BBS Core, Pseudonyms & Blind BBS

IETF Meeting 126 | 24.07.2026

Greg Bernstein Vasilis Kalos Anja Lehmann Mike Lodder
Tobias Looker Emil Lundberg Andrew Whitehead

BBS Draft Suite | Agenda

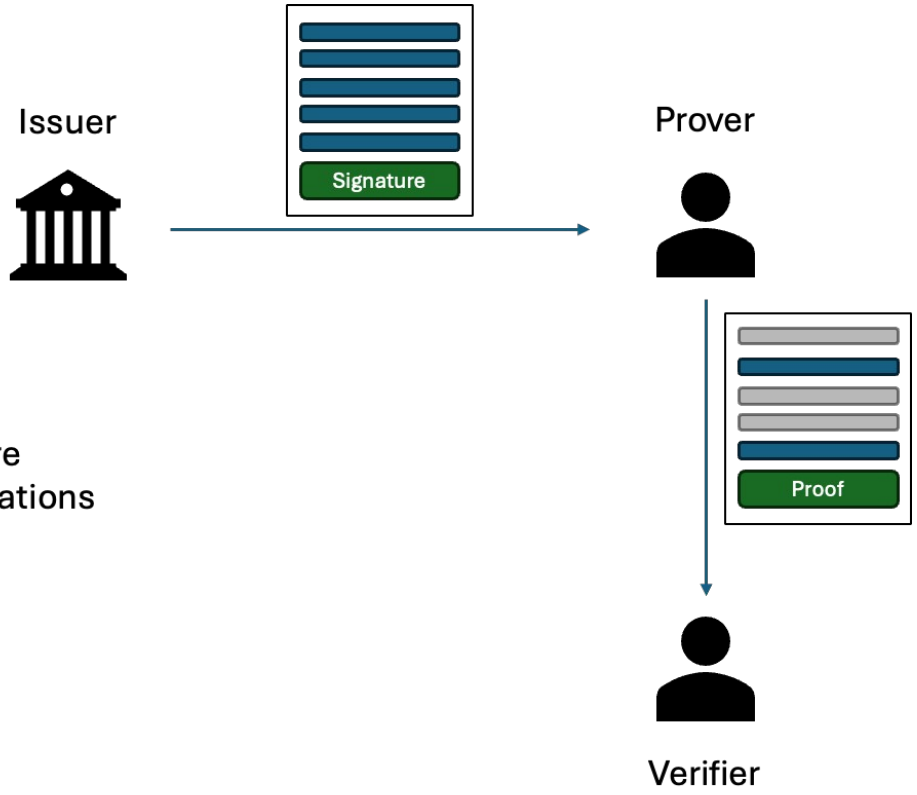
Current Status & Updates (Vasilis)

