UNIVERSITÄT
BERN

Privacy-Preserving Credentials with BBS

BBS Signatures for Enhanced Privacy in the Swiss E-ID: A Theoretical and Practical Analysis

Bachelor Thesis

Lukas Leuba

from

Bern, Switzerland

Faculty of Science, University of Bern

7. July 2025

Prof. Christian Cachin
François-Xavier Wicht
Cryptology and Data Security Group
Institute of Computer Science
University of Bern, Switzerland

Abstract

The digital transformation of identity systems holds the potential of making identification not only more convenient, but also more secure and privacy-respecting. However, the design of such systems must achieve a careful balance between security, usability, performance, interoperability, and the protection of user privacy. In Switzerland, the ongoing development of a state-issued electronic identity (E-ID) currently relies on the SD-JWT credential format with traditional ECDSA signatures. This architecture is functional and standards-compliant. However, it lacks the important privacy feature of unlinkability.

This thesis investigates the BBS (Boneh-Boyen-Shacham) signature scheme as a privacy-enhancing alternative. BBS natively allows holders to selectively disclose only the necessary parts of a credential and prevents different identifications from being linked over time and space. The thesis presents a theoretical analysis of BBS in the context of electronic identity systems and compares it to the existing SD-JWT/ECDSA setup.

To support this analysis, a practical implementation of BBS credential issuance and verification is developed and benchmarked. The results highlight the feasibility of integrating BBS in real-world identity systems and provide insights into its trade-offs in terms of performance and complexity. Ultimately, this work contributes to the broader discourse on privacy-preserving digital identity systems.

Contents

Chapter 1

Introduction

1.1 Setting

The transition from physical to electronic identity (E-ID) systems is accelerating worldwide, driven by a need for security and convenience. In Switzerland, the current development of a state-issued E-ID is a response to these demands.

Existing physical identity systems rely on a single, tangible artifact—typically a government-issued ID card—that at the same time stores information, proves authenticity through physical security features, and is presented directly to verifiers. In digital identity systems however, the structure becomes more modular and distributed. Here, these roles are split among several components: digital wallets for credential storage, protocols for secure data exchange, authentication mechanisms for user verification and verifiable credentials (VCs)—digitally signed data structures that represent claims about the holder. These credentials are issued by trusted authorities and presented to verifiers during online or in-person interactions. This process is illustrated in Figure 1.1, which shows the roles of issuer, holder, and verifier in the E-ID setting:

- The **issuer** (e.g. the government) creates and signs a set of verifiable credentials;
- The **holder** (e.g. a citizen like Alice) stores these credentials and selectively presents them when needed;
- The **verifier** (e.g. a service provider like Bob) receives the credentials and checks their validity.

At the heart of this process lies the signature scheme, which guarantees the authenticity and integrity of the credentials. The design and selection of this cryptographic component affects both the system's security and privacy. This thesis focuses on evaluating the use of BBS (Boneh-Boyen-Shacham) signatures, a cryptographic scheme that enables native selective disclosure and unlinkability, as a privacy-enhancing alternative to traditional schemes like ECDSA in the context of the Swiss E-ID.

1.2 The Basic Identification Problem

To motivate BBS signatures, let us first look at the basic digital identification problem: Alice holds credentials (such as her name, date of birth and biometric data) and wants to prove their authenticity to Bob—a verifier. The naive solution uses traditional signature schemes like ECDSA: a trusted issuer (e.g., the government) cryptographically signs Alice's credentials. This allows Alice to present the issuer's signature alongside her credentials to Bob, who can verify said signature through the known public key of the issuer.

