BBS FOR VERIFIABLE CREDENTIALS - DATA INTEGRITY

Dr. Greg Bernstein

June 4th, 2024

BBS SIGNATURE WITH VC DATA INTEGRITY

OUTLINE

1. VC Data Integrity Review
2. BBS Signature Review
3. BBS Messages from JSON-LD VC
4. Mandatory and Selective Disclosure with JSON
5. Putting all the pieces together: Base and Derived Proofs

ME AND MY VC WORK

* W3C “Invited Expert”, semi-retired, website: Grotto Networking

- Helping (co-editor) with: VC-DI, VC-DI-EdDSA, VC-DI-ECDSA, VC-DI-BBS specs
- Cryptographic test vector generation for the specs EdDSA and ECDSA and ECDSA-SD and BBS (open source code)
- Open Source implementations: VC-DI-ECDSA-SD, BBS Scheme (IETF), VC-DI-BBS, VC-Server (tiny implementation for interoperability testing)
- Helping out with IETF BBS specs: BBS, Blind BBS, BBS per Verifier Id (pseudonym).

VC DATA INTEGRITY REVIEW

KEY SPECIFICATIONS

1. Verifiable Credentials Data Model v2.0 In particular JSON-LD based with information specified in section Verifiable Credentials. We are concerned here with **embedded proof** securing mechanism as compared to an *enveloping proof* mechanism.
2. Verifiable Credential Data Integrity 1.0 This specifies the **proof** field that gets embedded to secure a credential. Note that the embedded approach combined with DI allows for “parallel signatures”, proof sets, and proof chains

DATA MODEL VC GENERAL FORM

Data Model Verifiable Credentials, aka *unsecured document*

```
{
  "@context": [ "https://www.w3.org/ns/cred
  "type": ["VerifiableCredential", "more...
  "issuer": "URL or object", // Required
  "credentialSubject": {
    // Bulk of info here. Required!
  }
```

DATA INTEGRITY PROOF GENERAL FORM

From Proofs and DataIntegrityProof,

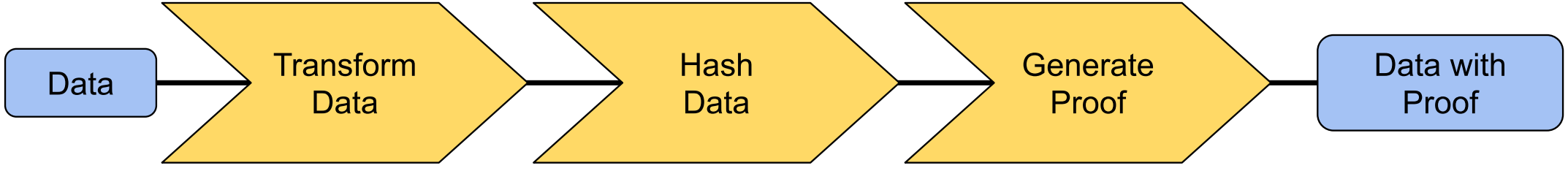
```
{
  "type": "DataIntegrityProof", // Required
  "proofPurpose": "assertionMethod", // Req
  "verificationMethod": "string/URL links t
  "proofValue": "string encoding binary dat
  "cryptosuite": "bbs-2023", // Required fo
```

DI (EMBEDDED) SECURED DOCUMENT FORMAT

Only Required Properties Shown

```
{
  "@context": [ "https://www.w3.org/ns/cred
  "id": "a URL", // For the VC itself, Opti
  "type": ["VerifiableCredential", "more...
  "issuer": "URL or object", // Required
  "credentialSubject": {
    // Bulk of info here. Required
```

GENERAL APPROACH HIGH LEVEL



Transform, Hash, Sign

GENERAL APPROACH I: EDDSA AND ECDSA

1. Starting with *proof options* which includes anything in the proof but the *proofValue*. This object is appropriately canonized (JCS or RDFC) and then hashed to produce *proofConfigHash*. **Protects proof meta data**
2. The *unsecured document* gets canonized (JCS or RDFC) and then hashed to produce *transformedDocumentHash*. **Protects document data and meta data**

GENERAL APPROACH II: EDDSA AND ECDSA

3. The two hashes above are concatenated then a signature value, *proofBytes*, is computed with the appropriate algorithm (ECDSA-P256, ECDSA-P384, EdDSA).
4. *proofBytes* is then (multibase) encoded to produce the *proofValue* field for inclusion in the *proof* field.

