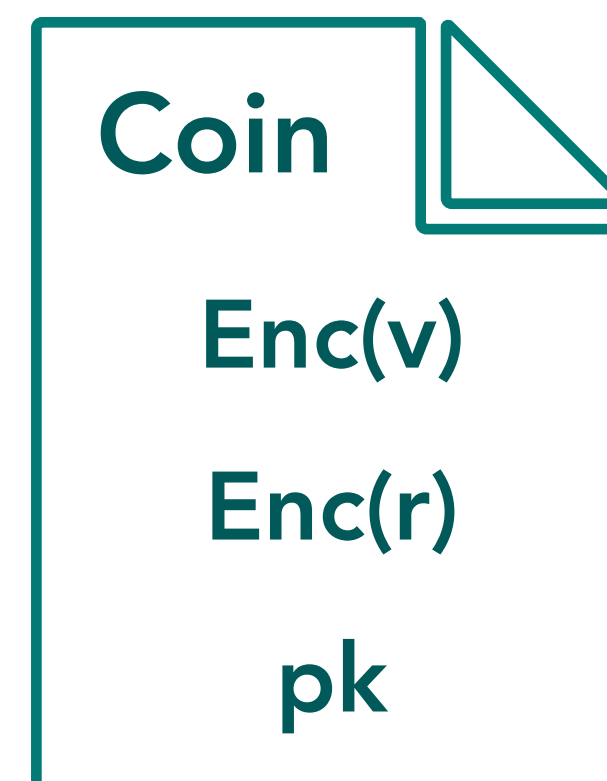
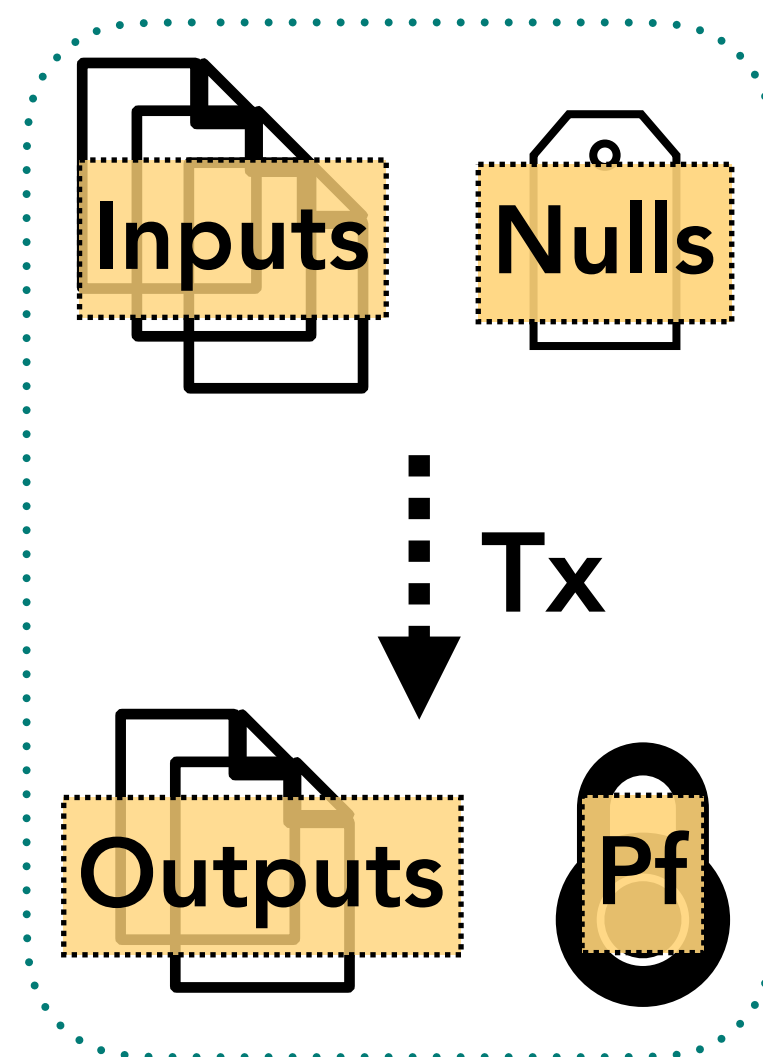
#375

Regulated **And** Decentralized Finance.

(as of August 18, 2023)

# Case Study: Dusk Network

Transaction Model (simplified):



Market cap ⓘ

\$42,646,372

#375

Regulated **And** Decentralized Finance.

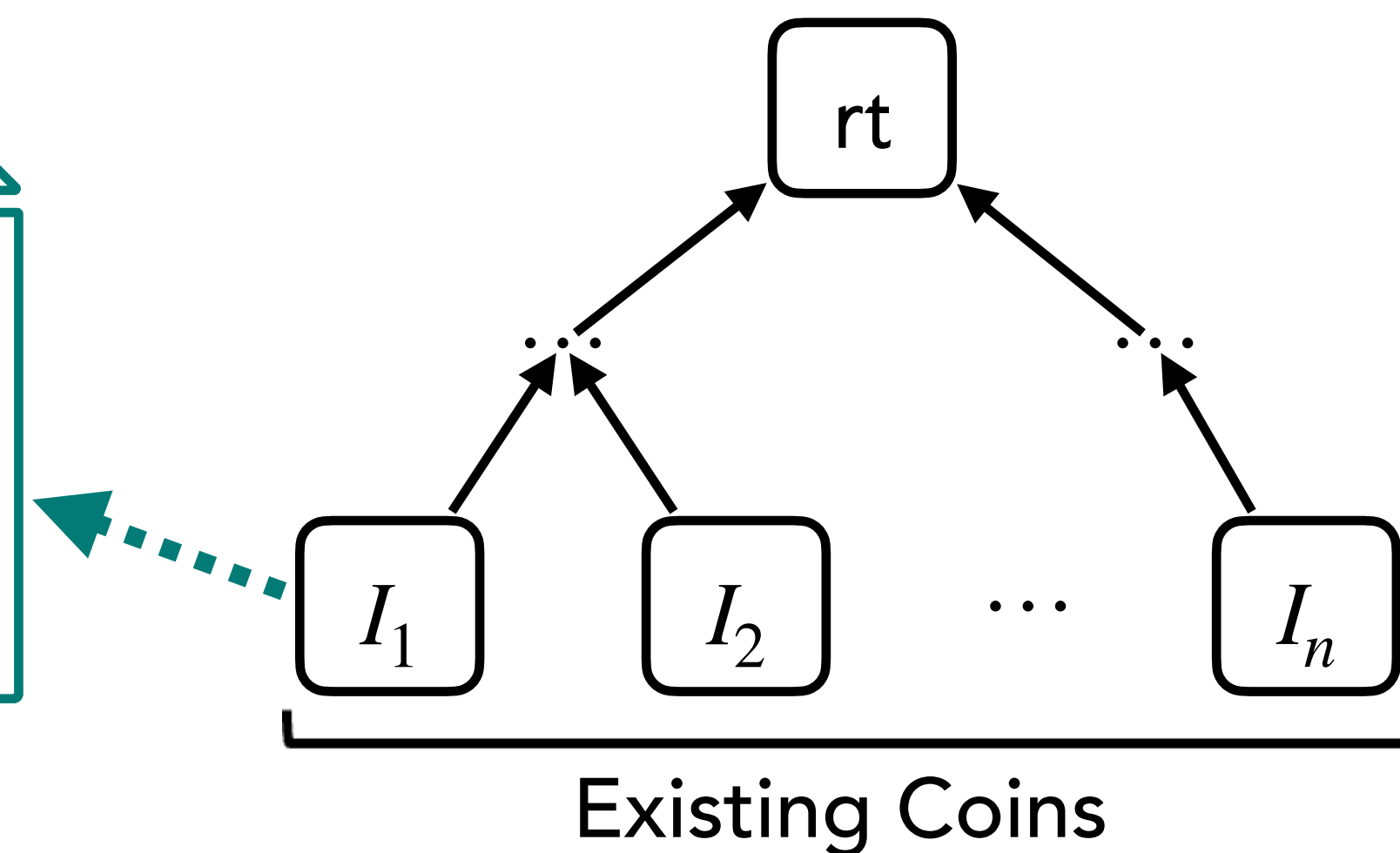
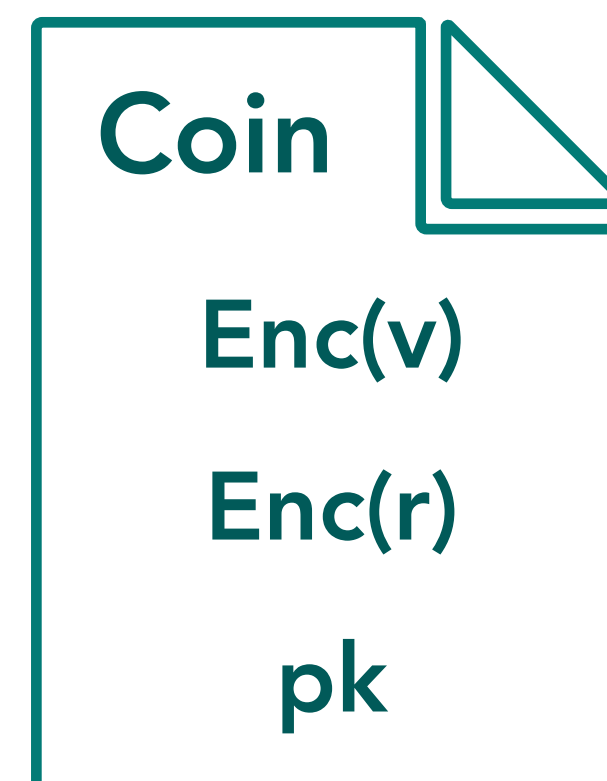
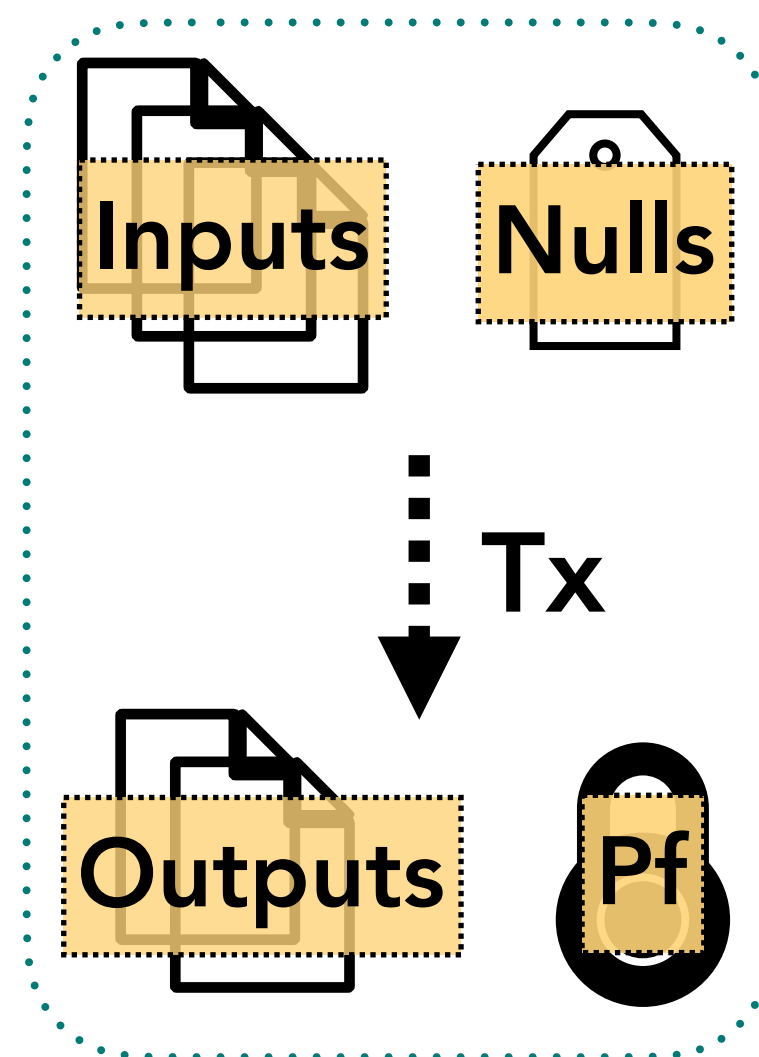
(as of August 18, 2023)

# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$



Market cap ⓘ

\$42,646,372

#375

Regulated **And** Decentralized Finance.

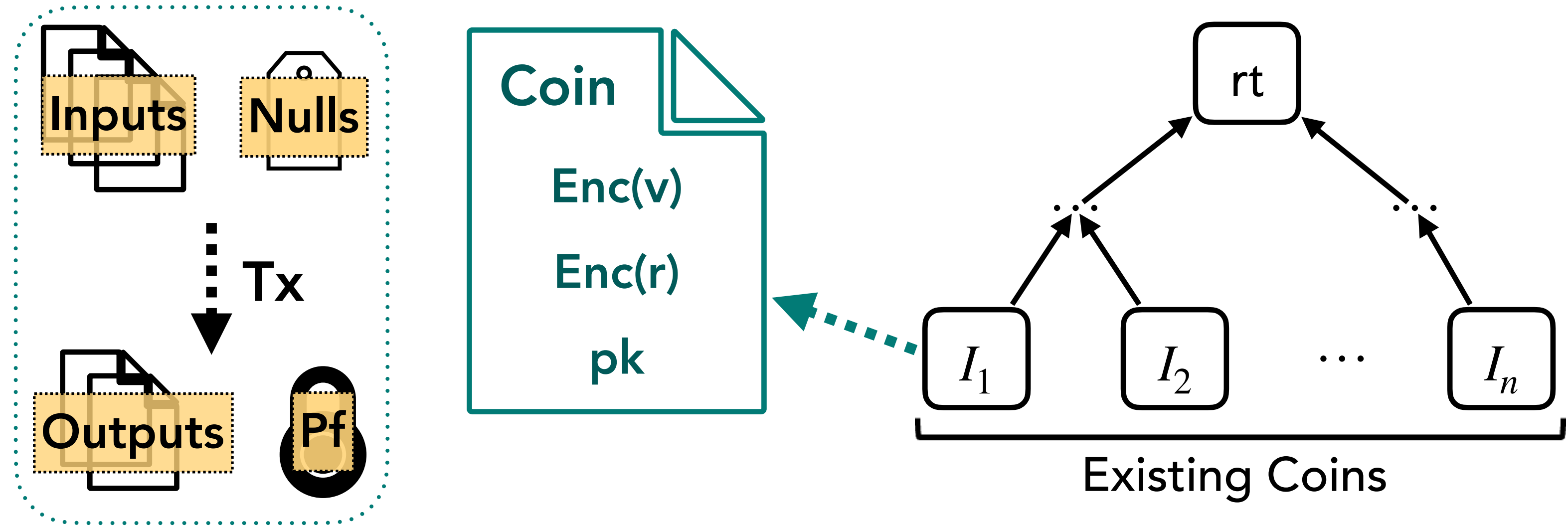
(as of August 18, 2023)

# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$

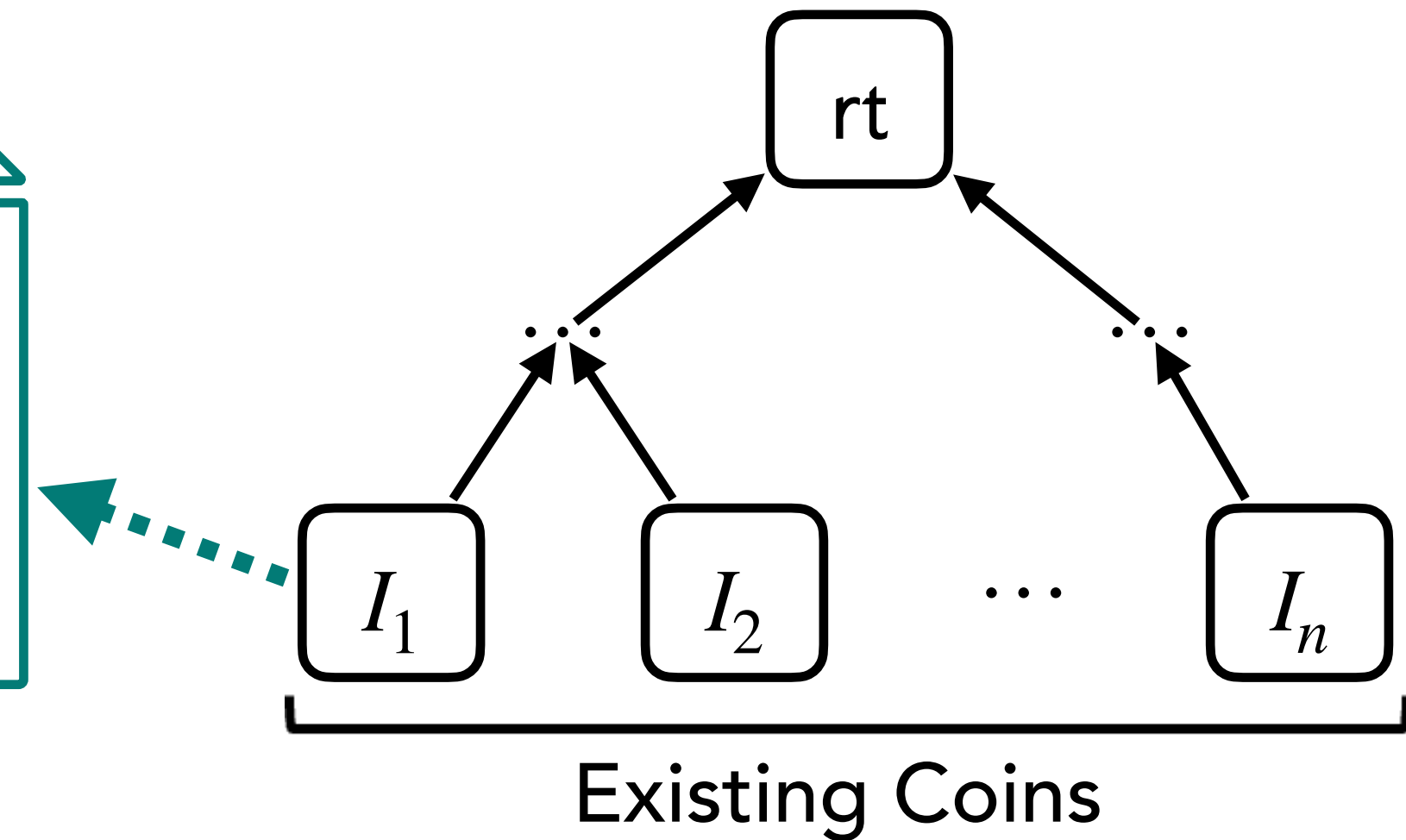
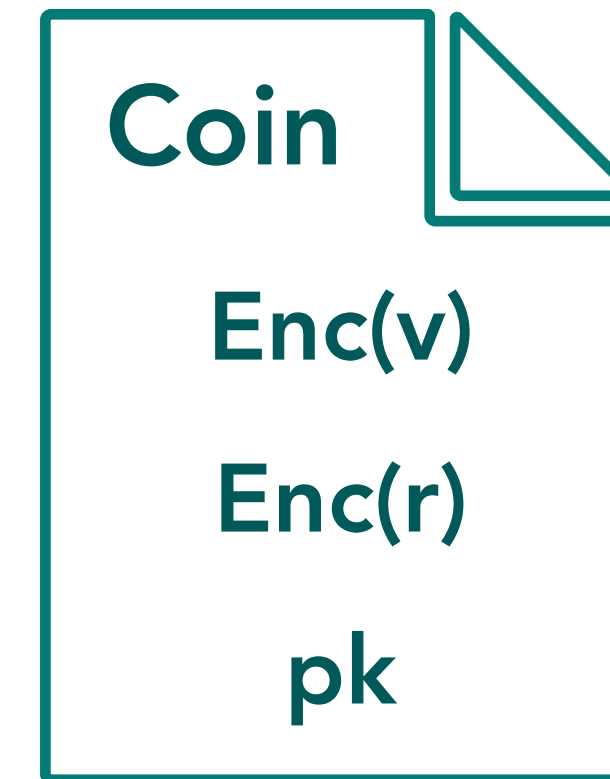
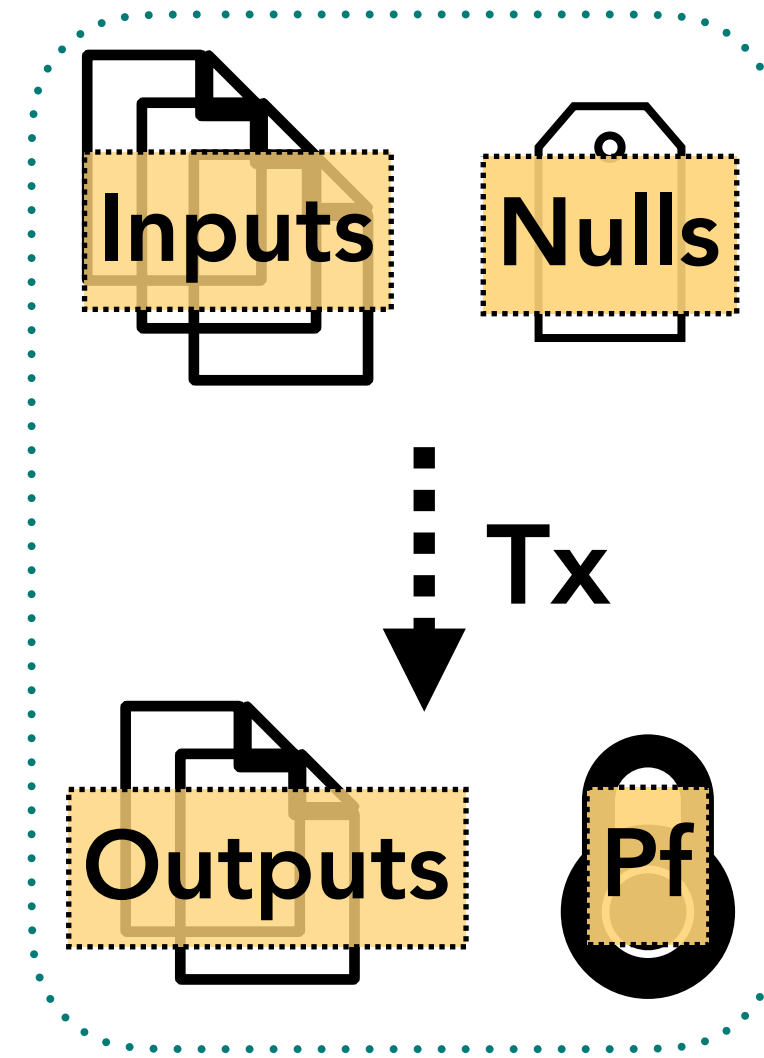


# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$



## Proof Relation: (proved using Plonk)

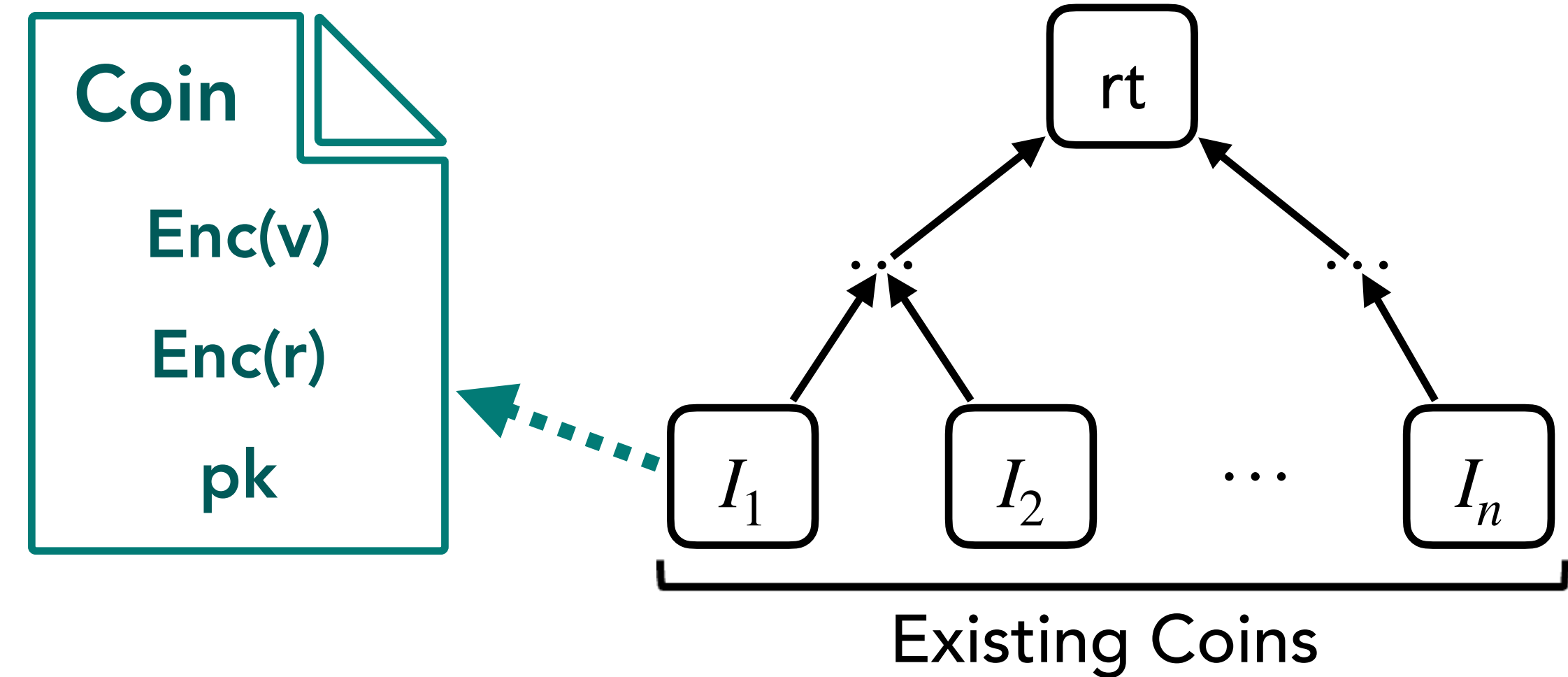
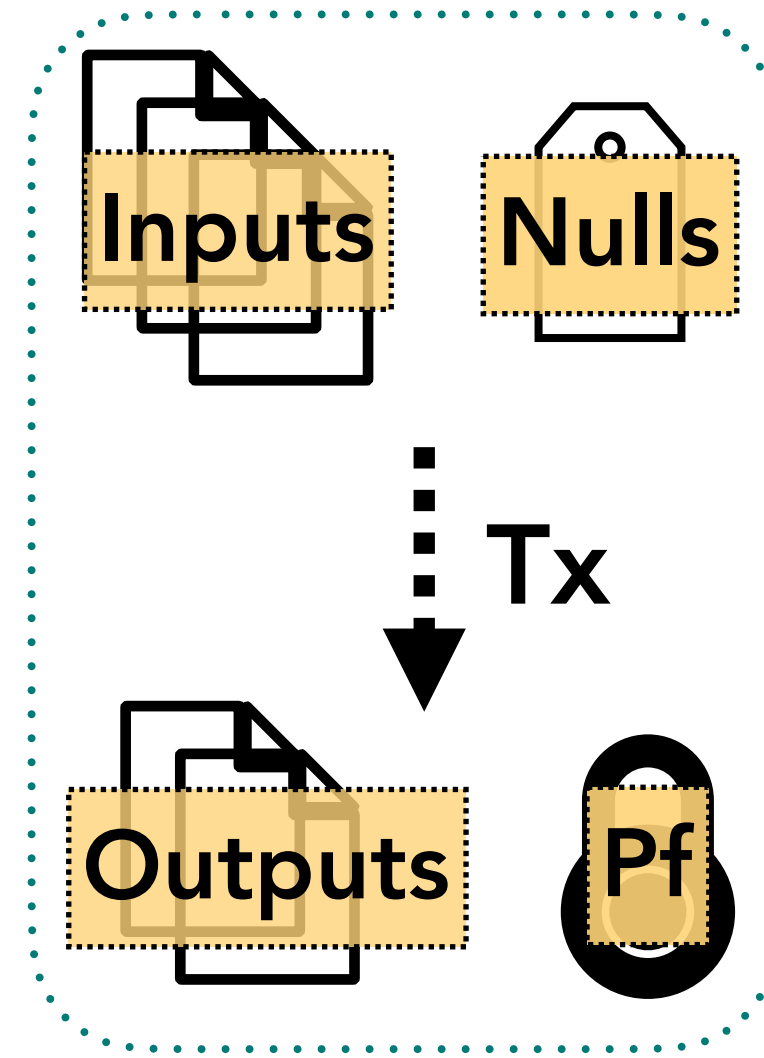
- Nullifier check:  $null_I = H(pk, pos_I), \forall$  input  $I$
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output
- Equality check:  $\sum v_{in} = \sum v_{out}$
- Merkle membership:  $I$  is in position  $pos_I$  w.r.t root  $rt$

# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$



### Weak F-S Attack:

## Proof Relation: (proved using Plonk)

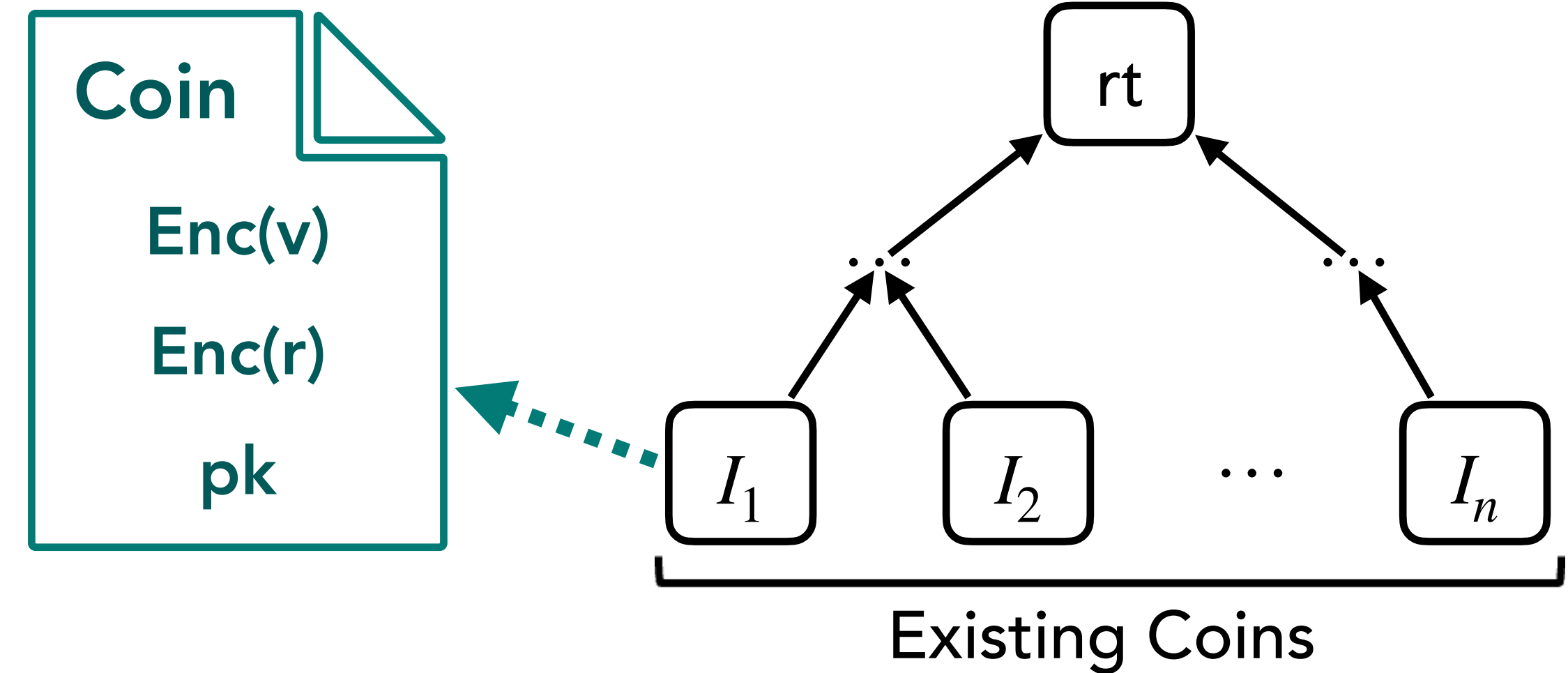
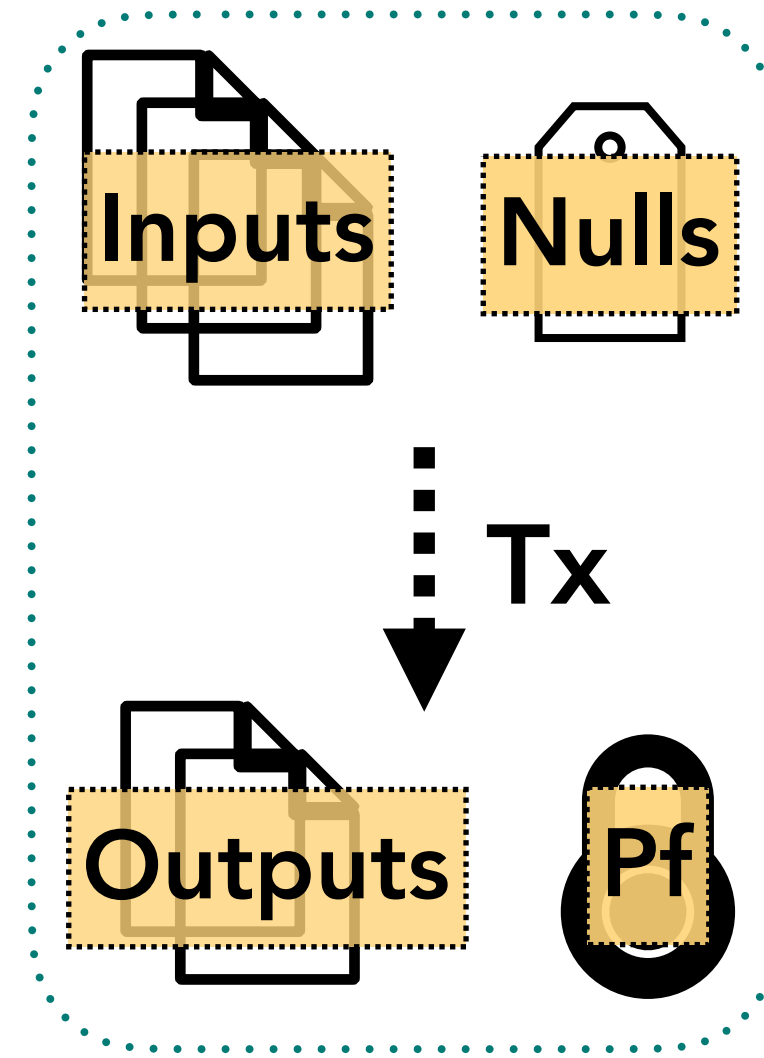
- Nullifier check:  $null_I = H(pk, pos_I), \forall$  input  $I$
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output
- Equality check:  $\sum v_{in} = \sum v_{out}$
- Merkle membership:  $I$  is in position  $pos_I$  w.r.t root  $rt$

# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$



## Proof Relation: (proved using Plonk)

- Nullifier check:  $null_I = H(pk, pos_I), \forall$  input  $I$
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output
- Equality check:  $\sum v_{in} = \sum v_{out}$
- Merkle membership:  $I$  is in position  $pos_I$  w.r.t root  $rt$

## Weak F-S Attack:

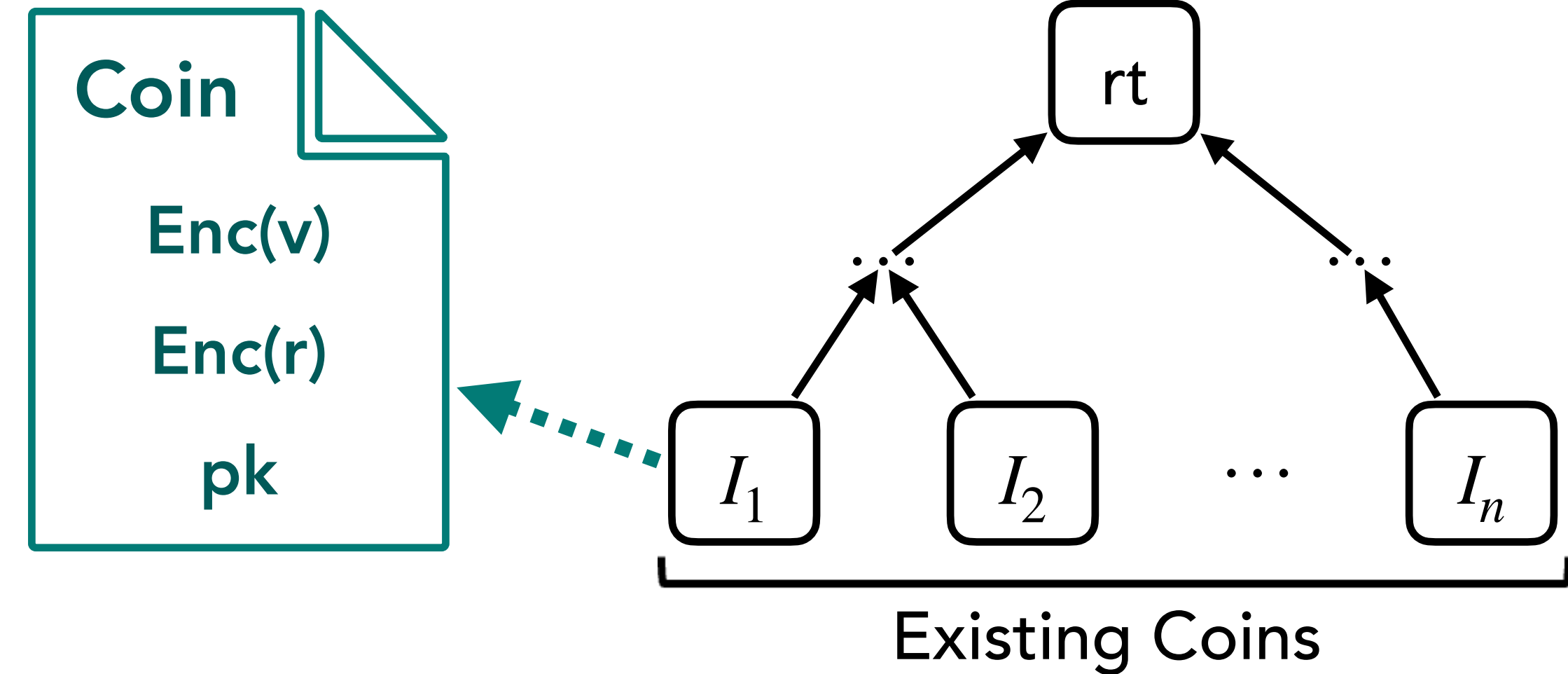
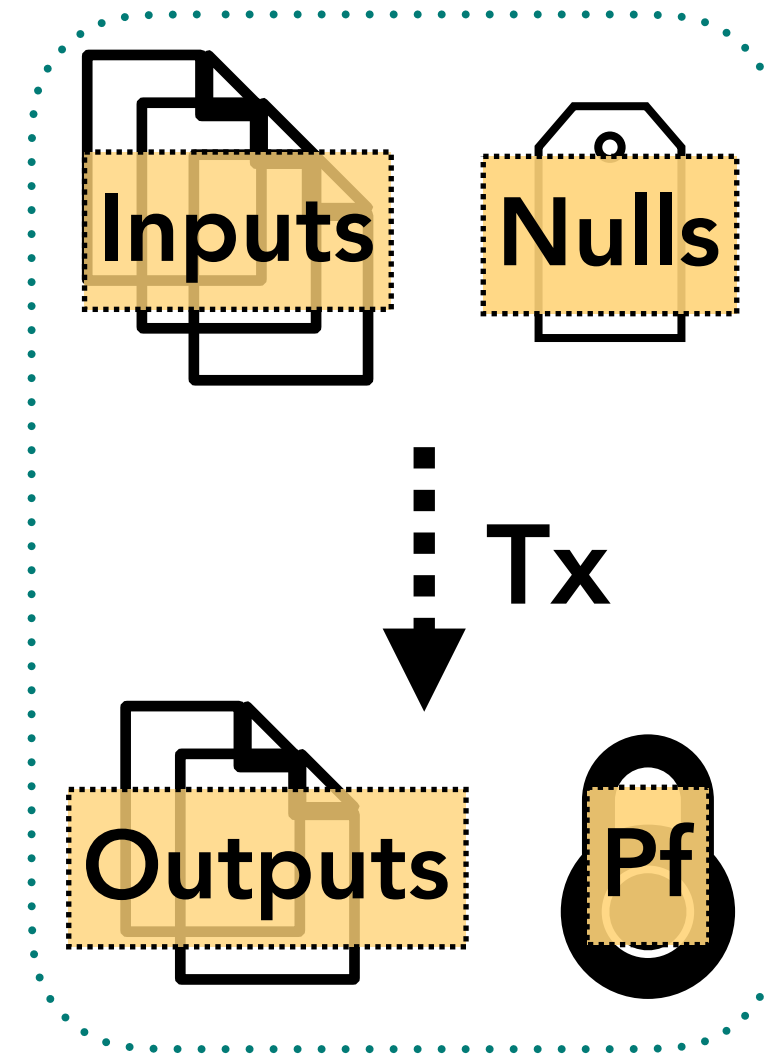


# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$



### Weak F-S Attack:

## Proof Relation: (proved using Plonk)

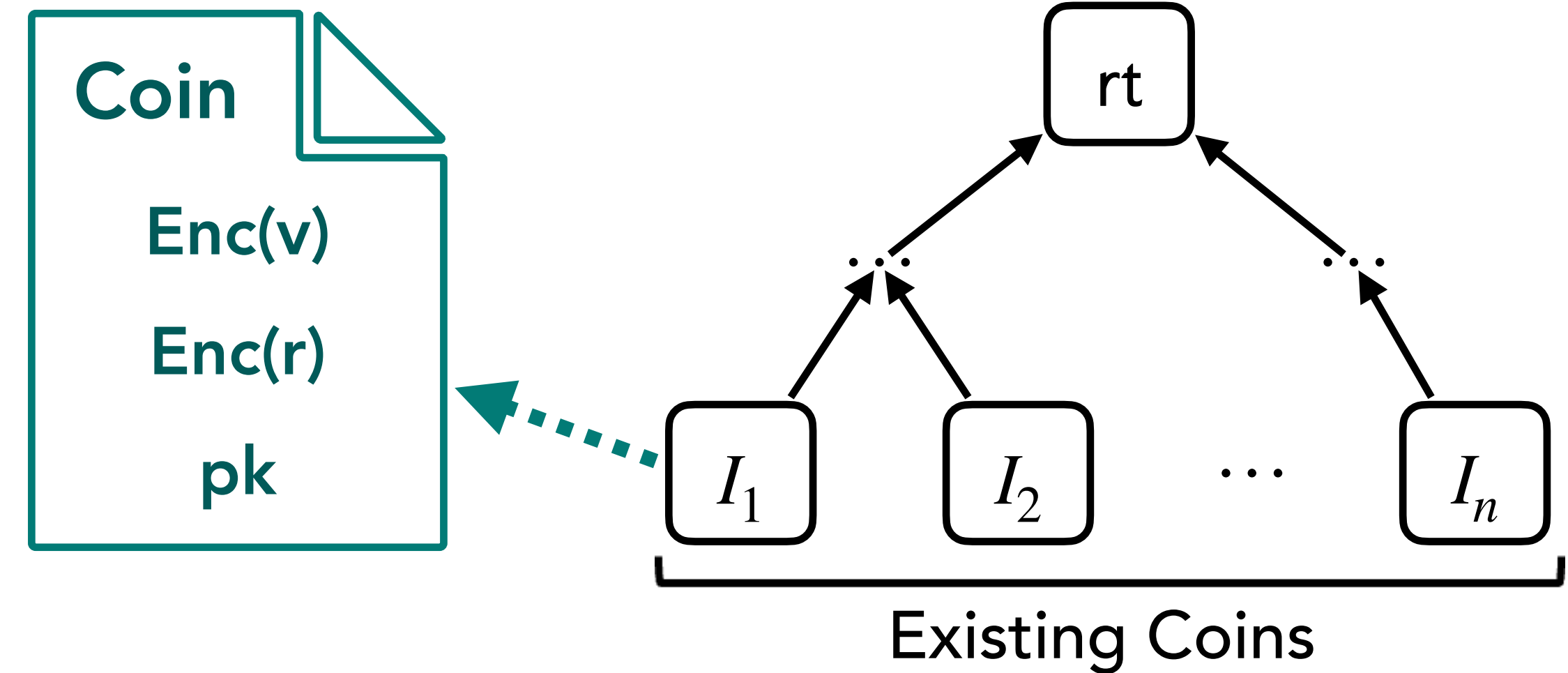
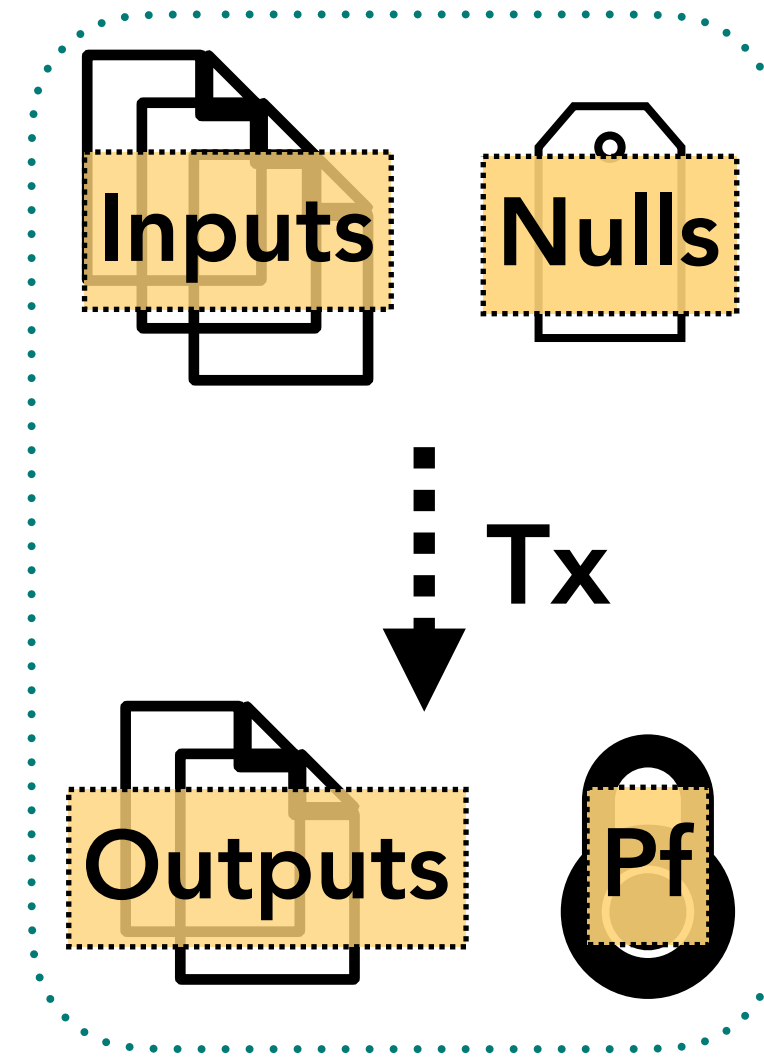
- Nullifier check:  $null_I = H(pk_{I_1})$ ,  $\forall$  input  $I$
- Range check:  $v_{in}, v_{out} \in [0, 2^{r-1}]$ ,  $\forall$  input & output
- Equality check:  $\sum v_{in} = \sum v_{out}$
- Merkle membership:  $I$  is in position  $pos_I$  w.r.t root  $rt$

# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$



## Proof Relation: (proved using Plonk)

- Nullifier check:  $null_I = H(pk, I)$ ,  $\forall$  input  $I$
- Range check:  $v_{in}, v_{out} \in [0, 2^{32} - 1]$ ,  $\forall$  input & output
- Equality check:  $\sum v_{in} = \sum v_{out}$
- Merkle membership:  $I$  is in position  $pos_I$  w.r.t root  $rt$