This solution is valid for proving authenticity. However, it is not viable from a privacy standpoint. There are two major flaws in the above approach:

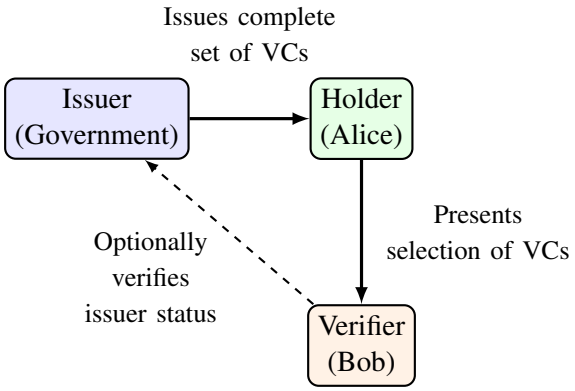


Figure 1.1. Role distribution in the E-ID setting: The **issuer** (e.g., a government agency) issues a complete set of verifiable credentials (VCs) to the **holder** (e.g., a citizen like Alice), who stores them in a digital wallet. The holder can then present a selected subset of these credentials to a **verifier** (e.g., a service provider like Bob). The verifier may optionally validate the issuer's authenticity.

- **Lack of selective disclosure:** Since all credentials are signed together, Alice must reveal all her credentials to the verifier, even when only specific information (such as proof of age) is required.
- **Linkability across presentations:** The unchanging signature allows different presentations to be correlated over time and space, enabling tracking by a single verifier across multiple interactions or collusion between different verifiers.

The current Swiss E-ID program mitigates these issues somewhat by using the SD-JWT (Selective Disclosure JSON Web Token) credential format combined with ECDSA signatures. SD-JWT is an extension of the standard JSON Web Token (JWT) format. In a typical JWT, all credentials are included in the token and are visible to any party that receives it. SD-JWT modifies this by allowing certain credentials to be "hidden" at the time of issuance. Specifically, the issuer replaces the values of these claims with their hashed versions and provides the original values separately to the holder. This approach was likely chosen for its practicality, interoperability and support for selective disclosure. For example, ECDSA is natively supported by the secure hardware components of Android and iOS devices (Android Keystore system and Apple's Secure Enclave). This support ensures secure key storage and operations, which makes ECDSA the practical choice for mobile authentication scenarios.

However, this approach still does not offer the important privacy-property of unlinkability. Each SD-JWT token is signed once by the issuer and reused during every presentation. Even if only some claims are disclosed each time, the underlying token (with its structure and undisclosed hash values) remains constant. This enables different verifiers to link repeated credential presentations and build behavioural profiles over time.

For example, consider a citizen using their Swiss E-ID to verify their identity when registering on Galaxus.ch (`https://www.galaxus.ch`, a large online retailer owned by the Migros cooperative federation), and then later again when signing up for a promotional service at a Migros supermarket. Even if these services are branded separately and run by different legal entities, they may share access to backend infrastructure and analytics. If the same SD-JWT token is presented in both cases, the shared hash structure allows Galaxus and Migros to link these interactions and infer that they involved the same person. Over time, such linkable interactions allow the creation of detailed user profiles—including purchase behavior, location history, and service usage—without the user's knowledge or consent.

This demonstrates that an approach of solving the basic identification problem without unlinkability fails to preserve true user privacy. Effective digital identity systems must not only let users choose what to reveal, but also ensure that their disclosures cannot be correlated across time and space. SD-JWT partially solves the former but not the latter.

In contrast, BBS signatures offer native support for selective disclosure and unlinkability, which

promises stronger privacy guarantees without fundamentally altering the credential format. This thesis investigates BBS signatures as an alternative to the approach taken by the Swiss E-ID program.

We provide both a theoretical foundation and a practical evaluation of BBS signatures. Chapter 2 introduces the necessary cryptographic background, followed by a detailed explanation of the BBS signature scheme construction in Chapter 3. Chapter 4 discusses how BBS compares against the current SD-JWT/ECDSA model within the E-ID ecosystem. The practical portion in Chapter 5 delivers an implementation of the BBS signature scheme, accompanied by benchmarks and an empirical discussion. We conclude with a summary of findings and future considerations in Chapter 6.

Chapter 2

Background

Physical identification documents such as passports are designed to be hard to forge through the use of physical security features like holograms, watermarks, microprinting and tamper-evident laminates. These features are easily verifiable by trained personnel or even machines, which makes forgery detection relatively easy in many cases.