WHY CANONICALIZATION?

Example: JSON Canonicalization Scheme (JCS)

1. Adding/removing whitespace between tokens in a JSON document does not change its meaning
2. Changing the order of properties in a JSON *object* does not change its meaning
3. Either of the above **will** change the value of a cryptographic hash over the text of the JSON
4. Need equivalent documents to produce same hash! Solution ==> put in a canonical (standard) form.

EXAMPLE FOR CANONICALIZATION

```
{
  "@context": "https://json-ld.org/contexts
  "name": "Manu Sporny",
  "homepage": "http://manu.sporny.org/",
  "image": "http://manu.sporny.org/images/m
}
```

EXAMPLE JCS CANONICALIZATION

Note ordering of fields, removal of white space.

```
{"@context":"https://json-ld.org/contexts/p
```

BBS SIGNATURE SCHEME REVIEW

BACKGROUND READING AND DEMO

- BBS for Verifiable Credentials - Basics, May 2023.
- The BBS Signature Scheme (DIF/IETF draft)
- BBS in Browser Demo

BBS FUNDAMENTAL PROPERTIES

From DIF/IETF draft

- **Fixed Sized Signatures**: The scheme allows a signer to sign multiple messages and produce a single -constant size- output signature, i.e., 80 bytes.
- **Selective Disclosure**: The receiver of the signature can generate a *BBS proof* that discloses only a subset of the original set of messages.
- **Unlinkable Proofs**: The *BBS proofs* are unlinkable to the original signature, and to each other.

EXAMPLE: TREE DRIVERS LICENSE

From Grotto BBS demo, not a VC or mDL…


```
{
  "publicKey": "b65b7cbff4e81b723456a13936b
  "header": "11223344556677889900aabbccddee
  "messages": [
    "FirstName: Sequoia",
    "LastName: Sempervirens",
    "Address: Jedediah Smith Redwoods State
```

