# Attestation Proof of Association

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

Eric R. Verheul eric.verheul@logius.nl

Version 1.2, 19th January 2025

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 [23] 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. This paper concentrates on irrefutable PoAs but also indicates how refutable PoAs can be formed providing plausible deniability which can be beneficial in some use cases.

As a side note this paper introduces in an annex the construction of Self Generated Verifiable Pseudonyms (SGVPs). These allow a wallet/user to generate pseudonyms based on information agreed with a relying party, e.g. an URL, and to prove these are correctly formed. Together with the proof of association this allows cryptographically binding (disclosed parts of) attestations with these pseudonyms. This enables various use cases such as an employee representing an organisation in a privacy friendly way using an chamber of commerce attestation cryptographically bound to a separate SGVP-pseudonym. Such functionality currently forms the privacy basis of the Dutch eRecognition scheme (eherkenning.nl).

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

# Contents

1	Introduction		
2	Security problem description		
	2.1 Three fundamental WSCD security requirements		
	2.2 Three fundamental WSCD instructions		
3	A proof of association proposal		
4	Further implementation notes		
	4.1 Three example WTE architectures (efficient, privacy friendly,		
	PID-bound) 24		
	4.2 Freshness of associated keys		
	4.3 Relation to Idemix/BBS+ protocols		
5	References		
A	Android StrongBox key-attestation		
В	Use of WSCD instructions in PID issuance (informative) 3		
$\mathbf{C}$	Use of WSCD instructions in PID based issuance (informative) 3		
D	Use of WSCD instructions in presentations of PID based attestations		
	(informative)		
$\mathbf{E}$	Cryptographic and mathematical background		
F	Example applications of Proposition 3.5 (informative)		
G	ASN.1 format for Proof of Association (informative)		
Η	Self Generated Verifiable Pseudonyms (SGVP) 44		
	H.1 SGVP building blocks 44		
	H.2 Sketches of how SGVP pseudonyms can be used in practice 4		
т.	. CD		
Lı	st of Figures		
1	EUDI wallet overview		
2	mDL outline		
3	PID and mDL trust relation		
4	Selectively combining and linking PID-bound attestations		
5	Chamber of commerce (CoC) use case example		
6	The distributed WSCD		
7	Optimally efficient WTE architecture		
8	Privacy friendly WTE architecture		
9	PID-bound WTE architecture		
10	Android StrongBox attested key (leaf)		
11	Third party challenge in key attestation		
12	Straightforward application of Proposition 3.6		
13	Demonstrating raw ECDSA signing without actually doing it		
14	Application of Proposition 3.7		

# 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)  The classical desiration of the control
0.3	2024-03-11	<ul> <li>Further clarification and typo fixing</li> <li>Added proof of association heuristics in Section 3</li> </ul>
		• Added Section 4.3 indicating compatibility with BBS+ and Idemix protocols
0.4	2024-03-19	Processed first comments
0.12		• 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
		<ul> <li>Added ECDH-MAC specification to Annex E following ISO 18013-5</li> <li>Added the distributed WSCD (Figure 6)</li> </ul>
		• Added the distributed WSCD (Figure 6) • Added security proof to Section 3
0.6	2024-04-26	• Fixed typos
0.0	20210120	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 F 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 show-
0.02	2021 00 20	ing three WTE architectures with different privacy and security properties
		whereby also introducing the Issuer Trust Evidence (ITE) object; issuer spe-
		cific WTEs avoiding linkability.
0.69	0004.05.00	• Added the HSM based wallet in Figure 6. Added comparison with "simplified WTE/PoA" of the draft standard ISO
0.63	2024-05-30	23220-3 [33, 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 al-
0.01	2021 00 00	gorithms specified in ISO 18013-5 [32], i.e. EdDSA, ECDSA and ECDH-MAC.
1.0	2024-09-18	Finalized for publication
1.1	2025-01-07	• Updated the motivation of security requirement SW1 by use case of crypto-
		graphically binding mobile driving license photo to PID resident address
		• Added in Annex G a pseudonym construction which together with the proof
		of association can cryptographically bind any (part of) an attestation/PID to a pseudonym
		• Added references to the eIDAS implementation regulations published on 28
		November 2024.
1.2	2025-01-19	Renamed the term Pseudonym Deriver to Pseudonymisation Domain follow-
		ing the European Data Protection Board (EDPD) draft Guidelines on Pseud-
	[	onymisation [20].

#### 1 Introduction

The update on 11 April 2024 [14] to the 2014 eIDAS regulation [13] introduces an European Digital Identity Wallet (hereafter: EUDI wallet or for brevity sometimes simply wallet). According to [14], 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 whereby the services can be either in online or offline mode. In offline mode the interaction between the user and relying party occurs at a physical location using close proximity technologies. One can think of a user presenting the mobile driving license to a police agent. 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 [14] 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. Compare the annex of eIDAS implementation regulation 2024/2977 [16] which specifies the mandatory and optional PID data elements. Mandatory elements include first and last name, date of birth, place of birth and nationality of the holder. Optional elements include the holder resident address and portrait (facial image) in JPEG format. 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 is further specified in eIDAS implementation regulation 2024/2982 [19] which refers to ISO 18013-5 ('mdoc') for offline mode and to ISO 18013-7 for online mode. The latter further refers to OpenID for Verifiable Credentials [41]. The EUDI wallet is described in more detail in the Architecture and Reference Framework [1].

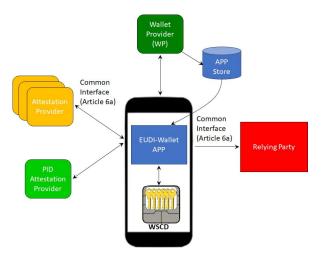


Figure 1. EUDI wallet overview

Although [14] allows for other types of attestations, in the context of this document a PID and attestations are functionally considered PKI-certificates. That is, a PID/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. We note that PID/attestation have a security/privacy feature called "selective disclosure" specified in Article 5 of the eIDAS implementation regulation 2024/2982 [19] on wallet protocols and interfaces. This feature is typically not supported by conventional PKI-certificates and allows a holder to only reveal certain attributes from a PID/attestation, i.e. not all attributes. This allows for instance to only reveal the resident address and nationality from the PID to a relying party and not the other attributes such as last name and date of birth. Cryptographically selective disclosure is implemented by not including the attributes themselves in the PID/attestation but seeded hash values of those, cf. [32]. During presentation the holder only provides the seeds/attributes it wants to present.

In this paper focus on the signature algorithms stipulated in the mobile driving licence standard [32], i.e., EdDSA [27,39], ECDSA [6,30,39] and "ECDH-MAC" signing as defined in [32] 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 [15] 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 [31] at assurance level EAL4+. For the EUDI wallet this interpretation is made explicit in eIDAS implementation regulation 2024/2981 [18].

