Attestation Proof of Association

— provability that attestation keys are bound to the same hardware and person —

Eric R. Verheul
eric.verheul@logius.nl

Version 1.0, 18th September 2024

Abstract We specify a wallet provider issued attestation called *Wallet Trust Evidence* (WTE) and three related specific instructions for the EUDI Wallet cryptographic hardware, most notably the generation of a *Proof of Association* (PoA). These allow the EUDI Wallet providing verifiable assurance to third parties (issuers, relying parties) that attestation private keys are not only bound to conformant cryptographic hardware but also that they are bound to the *same* such hardware. This allows the EUDI Wallet meeting eIDAS Level of Assurance "high" as well as operating in a privacy friendly manner. The instructions specified in this document cater for convenient implementation in all envisioned EUDI Wallet architectures including those based on a GlobalPlatform [17] based Secure Element such as an eID-card or an embedded SIM (eSIM). By their simplicity, the three instructions also allow for convenient Common Criteria certification. This document is a further refinement and cryptographic concretisation of the WTE/PoA logic specified in the wallet Wallet Architecture and Reference Framework [1], which is based on the EPIC-09 result developed in a cooperation between the NI-Scy consortium and the eIDAS expert group. However, the present draft document is meant for discussion only and not approved by the NI-Scy consortium, the eIDAS expert group or Dutch government.

Keywords: eIDAS assurance level High, EUDI Wallet, Key attestation, Privacy friendly attestation issuance and presentation, Wallet Trust Evidence

Contents

List of Figures

VERSION CONTROL

Version	Date	Description
0.1	2024-02-21	Initial Draft
0.2	2024-03-05	• Changed title • Further clarification and typo fixing • Added motivation and informative examples (annex)
0.3	2024-03-11	• Further clarification and typo fixing • Added proof of association heuristics in Section 3 • Added Section 4.3 indicating compatibility with BBS+ and Idemix protocols
0.4	2024-03-19	• Processed first comments • Added relevance of (WTE) attestation signature soundness to Section 3 • Only kept the Schnorr based PoA proposal and removed the ECDSA ones.
0.5	2024-04-10	• Added abstract • Added ECDH-MAC specification to Annex E following ISO 18013-5 • Added the distributed WSCD (Figure 2) • Added security proof to Section 3
0.6	2024-04-26	• Fixed typos • Added further clarification and examples • Added missing proofs of possession in informative Annexes B, C.
0.61	2024-05-03	• Added one extra mechanism in Section 3 avoiding the wallet to rawly sign an issuer challenge • Added new Annex G illustrating the three mechanisms avoiding the wallet to rawly sign an issuer challenge in data flows
0.62	2024-05-26	• Illustrated the flexibility of the WTE/PoA logic in Section 4.1 by showing three WTE architectures with different privacy and security properties whereby also introducing the Issuer Trust Evidence (ITE) object; issuer specific WTEs avoiding linkability. • Added the HSM based wallet in Figure 2.
0.63	2024-05-30	Added comparison with "simplified WTE/PoA" of the draft standard ISO 23220-3 [27, Annex C.6.5] at the end of Section 2.
0.64	2024-06-09	For simplicity the document now only discusses the attestation signature algorithms specified in ISO 18013-5 [26], i.e. EdDSA, ECDSA and ECDH-MAC.
1.0	2024-09-18	Finalized for publication

1   Introduction

The update on 11 April 2024 [13] to the 2014 eIDAS regulation [12] introduces an *European Digital Identity Wallet* (hereafter: EUDI Wallet or for brevity sometimes simply wallet). According to [13], the EUDI Wallet "shall enable the user, in a manner that is user-friendly, transparent, and traceable by the user, to [..] securely request, obtain, select, combine, store, delete, share and present, under the sole control of the user, person identification data and, where applicable, in combination with electronic attestations of attributes, to authenticate to relying parties".

These *Relying Parties* can be public and private services. The EUDI Wallet is provided to users by a *Wallet Provider*. As every European member state is required to provide an EUDI Wallet to its citizens, each member state shall have at least one Wallet Provider. An EUDI Wallet allows the user to present *attributes* to relying parties in the form of *electronic attestation of attributes* (hereafter: attestations). According to [13] an *attribute* means "a characteristic, quality, right or permission of a natural or legal person or of an object. Also, "electronic attestation of attributes" means an attestation in electronic form that allows attributes to be authenticated. Compare Figure 1 below.

Attestations are issued by *Attestation Providers*. Both provider types are considered trusted and can either be private or public. Public providers are typically government or state-affiliated organizations offering services to the public, while private providers are owned and operated by independent, non-governmental entities. Particular public providers provide *Personal Identification Data* (PID) which contain the basic identification data of the user comparable with a conventional identity document but then usable online. Although a PID technically resembles an attestation it formally is not necessarily an attestation. For ease of presentation we sometimes speak of the issuance or presentation of (PID) attestations. Typically, the PID are the first data issued to the EUDI Wallet.

The update of the eIDAS regulation stipulates a "Common Interface" between the EUDI Wallet, attestation providers and relying parties. This interface shall be further specified in implementation regulations and standards such as ISO 23220 and OpenID for Verifiable Credentials [34]. The EUDI Wallet is described in more detail in the Architecture and Reference Framework [1].
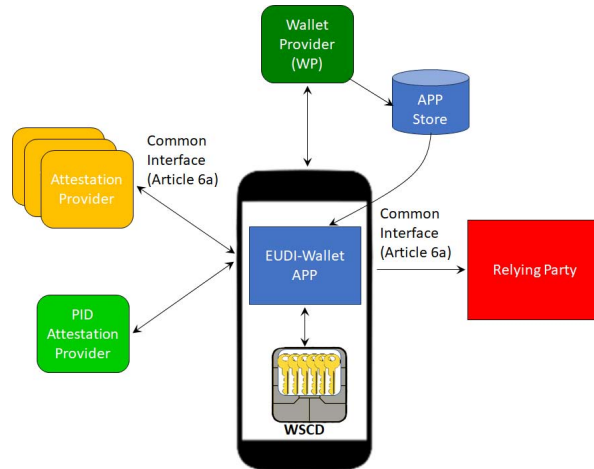
Figure 1. EUDI Wallet overview

Although [13] allows for other types of attestations, in the context of this document an attestation is functionally considered a PKI-certificate. That is, an attestation is a collection of user attributes supplemented with a public key and signed by a (PID) attestation provider. The user can prove attestation ownership to a party by electronically signing a message, e.g. a challenge, generated by the party. In this paper focus on the signature algorithms stipulated in the mobile driving licence standard [26], i.e., EdDSA [21,32], ECDSA [5,24,32] and "ECDH-MAC" signing as defined in [26] itself. Strictly speaking ECDH-MAC is not a digital signature scheme as it lacks the non-repudiation property which actually is the reason it is part of the mobile driving licence standard. For convenience we have also specified the generation and verification of ECDSA and ECDH-MAC signatures in Annex E in Algorithms 5 - 8.

During issuance of the attestation to a user, the attestation provider performs identity proofing of the user, ensuring that the issued attributes belong to the user. Typically, the PID issuance could be based on a national eID-card from whereas other attestations could be issued based on the PID itself. A fundamental security property of an attestation is that during attestation presence, the EUDI Wallet user can cryptographically prove holdership to relying parties. This is accomplished by proving possession of the private key of which the corresponding public key is bound by the attestation issuer during attestation issuance.

The attestation public/private keypair are managed by the EUDI Wallet in a component called the *Wallet Secure Cryptographic Device (WSCD)* in the [1], cf. Figure 1. The WSCD can perform basic key-management operations (e.g., generate signing public/private keypair, sign with certain private key, delete key), whereby keys are managed and controlled in a secure fashion. This includes that private keys managed in the WSCD (indicates as keys in Figure 1) cannot be exported in plaintext from the WCSD and that they are protected against other

type of attacks on the WSCD. This property is fundamental for EUDI Wallet security as otherwise attestations could for instance be cloned.

Both the issuance and presentation of (PID) attestations need to conform to [14] which includes that the EUDI Wallet authentication mechanism should protect against attackers with a "high attack potential". Although not formalized, the general interpretation of this protection is that this implies that at least the WSCD residing in the EUDI Wallet is Common Criteria certified [25] at assurance level EAL4+.

Next to security, also privacy protection plays an important role in the EUDI Wallet. Several articles of the eIDAS regulation update [13] stipulate specific privacy requirements the EUDI Wallet must adhere to. For instance, Article 12 stipulates adherence to the privacy by design principle. Also, adherence to data minimisation is stipulated in the preamble of the eIDAS regulation. Four WSCD architectures are envisioned:

External ("Smart Card") The WSCD here is a chip external to the mobile device, e.g., a GlobalPlatform [17] based Javacard Secure Element.

Internal (eUICC, eSIM, eSE) The WSCD here is based on a dedicated, internal chip integrated in the mobile device, e.g. eUICC, supporting Javacard based on GlobalPlatform.

Remote HSM The WSCD here is based on a Hardware Security Module (HSM) at the Wallet Provider and where the WSCA takes the form of a Wallet Provider Trusted Service Application interacting with the HSM.

Internal Native The WSCD is solely based on the native cryptographic hardware of a mobile device (Apple iOS/Secure Enclave and Android/Hardware Backed Keystore or Strongbox). In this situation it is hardest meeting the high EUDI Wallet security requirements.

This document

In this document we specify a wallet provider issued attestation called *Wallet Trust Evidence* (WTE) and three related specific WSCD instructions. These allow the EUDI Wallet providing verifiable assurance to third parties (issuers, relying parties) that attestation private keys are not only bound to a conformant WSCD but also that they are bound to the *same* WSCD. This allows the EUDI Wallet meeting eIDAS Level of Assurance "high" as well as operating in a privacy friendly manner. The instructions specified cater for convenient implementation in all envisioned EUDI Wallet architectures including those based on a GlobalPlatform [17] based Secure Element [17] such as an eID-card or an eSIM. By their simplicity, the three instructions also allow for convenient Common Criteria certification [25]. This document is a further refinement and concretisation of the EPIC-09 result [29] developed in a cooperation between the NI-Scy and the eIDAS expert group. The focus of this document are WSCD implementations allowing for trusted logic (e.g. Javacard), i.e. the first two WSCD architectures. The first instruction ("key-attestation") is quite common practice for cryptographic hardware and the second is easily implemented in trusted logic. Therefore this document focusses on the cryptographic specification of the third WSCD instruction, i.e., the generation of a proof of Association.

Document outline
- Section 2 starts with a security problem description from which we derive three fundamental WSCD security requirements leading us to three specific WSCD instructions. The last instruction is the generation of a proof of association.
- Section 3 is the core of the document. It proposes a proof of association based the on a Schnorr non-interactive zero-knowledge proof.
- In Section 4 we provide further implementation notes including an indicating that the WTE construction and the proof of association proposed in this document can also be implemented in the context of anonymous credentials such as based on BBS+ [2,7] or Idemix [8].
- Section 5 contains the references used in this document.
- Annex A is informative and contains an illustration of Android StrongBox key-attestation.
- Annexes B, C and D are informative illustrations of the use of the three WSCD instruction in three use-cases.
- Annex E contains the cryptographic and mathematical background used in Section 3.
- Annex G contains three example applications of Proposition 3.6, two of which avoid the use of raw ECDSA signing.
- Annex H contains a proposal for a proof of association specification in ASN.1 format.

2 Security problem description

In Section 2.1 we first heuristically derive three fundamental WSCD security requirements by analysing the following three common EUDI Wallet use cases:
1. Attestation issuance (in general).
2. Issuance of another attestation based on the PID.
3. Presentation of multiple attestations to a relying party.

These security requirements then lead us in Section 2.2 to three fundamental instructions a WSCD should support. We motivate that the first two instructions are either common practice or easily implemented in cryptographic hardware supporting trusted logic such as GlobalPlatform [17]. In Section 3 we propose cryptographic specifications implementing the third instruction.

2.1 Three fundamental WSCD security requirements
Historically, high assurance (e.g. qualified) PKI-certificates are based on smart cards, i.e. cryptographic hardware, holding private cryptographic keys in a non-exportable fashion. The public/private key generation in the smartcard is under full control of the certificate issuer taking place on the issuer premise. In this way, the issuer is assured that the public key he binds in the certificate has its private key residing in the smartcard and not for instance in a software based keystore.

