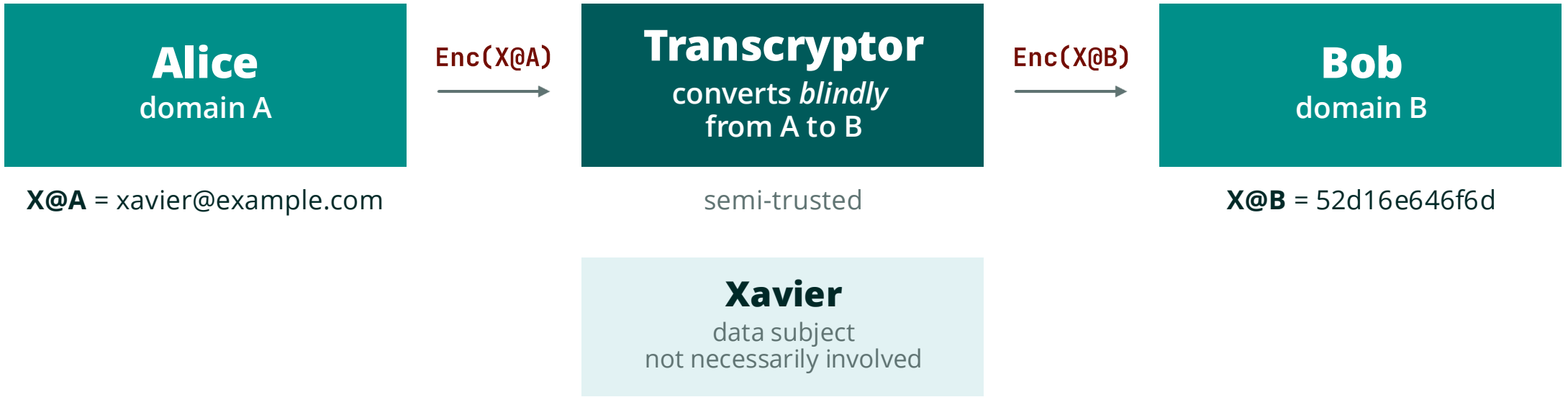
WPES '26

Verifiable blind pseudonymization

via polymorphic encryption using zero-knowledge proofs

Job Doesburg · Bernard van Gastel · Erik Poll
Radboud University

Blind pseudonymous data sharing



Alice and Bob cannot link their pseudonyms for Xavier.
The transcryptor sees neither (but can perform access control).

