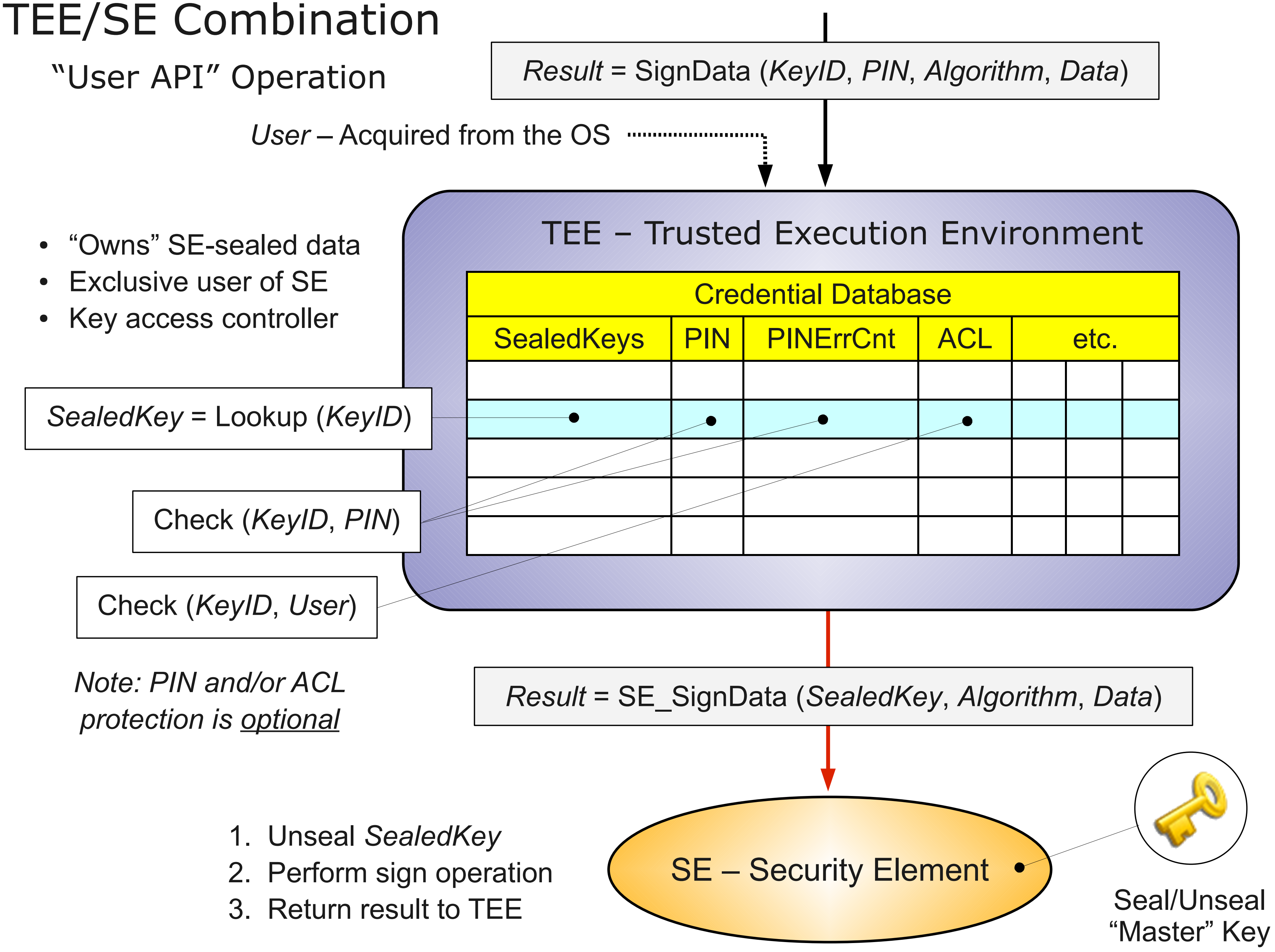
Mapping SKS into a TEE/SE "Combo"

An SKS (Secure Key Store) may be self-contained like in a smart card, but it may also be architected as a TEE (Trusted Execution Environment) and SE (Security Element) combination.

This very early TEE/SE draft represents SKS as described in V.60.

TEE/SE Combination

“User API” Operation



Result = SignData (*KeyID*, *PIN*, *Algorithm*, *Data*)

User – Acquired from the OS

- “Owns” SE-sealed data
- Exclusive user of SE
- Key access controller

TEE – Trusted Execution Environment

Credential Database

SealedKeys	PIN	PINErrCnt	ACL	etc.		
●	●	●	●			

SealedKey = Lookup (*KeyID*)

Check (*KeyID*, *PIN*)

Check (*KeyID*, *User*)

Note: PIN and/or ACL protection is optional

Result = SE_SignData (*SealedKey*, *Algorithm*, *Data*)

1. Unseal *SealedKey*
2. Perform sign operation
3. Return result to TEE

SE – Security Element

Seal/Unseal “Master” Key

Q & A

Question: Is this really secure?