The nature of the EUDI Wallet completely changes this setup. Here the public/private key generation takes place in the WSCD based on an instruction from the Wallet App which is under control of the user. Without further arrangements, an attestation provider has no assurance that the public key he binds in the attestation has its private key bound to the cryptographic hardware, i.e. resides in it. Indeed, a fraudulent user or malicious software running on the user mobile device could manipulate the key generation instruction from the Wallet App to the WSCD and replace it with a software based key generation or by a key generation instruction to a EUDI Wallet/WSCD of another user. In the first abuse case the attestation private key would be copyable making the attestation clonable. In the second abuse case the other user could present the attestation of the first user, e.g. a diploma, as being hers. Modern mobile operation systems support something called "Mobile App attestation" allowing parties (like an issuer in our context) to assess that a mobile application or the device is not tampered with ("rooted" or "jailbroken"). Both Apple's devicecheck [9] and Google's Safetynet [10] provide for Mobile App attestation. However, as Mobile App attestation is provided by the operation system, i.e. software, it has a large attack surface[1] and it commonly accepted amongst experts that it can never protect against a high attack potential as the eIDAS regulation requires.

In other words, an attestation provider cannot simply trust the EUDI Wallet, even when it is APP-attested, that the public key sent during attestation issuance is indeed bound to a WSCD. This brings us to the first fundamental WSCD security requirement:

ICW (InCertWSCD) During issuance the attestation provider must be able to verify that the (PID) attestation public key sent by the EUDI Wallet to be included in the attestation by the issuer, is bound to a certified WSCD. That is, that the corresponding private key resides in a certified WSCD.

If the attestation provides assurance that the attestation private key is bound to a WSCD, then the WSCD certification shall be such that it implies that the attestation private key cannot be exported out of the WSCD. Requirement **ICW** is well-known and is commonly addressed by a technique called *key-attestation*, a somewhat overloaded term in our context given the use of the term "attestation" in the eIDAS regulation and in Mobile App attestation.

A simple key-attestation implementation is that the cryptographic hardware supplier places a certified signing key ("attestation key") in the hardware during its production. That is, the attestation private key is placed in the hardware and the attestation public key is bound in an *attestation signer certificate* that is part of trusted certificate chain. During key generation, the hardware not only returns the newly generated public key but also a *key-attestation certificate* on this key signed by the attestation key. This certificate can also include a challenge of a relying party ensuring freshness of the generated key. This technique is supported by GlobalPlatform [17] but also by the Android Keystore for both its Hardware Backed Keystore (TEE based) and its EAL4+ certified StrongBox

---

[1] Compare https://github.com/kdrag0n/safetynet-fix

chip, cf. [38]. In Figure 6 of Annex A this setup and the introduced terminology is shown based a StrongBox chip of a Google Pixel 3a. We also remark that this key-attestation certificate can also be fulfilled by the Secure Area Attestation Object (SAAO) specified in the emerging ISO 23220-3 standard [27]. In the terminology of ISO 23220-3 the attested key is called the "SA-Attestation PublicKey".

Now assume that the EUDI Wallet of a user has been issued a PID meeting property **ICW**, i.e. the PID provider could verify that the PID private key is bound to a certified WSCD. Suppose that the EUDI Wallet user wants to have issued two additional attestations based on her PID:

- an "adult" attestation proving that she is over 18 years old, and
- a "photo" (facial image) attestation from a trusted photographer.

The issuance of both attestations starts with an identification of the user based on her PID. During the attestation issuance the issuer does not only need assurance that the diploma/photo public keys sent by the EUDI Wallet are bound to a certified WSCD, e.g. Requirement **ICW**, but also that the public keys are bound to the *same* WSCD as the PID is. Indeed, without such assurance a fraudulent user or malicious software running on the user mobile device could manipulate the key generation instruction from the Wallet App to the WSCD and replace it with a key generation instruction to a EUDI Wallet/WSCD of another user. In this abuse case the other user could then present the attestations of the first user, as being hers. This brings us to the second fundamental WSCD security requirement.

SW1 (SameWSCD1) During attestation issuance an issuer must be able to verify that the attestation public key sent by the EUDI Wallet to be included in the attestation is not only bound to a certified WSCD but is also bound to the *same* certified WSCD as the PID public key is.

The third fundamental WSCD security requirement is the counterpart of the second fundamental WSCD security requirement for relying parties. Assume that the EUDI Wallet user has been issued the adult and the photo attestations as discussed above and that the issuers have been assured of Requirement **SW1**, i.e. that the adult/photo private keys reside in the same WSCD as the PID private key. Now suppose that the user wants to present her adult and photo attestation in a shop, e.g. as part of buying cigarettes or alcohol. Then the shop would also need to be assured that adult/photo public key correspond to one and the same user, i.e. that they are bound to the *same* WSCD as the user PID is. Indeed, without such assurance a fraudulent user or malicious software running on the user mobile device could present the photo attestation of one user and the adult attestation of another user. This brings us to the third fundamental WSCD security requirement.

SW2 (SameWSCD2) During presentation of multiple attestations a relying party must be able to verify that public keys in different attestations are not

only bound to a certified WSCD but are also bound to the same certified WSCD as the PID public key is.

Claim-based binding

One can also base a EUDI Wallet on security requirement **ICW** only whereby avoiding the necessity of security requirements **SW1** and **SW2** by a technique called "claim-based binding", cf. [34]. In this context only the PID contains a WSCD bound public key, i.e. based on security requirement **ICW**. All other attestations are linked to the PID by letting the attestation issuer copy all PID data in the other attestations as well. So, during the issuance of, say, a diploma attestation the user presents her PID to the issuer. After the appropriate verifications the diploma issuer lets all PID data be part of the diploma attestation itself. During diploma attestation presentation the user also presents her PID and the diploma attestation. By verifying the common PID data the relying party can determine the diploma belongs to the PID user. One does not need to copy all PID data to the other attestations but only a part that is directly identifying, e.g. a social security number. The PID issuer could also place specifically designated data ('linking attributes') in the PID for this purpose.

In the discussed claim-based setup the non-PID attestations do not contain a public key and security requirements **SW1** and **SW2** are met in an empty way. Note that such non-PID attestations are not in scope of this document as we assumed that all attestations contain their own public key. If in the discussed claim-based setup the non-PID attestation would contain their own public key, then security requirements **SW1** and **SW2** are not met as the second abuse case discussed above would apply. Alternatively we could also reuse the PID public key in all attestations but that would give linkability issues (the public key becomes a "supercookie") and conflicts with several key management good practices, cf. [33]. One of the conflicting good practices is that cryptographic keys should only have one purpose. As an EUDI Wallet illustration for this: a signature that verifies with the PID public key would then also verify with the (diploma) attestation public key (as it is the same public key). This can give rise to a dispute between user and a relying party on whether the user authenticated with her PID or with the diploma attestation.

Although claim-based binding can be a valuable way of binding attestations to the user, its use of shared linking data in claim-based binding introduces privacy challenges related to linkability. The approach introduced in this document is based on binding (PID) attestations cryptographically which has less privacy and security challenges. Wallet implementations could use a mix of both techniques.

2.2 Three fundamental WSCD instructions

We first discuss a basic method to meet all three WSCD security requirements from Section 2.1 and motivate that method this is not suitable on ground of insufficient privacy protection and complexity.

In the basic method the WSCD has the ability to generate attested keys as indicated in Section 2.1. Each newly generated (PID) attestation key is part

of an attestation certificate that is part of a trusted certificate chain, cf. Figure 6 of Annex A. The key-attestation certificate holds a supplier statement on the WSCD (eIDAS conformity) certification. From this statement an issuer can also be assured that the attestation private key is WSCD bound and is properly managed there. This would allow for adherence with security requirement **ICW**. Additionally, to allow issuers and relying parties verification that two attestation private keys are bound to the same WSCD, the EUDI Wallet sends along a common attestation signer certificate. This would give adherence to security requirements **SW1** and **SW2**.

The basic method has the following issues:
(A) It conflicts with the EUDI Wallet privacy by design and data minimisation principles stipulated in the eIDAS regulation update [13]. Indeed, the common attestation signer certificate allows linking the user over various issuers and relying parties.
(B) The basic method uses that key-attestations uniquely identify the WSCD, e.g., to a serial number of the WSCD, which is avoided in modern key-attestation methods, e.g. used by Android [11] or [16] exactly to avoid the linking issue indicated in the previous point. That is, the basic method does not work and in fact conflicts with current privacy friendly key-attestation methods.
(C) The attestation signer certificate might contain WSCD information, e.g. serial numbers, date of production that is unnecessary for the issuers and relying parties. Such information might allow for further user linking or even allow for identification of the EUDI Wallet user. To illustrate, another mobile application can also use the cryptographic hardware the WSCD is based on. Then the other mobile application can link the user through the attestation signer certificate. Actually, the other mobile application might be specifically developed to facilitate this linking and designed such that users are tempted to install it.
(D) It burdens issuers and relying parties as they would need to have access to all WSCD supplier trust chains and to be able verify if the WSCD statement in the attestation certificate is adequate for use in an eIDAS EUDI Wallet.

To address issues (C) and (D) point we start by introducing the *Wallet Trust Evidence (WTE))*. The WTE is an attestation itself issued by the Wallet Provider based on a WSCD specific key-attestation certificate. During WTE issuance, the EUDI Wallet generates an attested public/private key pair ($Pub_{\mathrm{WTE}}, Priv_{\mathrm{WTE}}$) on request of the wallet provider, i.e. a limited version of Requirement **ICW**. The wallet provider verifies the key-attestation certificate, the WSCD conformity statement therein and the trust chain. Additionally, the wallet provider requires the wallet to prove possession of the private key $Priv_{\mathrm{WTE}}$. If these verifications are successful, then the Wallet Provider issues the WTE attestation on the public key Pub_{WTE}. The WTE only contains minimal data; essentially nothing more than that the WSCD is eIDAS-conformant. Also, the WTE public key Pub_{WTE}

uniquely identifies the WTE and the WSCD it refers to. As indicated in Section 2.1, such key-attestation certificate are supported by GlobalPlatform, the Android Keystore and in the emerging ISO 23220-3 standard ("SAAO").

For the WTE construction, we do require the WSCD to support general key-attestation as in the basic method but only its use it as part of WTE issuance. We formalize this as a first WSCD instruction.

WSCD-Instruction 1 Generate attested WTE-key
Input: key properties, challenge C
Output: WTE public key Pub_{WTE},
　　　　 WTE Key-Attestation Certificate $\mathcal{K}$ containing Pub_{WTE}, C

　Return Pub_{WTE}, $\mathcal{K}$ 　　　　　　　　　　　　　　// WSCD specific

---

Replacing the key-attestation certificate with the WTE addresses issues (C) and (D) but not issues (A) and (B). For this we introduce the new mechanism of *key association*. This mechanism avoids that each newly generated attestation key is issued a new key-attestation certificate but instead builds further on the WTE itself. Key association allows the EUDI wallet:

(a) to generate a new key in the WSCD that is *associated* to the WTE public key, and

(b) to cryptographically prove this association to issuers and relying parties.

The WSCD trusted logic then ensures that the new key resides in the *same* WSCD as is referred to by the WTE, or more specific referred to by the WTE public key Pub_{WTE}. If a public key Pub is associated to a WTE public key Pub_{WTE}, we will also allow the generation of new keypair that is associated to Pub and then by inheritance also to the WTE public key Pub_{WTE}. In this way we mathematically model association as a transitive relation.

The association mechanism is based on two additional WSCD instructions formalized below. Instruction #2 allows the generation of a new key associated with a given WTE key and Instruction #3 provides a proof of association for two keys that are associated. To support association, the WSCD maintains a secure association registration. One can think of an internal Association File holding multiple lines each of which corresponds to the associated keys in the WSCD. Instruction #1 (Generate attested WTE-key) then creates a new line in the Association File holding a reference to the newly generated WTE-key.

