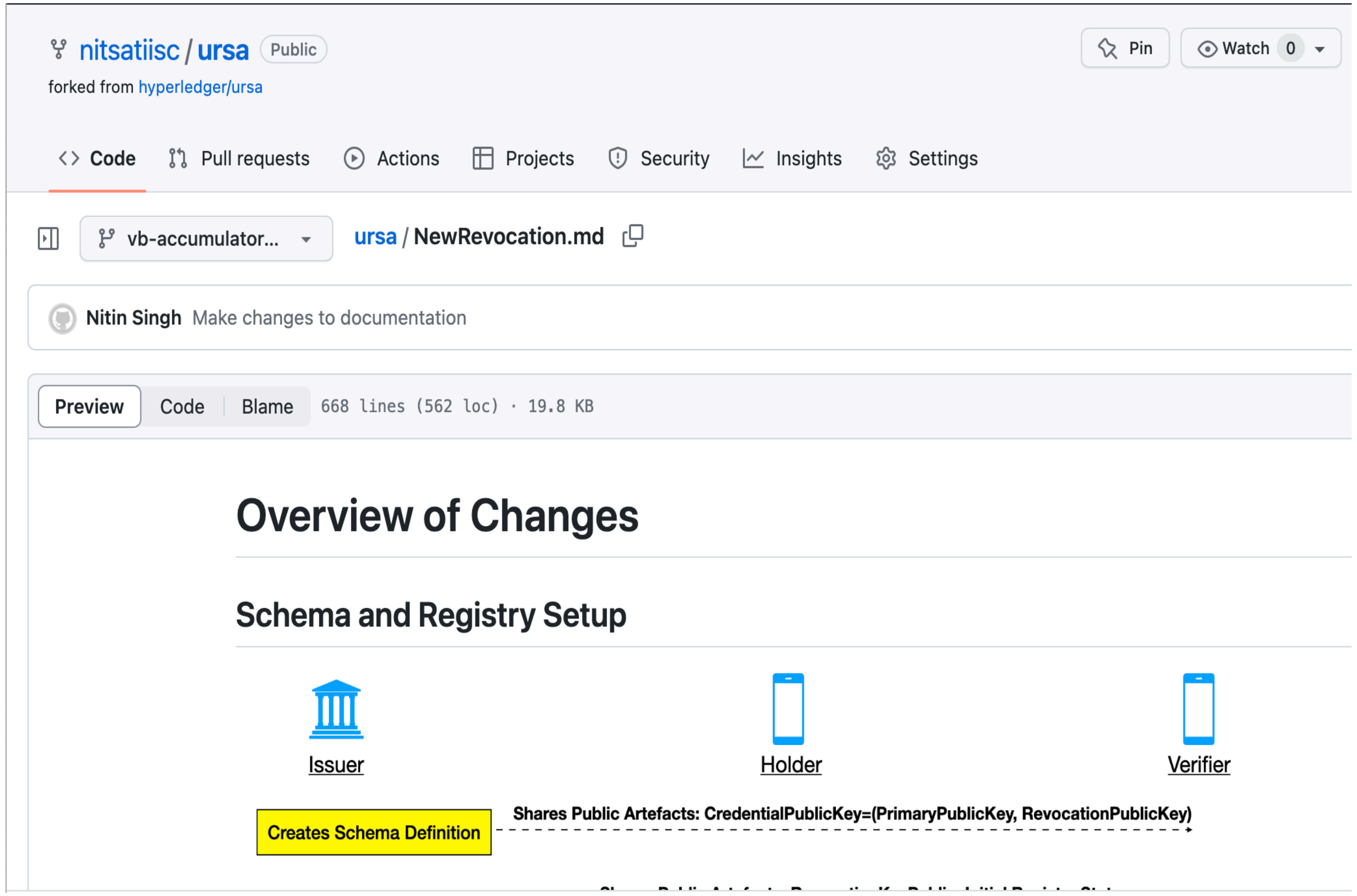
VB Accumulator Integration in Ursa

- Ursa Fork: https://github.com/nitsatiisc/ursa.git
- Branch: vb-accumulator-changes
- Overview: NewRevocation.md
- Tests & Benchmarks: https://github.com/nitsatiisc/ursa/blob/vb-accumulator-changes/libursa/tests/test_generic_interface.rs
- VB Scheme: https://link.springer.com/chapter/10.1007/978-3-030-95312-6_17



nitsatiisc / ursa Public

forked from hyperledger/ursa

ursa / NewRevocation.md

Nitin Singh Make changes to documentation

668 lines (562 loc) · 19.8 KB

Overview of Changes

Schema and Registry Setup

Creates Schema Definition

Shares Public Artefacts: CredentialPublicKey=(PrimaryPublicKey, RevocationPublicKey)

Motivation

- To overcome the practical challenges in rolling out eixsting credential revocation for mass issuers (like New York State) for several million credentials.

    - *For example, supporting revocation registries accounting for 20 million credentials would require dissemination of 16GB of public parameters (as tails vectors) to holder devices (hundreds of MB each).*

