

Security challenges in HSM-based wallets — and how Split-ECDSA solves them

(<https://wellet.nl/SECDSA-EUDI-wallet-latest.pdf>)

Eric Verheul — Eric.Verheul@wellet.nl

17 april 2026



Wellet

The wallet done well

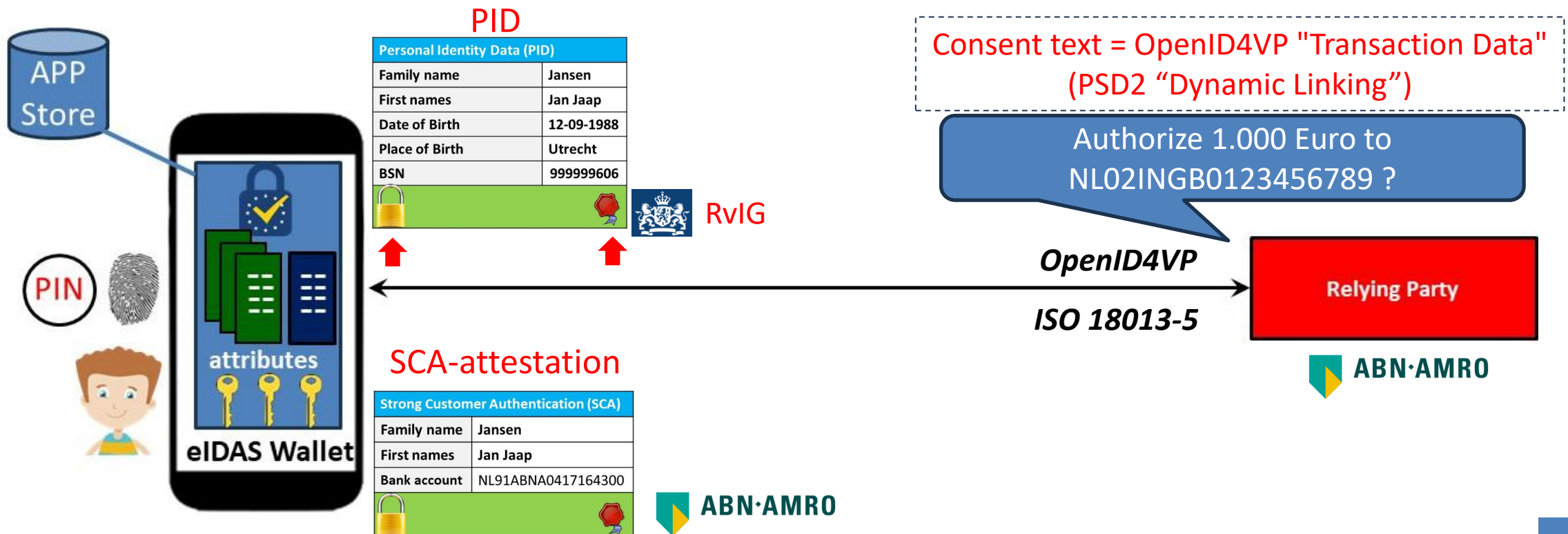
wellet.nl

Outline

- The HSM-based EUDI Wallet
- HSM-based wallet security and implementability challenges
- HSM-based wallet design on SECDSA (Split-ECDSA)
- Java reference implementation

EUDI wallet

- European Digital Identity (EUDI) wallet introduced in updated eIDAS regulation 2024/1183 (11 April 2024).
- Objective: empower EU citizens and residents with an EU-interoperable digital identity solution for:
 - secure and privacy-preserving **strong authentication**,
 - **qualified ('legally binding') signing** and **authorizing banking transactions** (SCA) — both requiring non-repudiation, i.e., cryptographic proof that the user consented
- No centralized identity provider — no single point of privacy exposure
- [Architecture Reference Framework \(ARF\)](#): EU standardized technical blueprint of the EUDI wallet



Wallet governance test case: liability for payment fraud

► Context

- Governments dictate the EUDI wallet architecture via the eIDAS regulation, including the authentication mechanisms used for **authorizing banking transactions** (SCA)
- Under PSD2, banks are liable for payment fraud unless the customer is demonstrably fraudulent or negligent
- Wallet architectures **lacking adequate non-repudiation** increase repudiation risk and therefore banks' fraud liability

► Question

- ? *Can banks hold governments liable for payment fraud caused by insufficient non-repudiation in government-mandated EUDI wallets?*
- ? *More generally, can parties hold governments liable for fraud resulting from insufficient non-repudiation in government-mandated EUDI wallets?*