WSCD-Instruction 2 Generate key associated to WTE
Input: Reference Rf_{WTE} to WTE key, key properties
Output: Generated public key Pub associated to WTE key

1: Look up Association File line of WTE key Rf_{WTE} 　　// error on failure
2: Generate new keypair $Pub, Priv$ with requested key properties
3: Write entry in Association File line reflecting that public key Pub
　 is associated with WTE key
4: Return public key Pub

WSCD-Instruction 3 Generate Proof of Association
Input: Associated public keys Pub_1, Pub_2
Output: $\text{PoA}(Pub_1, Pub_2)$

1: `Verify in Association File that public keys` Pub_1, Pub_2 `are associated`
   `// error on failure`
2: `Generate proof of association PoA(`Pub_1, Pub_2`)`   `// WSCD specific`
3: `Return proof of association PoA(`Pub_1, Pub_2`)`

In Annexes B, C and D we show how the WTE and the three WSCD instructions provide for the three fundamental WSCD security requirements formulated in Section 2.1. In the next Section 3 we propose a cryptographic algorithm for the generation of a proof of association, i.e. WSCD Instruction #3.

Draft ISO 23220-3 approach to WTE
We briefly discuss the approach in Annex C.6.5 of the emerging ISO 23220-3 standard [27] and compare it with the WTE/PoA approach. In the ISO 23220-3 approach the "mdoc app provider", i.e. the wallet provider in our context, re-issues individually attested keys in the form of a public key array as part of the issuing process. This approach implies that the wallet provider always observes all attestation public keys as he puts them in the public key array. This can be considered conflicting with [13, Article 5a(14)] and the GDPR data minimalization principle. This issue is avoided in the WTE/PoA approach; the wallet provider only observes the WTEs but not the attestation keys. Also, as the ISO 23220-3 approach is dedicated to one issuer only, one cannot provide for security objective SW2, cryptographically binding different attestations during presentation. Finally, as indicated as Issue (B) on page 8 this approach does not work with modern, privacy friendly key-attestation.

3 A proof of association proposal

In this section we specify a cryptographic method for the generation of a proof of association. The cryptographic and mathematical background and notation this section builds upon is placed in Annex E. In this proposal we only associate public keys that are based on the same elliptic curve group represented in additive notation as $\mathbb{G} = (\langle G \rangle, +)$ of order q generated by a *base point* (generator) G. That is, we can associate two public keys that are based on the same elliptic curve, e.g. the NIST P-256 curve or the brainpoolP256r1 curve. However, we cannot associate a NIST P-256 based public key with a brainpoolP256r1 based public, nor can we associate RSA public keys. We think that this drawback is acceptable in practice.

The cryptographic heuristic behind the association proposal is as follows. The context is a WSCD that supports WSCD-Instruction 1 as discussed in Section 2.1. Let $W = w \cdot G$ be a *certified WTE public key* based on WSCD-Instruction 1. That is, the key W is bound to an attestation/certificate verifiably issued by the wallet provider. From this attestation/certificate, parties can infer that the private key w is managed in a WSCD that is certified to be compliant with the updated eIDAS regulation [13]. What this means will be clarified later, but at

this moment we assume that this at least includes that the WSCD adheres to good practice key management and also that it supports the proof of association trusted logic (which follows). For ease of reference we formulate this as a definition.

Definition 3.1 *A certified WSCD is compliant with the updated eIDAS regulation [13], adheres to good practice key management, supports WSCD-Instruction 1 and also supports the proof of association trusted logic.*

Now suppose that $P = p{\cdot}G$ is a public key bound to the same WSCD as the WTE, i.e. the private key p is managed in the same WSCD as w is. The key idea is that when the WSCD has registered that public key P is associated to the WTE public key, the WSCD trusted logic will allow the computation of the *association* key $z = p{\cdot}w^{-1} \bmod q$. The proof of association is based on this association key. It follows that

$$z{\cdot}W = p{\cdot}w^{-1}{\cdot}W = p{\cdot}w^{-1}{\cdot}w{\cdot}G = p{\cdot}G = P.$$

That is, the key P can be considered a public key with respect to generator W with private key z, i.e. the association key. Now suppose that a party can prove to a verifying party that it has *full control* over both private keys z and p, i.e., can do arbitrary mathematical operations with these. Then this party can also compute $p{\cdot}z^{-1} = w$. That is, the party has full control over the private key w too. By construction this means that this party must be the WSCD, as that is the only party having full control over key w by construction.

Following this heuristic brings us to the following definition of proof of association. The definition encompasses association between general public keys and is thus broader than only between a WTE public key and an attestation public key as in the heuristic. We formally define that a public key is associated to itself, but we do not need a proof of association to prove this.

Definition 3.2 *We use the context described above. A proof of association (PoA) between different public keys P_1 and P_2 conveys to the verifier(issuer, relying party) that the party that generated the PoA has full control over the association key $z = p_2{\cdot}p_1^{-1}$, i.e., can do arbitrary mathematical operations with it.*

In Algorithm 1 we have specified a proposal for the generation of a PoA based on a Schnorr non-interactive zero-knowledge proof using the Fiat-Shamir heuristic [15] similar to RFC 8235 [22].

Algorithm 1 Proof of Association (PoA) generation
Input: optional verifier challenge C (byte array), two associated public keys $P_1 = p_1 \cdot G$, $P_2 = p_2 \cdot G$ with respective private keys p_1, p_2.
Output: $\text{PoA} = \{P_1, P_2, C, (r, s)\}$.

```
1: If P₁ = P₂ return error                    // P₁, P₂ need to be different
2: Compute association key z = p₂·p₁⁻¹ mod q.              // note P₂ = z·P₁
3: Convert public keys P₁ and P₂ to byte arrays P̄₁, P̄₂ respectively
4: Select random k ∈ {1,...,q − 1}.
5: Compute P₁′ = k·P₁ = (x,y) and convert to byte array P̄₁′.    // commitment
6: Compute byte array H(P̄₁′||P̄₁||P̄₂||C) and convert it to an integer r.
7: If r mod q = 0 then go to Line 4.
8: Compute s = k + r · z mod q.
9: If s = 0 then go to Line 4.
10: Return PoA = {P₁, P₂, C, (r, s)}.
```

The following algorithm specifies the verification of a PoA.

Algorithm 2 Proof of Association (PoA) verification
Input: WTE, $\text{PoA} = \{P_1, P_2, C, (r, s)\}$
Output: Acceptance of rejection of the PoA.

```
1: Verify P₁ ≠ P₂ on failure Return Error    // P₁, P₂ need to be different
2: Verify the input, including that
     r ∈ {1, 2^{8·|q|} − 1} and s ∈ {1, q − 1}, on failure Return False.
3: Convert public keys P₁ and P₂ to byte arrays P̄₁, P̄₂ respectively
4: Compute Q = s · P₁ − r · P₂ if Q = O Return False.
5: Convert Q to byte array Q̄.                          // i.e. of size 2·|p|
6: Compute byte array H(Q̄||P̄₁||P̄₂||C) and convert it to an integer v.
7: If v = r accept the PoA otherwise reject it.
```

The following proposition proves that the proof of attestation generated by Algorithm 1 meets the requirements.

Proposition 3.3 *The PoA generated by Algorithm 1 will be accepted by Algorithm 2 and meets Definition 3.2.*

Proof: For the first part of the proposition, let $\{P_1, P_2, C, (r, s)\}$ be a PoA generated by Algorithm 1. Then the following equalities hold for the point Q appearing in Line 4 of Algorithm 2:

$$Q = s \cdot P_1 - r \cdot P_2 = (k + r \cdot z) \cdot P_1 - r \cdot P_2 = k \cdot P_1 + r \cdot (z \cdot P_1 - P_2) = k \cdot P_1 \quad (1)$$

The first equality is Line 4 of Algorithm 2, the second equality follows from the construction of s in Line 8 of Algorithm 1, the third equality is straightforward and the last equality follows as $z \cdot P_1 = P_2$ by the definition of z in Line 2 of Algorithm 1. From Equality (1) it follows that point Q is equal to point P_1' appearing in Line 5 of Algorithm 1. It now follows that the hash inputs in Line 6 of both Algorithms 1 and 2 are equal and so are their outputs, i.e. $r = v$. It follows that Algorithm 2 accepts the PoA.

That the PoA generated by Algorithm 1 meets Definition 3.2 follows from the soundness of the Schnorr non-interactive zero-knowledge proof. Compare [36, Theorem 9.1]. □

We make some further notes on Algorithms 1 and 2:

1. As the Schnorr non-interactive zero-knowledge proof operates in zero-knowledge the PoA based on it can be securely used in combination with various attestation signing algorithms. Even simultaneous use is possible such as indicated in ISO 18013-5 [26] that allows a signing key to be used for EdDSA, ECDSA and ECDH-MAC signing.
2. The optional challenge choice in Algorithm 1 allows to make the PoA interactive allowing a challenge of a verifier, e.g. a (PID) issuer in the EUDI Wallet context, to be included in the PoA. Compare the notes in Section 4.2.
3. One can naturally extend Algorithm 1 for an arbitrary number of associated public keys by returning the pairwise proofs of association, e.g. $\mathrm{PoA}[P_1, P_2, P_3]$ consists of $\mathrm{PoA}[P_1, P_2]$ and $\mathrm{PoA}[P_2, P_3]$. The order of the public keys is irrelevant.
4. Generation of a PoA (Algorithm 1)and verification of a PoA (Algorithm 2) closely resembles the generation and a verification of an ECSDSA (Elliptic Curve Schnorr Digital Signature Algorithm) signature [5,24,35]. This means that if a platform supports ECSDSA then the proof of association is easily implemented.

We require that a certified WSCD only generates a proof of association for public keys that are bound to it and that are associated to the same WTE public key. Conversely, if we have two public keys P_1, P_2 that are known to be bound to certified WSCDs and for which a proof of association exists then the public keys must be bound to the same certified WSCD. Indeed, if they were bound to different certified WSCDs, then the party generating the proof of association would be able to solve the Discrete Logarithm problem with respect to public key P_1 generated by the first WSCD and public key P_2 generated by the second WSCD. This is not possible as public keys P_1, P_2 are randomly generated as certified WSCDs adhere to good key management practices (Definition 3.1). For easy reference, we formulate this result as a proposition.

Proposition 3.4 *If two associated public keys are known to be bound to certified WSCDs, then they must be bound to the same certified WSCD and associated to the same WTE public key.*

To solve the security problem described in Section 2 we need to show that this proof of association implementation coincides with the WSCD notion of association for which we need to prove the following fundamental result.

If a verifier is provided two proofs:

1. a proof of association passing Algorithm 2 between public key $P = p \cdot G$ and a certified WTE public key $W = w \cdot G$, and

2. a "suitable" proof of possession of the private key p,

then the public key P is bound to the WSCD the WTE refers to and is associated to the WTE public key W.

Note that the part "and is associated to the WTE public key W" allows for recursion whereby the public key P can take the role of W. Metaphorically this resembles the folktale "Swan, stick on" whereby the WTE public key is the swan and the public keys are the people recursively sticking to the swan. What "suitable" means depends on the attestation signature algorithm used; we distinguish EdDSA (or more generally "sound" signature algorithms), ECDSA and ECDH-MAC. The corresponding results are respectively Propositions 3.5, 3.6 and 3.8.

A proof of association by itself does not provide any guarantee on the association by the WSCD between the WTE public key W and public key P. Indeed, the wallet user (or an attacker) can choose any association key z, compute $P = z{\cdot}W$ and generate a proof of association following Algorithm 1. That is why the above heuristic also requires that the verifier was also provided a proof that the WSCD have full control over the private key p. In the situation where the user/attacker chooses the association key z itself this private key is equal to $z{\cdot}w$ to which the user/attacker has no full access.

This leads us to the question how the wallet can convey to the verifier (issuer, relying party) it has full access to the private key p. One might expect that by letting the wallet digitally sign a challenge of the verifier, i.e. a proof of possession, would cater for that. This actually holds for the EdDSA signature algorithm as explained in the proof of Proposition 3.5. However, it does not hold for the ECDSA and ECDH-MAC signature algorithms: there the user can sign with the private key p with only having partial access to it, cf. Algorithms 3 and 4. We will explain that this can be considered a feature too as it allows for an easily implementable WSCD supporting association.