- Core BBS, BBS per Verifier Linkability, BBS Blind Issuance

New Additions to Blind BBS

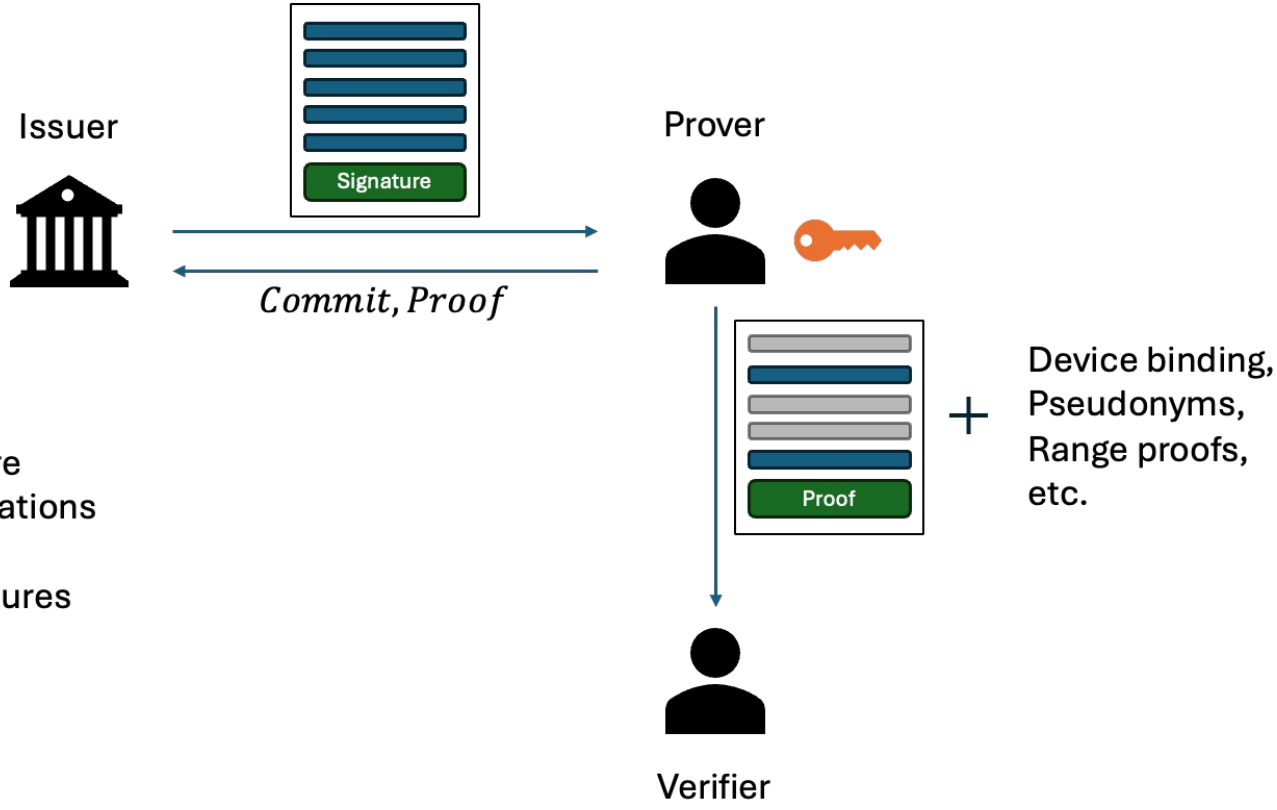
- Native Device Binding (Emil)
- Committed Disclosure (Anja)

BBS Core



- Selective Disclosure
- Unlinkable presentations

BBS Extensions



- Selective Disclosure
- Unlinkable presentations
- Blind signatures
- Committed disclosures

BBS Core - Updates

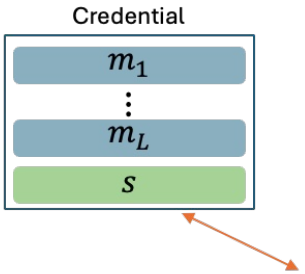
- Small re-organization, to minimize redundant checks: #PR340

- Object notation for intermediate values: #PR338

- Next steps??

BBS Extensions - Pseudonyms

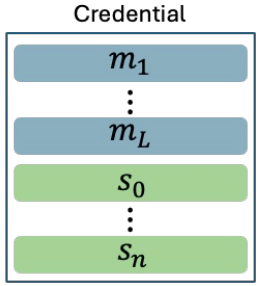
Option A



$$nym = H2C(context)^s$$

- ✓ Efficient
- Not CRPQC private

Option B



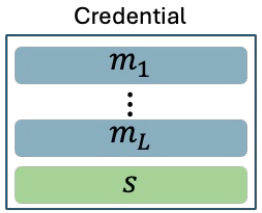
$$z = H(context)$$

$$s = s_0 + s_1 z + s_2 z^2 + \cdots + s_n z^n$$

$$nym = P^s$$

- ✓ CRPQC private up to $n + 1$ uses
- ✓ Efficient (for the Prover)
- Proof size linear to n