Next to security, also privacy protection plays an important role in the EUDI wallet. Several articles of the eIDAS regulation update [14] 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 [23] 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 hard-ware 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 Global-Platform [23] based Secure Element [23] such as an eID-card or an eSIM. By their simplicity, the three instructions also allow for convenient Common Criteria certification [31] of the EUDI wallet, as required in eIDAS implementation regulation 2024/2981 [18]. This document is a further refinement and concretisation of the EPIC-09 result [36] 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 on the Schnorr zero-knowledge proof. For simplicity we only work out the non-interactive version which is irrefutable which can be beneficial in some use cases and undesirable in other. However, we also indicate the use of two interactive proof techniques which are refutable, cf. Note 5 on page 16.
- 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+ [2,8] or Idemix [9].
- 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 F contains three example applications of Proposition 3.6, two of which avoid the use of raw ECDSA signing.
- Annex G contains a proposal for a proof of association specification in ASN.1 format.
- Annex H specifies a pseudonym construction which together with the proof of association can cryptographically bind any (part of) attestation/PID to a pseudonym.

# 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 [23]. 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 [10] and Google's Safetynet [11] provide for Mobile App attestation. However, as Mobile App attestation is provided by the mobile operation system, i.e. software, it has a large attack surface 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 certified WSCD. This brings us to the first fundamental WSCD security requirement:

<sup>&</sup>lt;sup>1</sup> Compare https://github.com/kdrag0n/safetynet-fix

ICW (InCertWSCD) During issuance the PID/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 [23] but also by the Android Keystore for both its Hardware Backed Keystore (TEE based) and its EAL4+ certified StrongBox chip, cf. [45]. In Figure 10 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 [33]. 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 a mobile driving license [32] based on her PID. A mobile driving license can be considered an attestation of attributes holding basic holder identification data, facial image and a set of attributes representing the holder permissions for a driving a vehicle. Compare Figure 2 below.



Figure 2. mDL outline

The issuance of the mobile driving license starts with an identification of the user based on her PID followed by a lookup in the national driving license data-

base. During issuance the issuer does not only need assurance that the mobile driving public key sent by the EUDI wallet is bound to a certified WSCD, e.g. Requirement ICW, but also that the public key is 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, or worse, to a software keystore making the private key copyable. In this abuse case another user could then present the mobile driving license of the first user, as being hers. If the private key was generated in a software keystore, the mobile driving license is freely distributable and could be presented by any user. In physical use cases, e.g., the user presenting the mobile driving license to a police agent, the biometric validation against the facial image can be bypassed by techniques commonly known as "look-alike fraud". In fact, such techniques are the reason that conventional RFID chip based driving licenses and passports support an anti-cloning technique known as Active Authentication, cf. [25]. The avoidance of such abuse cases 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 and last fundamental WSCD security requirement is the counterpart of the second WSCD security requirement for relying parties. Assume that the EUDI wallet user has been issued a PID and a mobile driving license and as discussed above and that the issuer have been assured of Requirement SW1. That is, that the mobile driving license private key resides in the same certified WSCD as the PID private key. Now suppose that the user wants to present her resident address from the PID together with the facial image from the mobile driving license to a municipal waste collection facility. Compare Figure 3 below.

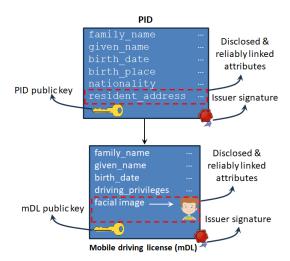


Figure 3. PID and mDL trust relation

The idea behind this use case is that this facility only provides its services to persons having their residency in that municipality which is actually the case in the Netherlands. Note that this use case deploys the selective disclosure feature of PID/attestations discussed in Section 1. The municipal waste collection facility would need to be assured that the PID resident address and the mobile driving license facial image 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 PID resident address of one user/wallet and the mobile driving license facial image from another. Similar wallet use cases exist where attributes from various (PID-bound) attestations need to reliably combined. Such security functionality has value at its own, but is also a building block for the GDPR data minimization principle as well as for the "need to know" principle practised in high-end security applications. Compare Figure 4 below.

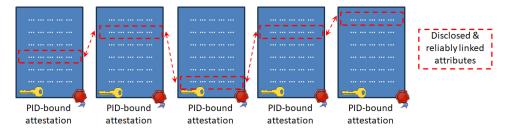


Figure 4. Selectively combining and linking PID-bound attestations

A more generic use case is related to Article 14 of eIDAS implementation regulation 2024/2979 [17]. This article stipulates that wallets must be able to generate unique user pseudonyms that are relying party specific and provide these to these relying parties "either standalone or in combination with any person

identification data or electronic attribute attestation requested by that walletrelying party". We note that pseudonyms allow relying parties recognizing the user without identifying her, which is required in many use cases. Although, Article 14 does not explicitly specify this, preamble (14) of implementation regulation 2024/2979 states that these pseudonyms "should enable wallet users to authenticate themselves without providing wallet-relying parties with unnecessary information" (our underlining). From a cryptographic perspective, a natural implementation to meet these requirements is to let pseudonyms be derived from some of the PID data and be provided to relying parties as separate attribute attestations. These attestations can be either issued by a pseudonym attribute attestation provider or by the user herself. The latter is more challenging but feasible as is illustrated by the construction of Self Generated Verifiable Pseudonyms (SGVP) specified in Annex H. When such pseudonym is combined and provided with data from the PID (or another attribute attestation) to a relying party as required by Article 14, it is vital that the relying party can be assured that the pseudonym attestation is bound to same WSCD as the user PID is. Such functionality would then for instance enable to reliable link a pseudonym to a power of attorney attestation by the chamber of commerce. This allows an employee of an organisation to act on behalf of that organisation to the tax authority in a persistent way whereby only providing minimal personal data, e.g. the name of the organisation, the pseudonym and (optionally) the last name of the person. We remark that the Dutch e-recognition scheme (eHerkenning.nl) functionally works this way, i.e. authenticates employees using a pseudonym and the employee last name to governmental organisations. Compare Figure 5 belows where this is illustrated for the CEO (James Quincey) of the Coca-Cola company.

