Weak Fiat-Shamir Attacks on Modern Proof Systems

Quang Dao
qvd@andrew.cmu.edu
Carnegie Mellon University

Jim Miller
james.miller@trailofbits.com

Opal Wright
opal.wright@trailofbits.com
Trail of Bits

Paul Grubbs
paulgrub@umich.edu
University of Michigan

Real World Cryptography 2024

Proof Systems and Applications

Proof Systems and 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

STARKWARE

STARK Proof Pioneers

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

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

Regulated And Decentralized Finance

Espresso helps rollups:

Ethereum's First zkRollup Layer 2

Ethereum, ncrypted

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

Aleo

era

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

The Native zkEVM Scal Solution for Ethereu

Scroll is a zkEVM-based zkRollup on Ethereum that ena native compatibility for existing Ethereum applications an

Mina is building the p security layer for web knowledge proofs.

The ZK Coprocessor for Ethereum

Proof Systems and 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

Regulated And

STARK Pr oof Pioneers

Bringing s calability, s

privacy to a blockchain near you

Do implementations of proof systems match their theoretical security?

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

Ethereum's First zkRollup Layer 2

ncrypted

Aleo

era

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

The Native zkEVM Scal Solution for Ethereu

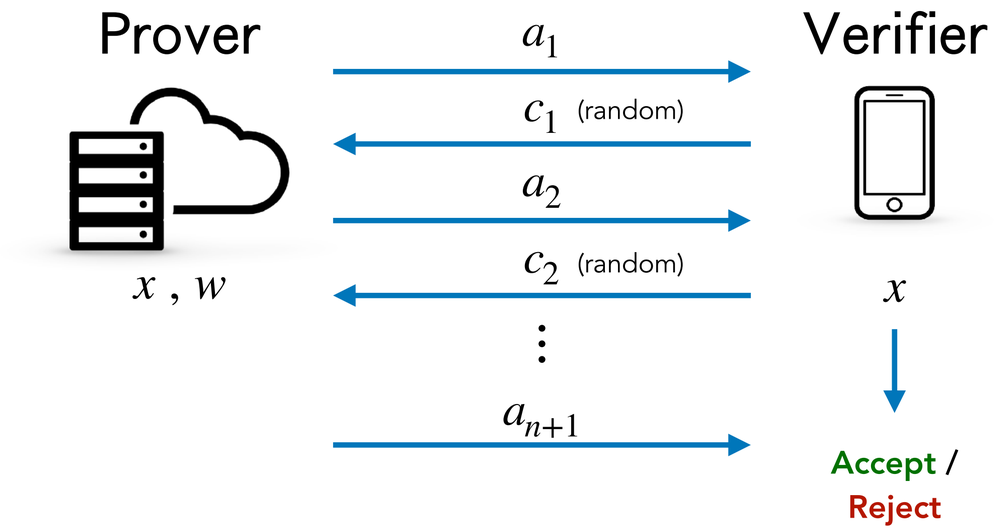
Scroll is a zkEVM-based zkRollup on Ethereum that ena native compatibility for existing Ethereum applications an

Mina is building the p security layer for web knowledge proofs.

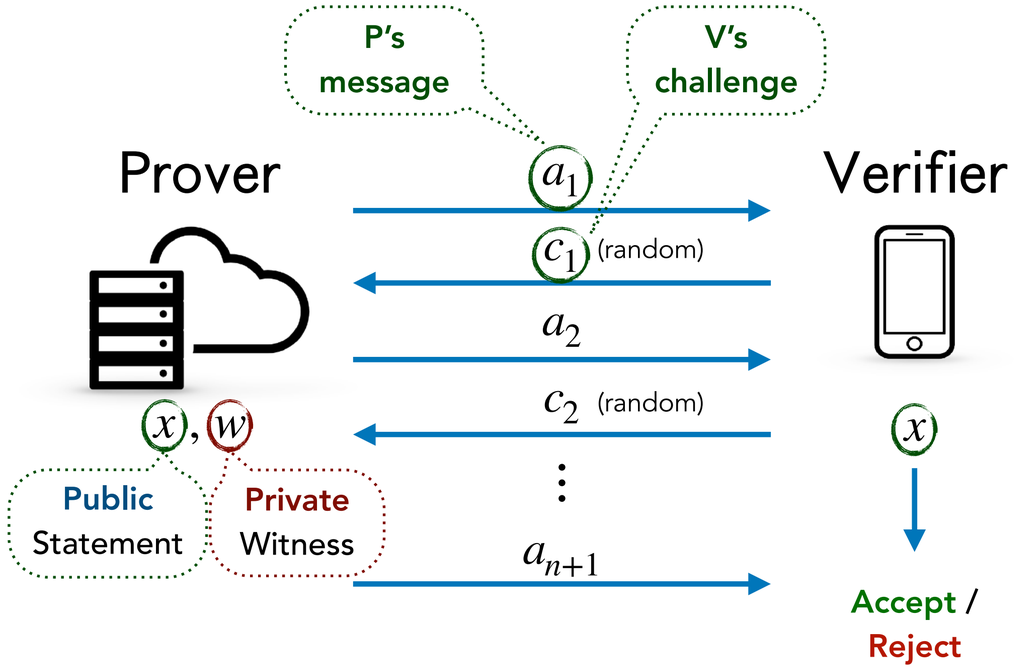
The ZK Coprocessor for Ethereum

Proof Systems from Fiat-Shamir

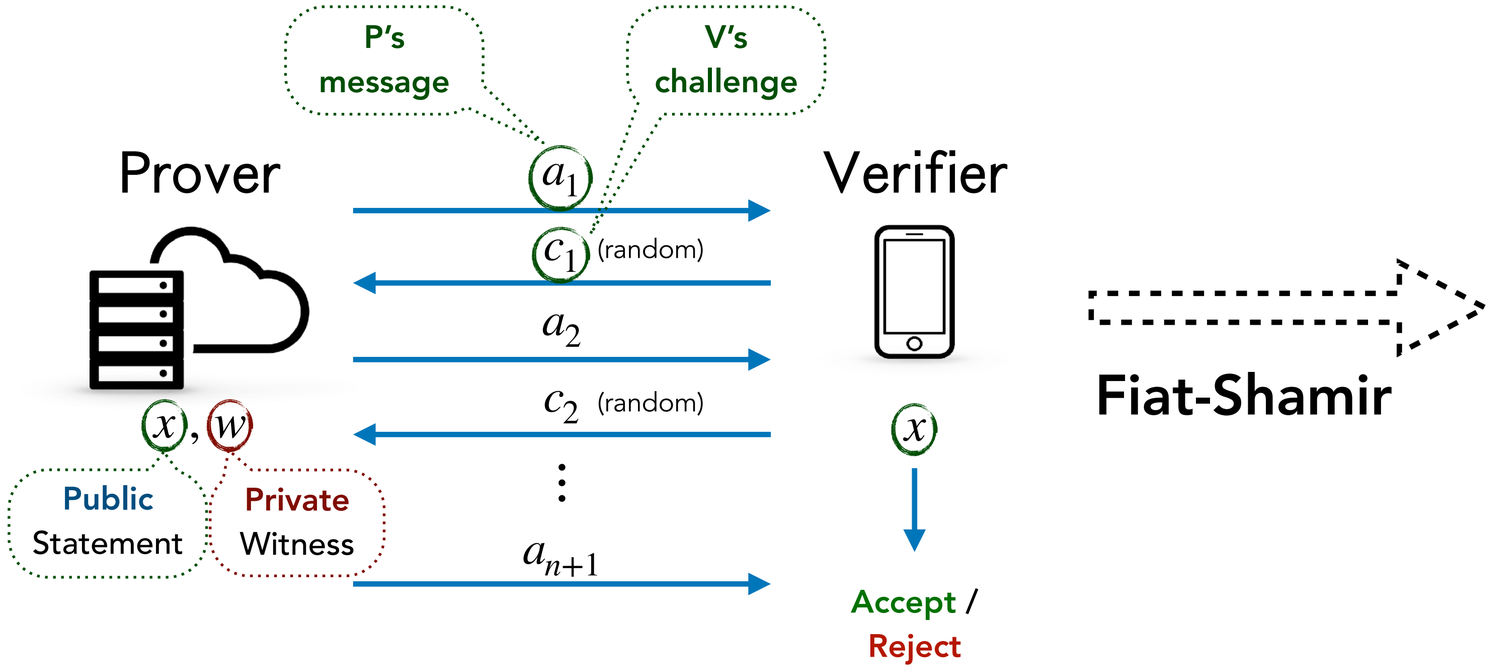
Proof Systems from Fiat-Shamir



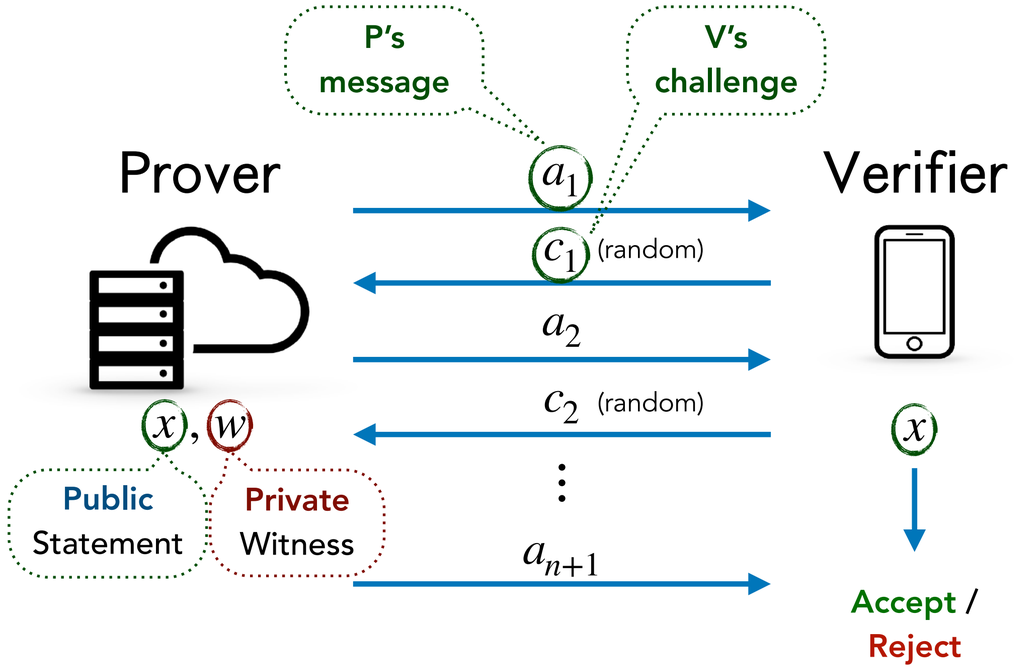
Proof Systems from Fiat-Shamir

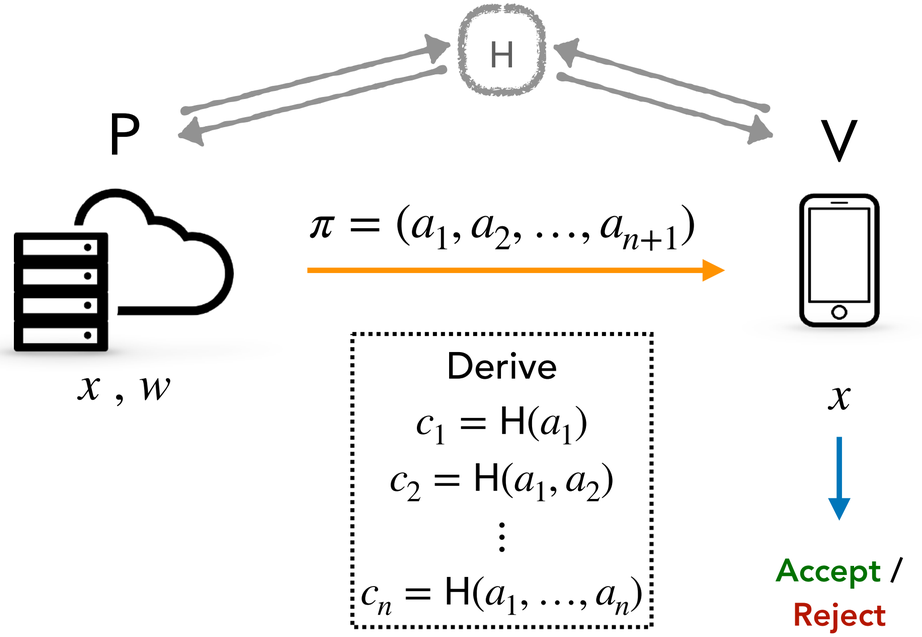


Proof Systems from Fiat-Shamir

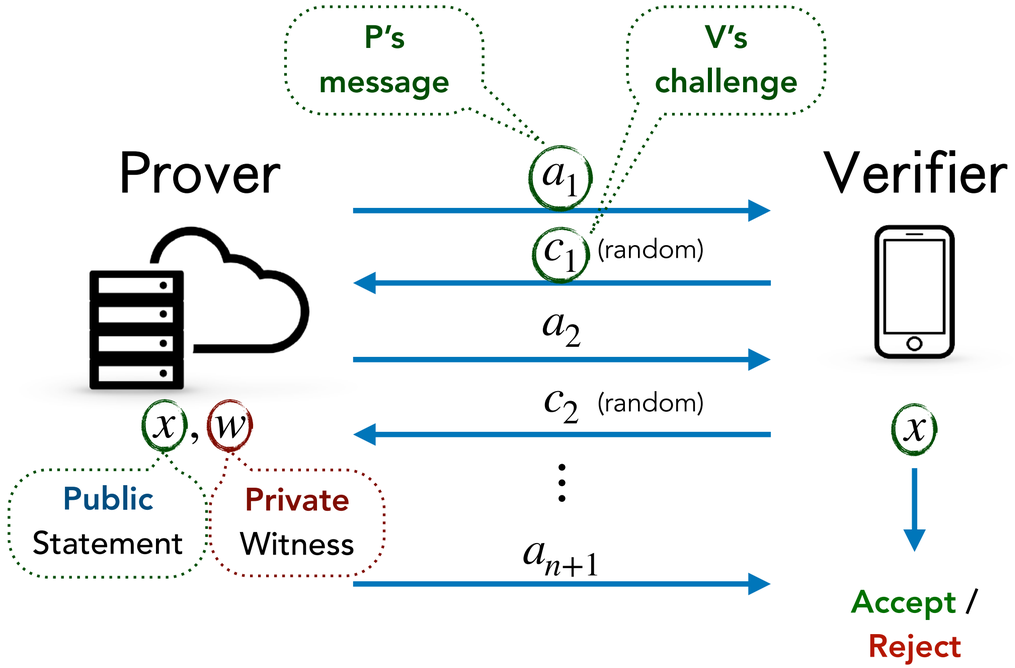


Proof Systems from Fiat-Shamir

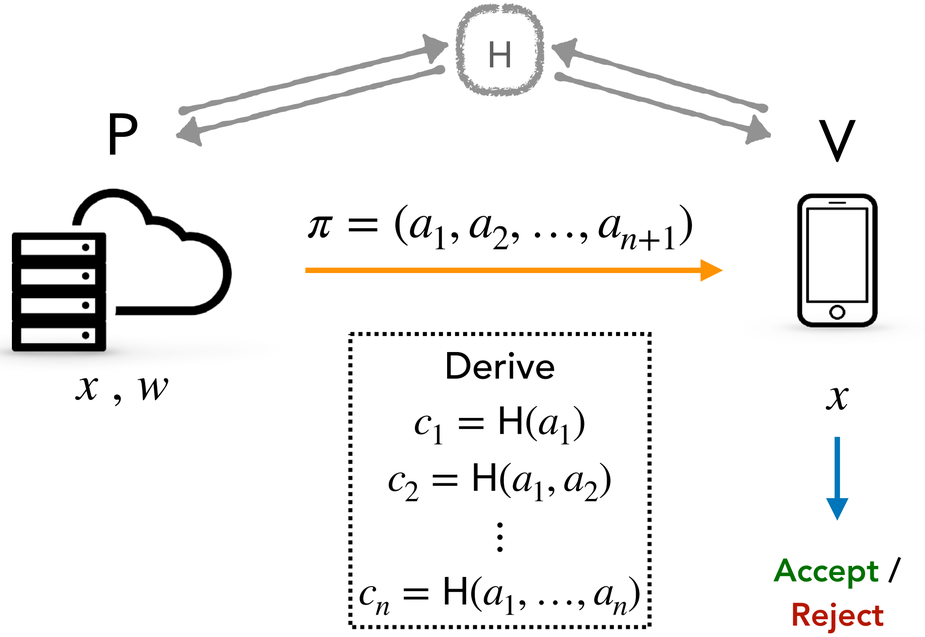




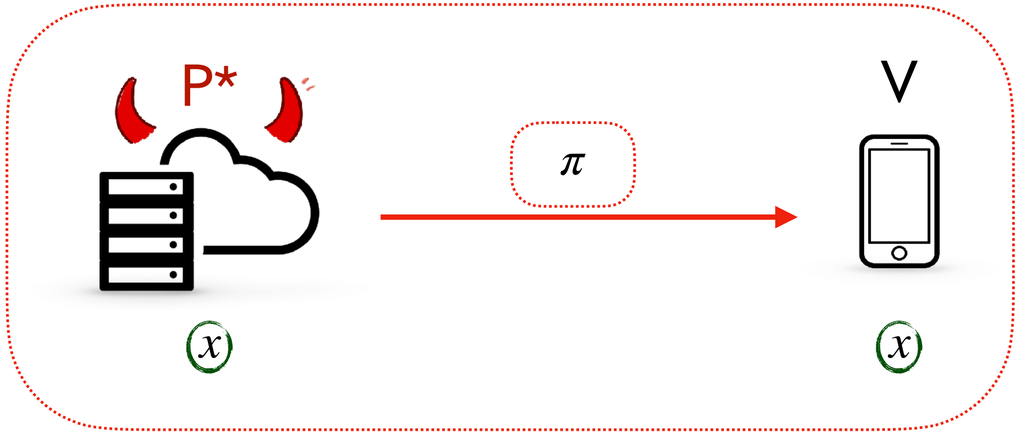
Proof Systems from Fiat-Shamir



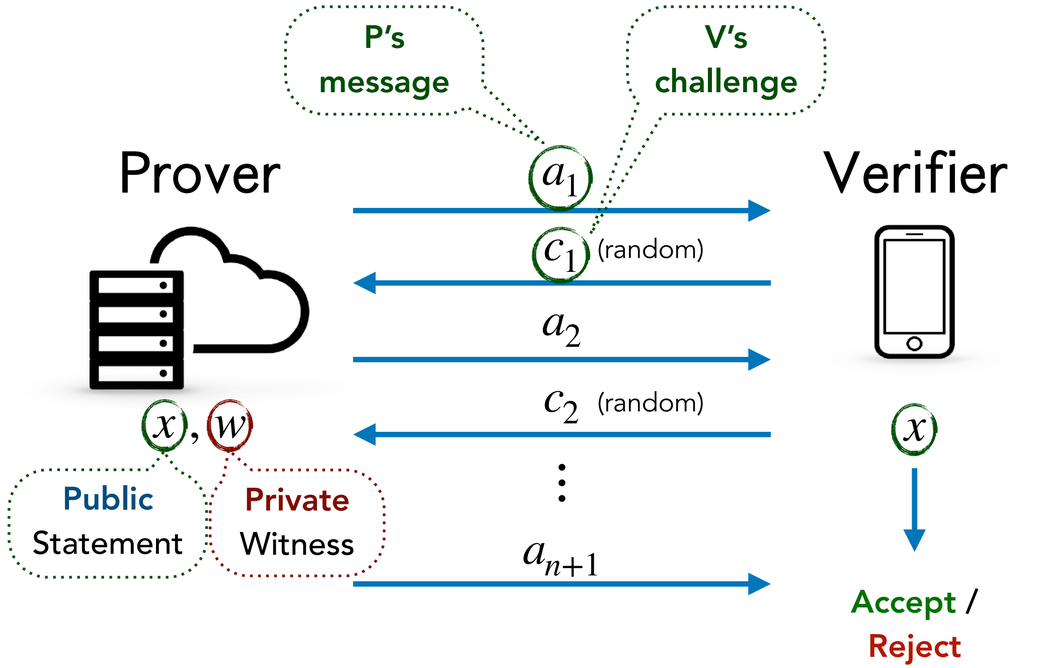
Fiat-Shamir

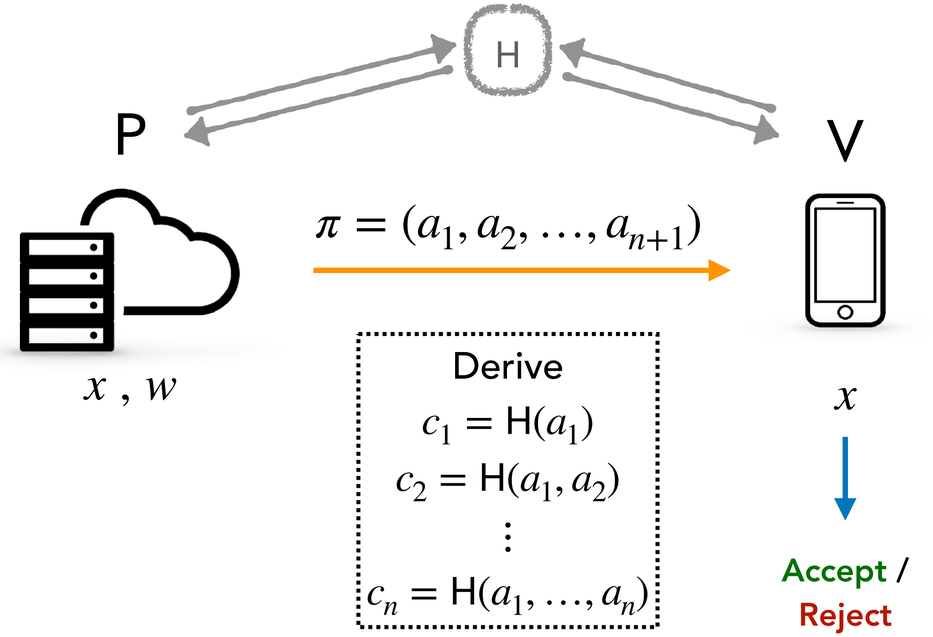


Security:



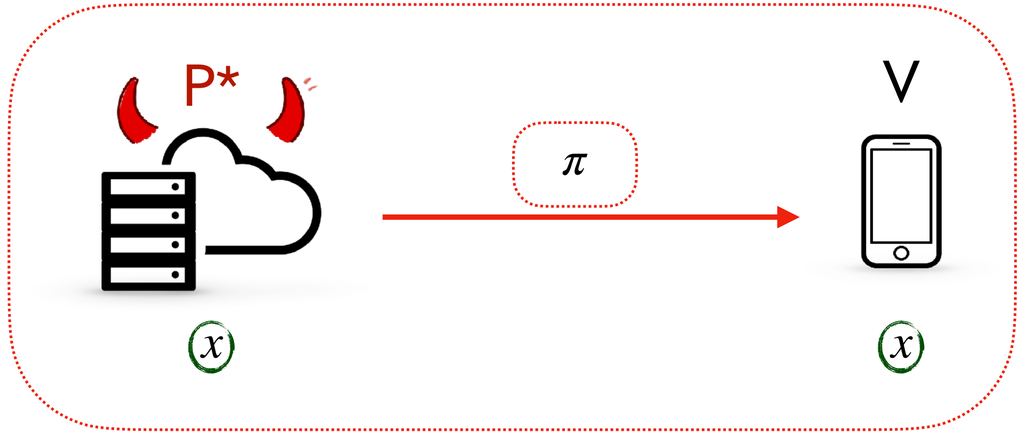
Proof Systems from Fiat-Shamir



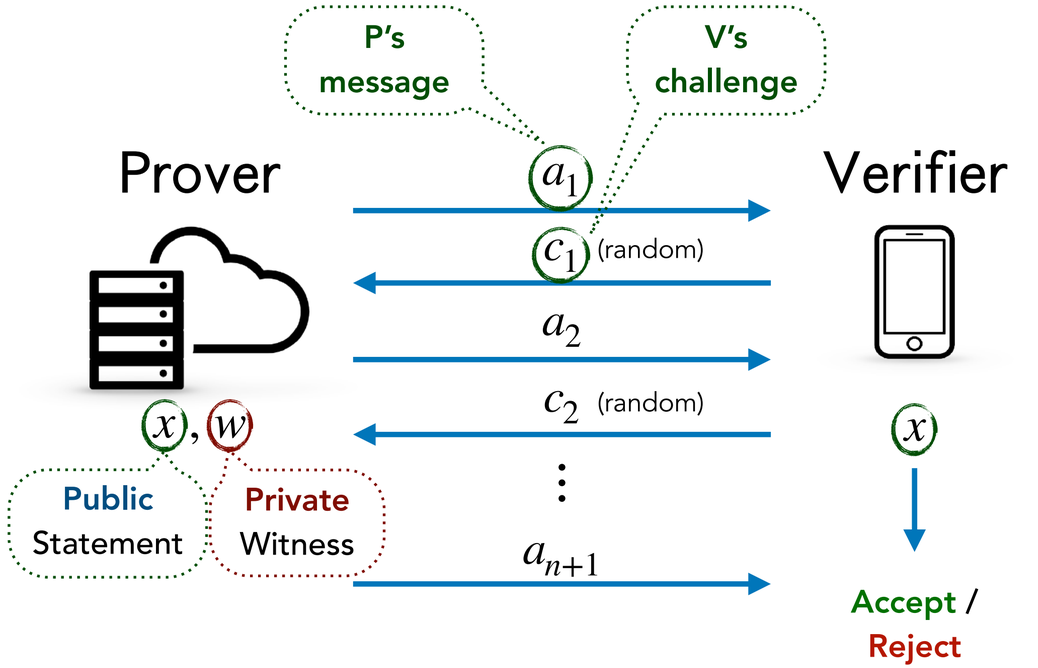


Security:

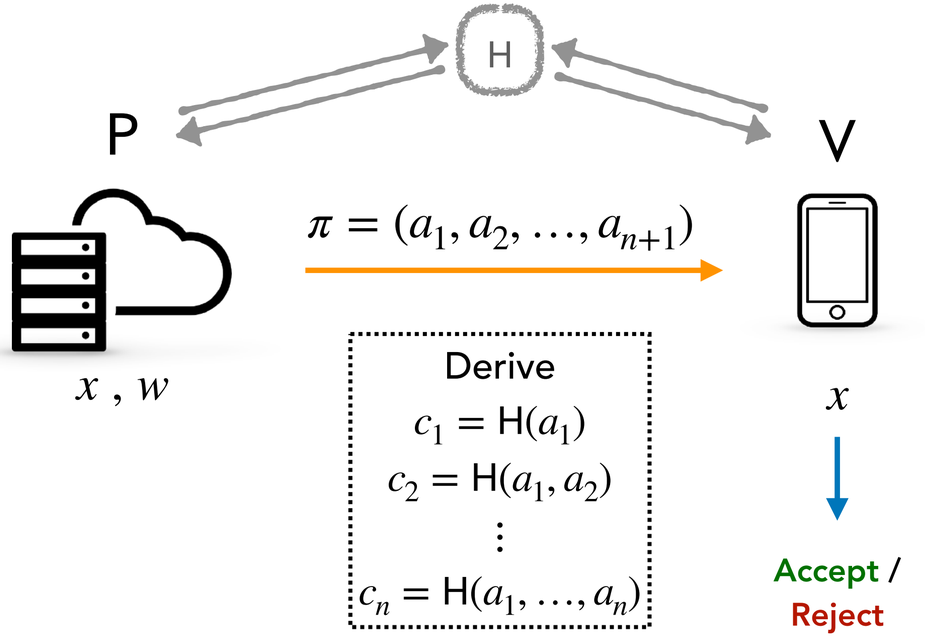
Soundness: If x has no w, then V rejects.



Proof Systems from Fiat-Shamir



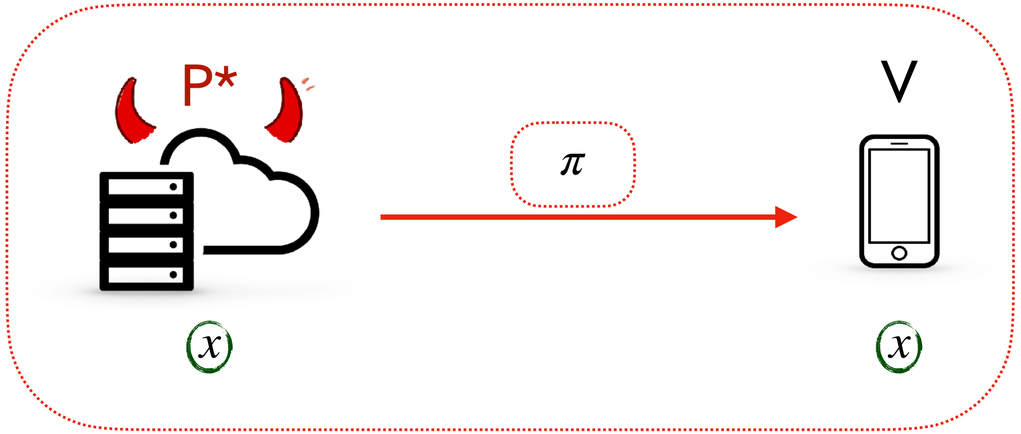
Fiat-Shamir



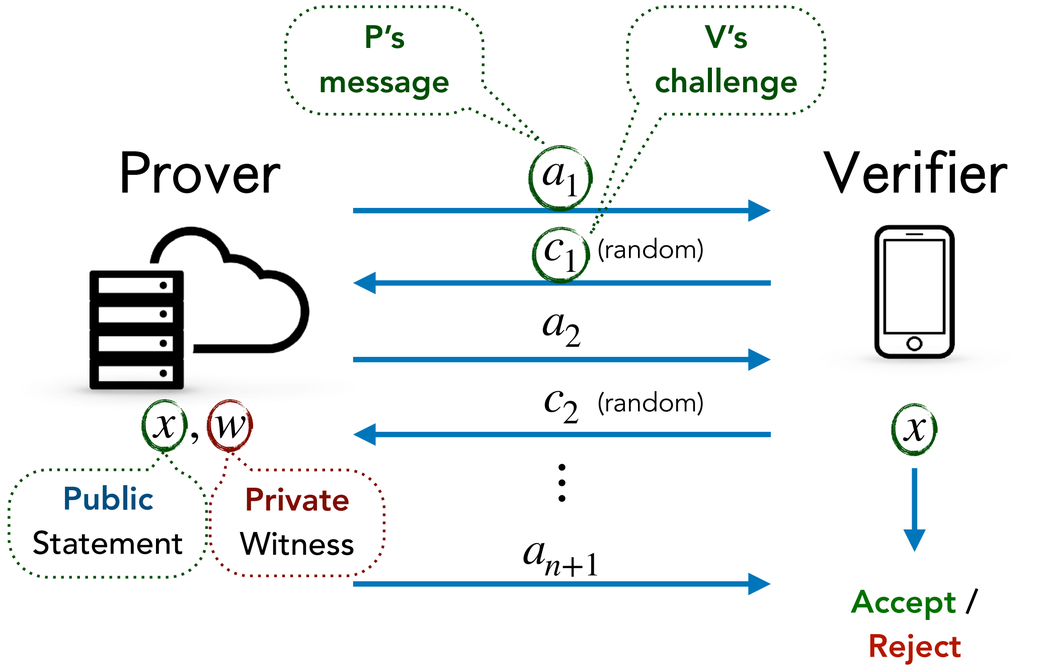
Security:

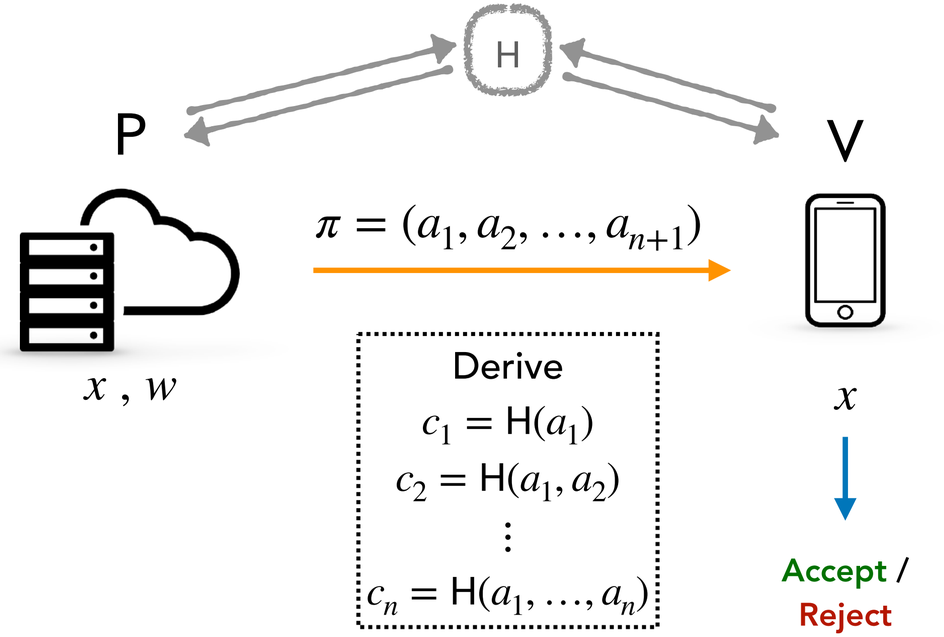
Soundness: If x has no w, then V rejects.

Knowledge Soundness: If V accepts, then P* must "know" w.



Proof Systems from Fiat-Shamir

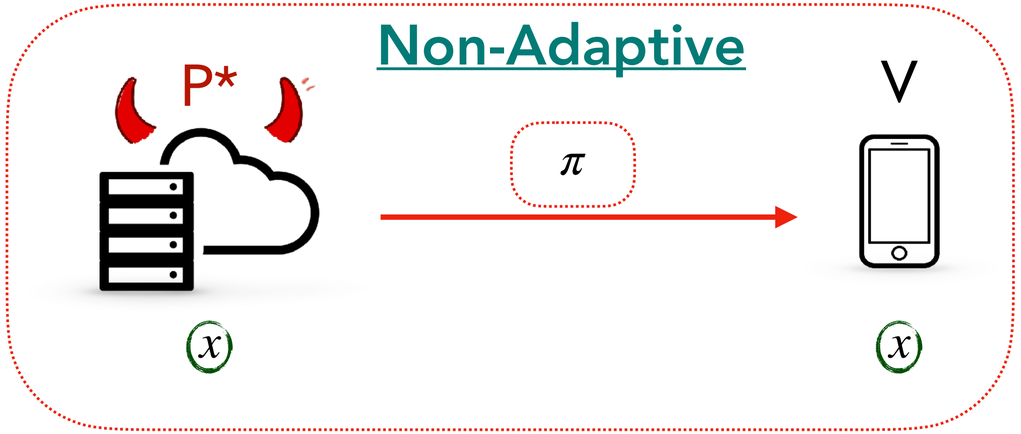




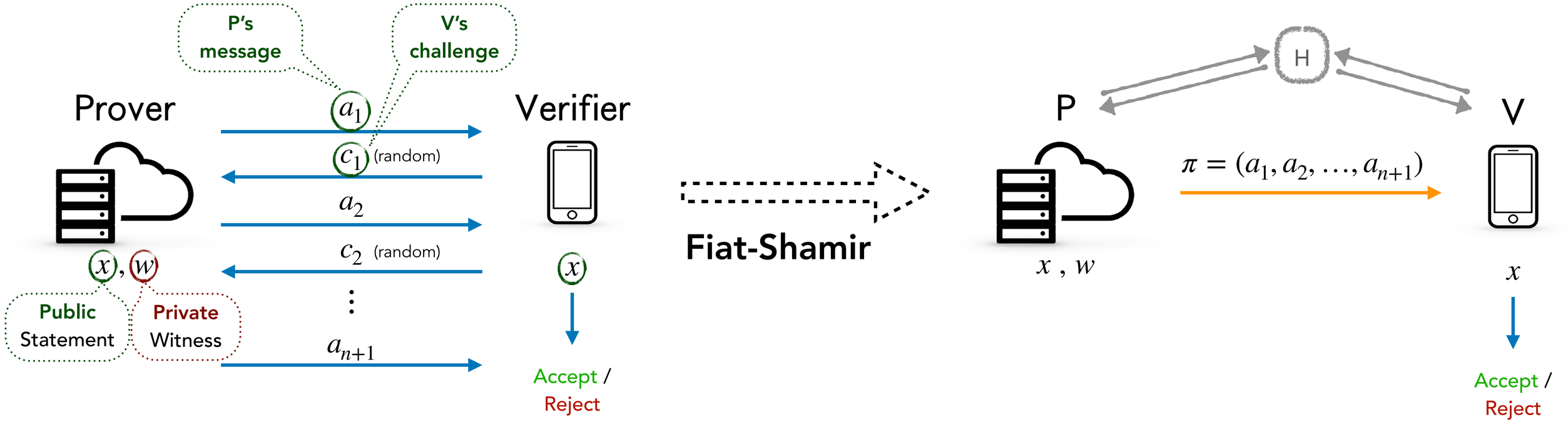
Security:

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

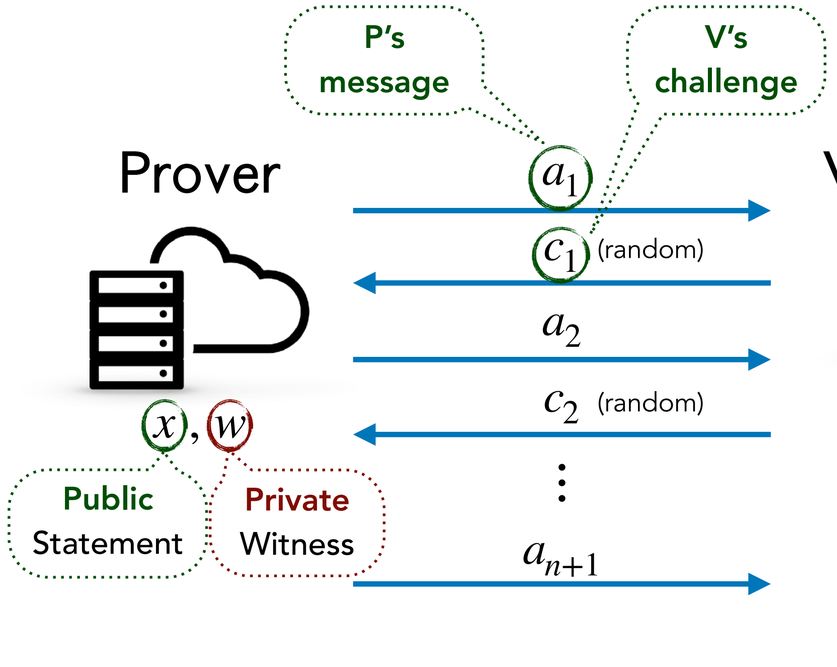


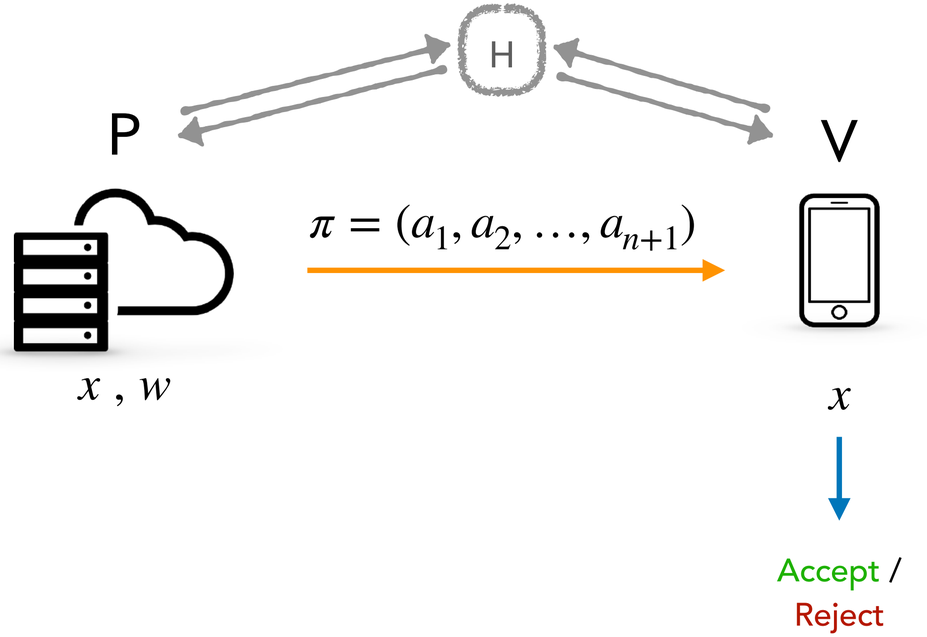
Security:

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

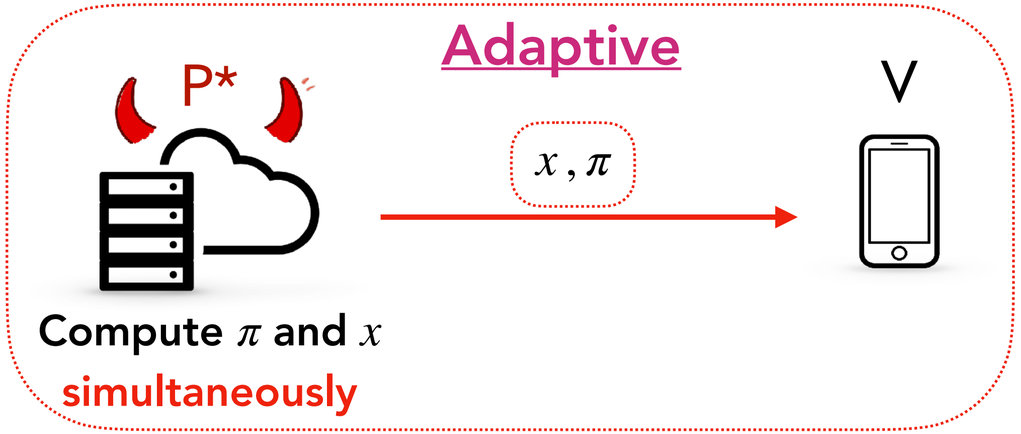




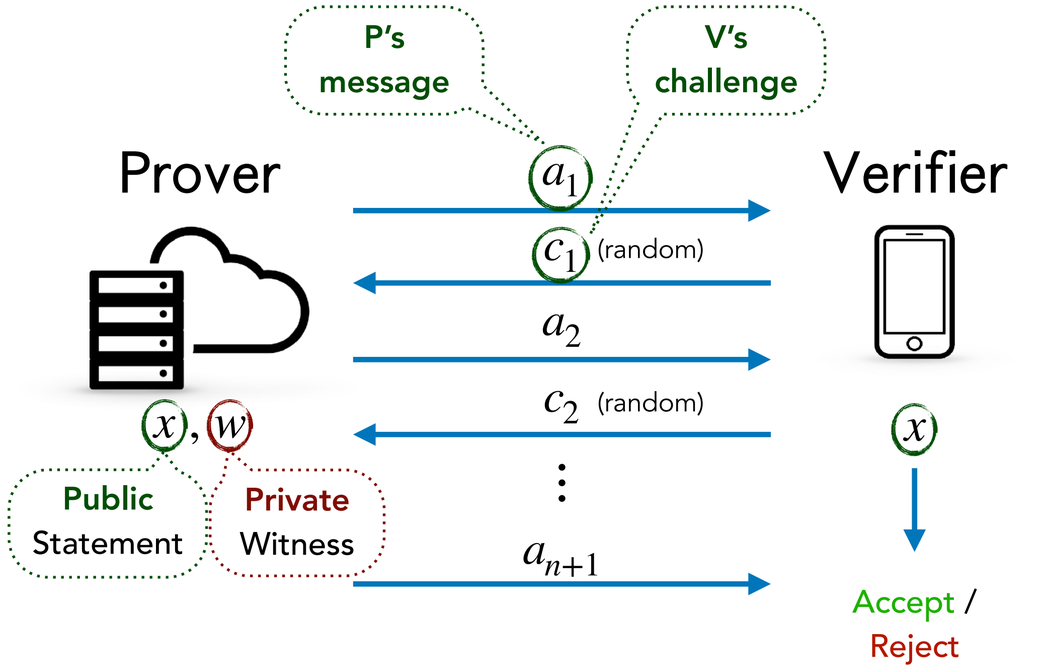
Security:

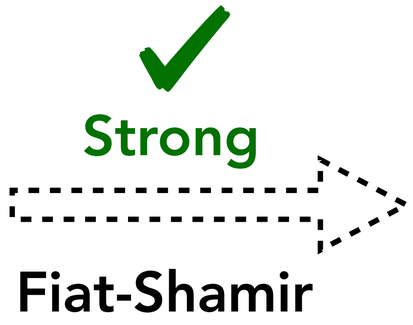
Soundness: If x has no w, then V rejects.

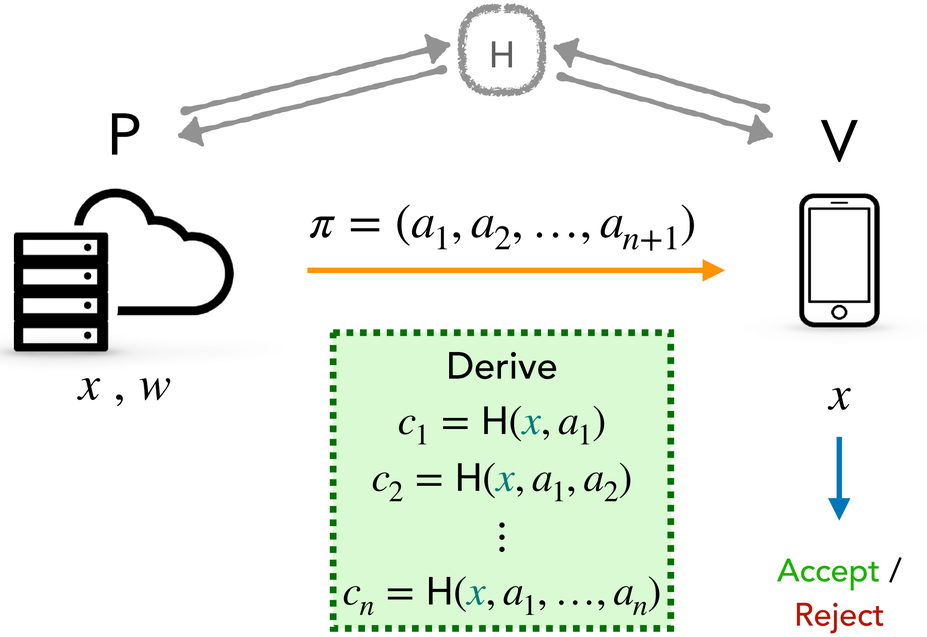
Knowledge Soundness: If V accepts, then P* must "know" w.