Key Advantages

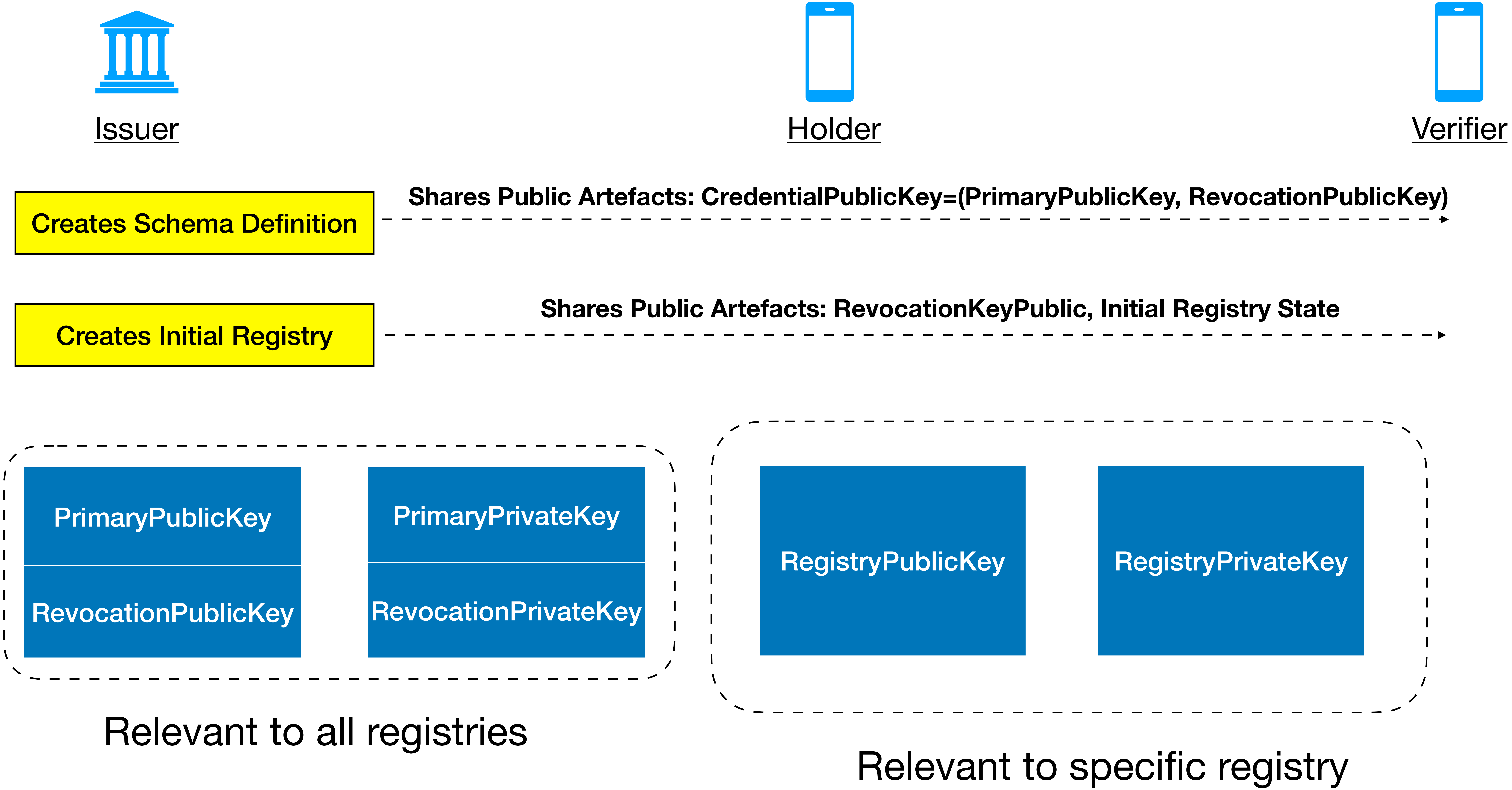
- The Vitto-Biryukov accumulator [1] supports efficient operations (proof, verification, witness update) without requiring dissemination of large public parameters (no gigabytes of public parameters).

- Verification of non-revocation proof requires just one pairing check using modified proof of knowledge on the lines of improved proof of knowledge of the BBS+ signature from [2].

1. **Giuseppe Vitto, Alex Biryukov: Dynamic Universal Accumulator with Batch Update over Bilinear Groups.** CT-RSA 2022.
2. **Jan Camenisch, Manu Drijvers, Anja Lehmann: Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited.** TRUST 2016.

Overiew of Key Interactions and Objects

Setup



Issuer

Holder

Verifier

Creates Schema Definition

Shares Public Artefacts: CredentialPublicKey=(PrimaryPublicKey, RevocationPublicKey)