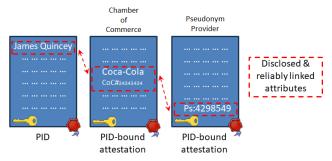


Figure 5. Chamber of commerce (CoC) use case example

An other application is based on a shareholder attestation issued by the chamber of commerce allowing for anonymous online voting during shareholder meetings. These applications bring 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. [41]. 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. [40]. 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 this 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 10 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 [14]. 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 [12] or [22] 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 ( $Pub_{\text{WTE}}, Priv_{\text{WTE}}$ ) 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  $Priv_{\text{WTE}}$ . If these verifications are successful, then the Wallet Provider issues the WTE attestation on the public key  $Pub_{\text{WTE}}$ . The WTE only contains minimal data; essentially nothing more than that the WSCD is eIDAS-conformant. Also, the WTE public key  $Pub_{\text{WTE}}$ 

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 keyattestation 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 COutput: WTE public key  $Pub_{\text{WTE}}$ ,

WTE Key-Attestation Certificate K containing  $Pub_{\text{WTE}}$ , C

Return  $Pub_{\mathtt{WTE}}$ ,  $\mathcal{K}$ 

// 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  $Pub_{\rm WTE}$ . If a public key Pub is associated to a WTE public key  $Pub_{\rm WTE}$ , we will also allow the generation of new keypair that is associated to Pub and then by inheritance also to the WTE public key  $Pub_{\rm WTE}$ . 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  $Rf_{ ext{WTE}}$  // 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:  $PoA(Pub_1, Pub_2)$ 

- 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 [33] 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 [14, 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 11 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 [14]. 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 [14], 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  $z = p \cdot w^{-1} \mod q$ . 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 [21] similar to RFC 8235 [28].

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: PoA =  $\{P_1, P_2, C, (r, s)\}$ .

The following algorithm specifies the verification of a PoA.

```
Algorithm 2 Proof of Association (PoA) verification
```