► Implication for wallet design

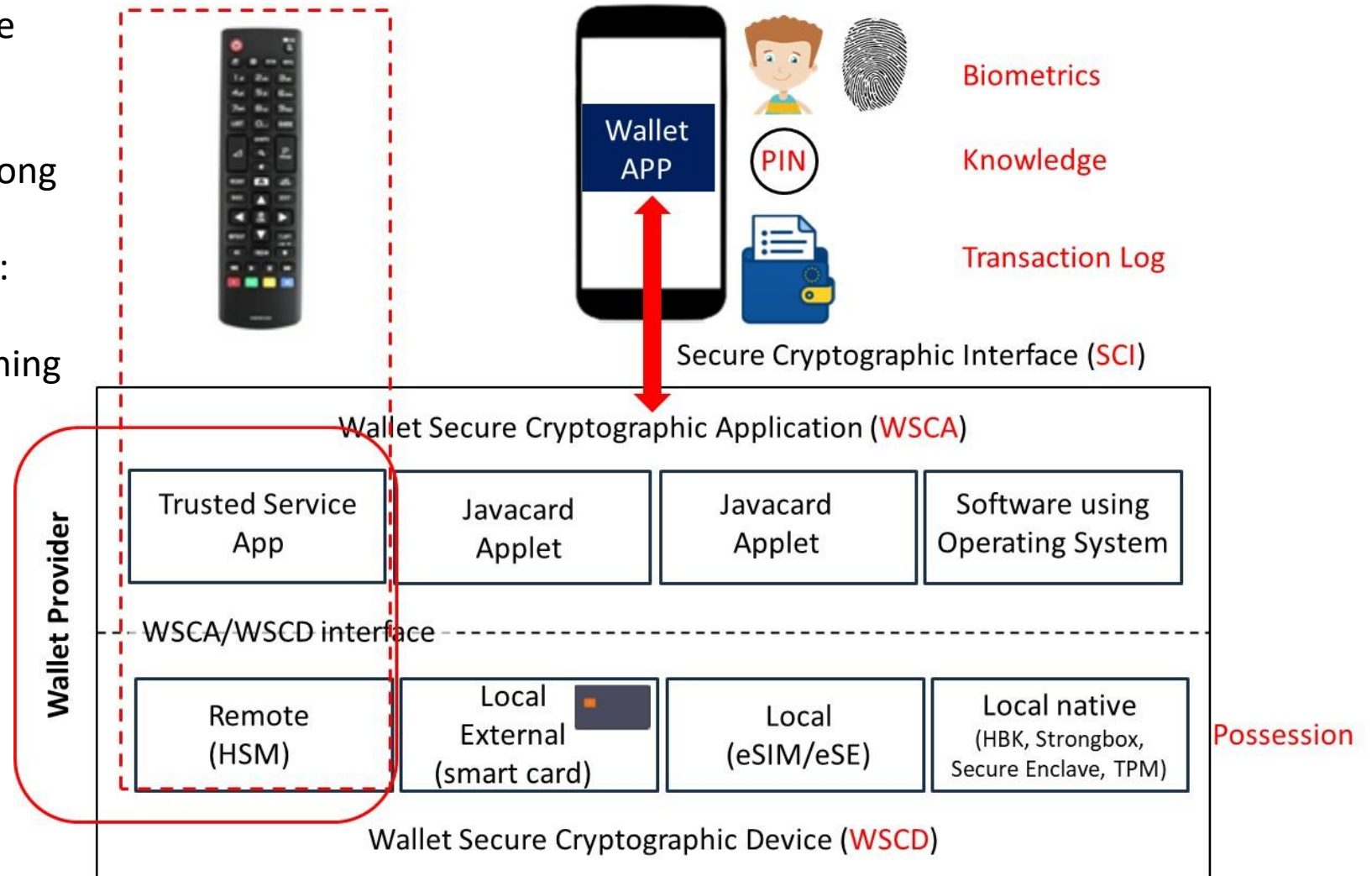
- ✓ Verifiable user consent (non-repudiation) is not merely a technical requirement but a necessary legal demarcation for allocating liability between government and banks and, more generally, parties requiring non-repudiation in authentication.