In contrast, digital identification systems rely entirely on cryptographic techniques for authenticity and integrity. Here, verification occurs via digital signatures and secure protocols. This shift not only changes the threat model—from physical forgery to cryptographic attacks—but also enables new privacy-preserving capabilities such as selective disclosure and unlinkability that are not possible in physical documents. In this context of digital identity, the following attributes are particularly important when constructing and evaluating a signature scheme:

- **Standard security properties:**
  - **Authenticity:** Digital signatures should prove that the credentials originate from a trusted source.
  - **Integrity:** Digital signatures should prove that there was no modification of the credentials after issuance.
  - **Non-repudiation:** The issuer cannot deny having issued digitally signed credentials.
- **Properties specific to identity applications:**
  - **Decentralized Trust:** The system can operate without constant online verification against a centralized authority.
  - **Privacy:**
    - **Selective Disclosure:** While presenting digitally signed credentials, only relevant information is disclosed.
    - **Unlinkability:** Different presentations cannot be traced back to the same person.

These properties require secure cryptographic foundations, which we introduce in this chapter. We begin with general digital signature schemes as a foundation, then build toward the specialized constructions that enable BBS signatures. The cryptographic definitions and concepts presented in this chapter follow the formulations of Boneh and Shoup [1] unless otherwise stated.

2.1 Digital Signatures

To establish a framework, we first define the general notion of cryptographic signature schemes.

Definition 2.1.1 (Signature Scheme)**.** For messages in the finite message space $\mathcal{M}$ and signatures in the finite signature space Σ, a *signature scheme* $\mathcal{S}$ is a triple of algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ defined over $(\mathcal{M}, \Sigma)$ as follows:

- $\mathsf{KeyGen}(\lambda) \rightarrow (sk, vk)$ is a probabilistic algorithm taking a security parameter λ as input and generating a keypair (sk, vk) as output. Here sk is called the *signing key*, and vk is called the *verification key*.
- $\mathsf{Sign}(m, sk) \rightarrow \sigma$ is a probabilistic algorithm taking a message $m \in \mathcal{M}$ and signing key sk as input, and generating a signature $\sigma \in \Sigma$ as output.
- $\mathsf{Verify}(vk, m, \sigma) \rightarrow \{0, 1\}$ is a deterministic algorithm taking the verification key vk, message $m \in \mathcal{M}$ and signature $\sigma \in \Sigma$ as input, and returning either accept (1) or reject (0).

2.1.1 Correctness of Digital Signatures

We require that a signature scheme is correct.

Definition 2.1.2 (Correctness)**.** A signature scheme $\mathcal{S}$ is *correct* if all signatures created by Sign are accepted by Verify:

$$\Pr[\mathsf{Verify}(vk, m, \mathsf{Sign}(sk, m)) = 1] = 1 \tag{2.1}$$

2.1.2 Security of Digital Signatures

Beyond correctness, a signature scheme must be secure against forgery attempts. The security notion we will use is existential unforgeability under a chosen message attack. This notion captures the idea that an adversary, even after seeing many valid signatures on messages of their choosing, cannot produce a valid signature on a new message. In this model, the adversary has access to a signing oracle that produces valid signatures for requested messages, simulating an attack scenario where the adversary might trick the legitimate signer into signing specific documents.

Definition 2.1.3 (Signature Security)**.** We define security through a game between a challenger and an adversary $\mathcal{A}$, parameterized by a security parameter λ:

1. The challenger generates a key pair $(sk, vk) \leftarrow \mathsf{KeyGen}(\lambda)$ and gives vk to $\mathcal{A}$.
2. $\mathcal{A}$ can adaptively request signatures on messages of its choice by querying a signing oracle. For each query m_i, the oracle returns $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$. By assumption, $\mathcal{A}$ is limited to a polynomial (in λ) number of such queries.
3. Eventually, $\mathcal{A}$ outputs a pair (m^*, σ^*).

$\mathcal{A}$ wins if $\mathsf{Verify}(vk, m^*, \sigma^*) = 1$ and m^* was not queried in step 2.

A signature scheme $\mathcal{S}$ is *existentially unforgeable under chosen-message attacks* if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ wins the above game is negligible in the security parameter λ..

2.2 Bilinear Pairings

BBS belongs to the family of pairing-based signature schemes. Such schemes take advantage of the properties of bilinear maps. These functions enable novel cryptographic constructions that address the privacy requirements outlined earlier.