Input: WTE, PoA =  $\{P_1, P_2, C, (r, s)\}$ 

```
Output: Acceptance of rejection of the PoA.

1: Verify P_1 \neq P_2 on failure Return Error //P_1, P_2 need to be different 2: Verify the input, including that r \in \{1, 2^{8\cdot |q|} - 1\} and s \in \{1, q - 1\}, on failure Return False.

3: Convert public keys P_1 and P_2 to byte arrays \bar{P}_1, \bar{P}_2 respectively 4: Compute Q = s \cdot P_1 - r \cdot P_2 if Q = \mathcal{O} Return False.

5: Convert Q to byte array \bar{Q}. // i.e. of size 2\cdot |p| 6: Compute byte array H(\bar{Q} || \bar{P}_1 || \bar{P}_2 || 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 [43, Theorem 9.1].  $\Box$ 

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 [32] 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.  $PoA[P_1, P_2, P_3]$  consists of  $PoA[P_1, P_2]$  and  $PoA[P_2, P_3]$ . 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 [6,30,42]. This means that if a platform supports ECSDSA then the proof of association is easily implemented.
- 5. For simplicity of presentation we only work present a non-interactive proof. This is quite efficient as this does not requires interaction with the verifier, e.g. an issuer of relying party The non-interactive proof has the property of being irrefutable, i.e. the user cannot deny on a later moment that certain keys and thus the attestations they are bound to are associated. Depending of the application this can be beneficial in some use cases and undesirable in other. If the association is required to be refutable, one can use also the interactive Schnorr zero-knowledge proof albeit at the expense of an extra round between the wallet and the verifier. Alternatively on can only use the "implicit zero-knowledge approach" from [3].

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,

**<u>t</u>hen** 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.

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 [44] 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 [44, 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 [44, 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 6. 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.

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 12 in Appendix F. 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 13 in Appendix  $\mathbf{F}$ .

The following proposition provides for another method avoiding raw signatures and can also more be conveniently implemented; it is indicated in Figure 14 in Appendix F.

**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 p 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 p of public key p on the message p of form p where p 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 p.

**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  $\mathcal{H}(C||P) \mod q$  for public key P. From Algorithm 3 it follows  $z \cdot \mathcal{H}(C||P) \mod q$  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  $\mathcal{H}(M') = z \cdot \mathcal{H}(C||P) \mod q$ . 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  $z \neq 1 \mod q$  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 \in \langle G \rangle, on error algorithm stops 2: Compute E' = z \cdot E 3: Compute S_{\mathtt{AB}} = w \cdot E' // compute shared Diffie-Hellman key 4: Convert S_{\mathtt{AB}} to byte array Z_{\mathtt{AB}} 5: Compute K = \mathtt{HKDF}(Z_{\mathtt{AB}},\mathtt{SharedInfo}) // derive MAC-key K 6: Compute \mathtt{HMAC} = E_{\mathtt{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 6. 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.

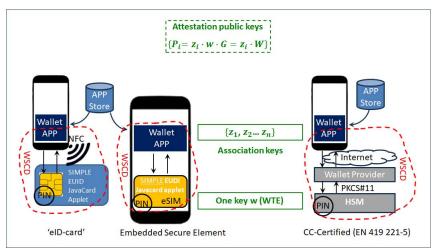


Figure 6. The distributed WSCD

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  $S_{\rm AB}$  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  $S_{\rm AB}$  using the X9.63 Key Derivation Function [6, Section 4.3.3] or the HKDF algorithm [26] as in ISO 18013-5 [32]. As both derivation functions are based on hashing the exchanged Diffie-Hellman key  $S_{\rm AB}$  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 p 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 p randomly generated by the verifier. Then the public key p is bound to this WSCD and is associated to the WTE public key p.

**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. [4]. 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  $0x02||Z_{AB}$  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 [25]. 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 7. It can be considered as the straightforward usage of the WTE/Proof-of-Association logic.

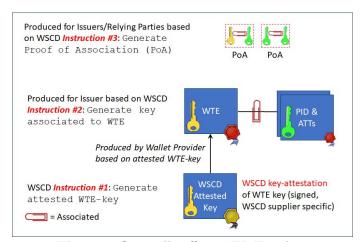


Figure 7. Optimally efficient WTE architecture

#### 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 8 illustrates the role of the ITEs; each attestation issuer gets it owns ITE.

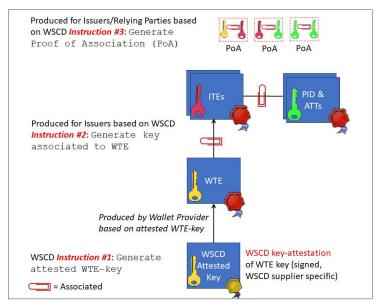


Figure 8. Privacy friendly WTE architecture

### 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 [23], the ISO 23220-3 SAAO [33] and the Android Keystore. For the latter compare Figure 11 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 [14, Article 5a(16b)]. Compare Annex D where this further elaborated on. Figure 9 depicts the PID-bound WTE architecture.

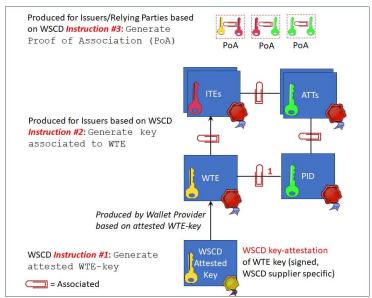


Figure 9. PID-bound WTE architecture

#### 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+ [2,8] or Idemix [9]. 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 [24]. 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. Fabrice Benhamouda, Geoffroy Couteau, David Pointcheval, Hoeteck Wee, Implicit Zero-Knowledge Arguments and Applications to the Malicious Setting, Crypto 2015, LNCS, Volume 9216, Springer, 2015.
- 4. Daniel R. L. Brown, Robert P. Gallant, The Static Diffie-Hellman Problem 2004. See https://eprint.iacr.org/2004/306
- 5. BSI, Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, Request for Comment (RFC) 5639, March 2010.
- 6. BSI, Elliptic Curve Cryptography, TR-03111, version 2.10, 2018-06-01, 2018.
- BSI, Technical Guideline BSI TR-03181, Cryptographic Service Provider 2 (CSP2), TR-03181, Version 0.94, 2023-04-03.
- 8. 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.
- 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.
- 10. https://developer.apple.com/documentation/devicecheck
- 11. https://developer.android.com/training/articles/
   security-key-attestation, https://developer.android.com/training/
   safetynet
- 12. https://source.android.com/static/docs/security/overview/reports/Android-Bootcamp-2016-Android-Keystore-Attestation.pdf
- 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.
- 14. 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.
- 15. European Commission, eIDAS implementing regulation 2015/1502, 8 September 2015 ('minimum technical specifications and procedures for assurance levels for electronic identification').
- 16. European Commission, eIDAS implementing regulation 2024/2977, 28 November 2024 ('person identification data and electronic attestations of attributes issued to European Digital Identity Wallets').
- 17. European Commission, eIDAS implementing regulation 2024/2979, 28 November 2024 ('integrity and core functionalities of European Digital Identity Wallets').
- European Commission, eIDAS implementing regulation 2024/2981, 28 November 2024 ('EUDI wallet certification').
- 19. European Commission, eIDAS implementing regulation 2024/2982, 28 November 2024 ('protocols and interfaces to be supported by the European Digital Identity Framework').

- 20. European Data Protection Board, draft Guidelines on Pseudonymisation, 16 January 2025. See https://www.edpb.europa.eu.
- 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.
- 22. See https://fidoalliance.org/specs/fido-v2.0-ps-20150904/
  fido-key-attestation-v2.0-ps-20150904.html#privacy
- 23. See https://globalplatform.org.
- 24. 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
- 25. ICAO, Doc 2303, Machine Readable Travel Documents, Seventh Edition, part 11: security mechanims for MRTDs, 2015. Available from www.icao.int/.
- Internet Engineering Task Force (IETF), HMAC-based Extract-and-Expand Key Derivation Function (HKDF), Request for Comment (RFC) 5869, May 2010.
- Internet Engineering Task Force (IETF), Edwards-Curve Digital Signature Algorithm (EdDSA), Request for Comment (RFC) 8032, January 2017.
- Internet Engineering Task Force (IETF), Schnorr Non-interactive Zero-Knowledge Proof, Request for Comment (RFC) 8235, September 2017.
- D. Hankerson, A. Menezes and S. Vanstone, Guide to Elliptic Curve Cryptography, Springer, 2004.
- 30. ISO, IT Security techniques Digital signatures with appendix Part 3: Discrete logarithm based mechanisms, ISO/IEC ISO 14888-3, Fourth edition, 2018-11.
- ISO, Information technology Security techniques Methodology for IT security evaluation, ISO/IEC 18045, version 2014-01-15.
- 32. ISO, Mobile driving licence (mDL), ISO/IEC 18013 part 5, version 2021-09. application
- $33.\ \, \mathrm{ISO},\,\mathrm{ISO}/\mathrm{IEC}$  TS WD13 23220-3 issuing phase, draft, version 2024-03-15.
- 34. 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.
- 35. J. Katz, Y. Lindell, Introduction to Modern Cryptography, CRC PRESS, 2008.
- 36. NI-Scy, Epic09 Wallet Trust Evidence, version 0.7.
- 37. NIST, The Keyed-Hash Message Authentication Code (HMAC), FIPS PUB 198-1, July 2008
- 38. NIST, Secure Hash Standard (SHS), FIPS PUB 180-4, August 2015.
- 39. NIST, Digital signature standard, FIPS PUB 186-5. February 3 2023.
- NIST, Recommendation for Key Management Part 1: General, SP 800-57, May 2020.
- 41. https://openid.net/sg/openid4vc/
- C. P. Schnorr, Efficient signature generation for smart cards, Journal of Cryptology 4, 1991.
- 43. D.G. Stinson, Cryptography: theory and practice, CRC press, 1995.
- 44. E.R. Verheul, SECDSA: Mobile signing and authentication under classical "sole control", 16 March 2024. See https://eprint.iacr.org/2021/910.
- StrongBox specification in Section 9.11.2 of https://source.android.com/compatibility/11/android-11-cdd.

# A Android StrongBox key-attestation

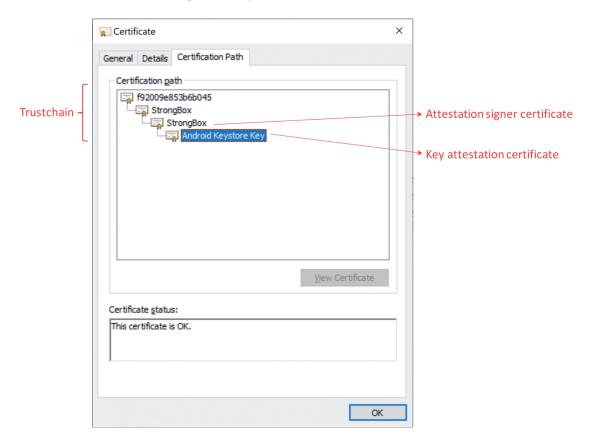


Figure 10. Android StrongBox attested key (leaf)

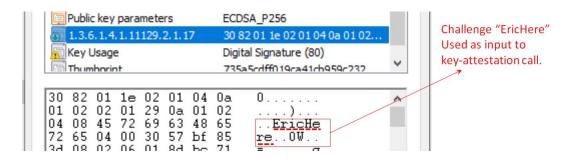


Figure 11. Third party challenge in key attestation

# B Use of WSCD instructions in PID issuance (informative)

Protocol 1 PID issuance

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.

```
Input: -
Output: User PID
1: Wallet requests PID from PID issuer
2: PID 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 \mathcal K containing WTE public key Pub_{	t WTE} and C
6: Wallet requests WP for WTE and sends Attestation Cert {\mathcal K}
7: WP verifies Attestation Cert \mathcal K, if unsuccessful the protocol ends in error
8: WP returns WTE on containing WTE public key Pub_{\mathtt{WTE}} and C
9: Wallet sends WTE on Pub_{\mathtt{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_{\mathtt{WTE}}
12: Wallet calls WSCD for a key associated with Pub_{\mathtt{WTE}}
                                                                    // Instruction #2
13: WSCD returns public key Pub_{	t PID} associated with Pub_{	t WTE}
14: Wallet calls WSCD for a signature on challenge C with Priv_{\text{PID}}
15: WSCD returns PoP
16: Wallet calls WSCD for PoA[Pub_{\text{WTE}}, Pub_{\text{PID}}]
                                                                   // Instruction #3
17: WSCD returns PoA[Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}}]
                                              // proof Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}} are associated
18: Wallet sends Pub, PoP and PoA[Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}}] to (PID) issuer
19: PID Issuer verifies PoP, PoA[Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}}], on failure protocol ends
20: PID Issuer issues PID on public key Pub_{	exttt{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.