EUDI wallet: architectures in the ARF

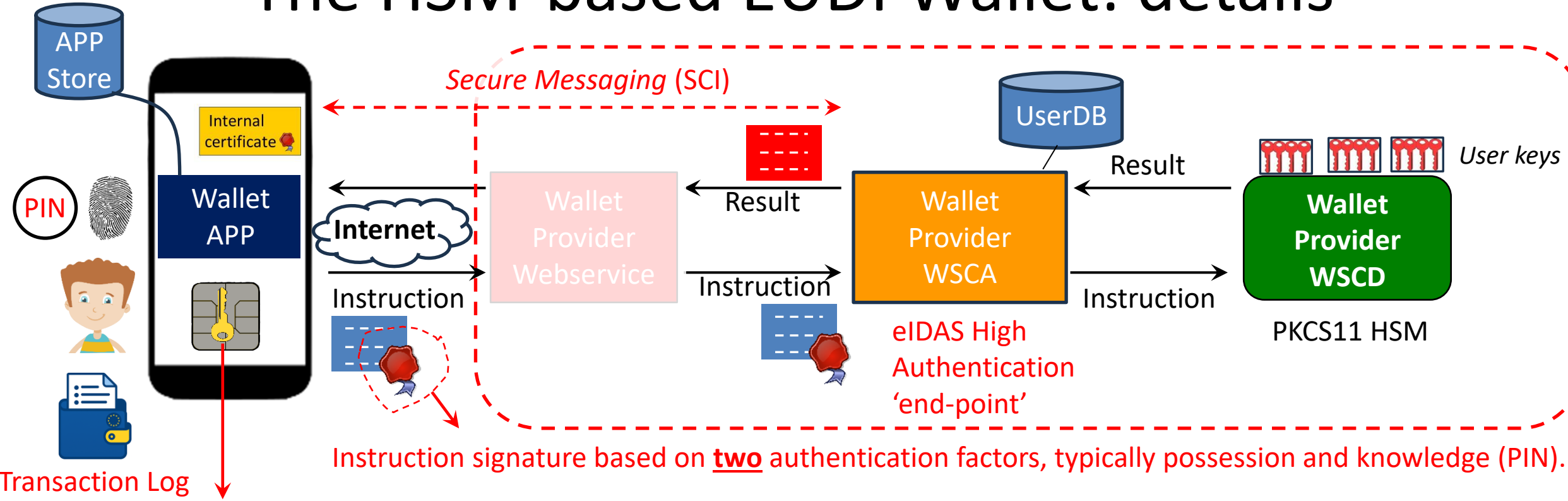
- **HSM**: Hardware Security Module
Tamper-resistant device for key storage and crypto functions.
Keys never leave the HSM → strong security
- **PKCS#11**: [Standard interface](#) for:
 - Key management,
 - Standard operations, e.g. signing and encryption,**Avoids vendor lock-in!**



HSM-based EUDI Wallet



The HSM-based EUDI Wallet: details



Instruction signature based on two authentication factors, typically possession and knowledge (PIN).

Native mobile cryptographic Hardware
(Secure Enclave, HBK, StrongBox, TPM)

- SCI:** Facilitates eIDAS High authentication & wallet-WSCA secure messaging (**Secure Cryptographic Interface**)
- WSCA:** eIDAS High Authentication 'end-point' & manages keys in WSCD on behalf of user
- WSCD:** Core cryptographic hardware