Strong Fiat-Shamir for Adaptive Security



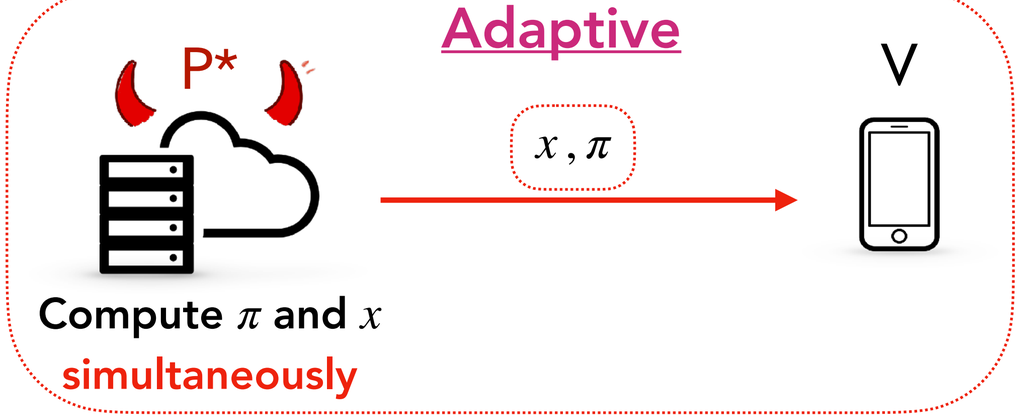




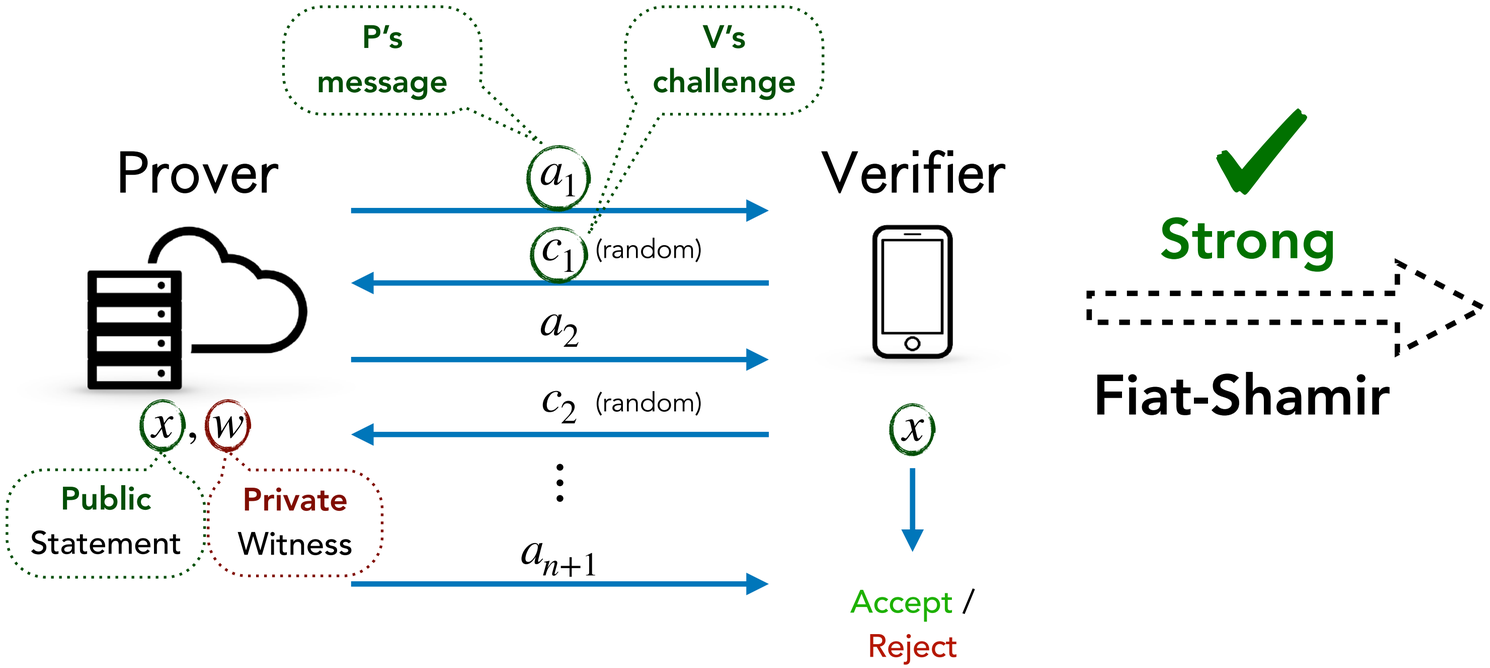
Security:

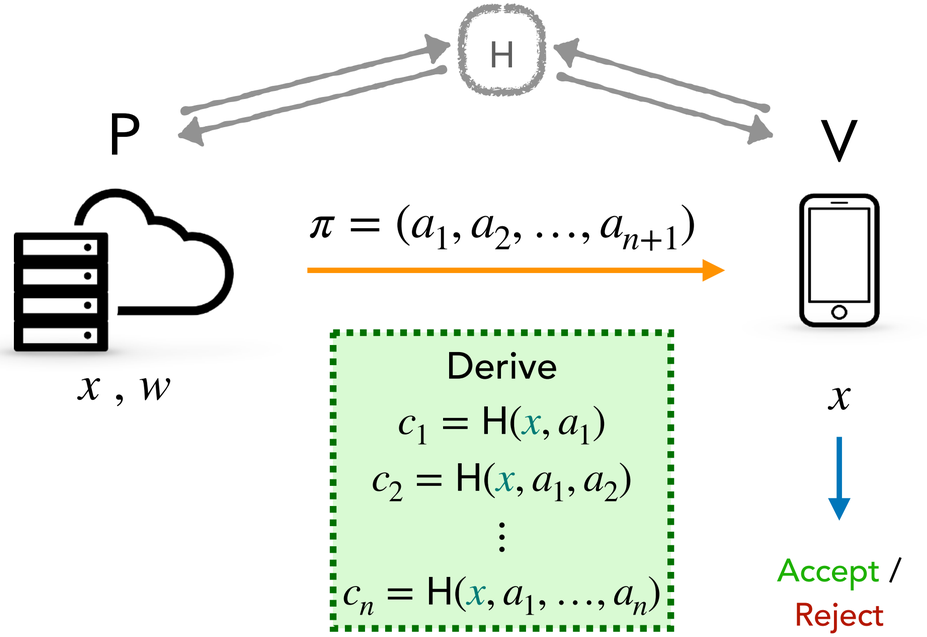
Soundness: If x has no w, then V rejects.

Knowledge Soundness: If V accepts, then P* must "know" w.



Strong Fiat-Shamir for Adaptive Security

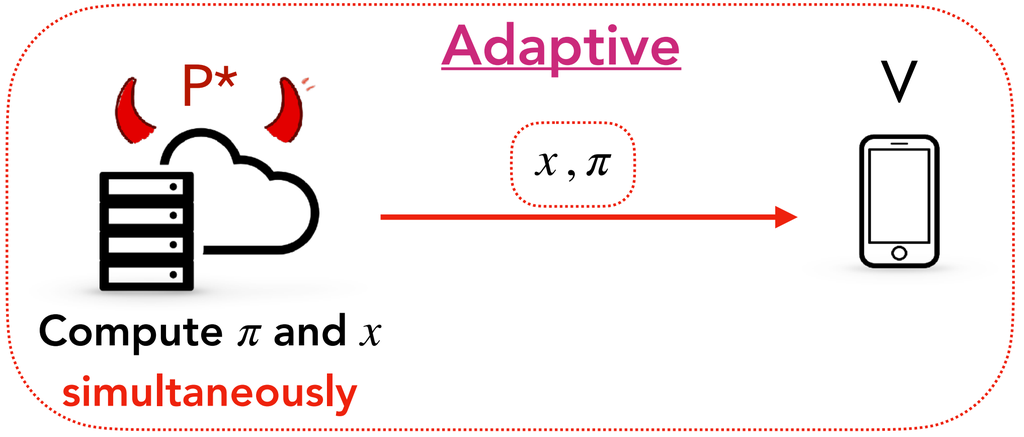




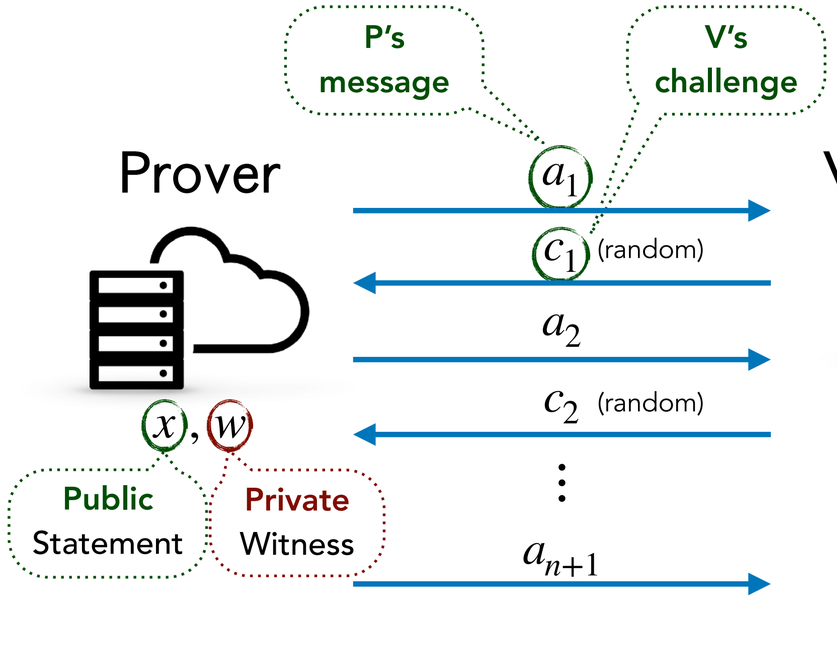
Security:

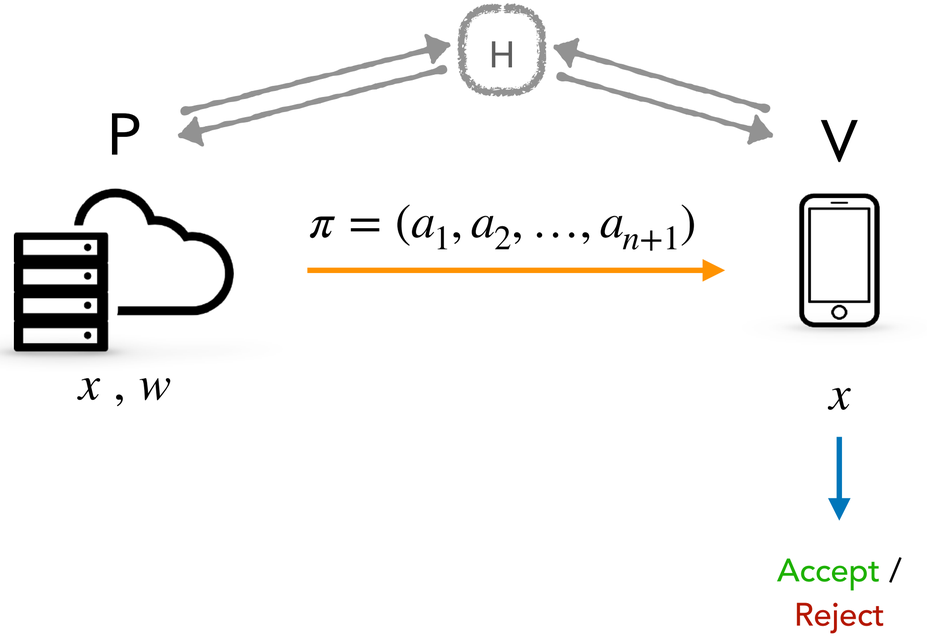
Soundness: If x has no w, then V rejects. ✓

Knowledge Soundness: If V accepts, then P* must "know" w. ✓



Weak Fiat-Shamir and Attacks

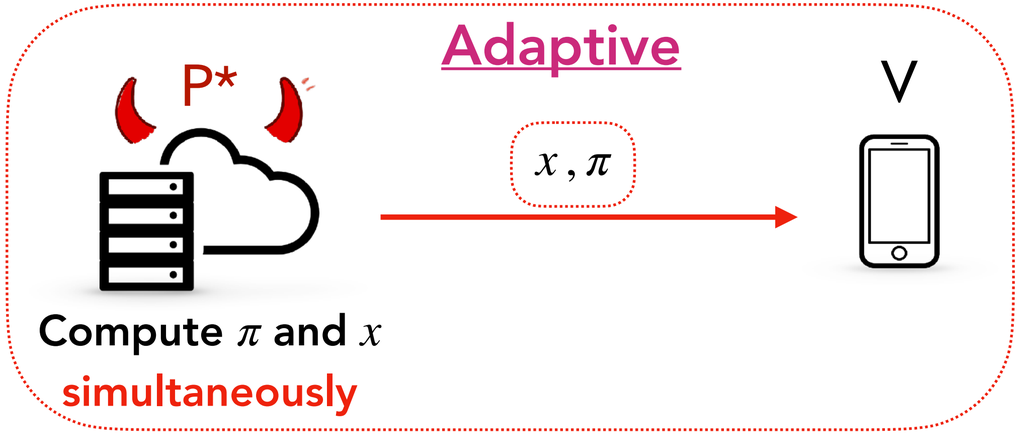




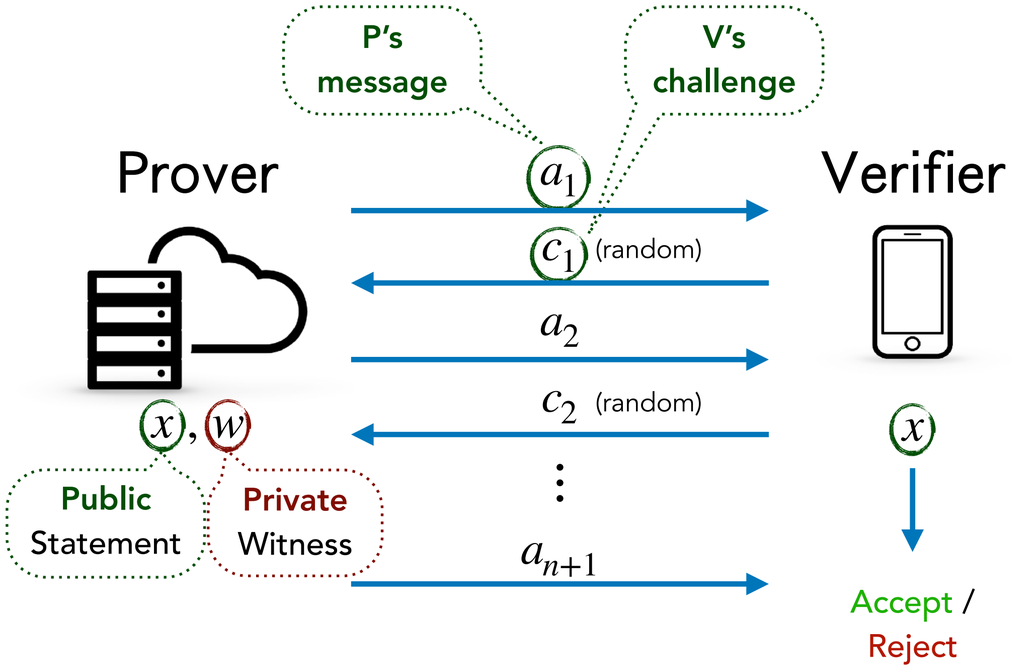
Security:

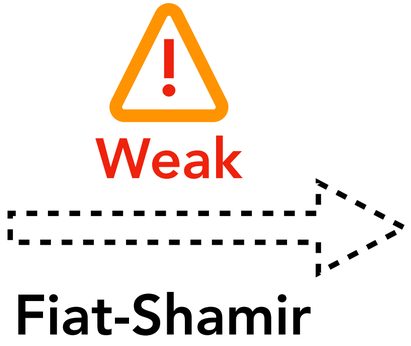
Soundness: If x has no w, then V rejects.

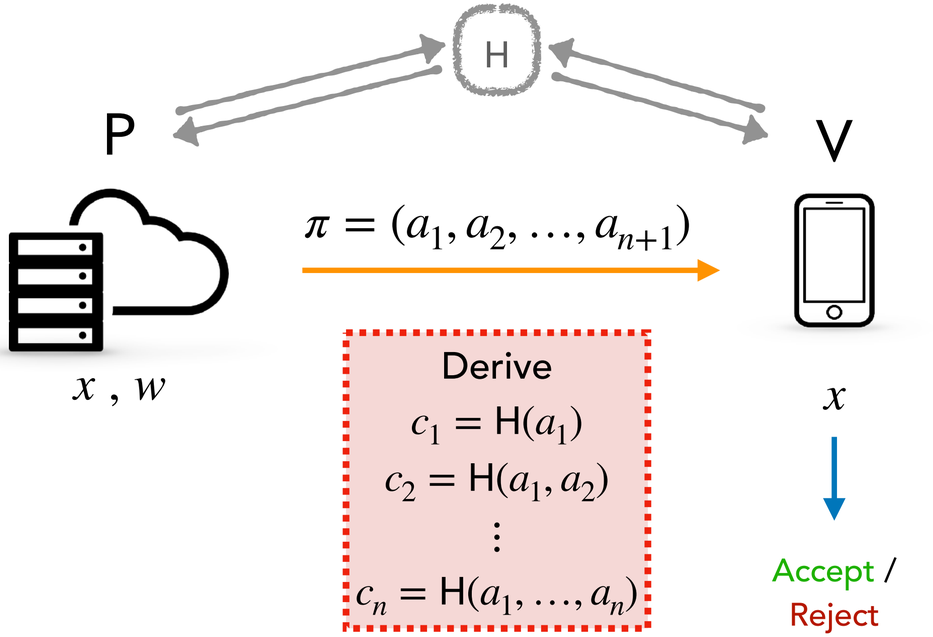
Knowledge Soundness: If V accepts, then P* must "know" w.



Weak Fiat-Shamir and Attacks



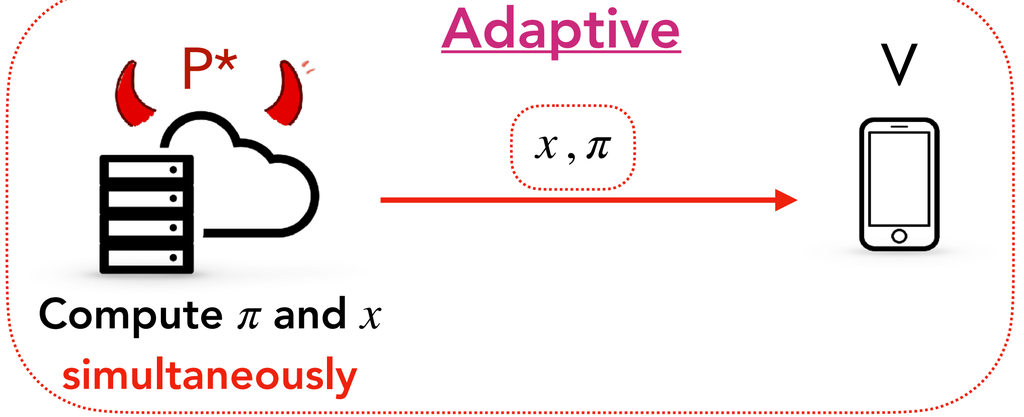




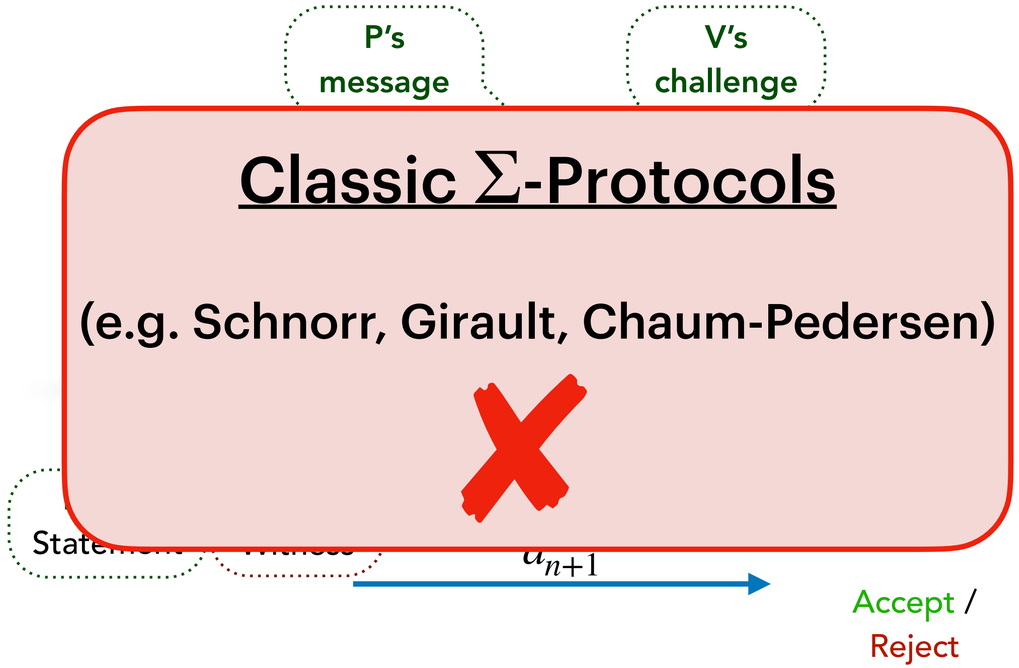
Security:

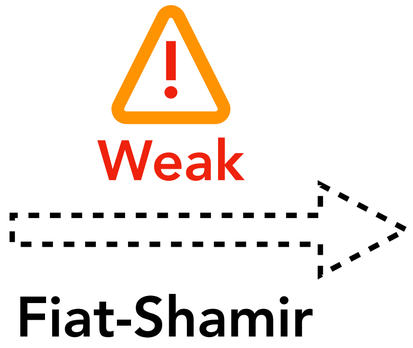
Soundness: If x has no w, then V rejects.

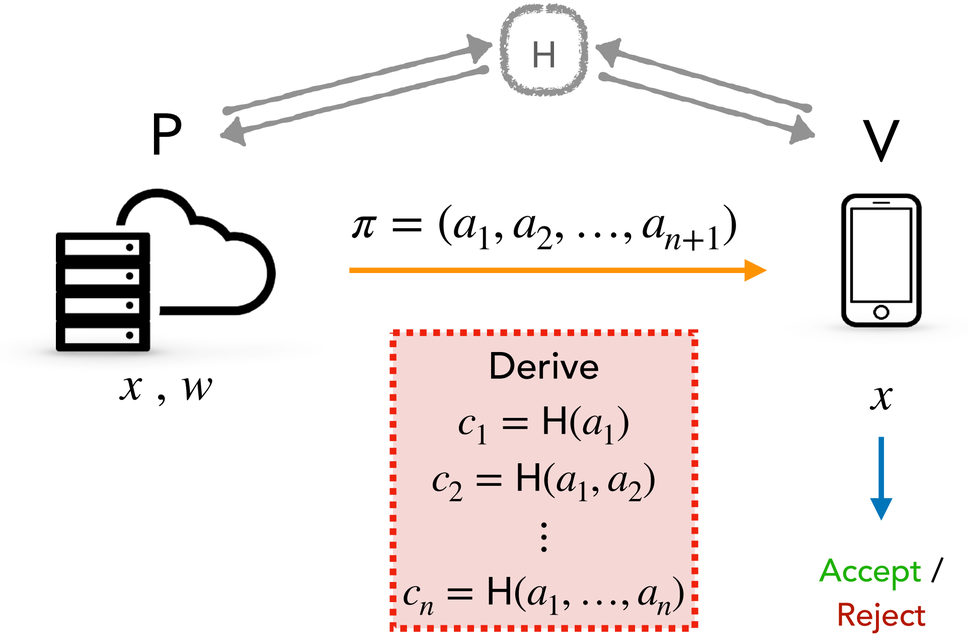
Knowledge Soundness: If V accepts, then P* must "know" w.



Weak Fiat-Shamir and Attacks



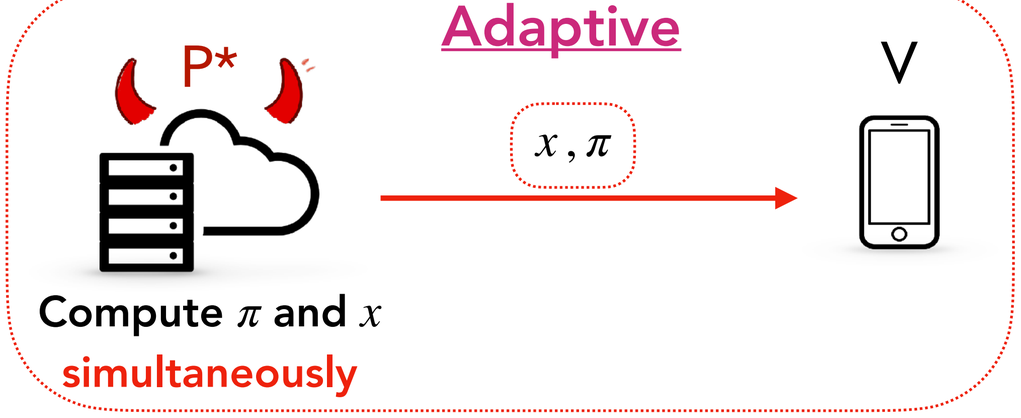




Security:

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

Knowledge Soundness: If V accepts, then P* must "know" w.