Creates Initial Registry

Shares Public Artefacts: RevocationKeyPublic, Initial Registry State

PrimaryPublicKey

RevocationPublicKey

PrimaryPrivateKey

RevocationPrivateKey

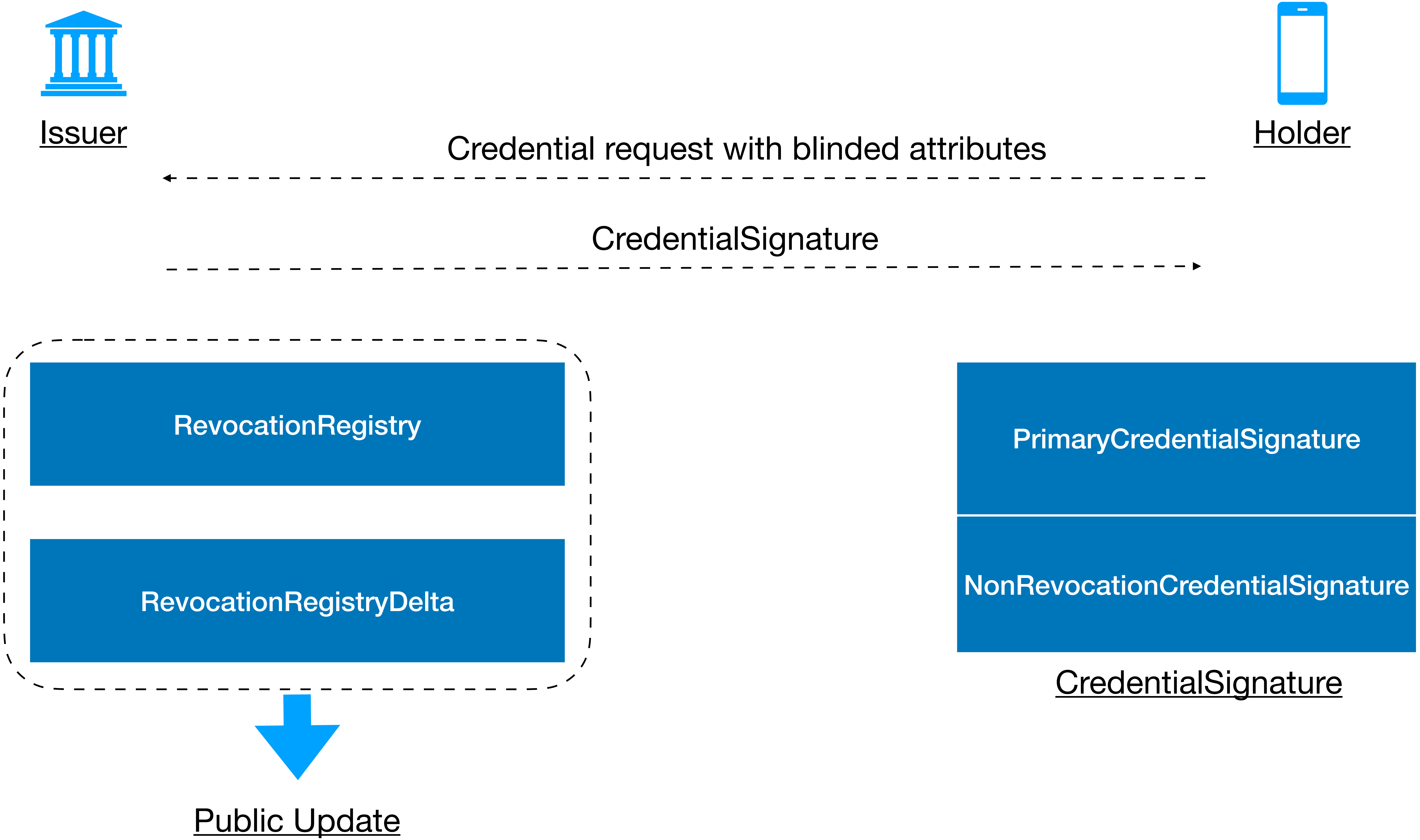
Relevant to all registries

RegistryPublicKey

RegistryPrivateKey

Relevant to specific registry

Credential Issuance



Issuer

Holder

Credential request with blinded attributes

CredentialSignature

RevocationRegistry

RevocationRegistryDelta

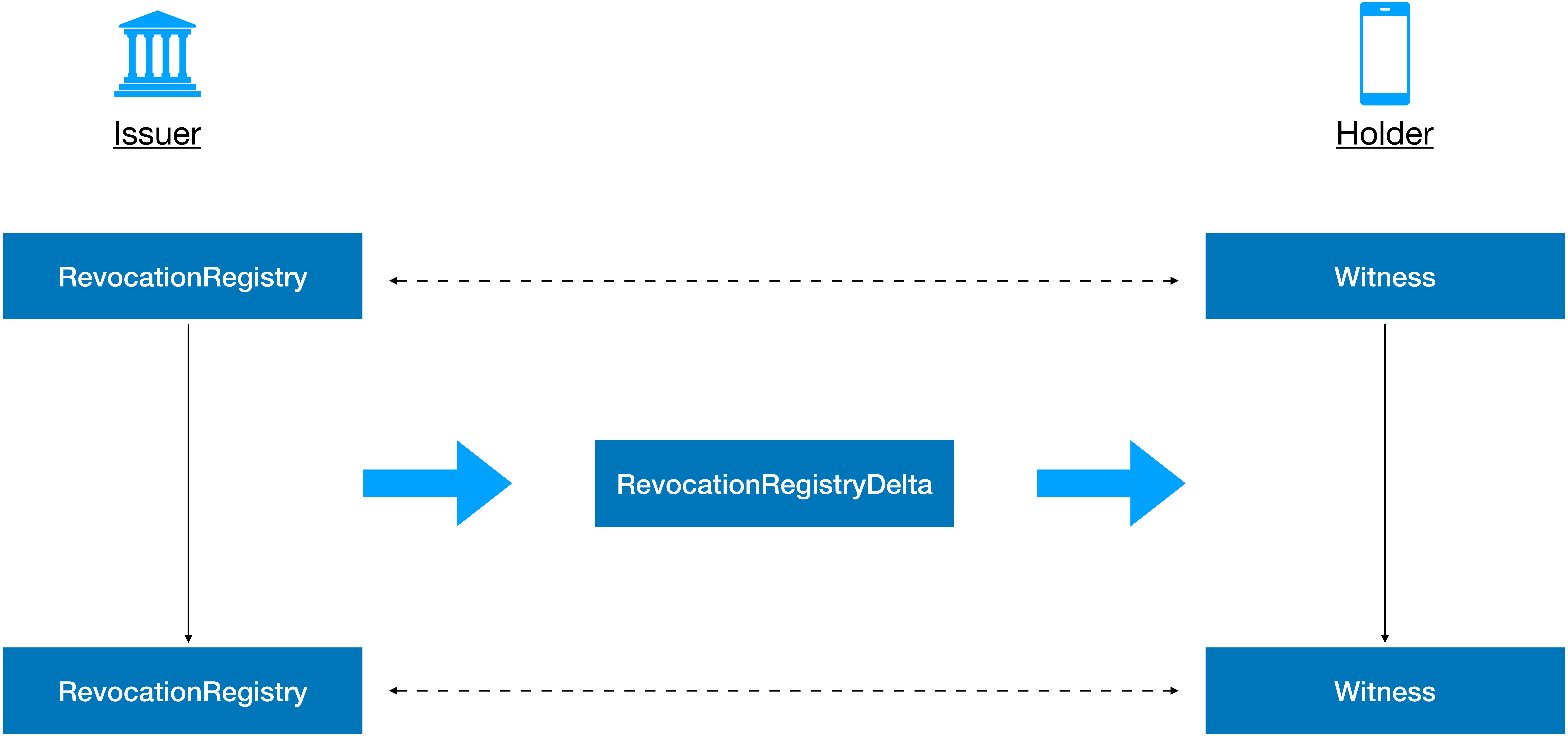
Public Update

PrimaryCredentialSignature

NonRevocationCredentialSignature

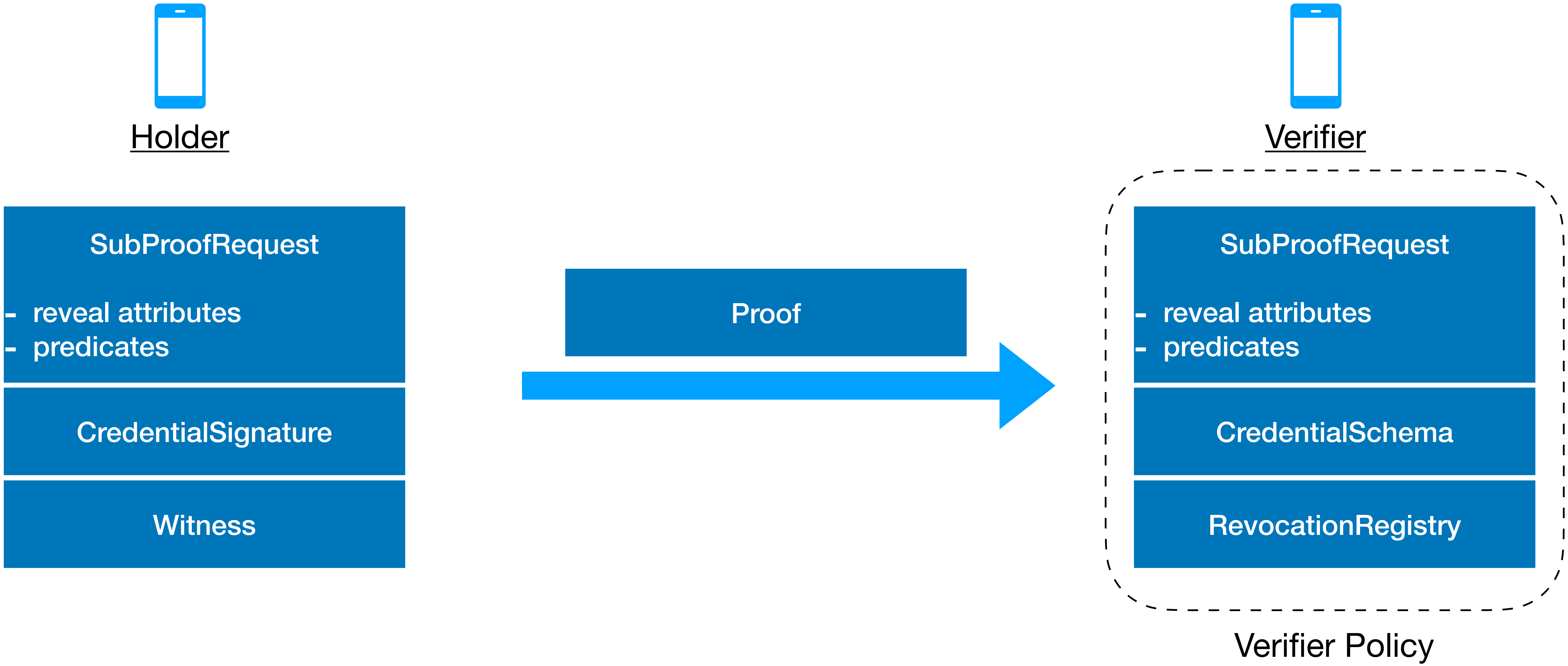
CredentialSignature

Witness Management



***Witness is the part of credential that needs to be continually updated in sync with Revocation Registry**

Proof Presentation



Holder

SubProofRequest

- reveal attributes
- predicates

CredentialSignature

Witness