Option C



$$nym = H2C(context, cnt)^s$$

$$+ ZKP\{0 \leq cnt \leq m\}$$

- ✓ CRPQC private
- ✓ Small proof size overhead
- Targeted for rate limiting

BBS Draft Suite | Agenda

Current Status & Updates (Vasilis)

- Core BBS, BBS per Verifier Linkability, BBS Blind Issuance

New Additions to Blind BBS

- Native Device Binding (Emil)
- Committed Disclosure (Anja)

Blind BBS & Native Key Binding | Overview

- “Device Bound Anonymous Credentials With(out) Trusted Hardware” (DBAC): https://eprint.iacr.org/2025/1995
- Bind public keys as blind *commitments* to private key scalars
- Each presentation then requires fresh PoK of each private key

- However - hardware-bound private keys cannot use same challenge as the other message proofs in the BBS proof
  - BBS challenge depends on all random nonces for all messages; hardware signer cannot reveal its nonce
- Instead: prove a *blinding relationship* and separately prove possession of blinded key

```
dpk, dsk = KeyGen(H0)
rk       = random()
Add and prove rkey as hidden message in BBS NIZK proof
dpkbar   = dpk + rk * H0
(s, c)   = Schnorr.Sign(dsk, challenge)
sig      = (s + rk * c, c)
Append (dpkbar, sig) to proof
```

Blind BBS & Native Key Binding | Summary of Changes

- New generators: `create_generators(count, "KEYBIND_" || api_id)`

- Split `Commit` into init - prove - finalize (keep `Commit` alias for 0 keys case)
- Add serialization format for `Commit` state
- Add public keys to `commitment_with_proof` serialization format
- Verify key proofs in `CoreCommitVerify`
- Override `CoreVerify` - `BBS.CoreVerify` requires direct knowledge of all message scalars
- Add public keys to `VerifyBlindSign` proof verification

- Split `BlindProofGen` into init - prove - finalize (keep `BlindProofGen` alias for 0 keys case)
- Add public keys to pairing proof, add key blinding proof, adjust proof commitment
- Add blinded public keys and signatures to proof serialization format
- Verify key binding proofs and signatures in `BlindCoreProofVerify`

- `BBS.ProofInit`, `BBS.ProofVerifyInit`, `BBS.ProofFinalize` still used as

Blind BBS & Native Key Binding | Remarks

- 0..1 dpk or 0..N dpk?
  - DBAC paper: exactly 1 dpk
  - 0..N easy to implement, but changes the model
  - If BLS: 1 dpk = N dpk

- Schnorr/BLS signatures *must be malleable*
  - RFC 8235 (Schnorr Non-interactive Zero-Knowledge Proof) is non-malleable
- Needs generator agility in hardware signers
  - Alt. hardcode G1 => only 1 dpk
- Signature hash function?
  - SHA-256 w/ rejection sampling?
  - BBS.hash_to_scalar?
  - hash_to_field with other DST?
- Additive or subtractive Schnorr?
  - `s = r + dsk * c` (DBAC paper) vs `s = r - dsk * c` (RFC 8235)
  - Can translate by negating public key
- These all affect Schnorr/BLS signature interop with other protocols

BBS Draft Suite | Agenda

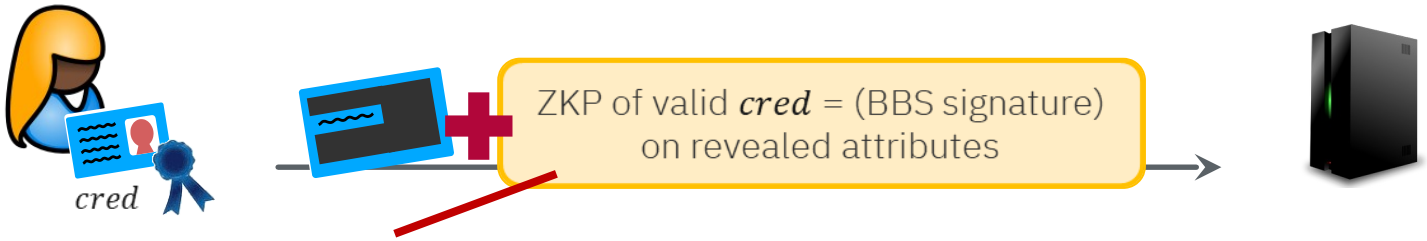
Current Status & Updates (Vasilis)