SELECTIVE DISCLOSURE EXAMPLE: A TREE GOES TO A BAR…

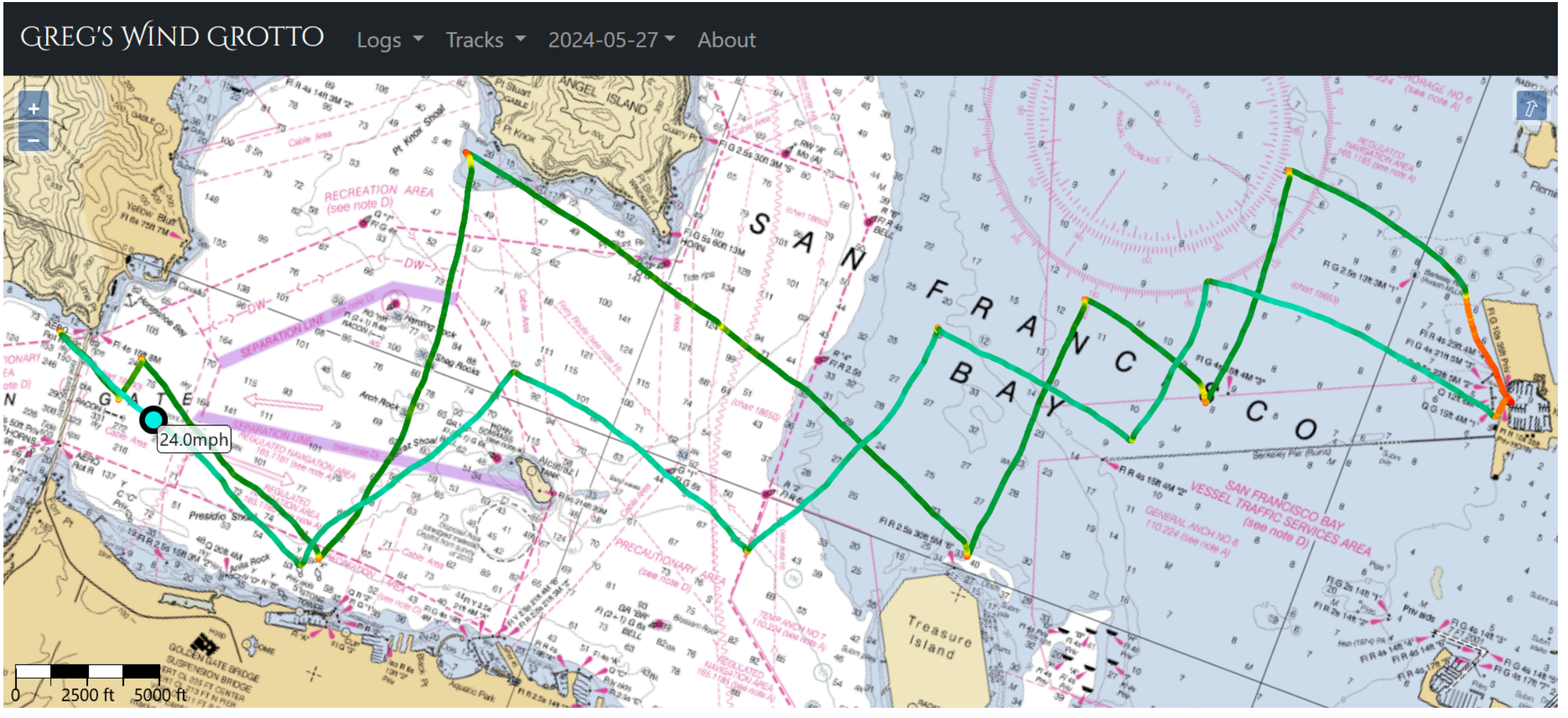
Messages (select to include):

- [x] FirstName: Sequoia
- [ ] LastName: Sempervirens
- [ ] Address: Jedediah Smith Redwoods State Park, California
- [x] Date of Birth: 1200/03/21
- [ ] Height: 296 feet
- [ ] Eyes: None
- [ ] Hair: Brown bark, green needles
- [x] Picture: Encoded photo
- [ ] License Class: None, Trees can't drive

EXAMPLE DERIVED “PROOF”


```
{
  "pk": "b65b7cbff4e81b723456a13936b6bcc77a
  "header": "11223344556677889900aabbccddee
  "ph": "",
  "disclosedIndexes": [
    0,
    2
```

