Anonymous Credentials from ECDSA and Zero-Knowledge Proofs

My Name Here
My Affiliation

Your Name Here
Your Affiliation

Their Name Here
Their Affiliation

Our Name Here
Our Affiliation

August 9, 2024

Abstract

Issuing digital identity documents to citizens at large scale so far has failed despite many attempts, mainly because citizens did not (want to) buy the hardware necessary to securely manage and use cryptographic keys. Today, for the first time in history, many users do have mobile phones that contain a piece of secure hardware that is able to manage such keys and their secure use part of by the operating system. This technology is generally referred to as Passkeys. Similarly, there are hardware and software systems available that allow one to build systems to securely issue digital credentials. While there are many standards for cryptographic (signature) schemes for such systems, the most widely used is ECDSA, and for Passkeys that is the only scheme that is currently supported. The use of identity and personal information is very sensitive and requires sufficient protection, in particular in connection with any electronic media. That is, a digital identity scheme must allow users to have different identities with different parties and be able to selectively reveal attributes from a credential obtained. The goal of this document is to describe an identity system that provides privacy and can be implemented with the existing hardware and operating systems and the algorithms that (secure) hardware supports. I.e., it can be realized solely in the application space and can be rolled out without requiring any change in hardware, firmware, or operating systems. In particular, the document describes a digital identity and credential scheme that makes the following guarantees:

- a user's identity is bound to a passkey device that supports (only) ECDSA, and secret key material never leaves the hardware,
- an identity provider can issue credentials to a user's identity, the credentials containing a number of attributes (for ECDSA),
- users can present a credential to a relying party while revealing only a selection of its attributes (and where the presentation transcript can not be linked to the original credential issuance transaction unless a unique attribute was revealed). This presentation is realized with zero-knowledge proofs.

For simplicity of the exposition and to initiate an iterative discussion, this document does not (yet) describe 1) how to present predicates over attributes, 2) describes only the very simple case of one issuer (and not a system where there are several issuers and users are not fully identified to all of them, and 3) does not consider revocation. These functionalities can be added at a small increase in complexity. The scheme is described at a high-level only, in particular, 1) a number of cryptographic implementation details are left out and 2) no security proofs are given. Both can (quickly) be done if discussions become more concrete.

TODO: update overall text once we've updated the rest of the document to have the things the original text said we didn't have :) lol

1 Introduction

The purpose of this document is to outline a basic construction that 1) is implementable in short time and 2) addresses the main requirements as we understand them (foremost hardware binding, efficiency, practical deployability, and privacy). It does not yet offer all features that one would expect such as revocation and

supporting multiple issuers where the user is not fully identified to the issuers(rene: Do we mean issuer hiding with this sub-sentence?). While these features can be added, we prefer to follow an iterative approach towards the final specification. Solutions adding these additional features are well-researched and can be added to the draft at a later point. Thus the document serves as a way forward to 1) agree on the core requirements, 2) what specification(s) of the scheme is needed in 3) what time frame, and 4) forming a team to work on this specification.

With digital credentials (anonymous or not), the issue of preventing a user from sharing their credential is inherently difficult due to digital data being easy to copy. While we can mitigate this through various strategies (such as adding some biometric aspect, requiring for presenting multiple credentials belonging to a single person, or disincentivizing sharing through risk or policies), we give a method of adding a physical aspect that requires transferring the physical component to share the credential. Specifically, we require credential presentation to require interaction with trusted hardware on the user's phone, and therefore require a user to share their phone in order to allow others to use their credentials.

Construction Outline. Working with the constraint that secure elements on phones must be used as-is (i.e., the passkey technology supporting ECDSA signatures), we give a straightforward way of integrating the secure element with any credential.

The general idea is as follows. Let PK_D be the public key of a secure element of a user's phone. When the user is issued a credential from the Identity Provider (henceforth also called issuer), the user's device's public key PK_D is included as an attribute in its credential. The credential is then signed by the issuer with PK_I being the public key of the issuer.