Proof

Verifier

SubProofRequest

- reveal attributes
- predicates

CredentialSchema

RevocationRegistry

Verifier Policy

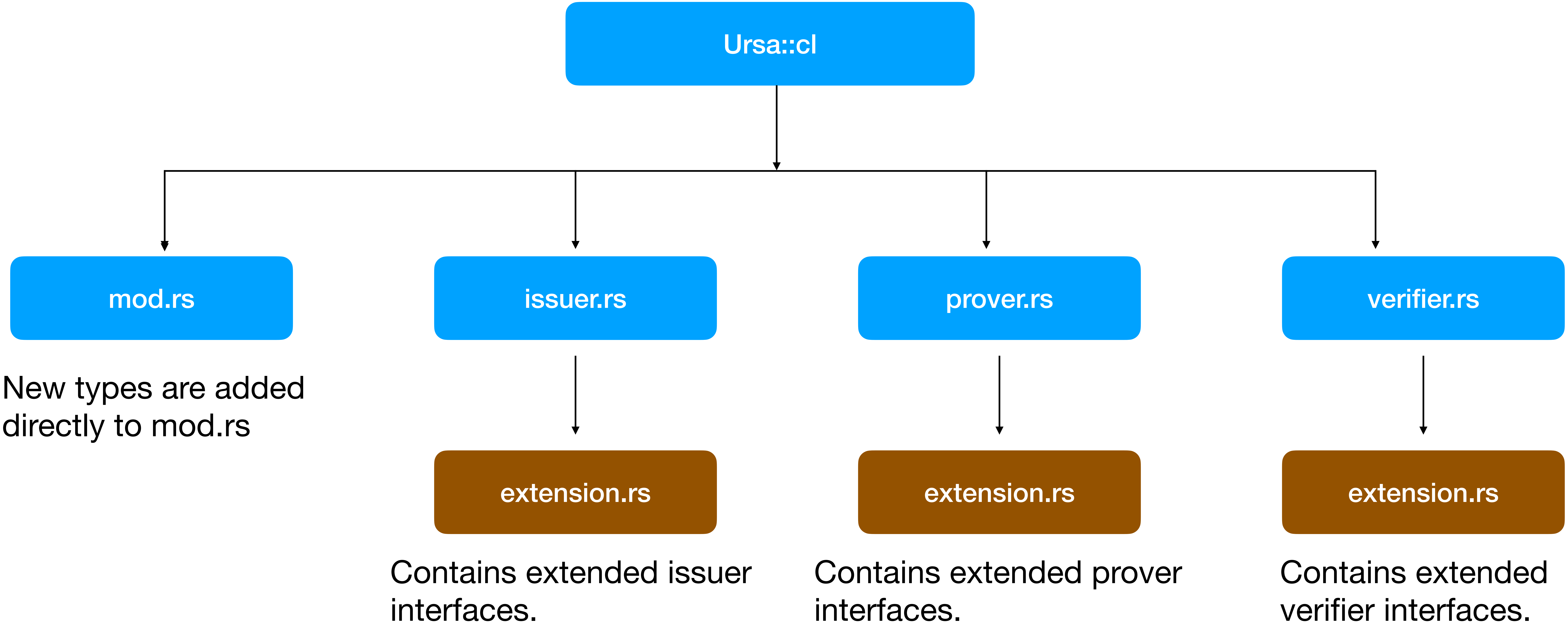
Design Goals

- Backward Compatibility (Strict): Existing artefacts should work as is using existing interfaces. No changes to existing interfaces, or changes affecting them (Zero changes to existing code).

- Forward Compatibility (Weak): It should be easy to use artefacts generated using existing interface with newer interface with minimal “wrapper” code.

- General Interface: Interfaces are revocation-scheme agnostic as much as possible.

Overview of changes to libursa CL module



Ursa::cl

mod.rs

issuer.rs

prover.rs

verifier.rs

New types are added directly to mod.rs

extension.rs

Contains extended issuer interfaces.

extension.rs

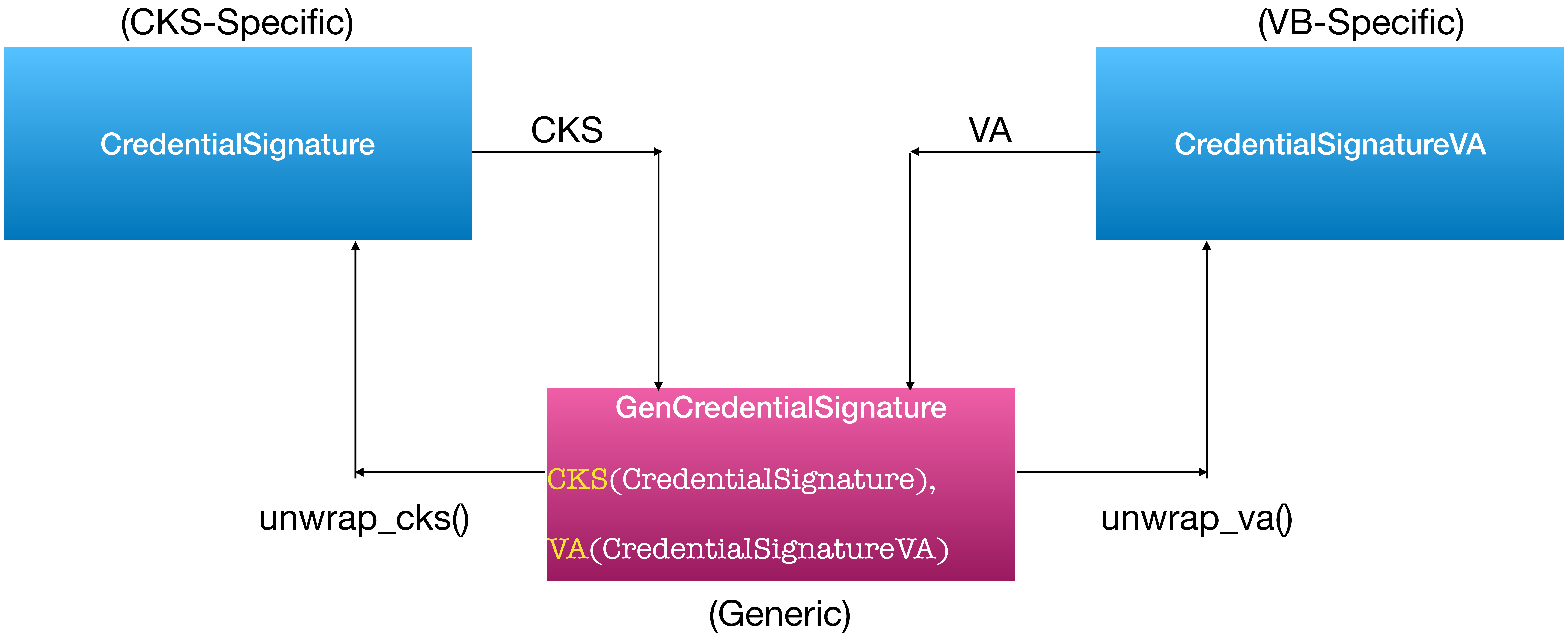
Contains extended prover interfaces.