GOOD TRACKING I

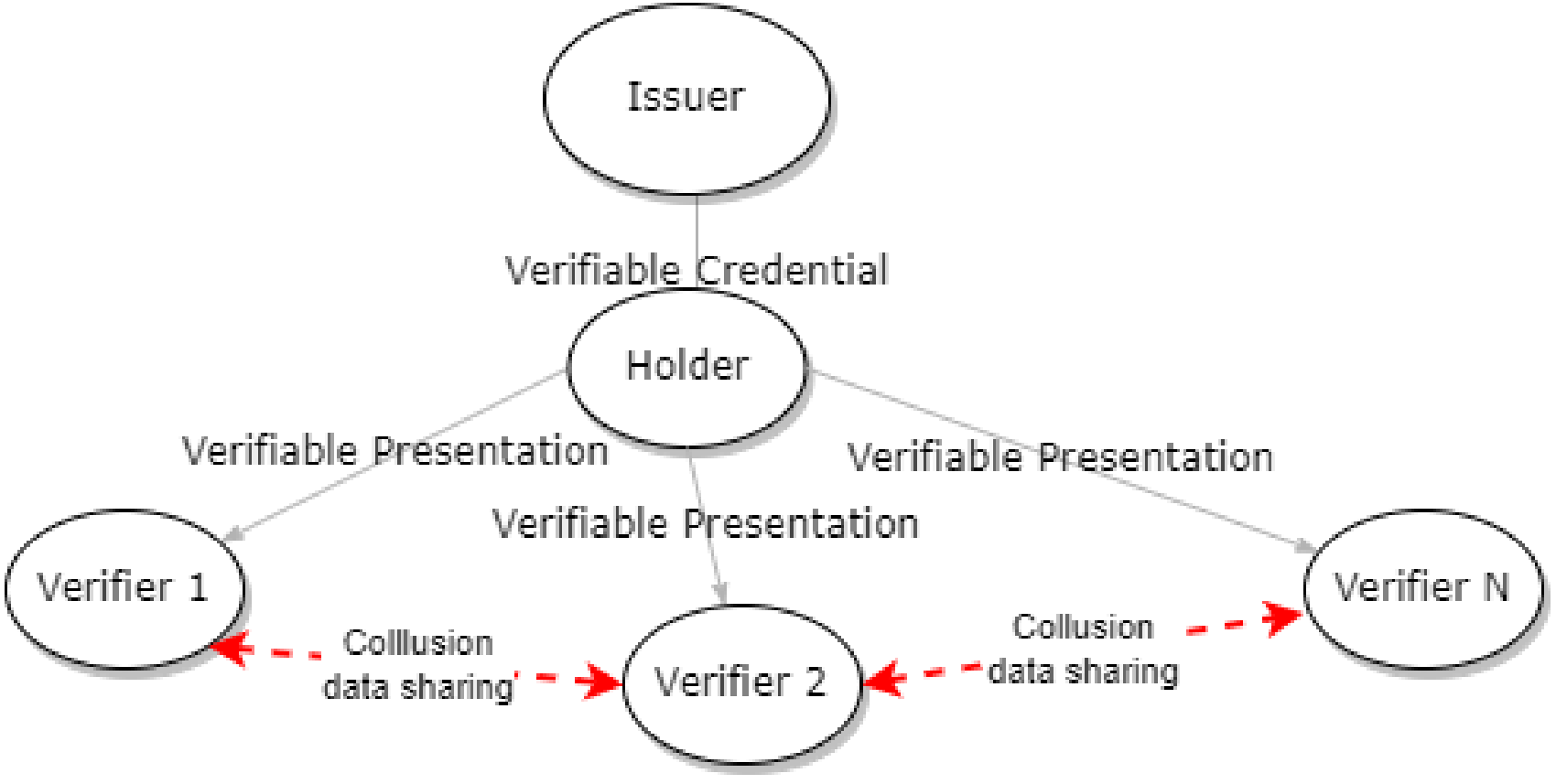


Tracking for Safety

GOOD TRACKING II

VERIFIERS TRACKING HOLDERS

Not the good kind of tracking…



UNLINKABLE PROOFS EXAMPLE 1

- A tree goes into a bar and needs to prove it lives in a local state park in order to get a very large glass of water.
- It then goes to another local bar for another very large glass of water.
- It doesn’t want to be tracked across bars or have its water consuming habits tracked.
- Solution: Generate a separate **BBS proof** for each bar it visits

UNLINKABLE PROOFS EXAMPLE 2

BBS Proof presented at first bar


```
{
  "pk": "b65b7cbff4e81b723456a13936b6bcc77a
  "header": "112233445566778899 00aabbccddee
  "ph": "",
  "disclosedIndexes": [
    2
  ],
```