## Weak F-S Attack:

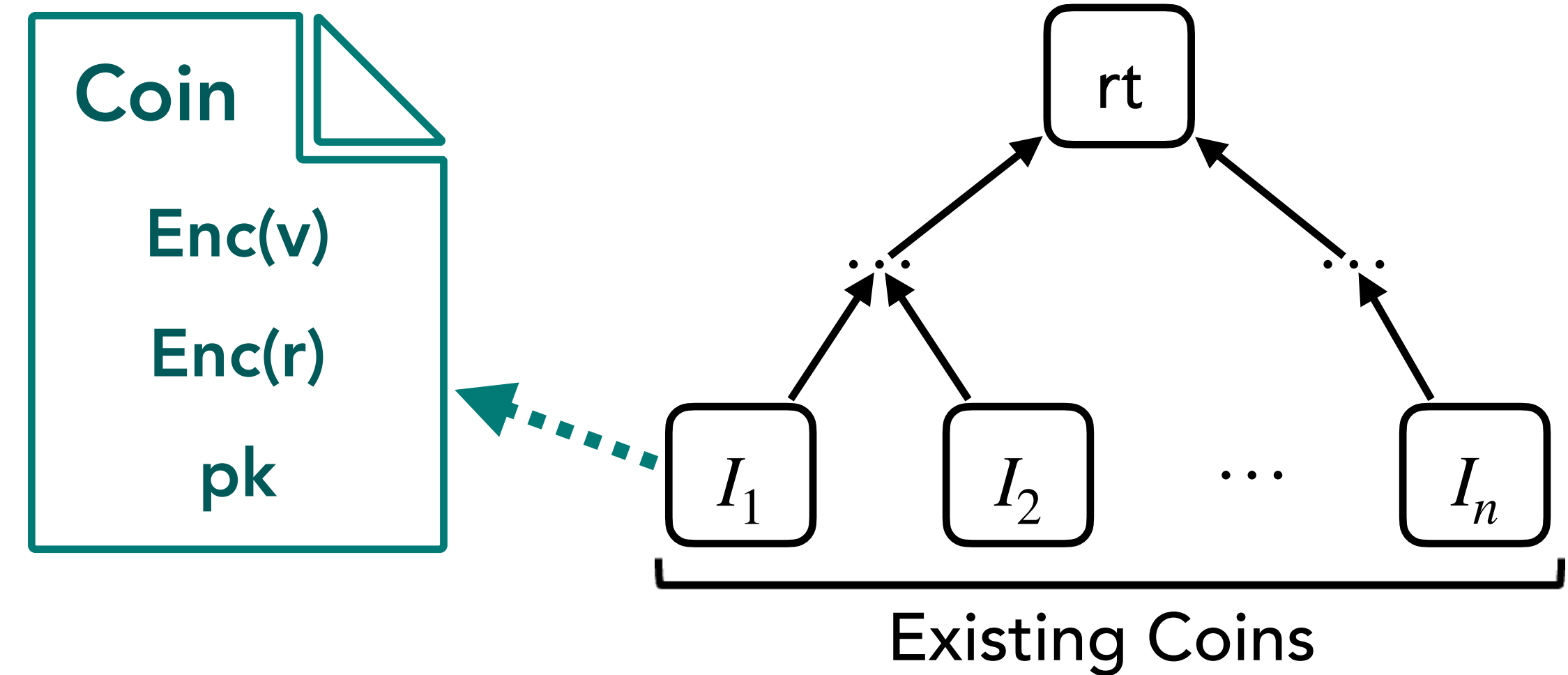
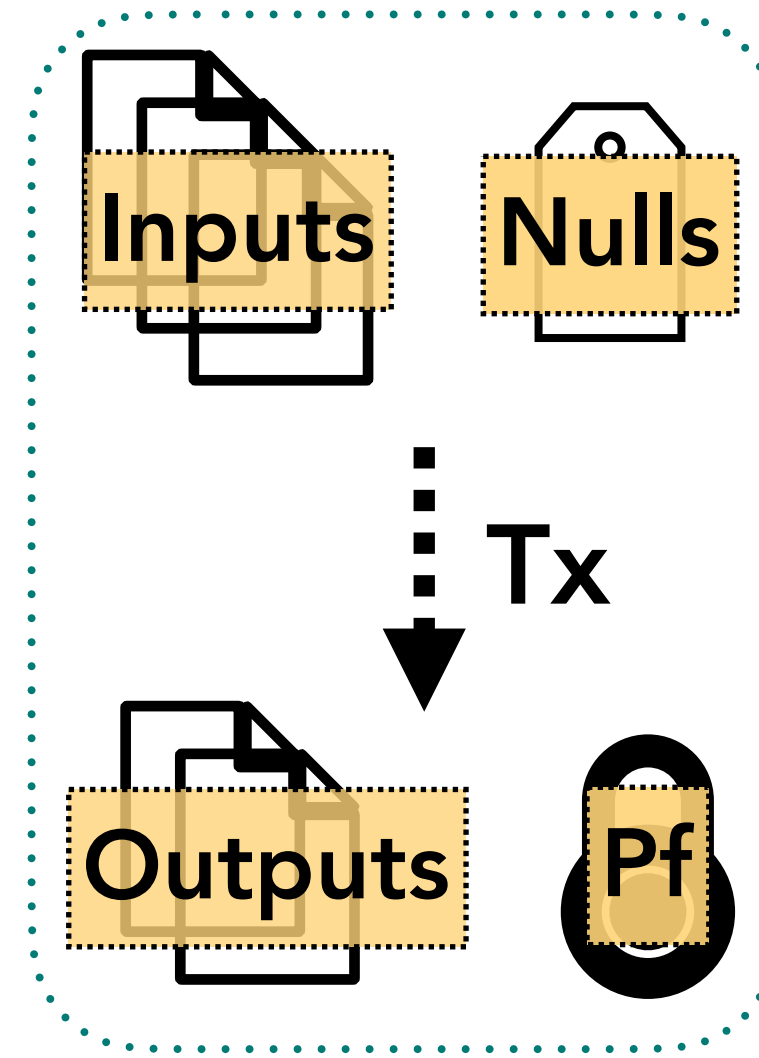
1. Create output coin w/ value 1 trillion DUSK.

# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$



## Proof Relation: (proved using Plonk)

- Nullifier check:  $null_I = H(pk, I)$ ,  $\forall$  input  $I$
- Range check:  $v_{in}, v_{out} \in [0, 2^{32} - 1]$ ,  $\forall$  input & output
- Equality check:  $\sum v_{in} = \sum v_{out}$
- Merkle membership:  $I$  is in position  $pos_I$  w.r.t root  $rt$

## Weak F-S Attack:

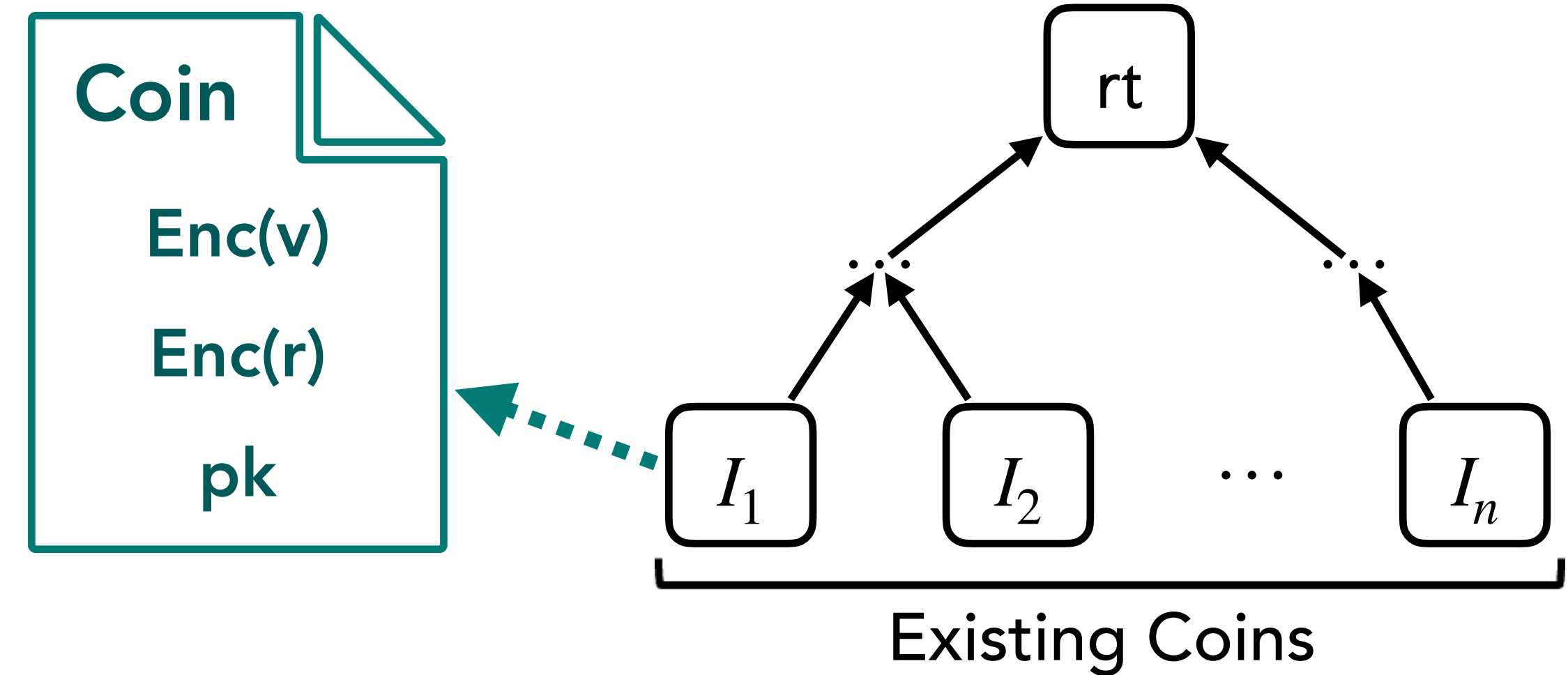
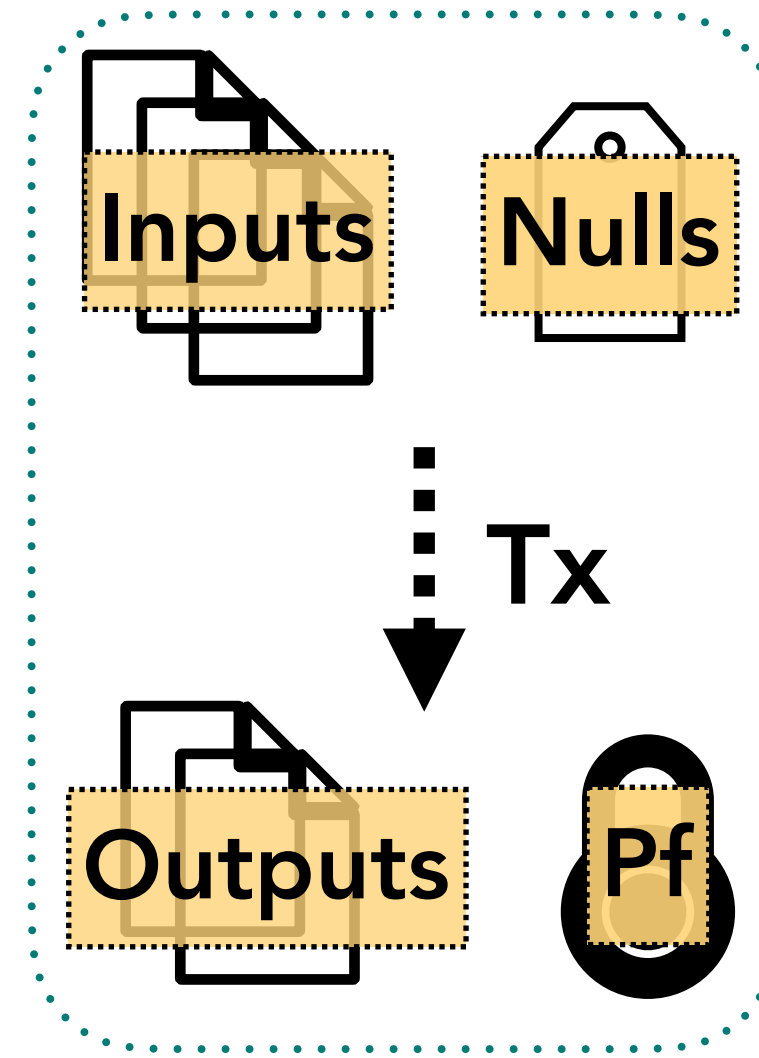
1. Create output coin w/ value **1 trillion DUSK.**
2. **Forge** Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .

# Case Study: Dusk Network

## Transaction Model (simplified):

### Public Inputs:

- Set of inputs & output coins
- Nullifier  $null_I$  for each input  $I$



## Proof Relation: (proved using Plonk)

- Nullifier check:  $null_I = H(pk, I)$ ,  $\forall$  input  $I$
- Range check:  $v_{in}, v_{out} \in [0, 2^{32} - 1]$ ,  $\forall$  input & output
- Equality check:  $\sum v_{in} = \sum v_{out}$
- Merkle membership:  $I$  is in position  $pos_I$  w.r.t root  $rt$

## Weak F-S Attack:

1. Create output coin w/ value **1 trillion DUSK.**
2. **Forge** Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:

### Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:

March 18



Vulnerability  
disclosed

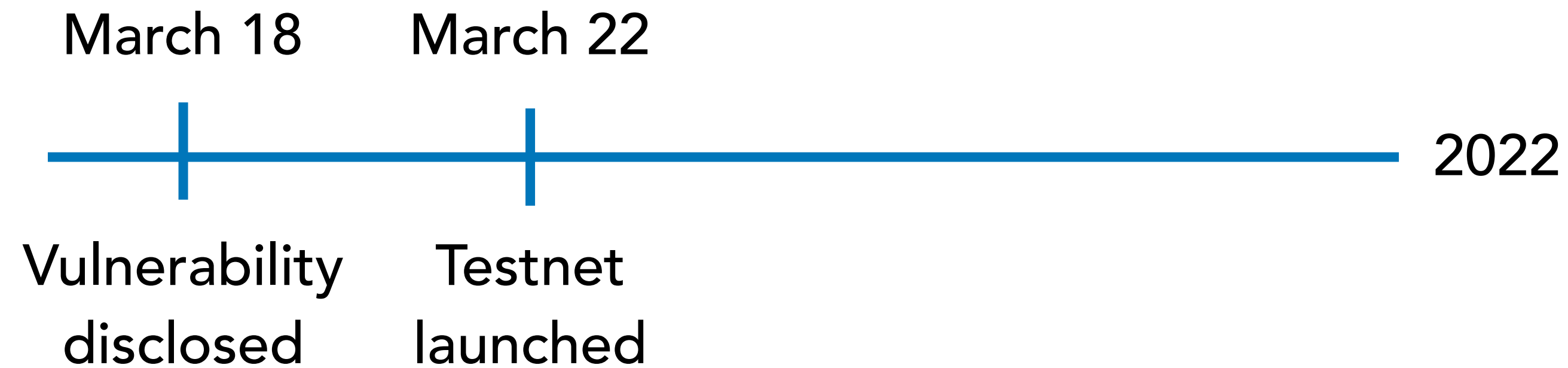
## Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:



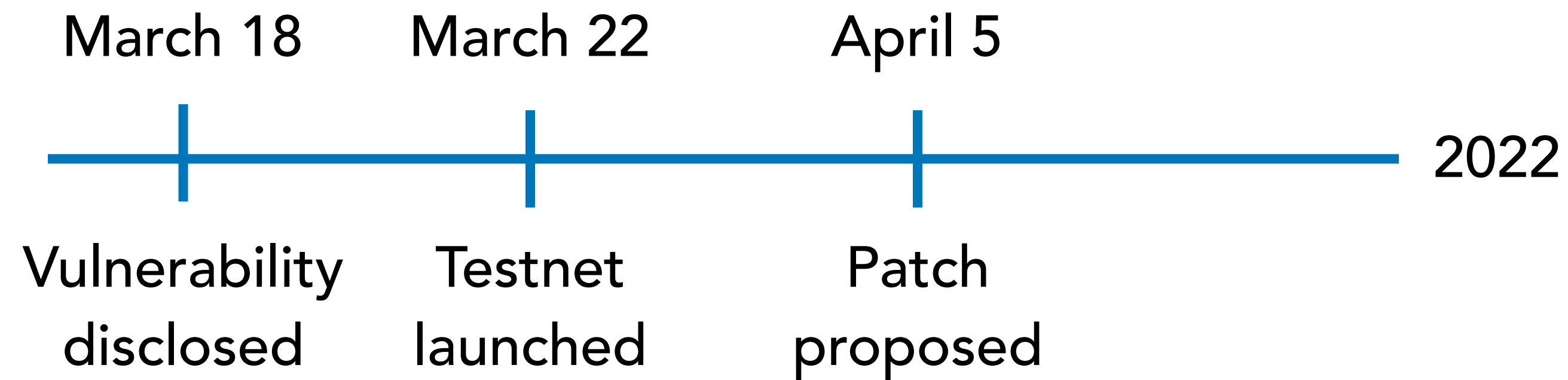
## Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:



## Weak F-S Attack:

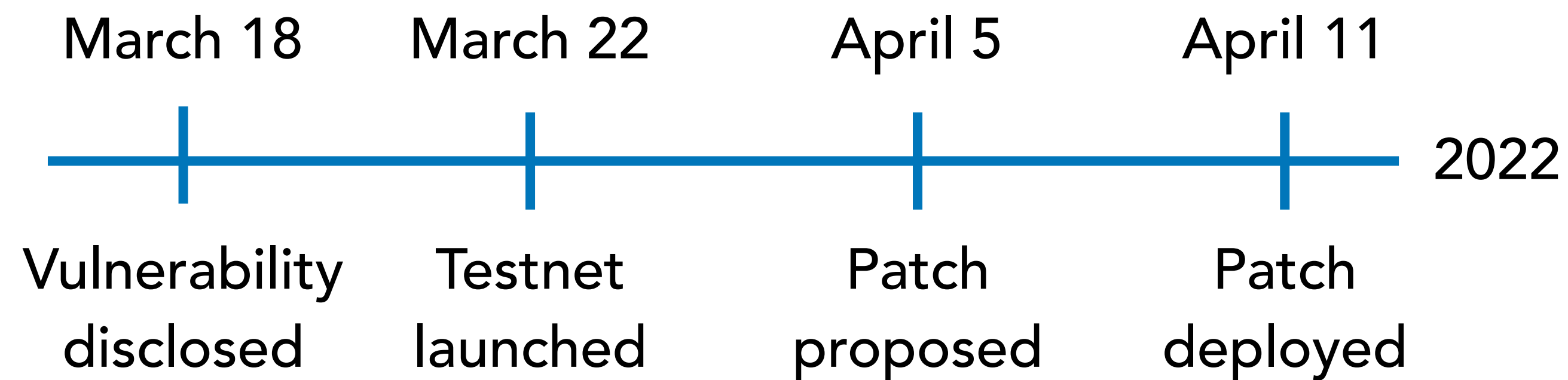
1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .





# Case Study: Dusk Network

## Disclosure Timeline:



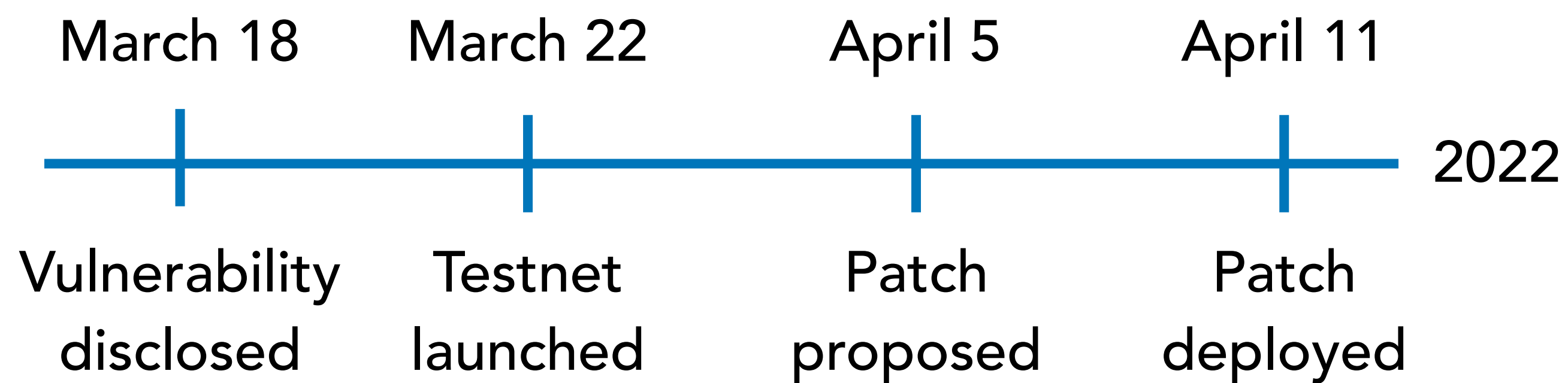
## Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:



Apr 12, 2022 - Mels Dees

## PLONK Critical Vulnerability Successfully Remediated

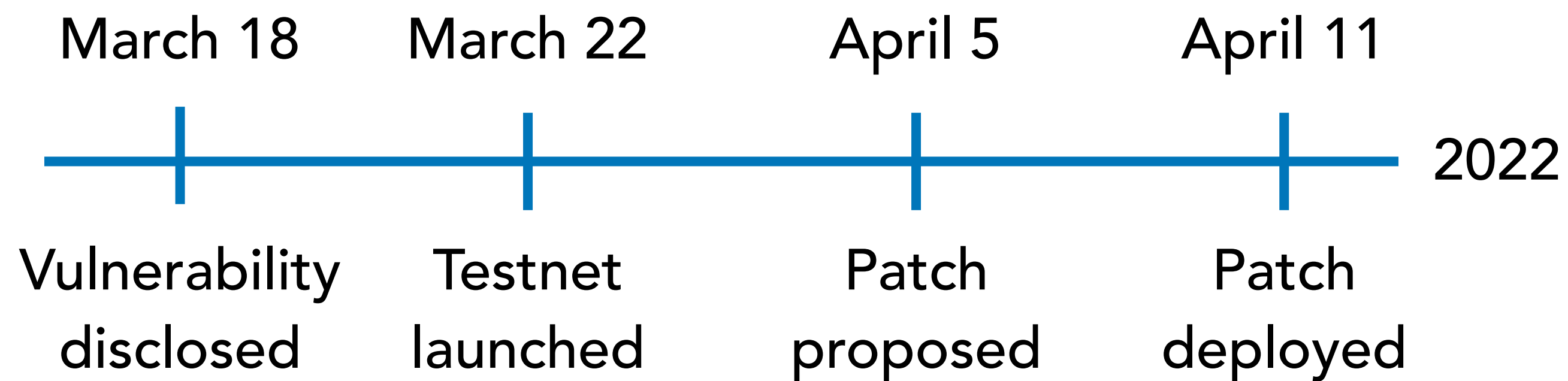
### Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:



## Was this attack exploited?

Apr 12, 2022 - Mels Dees

PLONK Critical Vulnerability  
Successfully Remediated

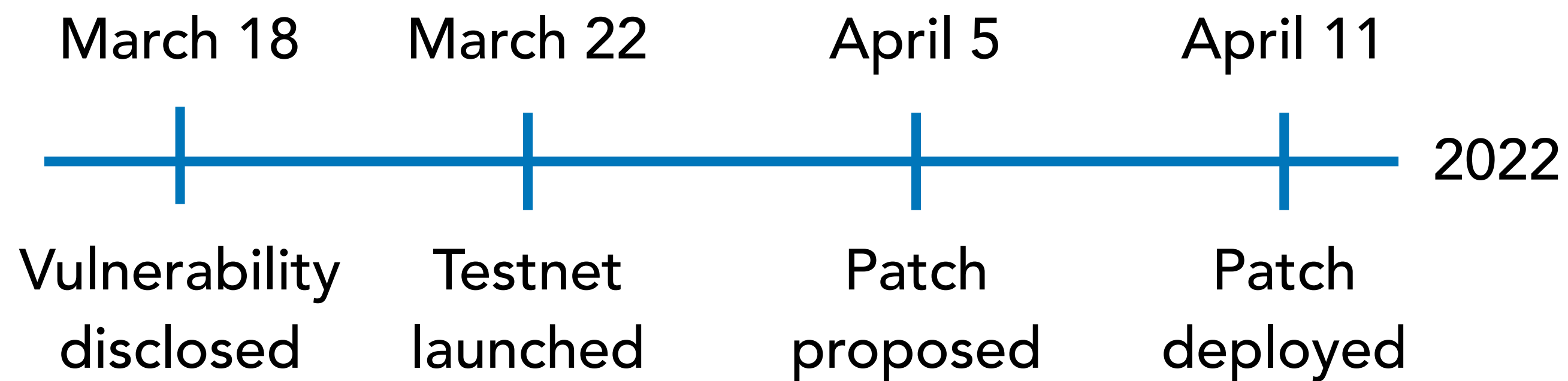
### Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:



## Was this attack exploited?

- Unlikely given short timeline...

Apr 12, 2022 - Mels Dees

PLONK Critical Vulnerability  
Successfully Remediated

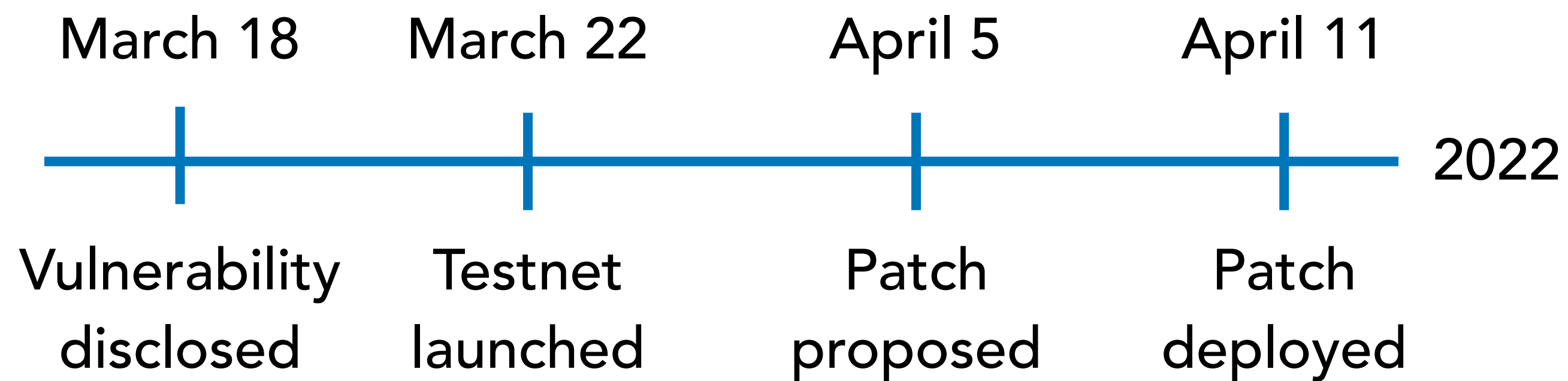
## Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:



## Was this attack exploited?

- Unlikely given short timeline...
- But cannot be ruled out!

Apr 12, 2022 - Mels Dees

PLONK Critical Vulnerability  
Successfully Remediated

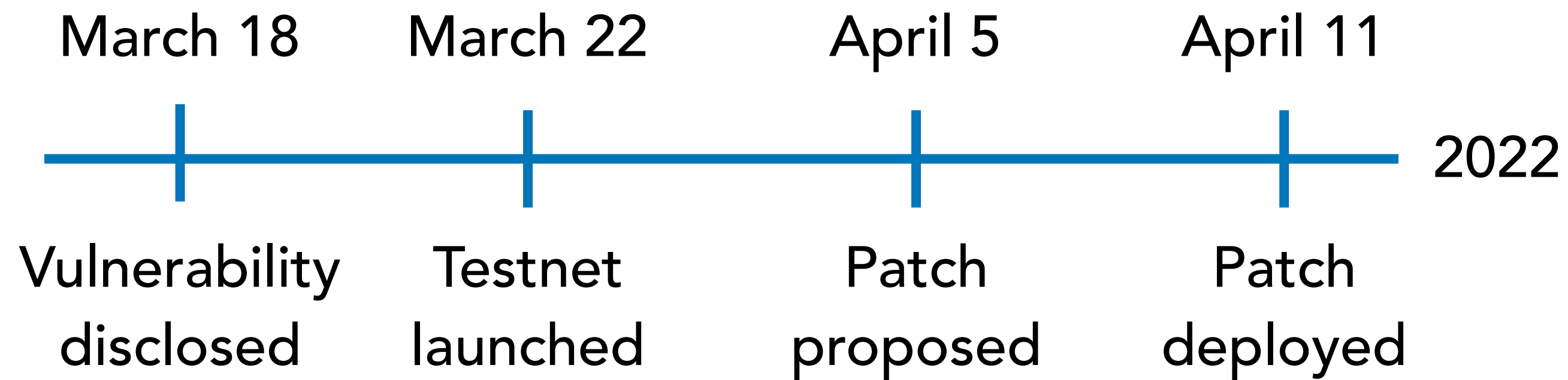
## Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Dusk Network

## Disclosure Timeline:



## Was this attack exploited?

- Unlikely given short timeline...
- But cannot be ruled out!
- Forged proofs are indistinguishable from honest proofs

Apr 12, 2022 - Mels Dees

PLONK Critical Vulnerability  
Successfully Remediated

## Weak F-S Attack:

1. Create output coin w/ value 1 trillion DUSK.
2. Forge Plonk proof  $\pi$  w/ arbitrary input, setting nullifier to satisfy  $\pi$ .