The following proposition shows that the proof of association Algorithm 1 in combination with EdDSA based attestation keys is meeting the PoA requirements.

Proposition 3.5 *We use the context described above whereby the public key $P = p{\cdot}G$ with private key p is an EdDSA keypair. Suppose a party provides to a verifier a proof of association that passes Algorithm 2 and a proof of possession of private key p consisting of EdDSA signature on a random challenge generated by the verifier. Then the public key P is bound to the WSCD the WTE refers to and is associated to the WTE public key W.*

Proof: Suppose that public key P is not managed in the WSCD the WTE refers to. This means that the proof of association is not generated by the WSCD the WTE refers to. As the PoA generated by Algorithm 1 meets Definition 3.1 (see the notes following Algorithm 1) it follows there is another party than the WSCD having full control over the key z for which it holds $P = z{\cdot}W$. As public key P is not managed in the WSCD it follows that the proof of possession of private key p is also not generated by the WSCD the WTE refers to but by a second

party, perhaps the first and second party are the same. The EdDSA signature algorithm is (like the Proof of Assocation) based on a Schnorr non-interactive zero-knowledge proof and is thus sound. Compare the notes following Algorithm 1. So it follows that the second party has full control over the key p. This means that if the first and second party work together they can compute $p \cdot z^{-1} = w$ which contradicts that private key w is stored in the WSCD in a non-extractable manner. We conclude that public key P is managed in the WSCD the WTE refers to.

Now suppose that the proof of association was not generated by the WSCD. As before this means there is another party than the WSCD having full control over the key z for which it holds $P = z \cdot W$. As the certified WSCD adheres to good practice key management, public keys W, P are randomly generated. This means that the other party is able to solve the Discrete Logarithm problem of P with respect to W, which is not possible. We conclude that the proof of association was generated by the WSCD and that public key P is associated to public key W. $\qquad\square$

The practical application of Proposition 3.5 is during the issuance of an attestation on the public key P. From this attestation parties can infer that public key P is bound to a certified WSCD and associated to the WTE public key. This allows the proof of association to be used recursively like in the folktale "Swan, stick on" mentioned above. This also means that further proof of association applications involving P can be based on Proposition 3.4.

We now show that an ECDSA proof of possession signature does not prove that the signer has full control over the private key, i.e., can do arbitrary mathematical operations with it. We work in the same context as before: a wallet user has generated an association key z itself and computed the corresponding WTE associated key $P = z \cdot W$. As the user has access to the association key, he can also generate a proof of association using Algorithm 1. The following algorithm from [37] shows the user is also able to generate ECDSA signatures on messages with the private key corresponding to P, i.e. $z \cdot w$, provided the WTE key w allows for raw signing. Raw signing is the generation by the WSCD of a signature directly on basis of a hash value input, i.e., without the WSCD deploying the hash operation. In the remarks following Algorithm 5 in Annex E we have provided background on this and its common use in practice. We show in Proposition 3.6 that by precluding ECDSA raw signing by the WTE key, it can be proven that this ability is no longer possible.

Algorithm 3 Split-ECDSA (SECDSA) signature generation

Input: message M, WTE private key $w \in \mathbb{F}_q^*$ supporting ECDSA raw signing, association key $z \in \mathbb{F}_q^*$

Output signature (r, s).

1: Compute $\mathcal{H}(M)$ and convert this to an integer e.
2: Compute $e' = z^{-1}{\cdot}e \bmod q$
3: Select random $k \in \{1, ..., q-1\}$
4: Compute $kG = (x, y)$ and convert x to integer $\bar{x}$
5: Compute $r = \bar{x} \bmod q$. If $r = 0$ go to Line 3
6: If $r \bmod q = 0$ then go to Line 3
7: Compute $s = k^{-1}(e' + w{\cdot}r) \bmod q$. If $s = 0$ go to Line 3
8: Compute $s' = z{\cdot}s \bmod q$
9: Return (r, s')

It is shown in [37, Proposition 3.1] that Algorithm 3 returns a valid ECDSA signature corresponding to public key P. Note that the pair (r, s) appearing in Lines 3-7 of Algorithm 3 is just a ECDSA raw signature on e' with respect to the WTE private key w. Compare the remarks following Algorithm 5 describing ECDSA. This means that Lines 3-7 simply consist of calling the WSCD to generate a raw signature on e' with respect to the WTE private key w. In Line 2 the input of the hardware generated signature is modified using association key as is the outputted signature itself in Line 8. From [37, Proposition 3.2] it follows that forging an ECDSA signature for private key p is equivalent to forging an ECDSA signature for private key w.

Based on Algorithm 3 one can envision an ECDSA based distributed WSCD. This wallet is based on only one (WTE) ECDSA hardware bound pubic key W under PIN access control. All attestation keys are then constructed as $P = z{\cdot}W$ with association keys w managed in the wallet mobile application. Compare Figure 2. This model is not further explored in this document.

The following proposition shows that the proof of association generated by Algorithm 1 in combination with ECDSA based attestation keys is meeting the requirements provided the WTE private key does not provide for ECDSA raw signing while the attestation keys do support this. That also means that by precluding raw signing by the WTE key, the distributed WSCD is no longer possible making it an option controllable by the WSCD configuration of the WTE key. We remark that the WTE Key-Attestation Certificate produced by WSCD Instruction 1 must convey to the wallet provider that the WTE key only supports regular ECDSA signing where the WSCD performs the hash operation. This obviously implies that the WTE key does not support raw signing. Indeed, an attacker able to sign a chosen hashvalue not implicitly requested in a regular ECDSA signing request would be able to break ECDSA signing.

Proposition 3.6 *We use the context described above whereby the public key $P = p{\cdot}G$ with private key p is an ECDSA key. The WTE private key w only supports ECDSA signing where the WSCD performs the hash operation, i.e. does not support ECDSA raw signing. Suppose a party provides a verifier a proof of association that passes Algorithm 2 and a proof that it can generate ECDSA raw*

signatures based on private key p. Then public key P is bound to the WSCD the WTE refers to and is associated to the WTE public key W.

Proof: Assume that the proof of association was generated by another party than the WSCD the WTE refers to. As the PoA generated by Algorithm 1 meets Definition 3.1 (see the notes following Algorithm 1) it follows that this party has full control over the key z for which it holds $P = z \cdot W$, i.e. $p = z \cdot w$. As private key p allows for raw signing, it follows that also private key w allows for raw signing by Algorithm 3. This contradicts that the WTE private key w does not support raw signing. This means that the proof of association is generated by the WSCD the WTE refers to and consequently that public key P is associated to public key W. $\square$

The practical application of Proposition 3.6 is during the issuance of an attestation on the public key P proposed by the EUDI Wallet. The issuer indicates in the attestation that public key P is bound to a certified WSCD, i.e. the result of Proposition 3.6. Further proof of association applications involving P by relying parties can then be based on Proposition 3.4.

Proposition 3.6 is kept generic allowing for various ways the EUDI Wallet can prove to the issuer that it can compute raw ECDSA signatures. The simplest way to prove this is, is letting the EUDI Wallet rawly sign a challenge generated by the attestation issuer with the private key p corresponding to the attestation public key P proposed by the EUDI Wallet. In this case the challenge is of the byte size of the hash function used, e.g. 32 bytes in case of P-256 based ECDSA. Note that this is required only during attestation issuance, i.e. only once. This is indicated in Figure 8 in Appendix G. As argued in the remarks following Algorithm 5 in Annex E, raw ECDSA signing is commonly use in practice so one can argue that rawly signing an issuer generated challenge once is not a security issue. Theoretically, there could exist an attack whereby a rogue issuer sends such a challenge whereby secret information leaks in the resulting signature. If desired this theoretical issue can be easily addressed by forcing the issuer to generate the challenge as the hash of another challenge and to prove that later on in the process. In this way the issuer only receives a regular ECDSA signature on a challenge which is common practice. That is, the issuer generates a challenge C, computes the hash $C' = \mathcal{H}(C)$ and requests a raw ECDSA signature on challenge C' with private key p. Through the WSCD the EUDI wallet computes this signature (r, s), computes $hSig = \mathcal{H}(r||s)$ and sends this to the issuer. The issuer send challenge C to the EUDI Wallet that verifies that $C' = \mathcal{H}(C)$. If this correct, the EUDI Wallet sends (r, s) to the issuer that verifies that $hSig = \mathcal{H}(r||s)$ and that (r, s) is a correct signature for public key P. In this way, the EUDI wallet can prove to the issuer it can rawly ECDSA sign with p without actually doing it. This setup is indicated in Figure 9 in Appendix G.

The following proposition provides for another method avoiding raw signatures and can also more be conveniently implemented; it is indicated in Figure 10 in Appendix G.

Proposition 3.7 *We use the context described above whereby the public key $P = p\cdot G$ with private key p is an ECDSA key. The WTE private key w only supports ECDSA signing where the WSCD performs the hash operation, i.e. does not support ECDSA raw signing. Suppose a party provides a verifier a proof of association that passes Algorithm 2 and an ECDSA signature (r, s) for public key P on the message M of form $C||P$ where C is a random challenge generated by the verifier. Then public key P is bound to the WSCD the WTE refers to and is associated to the WTE public key W.*

Proof: We argue as in Proposition 3.6. Assume that the proof of association was generated by another party than the WSCD the WTE refers to. As the PoA generated by Algorithm 1 meets Definition 3.1 (see the notes following Algorithm 1) it follows that this party has full control over the key z for which it holds $P = z\cdot W$, i.e. $p = z\cdot w$. By construction (r, s) is a raw signature on $\mathcal{H}(C||P) \bmod q$ for public key P. From Algorithm 3 it follows $z\cdot\mathcal{H}(C||P) \bmod q$ is a raw signature for $z^{-1}\cdot P = W$. As the WTE private key w only supports ECDSA signing where the WSCD performs the hash operation, there must be a message M' such that $\mathcal{H}(M') = z\cdot\mathcal{H}(C||P) \bmod q$. As $P = z\cdot W$ the hash value $\mathcal{H}(C||P)$ commits to z and by the challenge C, the hash value $\mathcal{H}(C||P)$ cannot be predicted by the party. That is, the message M' must be constructed after the z has been chosen. As further $z \neq 1 \bmod q$ the party cannot choose $M' = C||P$. It follows that the party is able to find pre-images for hash function $\mathcal{H}(.)$ which is not possible. □

Similary to ECDSA, we now show that an ECDH-MAC proof of possession signature does not prove that the signer has full control over the private key, i.e., can do arbitrary mathematical operations with it.. We work in the same context as before: a wallet user that has generated an association key z itself and the corresponding WTE associated key $P = z\cdot W$. As the user has access to the association key, he can generate a proof of association following Algorithm 1. The following algorithm shows the user is also able to generate ECDH-MAC signatures on messages/challenges with the private key corresponding to P, i.e. $z\cdot w$, provided the WTE key allows for full Diffie-Hellman, i.e. returning the full Diffie-Hellman key. See the remarks following Algorithm 8 in Annex E for background. By precluding that the WTE key supports full Diffie-Hellman, we prove in Proposition 3.8 that this is no longer possible.

Algorithm 4 Split-ECDH-MAC signature generation

Input: message M, WTE private key w supporting full Diffie-Hellman, ephemeral public key E, byte array SharedInfo, association key $z \in \mathbb{F}_q^*$,

Output byte array MAC

```
1: Verify that E ∈ ⟨G⟩, on error algorithm stops
2: Compute E' = z·E
3: Compute S_AB = w·E'                    // compute shared Diffie-Hellman key
4: Convert S_AB to byte array Z_AB
5: Compute K = HKDF(Z_AB, SharedInfo)              // derive MAC-key K
6: Compute HMAC = E_MAC(K, M).
7: Return MAC.
```

Observe that the shared Diffie-Hellman key in Step 3 for public key E' and private key w is equal to the shared Diffie-Hellman key for ephemeral public key E and private key $z \cdot w$. One can easily verify that Algorithm 4 returns an ECDH-MAC signature with respect to public key P. It is also easily verified that forging an ECDH-MAC attestation signature corresponding to public key P is equivalent to forging a WTE ECDH-MAC signature. That is, the security of ECDH-MAC attestation signing using Algorithm 4 is equivalent to ECDH-MAC WTE signing.