UNLINKABLE PROOFS EXAMPLE 3

BBS Proof presented at second bar


```
{
  "pk": "b65b7cbff4e81b723456a13936b6bcc77a
  "header": "11223344556677889900aabbccddee
  "ph": "",
  "disclosedIndexes": [
    2
  ]
```

UNLINKABLE PROOFS

- The values (cryptographic byte array) contained in the *BBS proofs* are unlinkable. In particular they appear essentially random
- Unlinkable proofs do not prevent correlation on disclosed messages! This information has been disclosed by the holder.
- See Selective Disclosure and Unlinkability for a full discussion of *unlinkability* and its limits.

BBS API AND MESSAGES

TERMINOLOGY COLLISION!

Important: We have a terminology collision! Both BBS signatures and verifiable credentials use the terms *proof* and *signature*. However these can have very different meanings, hence we will be verbose and use terms like “BBS signature”, “BBS proof”, and “VC data integrity proof”.

BBS SIGNATURE SCHEME API 1

BBS has *sign* and *verify*

1. `signature = Sign(Secret_Key, Public_key, header, messages)` used by *issuer*/(*signer*)
2. `result = Verify(Public_key, signature, header, messages)` is used when the *holder*/(*prover*) verifies the VC it receives from the *issuer*/(*signer*).

BBS SIGNATURE SCHEME APIS 2

but BBS is more than *sign* and *verify*

3. `bbs_proof = ProofGen(Public_key, signature, header, ph, messages, disclosed_indexes)` is used when the *holder* want to prepare a derived VC that selectively discloses original VC information.
4. `result = ProofVerify(Public_key, bbs_proof, header, ph, disclosed_messages, disclosed_indexes)` is used by the *verifier* to validate the derived VC against the original issuers public key.

BBS MESSAGES FROM JSON-LD?

From VC BBS Test Vector

```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    {
      "@vocab": "https://windsurf.grotto-ne
    }
  ],
```

STATEMENTS/MESSAGES FROM JSON-LD

- JSON-LD is a serialization, storage and exchange format for the Resource Description Framework (RDF)
- RDF is a graph based data model consisting of: *Subject*, *Predicate*, *Object*, and if needed containing *Graph*.
- RDF Canonicalization transforms a JSON-LD document into an ordered set of statements called *quads*.
- We use these statements/quads as our BBS messages!

EXAMPLE STATEMENTS/MESSAGES

From VC BBS Test Vector


```
[
  "_:c14n0 <https://windsurf.grotto-network
  "_:c14n0 <https://windsurf.grotto-network
  "_:c14n0 <https://windsurf.grotto-network
  "_:c14n1 <https://windsurf.grotto-network
  "_:c14n1 <https://windsurf.grotto-network
  "_:c14n1 <https://windsurf.grotto-network
```

MANDATORY AND SELECTIVELY DISCLOSED STATEMENTS

SELECTIVE DISCLOSURE REQUIREMENTS

- An *issuer* can specify that a subset of the statements must be revealed by the *holder* to the *verifier*. These are the **mandatory** statements.
- A *holder* can specify a subset of the non-mandatory statements to be revealed. These are the **selectively disclosed** statements.
- Want to do this in a user and developer friendly way.

JSON POINTERS

JavaScript Object Notation (JSON) Pointer (RFC6901)

- “JSON Pointer defines a string syntax for identifying a specific value within a JavaScript Object Notation (JSON) document.”
- Example: `"/issuer"` is used to match the *issuer* field.