Definition 2.2.1 (Bilinear Pairing)**.** $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T$ are cyclic groups of prime order p. $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ are generators. A (bilinear) *pairing* is a map: $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ satisfying the following properties:

1. Bilinear: for all $u, u' \in \mathbb{G}_0$ and $v, v' \in \mathbb{G}_1$ we have

$$e(u \cdot u', v) = e(u, v) \cdot e(u', v) \quad \text{and} \quad e(u, v \cdot v') = e(u, v) \cdot e(u, v'),$$

2. Non-degenerate: $g_T := e(g_0, g_1)$ is a generator of $\mathbb{G}_T$.

3. Computable: e can be efficiently computed.

As a direct consequence of the bilinearity property, for any exponents $\alpha, \beta \in \mathbb{Z}_p$, we can derive the following relationship:

$$e(g_0^\alpha, g_1^\beta) = e(g_0, g_1)^{\alpha\beta} = e(g_0^\beta, g_1^\alpha).$$

This behaviour is very useful for cryptographic applications since it lets us test relevant relationships without knowing all the individual parts. For example, we can verify $g_0^a = h$ without knowing a by checking if $e(h, g_1) = e(g_0, g_1^a)$ when g_1^a is publicly available. Later, we will leverage bilinear pairings by using this and similar properties in our construction of BBS signatures.

2.3 Elliptic Curves

Elliptic curves provide the mathematical foundation for the groups used in pairing-based cryptography. In this section, we present the formal definition of elliptic curves and their properties relevant to BBS signatures.

2.3.1 Definition and Group Structure

Definition 2.3.1 (Elliptic Curve)**.** An *elliptic curve* E over a field $\mathbb{F}$ is defined by the Weierstrass equation:

$$E : y^2 = x^3 + ax + b \tag{2.2}$$

where $a, b \in \mathbb{F}$ and the discriminant $\Delta = 4a^3 + 27b^2 \neq 0$. The set of points $E/\mathbb{F}$ conssists of all solutions $(x, y) \in \mathbb{F} \times \mathbb{F}$ to this equation, together with a special point $\mathcal{O}$ called the point at infinity.

The non-zero discriminant condition ensures that the curve has no singular points (cusps or self-intersections), which is necessary for the following group structure to be well-defined:

Definition 2.3.2 (Group Law)**.** For an elliptic curve E over a field $\mathbb{F}$, the points in $E/\mathbb{F}$ *form an abelian group* where:

- The identity element is the point at infinity $\mathcal{O}$.
- For any point $P = (x, y) \in E/\mathbb{F}$, the inverse of P is the point $-P = (x, -y)$.
- For distinct points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, the sum $P_3 = P_1 + P_2 = (x_3, y_3)$ is computed as follows:

$$\lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & \text{if } P_1 \neq P_2 \\ \frac{3x_1^2 + a}{2y_1} & \text{if } P_1 = P_2 \text{ and } P_1 \neq -P_2 \end{cases} \tag{2.3}$$

$$x_3 = \lambda^2 - x_1 - x_2 \tag{2.4}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{2.5}$$

- If $P_2 = -P_1$ (i.e. $x_1 = x_2$ and $y_1 = -y_2$), then $P_1 + P_2 = \mathcal{O}$.

2.3.2 Elliptic Curves over Finite Fields

Elliptic curves can be defined over various fields. For cryptographic applications, we typically use curves over finite fields.

Definition 2.3.3 (Elliptic Curve over Finite Field)**.** An *elliptic curve E over a finite field $\mathbb{F}_q$* (where $q > 3$ is a prime) consists of the solutions to the Weierstrass equation in $\mathbb{F}_q$, forming the group $E/\mathbb{F}_q$.

We write $E(\mathbb{F}_q)$ to denote the points on the curve E that are defined over $\mathbb{F}_q$. The order of the group $E(\mathbb{F}_q)$, denoted by $|E(\mathbb{F}_q)|$, is the number of points on the curve. Hasse's Theorem states that this value is bounded as $|E(\mathbb{F}_q)| = q + 1 - t$, for $|t| \leq 2\sqrt{q}$. There also exists an efficient algorithm to count $|E(\mathbb{F}_q)|$ precisely.

