ark-pq

post-quantum zero-knowledge arguments of knowledge

Alessandro Chiesa
Giacomo Fenzi
Christian Knabenhans
Andrew Zitek-Estrada

NIST Threshold Call Preview Talks #3

Presented by Alessandro Chiesa on 2026-09-30

zkSNARKs in context

compute



client-side proving

server-side proving

Applications have heterogeneous constraints

bandwidth

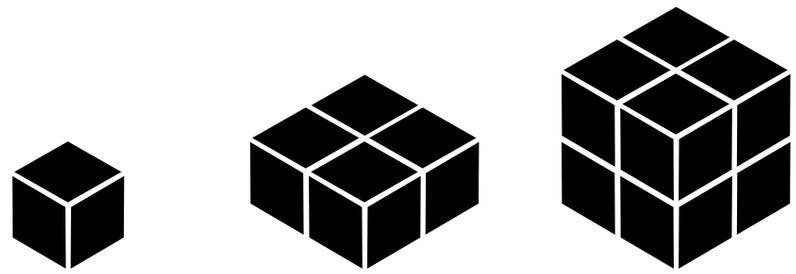


4G/5G

WiFi

Fiber/TbEth

statement complexity



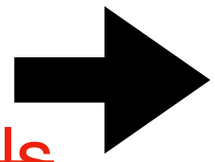
No one-size-fits-all

not scalable

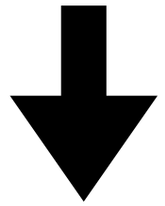
Create portfolio of protocols for every possible application?
Keep standardizing new protocols as applications arise?

ark-pq

~~Create portfolio of protocols for every possible application? Keep standardizing new protocols as applications arise?~~



Standardize **generic paradigm**



Swap out well-defined modules to achieve different trade-offs w.r.t. prover time, prover space, proof size, and verifier time

Previously:

fix a protocol ➔ ultra optimization, formal verification
stay general ➔ flexible, but not as optimized or verified

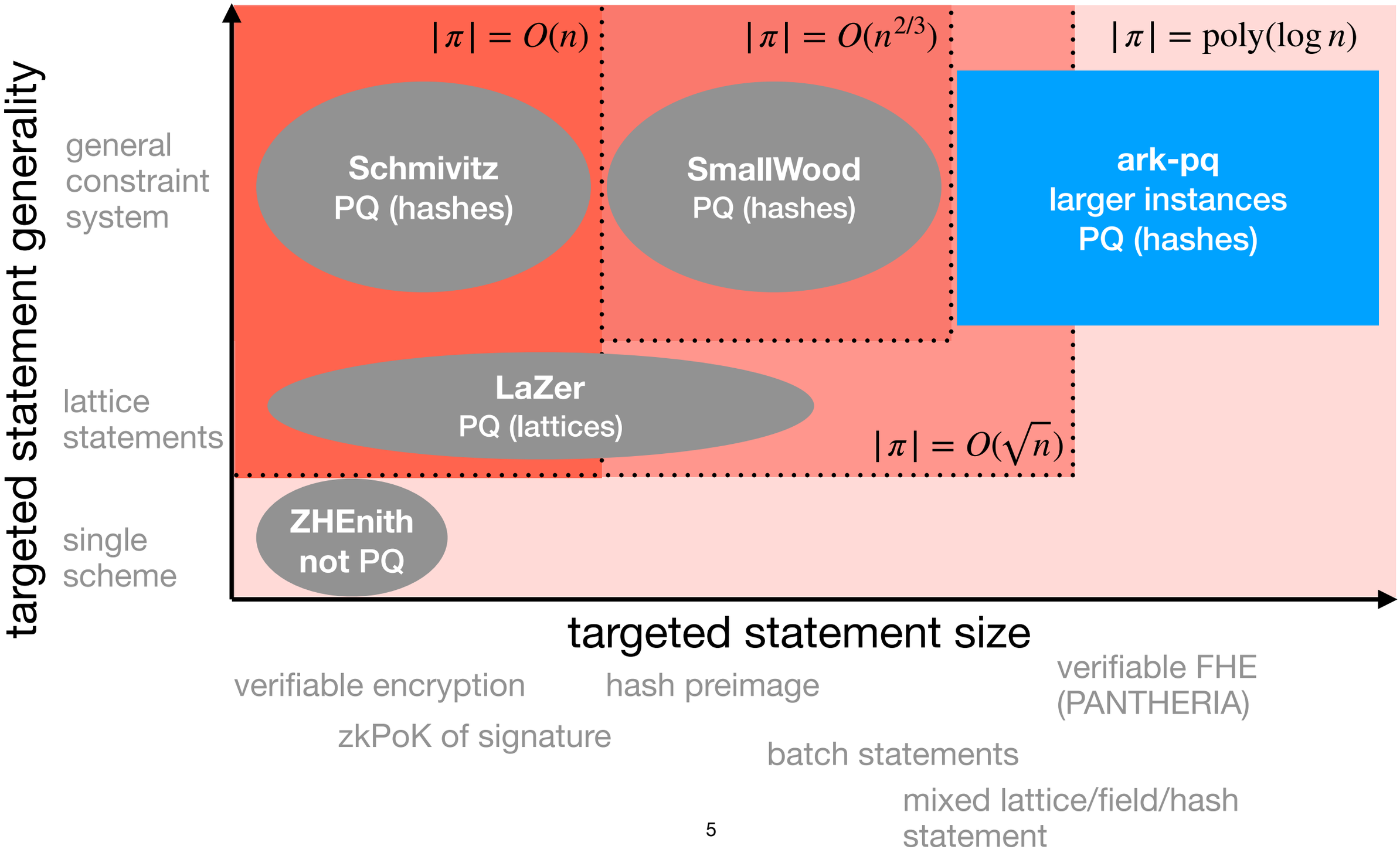
Now:

We can have both, particularly with modularity that gives well defined targets of optimization