Based on Algorithm 4 one can envision an ECDH-MAC based distributed WSCD, similar to the ECDSA based distributed WSCD. This wallet is based on only one (WTE) ECDH-MAC hardware bound pubic key W under PIN access control that supports full Diffie-Hellman. Compare Figure 2. All attestation keys are then constructed as $P = z \cdot W$ with association keys w managed in the wallet mobile application. This model is not further explored in this document.
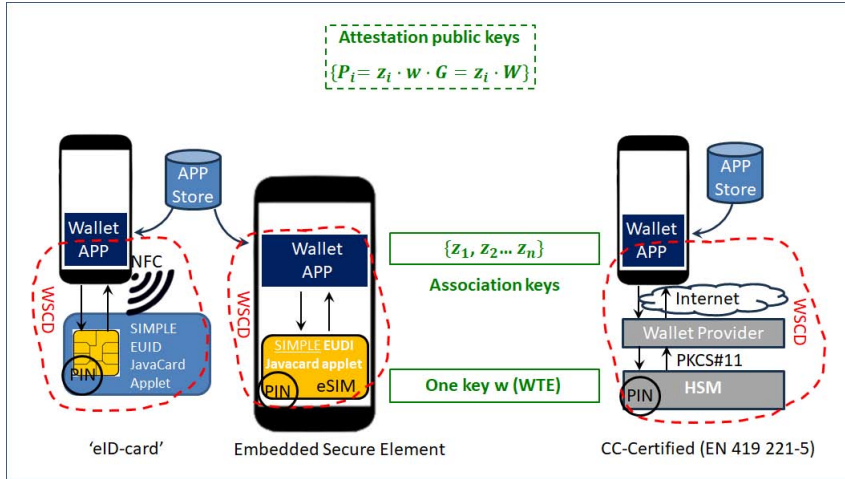

Figure 2. The distributed WSCD

The following proposition shows that the proof of association generated by Algorithm 1 in combination with ECDH-MAC based attestation keys is meet-

ing the requirements providing the WTE private key does not provide for full Diffie-Hellman but the attestation private keys do support that. The WTE Key-Attestation Certificate produced by WSCD Instruction 1 must provide assurance to the wallet provider that the WTE key only supports returning a derived key from the exchanged Diffie-Hellman key S_{AB} and does not provide for returning the full Diffie-Hellman key. This can be arranged by letting the WTE key only support returning the key derived from S_{AB} using the X9.63 Key Derivation Function [5, Section 4.3.3] or the HKDF algorithm [20] as in ISO 18013-5 [26]. As both derivation functions are based on hashing the exchanged Diffie-Hellman key S_{AB} it is guaranteed one cannot derive this key from the derived key.

Proposition 3.8 *We use the context described above whereby the public key $P = p{\cdot}G$ with private key p is an ECDH-MAC key. The WTE private key w does not support for full Diffie-Hellman. Suppose a party can provide to a verifier a proof of association that passes Algorithm 2 and a proof of possession of private key p consisting of the full Diffie-Hellman key based on an ephemeral public key E randomly generated by the verifier. Then the public key P is bound to this WSCD and is associated to the WTE public key W.*

Proof: Assume that the proof of association was generated by another party than the WSCD the WTE refers to. As the PoA generated by Algorithm 1 meets Definition 3.1 (see the notes following Algorithm 1) it follows that this party has full control over the key z for which it holds $P = z{\cdot}W$, i.e. $p = z{\cdot}w$. As private key p supports for full Diffie-Hellman so does private key w, cf. the observation after Algorithm 4. This contradicts that the WTE private key w does not support full Diffie-Hellman. This means that the proof of association is generated by the WSCD the WTE refers to and consequently that public key P is associated to public key W. □

The practical application of Proposition 3.8 is during the issuance of an attestation on the public key P proposed by the EUDI Wallet. The issuer indicates in the attestation that public key P is bound to a certified WSCD, i.e. the result of Proposition 3.8. Further proof of association applications involving P by relying parties can then be based on Proposition 3.4.

For proof simplicity we have chosen in Proposition 3.8 to let the wallet prove to the verifier it can compute full Diffie-Hellman keys by simply sending them to the verifier. This would constitute a Diffie-Hellman oracle allowing for a specific recovery attack on private key d, cf. [3]. This attack can be argued not to be of practical concern for the EUDI Wallet context, e.g. as only one full Diffie-Hellman key per new attestation key will be provided and only to the attestation issuer. However, avoiding the attack could be considered beneficial from a theoretical perspective. The essence of Proposition 3.8 is that a regular attestation private key is able to show an essentially different use of the exchanged Diffie-Hellman key S_{AB} than can performed with the WTE key. This can be conveniently catered for by letting regular attestation keys support ECDH-MAC signing using a derived key of form $K' = \text{HKDF}(Z_{AB}||0x02, \text{SharedInfo})$, i.e. different from the regular derived key $K = \text{HKDF}(Z_{AB}, \text{SharedInfo})$ used in ECDH-MAC

signing and verification. Here $0x02||Z_{\text{AB}}$ represents concatenating the byte $0x02$ to the byte string Z_{AB}. Note that the MAC-key is formed similarly as the MAC-key used in electronic passport secure messaging based on Chip Authentication, cf. the ICAO9303 specification [19]. Compare Algorithms 7, 8 in Annex E.

4 Further implementation notes

4.1 Three example WTE architectures (efficient, privacy friendly, PID-bound)

In Section 2.2 we have introduced the WTE/Proof-of-Association logic and in Section 3 we proposed a cryptographic method implementing this logic. In this section we demonstrate that the WTE/Proof-of-Association logic can be used to form different EUDI wallet architectures by varying the WTE role. Each of these EUDI wallet architectures have a different tradeoff between efficiency, privacy, functionality and security. That is, a WSCD supporting WTE/Proof-of-Association allows wallet providers a broad choice in developing different EUDI wallet architectures with very different properties. We demonstrate this flexibility by three example EUDI wallet architectures; further variants exist.

Optimally efficient WTE architecture
In an optimally efficient architecture the EUDI Wallet uses the WTE for all issuers, cf. Figure 3. It can be considered as the straightforward usage of the WTE/Proof-of-Association logic.
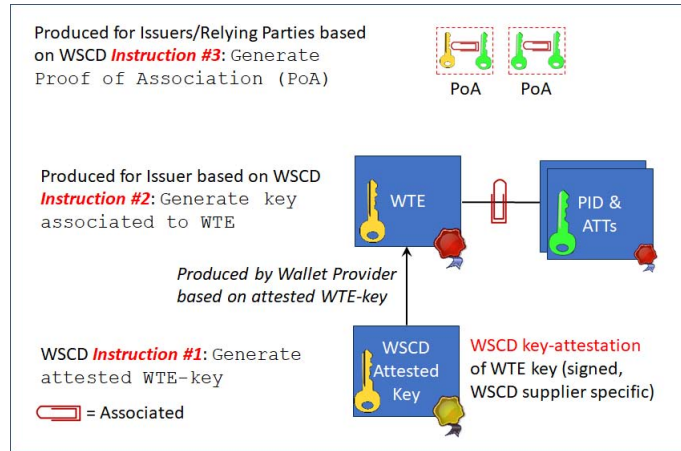

Figure 3. Optimally efficient WTE architecture

Privacy friendly WTE architecture
In the previous (optimally efficient) architecture the WTE becomes an object linking the EUDI wallet/user amongst the issuers. The resulting privacy risk can be accepted, e.g. in the situation that issuers process information directly identifying the user anyway, but can also be avoided. To this end, we introduce

Issuer Trust Evidences (ITEs) which are functionally the same as the WTE, i.e., hold the same information, but are not linkable to it. An ITE is issued by the Wallet Provider based on (and associated to) the WTE similar to Protocol 1 in Annex B. Figure 4 illustrates the role of the ITEs; each attestation issuer gets it owns ITE.
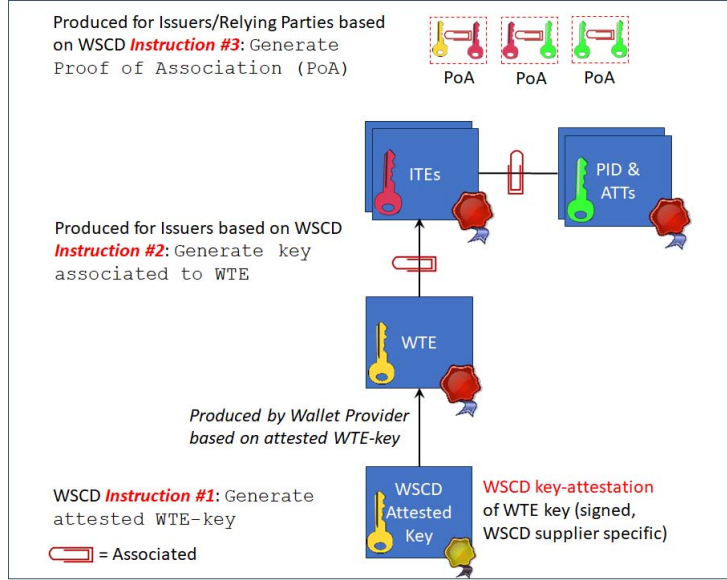

Figure 4. Privacy friendly WTE architecture

PID-bound WTE architecture
It can be beneficial from a security, privacy and functional perspective to let the PID issuer ensure that *only one* PID is associated to the WTE. This can be easily accomplished by combining the WTE and PID issuance whereby a PID challenge is part of the WTE and the key-attestation it is based on. By verifying that the PID challenge is indeed part of the WTE, the PID issuer can be sure it has never associated a PID to it. Such usage of challenges in key-attestation is actually standard and supported in GlobalPlatform [17], the ISO 23220-3 SAAO [27] and the Android Keystore. For the latter compare Figure 7 in Annex A.

As is indicated in Annexes B, C and D a uniquely associated PID gives rise to *PID-bound* attestations. These are attestations whereby the issuer has performed identity proofing using the PID and indicates this in the attestation. If there is only one PID associated to the WTE, then two associated and PID-bound attestations must then belong to the *same* PID holder. In other words, when a relying party is presented two PID-bound attestations and a proof of their association then they belong to the same PID holder, i.e. without having to show this PID. This constitutes a "privacy preserving technique ensuring unlikeability, where the attestation of attributes does not require the identification of the user" as requested in [13, Article 5a(16b)]. Compare Annex D where this further elaborated on. Figure 5 depicts the PID-bound WTE architecture.
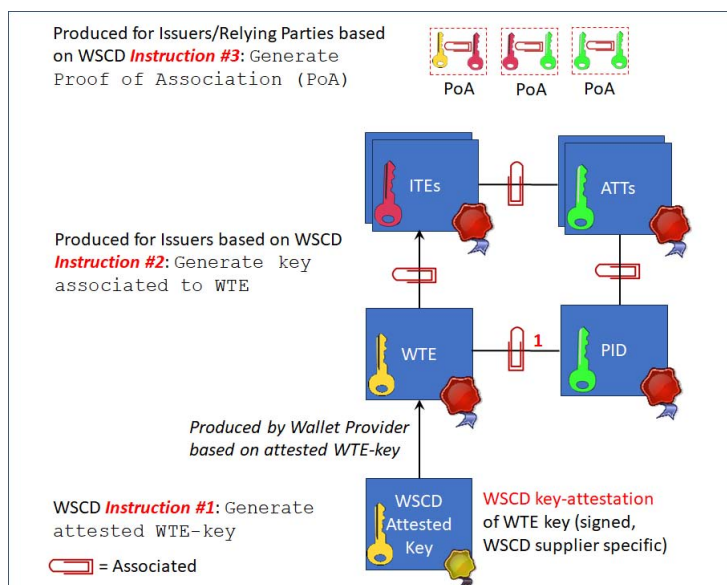
Figure 5. PID-bound WTE architecture

4.2 Freshness of associated keys

The proposed proof of association Algorithm 1 can be bound to a verifier challenge. Like in regular key-attestation, such challenges can constitute a mechanism to convey to a verifier that a proof and a certain key is fresh. For instance, we can include a 16 byte challenge of the verifier whereas the proofs of association binds to a 32 byte challenge where the last 16 bytes are chosen by the WSCD. By letting these bytes be all zero, the WSCD conveys that the key attested through the proof of association is fresh as otherwise it is not. For such fresh attested key generation, it seems convenient to combine the key generation and the proof-of-association in one WSCD instruction, i.e. a combination of WSCD instruction #1 and #2. This functionality is not further explored in this document.