#### $\mathbf{C}$ 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 of presentation we leave out the possibility of only selective disclose PID data and 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
- // Proof of Possession 5: Issuer verifies signature with  $Pub_{\scriptscriptstyle{ exttt{PID}}}$
- 6: Issuer uses PID data to form attestation attributes // e.g. diploma
- 7: Issuer requests for attestation public key associated to  $Pub_{\mathtt{WTE}}$
- 8: Wallet calls WSCD for keypair associated with  $Pub_{ t WTE}$  // Instruction #2
- 9: WSCD returns WTE associated public key Pub
- 10: Wallet calls WSCD for PoA[ $Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}}, Pub$ ]
- 11: WSCD returns PoA[ $Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}}, Pub$ ] // proof  $Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}}, Pub$  associated
- 12: Wallet calls WSCD for a signature on challenge C with Priv
- 13: WSCD returns PoP
- 14: Wallet sends Pub, PoP and PoA  $[Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}}, Pub]$  to issuer
- 15: Issuer verifies PoP, PoA[ $Pub_{\mathtt{WTE}}, Pub_{\mathtt{PID}}, Pub$ ], 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 (hereafter: RP) 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 whereby the user only selectively discloses data. The first, respectively second, attestations holds a public key  $Pub_1$ , respectively  $Pub_2$ , with corresponding private key  $Priv_1$ , respectively  $Priv_2$ . Think of the first attestation being the PID and whereby only the resident address is disclosed and second attestation being the mobile driving license whereby only the facial image is disclosed. Compare Section 2.1). For simplicity of presentation we leave out 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-bound attestations  $A_1, A_2$  on public keys  $Pub_1, Pub_2$ Output: Assurance attestations are bound to one WSCD and PID (person)

- 1: RP requests certain attributes // e.g. resident address, facial image
- 2: Wallet sends PID based attestations  $\mathcal{A}_1, \mathcal{A}_2$  to RP whereby selectively disclosing the requested attributes
- 3: RP verifies attestations  $A_1, A_2$  // signatures, seeded hashes etc.
- 4: RP verifies  $\mathcal{A}_1, A_2$  are both WTE & PID based
  - // attestations state to be bound a WSCD and PID
- $5{:}\ \ensuremath{\mathsf{RP}}\ \ensuremath{\mathsf{generates}}\ \ensuremath{\mathsf{challenge}}\ C$  and sends it to wallet
- 6: Wallet signs C with  $Priv_1, Priv_2$  and sends results to RP
- 7: RP verifies signatures with  $Pub_1, Pub_2$  // Proof of possession
- 8: RP requests for proof-of-association  $Pub_1, Pub_2$
- 9: Wallet calls WSCD for  $PoA[Pub_1, Pub_2]$
- 10: WSCD returns PoA[ $Pub_1, Pub_2$ ] // proof  $Pub_1$  and  $Pub_2$  are associated
- 11: Wallet sends PoA[ $Pub_1, Pub_2$ ] to RP
- 12: RP verifies PoA[ $Pub_1, Pub_2$ ], on failure protocol ends
- 13: RP accepts attestations  $A_1, A_2$  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 [43]. 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  $n = m \mod q$  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 "·" 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  $x = DL_G(A)$ . 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  $0 \le x,y < p$  satisfying Equation (2) modulo p. Compare [29]. We denote the zero element (point at infinity) as  $\mathcal{O}$ . For practical implementations one can use one of the NIST curves [39], e.g. P-256 or Brainpool curves [5], e.g. brainpoolP320r1.

Below we describe the working of the ECDSA [39] and ECDH-MAC [32] 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. [43,38]. 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 [29].

```
Algorithm 5 ECDSA signature generation Input: message M, private key d Output: signature (r,s).

1: Compute \mathcal{H}(M) and convert this to an integer e.

2: Select random k \in \{1,...,q-1\}.

3: Compute kG = (x,y) and convert x to integer \bar{x}. // commitment 4: Compute r = \bar{x} \bmod q. If r = 0 go to Line 2.

5: Compute s = k^{-1}(e + d \cdot r) \bmod 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 \mathcal{H}(M) and convert this to an integer e.

3: Compute w=s^{-1} \bmod q.

4: Compute t_1=e\cdot w \bmod q and t_2=r\cdot w \bmod q.

5: Compute X=t_1\cdot G+t_2\cdot D.

6: If X=\mathcal{O} reject the signature.

7: Convert the x-coordinate of X to an integer \bar{x}; compute v=\bar{x} \bmod 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 [32]. It is based on a Message Authentication Code (MAC) on a message M generated using a conventional MAC Algorithm. ISO 18013-5 [32] stipulates

the use of the HMAC algorithm [37]. This MAC is based on a key K of type byte array; the MAC computation is denoted by  $\mathrm{HMAC}(K,M)$ .

Key K is derived from a byte array representation  $Z_{\rm AB}$  of the Diffie-Hellman key  $S_{\rm AB}$  shared between the signer and verifier and a byte array SharedInfo. The latter holds additional information shared between the signer and the verifier. In [32] the shared information includes a session transcript. For this key derivation, ISO 18013-5 [32] stipulates the use of the HKDF algorithm [26]. In general this algorithm takes as input a hash function, a byte array key IKM holding input material, an optional salt byte array, an optional info byte array and an parameter L representing the number of output bytes. ISO 18013-5 stipulates using the SHA256 hash function from [38], no salt, letting IKM to be equal to  $Z_{\rm AB}$ , letting the info byte array to be SharedInfo and L=16. We denote this instantiation of HKDF by HKDF( $Z_{\rm AB}$ , SharedInfo).

ISO standard 18013-5 [32, 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 [32]. The notation used is also in line with Section 4.3 of BSI publication TR-03111 [6] 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 \in \langle G \rangle, on error algorithm stops
```

- 2: Compute  $S_{\mathtt{AB}} = d \cdot E$  // compute shared Diffie-Hellman key
- $3{:}$  Convert  $S_{{\mbox{\tiny AB}}}$  to byte array  $Z_{{\mbox{\tiny AB}}}$
- 4: Compute  $K = \text{HKDF}(Z_{AB}, \text{SharedInfo})$  // 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 \cdot D // shared Diffie-Hellman key

2: Convert S_{AB} to byte array Z_{AB}

3: Compute K = \text{HKDF}(Z_{AB}, \text{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  $Z_{\rm AB}$  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 E supports the second case, i.e. returning the full Diffie-Hellman key E supports the second case, i.e. returning the full Diffie-Hellman key E supports full Diffie-Hellman. With saying that private key E 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 E in the second case.

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. [4]. 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.