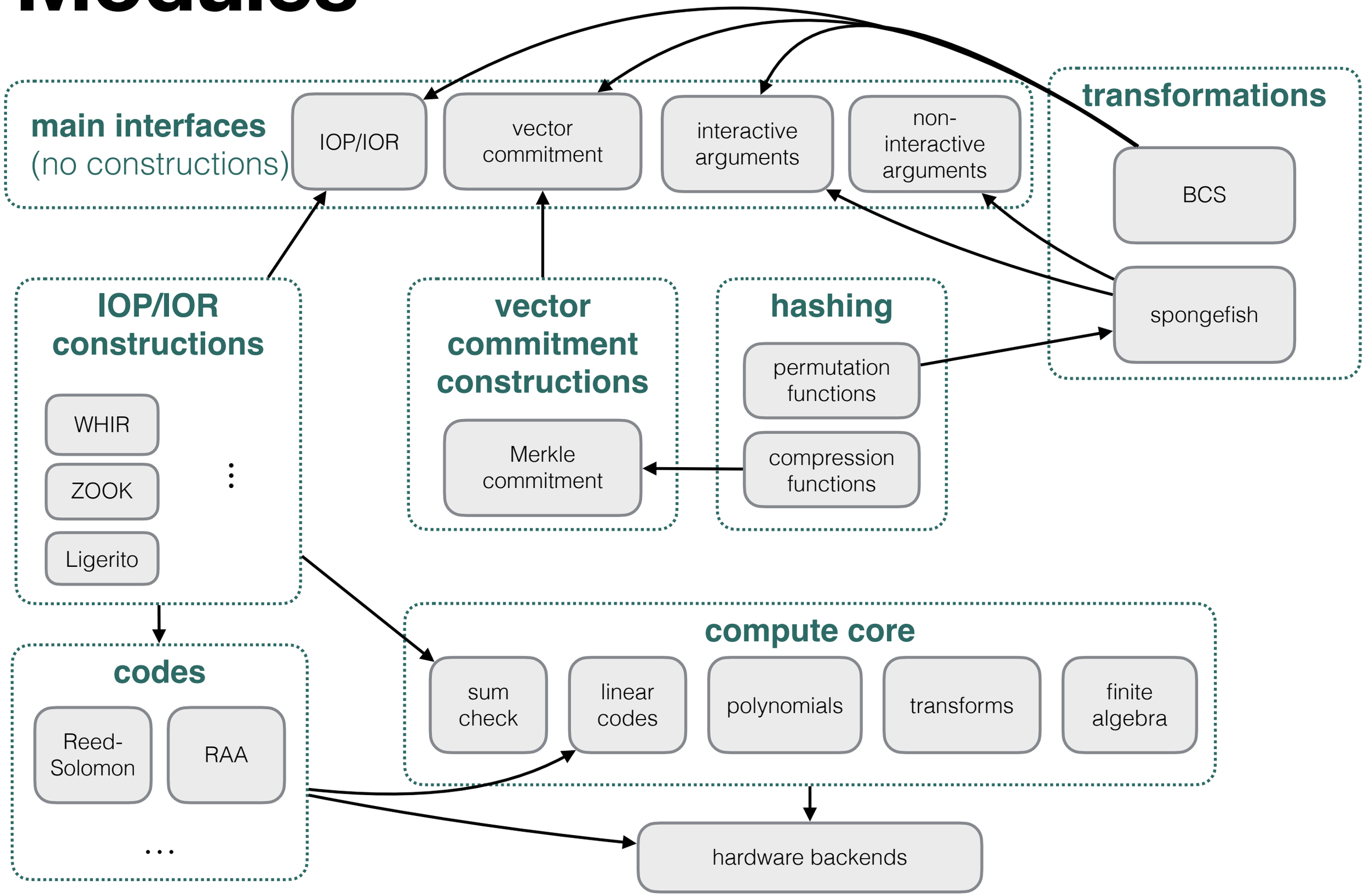
ark-pq is ***modular*** and ***configurable***, allowing for the construction of zkSNARKs trading off

- proof size,
- prover time, and
- verifier time.

ark-pq vs other S6 submissions

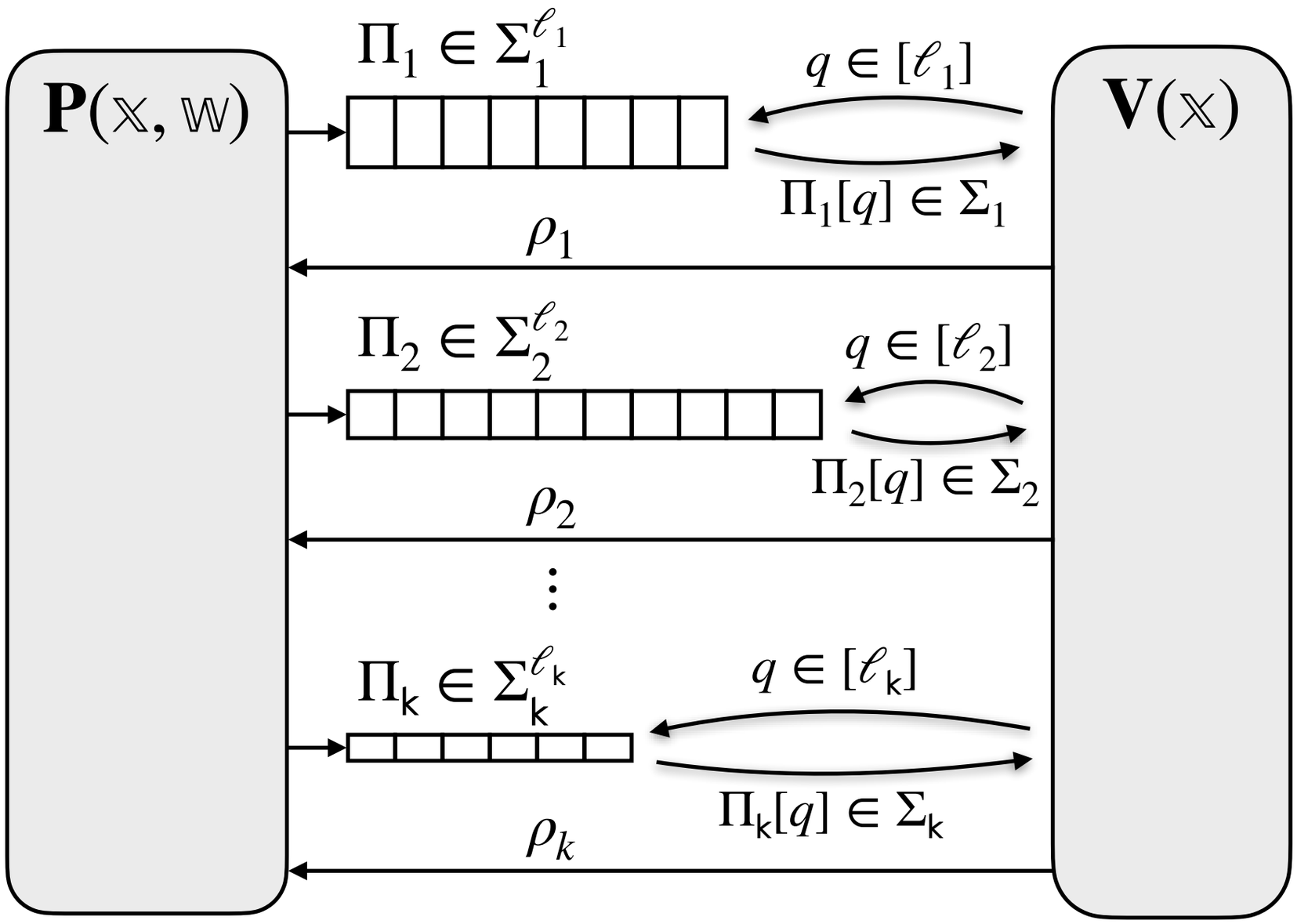


Modules


main interfaces
(no constructions)
IOP/IOR
vector commitment
interactive arguments
non-interactive arguments
transformations
BCS
spongefish
IOP/IOR constructions
WHIR
ZOOK
Ligerito
vector commitment constructions
Merkle commitment
hashing
permutation functions
compression functions
codes
Reed-Solomon
RAA
compute core
sum check
linear codes
polynomials
transforms
finite algebra
hardware backends