# Case Study: Incognito Chain

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

**\$250M+**

Volume shielded

**+6M**

Anonymous transactions

**100+**

Coins supported

**16**

Bridges supported



# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

**\$250M+**

Volume shielded

**+6M**

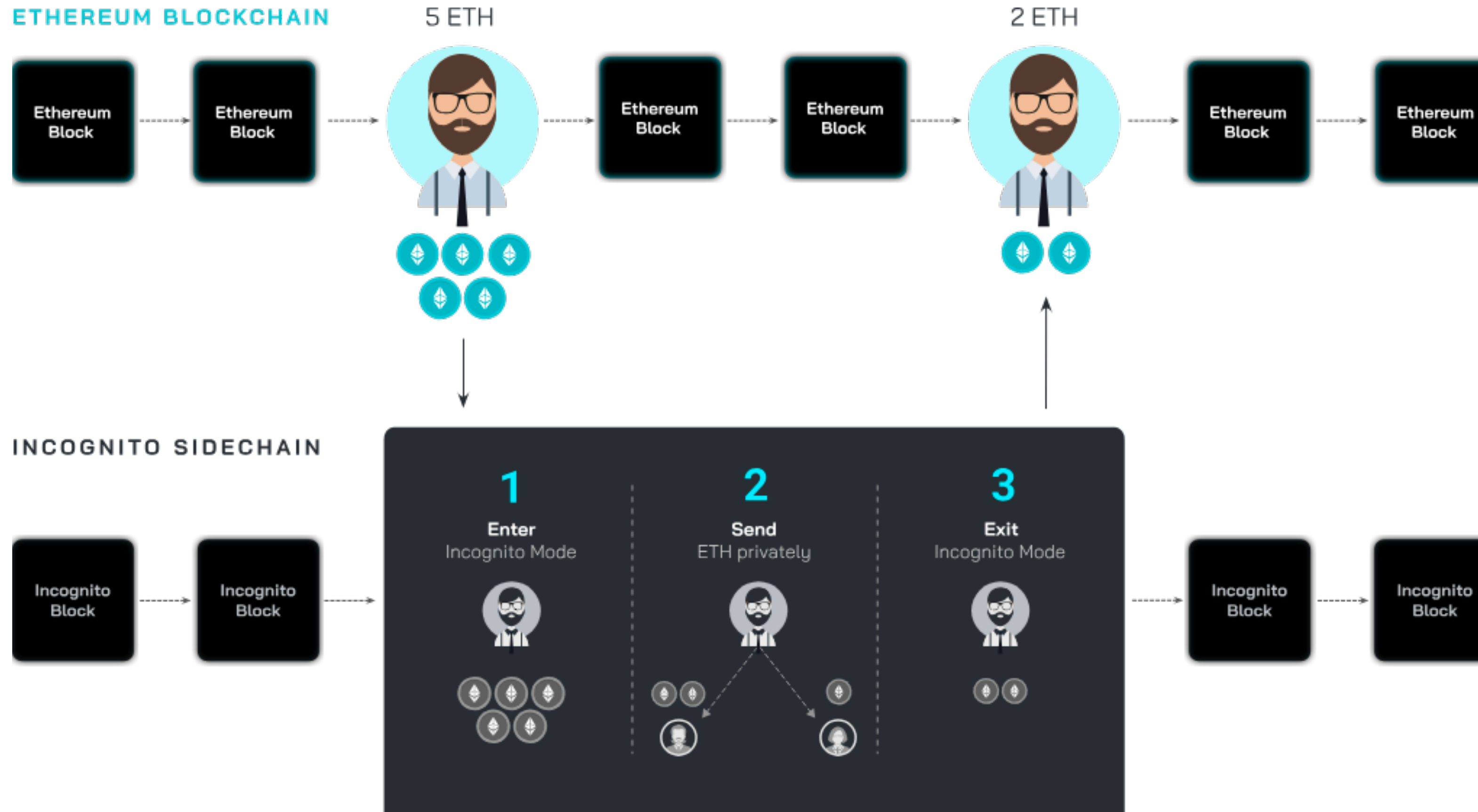
Anonymous transactions

**100+**

Coins supported

**16**

Bridges supported



# Case Study: Incognito Chain

Description:

Incognito

The privacy layer of crypto

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

Proof Relation:

- Equality check:  $\sum v_{in} = \sum v_{out}$
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

Proof Relation:

- Equality check:  $\sum v_{in} = \sum v_{out}$   $\Leftarrow$  enforced by (linkable) ring signature
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

Proof Relation:

- **Equality check:**  $\sum v_{in} = \sum v_{out}$   $\Leftarrow$  enforced by (linkable) ring signature
- **Range check:**  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output  $\Leftarrow$  enforced by BP aggregate range proofs

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

Proof Relation:

- **Equality check:**  $\sum v_{in} = \sum v_{out}$   $\Leftarrow$  enforced by (linkable) ring signature
- **Range check:**  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output  $\Leftarrow$  enforced by BP aggregate range proofs

Weak F-S Attack:

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

Proof Relation:

• Equality check:  $\sum v_{in} = \sum v_{out}$

⇐ enforced by (linkable) ring signature

• Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output

⇐ enforced by BP aggregate range proofs

Weak F-S Attack:

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

Proof Relation:

• Equality check:  $\sum v_{in} = \sum v_{out}$

⇐ enforced by (linkable) ring signature

• Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output

⇐ enforced by BP aggregate range proofs

Weak F-S Attack:

• Choose  $\vec{v}_{in}, \vec{v}_{out}$  to satisfy equality check as well as BP verification equation



# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

Proof Relation:

• Equality check:  $\sum v_{in} = \sum v_{out}$

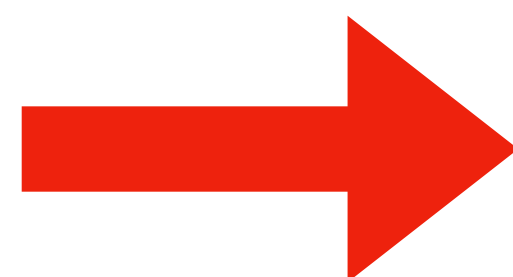
⇐ enforced by (linkable) ring signature

• Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output

⇐ enforced by BP aggregate range proofs

Weak F-S Attack:

• Choose  $\vec{v}_{in}, \vec{v}_{out}$  to satisfy equality check as well as BP verification equation



# Case Study: Incognito Chain

Description:

○ Incognito

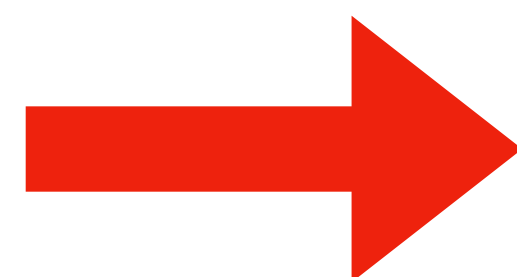
The privacy layer of crypto

Proof Relation:

- Equality check:  $\sum v_{in} = \sum v_{out}$   $\Leftarrow$  enforced by (linkable) ring signature
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output  $\Leftarrow$  enforced by BP aggregate range proofs

Weak F-S Attack:

- Choose  $\vec{v}_{in}, \vec{v}_{out}$  to satisfy equality check as well as BP verification equation



$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

Description:

○ Incognito

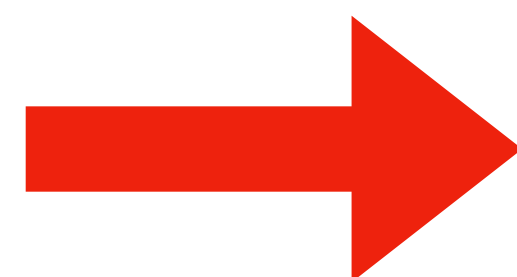
The privacy layer of crypto

Proof Relation:

- Equality check:  $\sum v_{in} = \sum v_{out}$   $\Leftarrow$  enforced by (linkable) ring signature
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output  $\Leftarrow$  enforced by BP aggregate range proofs

Weak F-S Attack:

- Choose  $\vec{v}_{in}, \vec{v}_{out}$  to satisfy equality check as well as BP verification equation



$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

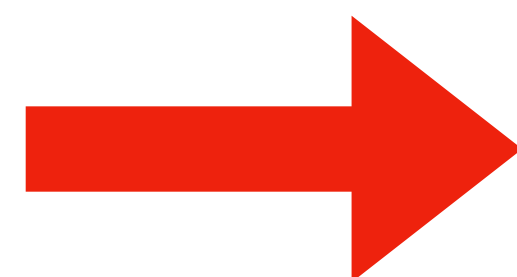
Proof Relation:

- Equality check:  $\sum v_{in} = \sum v_{out}$   $\Leftarrow$  enforced by (linkable) ring signature
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output  $\Leftarrow$  enforced by BP aggregate range proofs

Weak F-S Attack:

- Choose  $\vec{v}_{in}, \vec{v}_{out}$  to satisfy equality check as well as BP verification equation

1 PRV



$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

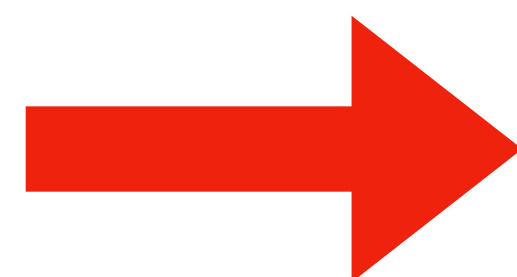
Proof Relation:

- Equality check:  $\sum v_{in} = \sum v_{out}$   $\Leftarrow$  enforced by (linkable) ring signature
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output  $\Leftarrow$  enforced by BP aggregate range proofs

Weak F-S Attack:

- Choose  $\vec{v}_{in}, \vec{v}_{out}$  to satisfy equality check as well as BP verification equation

1 PRV



$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

1 bazillion PRV!

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

Description:

○ Incognito

The privacy layer of crypto

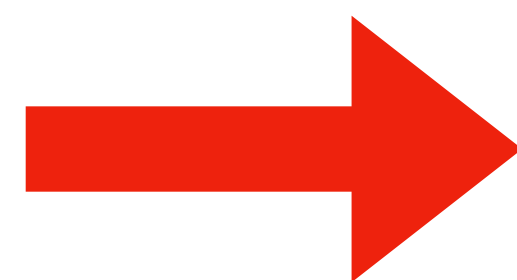
Proof Relation:

- Equality check:  $\sum v_{in} = \sum v_{out}$   $\Leftarrow$  enforced by (linkable) ring signature
- Range check:  $v_{in}, v_{out} \in [0, 2^{64} - 1], \forall$  input & output  $\Leftarrow$  enforced by BP aggregate range proofs

Weak F-S Attack:

- Choose  $\vec{v}_{in}, \vec{v}_{out}$  to satisfy equality check as well as BP verification equation

1 PRV



$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

1 bazillion PRV!

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

## Disclosure Timeline:

1 PRV

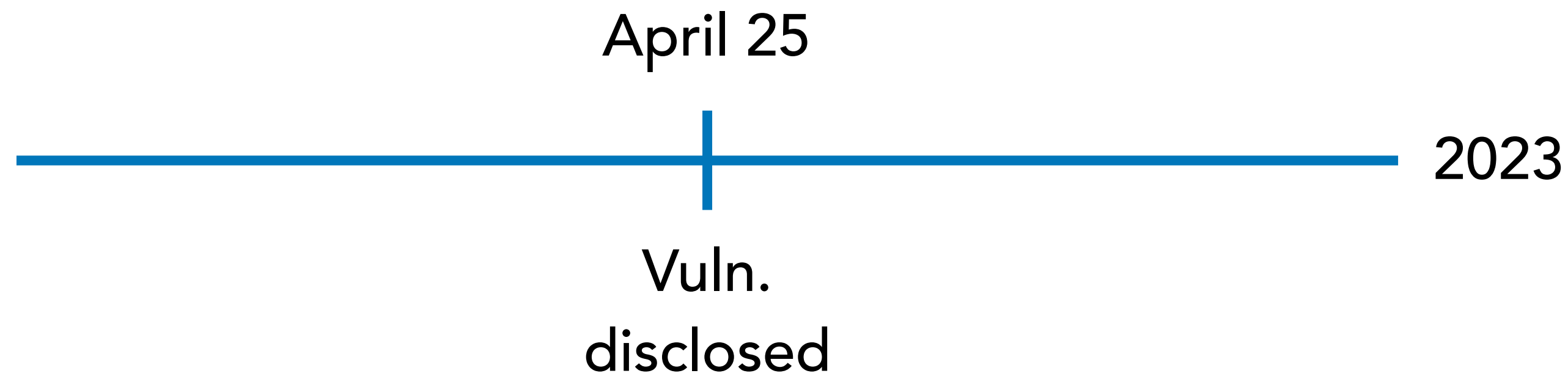
$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

1 bazillion PRV!

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

## Disclosure Timeline:



1 PRV

$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

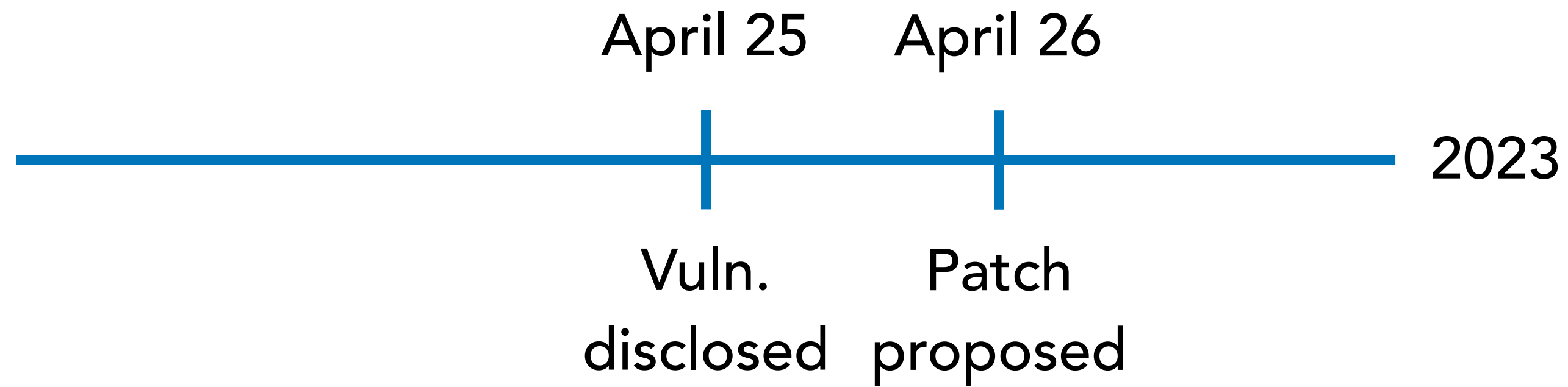
1 bazillion PRV!

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )



# Case Study: Incognito Chain

## Disclosure Timeline:



1 PRV

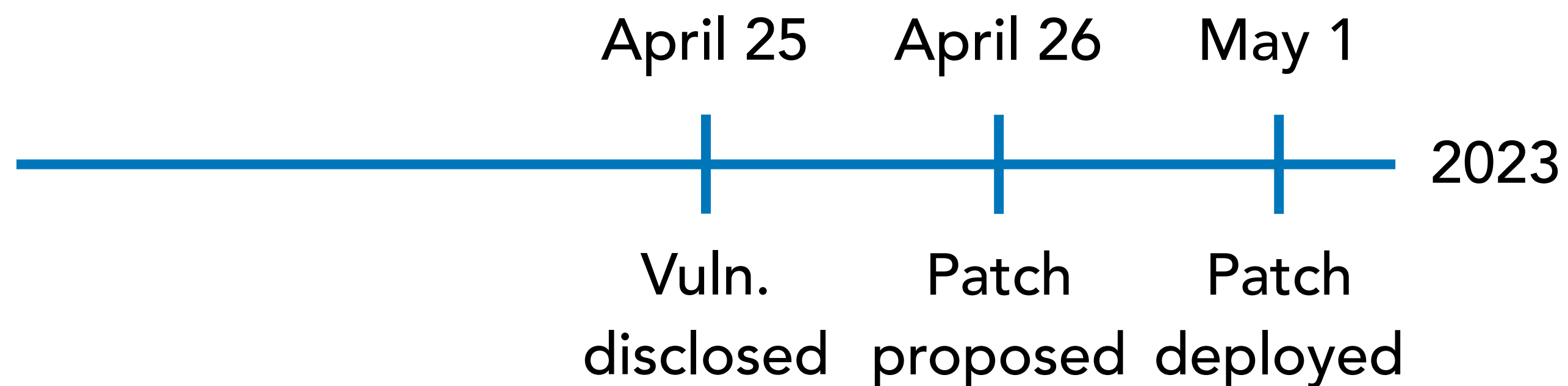
$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

1 bazillion PRV!

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

## Disclosure Timeline:



## What Happened?

A few days ago, the group Trail of Bits (<https://www.trailofbits.com/about/>), a security research company, contacted us to inform they have identified an issue in our bulletproof implementation, that we use with our privacy layer.

We have patched the issue with Docker tag 20230429\_1, the fix has been reviewed and confirmed by the Trail of Bits team.

1 PRV

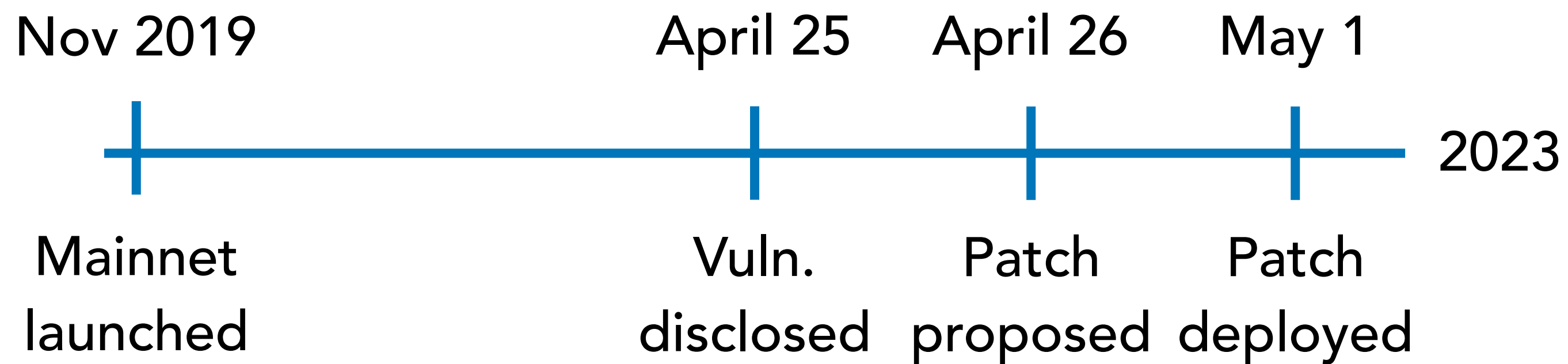
$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

1 bazillion PRV!