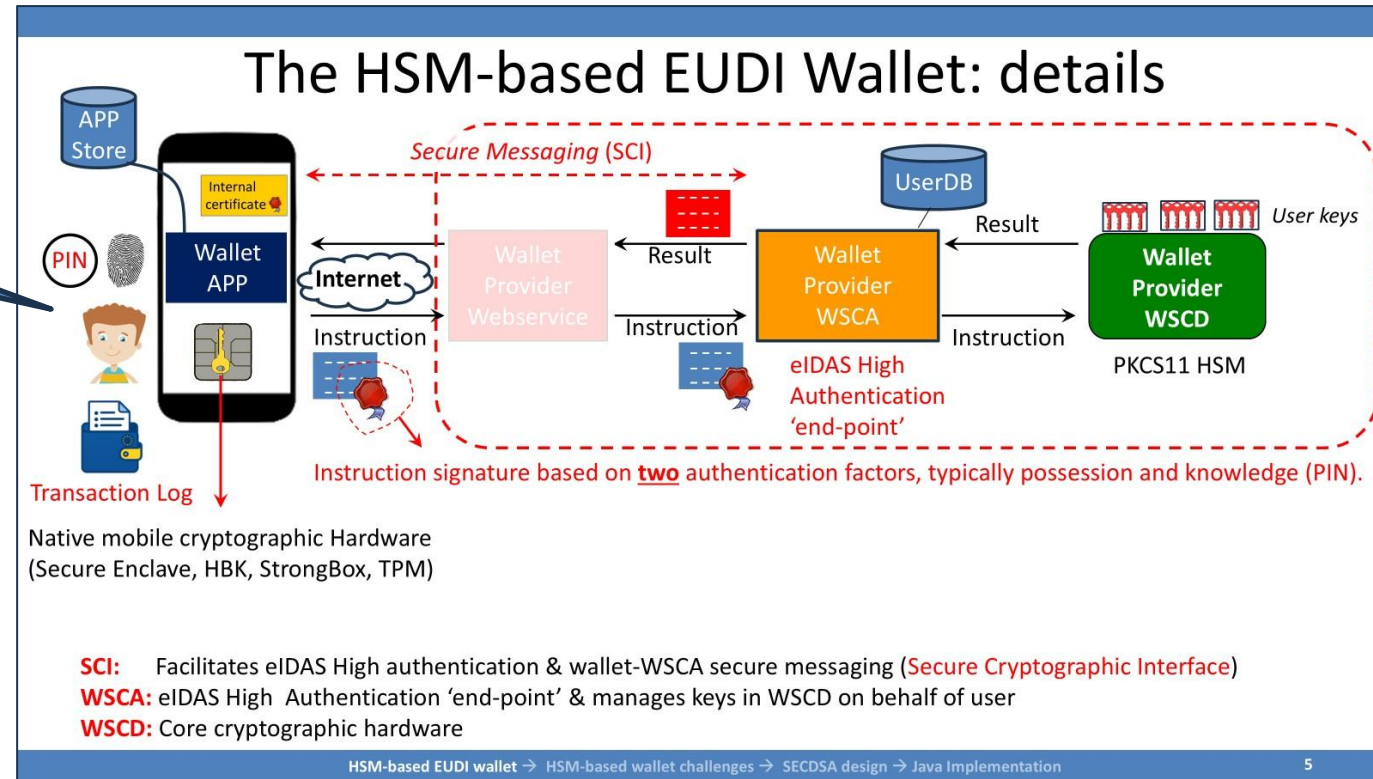
The HSM-based EUDI Wallet: consent handling

Consent text

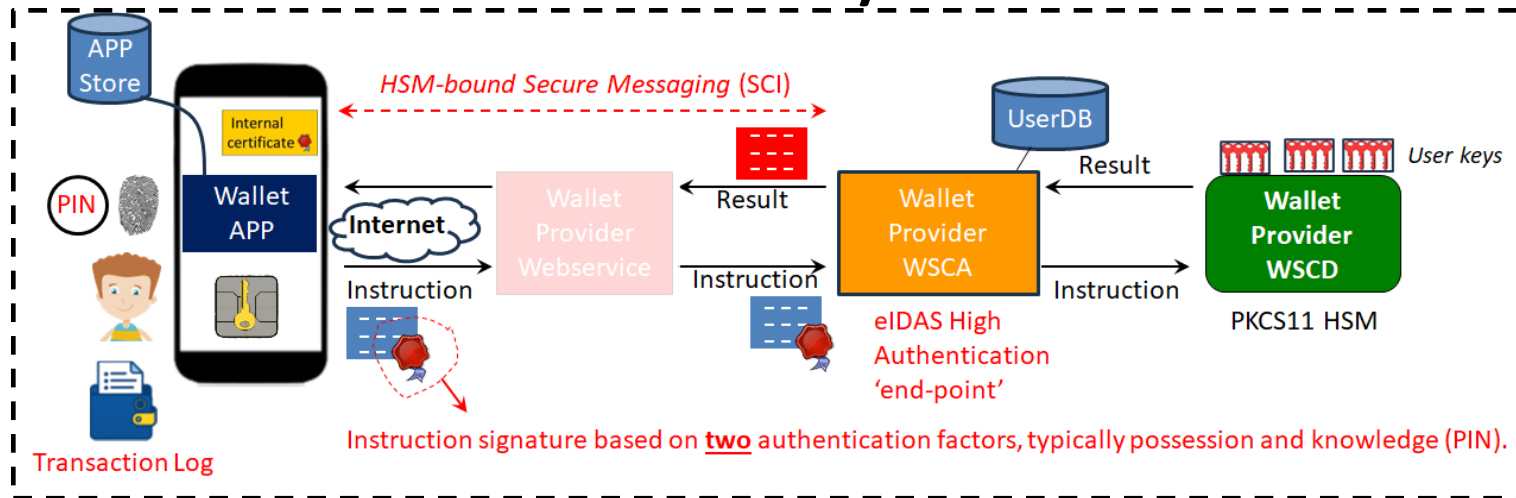
“Dynamic Linking” (PSD2)

Authorize 1.000 Euro to
NL02INGB0123456789?