Interactive Oracle Proofs

Interactive Oracle Proofs (theory)



Interactive Oracle Proofs (ark-pq)

IOP DSL

Interfaces for translating an IOP into corresponding code

send_oracle(&msg)

send_non_oracle(&msg)

send_verifier_message(len)

Written in

IOP library

WHIR

ZOOK

Ligerito

⋮

Codes

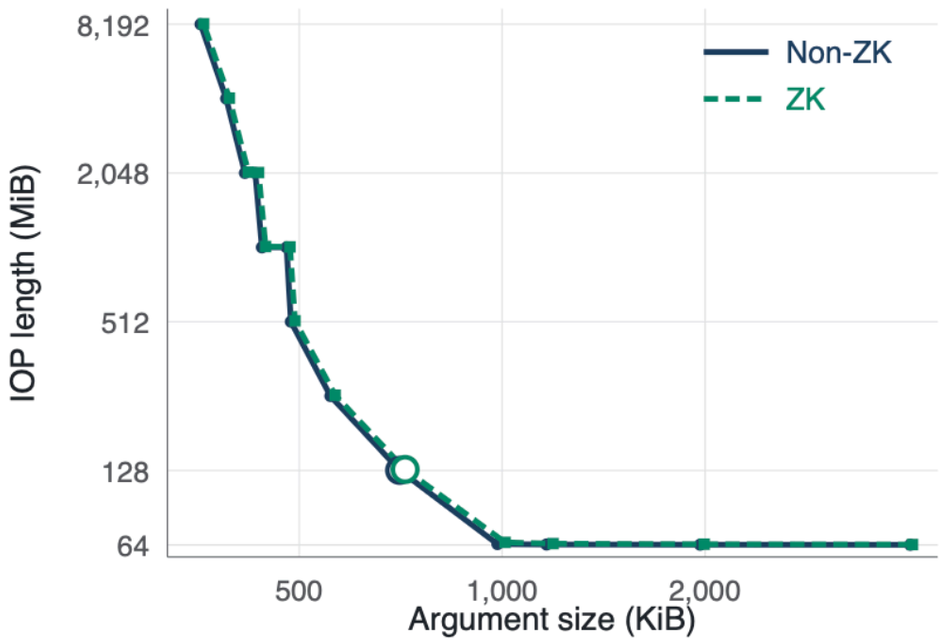
Leads to

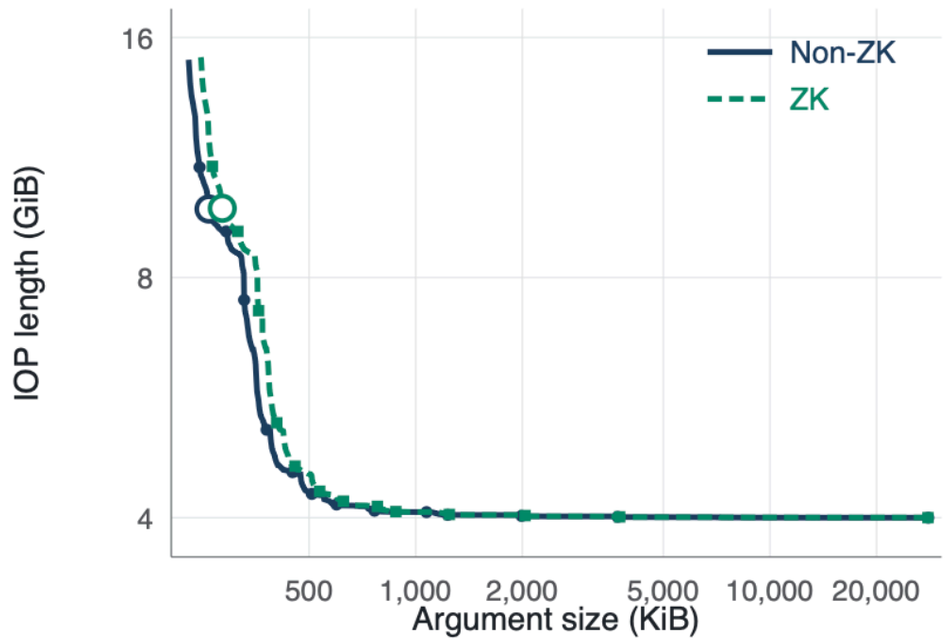
Transparent setup (zk)SNARKs

Secure in the **pure** ROM

Fast and **configurable**

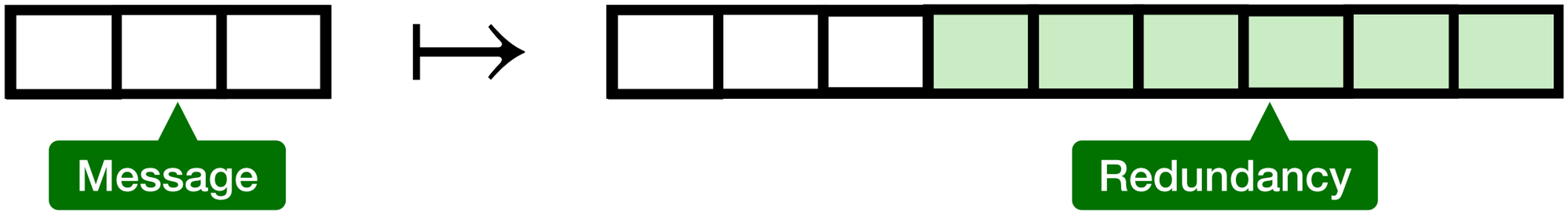
Proof size ~ 100 KiB to few MiB





Error correcting codes (ark-pq)

Modern IOP are “code-agnostic”: generic in the underlying linear code $\mathscr{C}$



Codes

+ ZK!