It is fruitful to have techniques allowing a Prover to prove to a Verifier that a certain private key exists and that he have possession of this private key. For this we first deploy the technique of non-interactive zero-knowledge proofs of knowledge (ZPK). In this section we recall the ZKPK techniques of Schnorr [42]. To this end, let  $d \in \mathbb{F}_q^*$  be a private key with public key  $D_0 = d \cdot G_0$ . Here  $G_0$  can be any generator of the group G but will typically be the standard generator G. Let n be an integer and let  $G_i$  for  $i = 1, \ldots, n$  be generators of the group G, i.e. they can be different from the standard generator G. Let  $D_i = d \cdot G_i$  for  $i = 1, \ldots, n$ . That is, each  $D_i$  can be considered a public key with respect to the generator  $G_i$  and private key d. Note that these public keys have a particular form: they all have the same private key as public key  $D_0$ . That is, the following statement holds:

$$\exists d \in \mathbb{F}_q^* : D_0 = d \cdot G_0 \text{ and } D_i = d \cdot G_i \quad i = 1, \dots, n.$$
 (3)

The simplest way to prove to a party that Statement (3) holds is to reveal private key d. However, we want to convey this knowledge to the party without revealing any information on d, i.e. in zero-knowledge. The data structure  $\mathcal{ZKP}$  generated in Algorithm 9 achieves this. This algorithm is based on the Schnorr non-interactive zero-knowledge proof [42] using the Fiat-Shamir heuristic [21] similar to RFC 8235 [28]. In Algorithm 9 we also bind this proof to a challenge N (byte array) which is convenient in certain situations.

Algorithm 9 Generation of zero knowledge proof on Statement (3) Input: Public key  $D_0 = d \cdot G_0$ , private key d, generators  $G_i$ , public keys  $D_i$  for i = 1, 2, ..., n, Challenge N (byte array) Output:  $\mathcal{ZKP}[(G_i, D_i)_{i=0}^n, N]$ 

```
1: For i=1,\ldots,n verify that D_i=d\cdot G_i, on failure return error 2: For i=0,\ldots,n convert G_i to byte array \bar{G}_i and form \bar{G}=\bar{G}_0||\ldots||\bar{G}_n 3: For i=0,\ldots,n convert D_i to byte array \bar{H}_i and form \bar{D}=\bar{D}_0||\ldots||\bar{D}_n 4: Select random k\in\{1,\ldots,q-1\} 5: For i=0,1,\ldots,n compute G_i'=k\cdot G_i 6: For i=0,1,\ldots,n convert G_i' to byte array \bar{G}_i' 7: Form \bar{G}'=\bar{G}_0'||\ldots||\bar{G}_n' 8: Compute byte array \mathcal{H}(\bar{G}||\bar{G}'||\bar{D}||N) and convert it to an integer r 9: If r=0 then go to Line 4. 10: Compute s=k+r\cdot d \bmod q 11: If s=0 then go to Line 4 12: Return (r,s)
```

The following algorithm specifies the verification of the proof generated by Algorithm 9. This is a classical result from [42,21].

```
Algorithm 10 Verification of zero knowledge proof on Statement (3) Input: \mathcal{ZKP} = (r,s) Output: Acceptance of rejection of Statement (3)

1: For i=0\ldots,n verify that G_i,H_i\in\mathbb{G} and that G_i is a group generator, on failure Return False.

2: Verify that r\in\{1,2^{8\cdot|q|}-1\} and s\in\{1,q-1\}, on failure Return False.

3: For i=0,\ldots,n convert G_i to byte array \bar{G}_i and form \bar{G}=\bar{G}_0||\ldots||\bar{G}_n

4: For i=0,\ldots,n convert D_i to byte array \bar{D}_i and form \bar{D}=\bar{D}_0||\ldots||\bar{D}_n

5: For i=0,\ldots,n compute G_i'=s\cdot G_i-r\cdot D_i

6: If any G_i'=\mathcal{O} Return False.

7: For i=0,\ldots,n convert G_i' to byte array \bar{G}_i' and form \bar{G}'=\bar{G}_0'||\ldots||\bar{G}_n'

8: Compute byte array \mathcal{H}(\bar{G}||\bar{G}'||\bar{D}||N) and convert it to an integer v.

9: If v=r accept Statement 3 otherwise reject it
```

We make some notes on Algorithm 9:

1. By the nature of Schnorr based proofs of knowledge there is a negligible probability (in the order or  $2^{-8\cdot|q|}$ , i.e.  $2^{-256}$  in the context of the NIST curve P-256, that Algorithm 10 is erroneously successful. For simplicity we do not further stipulate that in the algorithms.

- 2. For n=0 Algorithm 9 generates a proof knowledge (also known as a proof of possession) of private key d. This proof closely resembles the ECSDSA (Schnorr signature) from [6,30,42].
- 3. The proof generated by Algorithm 9 provides a non-refutable proof that Statement (3) holds. There might be situations where the prover wants to convince a verifier that Statement (3) holds in a way refutable by the prover, i.e. can be denied by the verifier on a later moment. In these situations, one can use the original Schnorr interactive proof from [42]. Here the integer r appearing in Algorithms 9 and 10 is formed as a challenge from the prover. The proof knowledge is then refutable as verifier could have generated it himself, cf. [42]. Alteratively, one can use the "implicit zero-knowledge approach" from [3].