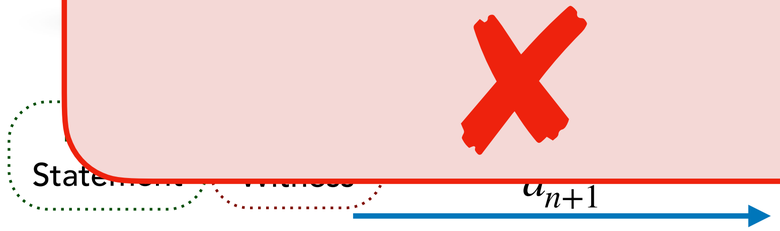
Weak Fiat-Shamir and Attacks

P's message

V's challenge

Classic Σ-Protocols

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

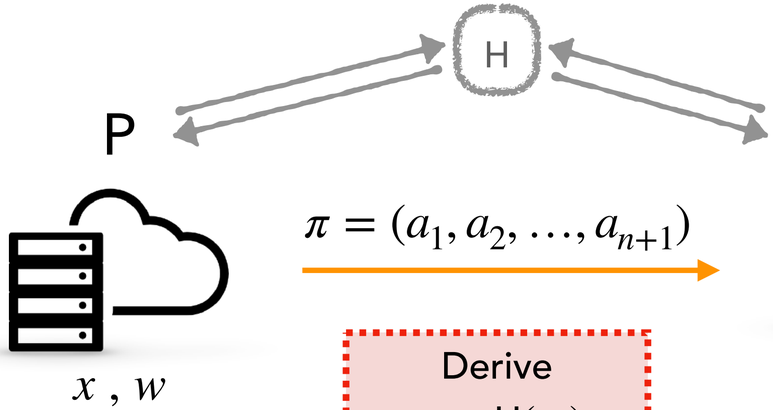


a_{n+1}

Accept /

Weak

Fiat-Shamir



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

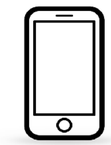
David Bernhard[1], Olivier Pereira[2], and Bogdan Warinschi[1]

n

How not to prove your election outcome

Thomas Haines*, Sarah Jamie Lewis†, Olivier Pereira‡, and Vanessa Teague§

Adaptive



x , π

Compute π and x simultaneously

Weak Fiat-Shamir and Attacks

P's message

V's challenge

Classic Σ-Protocols

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

✗

Weak

Fiat-Shamir

H

Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)

?

$c_n = \mathsf{H}(a_1, \ldots, a_n)$

Accept / Reject

a_{n+1}

Accept /

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

David Bernhard[1], Olivier Pereira[2], and Bogdan Warinschi[1]

✗ / ?

n

✗ / ?

How not to prove your election outcome

Thomas Haines*, Sarah Jamie Lewis†, Olivier Pereira‡, and Vanessa Teague§

Adaptive

P*

V

x , π

Compute π and x simultaneously

Weak Fiat-Shamir and Attacks

P's message

V's challenge

Classic Σ-Protocols

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

✗

Weak

Fiat-Shamir

Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)

?

1. *Are modern proof systems insecure under weak Fiat-Shamir?*

Weak Fiat-Shamir and Attacks

P's message

V's challenge

Classic Σ-Protocols

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

Weak

Fiat-Shamir

Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)

?

1. *Are modern proof systems insecure under weak Fiat-Shamir?*
2. *Are these vulnerabilities present in implementations?*

Weak Fiat-Shamir and Attacks

P's message

V's challenge

Classic Σ-Protocols

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

✗

Weak

Fiat-Shamir

Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)

?

1. *Are modern proof systems insecure under weak Fiat-Shamir?*
2. *Are these vulnerabilities present in implementations?*
3. *Do they lead to attacks on larger/surrounding protocols?*

Weak Fiat-Shamir and Attacks

P's message

V's challenge

Classic Σ-Protocols

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

✗

Weak

Fiat-Shamir

H

Modern Proof Systems

(e.g. Bulletproofs, Plonk, Spartan)

?

1. *Are modern proof systems insecure under weak Fiat-Shamir?*
2. *Are these vulnerabilities present in implementations?*
3. *Do they lead to attacks on larger/surrounding protocols?*
4. *What are the consequences of these attacks?*

Our Contributions

Our Contributions

1. **Survey** of **75+** implementations:

   36 weak F-S vulnerabilities across

   12 different proof systems.

Proof System	Codebase	Weak F-S?
Bulletproofs [22]	bp-go [88]	✓
	bulletproof-js [2]	✓
	simple-bulletproof-js [84]	✓
	BulletproofSwift [20]	✓
	python-bulletproofs [79]	✓
	adjoint-bulletproofs [3]	✓
	zkSen [99]	✓
	incognito-chain [52]	✓♦
	encoins-bulletproofs [33]	✓♦
	ZenGo-X [97]	✓♦
	zkrp [53]	✓♦
	ckb-zkp [82]	✓♦
	bulletproofsrb [21]	✓♦
	monero [69]	✗
	dalek-bulletproofs [29]	✗
	secp256k1-zkp [76]	✗
	bulletproofs-ocaml [75]	✗
	tari-project [86]	✗
	Litecoin [60]	✗
	Grin [44]	✗
Bulletproofs variant [40]	dalek-bulletproofs [29]	✓♦
	cpp-lwevss [61]	✗
Sonic [62]	ebfull-sonic [18]	✓
	lx-sonic [59]	✓
	iohk-sonic [54]	✗
	adjoint-sonic [4]	✗
Schnorr [80]	noknow-python [7]	✓

Proof System	Codebase	Weak F-S?
Plonk [37]	anoma-plonkup [6]	✓
	gnark [17]	✓♦
	dusk-network [31]	✓♦
	snarkjs [50]	✓♦
	ZK-Garage [98]	✓♦
	plonky [68]	✗
	ckb-zkp [82]	✗
	halo2 [94]	✗
	o1-labs [72]	✗
	jellyfish [34]	✗
	matter-labs [63]	✗
	aztec-connect [8]	✗
Wesolowski's VDF [91]	0xProject [1]	✓
	Chia [70]	✓
	Harmony [47]	✓
	POA Network [71]	✓
	IOTA Ledger [55]	✓
	master-thesis-ELTE [48]	✓
Hyrax [90]	ckb-zkp [82]	✓♦
	hyraxZK [49]	✗
Spartan [83]	Spartan [65]	✓♦
	ckb-zkp [82]	✓♦
Libra [92]	ckb-zkp [82]	✓♦
Brakedown [43]	Brakedown [19]	✓
Nova [58]	Nova [64]	✓♦
Gemini [16]	arkworks-gemini [38]	✓♦
Girault [42]	zk-paillier [96]	✓♦

Our Contributions

1. **Survey** of **75+** implementations:

   36 weak F-S vulnerabilities across

   12 different proof systems.

vulnerable

patched

Proof System	Codebase	Weak F-S?
Bulletproofs [22]	bp-go [88]	✓
	bulletproof-js [2]	✓
	simple-bulletproof-js [84]	✓
	BulletproofSwift [20]	✓
	python-bulletproofs [79]	✓
	adjoint-bulletproofs [3]	✓
	zkSen [99]	✓
	incognito-chain [52]	✓♦
	encoins-bulletproofs [33]	✓♦
	ZenGo-X [97]	✓♦
	zkrp [53]	✓♦
	ckb-zkp [82]	✓♦
	bulletproofsrb [21]	✓♦
	monero [69]	✗
	dalek-bulletproofs [29]	✗
	secp256k1-zkp [76]	✗
	bulletproofs-ocaml [75]	✗
	tari-project [86]	✗
	Litecoin [60]	✗
	Grin [44]	✗
Bulletproofs variant [40]	dalek-bulletproofs [29]	✓♦
	cpp-lwevss [61]	✗
Sonic [62]	ebfull-sonic [18]	✓
	lx-sonic [59]	✓
	iohk-sonic [54]	✗
	adjoint-sonic [4]	✗
Schnorr [80]	noknow-python [7]	✓

Proof System	Codebase	Weak F-S?
Plonk [37]	anoma-plonkup [6]	✓
	gnark [17]	✓♦
	dusk-network [31]	✓♦
	snarkjs [50]	✓♦
	ZK-Garage [98]	✓♦
	plonky [68]	✗
	ckb-zkp [82]	✗
	halo2 [94]	✗
	o1-labs [72]	✗
	jellyfish [34]	✗
	matter-labs [63]	✗
	aztec-connect [8]	✗
Wesolowski's VDF [91]	0xProject [1]	✓
	Chia [70]	✓
	Harmony [47]	✓
	POA Network [71]	✓
	IOTA Ledger [55]	✓
	master-thesis-ELTE [48]	✓
Hyrax [90]	ckb-zkp [82]	✓♦
	hyraxZK [49]	✗
Spartan [83]	Spartan [65]	✓♦
	ckb-zkp [82]	✓♦
Libra [92]	ckb-zkp [82]	✓♦
Brakedown [43]	Brakedown [19]	✓
Nova [58]	Nova [64]	✓♦
Gemini [16]	arkworks-gemini [38]	✓♦
Girault [42]	zk-paillier [96]	✓♦

Our Contributions

1. **Survey** of **75+** implementations:

   36 weak F-S vulnerabilities across

   12 different proof systems.

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

   $\Longrightarrow$ **Provable** break of soundness

vulnerable patched

Proof System	Codebase	Weak F-S?
Bulletproofs [22]	bp-go [88]	✓
	bulletproof-js [2]	✓
	simple-bulletproof-js [84]	✓
	BulletproofSwift [20]	✓
	python-bulletproofs [79]	✓
	adjoint-bulletproofs [3]	✓
	zkSen [99]	✓
	incognito-chain [52]	✓♦
	encoins-bulletproofs [33]	✓♦
	ZenGo-X [97]	✓♦
	zkrp [53]	✓♦
	ckb-zkp [82]	✓♦
	bulletproofsrb [21]	✓♦
	monero [69]	✗
	dalek-bulletproofs [29]	✗
	secp256k1-zkp [76]	✗
	bulletproofs-ocaml [75]	✗
	tari-project [86]	✗
	Litecoin [60]	✗
	Grin [44]	✗
Bulletproofs variant [40]	dalek-bulletproofs [29]	✓♦
	cpp-lwevss [61]	✗
Sonic [62]	ebfull-sonic [18]	✓
	lx-sonic [59]	✓
	iohk-sonic [54]	✗
	adjoint-sonic [4]	✗
Schnorr [80]	noknow-python [7]	✓

Proof System	Codebase	Weak F-S?
Plonk [37]	anoma-plonkup [6]	✓
	gnark [17]	✓♦
	dusk-network [31]	✓♦
	snarkjs [50]	✓♦
	ZK-Garage [98]	✓♦
	plonky [68]	✗
	ckb-zkp [82]	✗
	halo2 [94]	✗
	o1-labs [72]	✗
	jellyfish [34]	✗
	matter-labs [63]	✗
	aztec-connect [8]	✗
Wesolowski's VDF [91]	0xProject [1]	✓
	Chia [70]	✓
	Harmony [47]	✓
	POA Network [71]	✓
	IOTA Ledger [55]	✓
	master-thesis-ELTE [48]	✓
Hyrax [90]	ckb-zkp [82]	✓♦
	hyraxZK [49]	✗
Spartan [83]	Spartan [65]	✓♦
	ckb-zkp [82]	✓♦
Libra [92]	ckb-zkp [82]	✓♦
Brakedown [43]	Brakedown [19]	✓
Nova [58]	Nova [64]	✓♦
Gemini [16]	arkworks-gemini [38]	✓♦
Girault [42]	zk-paillier [96]	✓♦

Our Contributions

1. **Survey** of **75+** implementations:
   36 weak F-S vulnerabilities across
   12 different proof systems.

2. **Explicit Attacks** against ***Bulletproofs***, ***Plonk***, ***Spartan***, and ***Wesolowski's VDF***:
   $\Longrightarrow$ **Provable** break of soundness

3. **Practical Impacts:** ***unlimited*** currency minting in two blockchain protocols

vulnerable

patched

Proof System	Codebase	Weak F-S?
Bulletproofs [22]	bp-go [88]	✓
	bulletproof-js [2]	✓
	simple-bulletproof-js [84]	✓
	BulletproofSwift [20]	✓
	python-bulletproofs [79]	✓
	adjoint-bulletproofs [3]	✓
	zkSen [99]	✓
	incognito-chain [52]	✓♦
	encoins-bulletproofs [33]	✓♦
	ZenGo-X [97]	✓♦
	zkrp [53]	✓♦
	ckb-zkp [82]	✓♦
	bulletproofsrb [21]	✓♦
	monero [69]	✗
	dalek-bulletproofs [29]	✗
	secp256k1-zkp [76]	✗
	bulletproofs-ocaml [75]	✗
	tari-project [86]	✗
	Litecoin [60]	✗
	Grin [44]	✗
Bulletproofs variant [40]	dalek-bulletproofs [29]	✓♦
	cpp-lwevss [61]	✗
Sonic [62]	ebfull-sonic [18]	✓
	lx-sonic [59]	✓
	iohk-sonic [54]	✗
	adjoint-sonic [4]	✗
Schnorr [80]	noknow-python [7]	✓

Proof System	Codebase	Weak F-S?
Plonk [37]	anoma-plonkup [6]	✓
	gnark [17]	✓♦
	dusk-network [31]	✓♦
	snarkjs [50]	✓♦
	ZK-Garage [98]	✓♦
	plonky [68]	✗
	ckb-zkp [82]	✗
	halo2 [94]	✗
	o1-labs [72]	✗
	jellyfish [34]	✗
	matter-labs [63]	✗
	aztec-connect [8]	✗
Wesolowski's VDF [91]	0xProject [1]	✓
	Chia [70]	✓
	Harmony [47]	✓
	POA Network [71]	✓
	IOTA Ledger [55]	✓
	master-thesis-ELTE [48]	✓
Hyrax [90]	ckb-zkp [82]	✓♦
	hyraxZK [49]	✗
Spartan [83]	Spartan [65]	✓♦
	ckb-zkp [82]	✓♦
Libra [92]	ckb-zkp [82]	✓♦
Brakedown [43]	Brakedown [19]	✓
Nova [58]	Nova [64]	✓♦
Gemini [16]	arkworks-gemini [38]	✓♦
Girault [42]	zk-paillier [96]	✓♦

Our Contributions

1. **Survey** of **75+** implementations:
   36 weak F-S vulnerabilities across
   12 different proof systems.

2. **Explicit Attacks** against ***Bulletproofs***, ***Plonk***, ***Spartan***, and ***Wesolowski's VDF***:
   $\Longrightarrow$ **Provable** break of soundness

3. **Practical Impacts:** ***unlimited*** currency minting in two blockchain protocols

vulnerable

patched

Proof System	Codebase	Weak F-S?
Bulletproofs [22]	bp-go [88]	✓
	bulletproof-js [2]	✓
	simple-bulletproof-js [84]	✓
	BulletproofSwift [20]	✓
	python-bulletproofs [79]	✓
	adjoint-bulletproofs [3]	✓
	zkSen [99]	✓
	incognito-chain [52]	✓♦
	encoins-bulletproofs [33]	✓♦
	ZenGo-X [97]	✓♦
	zkrp [53]	✓♦
	ckb-zkp [82]	✓♦
	bulletproofsrb [21]	✓♦
	monero [69]	✗
	dalek-bulletproofs [29]	✗
	secp256k1-zkp [76]	✗
	bulletproofs-ocaml [75]	✗
	tari-project [86]	✗
	Litecoin [60]	✗
	Grin [44]	✗
Bulletproofs variant [40]	dalek-bulletproofs [29]	✓♦
	cpp-lwevss [61]	✗
Sonic [62]	ebfull-sonic [18]	✓
	lx-sonic [59]	✓
	iohk-sonic [54]	✗
	adjoint-sonic [4]	✗
Schnorr [80]	noknow-python [7]	✓

Proof System	Codebase	Weak F-S?
Plonk [37]	anoma-plonkup [6]	✓
	gnark [17]	✓♦
	dusk-network [31]	✓♦
	snarkjs [50]	✓♦
	ZK-Garage [98]	✓♦
	plonky [68]	✗
	ckb-zkp [82]	✗
	halo2 [94]	✗
	o1-labs [72]	✗
	jellyfish [34]	✗
	matter-labs [63]	✗
	aztec-connect [8]	✗
Wesolowski's VDF [91]	0xProject [1]	✓
	Chia [70]	✓
	Harmony [47]	✓
	POA Network [71]	✓
	IOTA Ledger [55]	✓
	master-thesis-ELTE [48]	✓
Hyrax [90]	ckb-zkp [82]	✓♦
	hyraxZK [49]	✗
Spartan [83]	Spartan [65]	✓♦
	ckb-zkp [82]	✓♦
Libra [92]	ckb-zkp [82]	✓♦
Brakedown [43]	Brakedown [19]	✓
Nova [58]	Nova [64]	✓♦
Gemini [16]	arkworks-gemini [38]	✓♦
Girault [42]	zk-paillier [96]	✓♦

Our Contributions

1. **Survey** of **75+** implementations:
   36 weak F-S vulnerabilities across
   12 different proof systems.

2. **Explicit Attacks** against ***Bulletproofs***, ***Plonk***, ***Spartan***, and ***Wesolowski's VDF***:
   $\Longrightarrow$ **Provable** break of soundness

3. **Practical Impacts:** ***unlimited*** currency minting in two blockchain protocols

4. **New Tool:** Decree for preventing weak F-S

vulnerable

patched

Proof System	Codebase	Weak F-S?
Bulletproofs [22]	bp-go [88]	✓
	bulletproof-js [2]	✓
	simple-bulletproof-js [84]	✓
	BulletproofSwift [20]	✓
	python-bulletproofs [79]	✓
	adjoint-bulletproofs [3]	✓
	zkSen [99]	✓
	incognito-chain [52]	✓♦
	encoins-bulletproofs [33]	✓♦
	ZenGo-X [97]	✓♦
	zkrp [53]	✓♦
	ckb-zkp [82]	✓♦
	bulletproofsrb [21]	✓♦
	monero [69]	✗
	dalek-bulletproofs [29]	✗
	secp256k1-zkp [76]	✗
	bulletproofs-ocaml [75]	✗
	tari-project [86]	✗
	Litecoin [60]	✗
	Grin [44]	✗
Bulletproofs variant [40]	dalek-bulletproofs [29]	✓♦
	cpp-lwevss [61]	✗
Sonic [62]	ebfull-sonic [18]	✓
	lx-sonic [59]	✓
	iohk-sonic [54]	✗
	adjoint-sonic [4]	✗
Schnorr [80]	noknow-python [7]	✓

Proof System	Codebase	Weak F-S?
Plonk [37]	anoma-plonkup [6]	✓
	gnark [17]	✓♦
	dusk-network [31]	✓♦
	snarkjs [50]	✓♦
	ZK-Garage [98]	✓♦
	plonky [68]	✗
	ckb-zkp [82]	✗
	halo2 [94]	✗
	o1-labs [72]	✗
	jellyfish [34]	✗
	matter-labs [63]	✗
	aztec-connect [8]	✗
Wesolowski's VDF [91]	0xProject [1]	✓
	Chia [70]	✓
	Harmony [47]	✓
	POA Network [71]	✓
	IOTA Ledger [55]	✓
	master-thesis-ELTE [48]	✓
Hyrax [90]	ckb-zkp [82]	✓♦
	hyraxZK [49]	✗
Spartan [83]	Spartan [65]	✓♦
	ckb-zkp [82]	✓♦
Libra [92]	ckb-zkp [82]	✓♦
Brakedown [43]	Brakedown [19]	✓
Nova [58]	Nova [64]	✓♦
Gemini [16]	arkworks-gemini [38]	✓♦
Girault [42]	zk-paillier [96]	✓♦

Our Contributions

1. **Survey** of **75+** implementations:
   36 weak F-S vulnerabilities across
   12 different proof systems.

2. **Explicit Attacks** against ***Bulletproofs***, ***Plonk***, ***Spartan***, and ***Wesolowski's VDF***:
   $\Longrightarrow$ **Provable** break of soundness

3. **Practical Impacts:** ***unlimited*** currency minting in two blockchain protocols

4. **New Tool:** Decree for preventing weak F-S

vulnerable

patched

Proof System	Codebase	Weak F-S?
Bulletproofs [22]	bp-go [88]	✓
	bulletproof-js [2]	✓
	simple-bulletproof-js [84]	✓
	BulletproofSwift [20]	✓
	python-bulletproofs [79]	✓
	adjoint-bulletproofs [3]	✓
	zkSen [99]	✓
	incognito-chain [52]	✓♦
	encoins-bulletproofs [33]	✓♦
	ZenGo-X [97]	✓♦
	zkrp [53]	✓♦
	ckb-zkp [82]	✓♦
	bulletproofsrb [21]	✓♦
	monero [69]	✗
	dalek-bulletproofs [29]	✗
	secp256k1-zkp [76]	✗
	bulletproofs-ocaml [75]	✗
	tari-project [86]	✗
	Litecoin [60]	✗
	Grin [44]	✗
Bulletproofs variant [40]	dalek-bulletproofs [29]	✓♦
	cpp-lwevss [61]	✗
Sonic [62]	ebfull-sonic [18]	✓
	lx-sonic [59]	✓
	iohk-sonic [54]	✗
	adjoint-sonic [4]	✗
Schnorr [80]	noknow-python [7]	✓

Proof System	Codebase	Weak F-S?
Plonk [37]	anoma-plonkup [6]	✓
	gnark [17]	✓♦
	dusk-network [31]	✓♦
	snarkjs [50]	✓♦
	ZK-Garage [98]	✓♦
	plonky [68]	✗
	ckb-zkp [82]	✗
	halo2 [94]	✗
	o1-labs [72]	✗
	jellyfish [34]	✗
	matter-labs [63]	✗
	aztec-connect [8]	✗
Wesolowski's VDF [91]	0xProject [1]	✓
	Chia [70]	✓
	Harmony [47]	✓
	POA Network [71]	✓
	IOTA Ledger [55]	✓
	master-thesis-ELTE [48]	✓
Hyrax [90]	ckb-zkp [82]	✓♦
	hyraxZK [49]	✗
Spartan [83]	Spartan [65]	✓♦
	ckb-zkp [82]	✓♦
Libra [92]	ckb-zkp [82]	✓♦
Brakedown [43]	Brakedown [19]	✓
Nova [58]	Nova [64]	✓♦
Gemini [16]	arkworks-gemini [38]	✓♦
Girault [42]	zk-paillier [96]	✓♦

IEEE S&P Distinguished Paper Award!