Additive codes

Linear codes

- Reed-Solomon
- Generalized Reed-Solomon
- RAA codes
- Generalized Reed-Muller
- Sketched codes
- ⋮

Code transformations

- Tensoring
- Interleaving
- Distance Amplification

Code switching

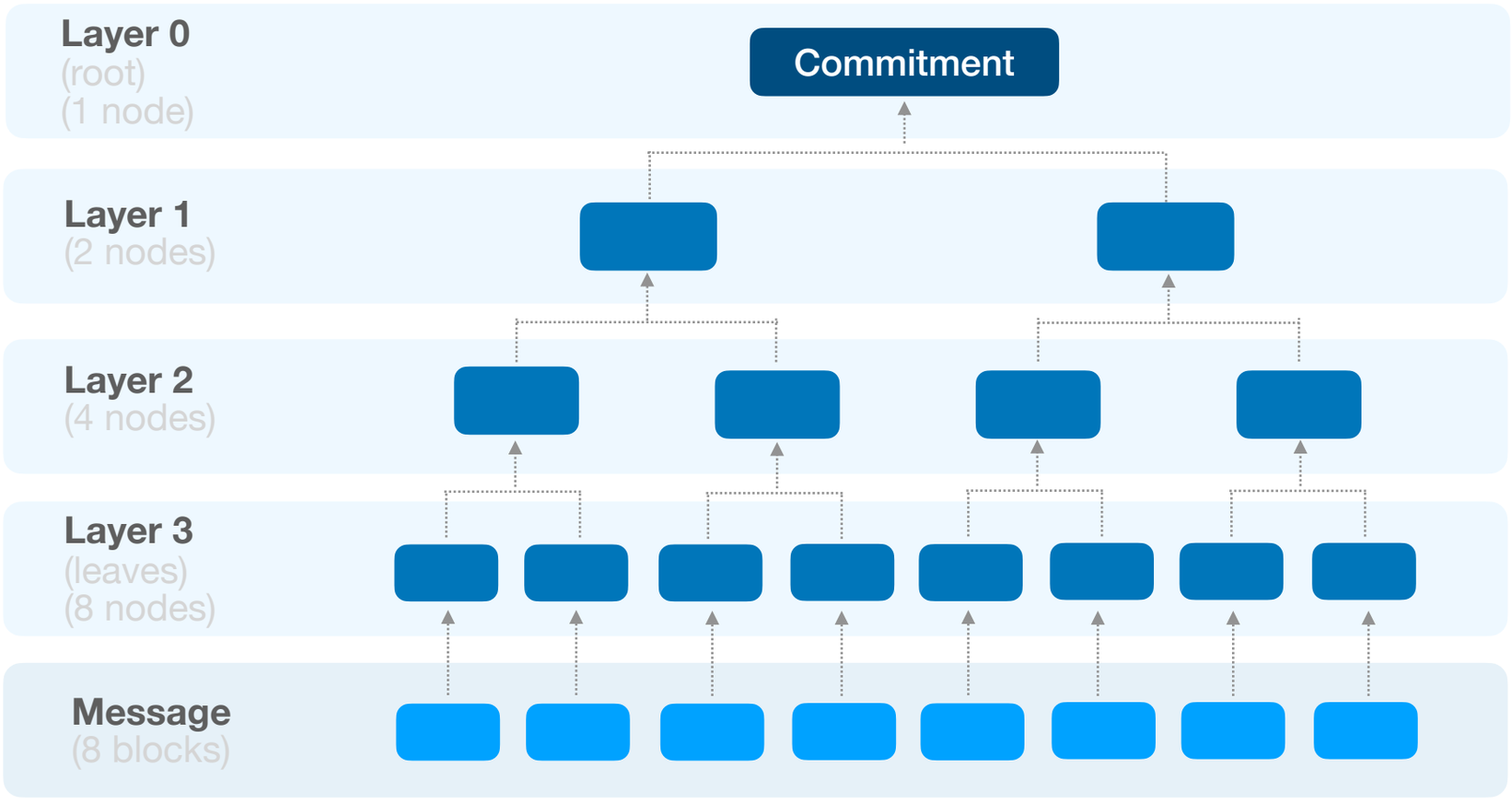
- Succinct code switching
- Generic code switching

Easily swappable

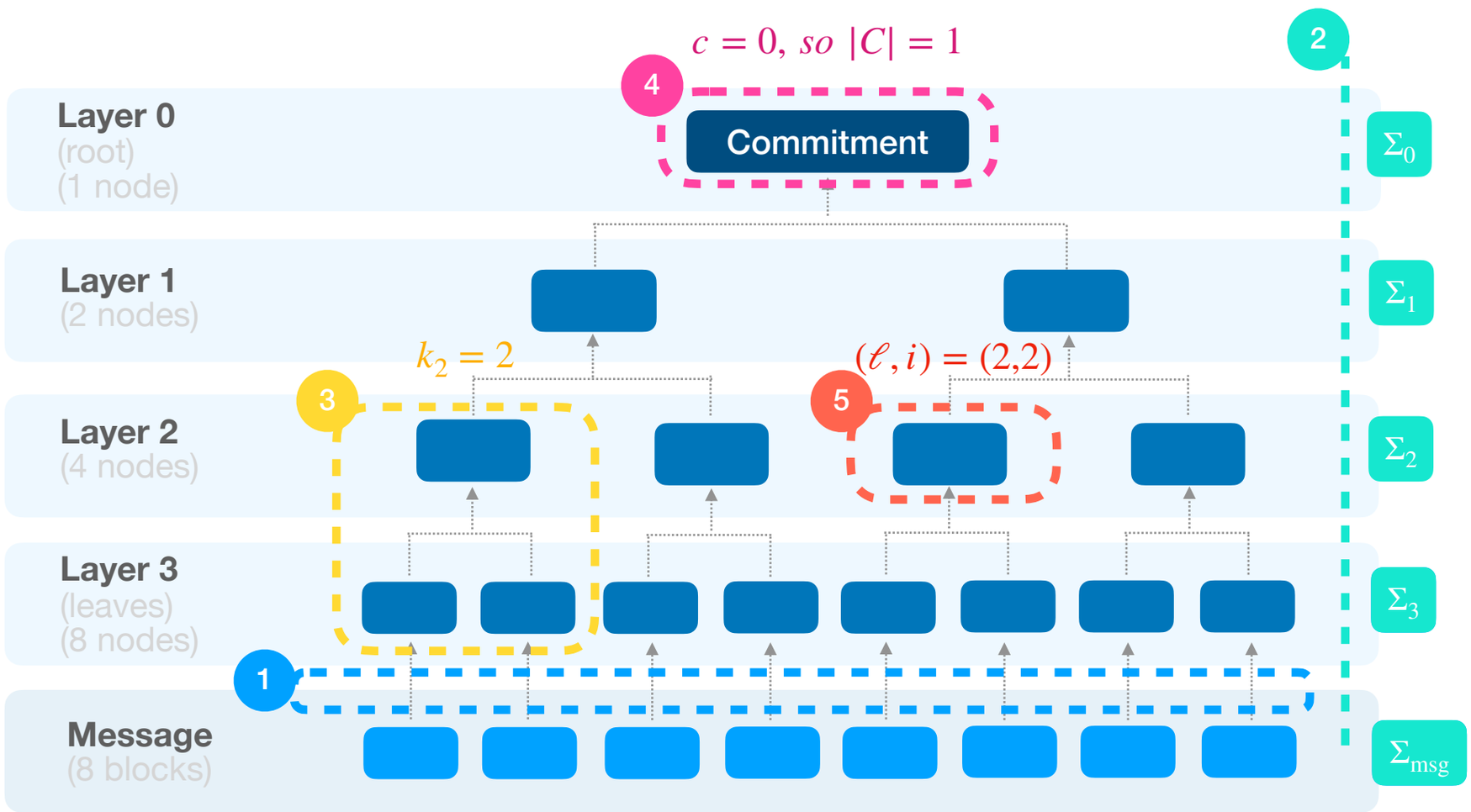
Distance, list-decoding and MCA guarantees **enforced** via ArkLib