WHERE THIS IS USED

Use cases

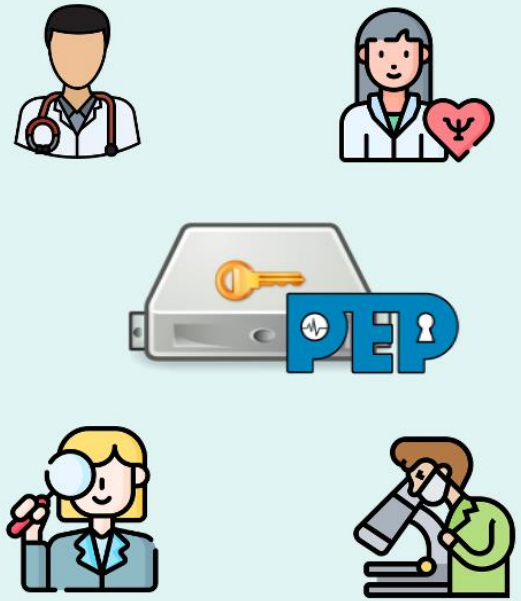
Government agencies

Dutch eID login: DigiD, BSNk



Medical research

PEP-repo



Education

schools, distributors, learning material suppliers

Financial crime detection

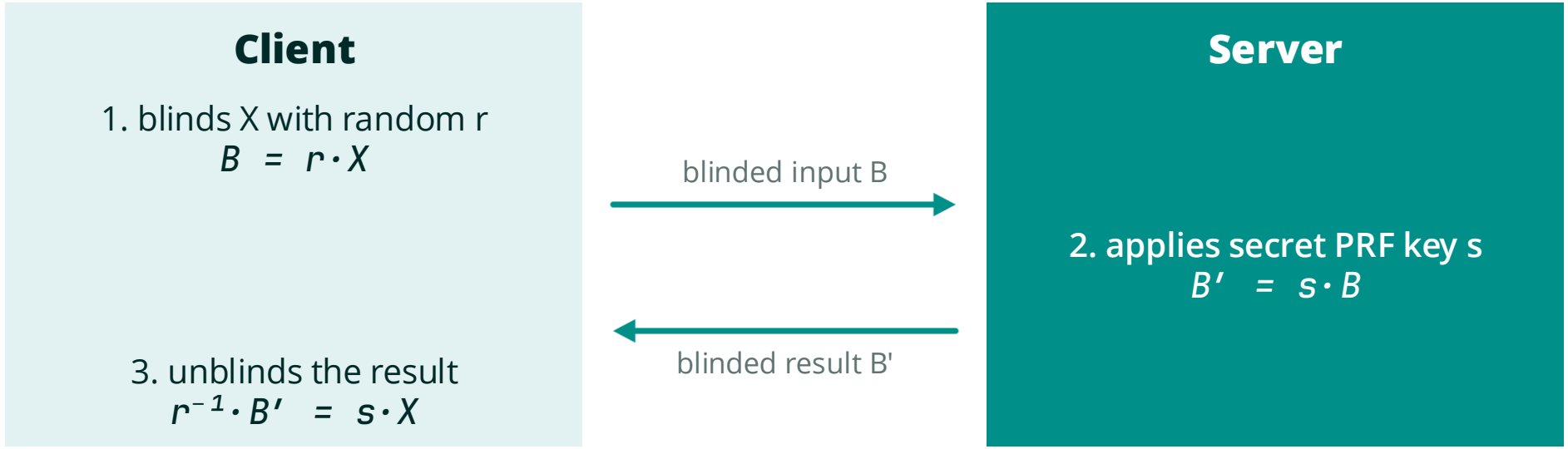
European data spaces

Anywhere different parties need **controlled**, privacy-friendly data sharing.

Blind pseudonymization is an OPRF

Pseudonymization is a PRF. Doing it blindly is an oblivious PRF.

Here: the Diffie–Hellman PRF $X \mapsto s \cdot X$ on an elliptic curve.
Strictly $X = H(x)$; without the hash, $s \cdot X$ is only a weak PRF, pseudorandom only for uniformly random X.



RFC 9497, used in Privacy Pass (anonymous authorization tokens), OPAQUE password checks and private set intersection protocols

The verifiable OPRF (VOPRF)

How does the client know the server computed it correctly?

A PRF output appears random by definition, so a wrong pseudonym looks just as valid as the right one.

1 · COMMIT

$S = s \cdot G$

published once

2 · PROVE

$B' = s \cdot B$

plus a **non-interactive zero-knowledge proof** for that transformation

3 · VERIFY

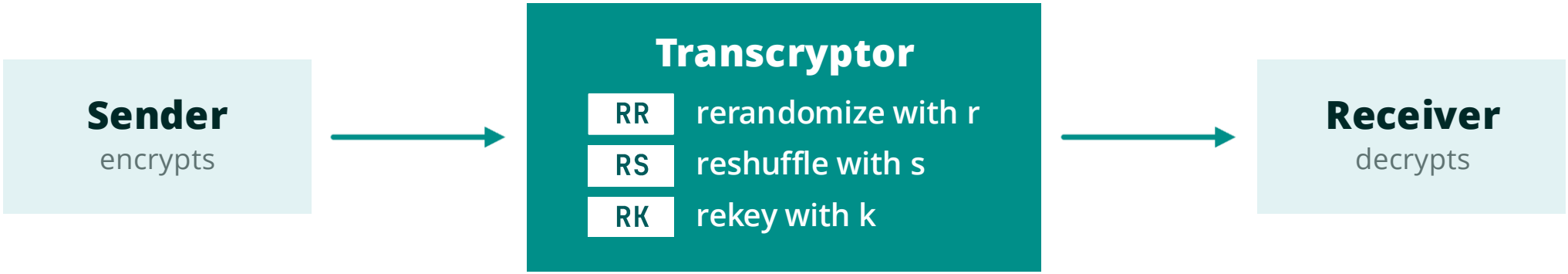
✓ s used

the client checks, without learning s

Schnorr-style DLEQ ZKP: the same s in S = s·G and B' = s·B

Made non-interactive with Fiat–Shamir using a hash.