Weak Fiat-Shamir Attacks

(as easy as solving a linear equation)

Template for Weak F-S Attacks

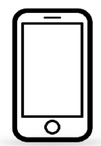
Template for Weak F-S Attacks

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

(X, x)



X

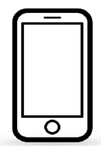
Template for Weak F-S Attacks

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

(X, x)



X

$$a \xleftarrow{R} \mathbb{F}_p$$

$$A := g^a$$

$$c := H(A) \in \mathbb{F}_p$$

$$z := a + cx$$

Template for Weak F-S Attacks

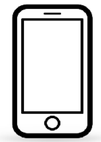
Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

P

(X, x)



V

X

$a \xleftarrow{R} \mathbb{F}_p$

$A := g^a$

$c := H(A) \in \mathbb{F}_p$

$z := a + cx$

$\pi := (A, z)$ →

Template for Weak F-S Attacks

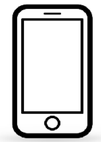
Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

P

(X, x)



V

X

$a \xleftarrow{R} \mathbb{F}_p$

$A := g^a$

$c := H(A) \in \mathbb{F}_p$

$z := a + cx$

$\pi := (A, z)$

$c := H(A) \in \mathbb{F}_p$

Check that

$g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

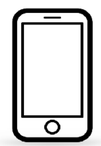
Attack Strategy

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

(X, x)



X

$a \xleftarrow{R} \mathbb{F}_p$

$A := g^a$

$c := H(A) \in \mathbb{F}_p$

$z := a + cx$

$\pi := (A, z)$ →

$c := H(A) \in \mathbb{F}_p$

Check that

$g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

Attack Strategy

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

$\pi := (A, z)$

$c := H(A) \in \mathbb{F}_p$

Check that

$g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

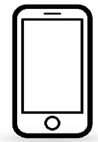
Attack Strategy

1. **Identify** the public input(s) that are ***not*** included in Fiat-Shamir,

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$



$\pi := (A, z)$ →

$c := H(A) \in \mathbb{F}_p$

Check that

$g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

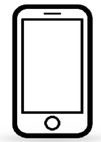
Attack Strategy

1. **Identify** the **public input(s)** that are ***not*** included in Fiat-Shamir,

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$



$\pi := (A, z)$ →

$c := H(A) \in \mathbb{F}_p$

Check that

$g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

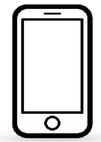
Attack Strategy

1. **Identify** the public input(s) that are ***not*** included in Fiat-Shamir,
2. **Find** the verification check(s) that rely on said public input(s),

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$



$\pi := (A, z)$ →

$c := H(A) \in \mathbb{F}_p$

Check that

$g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

Attack Strategy

1. **Identify** the public input(s) that are ***not*** included in Fiat-Shamir,
2. **Find** the verification check(s) that rely on said public input(s),

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

$\pi := (A, z)$

$c := H(A) \in \mathbb{F}_p$

Check that

$g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

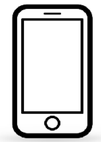
Attack Strategy

1. **Identify** the public input(s) that are ***not*** included in Fiat-Shamir,
2. **Find** the verification check(s) that rely on said public input(s),
3. **Compute** a proof with ***arbitrary*** witness and randomness,

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$



$\pi := (A, z)$

$c := H(A) \in \mathbb{F}_p$

Check that

$g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

Attack Strategy

1. **Identify** the public input(s) that are ***not*** included in Fiat-Shamir,
2. **Find** the verification check(s) that rely on said public input(s),
3. **Compute** a proof with ***arbitrary*** witness and randomness,

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

$A \xleftarrow{R} \mathbb{G}$

$c := H(A) \in \mathbb{F}_p$

$z \xleftarrow{R} \mathbb{F}_p$

$\pi := (A, z)$ →

$c := H(A) \in \mathbb{F}_p$

Check that $g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

Attack Strategy

1. **Identify** the public input(s) that are ***not*** included in Fiat-Shamir,
2. **Find** the verification check(s) that rely on said public input(s),
3. **Compute** a proof with ***arbitrary*** witness and randomness,
4. **Solve** for the public input value(s) that would pass verification.

Schnorr with Weak F-S

Relation:

$$\left\{ \left((\mathbb{G}, g, p), X, x \right) : X = g^x \right\}$$

$A \xleftarrow{R} \mathbb{G}$

$c := H(A) \in \mathbb{F}_p$

$z \xleftarrow{R} \mathbb{F}_p$

$\pi := (A, z)$ →

$c := H(A) \in \mathbb{F}_p$

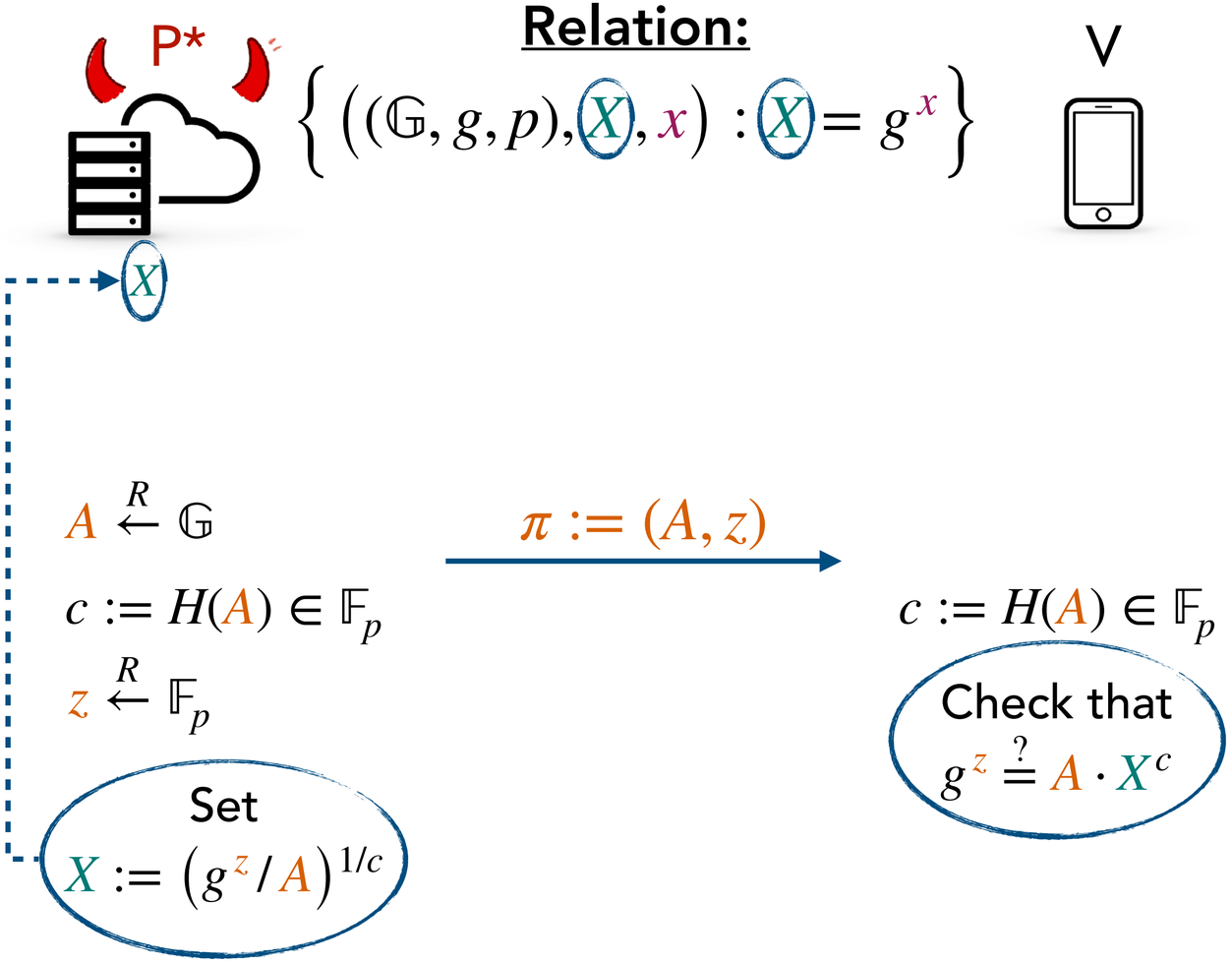
Check that $g^z \stackrel{?}{=} A \cdot X^c$

Template for Weak F-S Attacks

Attack Strategy

1. **Identify** the public input(s) that are ***not*** included in Fiat-Shamir,
2. **Find** the verification check(s) that rely on said public input(s),
3. **Compute** a proof with ***arbitrary*** witness and randomness,
4. **Solve** for the public input value(s) that would pass verification.

Schnorr with Weak F-S

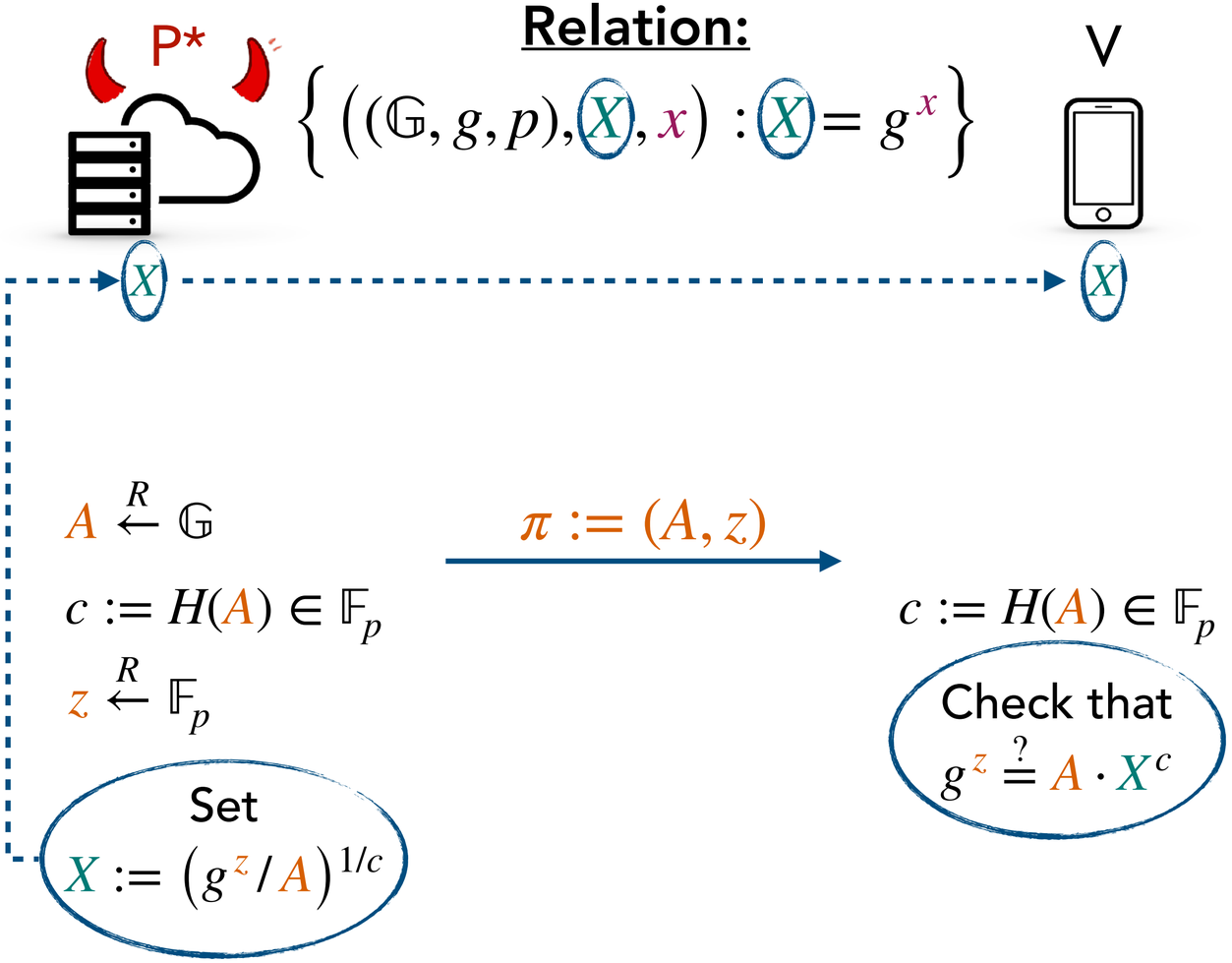


Template for Weak F-S Attacks

Attack Strategy

1. **Identify** the public input(s) that are ***not*** included in Fiat-Shamir,
2. **Find** the verification check(s) that rely on said public input(s),
3. **Compute** a proof with ***arbitrary*** witness and randomness,
4. **Solve** for the public input value(s) that would pass verification.

Schnorr with Weak F-S



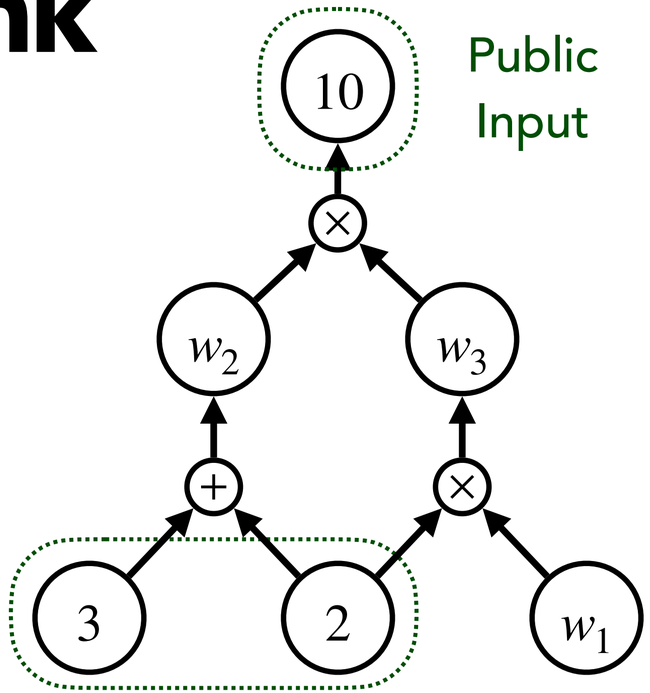
Weak Fiat-Shamir Attack on Plonk

Weak Fiat-Shamir Attack on Plonk

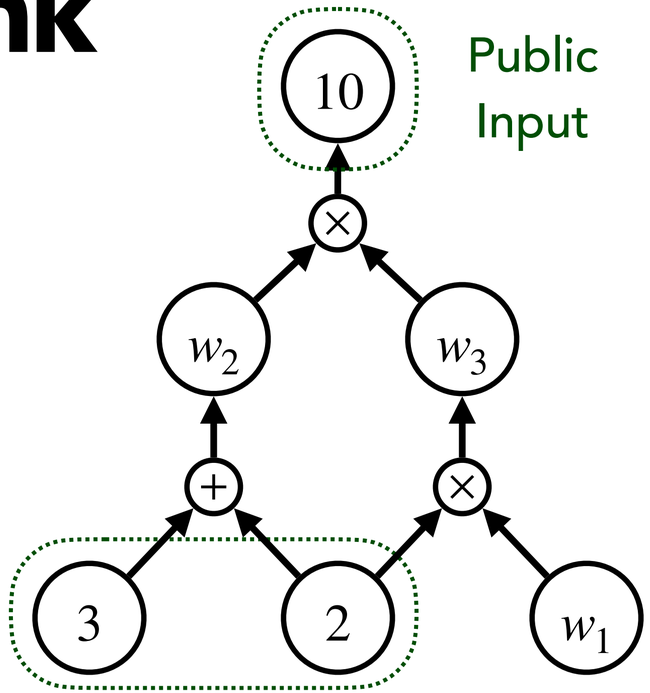
Constraint System: general (fan-in 2) arithmetic circuits

Weak Fiat-Shamir Attack on Plonk

Constraint System: general (fan-in 2) arithmetic circuits



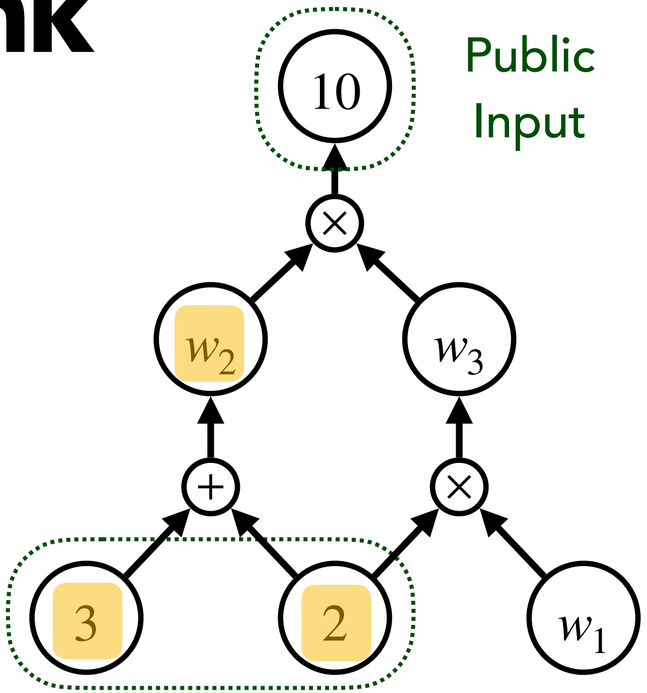
Weak Fiat-Shamir Attack on Plonk



<u>**Constraint System:**</u> general (fan-in 2) arithmetic circuits

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

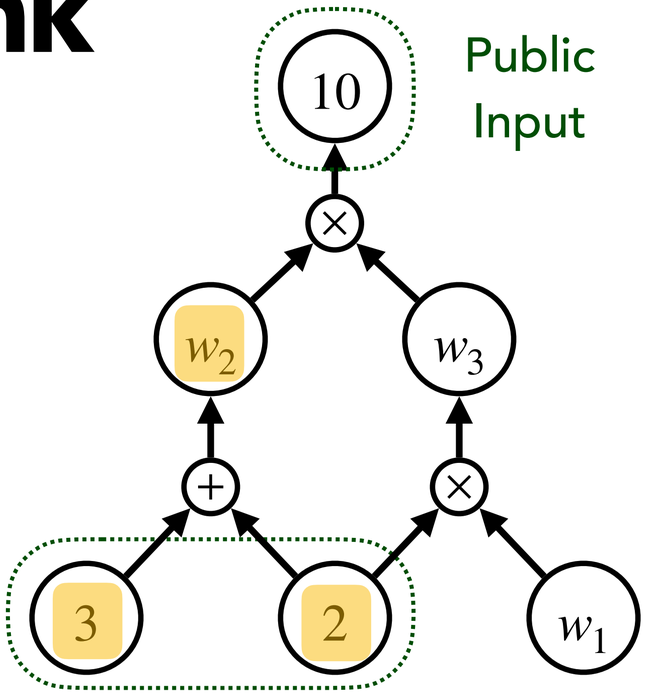
Weak Fiat-Shamir Attack on Plonk



<u>**Constraint System:**</u> general (fan-in 2) arithmetic circuits

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

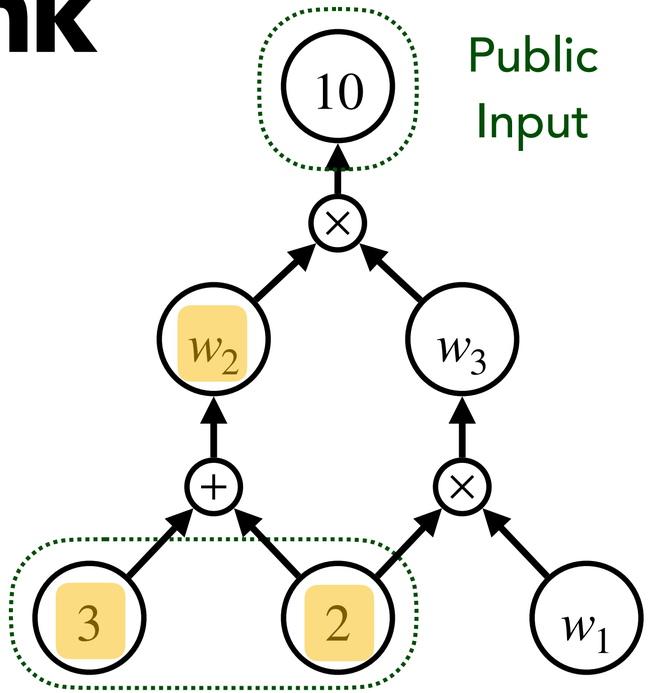
Weak Fiat-Shamir Attack on Plonk



<u>Constraint System:</u> general (fan-in 2) arithmetic circuits

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

Weak Fiat-Shamir Attack on Plonk

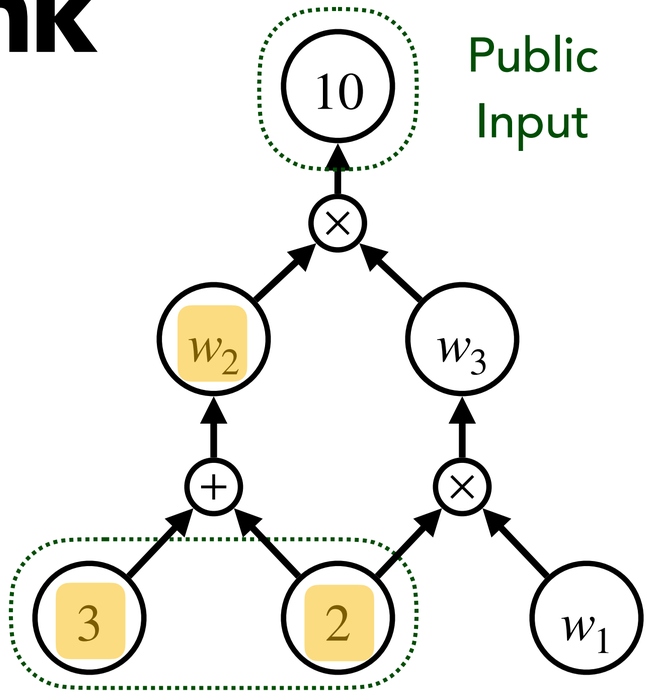


Constraint System: general (fan-in 2) arithmetic circuits

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

Verification Equation:

Weak Fiat-Shamir Attack on Plonk



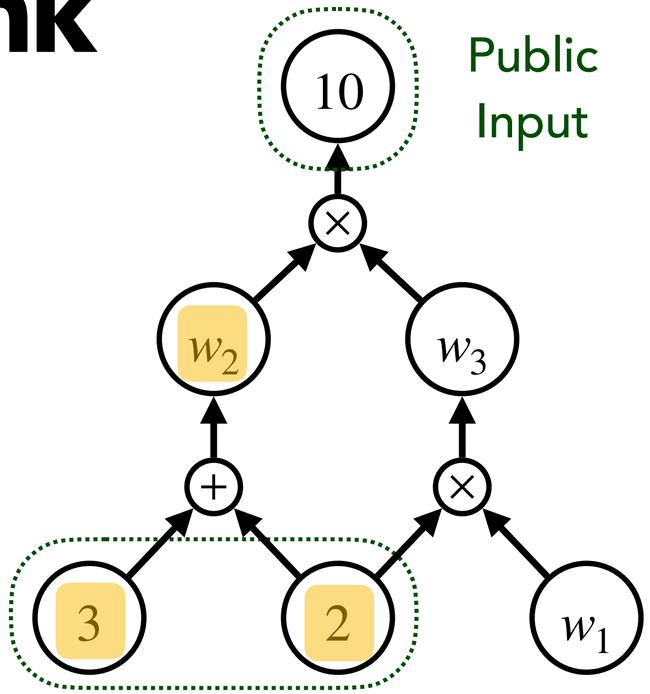
Constraint System: general (fan-in 2) arithmetic circuits

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

Verification Equation:

$$\mathsf{PI}(\zeta) + \mathsf{Eq}(\zeta) \; + \; \alpha \cdot \mathsf{Per}(\zeta) + \alpha^2 \cdot (\mathsf{z}(\zeta) - 1)\mathsf{L}_1(\zeta) \; = \; \mathsf{Z}_\mathsf{H}(\zeta) \cdot \mathsf{t}(\zeta)$$

Weak Fiat-Shamir Attack on Plonk



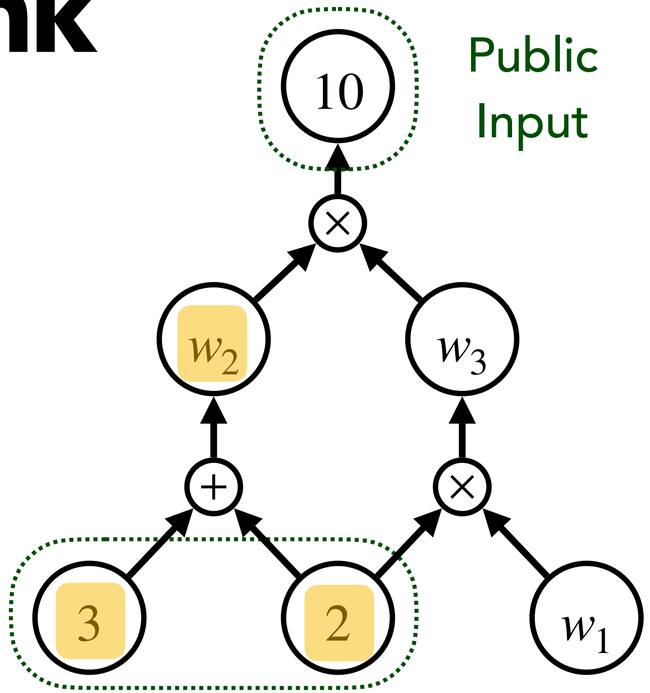
Constraint System: general (fan-in 2) arithmetic circuits