In qualified ('legally binding') signing and authorizing banking transactions (SCA) use cases the “consent text” should somehow be included in the wallet Instruction to the wallet provider, e.g. as hash value.



Wallet security certification



- Certification requirements mostly in [eIDAS implementation regulation 2024/2981](#)
- eIDAS CIR 2024/2981 stipulates risks to be mitigated (**Risk Register**)
- National certification schemes apply in anticipation of final [European cybersecurity certification scheme](#)
- Formally, WSCD and WSCA not required to be Common Criteria certified as long as they are proven resistant against “high attack potential”
- WSCD best practice: **eIDAS HSM** = Common Criteria HSM certified against EN 419221-5 (‘PP’) of which several exist (see paper Section 3.6).
- Wallet – WSCA needs to adhere to CIR 2015/1502 (eIDAS High authentication): WSCA as bespoke **HSM-firmware** or as **software (container)**

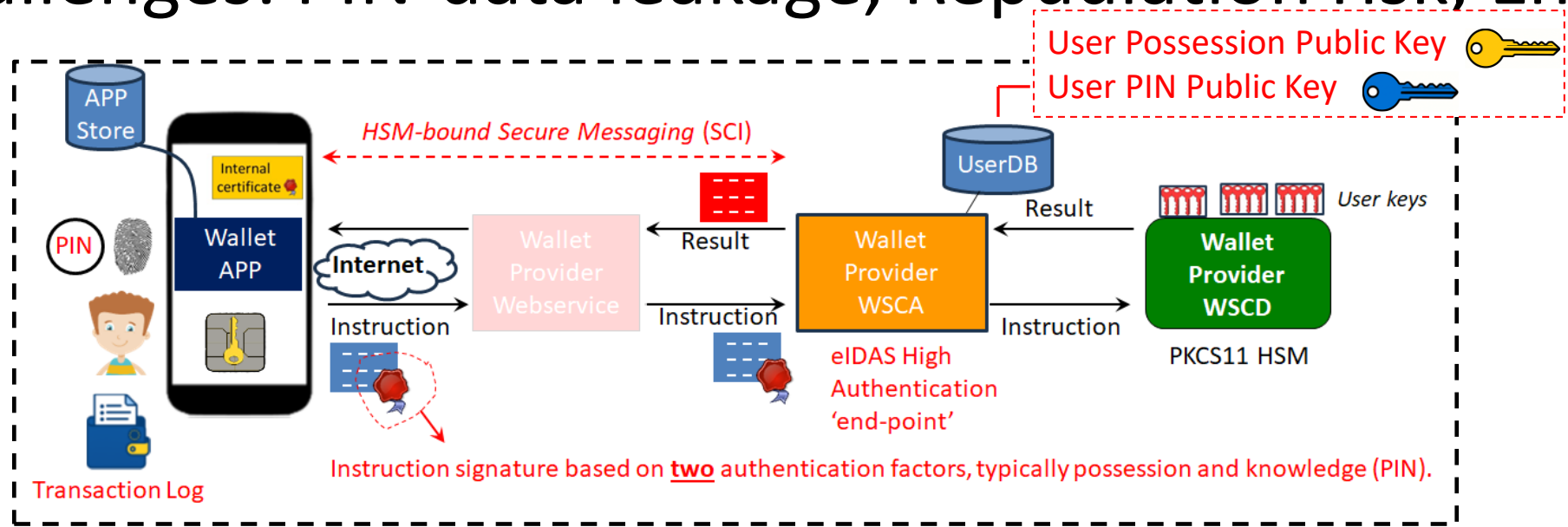
3. Evaluation activities related to the wallet secure cryptographic application (WSCA)

eIDAS CIR 2024/2981

- (1) National certification schemes shall require that a WSCA, as part of a wallet solution, is evaluated against the requirements of at least assurance level high as set out in Implementing Regulation (EU) 2015/1502.
- (2) This evaluation shall include a vulnerability assessment, as set out in EN ISO/IEC 15408-3:2022 at level AVA_VAN.5, as set out in Annex I of Implementing Regulation (EU) 2024/482, unless it is duly justified to the certification body that the security characteristics of the WSCA make it possible to use a lower assessment level while keeping the same overall assurance level high as set out in Implementing Regulation (EU) 2015/1502.