Now, when a user wishes to present his credential to the Relying Party, he will prove that he is in possession of the credential as well as the secure element containing the secret key SK_D corresponding to the public key PK_D in his credential. This can be done by having the Relying Party send a unique (unpredictable) string *ctxt* and the user returning a signature on that value. The signature is produced by his phone's secure element, and is verifiable under PK_D. The string *ctxt* should bind the presentation to the context where the credential is presented and it must be ensured that it has not been used before. To create the presentation without requiring the user to reveal PK_D to the Relying Party, the user uses generic zero-knowledge techniques to prove they are 1) in possession of a signature on *ctxt* that verifies under an undisclosed attribute of their credential (namely the key PK_D), 2) in possession of a signature on that credential that verifies under the issuer's public key PK_I, and 3) if an additional attribute is revealed, that attribute is also contained in the credential.

Though there exist specific signature schemes that were designed with discrete logarithm proof systems (such as BBS+ [BBS04, CL04] or CL [CL01, CL04] signatures), those are not compatible with deployed hardware (and in particular Passkeys). Fortunately, advances in modern generic zero-knowledge proof systems have made generic approaches efficient enough to be competitive with such specialized protocols for use in practice. Most importantly, they are compatible with deployed hardware and thus can be rolled out immediately.

Having said this, we still need to consider how the attributes are encoded into the message signed by the issuer to achieve a very practical system. Let us explain. A cryptographic signature scheme (and ECDSA in particular) takes as input a message in the format of a binary string, hashes it to obtain a binary string which is interpreted as an algebraic value that is then used in the signing and verification algorithms. The system sketched above needs to make some statements about attributes (i.e., the user's key PK_D and the revealed attributes) and hence needs to consider the encoding of the attributes into the binary string that is hashed and signed. In practice such a mapping from structured data (such as a credential) is done using encoding scheme such as JSON. While such encoding have advantages in terms of developer friendliness, they will make the zero-knowledge proofs slow and may lead to interoperability issues. It is thus important that in the construction, a more low level encoding is used. That does not mean that at a higher level API encodings such as JSON cannot be used, it just means that we need to specify a low level API encoding for the credential format that is less "bloated."

We note that while the described scheme is tailored towards ECDSA, the reason being that it is the

scheme that is prevalent in the hardware in the population, the approach itself is "crypto-agile": it can be instantiated with other cryptographic schemes including ones that are post-quantum secure. In fact, the approach presented here seems to be the best one for a post-quantum credential system. In that sense, the scheme proposed is "post-quantum ready."

Document Organization. We review basic preliminaries in Section 2 before going into the construction in Section 3. In Section 4 we prove security of our general protocols given in Section 3. In Section 5 we give specifications for how to instantiate our overall scheme. Finally, in Section 6, we give how to expand our protocol to support desired properties such as revocation and multi-issuer credentials.

2 Preliminaries

2.1 ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analog of the Digital Signature Algorithm (DSA) and is standardized by NIST [oST23]. We let $\mathbb{G}$ denote the elliptic curve group, q the order of the group, and G a generator of the group. ECDSA additionally makes use of a hash function H, which we instantiate with SHA-256.
A public and private key pair for ECDSA is generated as:

1. Uniformly choose a secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q^*$.
2. Calculate the public key as $\mathsf{pk} = \mathsf{sk} \cdot G$.
3. Output $(\mathsf{pk}, \mathsf{sk})$.

To sign a message m under a secret key sk, the signer does:

1. Uniformly choose an instance key $r \leftarrow \mathbb{Z}_q^*$.
2. Calculate $R = r \cdot G$ and let r_x be the x-coordinate of R, modulo q.
3. Calculate
$$s = \frac{H(m) + \mathsf{sk} \cdot r_x}{r}$$
4. Output the signature as (s, r_x).

To verify a signature (s, r_x) on a message m under a public key pk, the verifier does as follows:

1. Calculate
$$R' = \frac{H(m) \cdot G + r_x \cdot \mathsf{pk}}{s}$$
and let $r_x{}'$ be the x-coordinate of R', modulo q.
2. Output 1 if and only if $r_x{}' = r_x$. Output 0 otherwise.

2.2 Zero-Knowledge Proofs

(ey: recall ZKPs in general before going into the specific flavor of ZKP we're going to use)

Definition 1. *(ey: recall a zkp)*

Interactive Oracle Proofs (IOP) are a type of zero-knowledge proof (ZKP) system that aims to provide efficient and scalable proofs with strong security guarantees. IOPs enable a verifier to be convinced after having only queried the proof at random locations rather than reading the entire proof and require interaction between a prover and a verifier. Much work has been done in this space minimizing the computational and communication overhead typically associated with zero-knowledge proofs, making it well-suited for practical applications, especially in environments such as credentials.