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

## Disclosure Timeline:



## What Happened?

A few days ago, the group Trail of Bits (<https://www.trailofbits.com/about/>), a security research company, contacted us to inform they have identified an issue in our bulletproof implementation, that we use with our privacy layer.

We have patched the issue with Docker tag 20230429\_1, the fix has been reviewed and confirmed by the Trail of Bits team.

1 PRV

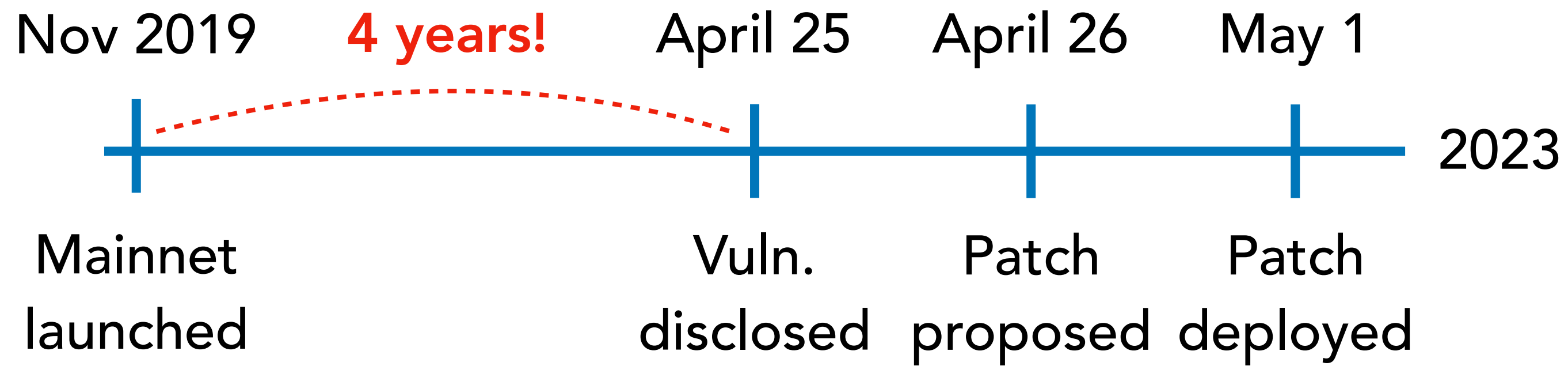
$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

1 bazillion PRV!

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

## Disclosure Timeline:



## What Happened?

A few days ago, the group Trail of Bits (<https://www.trailofbits.com/about/>), a security research company, contacted us to inform they have identified an issue in our bulletproof implementation, that we use with our privacy layer.

We have patched the issue with Docker tag 20230429\_1, the fix has been reviewed and confirmed by the Trail of Bits team.

1 PRV

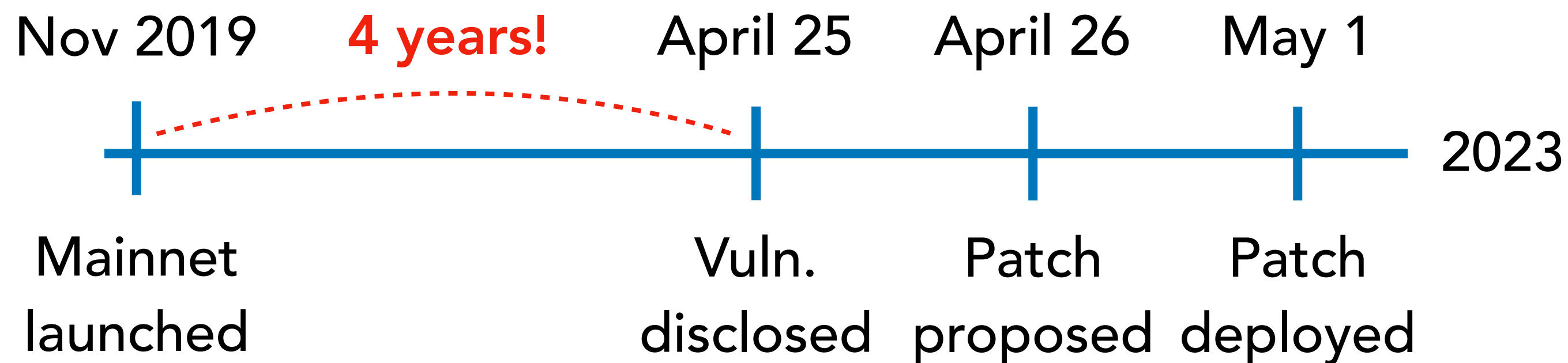
$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

1 bazillion PRV!

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Case Study: Incognito Chain

## Disclosure Timeline:



## What Happened?

A few days ago, the group Trail of Bits (<https://www.trailofbits.com/about/>), a security research company, contacted us to inform they have identified an issue in our bulletproof implementation, that we use with our privacy layer.

We have patched the issue with Docker tag 20230429\_1, the fix has been reviewed and confirmed by the Trail of Bits team.

## Was this attack exploited?

- As with Plonk, forged BP proofs are indistinguishable from honest proofs
- So we don't know...

**1 PRV**

$$\begin{cases} v_1 = v_2 + v_3 + v_4 \\ v_1 z^2 + v_2 z^3 + v_3 z^4 + v_4 z^5 = \hat{t} - \delta(y, z) - t_1 x - t_2 x^2 \\ \gamma_1 z^2 + \gamma_2 z^3 + \gamma_3 z^4 + \gamma_4 z^5 = \beta_x - \beta_1 x - \beta_2 x^2 \end{cases}$$

**1 bazillion PRV!**

(input  $v_1$  and outputs  $v_2, v_3, v_4$ )

# Weak Fiat-Shamir Attacks

## Practical Impacts

**Why is Weak F-S so widespread?**

**(do practitioners know about the dangers of weak F-S?)**

# **Insufficient Coverage of "correct" Fiat-Shamir**

# Insufficient Coverage of “correct” Fiat-Shamir

How is Fiat-Shamir presented in academic papers?



# Insufficient Coverage of “correct” Fiat-Shamir

How is Fiat-Shamir presented in academic papers?

1. Mention that Fiat-Shamir can be applied, with no specification for the transform.

# Insufficient Coverage of “correct” Fiat-Shamir

## How is Fiat-Shamir presented in academic papers?

### 1. Mention that Fiat-Shamir can be applied, with no specification for the transform.

**Removing interaction.** Our construction can be made non-interactive in the random oracle model using Fiat-Shamir heuristic [28]. Though GKR protocol is not constant round, recent results [14, 22] show that

as well. Finally, public-coin interactive arguments may be cryptographically compiled into SNARKs using the Fiat-Shamir transform.

subsequent step, the argument can be made non-interactive via the Fiat-Shamir transformation, and thereby obtain a preprocessing SNARG with universal SRS.

We apply the Fiat-Shamir heuristic to the protocol from Section 5 to obtain a non-interactive argument of knowledge that is secure in the random oracle model

Challenges are random field elements. In practice we assume that the Fiat-Shamir heuristic would be applied in order to obtain a non-

Hyrax-I is a public-coin protocol, we apply the Fiat-Shamir heuristic [45] to produce a zkSNARK that we call Hyrax whose

The above SNARK is obtained via a popular paradigm that combines a polynomial IOP and a polynomial commitment scheme in order to obtain an interactive argument, and then relies on the Fiat-Shamir paradigm

Finally, since our protocol is public coin, it can be made non-interactive in the random oracle model using the Fiat-Shamir transform [55], thereby obtaining a family

be made non-interactive in the random oracle model using the Fiat-Shamir transform [FS86], and be instantiated (heuristically) in the plain model using a

witness-extended emulation. Applying the Fiat-Shamir transform [FS86] to the public-coin interactive argument results in the claimed SNARK for  $\mathcal{R}_{\text{R1CS}}$ .<sup>14</sup>

# Insufficient Coverage of “correct” Fiat-Shamir

How is Fiat-Shamir presented in academic papers?

1. Mention that Fiat-Shamir can be applied, with no specification for the transform.

# Insufficient Coverage of “correct” Fiat-Shamir

How is Fiat-Shamir presented in academic papers?

1. Mention that Fiat-Shamir can be applied, with no specification for the transform.
2. Attempt to specify Fiat-Shamir:

# Insufficient Coverage of “correct” Fiat-Shamir

## How is Fiat-Shamir presented in academic papers?

1. Mention that Fiat-Shamir can be applied, with no specification for the transform.
2. Attempt to specify Fiat-Shamir:  
⇒ (some) does not get it right on the first try!

# Insufficient Coverage of “correct” Fiat-Shamir

How is Fiat-Shamir presented in academic papers?

1. Mention that Fiat-Shamir can be applied, with no specification for the transform.
2. Attempt to specify Fiat-Shamir:  
⇒ (some) does not get it right on the first try!

Plonk:

# Insufficient Coverage of “correct” Fiat-Shamir

## How is Fiat-Shamir presented in academic papers?

1. Mention that Fiat-Shamir can be applied, with no specification for the transform.
2. Attempt to specify Fiat-Shamir:

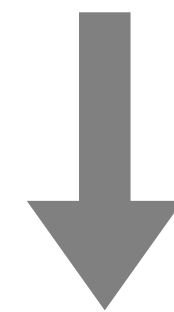
⇒ (some) does not get it right on the first try!

## Plonk:

Compute quotient challenge  $\alpha \in \mathbb{F}_p$  :

$$\alpha = H([a]_1, [b]_1, [c]_1, [z]_1)$$

(December 2019)



We describe the protocol below as a non-interactive protocol using the Fiat-Shamir heuristic. For this purpose we always denote by transcript the concatenation of the common preprocessed input, and public input, and the proof elements written by the prover up to a certain point in time. We use transcript for obtaining random challenges via

(March 2020)

# Insufficient Coverage of “correct” Fiat-Shamir

How is Fiat-Shamir presented in academic papers?

1. Mention that Fiat-Shamir can be applied, with no specification for the transform.
2. Attempt to specify Fiat-Shamir:  
⇒ (some) does not get it right on the first try!

Plonk:

Bulletproofs:



# Insufficient Coverage of “correct” Fiat-Shamir

## How is Fiat-Shamir presented in academic papers?

1. Mention that Fiat-Shamir can be applied, with no specification for the transform.
2. Attempt to specify Fiat-Shamir:

⇒ (some) does not get it right on the first try!

## Plonk:

challenges are replaced by hashes of the transcript up to that point. For instance  $y = H(A, S)$  and  $z = H(A, S, y)$

(July 2018)

## Bulletproofs:

random challenges are replaced by hashes of the transcript up to that point, including the statement itself. For example, one could set  $y = H(st, A, S)$  and  $z = H(A, S, y)$ , where  $st$  is the statement.

(April 2022)



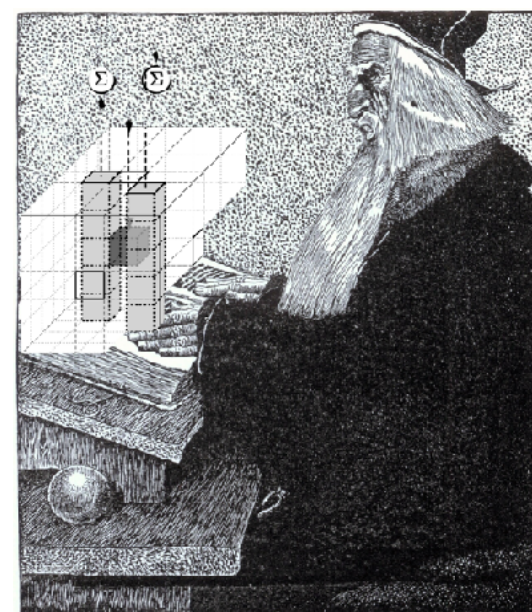
# How to prevent weak Fiat-Shamir?

# How to prevent weak Fiat-Shamir?

Existing tooling does not prevent weak F-S

# How to prevent weak Fiat-Shamir?

## Existing tooling does not prevent weak F-S



### Merlin: composable proof transcripts for public-coin arguments of knowledge

---

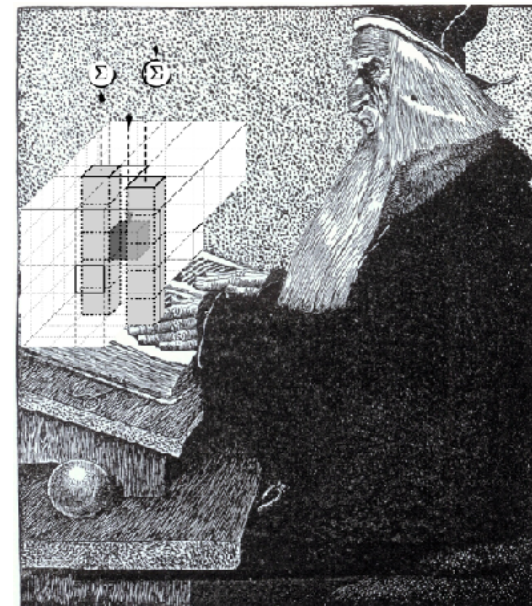
Merlin is a [STROBE](#)-based transcript construction for zero-knowledge proofs. It automates the Fiat-Shamir transform, so that by using Merlin, non-interactive protocols can be implemented as if they were interactive.

This is significantly easier and less error-prone than performing the transformation by hand, and in addition, it also provides natural support for:

- multi-round protocols with alternating commit and challenge phases;
- natural domain separation, ensuring challenges are bound to the statements to be proved;
- automatic message framing, preventing ambiguous encoding of commitment data;
- and protocol composition, by using a common transcript for multiple protocols.

# How to prevent weak Fiat-Shamir?

## Existing tooling does not prevent weak F-S



**Merlin: composable proof transcripts for public-coin arguments of knowledge**

Merlin is a [STROBE](#)-based transcript construction for zero-knowledge proofs. It automates the Fiat-Shamir transform, so that by using Merlin, non-interactive protocols can be implemented as if they were interactive.

This is significantly easier and less error-prone than performing the transformation by hand, and in addition, it also provides natural support for:

- multi-round protocols with alternating commit and challenge phases;
- natural domain separation, ensuring challenges are bound to the statements to be proved;
- automatic message framing, preventing ambiguous encoding of commitment data;
- and protocol composition, by using a common transcript for multiple protocols.

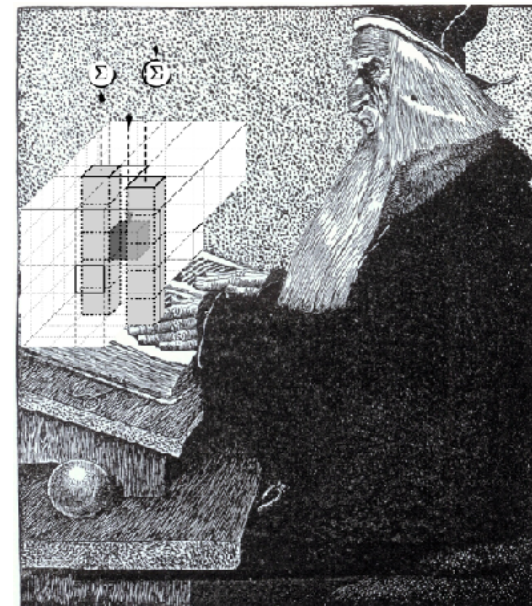
## Dusk Network patch (April 2022)

```
use merlin::Transcript;
```

```
// PIs have to be part of the transcript  
for pi in public_inputs.iter() {  
    .. cloned_transcript.append_scalar(b"pi", pi);  
}
```

# How to prevent weak Fiat-Shamir?

## Existing tooling does not prevent weak F-S



### Merlin: composable proof transcripts for public-coin arguments of knowledge

---

Merlin is a [STROBE](#)-based transcript construction for zero-knowledge proofs. It automates the Fiat-Shamir transform, so that by using Merlin, non-interactive protocols can be implemented as if they were interactive.