Merkle Vector Commitments

Merkle Vector commitments (theory)



Merkle Vector commitments (ark-pq)



1 Message vs. leaf alphabet

$$\Sigma_{msg} \stackrel{?}{=} \Sigma_3$$

2 Layer compatibility

$$\text{out}(H_\ell) \stackrel{?}{=} \text{in}(H_{\ell+1})$$

3 Arity and Hashing

$$\text{choose } k_\ell \text{ and } H_\ell$$

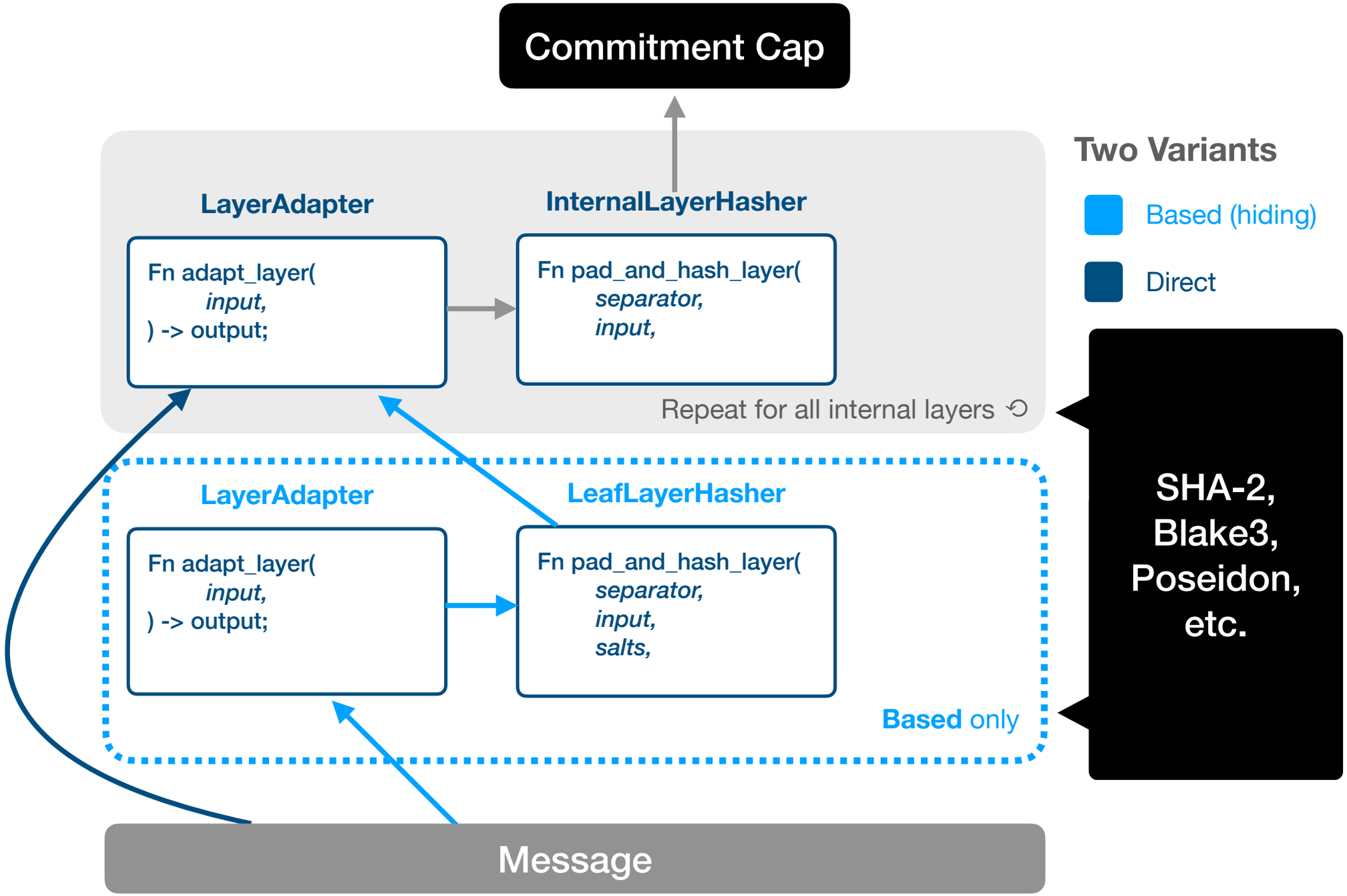