- Core BBS, BBS per Verifier Linkability, BBS Blind Issuance

New Additions to Blind BBS

- Native Device Binding (Emil)
- Committed Disclosure (Anja)

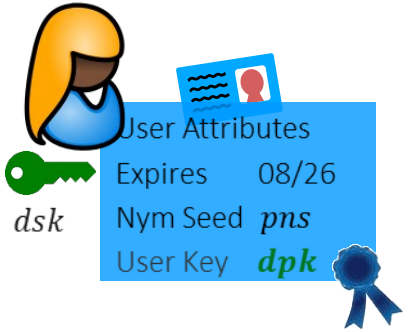
Blind BBS & Committed Disclosure | Motivation



- Core&blind BBS focus on **selective attribute disclosure** (presentation either reveals or hides attributes)
- But in context of anonymous credentials, they can do so much more
  - Pseudonyms, e.g., $nym = PRF(pns, scope)$ with pns being secret attribute in $cred$
  - Range Proofs, e.g., $age > 18$, or $expiration > today$
  - Device binding, e.g., requiring fresh $\sigma_{PoP} = Sign(dsk, nonce)$ for *hardware* key dpk
  - Deniability, cross-credential proofs, issuer-hiding, …
- Required combination of properties (and their instantiations!) is highly use-case specific

Challenge: How to standardize such a modular primitive and protocols built upon that?

Modular Framework instead of Monolithic Standards



Status quo: monolithic ZKP for every use case

ZKP of $cred$ on hidden dpk, pns, exp
and of σ_{PoP} for certified dpk
and $nym = PRF(pns, scp)$ for certified pns
and valid, i.e., $exp > today$ for certified exp

Ignoring revealed/fully hidden attributes for simplicity here

Modular Framework for Anonymous Credentials [LSZ25]

- Core module for $cred$
- Extension Modules working on committed inputs
  Also the ZKP is per module!
- Anonymous Credential system: plug-and-play composition of core + needed modules

ZKP of $cred$ on dpk, pns, exp committed in $C_{dpk}, C_{pns}, C_{exp}$ — **Core Module**: Multi-Message Signature with **Commited Disclosure**

+ ZKP of σ_{PoP} for $dpk \in C_{dpk}$ — **PoP Module**: explicit device binding

+ ZKP of $nym = PRF(pns, scp)$ for $pns \in C_{pns}$ — **Pseudonym Module**

+ ZKP of $exp > today$ for $exp \in C_{exp}$ — **Range Proof Module**

Blind BBS & Committed Disclosure | Concrete Changes

- Committed Disclosure extension to Blind BBS draft → partially blind *presentations*

  Committed disclosure can be combined with both blind BBS and core BBS

- Given $cred := BBS.Sign(isk, (a_1, \dots, a_\ell))$ on attributes $a_1, \dots, a_\ell$ signed by issuer key isk
  - Generate presentations where each attribute a_i can be in one of following states: *hidden, revealed,* ***committed*** New

High-level idea

$$\pi = NIZK\left\{\begin{pmatrix} cred, a_i \in A_{hide} \\ a_j \in A_{com}, o_j \end{pmatrix} : \begin{matrix} Sign.Vf(ipk, cred, a_1, \dots, a_\ell) = 1 \\ \wedge\ \forall a_j \in A_{com} : c_j := Com(a_j, o_j) \end{matrix}\right\}(ctxt, \forall a_j \in A_{com} : c_j)$$

  - Construction uses Pedersen Commitments (with perfect privacy)
  - Public output: proof π, commitments c_j, revealed attributes
  - Private output (to be consumed by other modules): openings o_j