extension.rs

Contains extended verifier interfaces.

Implementation : Generic/Wrapper Types

For each type which depends on revocation scheme, introduce a VB-specific type, and a generic type which can represent both.



Disclaimer/Apology: All the VB-specific types end with VA !!!. Mixup between first and second name of second author Alex Biryukov.

Implementation: Generic Functions - I

Define analogous functions for the new revocation scheme:

CKS

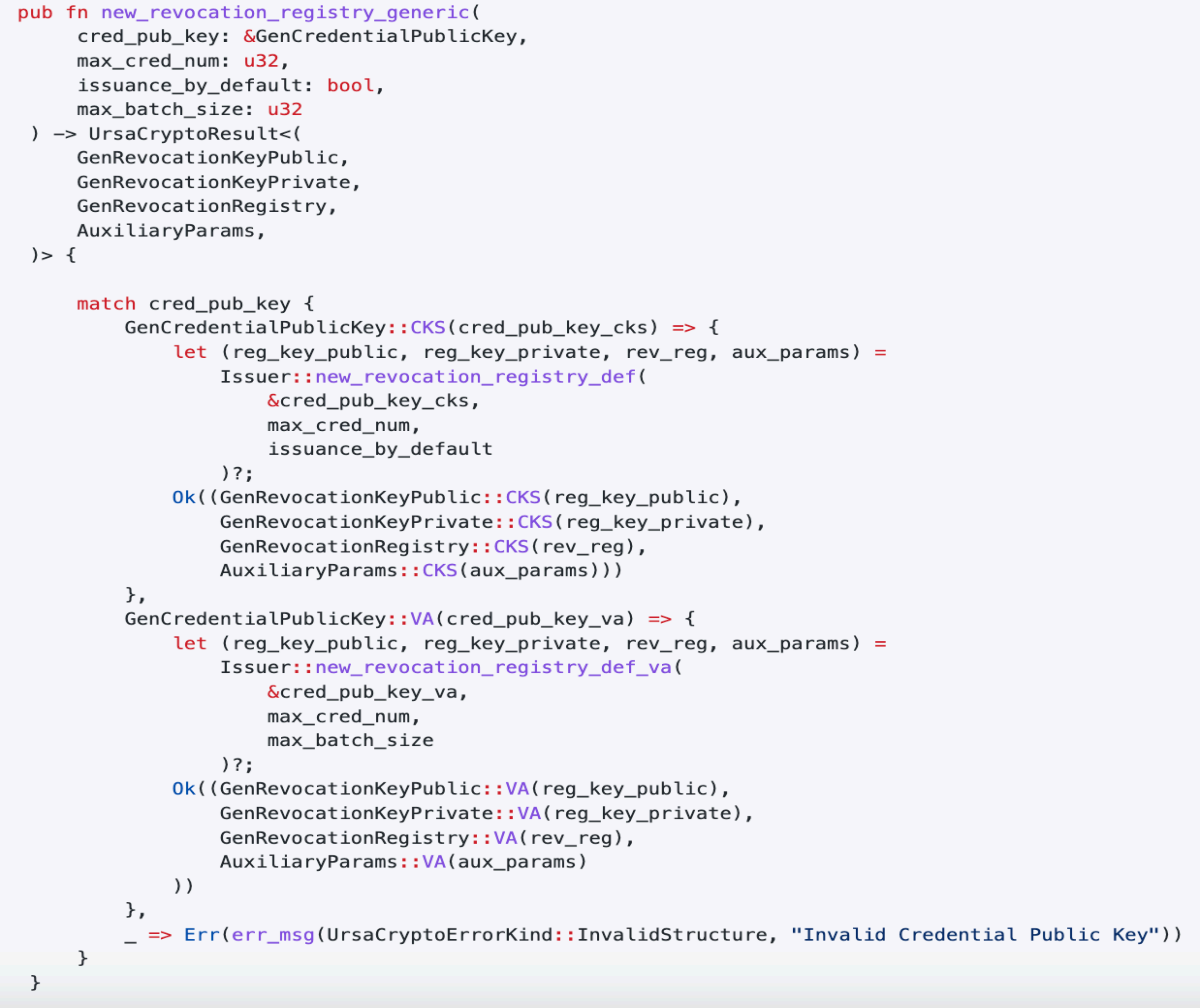
```
pub fn new_credential_def(
        credential_schema: &CredentialSchema,
        non_credential_schema: &NonCredentialSchema,
        support_revocation: bool,
    ) -> UrsaCryptoResult<(
        CredentialPublicKey,
        CredentialPrivateKey,
        CredentialKeyCorrectnessProof,
    )>
```