4.3 Relation to Idemix/BBS+ protocols

The WTE construction and the proof of association proposed in this document can also easily implemented in the context of anonymous credentials such as based on BBS+ [2,7] or Idemix [8]. This means that the WTE construction and the proof of association are future proof constructions which are also in line with the GSM Association (GSMA) vision of BBS+/Idemix support in the EUDI Wallet through the embedded SIM (eSIM). Compare [18]. Although the WTE/-Proof of Concept functionality in the context of anonymous credentials is the same, we note that the WTE format and the proof of association cryptographic specifications are somewhat different.

To further elaborate; anonymous credentials attributes contain encrypted attributes in such a way that the EUDI wallet can selectively disclose attributes

with the additional property of "multi-show unlinkablity". This means that, other than through the disclosed attributes themselves, the presentation leaves no trace allowing relying parties to link various presentations at relying parties. So if the user has shown she is over 18 years old at two relying parties, these parties cannot link both presentations to one person. To show that multiple anonymous credentials belong to one EUDI wallet, one typically shares a common secret attribute value over all the anonymous credentials. The user then uses a zero-knowledge proof of knowledge to show the existence of the common secret attribute value to verifying parties.

The WTE construction and the proof of association naturally extend to anonymous credentials. The wallet provider then provides a WTE in the form of a anonymous credential holding a secret attribute value. The WSCD certification as indicted by the Wallet Provider in the WTE then ensures all anonymous credential secrets are securely managed. Issuers of BBS+/Idemix credentials then associate anonymous credentials to the WTE, by incorporating the common secret attribute value. The proof of association then constitutes to the zero-knowledge proof of knowledge showing existence of the common secret attribute value in the anonymous credentials.

5 References

1. The European Digital Identity Wallet Architecture and Reference Framework. https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework

2. M. H. Au, W. Susilo, and Y. Mu. Constant-Size Dynamic k-TAA. In: Security and Cryptography for Networks. Ed. by R. De Prisco and M. Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 111–125. isbn: 978-3-540-38081-8.

3. Daniel R. L. Brown, Robert P. Gallant, The Static Diffie-Hellman Problem 2004. See https://eprint.iacr.org/2004/306

4. BSI, Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, Request for Comment (RFC) 5639, March 2010.

5. BSI, Elliptic Curve Cryptography, TR-03111, version 2.10, 2018-06-01, 2018.

6. BSI, Technical Guideline BSI TR-03181, Cryptographic Service Provider 2 (CSP2), TR-03181, Version 0.94, 2023-04-03.

7. J. Camenisch, M. Drijvers, and A. Lehmann, Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited". In: Trust and Trustworthy Computing. Ed. by M. Franz and P. Papadimitratos.

8. Jan Camenisch, Anna Lysyanskaya, An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation, Proceedings of Eurocrypt 2001, LNCS 2045, Springer-Verlag 2001, 93-118.

9. https://developer.apple.com/documentation/devicecheck

10. https://developer.android.com/training/articles/security-key-attestation, https://developer.android.com/training/safetynet

11. https://source.android.com/static/docs/security/overview/reports/Android-Bootcamp-2016-Android-Keystore-Attestation.pdf

12. European Parliament and the Council of the European Union, Electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC, regulation 910/2014, 23 July 2014.

13. Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework. See https://eur-lex.europa.eu/eli/reg/2024/1183/oj.

14. European Commission, eIDAS implementing regulation 2015/1502, 8 September 2015.

15. A. Fiat, A. Shamir, How To Prove Yourself: Practical Solutions to Identification and Signature Problems, Crypto '86, Lecture Notes in Computer Science, Volume 263, Springer, 1986.

16. See https://fidoalliance.org/specs/fido-v2.0-ps-20150904/fido-key-attestation-v2.0-ps-20150904.html#privacy

17. See https://globalplatform.org.

18. See https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/files/11891976/GSMA.Official.Response.-.Privacy.for.eIDAS.-.June.2023.pdf

19. ICAO, Doc 2303, Machine Readable Travel Documents, Seventh Edition, part 11: security mechanims for MRTDs, 2015. Available from www.icao.int/.

20. Internet Engineering Task Force (IETF), HMAC-based Extract-and-Expand Key Derivation Function (HKDF), Request for Comment (RFC) 5869, May 2010.

5. REFERENCES

21. Internet Engineering Task Force (IETF), Edwards-Curve Digital Signature Algorithm (EdDSA), Request for Comment (RFC) 8032, January 2017.
22. Internet Engineering Task Force (IETF), Schnorr Non-interactive Zero-Knowledge Proof, Request for Comment (RFC) 8235, September 2017.
23. D. Hankerson, A. Menezes and S. Vanstone, Guide to Elliptic Curve Cryptography, Springer, 2004.
24. ISO, IT Security techniques — Digital signatures with appendix — Part 3: Discrete logarithm based mechanisms, ISO/IEC ISO 14888-3, Fourth edition, 2018-11.
25. ISO, Information technology - Security techniques - Methodology for IT security evaluation, ISO/IEC 18045, version 2014-01-15.
26. ISO, Mobile driving licence (mDL), ISO/IEC 18013 part 5, version 2021-09. application
27. ISO, ISO/IEC TS WD13 23220-3 issuing phase, draft, version 2024-03-15.
28. ITU-T, Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), X.690, 08/2015.
29. NI-Scy, Epic 09 - Wallet Trust Evidence, version 0.7.
30. NIST, The Keyed-Hash Message Authentication Code (HMAC), FIPS PUB 198-1, July 2008.
31. NIST, Secure Hash Standard (SHS), FIPS PUB 180-4, August 2015.
32. NIST, Digital signature standard, FIPS PUB 186-5. February 3 2023.
33. NIST, Recommendation for Key Management Part 1: General, SP 800-57, May 2020.
34. https://openid.net/sg/openid4vc/
35. C. P. Schnorr, Efficient signature generation for smart cards, Journal of Cryptology 4, 1991.
36. D.G. Stinson, Cryptography: theory and practice, CRC press, 1995.
37. E.R. Verheul, SECDSA: Mobile signing and authentication under classical "sole control", 16 March 2024. See https://eprint.iacr.org/2021/910.
38. StrongBox specification in Section 9.11.2 of https://source.android.com/compatibility/11/android-11-cdd.

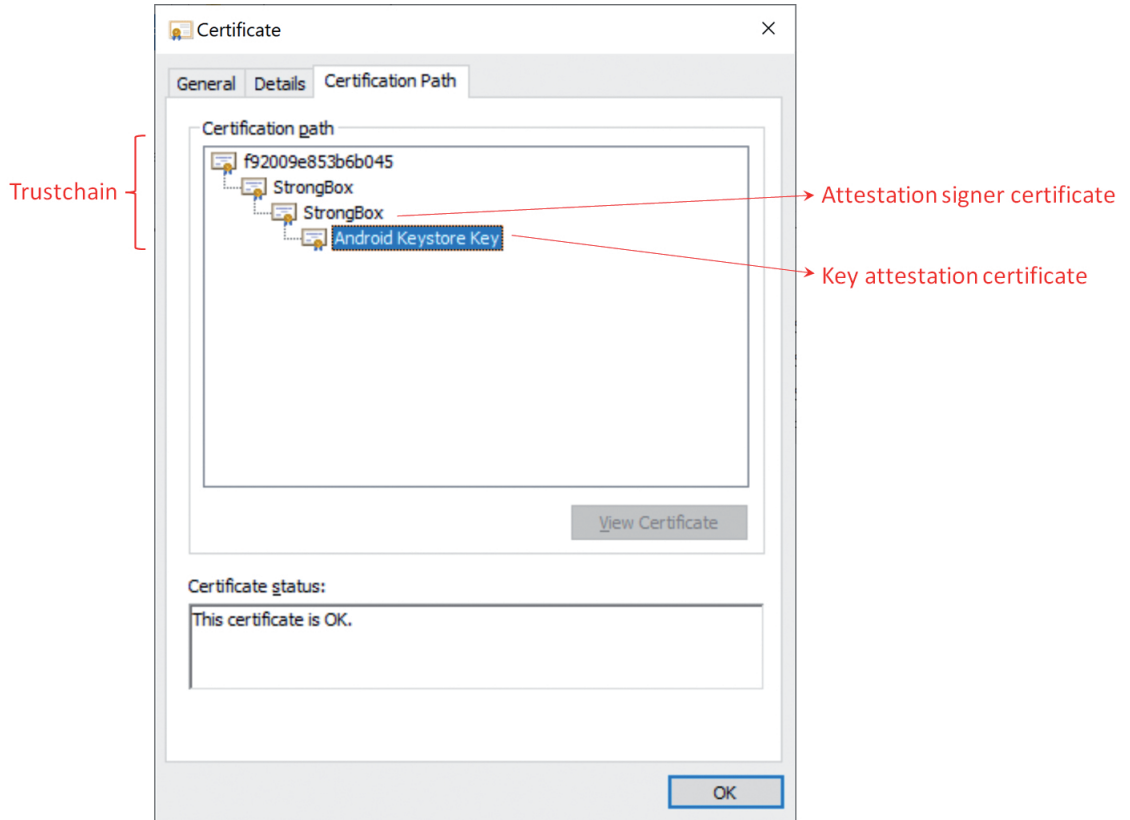# A   Android StrongBox key-attestation

Figure 6. Android StrongBox attested key (leaf)
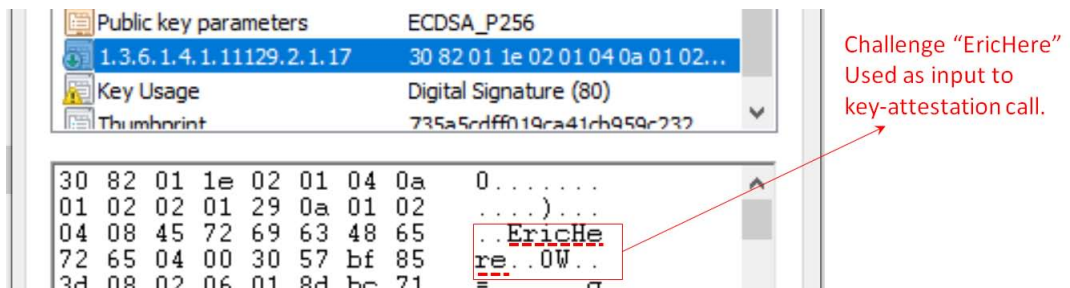
Figure 7. Third party challenge in key attestation

B Use of WSCD instructions in PID issuance (informative)

In Protocol 1 below we illustrate how we can use the WTE and the three WSCD instruction to issue a PID that is associated to the WTE. This is just an illustration on which many variants can be based. For simplicity we leave out the user (consent) involvement. In this particular variant we have chosen to let the WTE be fresh as it easily allows the PID issuer validation an issued PID is only associated to one WTE which can be security beneficial.