For cryptographic purposes, we typically work with a subgroup of $E(\mathbb{F}_q)$ of prime order p. This subgroup is generated by a point $P \in E(\mathbb{F}_q)$ of order p, resulting in the cyclic group $\langle P \rangle = \{P, 2P, 3P, \ldots, pP = \mathcal{O}\}$. The security of elliptic curve cryptography relies on the computational difficulty of finding the discrete logarithm in this group, which we will discuss in more detail in Section 2.4.

2.3.3 Pairing-Friendly Elliptic Curves

Pairing based signature schemes such as BBS require efficient bilinear pairings. Such pairings are only possible when using a special class of elliptic curves known as pairing-friendly curves.

Definition 2.3.4 (Pairing-Friendly Curve)**.** An elliptic curve E over a finite field $\mathbb{F}_q$ is *pairing-friendly* if:

Property 1 There exists a subgroup of $E(\mathbb{F}_q)$ with a large prime order p.

Property 2 The embedding degree k (the smallest integer such that r divides $q^k - 1$) is sufficiently small to allow efficient computation of pairings, but large enough to provide security.

These properties allow us to construct a bilinear pairing $\mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ as follows:

- $\mathbb{G}_0$: an order p subgroup of $E(\mathbb{F}_q)$ (Property 1)
- $\mathbb{G}_1$: an order p subgroup of $E(\mathbb{F}_{q^k})$ where $\mathbb{G}_1 \cap \mathbb{G}_0 = \{\mathcal{O}\}$ (Property 2)
- $\mathbb{G}_T$: an order p multiplicative subgroup of $\mathbb{F}_{q^k}$

The size of the embedding degree is important because it determines the field size where the pairing values "live". If k is too large, pairing computation becomes expensive. The bigger group $\mathbb{G}_1$ also implies an asymmetry in representation size. Elements of $\mathbb{G}_1$ need significantly more storage than elements of $\mathbb{G}_0$. This lets us for example optimize for public key length by putting it into $\mathbb{G}_0$.

For BBS signatures, BLS12-381 (a BLS curve with $k = 12$ and 381-bit field size) is a popular choice and the one we will use in our implementation. However, for most of the following work, in particular the construction of BBS, it suffices to treat elliptic curves as another finite abelian group with desirable cryptographic properties.

2.4 Security Assumptions

2.4.1 Discrete Logarithm Problem (DLP)

The discrete logarithm problem builds the basis for the assumptions which are used in the construction of BBS.

Definition 2.4.1. (Discrete Logarithm (DL) Problem) Let $\mathbb{G}$ be a cyclic group of prime order p generated by $g \in \mathbb{G}$. Given a group element g^k, where $k \in \mathbb{Z}_p$, the *discrete logarithm (DL) problem* is to determine the value of k.

The *discrete logarithm assumption* states that solving the DL problem is computationally infeasible for certain groups when the order of the group is sufficiently large. More precisely, for any probabilistic polynomial-time algorithm $\mathcal{A}$ and security parameter λ:

$$\Pr[\mathcal{A}(q, g, g^k) = k] \leq \text{negl}(\lambda) \tag{2.6}$$

where the probability is taken over the random choice of $k \in \mathbb{Z}_p$ and the random coins of $\mathcal{A}$, and $\text{negl}(\lambda)$ is a negligible function in the security parameter λ.

2.4.2 Elliptic Curve Discrete Logarithm Problem

The hardness of the DL problem strongly depends on the choice of the group. For BBS signatures, we use bilinear groups derived from specific elliptic curves as we have seen in the last chapter.

Definition 2.4.2 (Elliptic Curve Discrete Logarithm Problem (ECDLP))**.** Given points P and $Q = kP$ on an elliptic curve $E(\mathbb{F}_q)$, where k is an integer, the *elliptic curve discrete logarithm problem* is to determine the value of k.

The *ECDLP assumption* states that for any probabilistic polynomial-time algorithm $\mathcal{A}$ and security parameter λ:

$$\Pr[\mathcal{A}(E, P, kP) = k] \leq \text{negl}(\lambda) \tag{2.7}$$

where the probability is taken over the random choice of k and the random coins of $\mathcal{A}$.