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

Verification Equation:

$$\underbrace{\mathsf{PI}(\zeta) + \mathsf{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \mathsf{Per}(\zeta) + \alpha^2 \cdot (\mathsf{z}(\zeta) - 1)\mathsf{L}_1(\zeta)}_{\text{Consistency Check}} = \underbrace{\mathsf{Z}_\mathsf{H}(\zeta) \cdot \mathsf{t}(\zeta)}_{\text{Vanishing Domain}}$$

Weak Fiat-Shamir Attack on Plonk



Constraint System: general (fan-in 2) arithmetic circuits

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

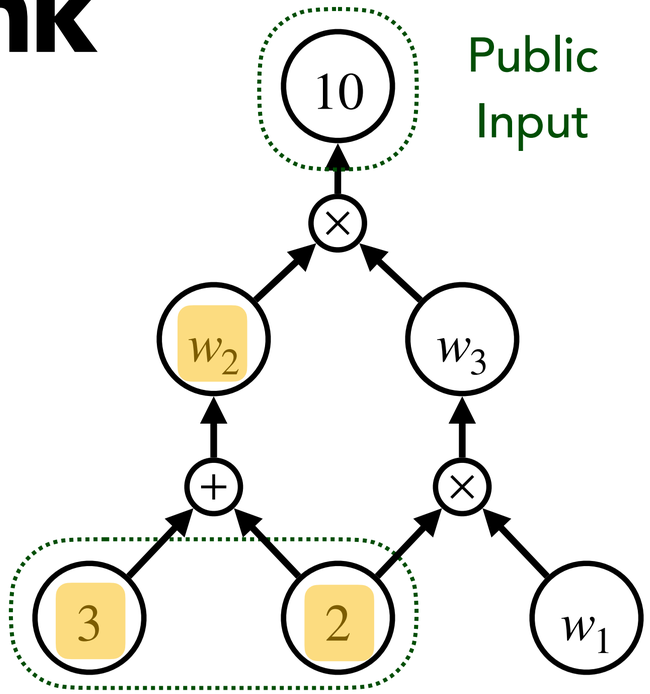
Verification Equation:

Batching Challenge

Evaluation Point

$$\underbrace{\mathsf{PI}(\zeta) + \mathsf{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \mathsf{Per}(\zeta) + \alpha^2 \cdot (\mathsf{z}(\zeta) - 1)\mathsf{L}_1(\zeta)}_{\text{Consistency Check}} = \underbrace{\mathsf{Z}_\mathsf{H}(\zeta) \cdot \mathsf{t}(\zeta)}_{\text{Vanishing Domain}}$$

Weak Fiat-Shamir Attack on Plonk



Constraint System: general (fan-in 2) arithmetic circuits

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

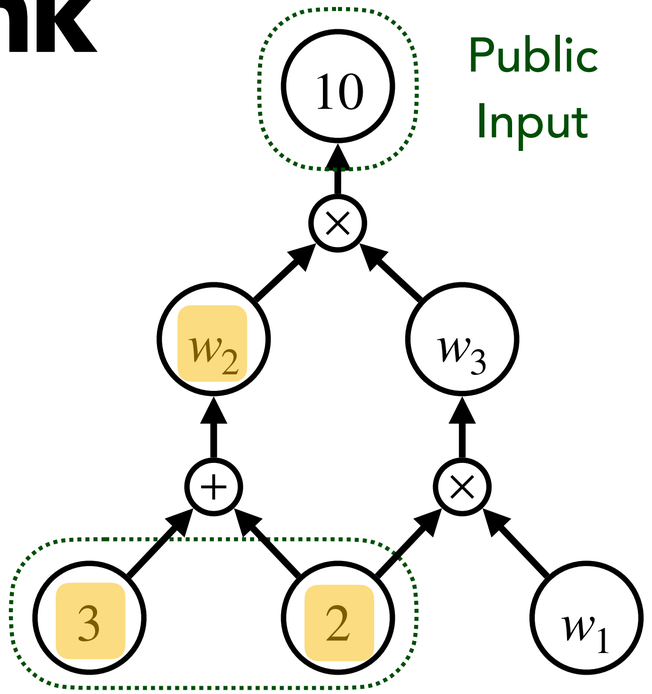
Verification Equation:

$$\sum_{i=1}^{k} \mathsf{PI}_i \cdot \mathsf{L}_i(\zeta) \leftrightarrow \underbrace{\mathsf{PI}(\zeta) + \mathsf{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \mathsf{Per}(\zeta) + \alpha^2 \cdot (\mathsf{z}(\zeta) - 1)\mathsf{L}_1(\zeta)}_{\text{Consistency Check}} = \underbrace{\mathsf{Z}_H(\zeta) \cdot \mathsf{t}(\zeta)}_{\text{Vanishing Domain}}$$

Batching Challenge: α

Evaluation Point: ζ

Weak Fiat-Shamir Attack on Plonk



Constraint System: general (fan-in 2) arithmetic circuits

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

Verification Equation:

$$\sum_{i=1}^{k} \mathsf{PI}_i \cdot \mathsf{L}_i(\zeta) \leftrightarrow \underbrace{\mathsf{PI}(\zeta) + \mathsf{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \mathsf{Per}(\zeta) + \alpha^2 \cdot (\mathsf{z}(\zeta) - 1)\mathsf{L}_1(\zeta)}_{\text{Consistency Check}} = \underbrace{\mathsf{Z}_H(\zeta) \cdot \mathsf{t}(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars

Batching Challenge

Evaluation Point

Weak Fiat-Shamir Attack on Plonk

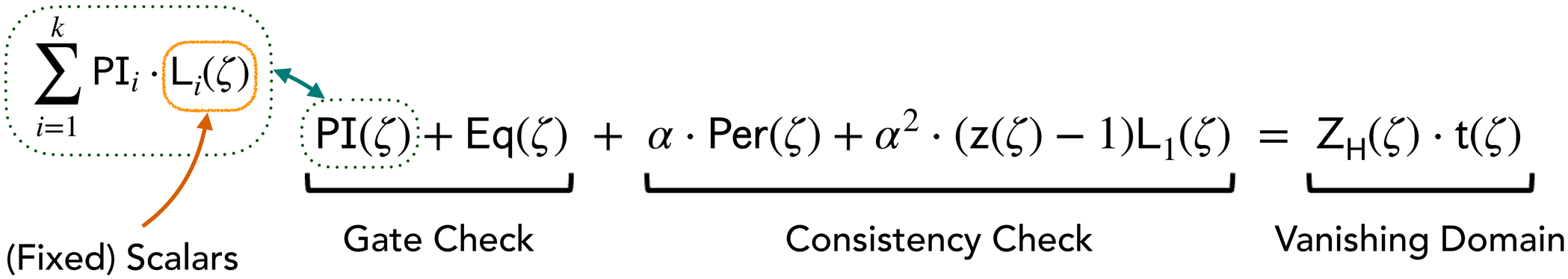
Verification Equation:

$$\sum_{i=1}^{k} \mathsf{PI}_i \cdot \mathsf{L}_i(\zeta) \qquad \underbrace{\mathsf{PI}(\zeta) + \mathsf{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \mathsf{Per}(\zeta) + \alpha^2 \cdot (\mathsf{z}(\zeta) - 1)\mathsf{L}_1(\zeta)}_{\text{Consistency Check}} = \underbrace{\mathsf{Z}_H(\zeta) \cdot \mathsf{t}(\zeta)}_{\text{Vanishing Domain}}$$

(Fixed) Scalars

Weak Fiat-Shamir Attack on Plonk

Verification Equation:



Weak F-S Attack: When PI is not part of hash computation (for deriving α, ζ)

Weak Fiat-Shamir Attack on Plonk

Verification Equation:

$$\sum_{i=1}^{k} \mathsf{PI}_i \cdot \mathsf{L}_i(\zeta) \qquad \underbrace{\mathsf{PI}(\zeta) + \mathsf{Eq}(\zeta)}_{\text{Gate Check}} + \underbrace{\alpha \cdot \mathsf{Per}(\zeta) + \alpha^2 \cdot (\mathsf{z}(\zeta) - 1)\mathsf{L}_1(\zeta)}_{\text{Consistency Check}} = \underbrace{\mathsf{Z}_\mathsf{H}(\zeta) \cdot \mathsf{t}(\zeta)}_{\text{Vanishing Domain}}$$

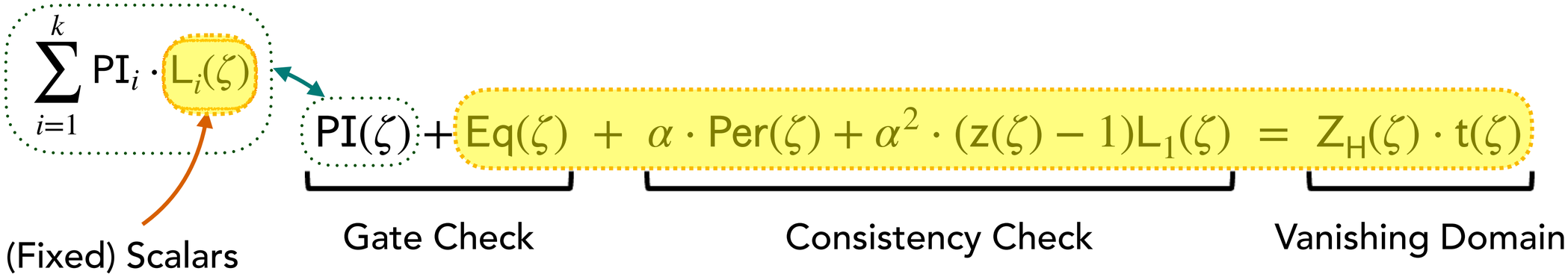
(Fixed) Scalars

Weak F-S Attack: When PI is not part of hash computation (for deriving α, ζ)

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

Weak Fiat-Shamir Attack on Plonk

Verification Equation:

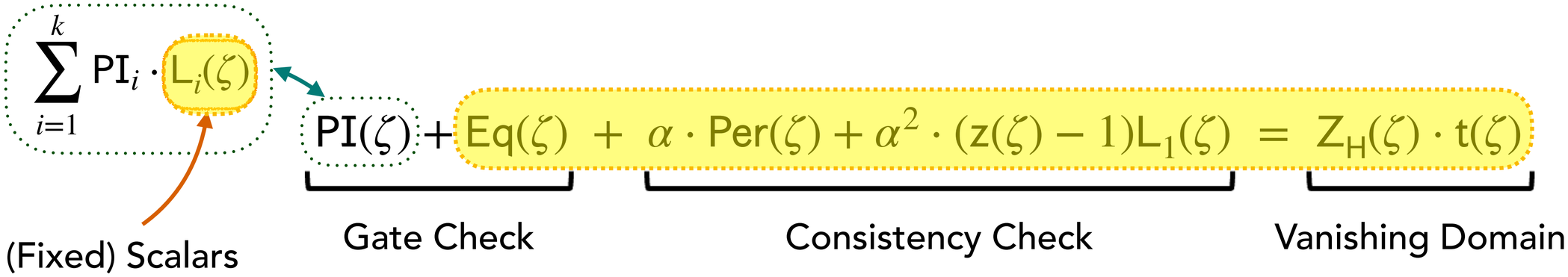


Weak F-S Attack: When PI is not part of hash computation (for deriving α, ζ)

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

Weak Fiat-Shamir Attack on Plonk

Verification Equation:

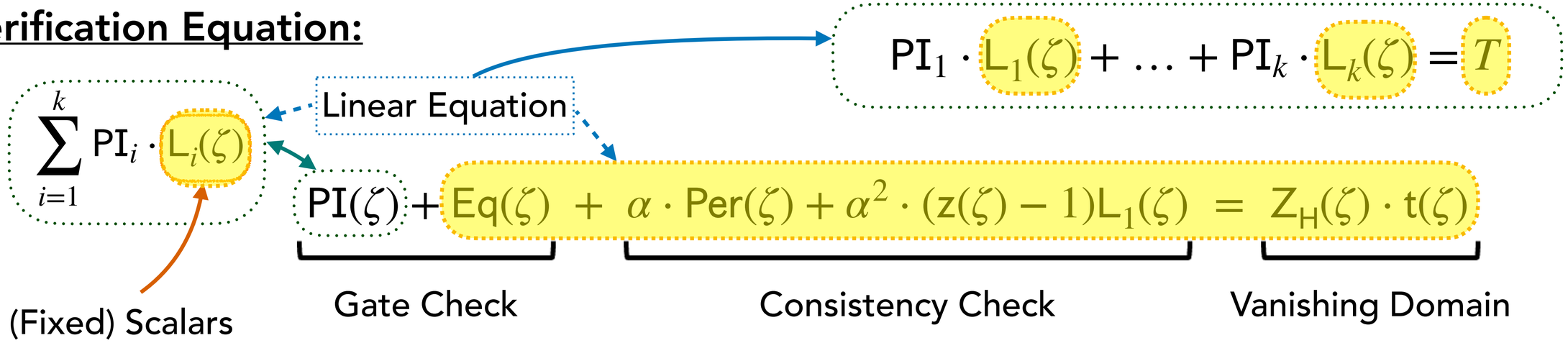


Weak F-S Attack: When PI is not part of hash computation (for deriving α, ζ)

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

Weak Fiat-Shamir Attack on Plonk

Verification Equation:

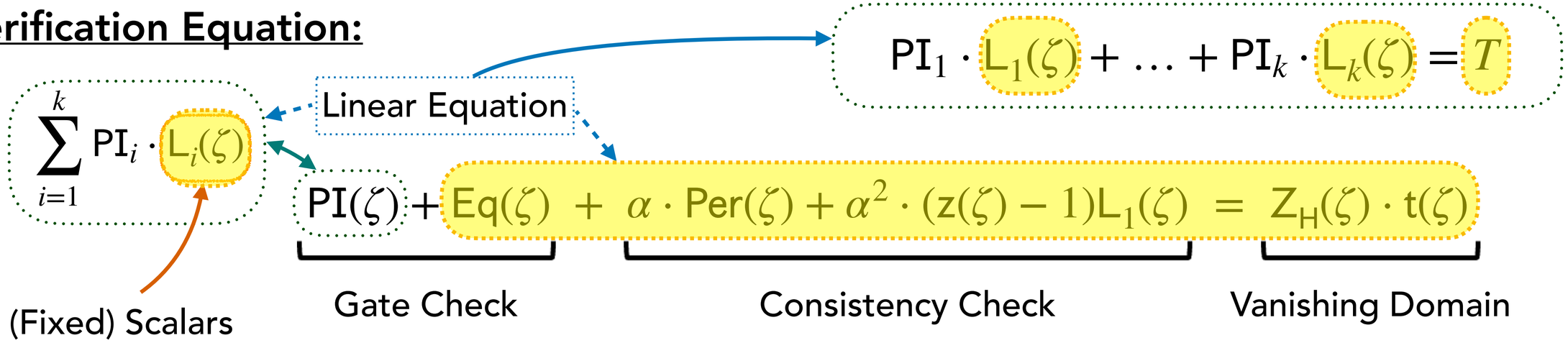


Weak F-S Attack: When PI is not part of hash computation (for deriving α, ζ)

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

Weak Fiat-Shamir Attack on Plonk

Verification Equation:



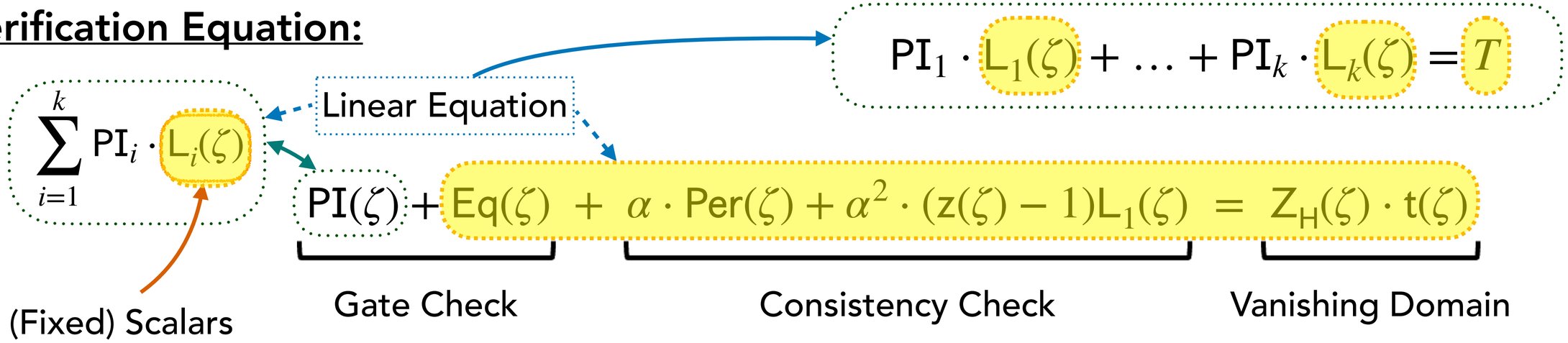
Weak F-S Attack: When PI is not part of hash computation (for deriving α, ζ)

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

   Degrees of freedom: can set *all but one* PI_i to be arbitrary.

Weak Fiat-Shamir Attack on Plonk

Verification Equation:



Weak F-S Attack: When PI is not part of hash computation (for deriving α, ζ)

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

   Degrees of freedom: can set *all but one* PI_i to be arbitrary.

In Contrast: For **strong** Fiat-Shamir, changing PI will also change α, ζ.

Weak Fiat-Shamir Attacks

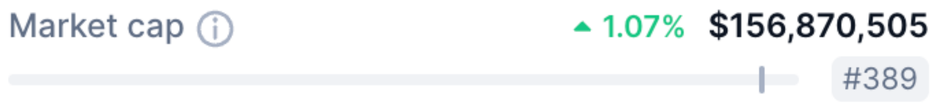
Practical Impacts

(unlimited money printing on blockchains)

Case Study: Dusk Network

DUSK

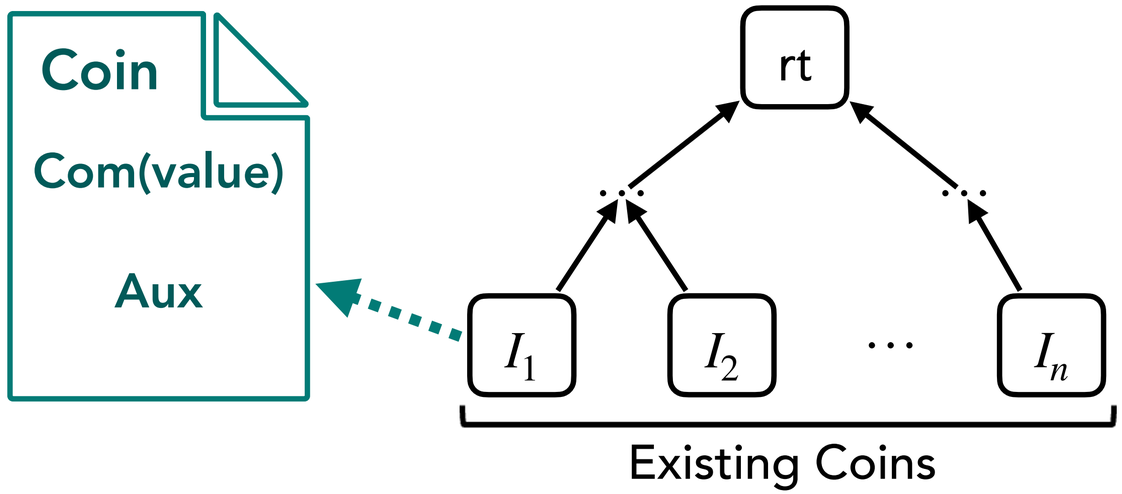
Regulated And Decentralized Finance.



(as of March 11, 2024)

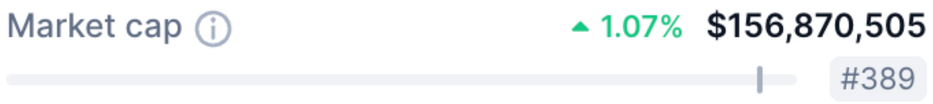
Case Study: Dusk Network

Transaction Model (simplified):



DUSK

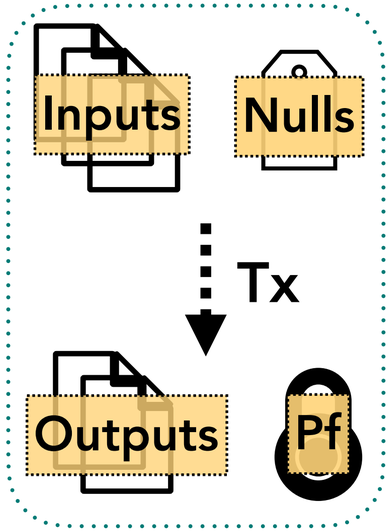
Regulated And Decentralized Finance.

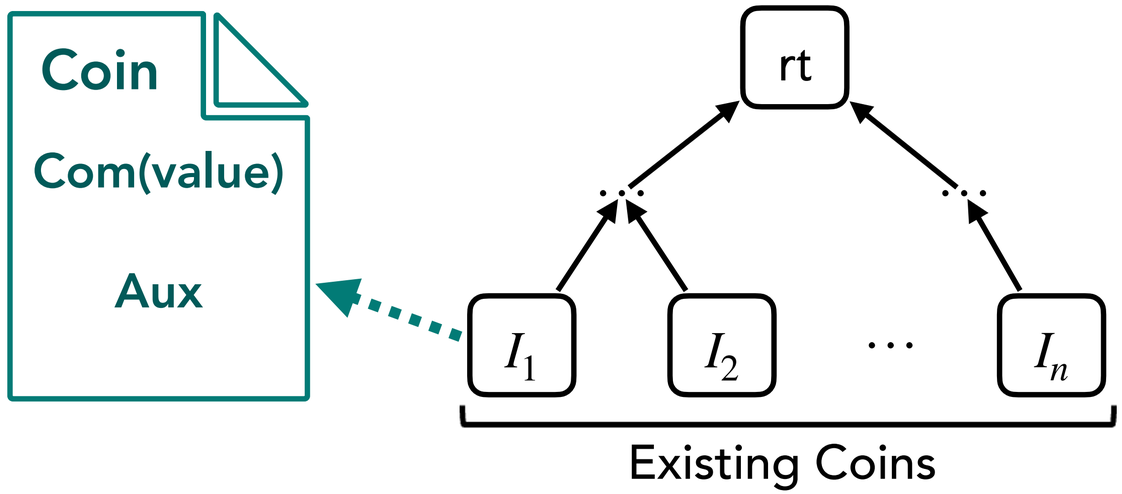


(as of March 11, 2024)

Case Study: Dusk Network

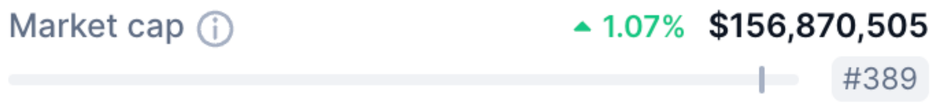
Transaction Model (simplified):





DUSK

Regulated And Decentralized Finance.