Protocol 1 PID issuance

Input: -

Output: User PID

```
 1: Wallet requests PID from PID issuer
 2: Issuer performs ‘‘proofing’’     // could also be elsewhere in process
 3: PID issuer generates challenge C and requests WTE bound to C
    // guaranteed fresh WTE
 4: Wallet calls WSCD with Instruction #1 including challenge C
 5: WSCD returns Attestation Cert K containing WTE public key Pub_WTE and C
 6: Wallet requests WP for WTE and sends Attestation Cert K
 7: WP verifies Attestation Cert K, if unsuccessful the protocol ends in error
 8: WP returns WTE on containing WTE public key Pub_WTE and C
 9: Wallet sends WTE on Pub_WTE and C to PID issuer in response to Step 3
10: PID Issuer verifies WTE, if unsuccessful the protocol ends in error
11: The PID issuer requests a PID public key associated with Pub_WTE
12: Wallet calls WSCD for a key associated with Pub_WTE    // Instruction #2
13: WSCD returns public key Pub_PID associated with Pub_WTE
14: Wallet calls WSCD for a signature on challenge C with Priv_PID    // PoP
15: WSCD returns PoP
16: Wallet calls WSCD for PoA[Pub_WTE, Pub_PID]                // Instruction #3
17: WSCD returns PoA[Pub_WTE, Pub_PID]      // proof Pub_WTE, Pub_PID are associated
18: Wallet sends Pub, PoP and PoA[Pub_WTE, Pub_PID] to (PID) issuer
19: PID Issuer verifies PoP, PoA[Pub_WTE, Pub_PID], on failure protocol ends
20: PID Issuer issues PID on public key Pub_PID indicating it is WTE associated
21: PID Issuer sends PID to wallet
```

The PoP (proof of possession) in Step 14 (verified in Step 19) depends on the signature algorithm it is based on. For EdDSA there are no particular requirements but for ECDSA (respectively ECDH-MAC) the requirements of Proposition 3.6 (respectively Proposition 3.8) apply.

As indicated in Section 4.2 we can arrange that the issuer can verify that the PID keypair is fresh by combining WSCD Instructions #2 and #3 in Steps 16 and 17 and the use of an issuer challenge.

The PID issuer indication in Step 20 that the PID is associated with the WTE is fundamental. It not only allows relying parties to verify that the PID private key is WSCD bound but it also allows other issuers to further bind attestations to this WSCD by associating their attestations to the PID public key. This makes thus use of the transitivity property of association. If we further arrange that there can only be one PID associated the WTE (as we have arranged in Protocol

B. USE OF WSCD INSTRUCTIONS IN PID ISSUANCE (INFORMATIVE)

1) then from the indication that two attestations are associated to a PID (and implicitly to a WTE) a relying party infer that these attestations are bound to the same PID, i.e. person. We further elaborate on this in Annexes C and D.

C Use of WSCD instructions in PID based issuance (informative)

In Protocol 2 below we illustrate how we can use the WTE and the three WSCD instructions to issue attestations based on the PID. For simplicity we leave out the user (consent) involvement. Protocol 2 is just an illustration on which many variants can be based. In Step 10 of this protocol we use the proof of association of three public keys as introduced in the notes following Algorithm 2.

Protocol 2 Attestation issuance based on PID
Input: WTE, User PID associated to WTE
Output: Attestation associated to both PID and WTE

```
 1: Wallet sends WTE, PID and requests attestation from issuer based on PID
 2: Issuer verifies validity WTE, PID                    // signatures etc.
 3: Issuer generates challenge C and sends it to wallet
 4: Wallet signs C with Priv_PID and sends result to issuer
 5: Issuer verifies signature with Pub_PID               // Proof of Possession
 6: Issuer uses PID data to form attestation attributes     // e.g. diploma
 7: Issuer requests for attestation public key associated to Pub_WTE
 8: Wallet calls WSCD for keypair associated with Pub_WTE   // Instruction #2
 9: WSCD returns WTE associated public key Pub
10: Wallet calls WSCD for PoA[Pub_WTE, Pub_PID, Pub]
11: WSCD returns PoA[Pub_WTE, Pub_PID, Pub]  // proof Pub_WTE, Pub_PID, Pub associated
12: Wallet calls WSCD for a signature on challenge C with Priv      // PoP
13: WSCD returns PoP
14: Wallet sends Pub, PoP and PoA[Pub_WTE, Pub_PID, Pub] to issuer
15: Issuer verifies PoP, PoA[Pub_WTE, Pub_PID, Pub], on failure protocol ends
16: Issuer issues attestation on the attributes from Step 6 and Pub indicating
    it is both PID & WTE associated
17: Issuer sends attestation to wallet
```

The PoP (proof of possession) in Step 12 (verified in Step 15) depends on the signature algorithm it is based on. For EdDSA there are no particular requirements but for ECDSA (respectively ECDH-MAC) the requirements of Proposition 3.6 (respectively Proposition 3.8) apply.

D Use of WSCD instructions in presentations of PID based attestations (informative)

In Protocol 3 below we illustrate how we can use the third WSCD instruction (proof of association) to prove to a relying party that multiple attestations originate from one EUDI Wallet and correspond to one person. This is just an illustration on which many variants can be based. For simplicity we only have two attestations (think of the "adult" (over 18 years old) and "photo" attestation from Section 2.1) and leave out the user (consent) involvement.

Protocol 3 is just an illustration on which many variants can be based.

Protocol 3 Multiple attestation presentation to relying party
Input: two PID-based attestations $\mathcal{A}_1, A_2$ on public keys Pub_1, Pub_2
Output: User PID

```
 1: Wallet sends PID based attestations 𝒜₁, 𝒜₂ to relying party (RP)
 2: Relying party verifies validity attestations 𝒜₁, 𝒜₂ // signatures etc.
 3: Relying party verifies 𝒜₁, 𝒜₂ are both WTE & PID based
    // attestations are bound to both one WSCD and one PID
 4: Relying party generates challenge C and sends it to wallet
 5: Wallet signs C with Priv₁, Priv₂ and sends results to RP
 6: RP verifies signatures with Pub₁, Pub₂            // Proof of possession
 7: RP requests for proof-of-association Pub₁, Pub₂
 8: Wallet calls WSCD for PoA[Pub₁, Pub₂]
 9: WSCD returns PoA[Pub₁, Pub₂]        // proof Pub₁ and Pub₂ are associated
10: Wallet sends PoA[Pub₁, Pub₂] to RP
11: RP verifies PoA[Pub₁, Pub₂], on failure protocol ends
12: RP accepts attestations 𝒜₁, 𝒜₂ and infers attestations are bound to one
    WSCD and one PID (person)
```

E   Cryptographic and mathematical background

We let $\mathbb{F}_r$ denote the Galois field consisting of the integers modulo a prime number r. We let $\mathbb{F}_r^*$ denote the multiplicative subgroup, i.e. the non-zero elements. See [36]. We sometimes implicitly use that $\mathbb{F}_r$, respectively $\mathbb{F}_r^*$, corresponds to the integers in the interval $[0, r-1]$, respectively $[1, r-1]$ and write operations in combination with "mod r". We let $|r| = \lceil \log_{256}(r) \rceil$ denote the size in bytes of r, i.e. the minimal number of bytes to represent r.

Central in our constructions is an additive group $\mathbb{G} = (\langle G \rangle, +)$ of order q generated by a *base point* (generator) G. We use additive notation as this is customary in the context of elliptic curve groups we deploy in practice. We require that q is prime. For any natural scalar n and element $H \in \langle G \rangle$ we define the *(point) multiplication* nH as adding H n-times, e.g. $2H = H + H$. As $nH = mH$ if and only if $n = m \bmod q$ we can represent scalars as elements of $\mathbb{F}_q$. This allows for compact notation as $x \cdot G$, $-x \cdot G$ for $x \in \mathbb{F}_q$ and $y^{-1} \cdot G$ for $y \in \mathbb{F}_q^*$. We sometimes omit the "$\cdot$" symbol and simply write xG. A cryptographically secure (pseudo) randomly chosen element from a set is denoted by $\in_R$.

The required cryptographic security of the group $(\langle G \rangle, +)$ can be formulated in the intractability of three problems. The first one is the *Diffie-Hellman problem*: computing the values of the function $DH_G(xG, yG) = xyG$ for any $x, y \in \mathbb{F}_q$ (implicitly given but unknown). The second problem is the *Decision Diffie-Hellman* (DDH) problem: given $A, B, C \in_R \langle G \rangle$ decide whether $C = DH_G(A, B)$ or not. An equivalent definition is as follows. Any quadruple of points (G, A, B, C) in $\langle G \rangle$ can be written as (G, A, xG, yA) for some (unknown) $x, y \in \mathbb{F}_q$. DDH amounts to deciding whether a random quadruple of points in G is a *DDH quadruple*, i.e. if $x = y$. The DH problem is at least as difficult as the DDH problem. The last related problem is the *discrete logarithm* (DL) problem in $\langle G \rangle$: given $A = xG \in \langle G \rangle$, with $x \in \mathbb{F}_q$ then find $x = \mathrm{DL}_G(A)$. It easily follows that the DL problem is at least as difficult as the DH problem.

We assume that all three introduced problems in $\langle G \rangle$ are intractable which implies that the size $|q|$ of the group order should be at least 256 bits. A prominent example of $\mathbb{G}$ is a group of points over a field $\mathbb{F}_p$ on a curve with simplified Weierstrass equation

$$y^2 = x^3 + ax + b \tag{2}$$

for some suitable $a, b \in \mathbb{F}_p$. That is, each non-zero group element takes the form (x, y) where $0 \leq x, y < p$ satisfying Equation (2) modulo p. Compare [23]. We denote the zero element (point at infinity) as $\mathcal{O}$. For practical implementations one can use one of the NIST curves [32], e.g. P-256 or Brainpool curves [4], e.g. brainpoolP320r1.

Below we describe the working of the ECDSA [32] and ECDH-MAC [26] signature generation and verification algorithms. In all settings the user has a private key $d \in \mathbb{F}_q^*$ and a corresponding public key $D = d{\cdot}G$. In these specification a secure hash function $\mathcal{H}(.)$ appears, cf. [36,31]. Such a function takes as input byte arrays of arbitrary size and outputs a byte array of fixed length equal to $|q|$. The latter can be accomplished by taking a secure hash function of appropriate output size or one with larger output size and truncating its output.

Algorithms 5 and 6 below specify ECDSA signing and verification following [23].

Algorithm 5 ECDSA signature generation

Input: message M, private key d

Output: signature (r, s).

```
1: Compute H(M) and convert this to an integer e.
2: Select random k ∈ {1,...,q − 1}.
3: Compute kG = (x,y) and convert x to integer x̄.          // commitment
4: Compute r = x̄ mod q. If r = 0 go to Line 2.
5: Compute s = k⁻¹(e + d·r) mod q. If s = 0 go to Line 2.
6: Return (r, s).
```

We remark that in the situations where cryptographic hardware is used, the calculation of the hash value of message M in Line 1 of Algorithm 5 is typically not performed by this hardware. This is typically due to communicational or computational restrictions in using the hardware. In these circumstances the hash value H of message M is pre-computed in the application calling the hardware and then sent to the hardware as input. The hardware then converts the hash value directly to the integer e of Line 1 of Algorithm 5 and performs the following Lines 2-6. This setup is known as *raw* signing, i.e. generation of a signature directly on basis of a hash value without a deploying a hash operation. Similarly one has raw verification where the hash value is directly converted to the integer e in Step 2 of Algorithm 6.

Algorithm 6 ECDSA signature verification

Input: message M, signature (r, s), public key $D = d{\cdot}G$

Output: Acceptance or rejection of the signature.

```
1: Verify r, s are integers in [1, q − 1], on failure reject signature.
2: Compute H(M) and convert this to an integer e.
3: Compute w = s⁻¹ mod q.
4: Compute t₁ = e·w mod q and t₂ = r·w mod q.
5: Compute X = t₁·G + t₂·D.
6: If X = O reject the signature.
7: Convert the x-coordinate of X to an integer x̄; compute v = x̄ mod q.
8: If v = r accept the signature otherwise reject it.
```

Algorithms 7 and 8 specify ECDH-MAC signing and verification based on ISO 18013-5 [26]. It is based on a Message Authentication Code (MAC) on a message M generated using a conventional MAC Algorithm. ISO 18013-5 [26] stipulates

the use of the HMAC algorithm [30]. This MAC is based on a key K of type byte array; the MAC computation is denoted by $\text{HMAC}(K, M)$.

Key K is derived from a byte array representation Z_{AB} of the Diffie-Hellman key S_{AB} shared between the signer and verifier and a byte array SharedInfo. The latter holds additional information shared between the signer and the verifier. In [26] the shared information includes a session transcript. For this key derivation, ISO 18013-5 [26] stipulates the use of the HKDF algorithm [20]. We therefore denote the key derivation by $\text{HKDF}(Z_{\text{AB}}, \text{SharedInfo})$.