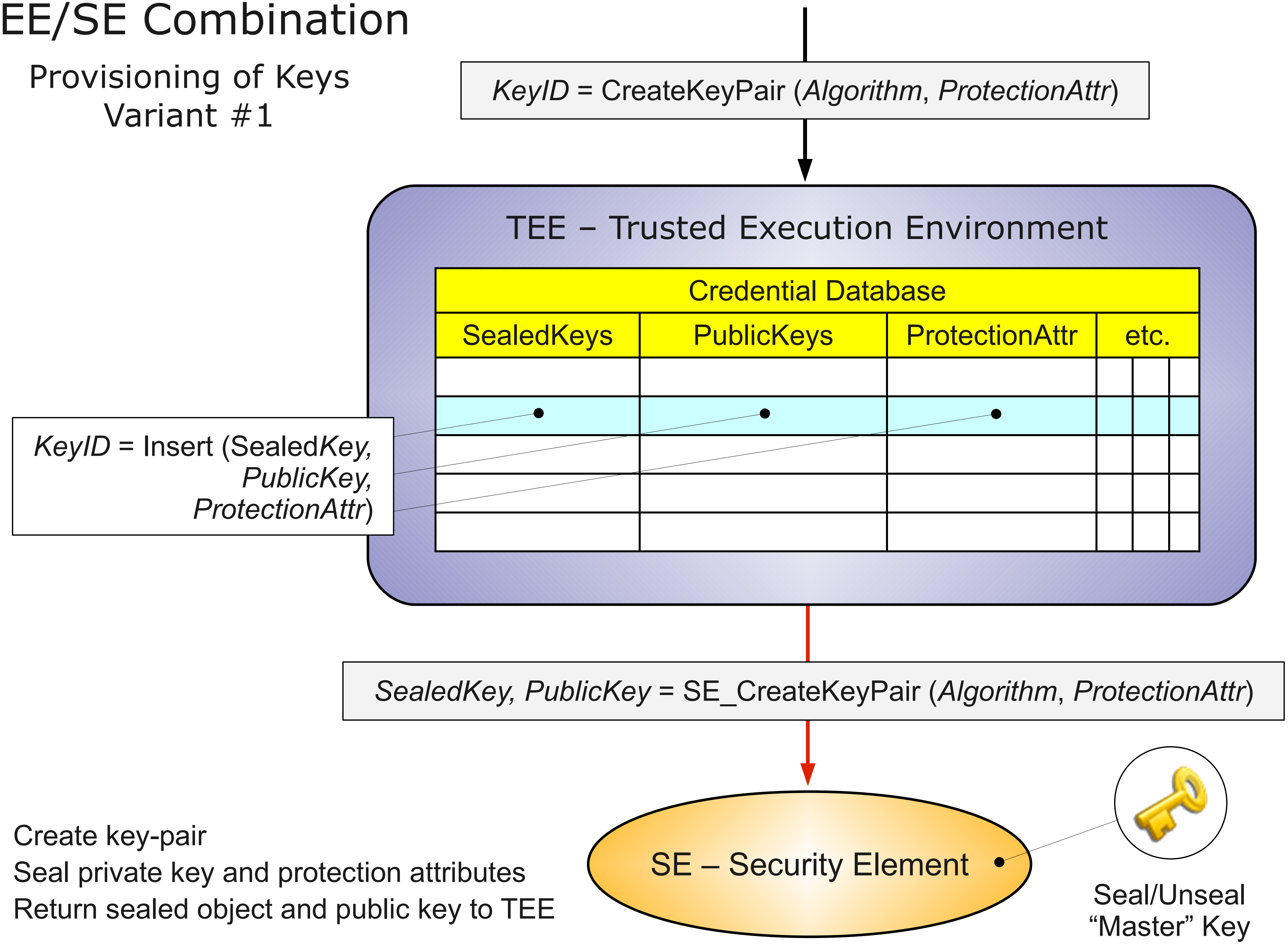
Rhetoric answer: Do TEE- or application-based embedded secrets and obfuscated code actually bring any sustainable and provable security values to the table?

Question: Could there even be advantages of using the TEE for access control?

Answer: Yes, it enables combining various kinds of access controls like restricting keys to specific applications or users, as well as using device-wide PINs. A TEE can also provide challenge-response authentication and encrypted tunnels without burdening the SE. A TEE typically also supports a "trusted GUI" removing PIN-entry from potentially untrusted applications

TEE/SE Combination

Provisioning of Keys
Variant #1



KeyID = CreateKeyPair (*Algorithm*, *ProtectionAttr*)

TEE – Trusted Execution Environment

Credential Database

SealedKeys	PublicKeys	ProtectionAttr	etc.
●	●	●	

KeyID = Insert (Sealed*Key*, *PublicKey*, *ProtectionAttr*)

SealedKey, *PublicKey* = SE_CreateKeyPair (*Algorithm*, *ProtectionAttr*)

SE – Security Element

Seal/Unseal “Master” Key

1. Create key-pair
2. Seal private key and protection attributes
3. Return sealed object and public key to TEE

TEE/SE Combination

Provisioning of Keys
Variant #1

The Good

- Keys are protected from theft
- Keys are stored with protection attributes like “non-exportable” which can be enforced by the SE
- Stateless SE operation – No storage or NVRAM wear-out issues

The Bad

- Does not provide a suitable foundation for importing encrypted data to both the TEE and SE
- Does not support transaction-based provisioning (makes very little use of the TEE)
- Does not provide SE binding information to issuers

TEE/SE Combination

Provisioning of Keys
Variant #2

A Completely Revised Scheme

- Create a shared, SE-attested `SessionKey` between the SE and the Issuer
- Seal the `SessionKey` and some additional data and store this object in the TEE
- Return the attestation to the Issuer who now (through specific SE provisioning methods using the sealed provisioning object), can securely *Generate+Attest*, *Import*, and *Export* data based on the `SessionKey`

Maintains stateless SE operation in spite of highly stateful, transaction-based provisioning

Documentation:
http://webpki.org/papers/SKS-mapped-into-a-TEE-SE-combo.txt