[LSZ25] *Vision: A Modular Framework for Anonymous Credential Systems*. A. Lehmann, A. Sidorenko, A. Zacharakis (Security Standardisation Research Conference

Status and Use Cases

- BBS-based credentials considered in EUDI Wallet: Technical Specification (TS 14)
  https://github.com/eu-digital-identity-wallet/eudi-doc-standards-and-technical-specifications/blob/main/docs/technical-specifications/ts14-zkps-from-mms.md

  EU TS14 already uses the Committed Disclosure concept

  Ongoing ETSI standardisation for ZKP-EUDI (TS 119 476-2, ideally references IETF standards/drafts)

- EUDI use cases will also need **extension modules** (listed in TS14):
  - ☐ Device-binding (next talk)
    https://datatracker.ietf.org/doc/draft-cllz-cfrg-ecdsa-pop/
  - ☐ Range proofs (already defined in PrivacyPass) https://datatracker.ietf.org/doc/draft-ietf-privacypass-arc-crypto/
    or
    https://datatracker.ietf.org/doc/draft-schlesinger-cfrg-act/
  - ☐ Pseudonyms https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-per-verifier-linkability/
  - ☐ Revocation …

- **Data format** for modular sub proofs:

→ Both IETF Drafts complement (Blind) BBS with Committed Disclosure

Applications Beyond EUDI & Crypto Agility

- Anonymous credentials also see increased attention beyond EUDI, e.g. Privacy Pass

→ Extension modules (e.g., range proofs, device binding, pseudonyms, ...) can also be combined with other Signature/MAC schemes, e.g., PS signatures, CMZ MAC, BBS MAC, BBS#, ...

- What about PQC ?
  - BBS with Committed Disclosure has **perfect privacy**; soundness is DL-based
    Suitable for use cases where privacy is critical, e.g., online age verification
  - Need solutions for digital identity infrastructures **now**
    Standards for PQC anonymous credentials are likely several years away
  - Modular design helps with **crypto agility**: critical modules can be upgraded gradually
    Pedersen Commitments as glue can be made quantum annoying (derive generator $h = H(sid)$)

BBS Draft Suite | Summary & Next Steps

Core BBS:

- 2 new PR's, meant to increase implementability.
- Next steps? Should we ask for the next round of cryptographic panel review?

BBS per Verifier Linkability:

- Use the committed disclosures mechanism.
- Decide on the approach and possibly merge it with the blind signatures draft.

BBS Blind Issuance:

- Extended functionality for modularity & native device binding
- Next Steps:
  - Adding native device binding to draft
  - Investigate more options for commitments (e.g. for more compact proofs)

Blind BBS & Native Key Binding | Spare slides

Blind BBS & Native Key Binding | API Changes

```
Commit(committed_messages: bstr[])
  -> (proof: bstr, secret_prover_blind: bigint)
```

⇩

```
CommitInit(committed_messages: bstr[], public_keys: bstr[])
  -> (state: bstr, secret_prover_blind: bigint, challenge: bstr)

CoreCommitProve(sk: bigint, generator: PointG1, challenge: bstr)
  -> (signature: bstr)

CommitFinalize(state: bstr, key_proofs: bstr[])
  -> (proof: bstr)
```

Blind BBS & Native Key Binding | API Changes

```
 deserialize_and_validate_commit(
        commitment_with_proof: bstr,
-       blind_generators: PointG1[],
        api_id: BufferSource,
 ) -> (commitment: PointG1, blind_generators: PointG1[],
keybind_generators: PointG1[])

VerifyBlindSign(
    PK: bstr,
    signature: bstr,
    header: bstr,
    messages: (bstr | bigint)[],
    blind_messages: (bstr | bigint)[],
+   public_keys: bstr[],
    secret_prover_blind: bigint,
) -> VALID or INVALID
```