The fastest known algorithms for ECDLP such as Pollard's Rho or baby-step giant-step remain exponential in the security parameter, whereas finite-field DLP can be solved with sub-exponential methods (e.g. the Number Field Sieve). This allows elliptic curve cryptosystems to achieve the same security level as other systems with significantly smaller security parameters.

The hardness of the ECDLP depends on several factors, including the curve structure, field size, and implementation details. For the elliptic curves used in BBS, the parameters are chosen to ensure that solving the ECDLP is computationally infeasible with current technology.

2.4.3 q-Strong Diffie-Hellman Problem

The q-Strong Diffie-Hellman (q-SDH) problem gives the solver more information than the standard (EC)DLP while still assuming hardness. We say that q-SDH implies (EC)DLP, since if one could efficiently solve (EC)DLP, one could break the q-SDH assumption as well.

Definition 2.4.3 (q-Strong Diffie-Hellman Problem [2])**.** Let $\mathbb{G}_1$, $\mathbb{G}_2$ be cyclic groups of prime order p. Let g_1 be a generator of $\mathbb{G}_1$ and g_2 a generator of $\mathbb{G}_2$. The *q-Strong Diffie-Hellman (q-SDH) problem* in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows: given a $(q+2)$-tuple $(g_1, g_2, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^q)})$ as input, output a pair $(g_1^{1/(\gamma+x)}, x)$ where $x \in \mathbb{Z}_p^*$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ if:

$$\Pr[\mathcal{A}(g_1, g_2, g_2^{\gamma}, \ldots, g_2^{(\gamma^q)}) = (g_1^{\frac{1}{\gamma+x}}, x)] \geq \epsilon \tag{2.8}$$

where the probability is over the random choice of generator g_2 in $\mathbb{G}_2$ (with $g_1 = \psi(g_2)$), of γ in $\mathbb{Z}_p^*$ and the random bits of $\mathcal{A}$.

We say that the (q, t, ϵ)-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if no t-time algorithm has advantage at least ϵ in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$.

2.4.4 Decision Linear Problem

In bilinear groups, the standard Decisional Diffie-Hellman (DDH) problem becomes easy to solve using the pairing operation. However, we still require alternative hardness assumptions to securely build cryptographic constructions, including BBS signatures. The Decision Linear problem provides an alternative hardness assumption that remains secure even when DDH is easy to solve.

Definition 2.4.4 (Decision Linear Problem [2])**.** With $g_1 \in \mathbb{G}_1$ as above, along with arbitrary generators u, v, h of $\mathbb{G}_1$, the *Decision Linear problem* in $\mathbb{G}_1$ is defined as follows: given $u, v, h, u^a, v^b, h^c \in \mathbb{G}_1$ as input, output "yes" if $a + b = c$ and "no" otherwise.

The advantage of an algorithm $\mathcal{A}$ in deciding the Decision Linear problem in $\mathbb{G}_1$ is defined as:

$$\text{Adv}_{\mathcal{A}}^{\text{Linear}} = \Big|\Pr[\mathcal{A}(u, v, h, u^a, v^b, h^{a+b}) = \text{yes} : u, v, h \xleftarrow{R} \mathbb{G}_1, a, b \xleftarrow{R} \mathbb{Z}_p] \tag{2.9}$$

$$- \Pr[\mathcal{A}(u, v, h, u^a, v^b, \eta) = \text{yes} : u, v, h, \eta \xleftarrow{R} \mathbb{G}_1, a, b \xleftarrow{R} \mathbb{Z}_p]\Big| \tag{2.10}$$

We say that the (t, ϵ)-Decision Linear Assumption (LA) holds in $\mathbb{G}_1$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$,

$$\text{Adv}_{\mathcal{A}}^{\text{Linear}} \leq \text{negl}(\lambda),$$

where λ is the security parameter.

The Decision Linear problem enables the construction of the Linear Encryption (LE) scheme:

Definition 2.4.5 (Linear Encryption Scheme (LE))**.** Let $\mathbb{G}_1$ be a group of prime order p with generators $u, v, h \in \mathbb{G}_1$ such that the relative discrete logarithms between these generators are unknown.