4 Capping

$$\text{choose } c, \text{defining } |C|$$

5 Domain separation

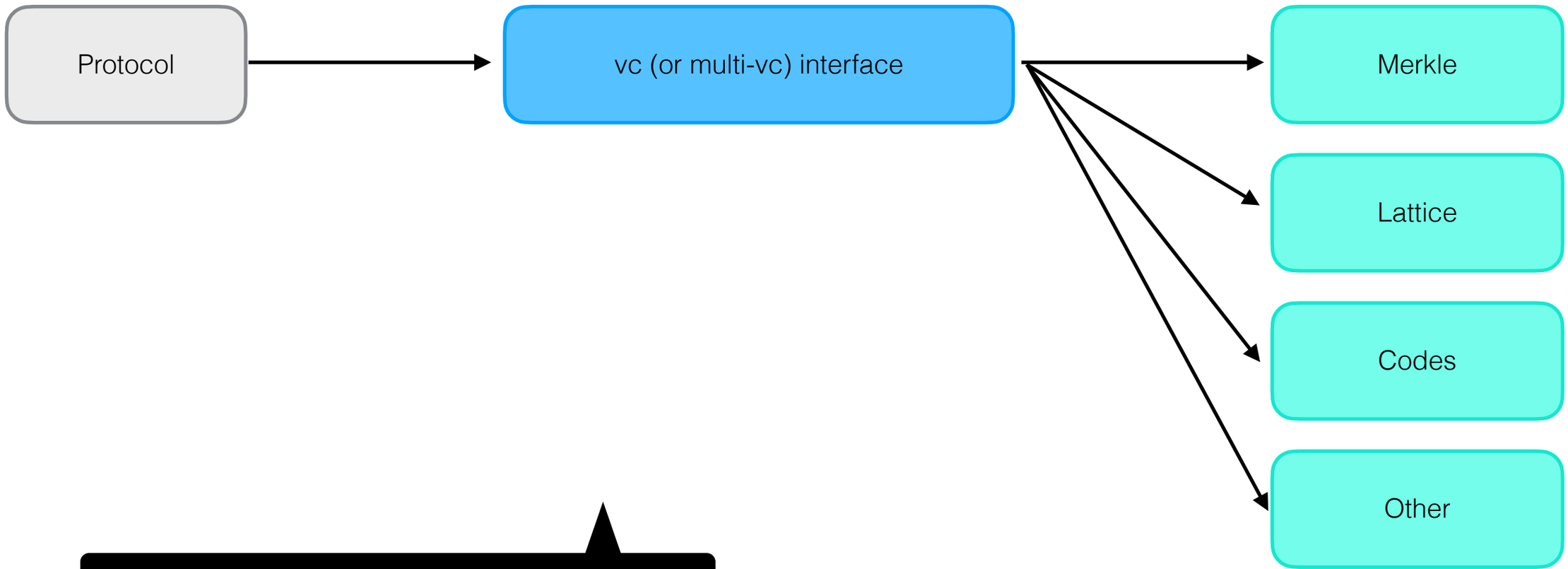
$$Hash(\underbrace{\text{pid}, \text{sid}, \ell, i}_{\text{domain sep.}}; \cdots)$$

Merkle Vector commitments (ark-pq)



Vector commitments

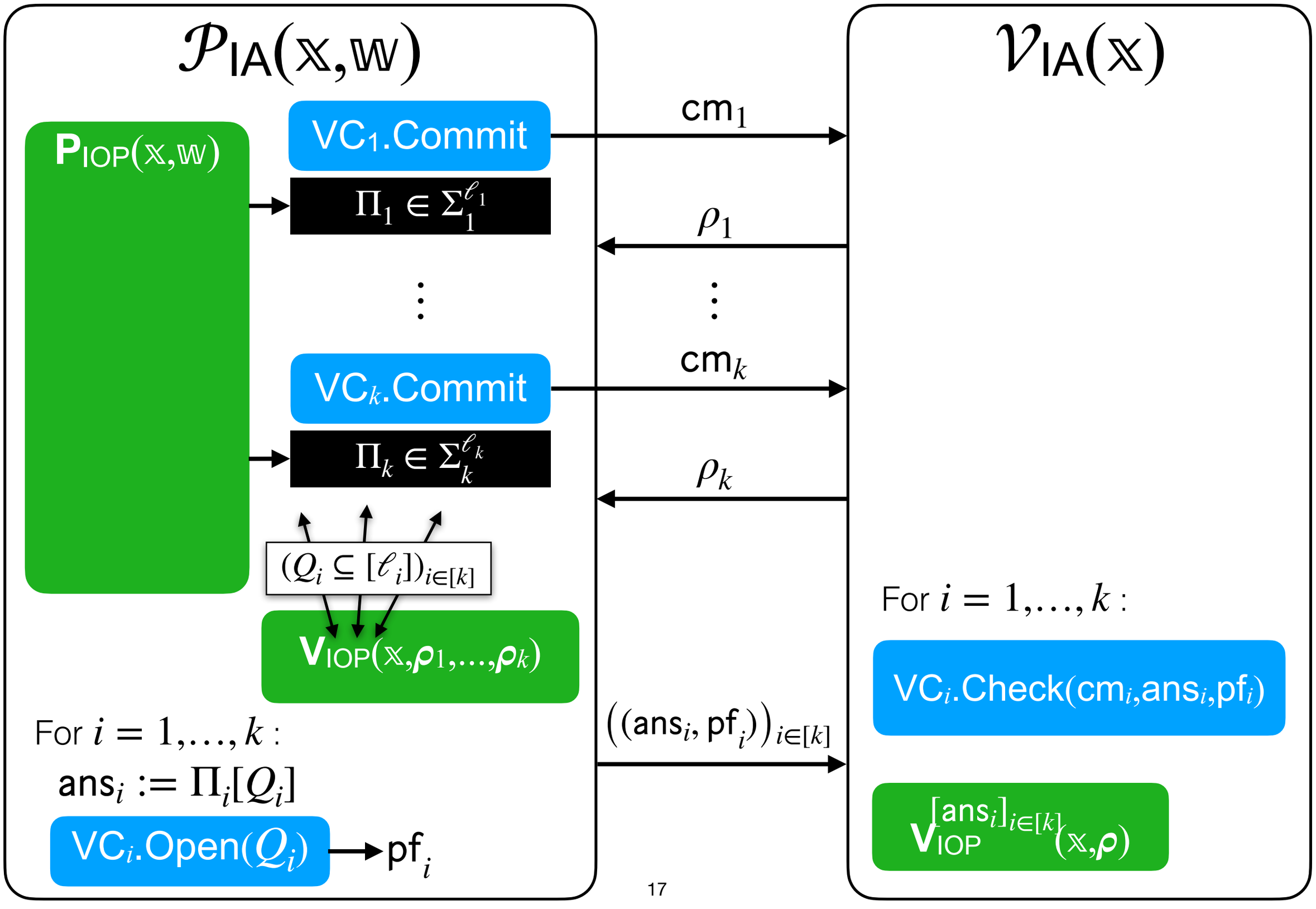
A generic vc interface keeps protocol stable while backend evolves