Blind BBS & Native Key Binding | API Changes

```
-BlindProofGen(
+BlindProofGenInit(
    PK: bstr,    signature: bstr,    header: bstr,    ph: bstr,
    messages: (bstr | bigint)[],     blind_messages: (bstr | bigint)[],
+   public_keys: bstr[],
    message_disclosures: (DISCLOSE | COMMIT | HIDE)[],
    blind_message_disclosures: (DISCLOSE | COMMIT | HIDE)[],
    secret_prover_blind: bigint,
-) -> (proof: bstr, add_zkp_info: (bigint[], bigint[]))
+) -> (state: bstr, add_zkp_info: (bigint[], bigint[]), pk_challenges:
bstr[])


BlindProofGenKeyProve(generator: PointG1, sk: bigint, challenge: bstr)
-> (signature: bstr)

BlindProofGenFinalize(state: bstr, keybind_signatures: bstr[])
-> (proof: bstr)
```

Blind BBS & Native Key Binding | API Changes

```
BlindProofVerify(
    PK: bstr,
    proof: bstr,
    header: bstr,
    ph: bstr,
    disclosed_messages: (bstr | bigint)[],
    disclosed_blind_messages: (bstr | bigint)[],
    message_disclosures: (DISCLOSE | COMMIT | HIDE)[],
) -> VALID or INVALID
```

API unchanged!

Blind BBS & Native Key Binding | Concrete Changes

```
BlindSign:

    [B] = B_calculate(
        PK,
        generators,
        [...blind_generators,  ...keybind_generators],
        commitment + sum(public_keys),
        message_scalars,
        header,
        api_id,
    )
    blind_sig = FinalizeBlindSign(SK, B, public_keys, api_id)
                |
                `>  e_octs = serialize([SK, B, ...public_keys])
```

Blind BBS & Native Key Binding | Concrete Changes

```
BlindCoreProofGenInit: (abridged)

[r1, r2, e~, r1~, r3`, ...m~, ...rk~, ...rk] = calculate_random_scalars(5 + U + 2 * K)
randomized_keys = dpk + keybind_generators[i] * rk[i] for i, dpk in public_keys

(Abar, Bbar_init, D_init, T1_init, T2_init, domain) = BBS.ProofInit(
   PK, signature_result,
   [...generators, ...blind_generators, ...keybind_generators],
   [r1, r2, e~, r1~, r3, ...m~, ...rk~],
   header,
   [...messages, ...(0 for _ in public_keys)],
   [...undisclosed_indexes, (i + L for index i in public_keys)],
   api_id)

D_add = sum(public_keys) * r2
D = D_init + D_add
Bbar = Bbar_init + D_add * r1
Y = P1 + Q1 * domain + sum(randomized_keys) + HJi[...disclosed_indexes] * messages
T1 = T1_init + D_add * r1~
T2 = T2_init + D_add * r3~
```

Blind BBS & Native Key Binding | Concrete Changes

```
BlindCoreProofVerify:

(Abar, Bbar, D, T1, T2_init, domain) = BBS.ProofVerifyInit(
        PK,
        [Abar, Bbar, D, ehat, r1hat, r3hat, hats, cp],
+       [...generators, ...blind_generators, ...keybind_generators],
        header,    disclosed_messages,    disclosed_indexes,    api_id,
)

(Q1, ...Hi) = generators    HJi = (...Hi, ...blind_generators)
Bv = P1 + Q1 * domain + HJi[...disclosed_indexes] * disclosed_messages
Bv_add = sum(randomized_keys)
Y = Bv + Bv_add
T2 = T2_init + Bv_add * cp

challenge = BlindProofChallengeCalculate(
+   [Abar, Bbar, D, Y, T1, T2, domain],
    commitment_init_res, disclosed_messages, disclosed_indexes, ph, api_id,
)

verify(pk, sig, challenge) for (pk, sig) in (public_keys, signatures)
```

Blind BBS & Native Key Binding | Concrete Changes

```
git diff --stat
 src/bbs/index.ts | 893 ++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++
--------------
 1 file changed, 767 insertions(+), 126 deletions(-)
```