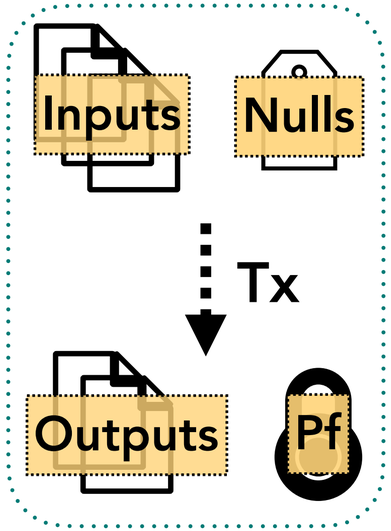
(as of March 11, 2024)

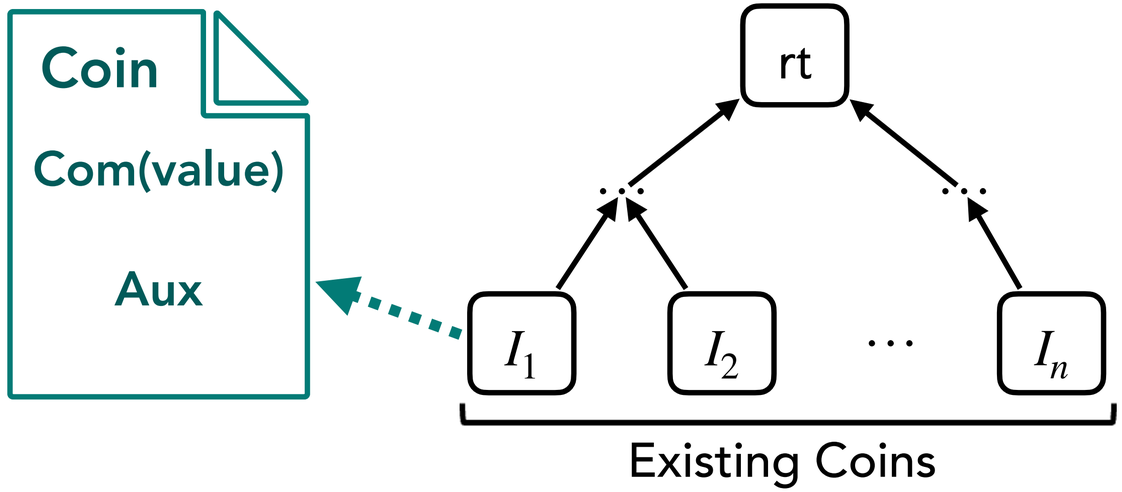
Case Study: Dusk Network

Transaction Model (simplified):

Public Inputs:

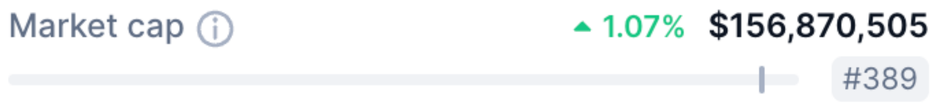
- Set of inputs & output coins
- Nullifier null_I for each input I





DUSK

Regulated And Decentralized Finance.



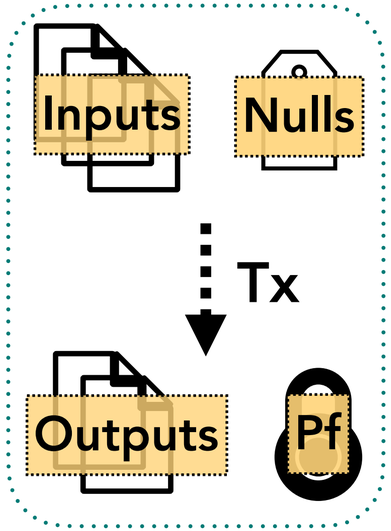
(as of March 11, 2024)

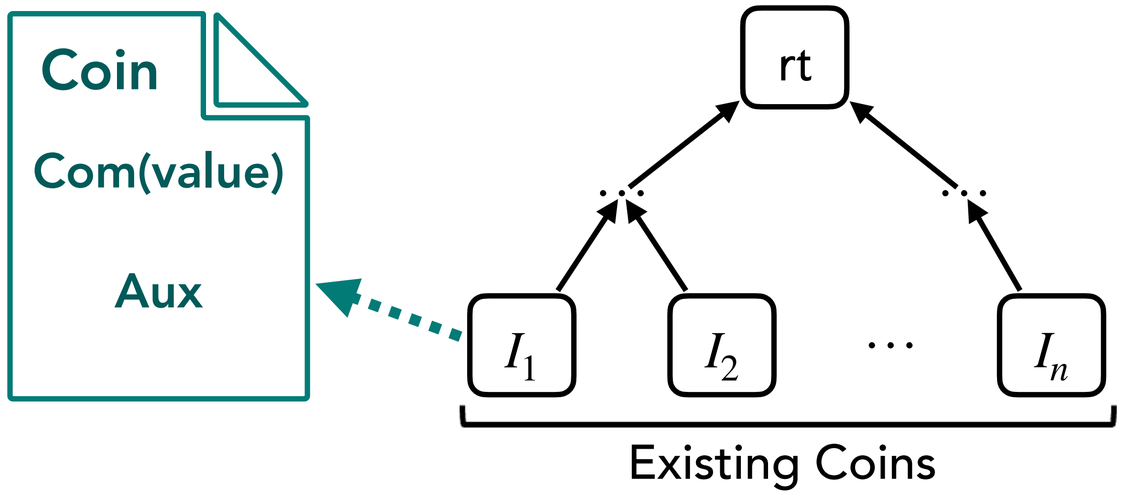
Case Study: Dusk Network

Transaction Model (simplified):

Public Inputs:

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



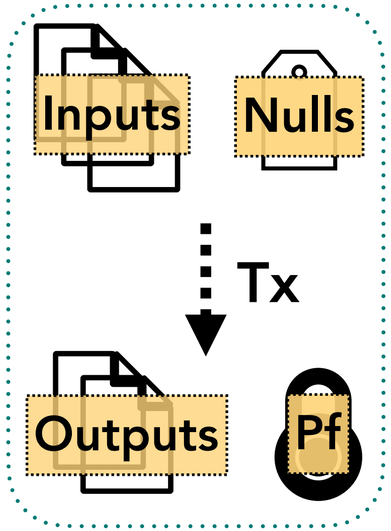


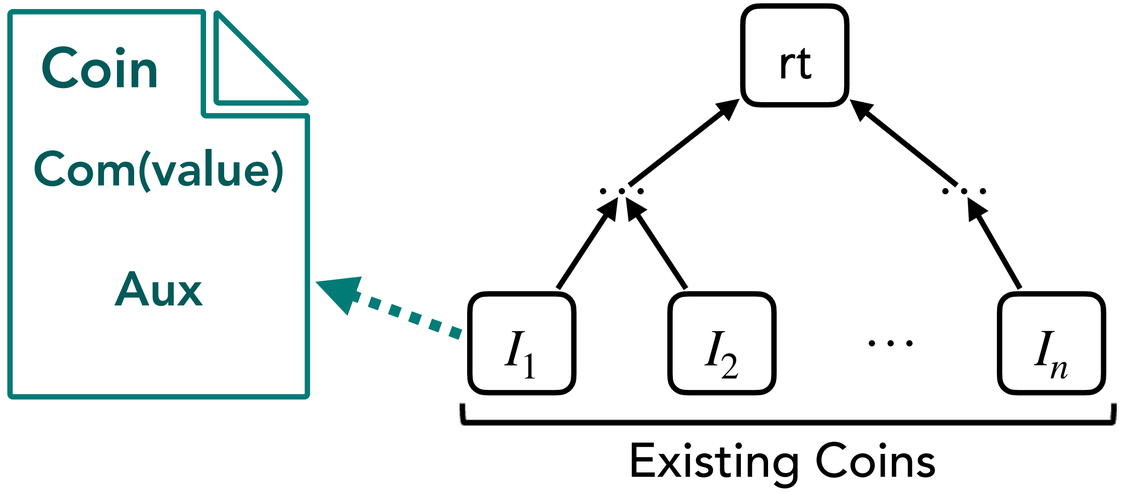
Case Study: Dusk Network

Transaction Model (simplified):

Public Inputs:

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



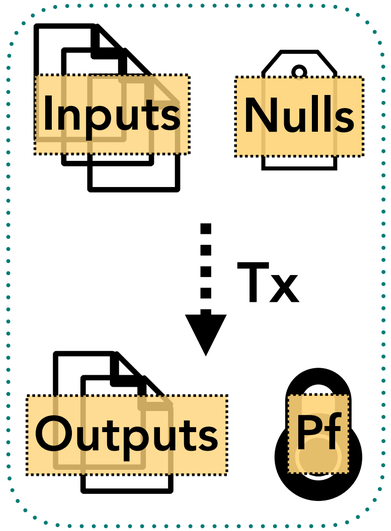


Proof Relation: (proved using Plonk)

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

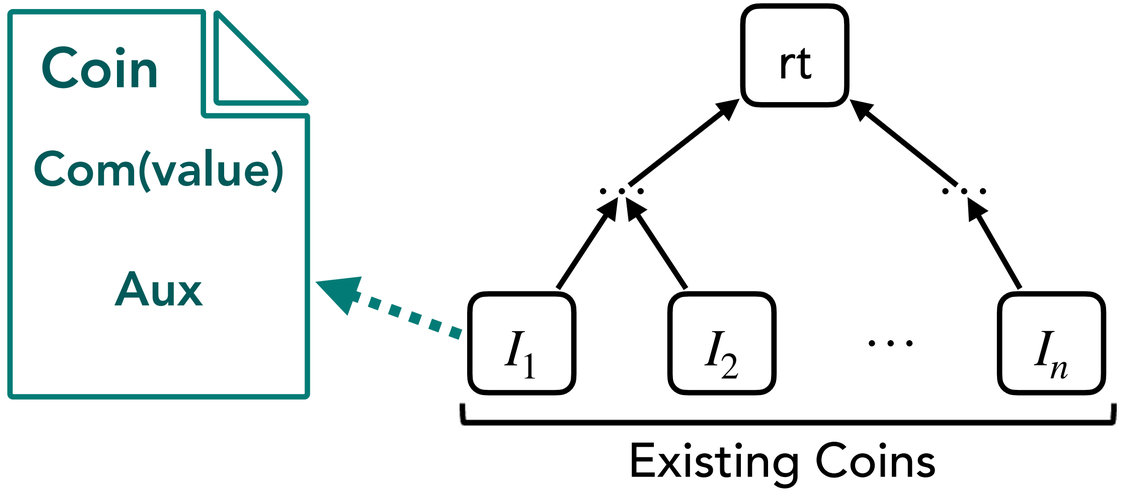
Case Study: Dusk Network

Transaction Model (simplified):



Public Inputs:

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



Proof Relation: (proved using Plonk)

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

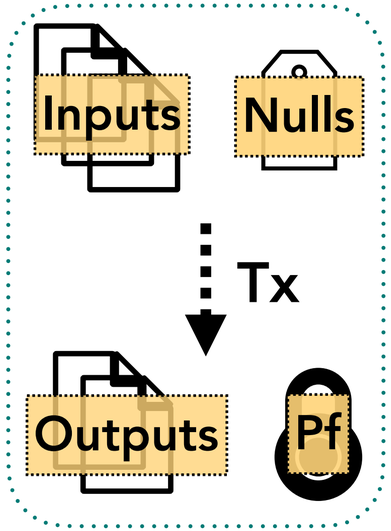
Weak F-S Attack:

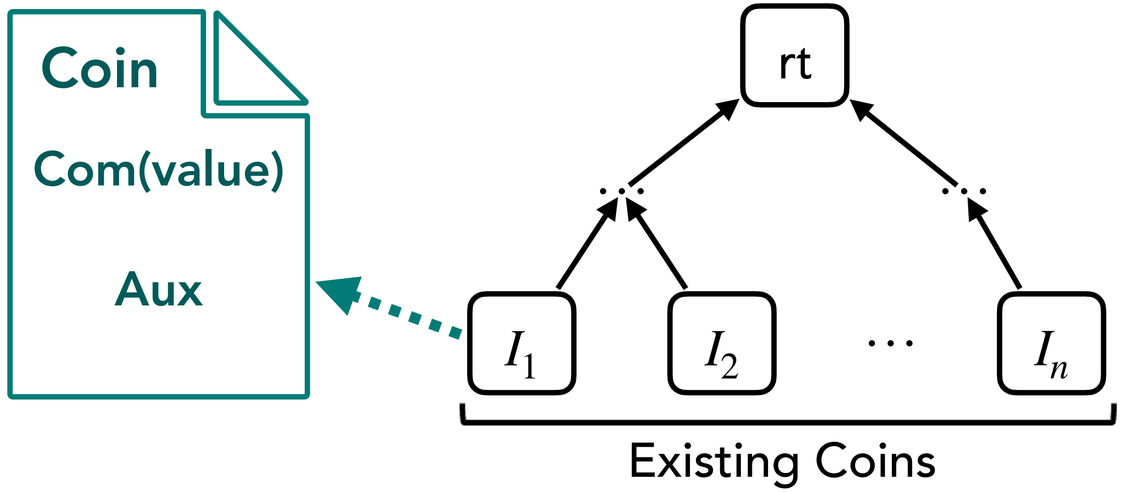
Case Study: Dusk Network

Transaction Model (simplified):

Public Inputs:

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





Proof Relation: (proved using Plonk)

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

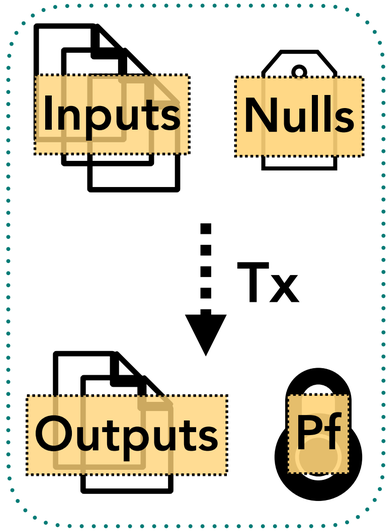
Weak F-S Attack:

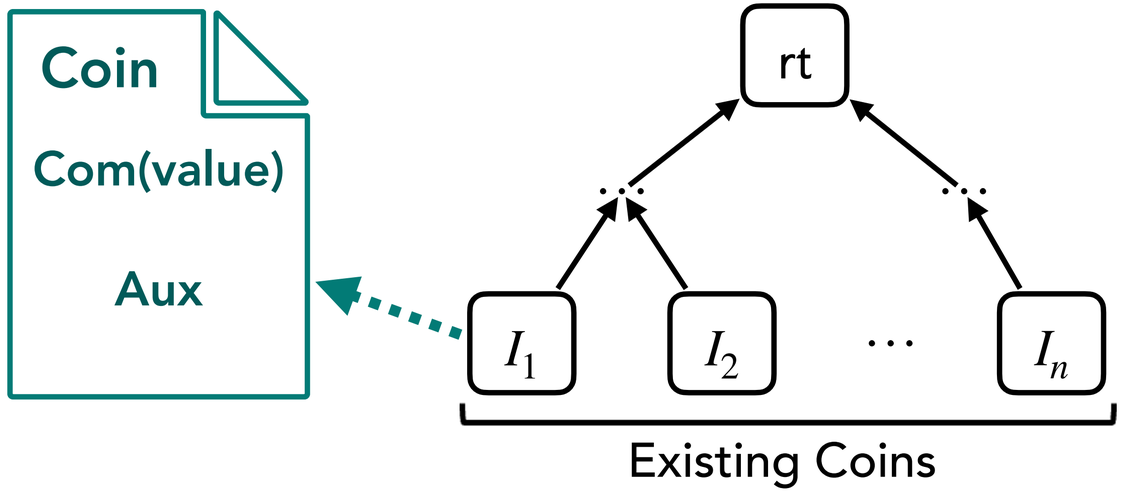
Case Study: Dusk Network

Transaction Model (simplified):

Public Inputs:

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





Proof Relation: (proved using Plonk)

- **Nullifier check:** null$_I$ $= H(pk$ $_I)$, ∀ input I
- **Range check:** v_{in}, v_{out} $r - 1]$, ∀ input & output
- **Equality check:** $\sum$ v_{out}
- **Merkle membersh**. I is in position pos_I w.r.t root rt

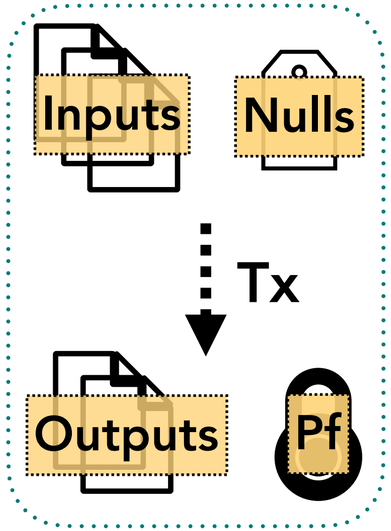
Weak F-S Attack:

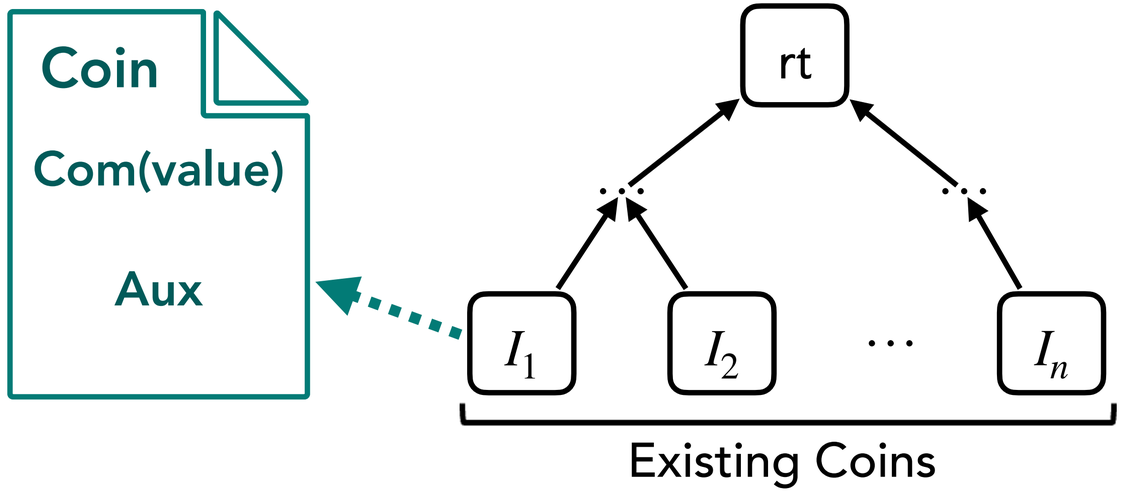
Case Study: Dusk Network

Transaction Model (simplified):

Public Inputs:

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





Proof Relation: (proved using Plonk)

- **Nullifier check:** null… $H(pk…_I)$, ∀ input I
- **Range check:** v_{in}, v_{out} … $r-1]$, ∀ input & output
- **Equality check:** $\sum$ … v_{out}
- **Merkle membersh…** I is in position pos_I w.r.t root rt

Weak F-S Attack:

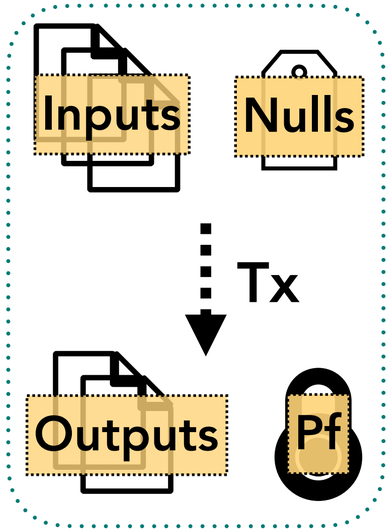
1. Create output coin w/ value 1 trillion DUSK.

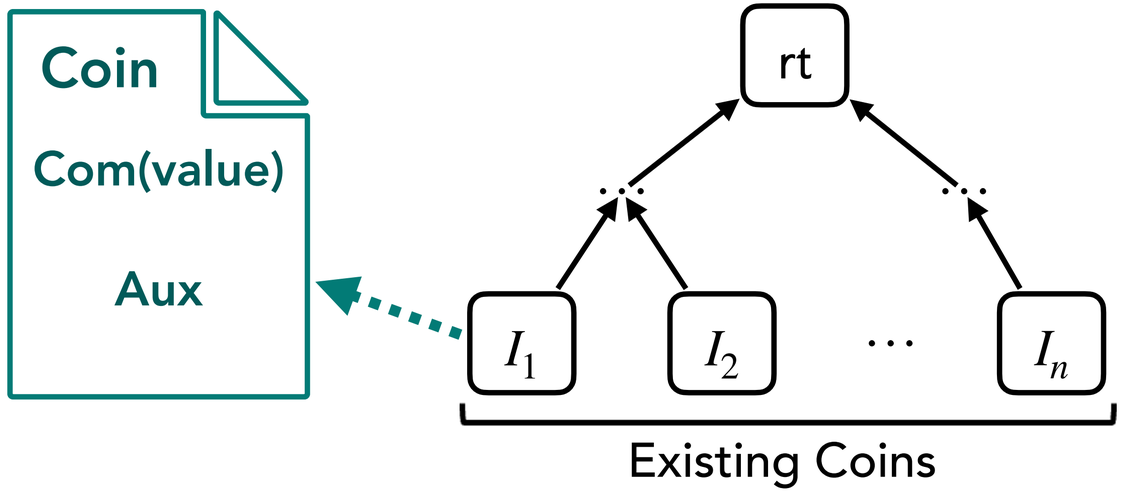
Case Study: Dusk Network

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… $H(pk…_I)$, ∀ input I
- **Range check:** v_{in}, v_{out} … $r-1]$, ∀ input & output
- **Equality check:** $\sum$ … v_{out}
- **Merkle membersh…** 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 π w/ arbitrary input, setting nullifier to satisfy π.