# $F.\ EXAMPLE\ APPLICATIONS\ OF\ PROPOSITION\ 3.5\ (INFORMATIVE)$

# F Example applications of Proposition 3.5 (informative)

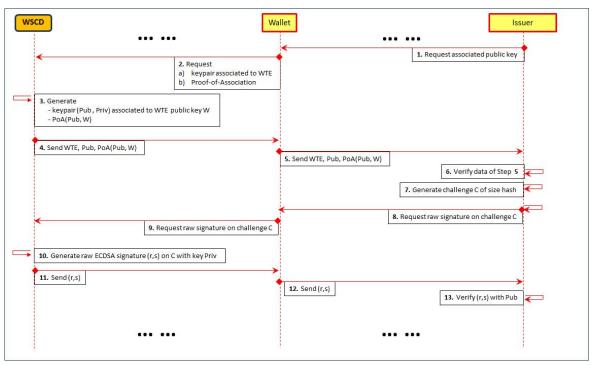


Figure 12. Straightforward application of Proposition 3.6

## F. EXAMPLE APPLICATIONS OF PROPOSITION 3.5 (INFORMATIVE)

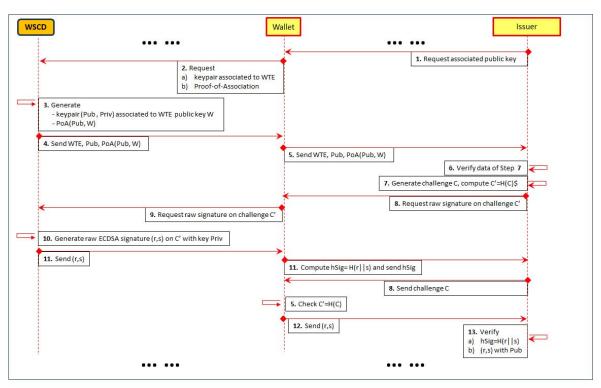


Figure 13. Demonstrating raw ECDSA signing without actually doing it

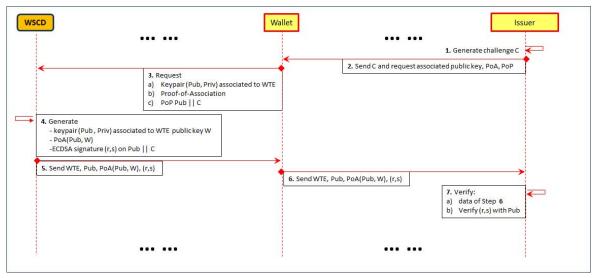


Figure 14. Application of Proposition 3.7

## G ASN.1 format for Proof of Association (informative)

Below we have specified a proposal for the proof of association in ASN.1 format, cf. [34]

```
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.
```

## H Self Generated Verifiable Pseudonyms (SGVP)

In this annex we introduce the concept of Self Generated Verifiable Pseudonyms (SGVPs), specify its cryptographic details and sketch it practical use. In Annex H.1 we first specify the SGVP building blocks. As a minimal implementation only requires Diffie-Hellman support, SGVP is implementable on all WSCA/WSCD architectures discussed in Section 1. We next provide sketches in Annex H.2 on how the SGVP building blocks can be used in practice. This illustrates that various implementations are possible as well as additional privacy enhancing functionalities.

#### H.1 SGVP building blocks

As before we use the (mathematical) notation and terms from Annex E. Within the concept of Self Generated Verifiable Pseudonyms (SGVP) each user has a pseudonym secret  $x \in_R \mathbb{F}_q^*$ . This secret x is maintained in the wallet WSCD or WSCA and there is a mechanism allowing the distribution of this secret to other WSCDs/WSCAs of the user. This allows for persistent pseudonyms over all the wallets of the user. In Annex H.2 we discuss some options for the distribution of the pseudonym secret.

The user also has a PID-bound attestation holding an encrypted pseudonym secret x which takes a particular form. This form consists of  $(H, x \cdot H)$  where H is a generator of the group  $\mathbb{G} = (\langle G \rangle, +)$ . For simplicity we assume that this attestation is the PID itself. The following two algorithms specify the generation of the pseudonym secret and an encrypted pseudonym secret which are quite straightforward.

```
Algorithm 11 Generation of user pseudonym secret
```

Input:

Output: Reference to  $x \in_R \mathbb{F}_q^*$  in WSCA/WSCD

- 1: Wallet App instructs WSCA/WSCD to generate random integer  $x \in \mathbb{F}_q^*$ .
- 2: WSCA/WSCD returns reference to user pseudonym secret x

#### Algorithm 12 Generation of encrypted pseudonym secret

Input: reference to user pseudonym secret  $\boldsymbol{x}$ 

Output: encrypted pseudonym secret  $(H, x \cdot H)$ 

- 1: Wallet App chooses generator  $E_1$  of group  $\mathbb G$  // e.g., standard generator G
- 2: Wallet App instructs WSCA/WSCD to compute  $E_2 = x \cdot E_1$
- 3: Return pair  $(E_1, E_2)$

We remark that the computation of  $E_2$  in Step 2 of Algorithm 12 constitutes a full Diffie-Hellman operation we discussed in Annex E and which is available in almost all cryptographic hardware most notably the native mobile cryptographic hardware. This implies the Algorithm 12 is implementable on such cryptographic hardware.

An important SGVP building block is that any party can *randomize* an encrypted pseudonym secret resulting into a copy thereof that holds the same

pseudonym secret but that is not linkable to the original. This implies that issuers can place different and non-linkable encrypted pseudonym secrets in different PIDs/attestations of the holder that still contain the same pseudonym secret. Such PIDs/attestations then all lead to compatible pseudonyms at relying parties where the PIDs/attestations used do not contain any linkable information.

There are two versions of this randomization algorithm. The first version (Algorithm 13) is basic and simply randomizes an encrypted pseudonym secret whereas the second version (Algorithm 14) is more advanced and practically relevant. This algorithm additionally provides a publicly verifiable proof that the randomized encrypted secret contains the same secret as the original which is relevant in some situations.

**Algorithm 13** Randomization of encrypted pseudonym secret Input: encrypted pseudonym secret  $(E_1, E_2)$  Output: randomized version  $(E'_1, E'_2)$ 

```
1: Choose random k in \mathbb{F}_q^*

2: Let E_1^{'}=k\cdot E_1 and E_2^{'}=k\cdot E_2

3: Return pair (E_1^{'},E_2^{'})
```

That the output  $(E'_1, E'_2)$  of Algorithm 13 contains the same pseudonym secret as the input  $(E_1, E_2)$  is a simple verification. That the output is not linkable to the input is due to the hardness of the Decision Diffie-Hellman (DDH) problem discussed in the beginning of Annex E. Compare cf. [35, Theorem 10.20].

Algorithm 14 Provable randomization of encrypted pseudonym secret Input: encrypted pseudonym secret  $(E_1, E_2)$  Output: randomized version  $(E'_1, E'_2)$  plus proof  $\mathcal{ZKP}$ 

```
1: Choose k \in_R \mathbb{F}_q^*

2: Let E_1^{'} = k \cdot E_1 and E_2^{'} = k \cdot E_2

3: Generate \mathcal{ZKP} = \mathcal{ZKP}[(E_1, E_2), \left(E_1^{'}, E_2^{'}\right), \emptyset] // Annex E Algorithm 9; d = k

4: Return (E_1^{'}, E_2^{'}) and \mathcal{ZKP}
```

Algorithms 13 and 14 are typically ran in software so are efficiently implementable. Verification of the ZKP from Algorithm 14 follows Algorithm 10. If this verification is successful then the verifier is assured there exists an integer k such that simultaneously holds  $E_1^{'}=k\cdot E_1$  and  $E_2^{'}=k\cdot E_2$  implying that  $(E_1^{'},E_2^{'})$  holds the same pseudonym secret as  $(E_1,E_2)$ .

In Algorithm 16 we specify how a pseudonym is derived based on a pseudonym secret and a *Pseudonymisation Domain (PD)*, i.e. any string agreed between the wallet/user and relying party to derive pseudonyms from. This term is taken from the guidelines on pseudonymisation of the European Data Protection Board (EDPD) [20]. We do not further discuss these guidelines but remark that SGVP pseudonymisation meets all requirements specified in these guidelines.

In wallet context, a Pseudonymisation Domain is typically envisioned to represent a relying party like an URL, a relying party service name or a cluster of relying parties or services. Such a cluster could be ad-hoc but could also represent a sector of relying parties, e.g. the financial, medical, or social sector.