PEP: an OPRF with three parties



RR injects fresh randomness so copies of a ciphertext become unlinkable

RS applies the PRF: the sender's pseudonym becomes the receiver's

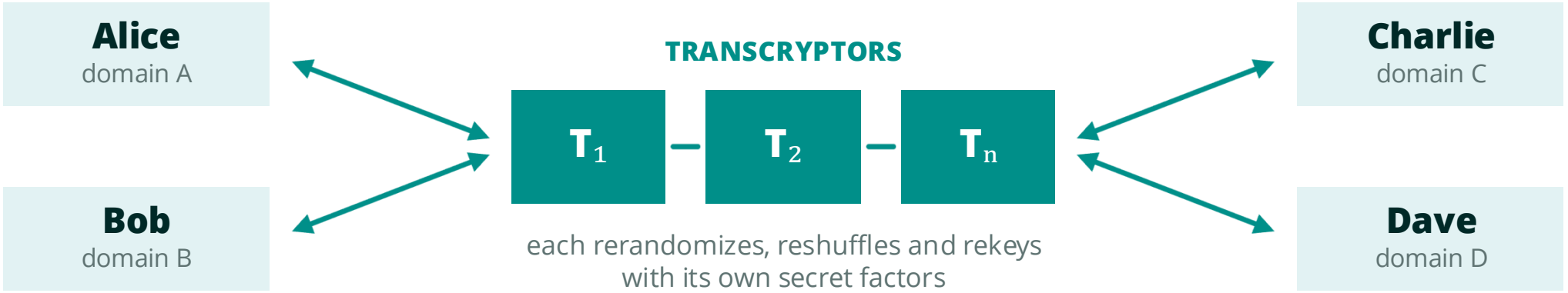
RK re-encrypts, so only the receiver can decrypt

DIFFERENCES WITH A REGULAR OPRF:

Sender ≠ receiver
no single client blinds and unblinds

Asynchronous
encrypt once, convert later

n-PEP: many parties, both ways



N-PEP EXTENSIONS (STM '26):

Distributed

n transcryptors with their own secrets,
no single point of failure

Transitive

multi-directional data sharing between multiple users

OUR WORK

A VOPRF for PEP

Same idea: commit, prove, verify. But the transformation is more complex.

VOPRF

$B \mapsto s \cdot B$

1 point multiplication
1 secret factor

1 ZKP

PEP: RRSK

$B \mapsto s \cdot k^{-1} \cdot B + r \cdot G$
$C \mapsto s \cdot C + r \cdot k \cdot Y$

4 point multiplications
2 point additions
3 secret factors to combine

5 ZKPs

N-PEP: RRSK2

$s = s_from^{-1} \cdot s_to$
$k = k_from^{-1} \cdot k_to$

4 point multiplications
2 point additions
5 secret factors to combine

7 ZKPs

Intermediate values must stay hidden from the verifier.

FINDINGS

Two observations

1 A proof of rerandomization reveals the link it should hide.
→ In n-PEP, **transcryptors can verify each other**, so recipients stay unlinkable.

2 How you construct the proofs decides which intermediate values leak and may break the whole scheme (not just integrity).
→ Prove **reshuffle and rekey together**, **rerandomization separately**.

EVALUATION

Cheap, on fast curves

VERIFIABLE PEP (OURS)

DLEQ zero-knowledge proofs for ciphertext transformations

Fast Curve25519 / Ristretto255

No pairings, 54–62 scalar multiplications

≈ 2 ms

measured (our open-source libpep implementation)

CL SYSTEM (CCS '15)

Convertible signatures on plaintext pseudonyms

Pairing-friendly curves required

28 pairings, 94 scalar multiplications

> 17 ms

projected (no public implementation available)

Ours: full single transcryptor flow (encrypt, transcrypt, verify, decrypt), single core Apple M2, no batching.
CL: 28 pairings × ~0.6 ms (blst). On BLS12-381, a pairing costs ~5–10 scalar multiplications, each ~2–3× slower than on Ristretto255.



GITHUB



PAPER

Takeaways

1. PEP is a three-party OPRF.
2. DLEQ zero-knowledge proofs make it verifiable, **efficiently: no pairings**.
3. Unlinkability is fragile: the structure of proofs matters.

---

github.com/NOLAI/libpep

job.doesburg@ru.nl