<-
--
>

VA

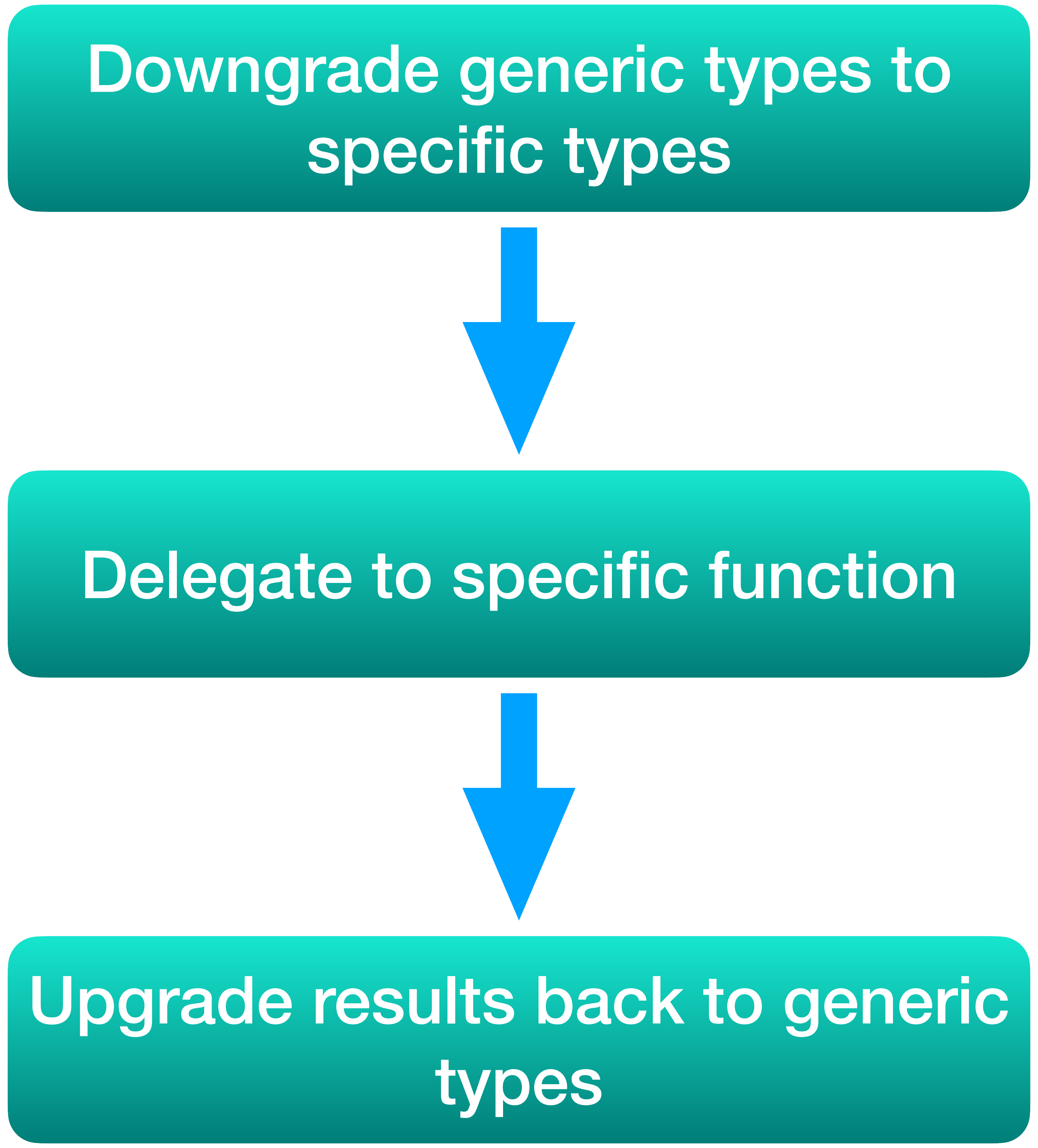
```
pub fn new_credential_def_va(
        credential_schema: &CredentialSchema,
        non_credential_schema: &NonCredentialSchema,
        support_revocation: bool
    ) -> UrsaCryptoResult<(
        CredentialPublicKeyVA,
        CredentialPrivateKeyVA,
        CredentialKeyCorrectnessProof
    )>
```

Implementation: Generic Functions - II



```
pub fn new_revocation_registry_generic(
     cred_pub_key: &GenCredentialPublicKey,
     max_cred_num: u32,
     issuance_by_default: bool,
     max_batch_size: u32
 ) -> UrsaCryptoResult<(
     GenRevocationKeyPublic,
     GenRevocationKeyPrivate,
     GenRevocationRegistry,
     AuxiliaryParams,
 )> {

     match cred_pub_key {
         GenCredentialPublicKey::CKS(cred_pub_key_cks) => {
             let (reg_key_public, reg_key_private, rev_reg, aux_params) =
                 Issuer::new_revocation_registry_def(
                     &cred_pub_key_cks,
                     max_cred_num,
                     issuance_by_default
                 )?;
             Ok((GenRevocationKeyPublic::CKS(reg_key_public),
                 GenRevocationKeyPrivate::CKS(reg_key_private),
                 GenRevocationRegistry::CKS(rev_reg),
                 AuxiliaryParams::CKS(aux_params)))
         },
         GenCredentialPublicKey::VA(cred_pub_key_va) => {
             let (reg_key_public, reg_key_private, rev_reg, aux_params) =
                 Issuer::new_revocation_registry_def_va(
                     &cred_pub_key_va,
                     max_cred_num,
                     max_batch_size
                 )?;
             Ok((GenRevocationKeyPublic::VA(reg_key_public),
                 GenRevocationKeyPrivate::VA(reg_key_private),
                 GenRevocationRegistry::VA(rev_reg),
                 AuxiliaryParams::VA(aux_params)
             ))
         },
         _ => Err(err_msg(UrsaCryptoErrorKind::InvalidStructure, "Invalid Credential Public Key"))
     }
 }
```