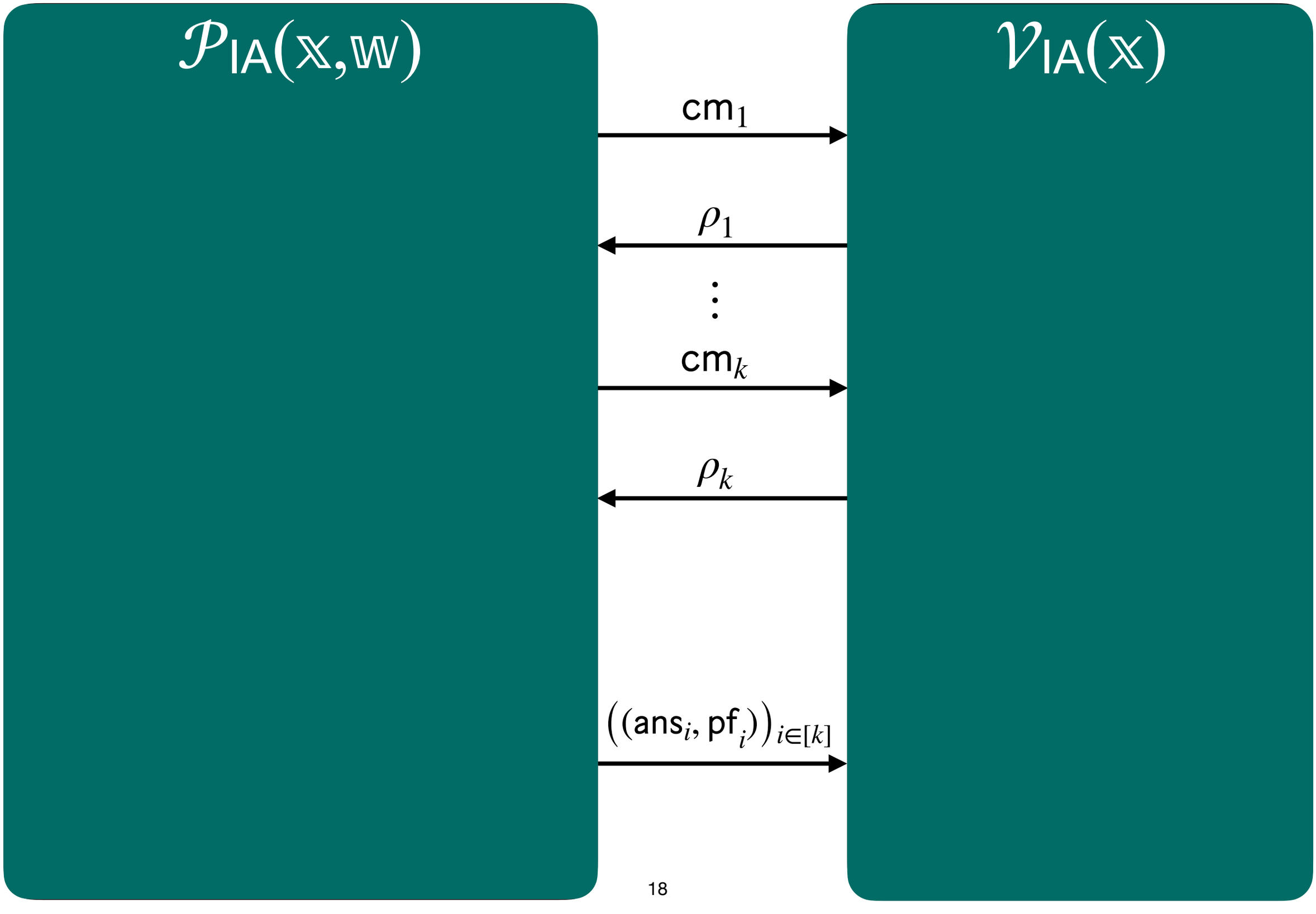
Stable interface. Flexible machinery below.

Transformations

BCS transformation (theory)



BCS transformation (theory)



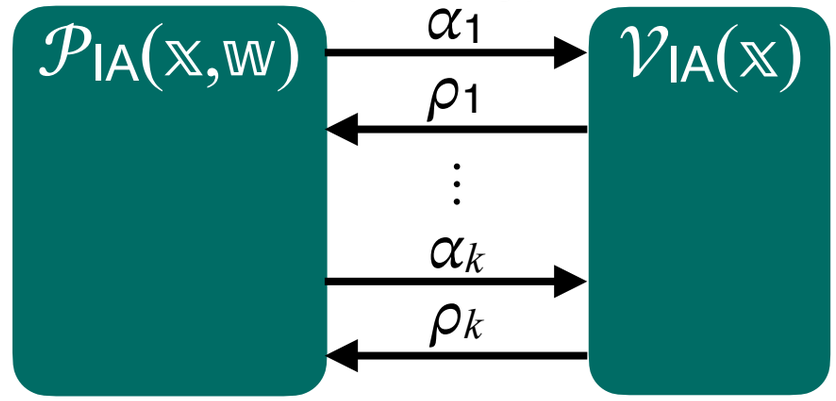
BCS transformation (ark-pq)

```
impl<IOP, VC, IopProver, IopVerifier, VcProver> InteractiveArgumentProver
    for IbcsArgumentProver<IOP, VC, IopProver, IopVerifier, VcProver>

impl<IOP, VC, IopVerifier, VcVerifier> InteractiveArgumentVerifier
    for IbcsArgumentVerifier<IOP, VC, IopVerifier, VcVerifier>
```

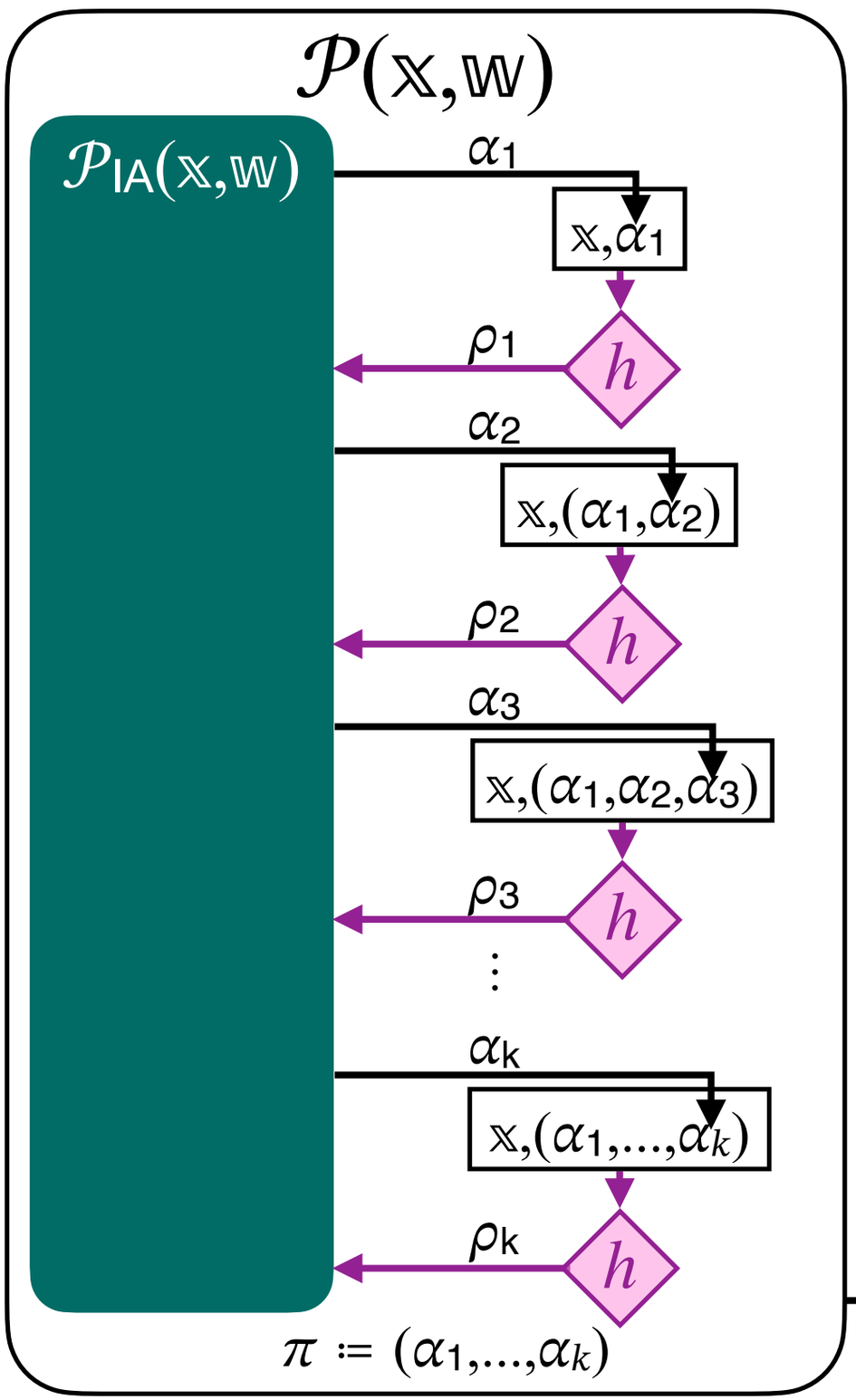
ark-pq automatically derives an argument prover and verifier, preserving security and efficiency, and preventing user error.