2.2.1 Choice of Zero-Knowledge Proof System for Our Application.

For the ZKP in our application, we also make use of a polynomial commitment scheme to commit to the witness[1] for the proof. A polynomial commitment scheme allows a prover to commit to a polynomial and later prove that the evaluation of this polynomial f on a specific point is correct, i.e. $f(x) = v$ for public values v and x. In our context, the function is defined by the witness for our zero-knowledge proof. Evaluating f in certain points (leading to predictable outcomes) corresponds to the verification of the ECDSA signature by the issuer and the one generated by the user's device as specified above. Ligero [AHIV17] is a zero-knowledge argument of knowledge protocol based on the MPC-in-the-head method of zero knowledge proofs and only relies on collision-resistant hash-functions. We suggest the use of the Ligero as polynomial commitment scheme. This is because Ligero offers proof sizes that are smaller than the witness (and the verified function). Ligero is a well-studied, mature cryptographic protocol that has been implemented multiple times before.

TODO: Combine text from here to above text

One can use an arithmetic circuit based proof system to implement the zero-knowledge proof system used in the credential presentation. One modern, generic recipe is to choose a IOP system and a compatible polynomial commitment scheme. While there are many possible choices, one specific choice is to use a sumcheck-based protocol for the IOP, and a hash-based polynomial commitment scheme. Sumcheck, for example, when used in a circuit-based protocol, has a linear-time prover, and a careful implementation can achieve a very small constant overhead. In particular, the sumcheck prover does not use a fast fourier transform (FFT) internally like many other proof systems. This approach is taken by a recent unpublished work by one of the authors. Their Hyrax implementation uses a discrete-log based commitment scheme and a bulletproof to prove the opening. For certain small proofs, this solution is performant. The Ligero polynomial commitment has a proof size that grows asymptotically as square-root of the witness size, but has a larger implementation constant. However, Ligero only requires hashing (and no other operations such as using elliptic curves), and the circuit used when creating a presentation exceeds the cross-over point (where Ligero becomes more performant than the first approach mentioned)—therefore, they chose to use it instead.

A paper describing the full system is planned to be submitted to a conference, but this work was done under a company arrangement, and thus must be internally reviewed first.

3 Construction

In this section we give the basic construction for an issuer, a user, and a relying party. It has two phases: 1) the issuer issues a credential to a user and 2) the user presents the credential to a relying party. The second phase can be repeated many times to different relying parties.

We first describe the basic setting and notations of our protocols. Then, we expand on the construction outlined in Section 1, which requires a user to be in possession of their phone when presenting a credential by incorporating the secure element of a user's phone into the credential presentation procedure.

TODO: Write the following sections out in a way that allows actually writing the proof lol

3.1 Credential Issuance

For an Identity Provider to issue a credential to a user, the two first establish a session. The user uses the public key of its secure element PK_D to authenticate the session. During this session, the user and Identity Provider perform any steps necessary to establish if a credential should be issued and what attributes should be included in the credential. Additional means might be required to establish that PK_D is indeed the public key of the specific user's device, such as being personal present at an office and using the device in the presence of supervising personnel. In Figure 1 we represent this process by "establish session" and "establish attributes" and give the the rest of the outline of the credential issuance protocol.

[1] Recall that "witness" in the context of a zero-knowledge proof of knowledge refers to the (typically secret) information that the prover knows and can use to convince the verifier that a statement is in the language. If the statement is that there exists a signature on a message m that verifies under a public key pk, the witness could be the signature.