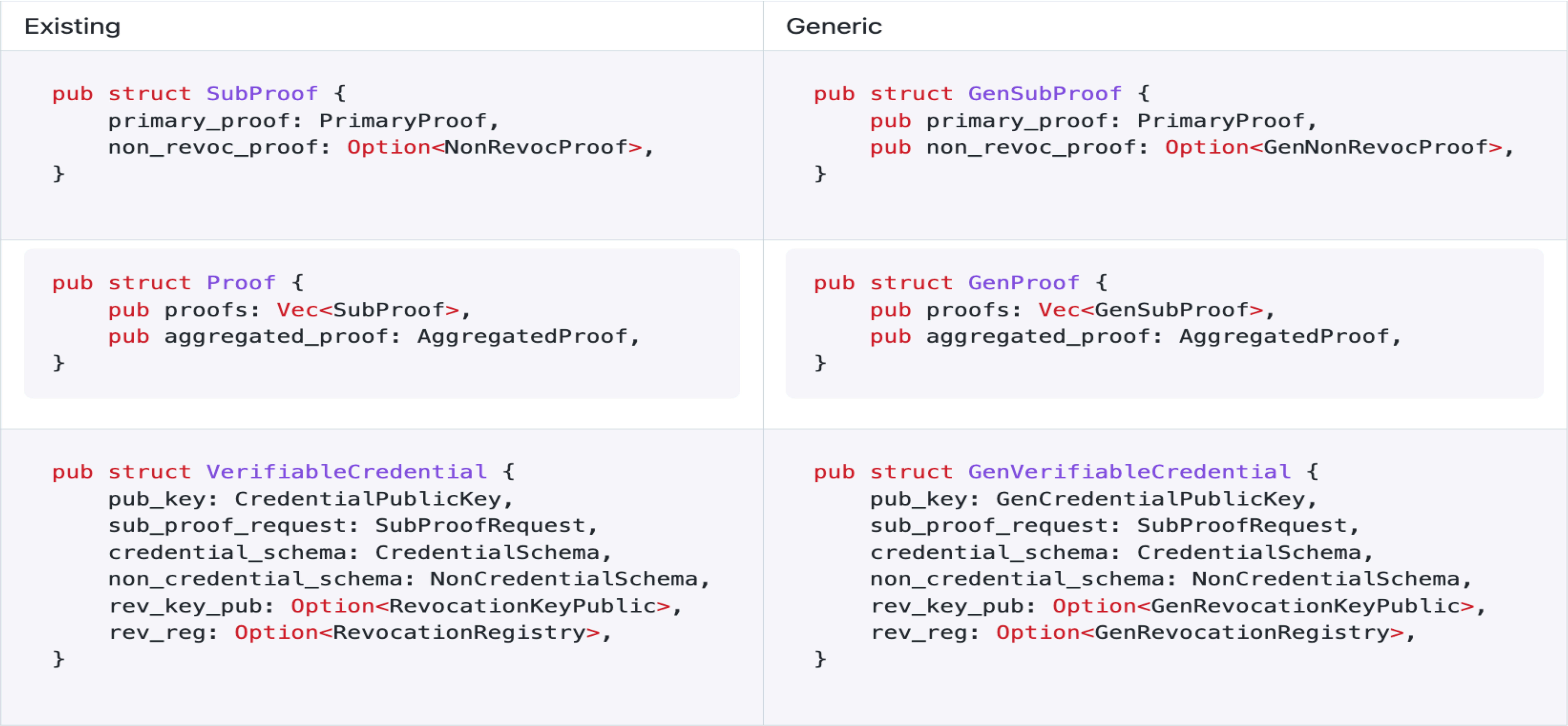


Downgrade generic types to specific types

Delegate to specific function

Upgrade results back to generic types

Extending presentation interface



Existing	Generic

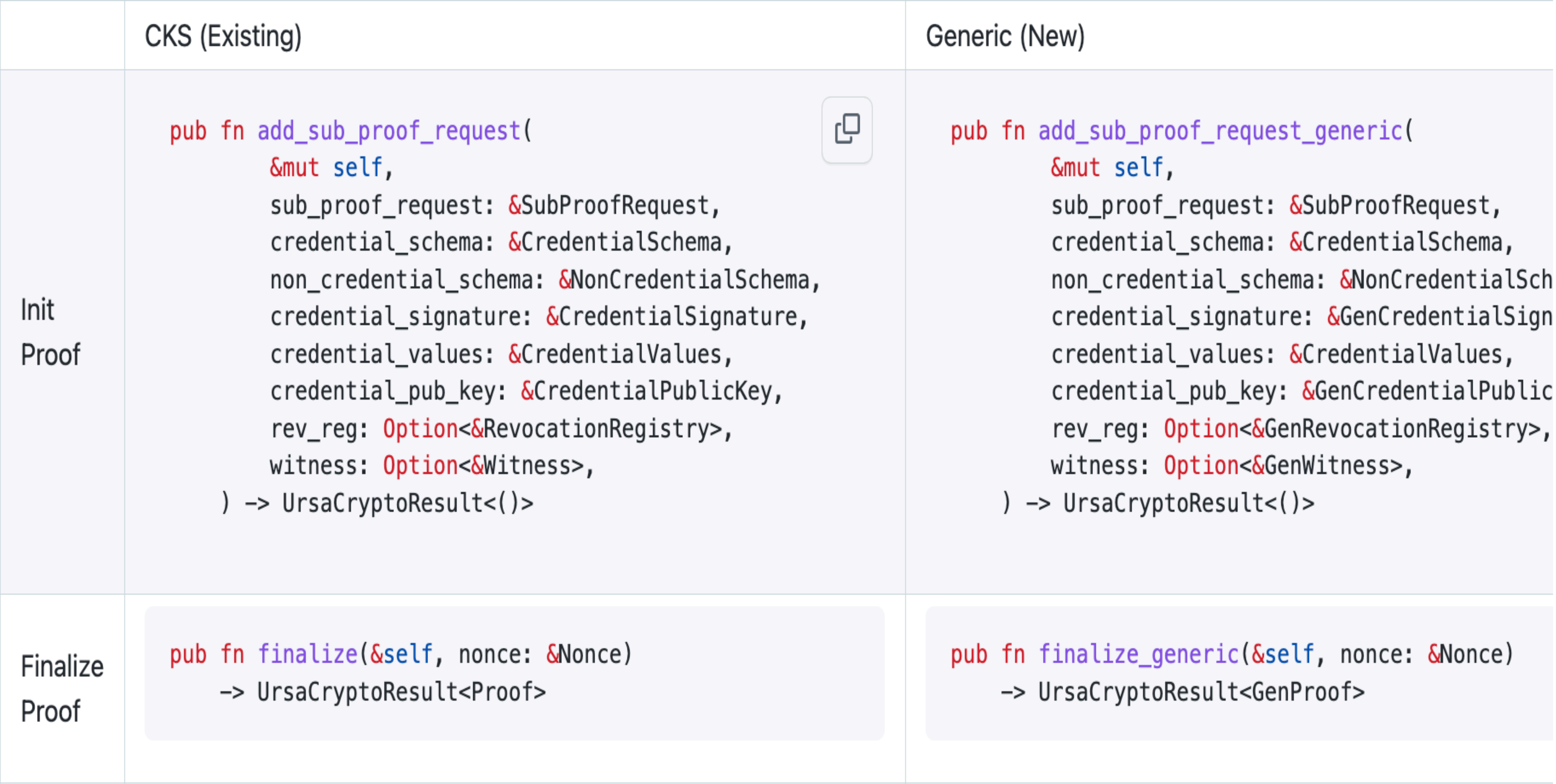
pub struct SubProof {
    primary_proof: PrimaryProof,
    non_revoc_proof: Option<NonRevocProof>,
}
	pub struct GenSubProof {
	    pub primary_proof: PrimaryProof,
	    pub non_revoc_proof: Option<GenNonRevocProof>,
	}

pub struct Proof {
    pub proofs: Vec<SubProof>,
    pub aggregated_proof: AggregatedProof,
}
	pub struct GenProof {
	    pub proofs: Vec<GenSubProof>,
	    pub aggregated_proof: AggregatedProof,
	}

pub struct VerifiableCredential {
    pub_key: CredentialPublicKey,
    sub_proof_request: SubProofRequest,
    credential_schema: CredentialSchema,
    non_credential_schema: NonCredentialSchema,
    rev_key_pub: Option<RevocationKeyPublic>,
    rev_reg: Option<RevocationRegistry>,
}
	pub struct GenVerifiableCredential {
	    pub_key: GenCredentialPublicKey,
	    sub_proof_request: SubProofRequest,
	    credential_schema: CredentialSchema,
	    non_credential_schema: NonCredentialSchema,
	    rev_key_pub: Option<GenRevocationKeyPublic>,
	    rev_reg: Option<GenRevocationRegistry>,
	}