Key Generation: A user's public key is the triple of generators $(u, v, h) \in \mathbb{G}_1^3$. The private key consists of exponents $x, y \in \mathbb{Z}_p$ such that $u^x = v^y = h$.

Encryption: To encrypt a message $M \in \mathbb{G}_1$, choose random values $a, b \in \mathbb{Z}_p$ and output the ciphertext triple:

$$(T_1, T_2, T_3) \coloneqq (u^a, v^b, M \cdot h^{a+b}) \tag{2.11}$$

Decryption: To recover the message from a ciphertext (T_1, T_2, T_3), the user computes:

$$M = T_3/(T_1^x \cdot T_2^y) \tag{2.12}$$

Correctness of Linear Encryption

The Linear Encryption scheme satisfies the correctness property: any properly formed ciphertext can be correctly decrypted using the corresponding private key. For a ciphertext $(T_1, T_2, T_3) = (u^a, v^b, M \cdot h^{a+b})$ encrypted under public key (u, v, h) with corresponding private key (x, y) where $u^x = v^y = h$:

$$T_3/(T_1^x \cdot T_2^y) = M \cdot h^{a+b}/(u^{ax} \cdot v^{by}) \tag{2.13}$$

$$= M \cdot h^{a+b}/(h^a \cdot h^b) \tag{2.14}$$

$$= M \cdot h^{a+b}/h^{a+b} \tag{2.15}$$

$$= M \tag{2.16}$$

Security of Linear Encryption

The security of the Linear Encryption scheme relies on the Decision Linear assumption. Intuitively, if an adversary cannot distinguish whether $c = a + b$ given (u, v, h, u^a, v^b, h^c), then it cannot determine whether a ciphertext $(u^a, v^b, M \cdot h^{a+b})$ encrypts a specific message M or some other message.

More formally, the Linear Encryption scheme is semantically secure against chosen-plaintext attacks under the Decision Linear assumption. This means that an adversary, given the public key and a challenge ciphertext that encrypts one of two messages of its choice, cannot determine which message was encrypted with probability significantly better than random guessing.

The formal security-proof of the Linear Encryption Scheme is a natural extension of the proof of security of ElGamal.

2.5 Zero-Knowledge Proofs

How can a user prove possession of attributes or credentials without unnecessarily revealing the actual values? For example, a user might want to prove they are above a certain age without revealing their exact birthdate (selective disclosure), or prove they possess a valid credential without exposing the credential itself (unlinkablility). Zero-Knowledge Proofs (ZKPs) can provide the solution to these problems. As the name suggests, ZKPs allow a prover (Alice) to convince a verifier (Bob) that a statement is true (for example that Alice knows a valid q-SDH-tuple) without disclosing any information about the statement (the actual q-SDH-tuple) to Bob.

The following chapter is based on Smart's textbook [3].

2.5.1 Formal Definitions

A zero-knowledge argument is characterized by three fundamental properties:

- **Completeness**: If the statement is true and both the prover and verifier follow the protocol, the verifier will accept the proof with probability 1.
- **Soundness**: If the statement is false, no cheating prover can convince the verifier that it is true, except with some negligible probability.
- **Zero-Knowledge**: If the statement is true, the verifier learns nothing other than the fact that the statement is true.

More formally, let $\mathcal{L}$ be a language in NP with associated relation $\mathcal{R}$. For a statement $x \in \mathcal{L}$, there exists a witness w such that $(x, w) \in \mathcal{R}$. A zero-knowledge argument for $\mathcal{L}$ is an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ that satisfies:

Definition 2.5.1 (Completeness)**.** For all $(x, w) \in \mathcal{R}$, if both $\mathcal{P}$ and $\mathcal{V}$ follow the protocol on input x and $\mathcal{P}$ has witness w, then $\mathcal{V}$ accepts with probability 1.

Definition 2.5.2 (Soundness)**.** For all $x \notin \mathcal{L}$ and any (potentially malicious) prover $\mathcal{P}^*$, the probability that $\mathcal{V}$ accepts is negligible in the security parameter.