Case Study: Dusk Network

Disclosure Timeline:

Weak F-S Attack:

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

Case Study: Dusk Network

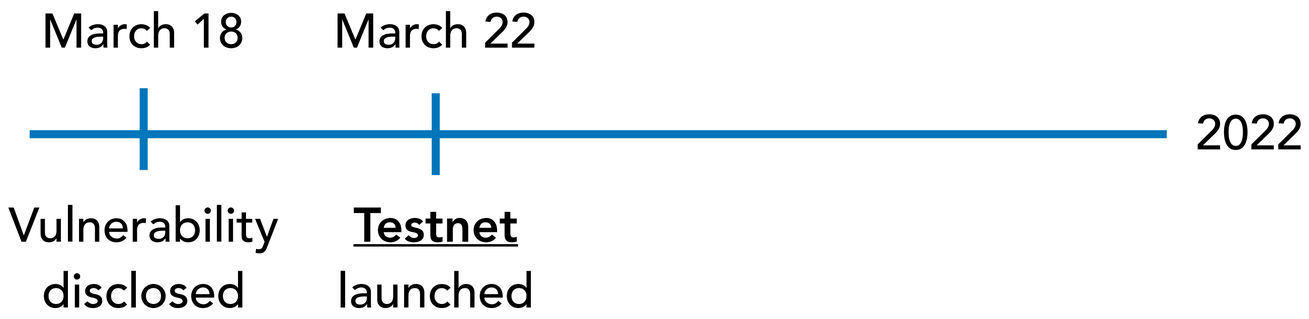
Disclosure Timeline:

Weak F-S Attack:

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

Case Study: Dusk Network

Disclosure Timeline:

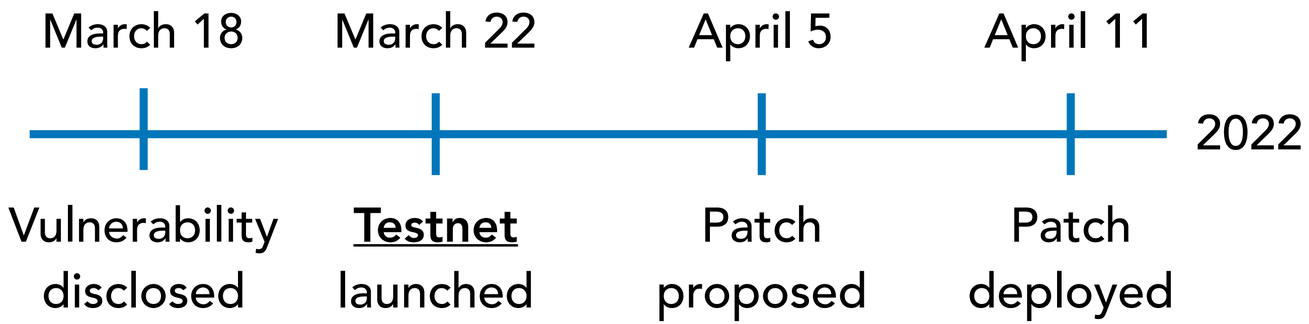


Weak F-S Attack:

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

Case Study: Dusk Network

Disclosure Timeline:

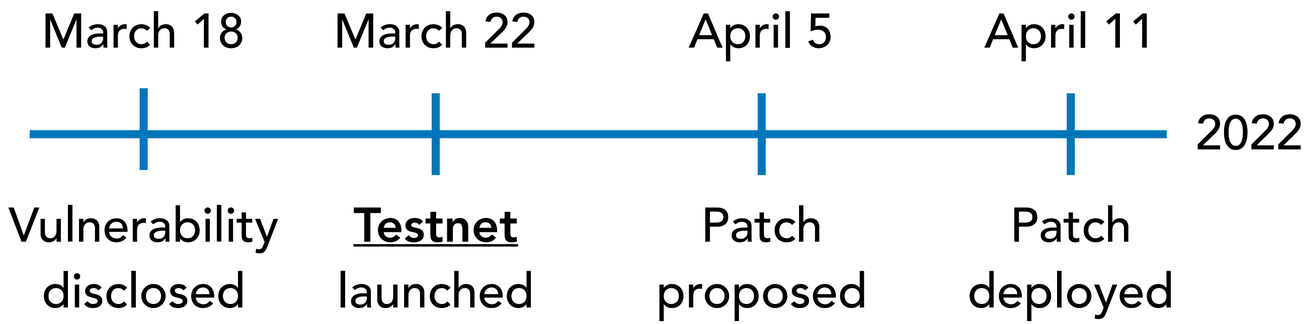


Weak F-S Attack:

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

Case Study: Dusk Network

Disclosure Timeline:



Apr 12, 2022 - Mels Dees

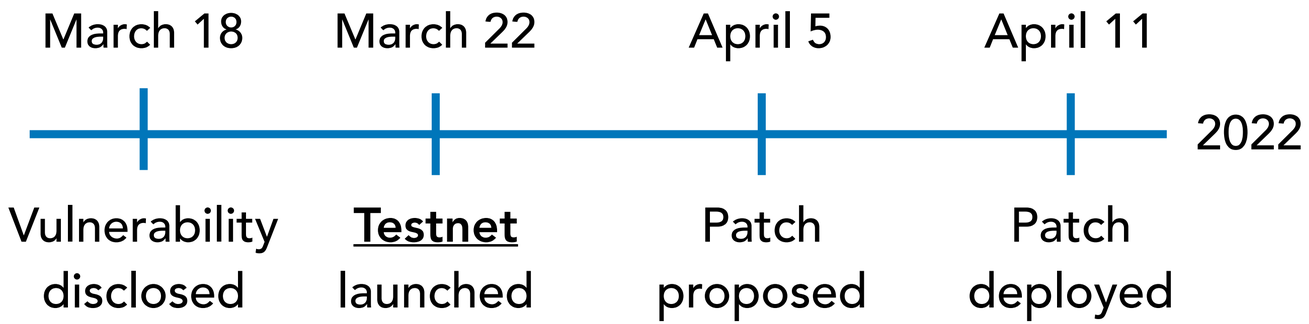
PLONK Critical Vulnerability Successfully Remediated

Weak F-S Attack:

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

Case Study: Dusk Network

Disclosure Timeline:



Apr 12, 2022 - Mels Dees

PLONK Critical Vulnerability Successfully Remediated

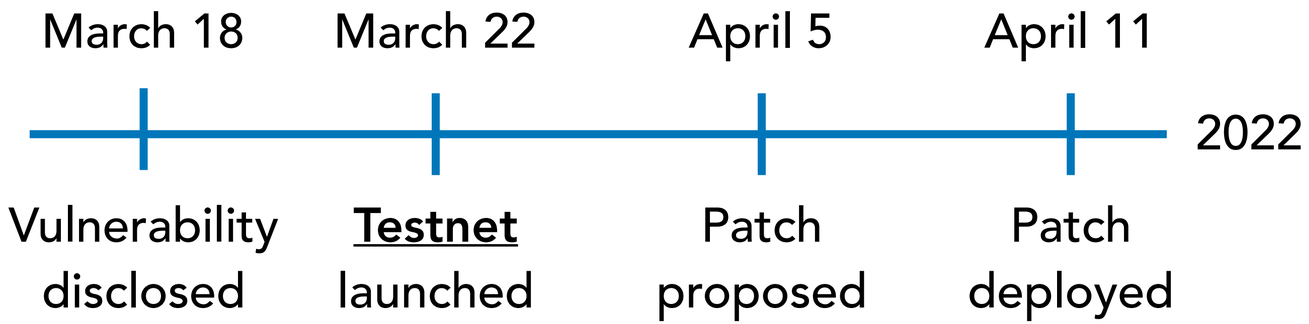
- No user funds were at risk
- However, we don't know if the attack was carried out

Weak F-S Attack:

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

Case Study: Dusk Network

Disclosure Timeline:



Apr 12, 2022 - Mels Dees

PLONK Critical Vulnerability Successfully Remediated

- No user funds were at risk
- However, we don't know if the attack was carried out
- Forged proofs are **indistinguishable** from honest proofs

Weak F-S Attack:

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

Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

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

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

Proof Relation:

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

Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

Proof Relation:

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

Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

Proof Relation:

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

Weak F-S Attack:

Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

Proof Relation:

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

Weak F-S Attack:

Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

Proof Relation:

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

Weak F-S Attack:

- Choose v_{in}, v_{out} to satisfy equality check **as well as** BP verification

Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

Proof Relation:

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

Weak F-S Attack:

- Choose v_{in}, v_{out} to satisfy equality check **as well as** BP verification
- Forged proofs are **indistinguishable** from honest proofs

Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

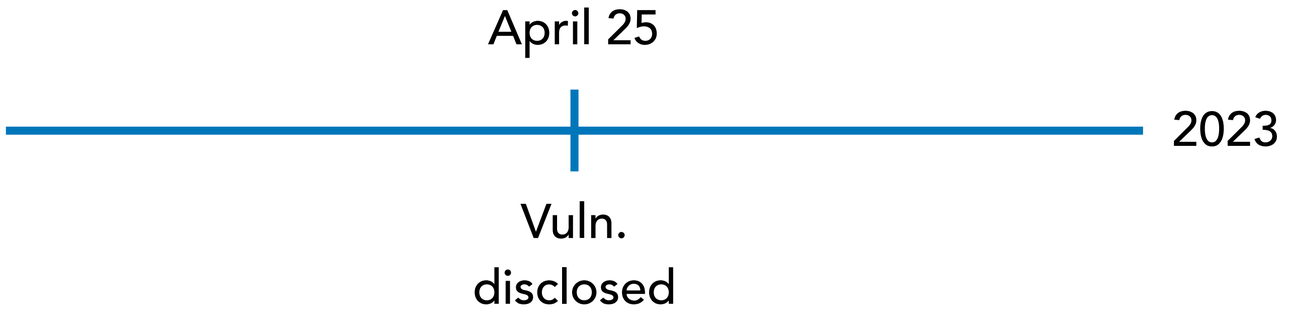
Proof Relation:

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

Weak F-S Attack:

- Choose v_{in}, v_{out} to satisfy equality check **as well as** BP verification
- Forged proofs are **indistinguishable** from honest proofs

Disclosure Timeline:



Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

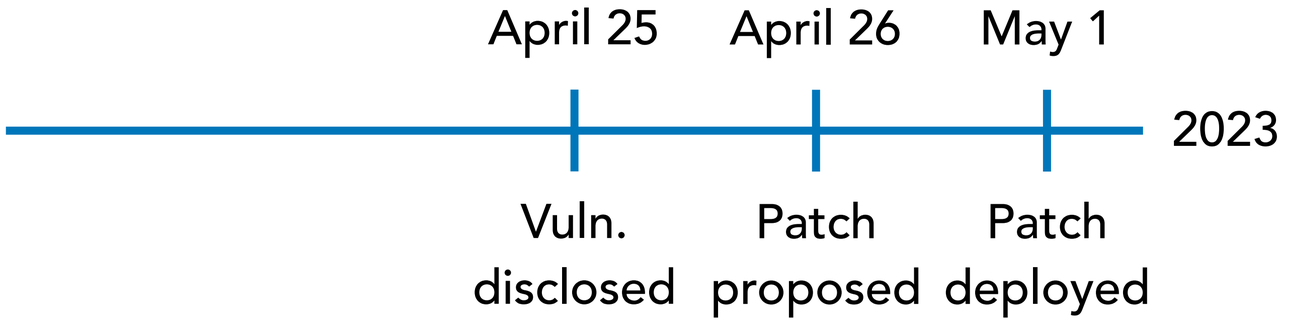
Proof Relation:

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

Weak F-S Attack:

- Choose v_{in}, v_{out} to satisfy equality check **as well as** BP verification
- Forged proofs are **indistinguishable** from honest proofs

Disclosure Timeline:



Case Study: Incognito Chain

Incognito

$250M+	+6M	100+	16
Volume shielded	Anonymous transactions	Coins supported	Bridges supported

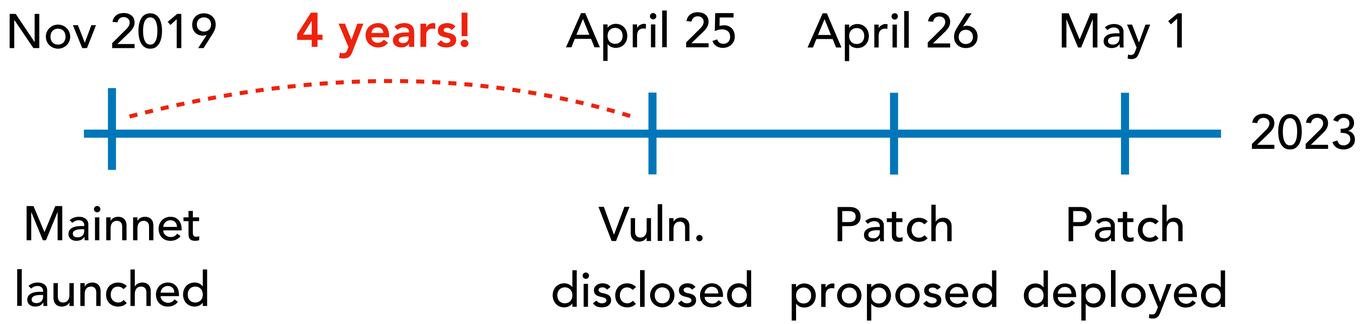
Proof Relation:

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

Weak F-S Attack:

- Choose v_{in}, v_{out} to satisfy equality check **as well as** BP verification
- Forged proofs are **indistinguishable** from honest proofs

Disclosure Timeline:



Weak Fiat-Shamir Attacks

Practical Impacts

Why is Weak F-S so widespread?

(and how to prevent it?)

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.

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

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

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

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

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 $\mathcal{R}_{\text{R1CS}}$.[14]

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:

$\Longrightarrow$ (some) do 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:

$\Longrightarrow$ (some) do 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:

$\Longrightarrow$ (some) do 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 hueristic. 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:

$\Longrightarrow$ (some) do 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:

$\Longrightarrow$ (some) do not get it right on the first try!

Plonk:

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

(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 $y = \mathsf{H}(\mathsf{st}, A, S)$ and $z = \mathsf{H}(A, S, y)$, where st is the statement.

(April 2022)

(in response to our FrozenHeart disclosure)

How to Prevent Weak Fiat-Shamir?

How to Prevent Weak Fiat-Shamir?

Merlin prevents some F-S issues

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

How to Prevent Weak Fiat-Shamir?

Merlin prevents some F-S issues

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

```rust
let mut transcript = Merlin::new("schnorr proof")?;

// Proof parameters
let target = BigInt::from(8675309u32);
let base = BigInt::from(43u32);
let log = BigInt::parse_bytes(b"187777970837149957259676149979333086l5", 10).unwrap();
let modulus = &BigInt::from(2u32).pow(127) - BigInt::from(1u32);

// Random exponent
let mut rng = rand::thread_rng();
let randomizer_exp = rng.gen_bigint(256) % (&modulus - BigInt::from(1u32));
let randomizer = base.modpow(&randomizer_exp, &modulus);
💡
// Update transcript — order of addition matters!
transcript.append_message("modulus", &modulus);
transcript.append_message("base", &base);
transcript.append_message("target", &target);
transcript.append_message("u", &randomizer);

// Generate challenge
let mut challenge_out: [u8; 32] = [0u8; 32];
transcript.challenge_bytes("c_challenge", &mut challenge_out);
```

How to Prevent Weak Fiat-Shamir?

Merlin prevents some F-S issues

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

```rust
let mut transcript = Merlin::new("schnorr proof")?;

// Proof parameters
let target = BigInt::from(8675309u32);
let base = BigInt::from(43u32);
let log = BigInt::parse_bytes(b"187777970837149957259676149979333086l5", 10).unwrap();
let modulus = &BigInt::from(2u32).pow(127) - BigInt::from(1u32);

// Random exponent
let mut rng = rand::thread_rng();
let randomizer_exp = rng.gen_bigint(256) % (&modulus - BigInt::from(1u32));
let randomizer = base.modpow(&randomizer_exp, &modulus);
💡
// Update transcript — order of addition matters!
transcript.append_message("modulus", &modulus);
transcript.append_message("base", &base);
transcript.append_message("target", &target);
transcript.append_message("u", &randomizer);

// Generate challenge
let mut challenge_out: [u8; 32] = [0u8; 32];
transcript.challenge_bytes("c_challenge", &mut challenge_out);
```

Limitations of Merlin

How to Prevent Weak Fiat-Shamir?

Merlin prevents some F-S issues

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

```
let mut transcript = Merlin::new("schnorr proof")?;

// Proof parameters
let target = BigInt::from(8675309u32);
let base = BigInt::from(43u32);
let log = BigInt::parse_bytes(b"187777970837149957259676149979333086l5", 10).unwrap();
let modulus = &BigInt::from(2u32).pow(127) - BigInt::from(1u32);

// Random exponent
let mut rng = rand::thread_rng();
let randomizer_exp = rng.gen_bigint(256) % (&modulus - BigInt::from(1u32));
let randomizer = base.modpow(&randomizer_exp, &modulus);
💡
// Update transcript — order of addition matters!
transcript.append_message("modulus", &modulus);
transcript.append_message("base", &base);
transcript.append_message("target", &target);
transcript.append_message("u", &randomizer);

// Generate challenge
let mut challenge_out: [u8; 32] = [0u8; 32];
transcript.challenge_bytes("c_challenge", &mut challenge_out);
```

Limitations of Merlin

- No detection of missing, duplicated, or incorrect inputs
- No enforcement of ordering for inputs & challenges
- Unclear boundaries for protocol stages
- Limited auditability, i.e., “the code is the spec”

Preventing Weak F-S with Decree

Improving on Merlin with Decree

Limitations of Merlin

- No detection of missing, duplicated, or incorrect inputs
- No enforcement of ordering for inputs & challenges
- Unclear boundaries for protocol stages
- Limited auditability, i.e., “the code is the spec”

Preventing Weak F-S with Decree

Improving on Merlin with Decree

- **Full specification of protocol flow:**
  - Missing, duplicate, or incorrect inputs raise an error
- **Canonical ordering for inputs & enforcing challenge ordering**
- **Explicit boundaries for multi-round protocols**

Limitations of Merlin

- No detection of missing, duplicated, or incorrect inputs
- No enforcement of ordering for inputs & challenges
- Unclear boundaries for protocol stages
- Limited auditability, i.e., “the code is the spec”

Preventing Weak F-S with Decree

Improving on Merlin with Decree

- **Full specification of protocol flow:**
  - Missing, duplicate, or incorrect inputs raise an error
- **Canonical ordering for inputs & enforcing challenge ordering**
- **Explicit boundaries for multi-round protocols**

Preventing Weak F-S with Decree

Improving on Merlin with Decree

- **Full specification of protocol flow:**
  - Missing, duplicate, or incorrect inputs raise an error
- **Canonical ordering for inputs & enforcing challenge ordering**
- **Explicit boundaries for multi-round protocols**

Implementing Schnorr with Decree

Preventing Weak F-S with Decree

Improving on Merlin with Decree

- Full specification of protocol flow:
  - Missing, duplicate, or incorrect inputs raise an error
- Canonical ordering for inputs & enforcing challenge ordering
- Explicit boundaries for multi-round protocols

Implementing Schnorr with Decree

```
let inputs: [InputLabel; 4] = ["modulus", "base", "target", "u"];
let challenges: [ChallengeLabel; 1] = ["c_challenge"];
let mut transcript = Decree::new("schnorr proof", &inputs, &challenges)?;

// Proof parameters
let target = BigInt::from(8675309u32);
let base = BigInt::from(43u32);
let log = BigInt::parse_bytes(b"187777970837149957259676149979333086l5", 10).unwrap();
let modulus = &BigInt::from(2u32).pow(127) - BigInt::from(1u32);

// Random exponent
let mut rng = rand::thread_rng();
let randomizer_exp = rng.gen_bigint(256) % (&modulus - BigInt::from(1u32));
let randomizer = base.modpow(&randomizer_exp, &modulus);
💡
// Update transcript - order of addition doesn't matter!
transcript.add_serial("u", &randomizer);
transcript.add_serial("target", &target);
transcript.add_serial("base", &base);
transcript.add_serial("modulus", &modulus);

// Generate challenge
let mut challenge_out: [u8; 32] = [0u8; 32];
transcript.get_challenge("c_challenge", &mut challenge_out);
```

Preventing Weak F-S with Decree

Improving on Merlin with Decree

```rust
let mut transcript = Merlin::new("schnorr proof")?;

// Proof parameters
let target = BigInt::from(8675309u32);
let base = BigInt::from(43u32);
let log = BigInt::parse_bytes(b"187777970837149957259676149979333308615", 10).unwrap();
let modulus = &BigInt::from(2u32).pow(127) - BigInt::from(1u32);

// Random exponent
let mut rng = rand::thread_rng();
let randomizer_exp = rng.gen_bigint(256) % (&modulus - BigInt::from(1u32));
let randomizer = base.modpow(&randomizer_exp, &modulus);
💡
// Update transcript - order of addition matters!
transcript.append_message("modulus", &modulus);
transcript.append_message("base", &base);
transcript.append_message("target", &target);
transcript.append_message("u", &randomizer);

// Generate challenge
let mut challenge_out: [u8; 32] = [0u8; 32];
transcript.challenge_bytes("c_challenge", &mut challenge_out);
```

Implementing Schnorr with Decree

```rust
let inputs: [InputLabel; 4] = ["modulus", "base", "target", "u"];
let challenges: [ChallengeLabel; 1] = ["c_challenge"];
let mut transcript = Decree::new("schnorr proof", &inputs, &challenges)?;

// Proof parameters
let target = BigInt::from(8675309u32);
let base = BigInt::from(43u32);
let log = BigInt::parse_bytes(b"187777970837149957259676149979333308615", 10).unwrap();
let modulus = &BigInt::from(2u32).pow(127) - BigInt::from(1u32);

// Random exponent
let mut rng = rand::thread_rng();
let randomizer_exp = rng.gen_bigint(256) % (&modulus - BigInt::from(1u32));
let randomizer = base.modpow(&randomizer_exp, &modulus);
💡
// Update transcript - order of addition doesn't matter!
transcript.add_serial("u", &randomizer);
transcript.add_serial("target", &target);
transcript.add_serial("base", &base);
transcript.add_serial("modulus", &modulus);

// Generate challenge
let mut challenge_out: [u8; 32] = [0u8; 32];
transcript.get_challenge("c_challenge", &mut challenge_out);
```

Preventing Weak F-S with Decree

Improving on Merlin with Decree

```rust
let mut transcript = Merlin::new("schnorr proof")?;

// Proof parameters
let target = BigInt::from(8675309u32);
let base = BigInt::from(43u32);
let log = BigInt::parse_bytes(b"187777970837149957259676149979333308615", 10).unwrap();
let modulus = &BigInt::from(2u32).pow(127) - BigInt::from(1u32);

// Random exponent
let mut rng = rand::thread_rng();
let randomizer_exp = rng.gen_bigint(256) % (&modulus - BigInt::from(1u32));
let randomizer = base.modpow(&randomizer_exp, &modulus);
💡
// Update transcript - order of addition matters!
transcript.append_message("modulus", &modulus);
transcript.append_message("base", &base);
transcript.append_message("target", &target);
transcript.append_message("u", &randomizer);

// Generate challenge
let mut challenge_out: [u8; 32] = [0u8; 32];
transcript.challenge_bytes("c_challenge", &mut challenge_out);
```

Implementing Schnorr with Decree

```rust
let inputs: [InputLabel; 4] = ["modulus", "base", "target", "u"];
let challenges: [ChallengeLabel; 1] = ["c_challenge"];
let mut transcript = Decree::new("schnorr proof", &inputs, &challenges)?;

// Proof parameters
let target = BigInt::from(8675309u32);
let base = BigInt::from(43u32);
let log = BigInt::parse_bytes(b"187777970837149957259676149979333308615", 10).unwrap();
let modulus = &BigInt::from(2u32).pow(127) - BigInt::from(1u32);

// Random exponent
let mut rng = rand::thread_rng();
let randomizer_exp = rng.gen_bigint(256) % (&modulus - BigInt::from(1u32));
let randomizer = base.modpow(&randomizer_exp, &modulus);
💡
// Update transcript - order of addition doesn't matter!
transcript.add_serial("u", &randomizer);
transcript.add_serial("target", &target);
transcript.add_serial("base", &base);
transcript.add_serial("modulus", &modulus);

// Generate challenge
let mut challenge_out: [u8; 32] = [0u8; 32];
transcript.get_challenge("c_challenge", &mut challenge_out);
```

Long-term: Standardization of Fiat-Shamir

ZKPROOF ZKDocs

Summary & Future Directions

Summary & Future Directions

Takeaways:

Summary & Future Directions

Takeaways:

1. **Never** implement **weak** Fiat-Shamir!

Summary & Future Directions

Takeaways:

1. **Never** implement **weak** Fiat-Shamir!
2. **For Academics:**

Summary & Future Directions

Takeaways:

1. **Never** implement **weak** Fiat-Shamir!
2. **For Academics:**
   - Specify the **correct** Fiat-Shamir transform!

Summary & Future Directions

Takeaways:

1. **Never** implement **weak** Fiat-Shamir!
2. **For Academics:**
   - Specify the **correct** Fiat-Shamir transform!
3. **For Practitioners:** Use tooling for Fiat-Shamir

Summary & Future Directions

Takeaways:

1. **Never** implement **weak** Fiat-Shamir!
2. **For Academics:**
   - Specify the **correct** Fiat-Shamir transform!
3. **For Practitioners:** Use tooling for Fiat-Shamir
   - Our tool **Decree** is coming soon!

Summary & Future Directions

Takeaways:

1. **Never** implement **weak** Fiat-Shamir!
2. **For Academics:**
   - Specify the **correct** Fiat-Shamir transform!
3. **For Practitioners:** Use tooling for Fiat-Shamir
   - Our tool **Decree** is coming soon!

Future Directions:

Summary & Future Directions

Takeaways:

1. **Never** implement **weak** Fiat-Shamir!
2. **For Academics:**
   - Specify the **correct** Fiat-Shamir transform!
3. **For Practitioners:** Use tooling for Fiat-Shamir
   - Our tool **Decree** is coming soon!

Future Directions:

- New attack vectors against proof systems?

Summary & Future Directions

Takeaways:

1. **Never** implement **weak** Fiat-Shamir!
2. **For Academics:**
   - Specify the **correct** Fiat-Shamir transform!
3. **For Practitioners:** Use tooling for Fiat-Shamir
   - Our tool **Decree** is coming soon!

Future Directions:

- New attack vectors against proof systems?

Read our paper
(ePrint 2023/691)

IEEE S&P Distinguished Paper Award!

Thank You! Questions?