Extend the types to allow different revocation(s)

Extending Presentation Interface



	CKS (Existing)	Generic (New)

Init
Proof
	`pub fn add_sub_proof_request(`
	`&mut self,`
	`sub_proof_request: &SubProofRequest,`
	`credential_schema: &CredentialSchema,`
	`non_credential_schema: &NonCredentialSchema,`
	`credential_signature: &CredentialSignature,`
	`credential_values: &CredentialValues,`
	`credential_pub_key: &CredentialPublicKey,`
	`rev_reg: Option<&RevocationRegistry>,`
	`witness: Option<&Witness>,`
	`) -> UrsaCryptoResult<()>`
		`pub fn add_sub_proof_request_generic(`
		`&mut self,`
		`sub_proof_request: &SubProofRequest,`
		`credential_schema: &CredentialSchema,`
		`non_credential_schema: &NonCredentialSch`
		`credential_signature: &GenCredentialSign`
		`credential_values: &CredentialValues,`
		`credential_pub_key: &GenCredentialPublic`
		`rev_reg: Option<&GenRevocationRegistry>,`
		`witness: Option<&GenWitness>,`
		`) -> UrsaCryptoResult<()>`

Finalize
Proof
	`pub fn finalize(&self, nonce: &Nonce)`
	`-> UrsaCryptoResult<Proof>`
		`pub fn finalize_generic(&self, nonce: &Nonce)`
		`-> UrsaCryptoResult<GenProof>`

Define generic interfaces with generic types

Tutorial: Complete Workflow Functions

CKS Scheme
VB Scheme
Create Schema Definition
Create Registry Definition
Issue Credentials
Initialize Witness
Initialize Witness
Prove and Verify Presentation
Revoke Credentials
Update Witness
Revoke Credentials
Update Witness
Prove and Verify Presentation

Detailed examples for complete worklfows in tutorials module @ https://github.com/nitsatiisc/ursa/blob/vb-accumulator-changes/libursa/tests/test_generic_interface.rs

Benchmarks

Issuer	CKS(s)	VA(s)
Gen Registry (100K)	126.2	52.7
Issue Credential	0.11	0.13
Issue Update(100)	0.002	0.02

Holder	CKS(ms)	VA(ms)
Init Witness(100K)	250	0
Update Witness(100)	0.0	1
Proof	53	37

Verifier	CKS(ms)	VA(ms)
Verify	49	26

Tests and benchmarks @: https://github.com/nitsatiisc/ursa/blob/vb-accumulator-changes/libursa/tests/test_generic_interface.rs

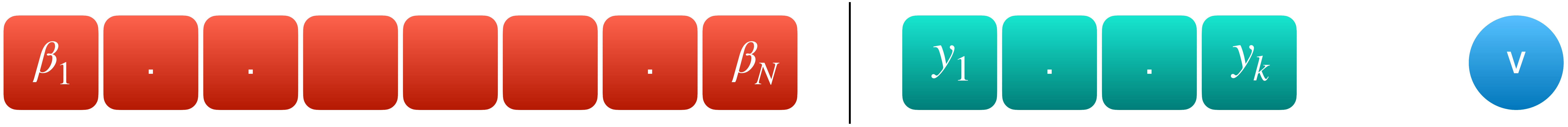
Custom Modifications to VB protocol

- Improved proof of knowledge of the non-revocation witness based on improved PoK for BBS+ signature in [2]. Reduces pairing checks for the verifier from ~10 to 1. Results in about 45% faster proving and verification on average.

- Maintain polynomials as evaluations: $f(x) \equiv \langle f(1), \ldots, f(d) \rangle$ and changes to protocol to work with this representation.

Questions?

Vitto-Biryukov Accumulator



β_1 . . . β_N | y_1 . . y_k

V

α

Registry Private State: $\alpha, Y_0 = \{\beta_1, \ldots, \beta_N\}$

Accumulated Set: $Y_V = \{y_1, \ldots, y_k\}$

Accumulator Value: V

$$V = \left(\prod_{z \in Y_0 \cup Y_V} (\alpha + z) \right) \cdot P$$

*** The accumulated set corresponds to revoked elements**

Non Membership Witness

For y not in the accumulated set, the Issuer issues a non-membership witness $w_y = (C, d)$ as:

$$C = \frac{f_V(\alpha) - f_V(-y)}{y + \alpha} \cdot P, \quad d = f_V(-y)$$

where $$f_V(x) = \prod_{z \in Y_0 \cup Y_V} (x + z)$$

We note that d is non-zero when y is not in the set. Also note that $V = f_V(\alpha) \, . \, P$

Batch Update

$$\begin{tikzcd}(Y_V, V) \arrow[r, "Y = \{y_1, \ldots, y_n\}"] & (Y_{V'}, V')\end{tikzcd}$$

$$\begin{tikzcd}w_y = (C, d) \arrow[r, "?"] & w'_y = (C', d')\end{tikzcd}$$

$$d_A(x) = \prod_{i=1}^{n} (x - y_i)$$

$$v_A(x) = \sum_{s=1}^{n} \left(\prod_{i=1}^{s-1} (y_i + \alpha) \prod_{j=s+1}^{n} (y_i - x) \right)$$

$$d' = d_A(y) \cdot d$$
$$C' = d_A(y) \cdot C + v_A(y) \cdot V$$

Batch Update -II

The issuer cannot publish polynomial v_A(x) in plain-text, as it can leak α. Thus as part of batch update protocol, issuer publishes:

$$\Omega = (c_0 \cdot V, c_1 \cdot V, \ldots, c_n \cdot V)$$

Where: $$v_A(x) = c_0 + c_1 x + \cdots + c_n x^n$$

From the above update, the holder can locally compute v_A(y).V as the following scalar product:

$$v_A(y) \cdot V = \langle (1, y, y^2, \ldots, y^n), \Omega \rangle$$

Modified Batch Update

The issuer cannot publish polynomial v_A(x) in plain-text, as it can leak α. Thus as part of batch update protocol, issuer publishes:

$$\Omega = (v_A(\omega_0) \cdot V, v_A(\omega_1) \cdot V, \ldots, v_A(\omega_n) \cdot V)$$

Where: $\omega_0, \ldots, \omega_n$ are fixed.

From the above update, the holder can locally compute v_A(y).V as the following scalar product:

$$v_A(y) \cdot V = \langle (\ell_0, \ldots, \ell_n), \Omega \rangle$$

In the above $\ell_0, \ldots, \ell_n$ are Lagrangian coefficients such that:

$$v_A(y) = \sum_{i=0}^{n} \ell_i v_A(\omega_i)$$