Wallet challenges: PIN-data leakage, Repudiation risk, Efficiency

[URL1 NL-Wallet](#)
[URL2 NL-Wallet](#)
[URL D-Wallet](#)



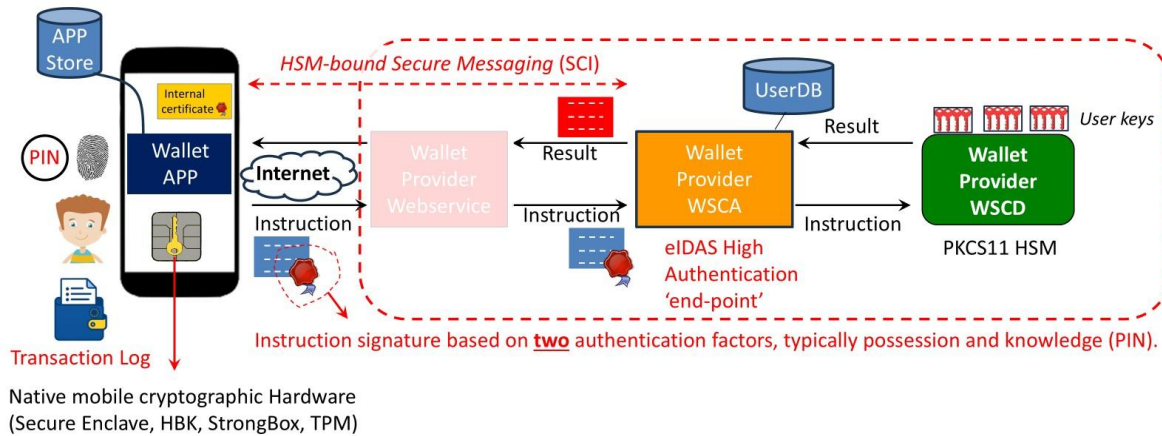
In Risk Register!

Common PIN Design vulnerabilities/issues

- Used in several wallets including current NL- and D- wallet
 - Wallet instructions are signed with two signatures: possession based and PIN based
 - **Vulnerability #1:** PIN based signatures (and keys) processed by WSCA allow for PIN brute-force ('PIN-data Leakage')
 - **Vulnerability #2:** PIN based signatures cannot be stored in wallet Transaction Log increasing 'Repudiation Risk' (and liability)
 - **Inefficient:** requires several HSM-calls: setup SCI, validation internal certificate, validation PIN-signature
- ✘ **Vulnerability #1** requires WSCA to be implemented as **HSM-firmware**, i.e. bespoke code running inside HSM.
 ✘ HSM-firmware still does not address the repudiation risk, i.e. **Vulnerability #2 plus** limits scalability, flexibility, performance, and introduces vendor lock-in.

The SECDSA (Split-ECDSA) based design

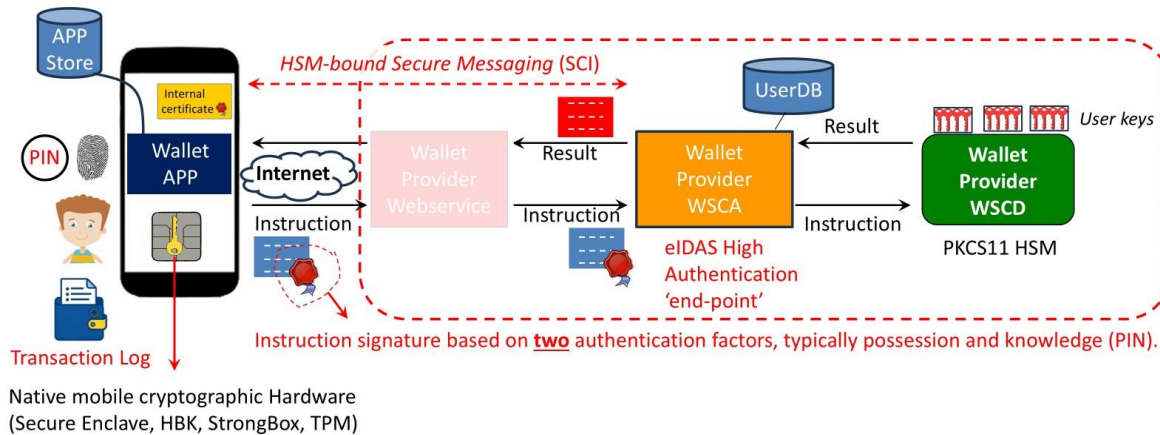
The HSM-based EUDI Wallet: details



- ✓ Avoids PIN-data Leakage by HSM-bound PIN verification
- ✓ Ensures non-repudiation through publicly verifiable, transferable user-signed instructions — essential for qualified signing and banking transaction authorization
- ✓ Direct setup of HSM-based Secure Messaging (SCI)
- ✓ No HSM-firmware WSCA but software-based WSCA (e.g., container) and standard ([PKCS#11](#)) HSM— no vendor lock-in
- ✓ Uses only one HSM call overhead per instruction — scalable, flexible, high-performant and cost-efficient **MINIMAL OVERHEAD**
- ✓ Allows for CC certified (EN 419221-5) HSM for optimal compliance