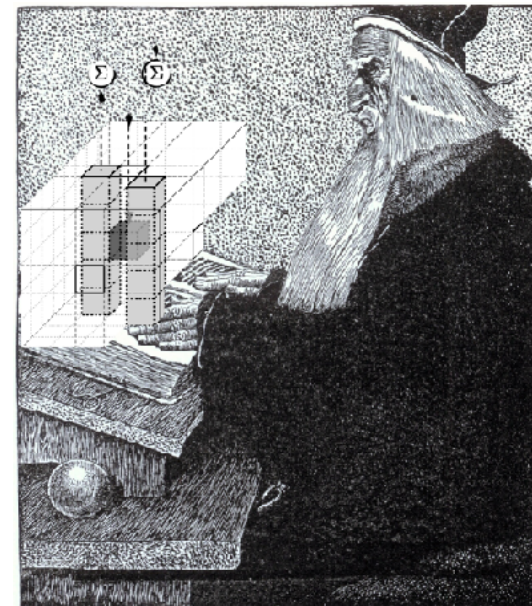
This is significantly easier and less error-prone than performing the transformation by hand, and in addition, it also provides natural support for:

- multi-round protocols with alternating commit and challenge phases;
- natural domain separation, ensuring challenges are bound to the statements to be proved;
- automatic message framing, preventing ambiguous encoding of commitment data;
- and protocol composition, by using a common transcript for multiple protocols.

# How to prevent weak Fiat-Shamir?

## Existing tooling does not prevent weak F-S

## Mitigation Idea:



### Merlin: composable proof transcripts for public-coin arguments of knowledge

Merlin is a [STROBE](#)-based transcript construction for zero-knowledge proofs. It automates the Fiat-Shamir transform, so that by using Merlin, non-interactive protocols can be implemented as if they were interactive.

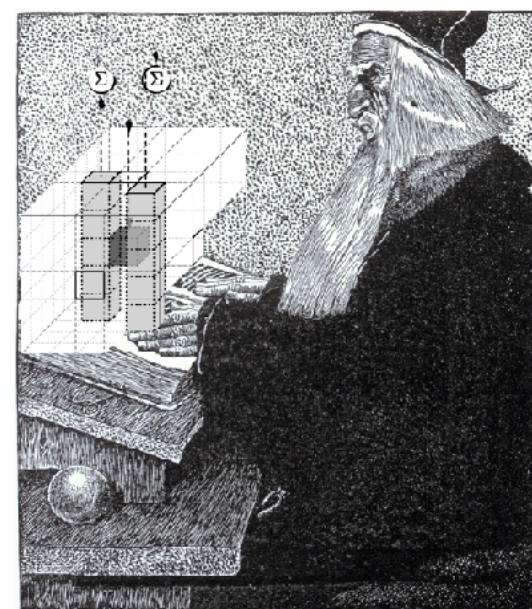
This is significantly easier and less error-prone than performing the transformation by hand, and in addition, it also provides natural support for:

- multi-round protocols with alternating commit and challenge phases;
- natural domain separation, ensuring challenges are bound to the statements to be proved;
- automatic message framing, preventing ambiguous encoding of commitment data;
- and protocol composition, by using a common transcript for multiple protocols.

## Detection Idea:

# How to prevent weak Fiat-Shamir?

## Existing tooling does not prevent weak F-S



**Merlin: composable proof transcripts for public-coin arguments of knowledge**

Merlin is a [STROBE](#)-based transcript construction for zero-knowledge proofs. It automates the Fiat-Shamir transform, so that by using Merlin, non-interactive protocols can be implemented as if they were interactive.

This is significantly easier and less error-prone than performing the transformation by hand, and in addition, it also provides natural support for:

- multi-round protocols with alternating commit and challenge phases;
- natural domain separation, ensuring challenges are bound to the statements to be proved;
- automatic message framing, preventing ambiguous encoding of commitment data;
- and protocol composition, by using a common transcript for multiple protocols.

## Mitigation Idea:

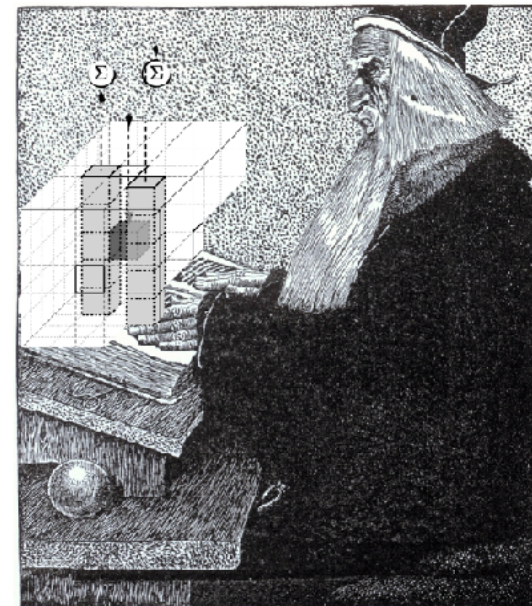
- Declare (to Merlin) protocol flow ahead of time
- Raise flag if this is not followed

## Detection Idea:



# How to prevent weak Fiat-Shamir?

## Existing tooling does not prevent weak F-S



**Merlin: composable proof transcripts for public-coin arguments of knowledge**

Merlin is a [STROBE](#)-based transcript construction for zero-knowledge proofs. It automates the Fiat-Shamir transform, so that by using Merlin, non-interactive protocols can be implemented as if they were interactive.

This is significantly easier and less error-prone than performing the transformation by hand, and in addition, it also provides natural support for:

- multi-round protocols with alternating commit and challenge phases;
- natural domain separation, ensuring challenges are bound to the statements to be proved;
- automatic message framing, preventing ambiguous encoding of commitment data;
- and protocol composition, by using a common transcript for multiple protocols.

## Mitigation Idea:

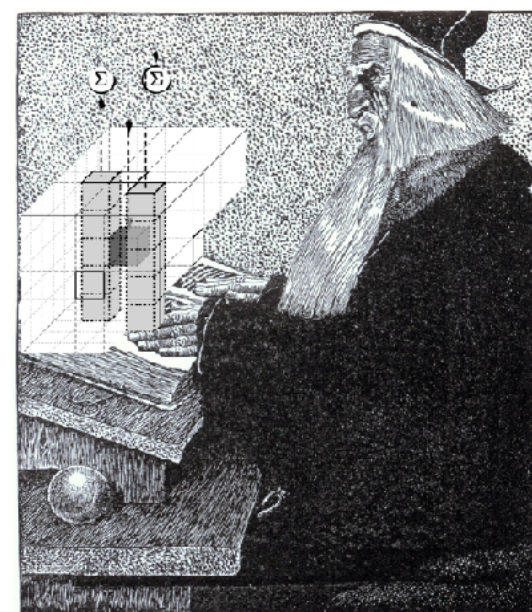
- Declare (to Merlin) protocol flow ahead of time
- Raise flag if this is not followed

## Detection Idea:

- Transcript should contain all objects flowed through both P & V
- If not, raise flag (w/ some error)

# How to prevent weak Fiat-Shamir?

## Existing tooling does not prevent weak F-S



**Merlin: composable proof transcripts for public-coin arguments of knowledge**

Merlin is a [STROBE](#)-based transcript construction for zero-knowledge proofs. It automates the Fiat-Shamir transform, so that by using Merlin, non-interactive protocols can be implemented as if they were interactive.

This is significantly easier and less error-prone than performing the transformation by hand, and in addition, it also provides natural support for:

- multi-round protocols with alternating commit and challenge phases;
- natural domain separation, ensuring challenges are bound to the statements to be proved;
- automatic message framing, preventing ambiguous encoding of commitment data;
- and protocol composition, by using a common transcript for multiple protocols.

## Mitigation Idea:

- Declare (to Merlin) protocol flow ahead of time
- Raise flag if this is not followed

## Detection Idea:

- Transcript should contain all objects flowed through both P & V
- If not, raise flag (w/ some error)

Long-term: Standardization of Fiat-Shamir



# Summary

# Summary

Never (,ever, ever) implement weak Fiat-Shamir in practice!

# Summary

**Never** (*,ever, ever*) implement **weak** Fiat-Shamir in practice!

- Hash **everything** (it's not that expensive anyway)

# Summary

**Never** (,ever, ever) implement **weak** Fiat-Shamir in practice!

- Hash **everything** (it's not that expensive anyway)

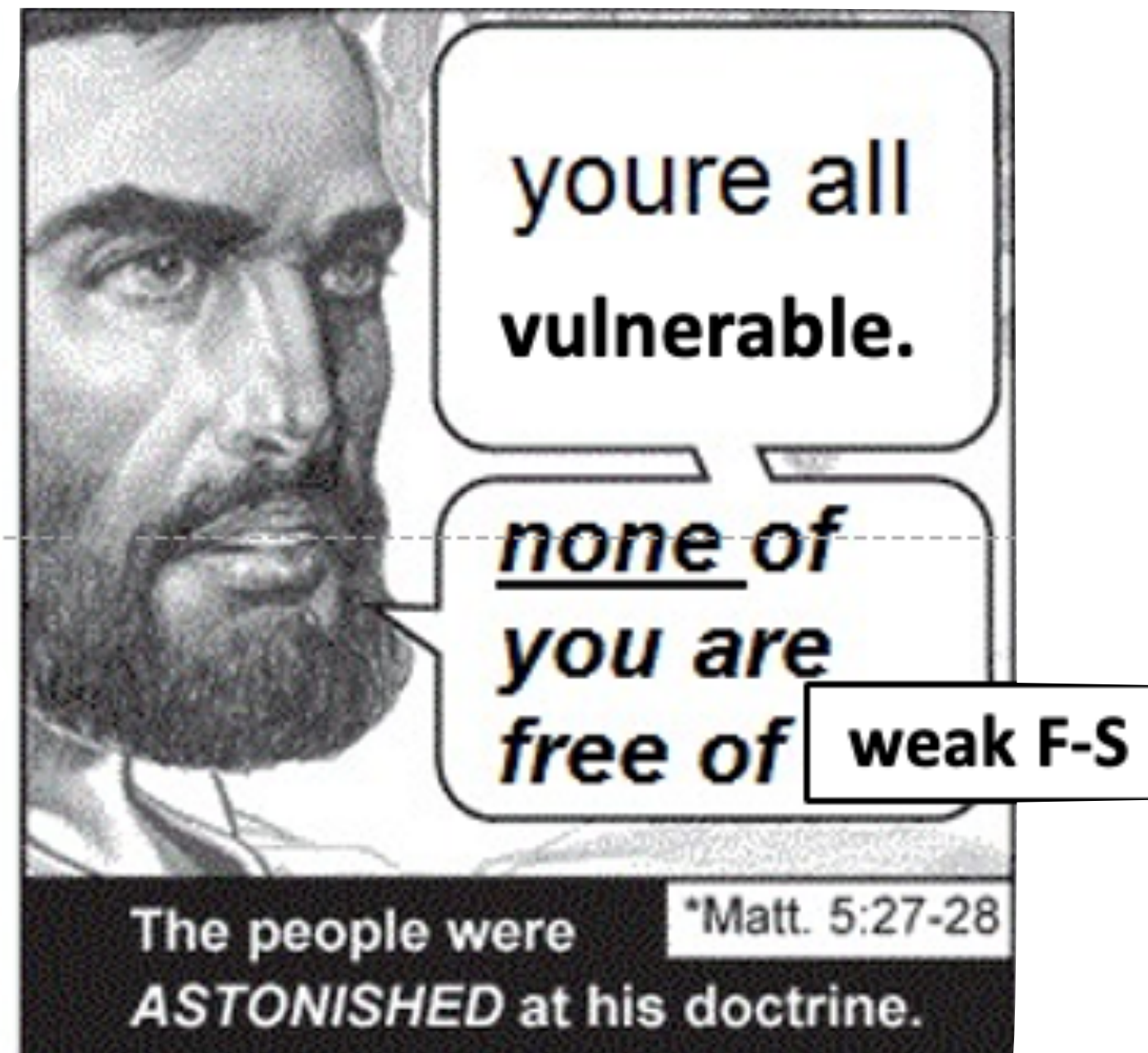
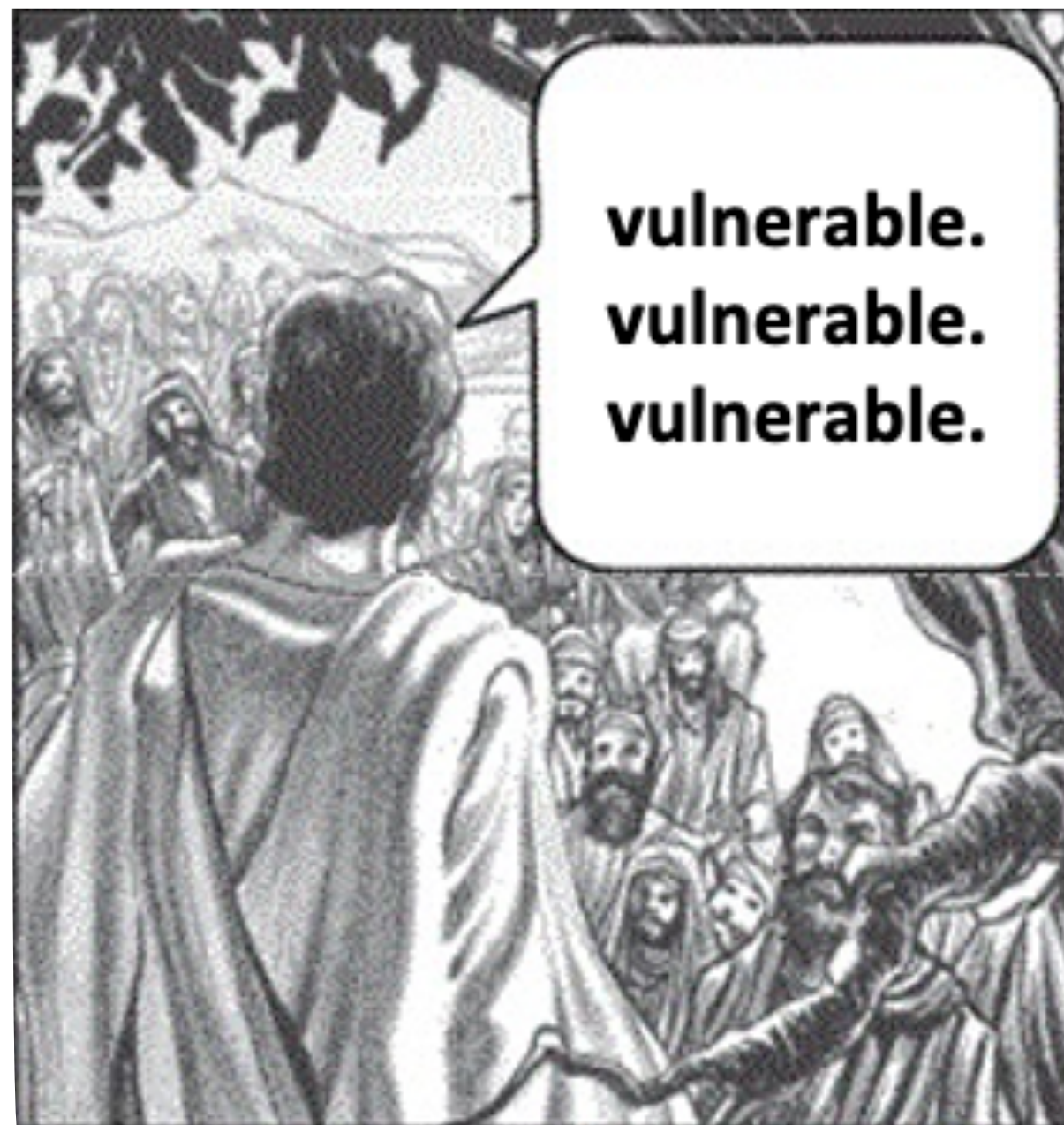
**For Academics:** Specify the **correct** Fiat-Shamir transform!

# Summary

**Never** (,ever, ever) implement **weak** Fiat-Shamir in practice!

- Hash everything (it's not that expensive anyway)

For Academics: Specify the **correct** Fiat-Shamir transform!



Read our paper  
(ePrint 2023/691)



Thank You!