Algorithm 17 additionally provides a publicly verifiable proof that this pseudonym is correctly formed.

Both Algorithms 16 and 17 require a cryptographic mapping a string into an element of the group  $\mathbb{G}$ . In practice this group is based on an elliptic curve group of prime order q over a finite field  $\mathbb{F}_p$  for which an example of such a map is specified in Algorithm 15. This algorithm uses the HDKF key derivation function [26] we already discussed on page 37 in Section  $\mathbb{E}$  in the context of ISO 18013-5 [32]. Here we use a somewhat different and atypical instantiation based on the SHA256 hash function from [38] and no salt or info byte array. This means that this HKDF instantiation only takes as input the input key material (IKM) and an parameter L representing the number of output bytes. We denote this instance by HKDF(IKM, L).

#### Algorithm 15 MPD

Mapping of a Pseudonymisation Domain PD to a point P in group  $\mathbb{G}$ 

```
1: Represent string PD as byte array PD
2: Set i=0
3: repeat
4: Represent i as byte array I
5: Compute \mathsf{HKDF}(PD||I,|p|+8) and represent as integer f
6: i=i+1
7: until A non-zero point P in \mathbb G exists with f as x-coordinate.
8: Return point P in \mathbb G with f as x-coordinate even.
```

Recall that |p| in Step 5 of Algorithm 15 denotes the size in bytes of the prime number p defining the finite field  $\mathbb{F}_p$ . The additional 8 bytes requested in Step 5 follows BSI publication TR-03111 [6] and avoid statistical bias in the HKDF output.

```
Algorithm 16 Generation of SGVP pseudonym
```

Input: pseudonym secret x, Pseudonymisation Domain PD

Output: pseudonym Ps

```
1: Wallet App maps PD to element P_1 in group \mathbb G using Algorithm 15 2: Wallet App instructs WSCA/WSCD to compute P_2=x\cdot P_1 3: Return Ps=(P_1,P_2)
```

## Algorithm 17 Provable generation of SGVP pseudonym

Input: pseudonym secret x, encrypted pseudonym secret  $(E_1, E_2)$ , Pseudonymisation Domain PD

Output: pseudonym Ps plus proof  $\mathcal{ZKP}$ 

```
1: Wallet-APP maps PD to element P_1 in group \mathbb G using Algorithm 15 2: Wallet APP instructs WSCA/WSCD to compute P_2=x\cdot P_1 3: Wallet APP instructs WSCA/WSCD to compute \mathcal{ZKP}=\mathcal{ZKP}[(E_1,E_2),(P_1,P_2),\emptyset] \hspace{1cm}//\hspace{1cm} \text{Annex E Algorithm 9; } d=x 4: Return Ps=(P_1,P_2) and \mathcal{ZKP}
```

Public verification of the ZKP from Algorithm 17 follows Algorithm 10. If this verification is successful then the verifier is assured there exists an integer x' such that simultaneously holds

$$E_2 = x' \cdot E_1 \tag{4}$$

$$P_2 = x' \cdot P_1 \tag{5}$$

From Equality (4) it now follows that x' is equal to the pseudonym secret x. It then follows from Equality (5) that pseudonym  $Ps = (P_1, P_2)$  is well formed, i.e. uses the same pseudonym secret as in encrypted pseudonym secret  $(E_1, E_2)$ .

As we remarked above, Step 2 in Algorithm 16 and Algorithm 17 is implementable in most cryptographic hardware and all native mobile cryptographic hardware. However, Step 3 of Algorithm 17, i.e. the generation of the Schnorr zero knowledge proof, is typically not implementable in cryptographic hardware but could be implementable in the WSCA outside the WSCD. Alteratively, one can use the "implicit zero-knowledge approach" from [3] which only requires Diffie-Hellman support. In this way the proof that a relying party receives on pseudonym correctness is not transferable which can be considered a feature too.

#### H.2 Sketches of how SGVP pseudonyms can be used in practice

There are several ways how the SGVP building blocks from Annex H.1 can be used in practice. In a simple implementation the wallet/user generates a pseudonym secret x using Algorithm 11 and a first encrypted pseudonym secret  $(E_1, E_2)$  using Algorithm 12 which could be based on the standard generator G. The PID provider places the encrypted pseudonym secret  $(E_1, E_2)$  as an attribute in the first PID. With this encrypted pseudonym secret attribute, the wallet/user can now generate SGVP pseudonyms to relying parties following Algorithm 17 based on any Pseudonymisation Domain (PD) as explained in Annex H.1. As explained, this Pseudonymisation Domain can be any string agreed between wallet/user and the relying party, but is typically a string representing the relying party URL, a relying party service name, or a cluster of relying parties or services. Such a cluster could be ad-hoc but could also represent a sector of relying parties, e.g. the financial, medical, or social sector. As indicated on page 2.1 in Section 2.1, the SGVP-pseudonyms can be cryptographically linked to other PID-bound attestations by using proofs of association between these attestations and PID.

To avoid that the PID (or rather the encrypted pseudonym secret therein) allows linking, the user presents different PIDs for different relying parties holding randomized versions of the encrypted pseudonym secret following Algorithm 14. It seems most efficient if these randomized versions of the encrypted pseudonym secret are generated by the PID provider based on the first PID. The PID provider then sends the correctness proofs, i.e. the ZKP data from Algorithm 14 to the wallet/user. These proofs then enable the wallet/user verification that the randomized encrypted pseudonym secrets in PID copies are indeed correctly

formed, i.e. hold the same pseudonym secret as the first encrypted pseudonym secret. Alternatively, these randomized versions of the encrypted pseudonym secret can also be generated by the wallet/user and then sent to the PID provider together with the correctness proofs. These proofs then enable the PID provider to verify that the randomized encrypted pseudonym secrets to be included in the PID copies are correctly formed.

We further note that the SGVP concept can also provide for additional pseudonymous functionality. For instance, under control of the user it allows a group of relying parties to link their SGVP-pseudonyms without the relying parties getting access to the pseudonyms of the other relying parties. This can be simply accomplished by letting the wallet/user deploy the same PID (and thus the same encrypted pseudonym secret) in the generation of the SGVP pseudonym for this group of relying parties. In this way the PID (or only the encrypted pseudonym secret therein) allows the relying parties to link their SGVP-pseudonyms. We emphasize that only the encrypted pseudonym secret needs to be disclosed from the PID to the relying parties, but in some cases it can be beneficial to also disclose the user last name to allow for easy user support.

Instead of placing encrypted pseudonym secrets in the PID, these could be placed in specific, PID-bound SGVP attestations provided by a specific SGVP attestation provider. To allow for persistent pseudonyms over all the wallets of the user, the wallet/user could make an encrypted backup of the pseudonym secret x and restore that into new wallet of the user. The pseudonym secret could also be maintained by the wallet provider, the PID provider, an SGVP attestation provider or a backup/revovery service provider.