The SECDSA (Split-ECDSA) based design

The HSM-based EUDI Wallet: details

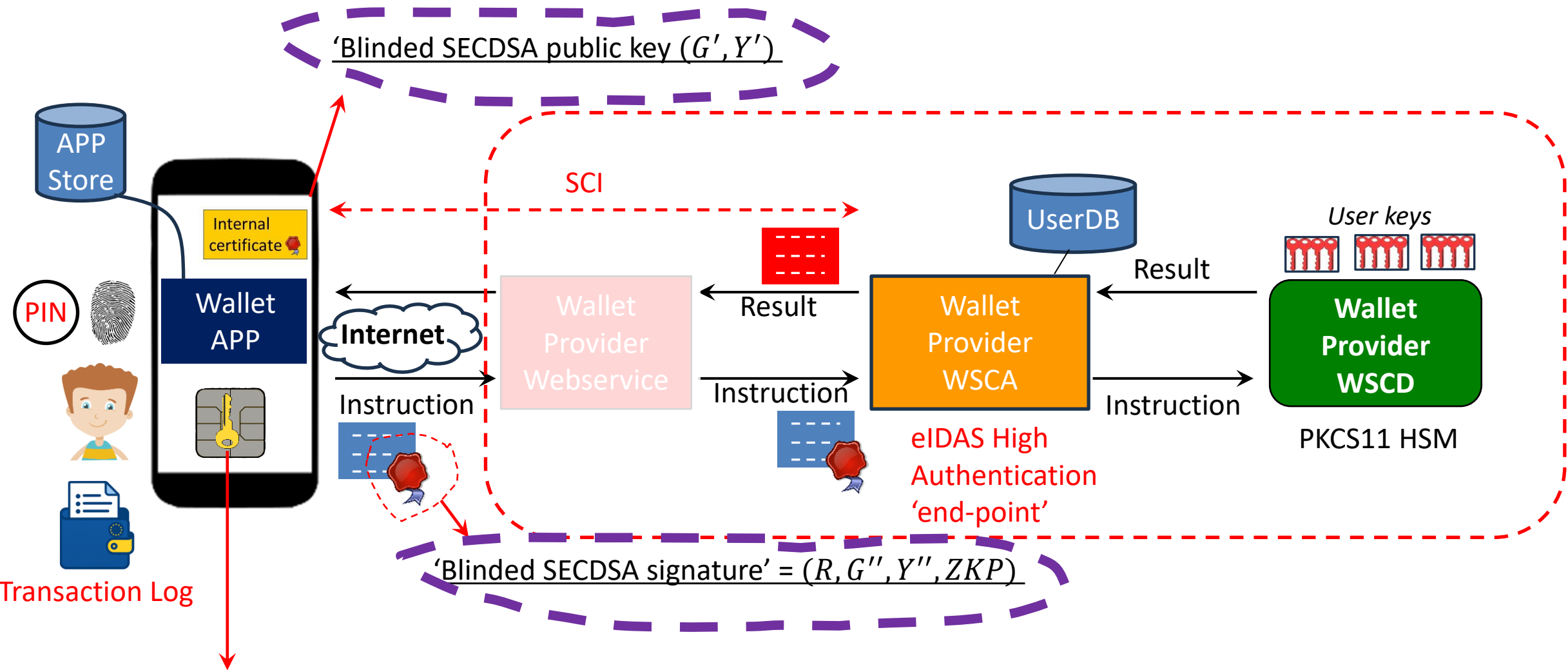


- ✓ Avoids PIN-data Leakage by HSM-bound PIN verification
- ✓ Ensures non-repudiation through publicly verifiable, transferable user-signed instructions — essential for qualified signing and banking transaction authorization
- ✓ Direct setup of HSM-based Secure Messaging (SCI)
- ✓ No HSM-firmware WSCA but software-based WSCA (e.g., container) and standard ([PKCS#11](#)) HSM— no vendor lock-in
- ✓ Uses only **one** HSM call overhead per instruction — scalable, flexible, high-performant and cost-efficient
- ✓ Allows for CC certified (EN 419221-5) HSM for optimal compliance