EXAMPLE: MANDATORY POINTERS

Prior to a fictional windsurfing race: declare two sails, most recent board

```
[ "/issuer",
  "/credentialSubject/sailNumber", // how t
  "/credentialSubject/sails/1",
  "/credentialSubject/boards/0/year",
  "/credentialSubject/sails/2"
]
```

EXAMPLE: MATCHING FIELDS

```
[
  {
    "pointer": "/issuer",
    "value": "https://vc.example/windsurf/r
  },
  {
    "pointer": "/credentialSubject/sailNumb
```

EXAMPLE: MANDATORY STATEMENTS (NQUADS)

Behind the scenes!

```
[
  [
    0,
    "_:b0 <http://www.w3.org/1999/02/22-rdf
  ],
  [
    1
```

DATA INTEGRITY BASE BBS PROOF (ISSUER => HOLDER)

ISSUE A BBS PROTECTED VC

Inputs:

- *proof options*: Required and any optional fields for use in the attached *proof*
- *Unsecured document*: The credential without proof
- *Key material*: For use by the signing algorithm
- **Mandatory Pointers**: Array (possibly empty) of JSON pointers to mandatory reveal information.

DETAILS: SUMMARIZE AND PROTECT PROOF OPTIONS

1. Canonicalize *proof options* using RDFC
2. Hash canonized *proof* options ==> *proofHash*

DETAILS: SUMMARIZE AND PROTECT MANDATORY STATEMENTS

- Inputs: *mandatory pointers* and *unsecured credential*
- Run “Canonicalize and Group” function to produce a list of *mandatory quads* and a list of *non-mandatory quads*.
- Hash list of mandatory quads ==> *mandatoryHash*

DETAILS: COMPUTE BBS SIGNATURE BYTES

```
Sign(Secret_Key, Public_key, header, messages)
```

- *header* = concatenation of *proofHash* and *mandatoryHash*
- *messages* = *non-mandatory quads*
- Note: BBS **header** is “associated data” that must be conveyed to *holder* and *verifier*

DETAILS: PACKAGE UP INFO INTO *PROOFVALUE*

- Add a proof header (CBOR tag) of bytes 0xd9, 0x5d, and 0x02 to indicate base BBS proof
- CBOR encode the components: BBS signature, BBS header, publicKey, hmacKey, and mandatoryPointers
- Multibase encode the above to produce the *proofValue* string

DETAILS: ATTACH PROOF

- Add *proofValue* string to proof options
- Add proof options as *proof* field of unsecured document to produce the secured document.

CREATE BASE PROOF EXAMPLE


```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    {
      "@vocab": "https://windsurf.grotto-ne
    }
  ]
```

DATA INTEGRITY DERIVED BBS PROOF (HOLDER => VERIFIER)

CREATE A SELECTIVELY DISCLOSED BBS VC

Inputs

- *Secured Document* containing a base BBS proof.
- **Selective Pointers**: JSON pointers to fields that *holder* want to disclose.

Output: Derived Data Integrity Secured Document

DETAILS: COMPUTE BBS PROOF

`ProofGen(Public_key, signature, header, ph, messages, disclosed_indexes)` where

- *signature* is the original BBS signature from issuer. **Not** sent to *verifier*!
- *messages* are the *non-mandatory quads*
- The *disclosed_indexes* are based on the *selective pointers* relative to the non-mandatory quads

DETAILS: CREATE DISCLOSURE DOCUMENT

- Recover the *unsecured* issued document from the secured document by removing the *proof* field.
- Create the unsecured *disclosed document* based on *mandatory* and *selective* pointers applied to the *unsecured* issued document

DETAILS: CREATE DERIVED *PROOFVALUE*

- Add a proof header (CBOR tag) of bytes 0xd9, 0x5d, and 0x03.
- CBOR encode the components: bbsProof, compressedLabelMap, mandatoryIndexes, selectiveIndexes, and presentationHeader.
- Multibase encode the above to produce the *proofValue* string

DETAILS: ATTACH DERIVED PROOF

- *proof options* is obtained from the *proof* field of the secured document with the *proofValue* removed.
- Add the new *proofValue* string to proof options
- Add proof options as *proof* field of unsecured document to produce the secured derived document.

CREATE DERIVED: EXAMPLE

```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    {
      "@vocab": "https://windsurf.grotto-ne
    }
  ]
```