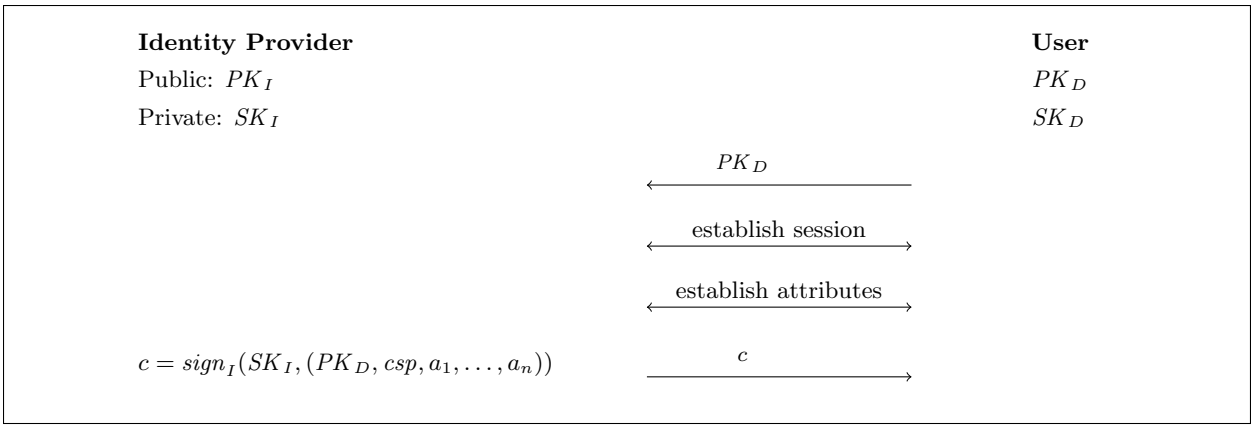

Figure 1: Credential Issuance.

Once the parties have determined which attributes should be included in the credential, the Identity Provider will then issue a credential of type *csp* with these attributes $a_1, \ldots, a_n$ to user that is bound to the user's device key PK_D by computing the signature on these values and then sending this signature to the user's device. Specifically, the credential is computed as $c = sign_I(SK_I, (PK_D, csp, a_1, \ldots, a_n))$, where $sign_I$ is the signature scheme of the Identity Provider (e.g., ECDSA), SK_I is the Identity Provider's signing key, and the message signed is the concatenation of the user's device's public key PK_D with the credential specification and attributes. Note that a valid credential's specification *csp* will be required to define the number of attributes in the credential and each attribute's length.

3.2 Credential Presentation

For a user to present a credential to a Relying Party, the two must first establish a session, e.g., using the WebAuthn standard. We note that the user is able and is very likely to be pseudonymous during this interaction, as WebAuthn generates a fresh public key for every url/origin. However, the protocol presented here can also be used in sessions that are established differently. Regardless of how a session is established, the credential presentation should be bound to this session. To this end, we require the user and Relying Party to jointly define a session context *ctxt* that is unique to the interaction. This context can for instance be defined as the transcript of the key exchange protocol used to setup the end-to-end encryption between the user and the Relying Party, or as the session key that was established, as long as it is ensured that *ctxt* is a unique value with very high probability. Once the session and context *ctxt* have been established, the user produces a signature on *ctxt* using its device's secure element's public key as $s = sign_D(SK_D, ctxt)$, where $sign_D$ is the signature scheme of the secure element. The user then proceeds to produce a proof of its credential as follows.

At a high level, the user will be proving to the Relying Party that it knows values $c, a_1, \ldots, a_n, PK_D$, and s corresponding to the statement "Given the Issuer's public key PK_I, a credential specification *csp*, and a context *ctxt*, there exists a credential c with attributes $a_1, \ldots, a_n$, public key PK_D, and signature s such that (1) c is a valid signature on $PK_D, csp, a_1, \ldots, a_n$ that verifies under PK_I and (2) s is a signature on *ctxt* that verifies under PK_D". The user begins by producing a polynomial commitment com to the function $f(PK_I, csp, ctxt) = ver_I(PK_I, (PK_D, csp, a_1, \ldots, a_n), c) \wedge ver_D(PK_D, ctxt, s)$ where ver_I, ver_D are the verification algorithms for signatures produced by the Identity Provider and device's secret element, respectively, and secret values PK_D, c, $a_1, \ldots, a_n$, and *ctxt* define f. Note this function is a rewriting of the statement the user is trying to prove. The user then sends this commitment com, the public key of the Identity Provider PK_I, and its credential specification *csp* to the Relying Party. Finally, the two parties execute an IOP, with the user acting as the prover and the Relying Party as the verifier, for the statement