Technique









- SECDSA combines possession and PIN-derived key into one ECDSA public key = **SECDSA public key Y**
- Avoids brute-force and repudiation risks by:
 1. Blinding of public key Y ('homomorphic' encryption) in Internal Certificate
 2. Blinding of SECDSA signature ('homomorphic' encryption) and added Zero-Knowledge Proof (ZKP)
 3. Allows wallet provider making SECDSA instruction signatures **publicly verifiable & transferable**

Cryptographic basics of the SECDSA HSM-based EUDI Wallet



See: https://wellet.nl/SECDSA_HSM-wallet_cryptography_Radboud.pdf

Java implementation

  GenerateInternalCert(String, String, String): InternalCert
  GenESInstruction(byte[], byte[], InternalCert, String, String, String): EncSignedInstruction
  ExecESInstruction(EncSignedInstruction, InternalCert, String, String): TransactionRecord
  VerifyTRecord(InternalCert, Certificate, TransactionRecord): boolean

Java implementation

Java reference implementation available, demonstrating ease of adoption:

1. Internal certificate issuance (Protocol 4)
→ *Wallet Activation: Internal Certificate with blinded SECDSA key*
2. Encrypted/signed instruction generation (Algorithm 37)
→ *Wallet Usage: Wallet generates End2End encrypted + blinded SECDSA signed Instruction*
3. Instruction execution, verifiable Transaction Record generation (Algorithm 38)
→ *Wallet Usage: Wallet Provider decrypts/verifies/executes Instruction + creates verifiable proof user signed instruction*
4. Third-party Transaction Record verification (Algorithm 39)
→ *Any third-Party, e.g. wallet, judge or mediator, verifies proof that user signed instruction; requires no secret data*

Full spec: Annex D of specification (<https://wellet.nl/SECDSA-EUDI-wallet-latest.pdf>)

Java implementation

```
*** HSM-based approach: activation/issuance of internal certificate
*** Request Wallet Provider for wallet activation and internal certificate issuance Protocol 4 Annex D (Enter Return):
Choose new PIN: 12345
Internal Certificate successfully issued.
Info: Internal Certificate :aced00057372001a5345434453414857616c6c657424496e7465726e616c43657274940fad8ee552b6fa0200054c00024964740

Enter instruction for Wallet Provider/WSCA (any string, e.g. "Generate Key"): GenKey
**Secure Cryptographic Interface (SCI)**
Enter additional data to be End2End (GCM-AES) encrypted between wallet and WSCA, e.g. EN 419221-5 authorization data: SECRET_DATA
Enter PIN to sign, End2End encrypt and send instruction Alg. 37: 12345
Info: Message sent to Wallet Provider consists of:
Info: 1a. Ephemeral Key R:          3059301306072a8648ce3d020106082a8648ce3d03010703420004f60218e282ec85b3fbb4fe3b1ca7635a9473b1003
Info: 1b. Blinded Ephemeral Key R': 3059301306072a8648ce3d020106082a8648ce3d03010703420004007606c86939fbe2f979811469dfd50f0135a05f2
Info: 1b. Message Serial Number (plaintext): 0000000000000001
Info: 2. CiphertextAndTag: aced0005757200025b42acf317f8060854e00200007870000006bb26e4092089e4d769a1a4c133f1889ddf6875def3e76f4eb08e
Info: 3. Challenge (plaintext): aced00057372001a5472616e73666572457150726f6f66245369676e65644368616c9081229c37d4949c0200025b0003536

Wallet Provider/WSCA: End2End encrypted data in received instruction: SECRET_DATA
PIN correct, instruction "GenKey" accepted and executed and Transaction Record generated (Alg. 38): aced00057372001f534543445341485
Let Transaction Record be verified wallet or by any party, e.g. judge or mediator Alg. 39 (Enter Return):
Transaction Record correct: true
```

Java implementation

Conclusion

▶ Four HSM-based wallet challenges addressed by Split-ECDSA (SECDSA)

- ✓ **PIN-data leakage** — HSM-bound PIN verification prevents brute-force attacks on the PIN
- ✓ **Non-repudiation** — user-signed instructions are publicly verifiable and transferable, essential for legally binding signing and authorizing banking transactions
- ✓ **Efficient WSCA** — software-based WSCA (e.g., container) instead of bespoke HSM-firmware: scalable, flexible, no vendor lock-in
- ✓ **Minimal HSM overhead** — only **one** HSM call per instruction, enabling scalable, high-performance deployments

▶ Governance

- ✓ Verifiable user consent (non-repudiation) is not merely a technical requirement but a **necessary legal demarcation** for allocating liability between government and relying parties