Fiat-Shamir transformation (theory)

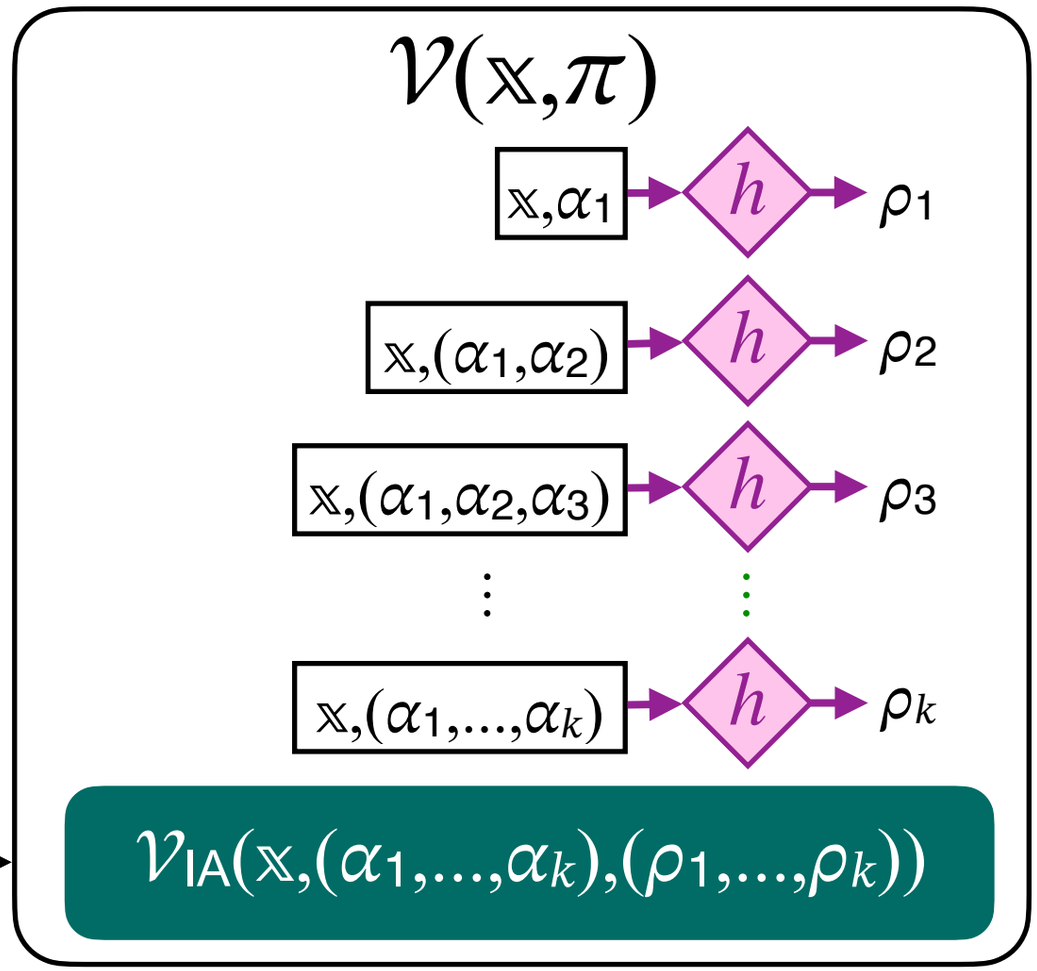


Derive i-th randomness by hashing $\mathbb{x}$ and $(\alpha_1,\ldots,\alpha_i)$.

$$(\mathcal{P},\mathcal{V}) := \mathsf{FS}_{\mathscr{H}}[(\mathcal{P}_{\mathsf{IA}},\mathcal{V}_{\mathsf{IA}})]$$



π



Fiat-Shamir transformation (ark-pq)

Crypto Forum
Internet-Draft
Intended status: Informational
Expires: 18 February 2027

M. Orrù
CNRS

17 August 2026

Fiat-Shamir Transformation
draft-irtf-cfrg-fiat-shamir-03

Abstract

This document describes the Fiat-Shamir transformation, which allows making a public-coin protocol non-interactive by means of a cryptographic hash function.

It specifies how the hash function is employed, how prover messages are encoded as hash-function input, and how verifier messages are decoded from the hash function's output, as well as the serialization and deserialization of the non-interactive argument string.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://mmaker.github.io/draft-irtf-cfrg-sigma-protocols/draft-irtf-cfrg-fiat-shamir.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-irtf-cfrg-fiat-shamir/.

Discussion of this document takes place on the Crypto Forum Research Group mailing list (mailto:cfrg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/cfrg. Subscribe at https://www.ietf.org/mailman/listinfo/cfrg/.

Source for this draft and an issue tracker can be found at

ark-pq

post-quantum zero-knowledge arguments of knowledge

Alessandro Chiesa
Giacomo Fenzi
Christian Knabenhans
Andrew Zitek-Estrada

NIST Threshold Call Preview Talks #3