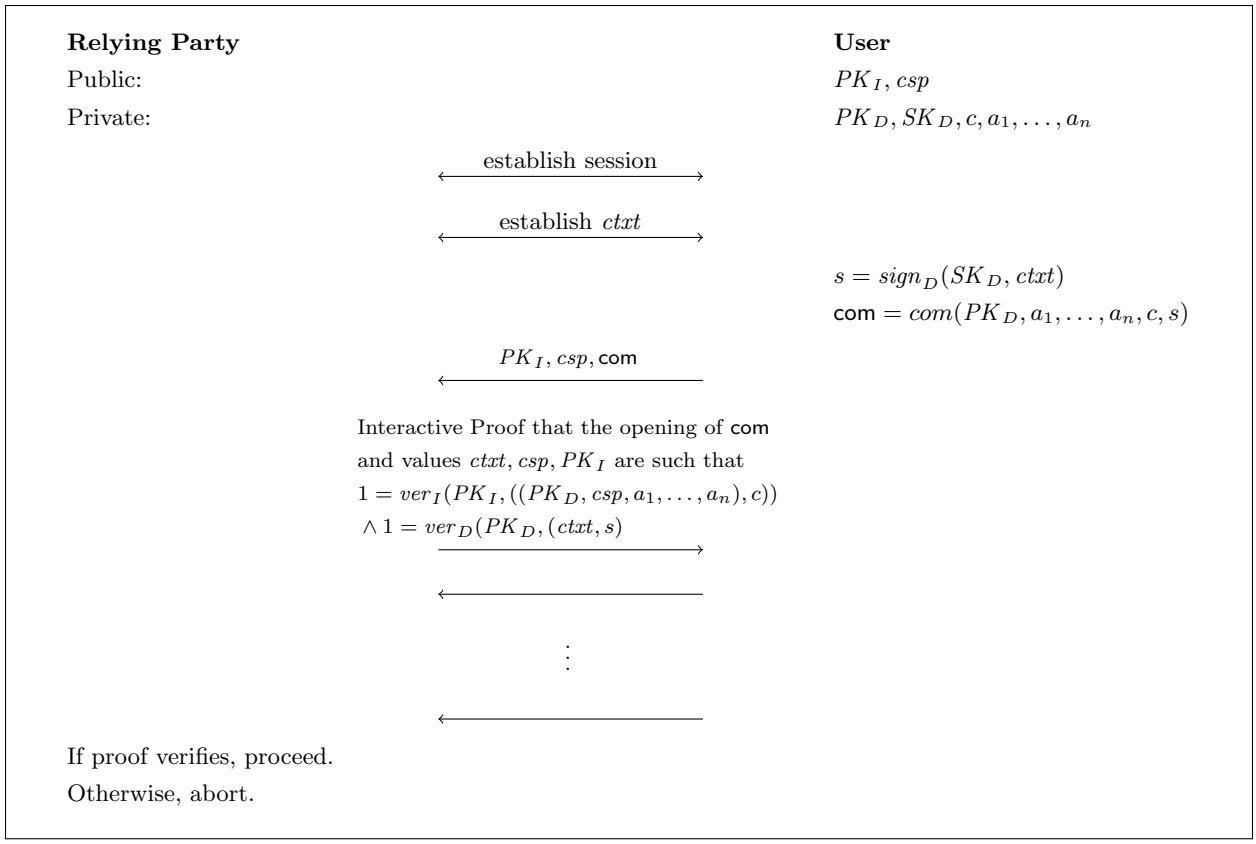

Figure 2: Credential Presentation.

$f(PK_I, csp, ctxt) = 1$, where the function f is unknown to the verifier but is the value committed to in com. At the conclusion of the interaction, if the Relying Party accepts the proof, then the Relying Party can be convinced the user holds both a valid credential and a signature on the unique *ctxt* established earlier in the session. Otherwise, if the proof fails, then the Relying Party can conclude that either the user does not hold a valid credential or the user is not in possession of the device which is associated the credential it attempted to present.

4 Proof

TODO: definition and proof of security for the generic construction - @eysa

Theorem 2. *The protocol given in Figure 1 is a secure abc assuming xyz.*

Theorem 3. *The protocol given in Figure 2 is a secure abc assuming xyz.*

5 Concrete Instantiation and Specifications

TODO: specification of how attributes map to strings that are then be encoded in the bitstring message that will signed

5.1 Public Parameters

Our constructions make use of ECDSA, which we briefly reviewed in Section 2. ECDSA requires specifying an elliptic curve group $\mathbb{G}$ of order q and a generator G for $\mathbb{G}$. It additionally requires a hash function H (e.g., SHA-256), which maps to bit strings of length q. For the zero-knowledge proofs, there are additional parameters that will be added in the next version.