ISO standard 18013-5 [26, Section 9.1.3.5] only implicitly defines ECDH-MAC signing and verification. This is also done in the particular context of a mobile driving license. Algorithms 7 and 8 are generic, explicit specifications meeting the essence of [26]. The notation used is also in line with Section 4.3 of BSI publication TR-03111 [5] specifying the Diffie-Hellman protocol. Algorithm 7 takes an ephemeral public key E as input, whereas Algorithm 8 takes an ephemeral private key k as input. This ephemeral public key takes the form $E = k{\cdot}G$ and is based on a (fresh) ephemeral private key k generated by the verifying party, e.g. the issuer or relying party in the context of the EUDI Wallet. The verifying party is guaranteed that the message is correctly signed by the signer but cannot transfer this guarantee to another party. Compare the comment following Algorithm 8. In other words ECDH-MAC signing supports plausible deniability for the user, i.e. the opposite of non-repudiation, which can be beneficial in certain use cases. As it lacks non-repudiation, an ECDH-MAC signature is strictly speaking not a digital signature.

Algorithm 7 Generic ECDH-MAC signature generation

Input: message M, private key d, ephemeral public key E, byte array SharedInfo
Output: byte array MAC.

```
1: Verify that E ∈ ⟨G⟩, on error algorithm stops
2: Compute S_AB = d·E                    // compute shared Diffie-Hellman key
3: Convert S_AB to byte array Z_AB
4: Compute K = HKDF(Z_AB, SharedInfo)              // derive MAC-key K
5: Compute MAC = HMAC((K, M)).
6: Return MAC.
```

Algorithm 8 ECDH-MAC signature verification

Input: message M, ephemeral private key k, SharedInfo, byte array MAC, public key $D = d{\cdot}G$
Output: Acceptance of rejection of the MAC.

```
1: Compute S_AB = k·D                     // shared Diffie-Hellman key
2: Convert S_AB to byte array Z_AB
3: Compute K = HKDF(Z_AB, SharedInfo)              // derive MAC-key K
4: Compute MAC' = HMAC(K, M)
5: If MAC' = MAC accept the MAC otherwise reject it
```

Note that in Algorithm 8 the verifier re-generates the MAC value itself based on the public key of the signer. This means that signer can always deny having generated the MAC.

When using cryptographic hardware, e.g. the WSCD in EUDI Wallet context, Steps 1-2 of Algorithm 7 are always performed there. In that context, an important design decision is where the MAC-key K in computed, i.e. Step 4. Step 4 can be performed in the cryptographic hardware or in the application calling the hardware. In the second case the cryptographic hardware returns Z_{AB} to the calling application following Step 3 which then generates the MAC-key K in Step 4. In the first case the calling application sends the ephemeral public key E and the shared information SharedInfo to the cryptographic hardware. The cryptographic hardware then performs Steps 2-4 and returns MAC-key K to the calling application. If cryptographic hardware for a private key d supports the second case, i.e. returning the full Diffie-Hellman key Z_{AB}, we say that private key d supports *full Diffie-Hellman*. With saying that private key d does not support full Diffie-Hellman we mean that it only returns the HKDF-derived key from Step 4 in Algorithm 7, i.e., a hash based value of Z_{AB}.

We note that with full Diffie-Hellman support, the cryptographic hardware provides for a so-called Diffie-Hellman oracle allowing for a specific recovery attack on private key d, cf. [3]. This attack can be argued not to be of practical issue for the EUDI Wallet context, but avoiding the attack could be considered beneficial from a theoretical perspective.

F   The EC-Schnorr-Sound algorithm

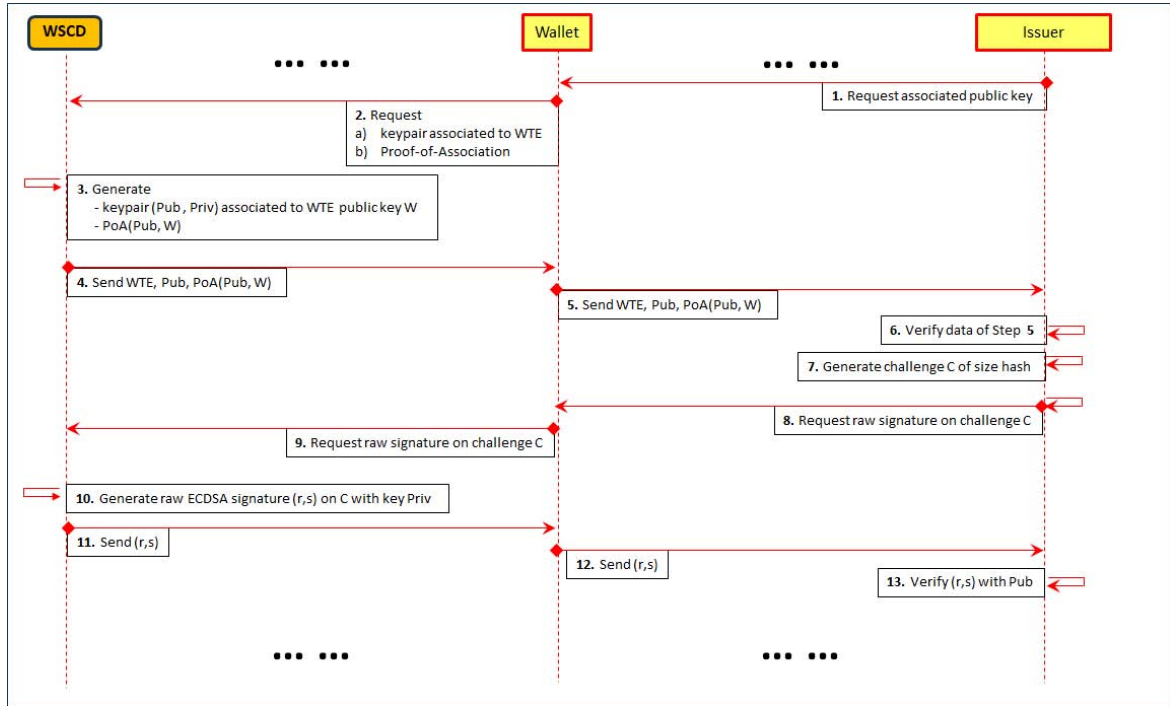## G   Example applications of Proposition 3.5 (informative)

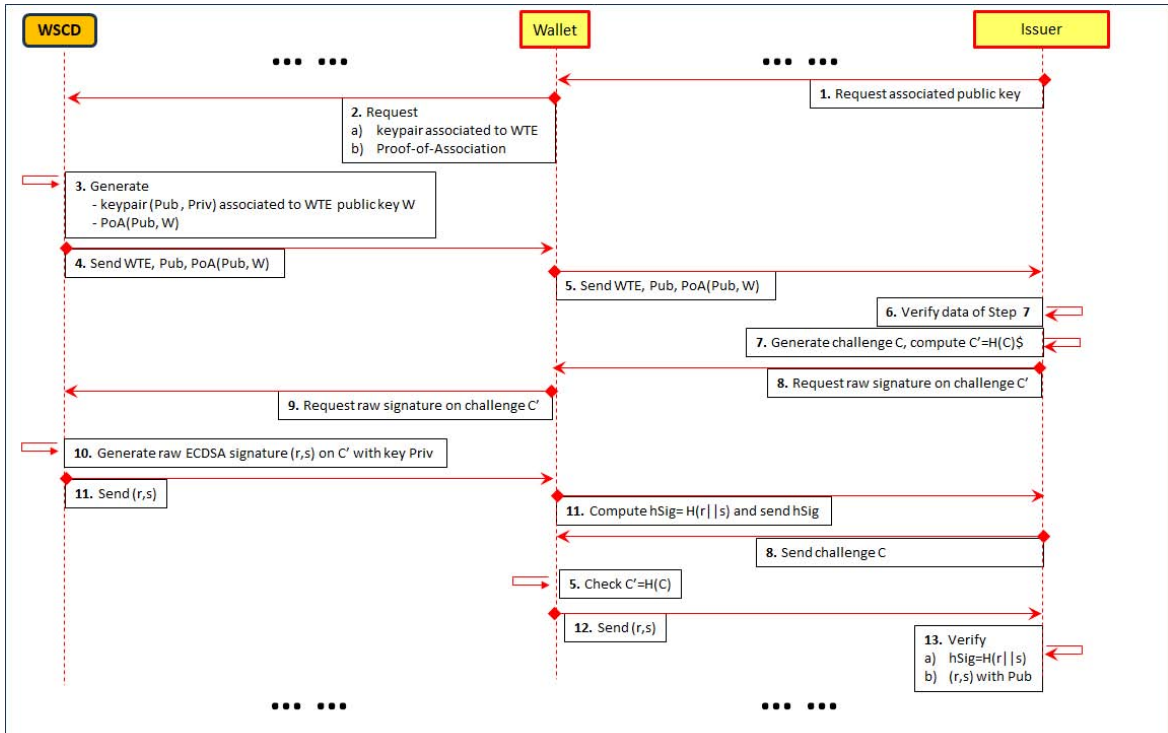**Figure 8.** Straightforward application of Proposition 3.6

Figure 9. Demonstrating raw ECDSA signing without actually doing it
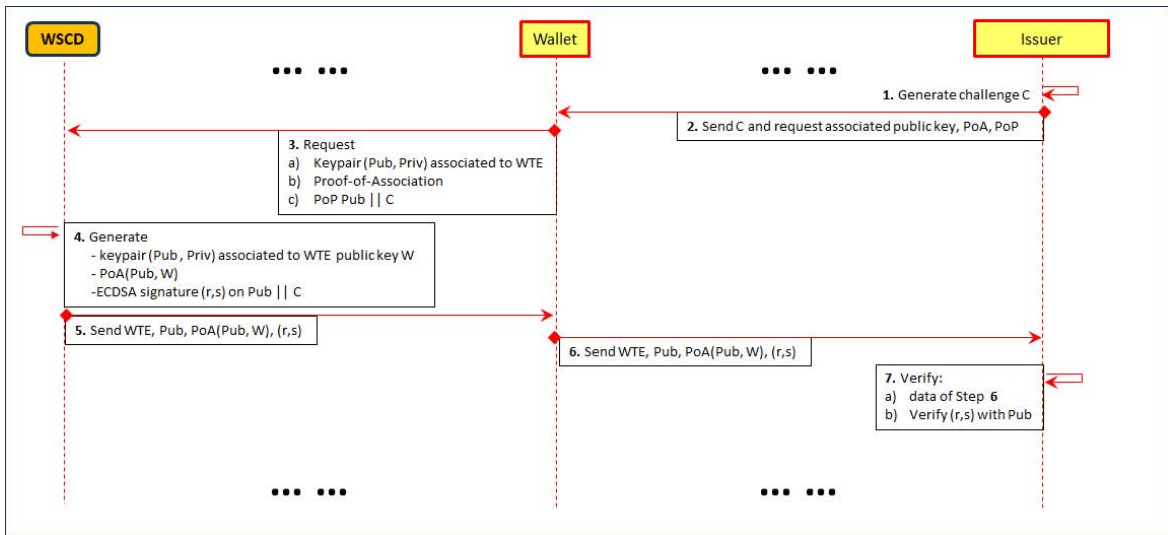
Figure 10. Application of Proposition 3.7

H   ASN.1 format for Proof of Association (informative)

Below we have specified a proposal for the proof of association in ASN.1 format, cf. [28]

```
ProofOfAssociation ::= SEQUENCE {
    NotationIdentifier OBJECT IDENTIFIER (id-proof-of-association),
    Version INTEGER,
    Challenge OCTET STRING OPTIONAL, // can be beneficial
    Generator ECPoint,
    PubKey    ECPoint
}


SignedProofOfAssociation ::= SEQUENCE {
NotationIdentifier OBJECT IDENTIFIER (id-proof-of-association-signed),
    ProofOfAssociation ProofOfAssociation,
    SignatureValue EC-Signature
}


EC-Signature :: SEQUENCE {
        SignatureType      OBJECT IDENTIFIER // e.g. ecschnorr-plain-SHA256)
        SignatureValue     EC-Sig-Value
-- EC-Sig-Value is identitical to BSI TR 03111 ECDSA-Sig-Value.
}
```