5.2 Credential Format

As mentioned, the construction requires an efficient mapping from credential format to the binary string that is input to the signature scheme's sign algorithm. As there will be many different credential formats, it is not possible to name a format that will fit all of them. We assume there is a public registry (standard) that defines mappings from a credentials' attributes to binary strings (and their length). There will be further work needed here. For now we assume that the first bits of the message encode PK_D, the next part a credential specifier *csp* (that defines how many attributes will follow and how much space they take and how they are encoded), and then followed by n attributes $a_1, \ldots, a_n$. The attributes will typically include an expiration date, a serial number of the purpose of revocation, and information about user.

5.3 User's Device

A user's device contains a secure component that is able to issue signatures under a public key where only the device's secure element knows the corresponding signing key for. ECDSA has widespread support among secure elements, and we give our constructions assuming the secure element issues ECDSA signatures. We denote the ECDSA public key associated with the user's device's secure element as PK_D.

5.4 Identity Provider

Our construction considers the Identity Provider to be issuing ECDSA signatures for the credentials, but the techniques can applied to other types of signatures. We specify ECDSA signatures for efficiency reasons during the credential presentation phase since the user's device is also assumed to use ECDSA. We denote the ECDSA public key of the Identity Provider as PK_I.

5.5 Relying Party

We note that for our suggested instantiations , we do *not* require the Relying Party to have performed extra setup prior to the credential presentation. If a different zero-knowledge protocol than the one in Section 2.2.1 is used, this may require the Relying Party to perform a one-time setup prior to any credential presentations.

5.6 Zero-Knowledge Proof

TODO: specification of the ZK proof - @abhi @Muthu @Carmit (ey: this means to give the circuit)

6 Extensions

In this section, we specify how to modify the basic protocols given in Section 3 to achieve additional desired properties.

6.1 Revocation

TODO: definition of revocation scheme (probably signing a hash-tree of valid serial number ranges, tbc)
TODO: Rene: I can try for a first draft for this subsection if nobody else wants to make a start

For instance, one might require that is it possible to revoke credentials. There are many approaches to this including short-lived credentials and regularly publishing signatures on all ranges of all valid serial numbers or a hash-tree of them. At first sight, the latter seems to be the best approach for our setting.

6.2 Multi-issuer Credential Issuance

TODO: definition of multi-issuer version (include salted unique identifier in each credentials, tbd)

We all have several credentials in our wallet. While these typically all include our full names as attributes, in an online setting, we have many more credentials which hardly ever contain our name but user some email address. Thus it might be important that it is possible to issue credentials to users that are pseudonym and then use different credentials together.

6.3 Proving Predicates

Next, one might require predicates over attributes such as proving that one is older than 18 given a certified birth date. With proper encoding of attributes, proving that such predicates are true will be easy due to the use of generic zero-knowledge proofs. Similarly, it is also possible to enable selective disclosure, i.e. revealing a subset of credential attributes in the clear during the credential presentation phase, by modifying the zero-knowledge statement proved.

Acknowledgements

(ey: heads up I'm doing a generic ack for the original document authors because I'm not clear who all wants to be listed as an author) We thank the authors of [BBC+24] for their contributions to this document, in particular (ey: if anyone wants a special shoutout (but not an authorship)).

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. pages 2087–2104, 2017. doi:10.1145/3133956.3134104.

[BBC+24] Carsten Baum, Olivier Blazy, Jan Camenisch, Jaap-Henk Hoepman, Eysa Lee, Anja Lehmann, Anna Lysyanskaya, René Mayrhofer, Hart Montgomery, Ngoc Khanh Nguyen, Bart Praneel, abhi shelat, Daniel Slamanig, Stefano Tessaro, Søren Eller Thomsen, and Carmela Troncoso. Cryptographers' feedback on the eu digital identity's ARF, 2024. URL: https://github.com/user-attachments/files/15904122/cryptographers-feedback.pdf.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. pages 41–55, 2004. doi:10.1007/978-3-540-28628-8_3.

[CL01] Jan Camenisch and Anna Lysyanskaya. An identity escrow scheme with appointed verifiers. pages 388–407, 2001. doi:10.1007/3-540-44647-8_23.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. pages 56–72, 2004. doi:10.1007/978-3-540-28628-8_4.

[oST23] National Institute of Standards and Technology. Digital signature standard (dss). Technical report, U.S. Department of Commerce, Washington, D.C., 2023. doi:10.6028/NIST.FIPS.186-5.