Definition 2.5.3 (Zero-Knowledge)**.** For all $(x, w) \in \mathcal{R}$ and any (potentially malicious) verifier $\mathcal{V}^*$, there exists a probabilistic polynomial-time simulator $\mathcal{S}$ that, given only x (and not w), can produce a transcript that is computationally indistinguishable from a real interaction between $\mathcal{P}(x, w)$ and $\mathcal{V}^*(x)$.

The zero-knowledge property is typically demonstrated through simulation: if a transcript computationally indistinguishable from a real interaction can be generated without access to the witness, then the verifier cannot have learned anything about the witness from the actual interaction.

Zero-knowledge proofs of knowledge (ZKPoK) extend this concept by requiring that the prover not only convinces the verifier that a statement is true but also demonstrates that they know a witness. This is formalized through the existence of a knowledge extractor—an algorithm that can extract the witness from a successful prover.

Definition 2.5.4 (Proof of Knowledge)**.** For all $(x, w) \in \mathcal{R}$ and any prover $\mathcal{P}^*$ that convinces $\mathcal{V}$ to accept with non-negligible probability, there exists a polynomial-time knowledge extractor $\mathcal{E}$ that, given access to $\mathcal{P}^*$, can extract a witness w' such that $(x, w') \in \mathcal{R}$ with overwhelming probability.

2.5.2 Sigma Protocols

Sigma protocols are a class of three-move zero-knowledge proofs that follow a specific structure: first a commitment from the prover, then a challenge from the verifier and finally a response again form the prover. A Sigma protocol for proving knowledge of a witness w for a statement x consists of the following three moves:

1. **Commitment**: The prover $\mathcal{P}$ computes a commitment a based on the witness w and random values, then sends a to the verifier $\mathcal{V}$.

2. **Challenge**: The verifier $\mathcal{V}$ selects a random challenge e and sends it to the prover.

3. **Response**: The prover $\mathcal{P}$ computes a response z based on the witness w, the randomness used in the commitment, and the challenge e, then sends z to the verifier.

The verifier decides whether to accept or reject based on x, a, e, and z.
Sigma protocols satisfy three key properties:

Definition 2.5.5 (Completeness)**.** If $(x, w) \in \mathcal{R}$, and the prover and verifier follow the protocol honestly, then the verifier always accepts.

Definition 2.5.6 (Special Soundness)**.** From any statement x and any two accepting transcripts (a, e, z) and (a, e', z') with the same commitment but different challenges $(e \neq e')$, there exists an efficient algorithm that can extract a witness w such that $(x, w) \in \mathcal{R}$.

Definition 2.5.7 (Special Honest-Verifier Zero-Knowledge)**.** There exists a polynomial-time simulator $\mathcal{S}$ that, given x and a challenge e, can produce a transcript (a, e, z) that is indistinguishable from a transcript of a real execution between an honest prover with a valid witness and an honest verifier who outputs challenge e.

2.5.3 Schnorr's Identification Protocol

Schnorr's identification protocol is a fundamental Sigma protocol. It allows a prover to demonstrate knowledge of a discrete logarithm without revealing it. Specifically, given a public group element $\beta = g^\alpha$ in a group $\mathbb{G}$ of prime order p with generator g, Schnorr's protocol allows the prover to convince a verifier that they know the exponent α without revealing any information about α. Notice that α corresponds to the witness w and $\beta = g^\alpha$ corresponds to the statement x.

The protocol operates as follows:

1. **Setup**: Both parties agree on a group $\mathbb{G}$ of prime order p with generator g. The prover's public key is $\beta = g^\alpha$, and their private key is $\alpha \in \mathbb{Z}_p$.

2. **Commitment**: The prover selects a random value $k \xleftarrow{R} \mathbb{Z}_p$ and computes a commitment $r = g^k$. The prover sends r to the verifier.

3. **Challenge**: The verifier chooses a random challenge $e \xleftarrow{R} \mathbb{Z}_p$ and sends it to the prover.

4. **Response**: The prover computes $s = k + \alpha \cdot e \mod p$ and sends s to the verifier.

5. **Verification**: The verifier checks whether $g^s = r \cdot \beta^e$. If the equation holds, the verifier accepts; otherwise, it rejects.

Security Analysis of Schnorr's Protocol

Let us analyze Schnorr's protocol according to the three properties of Sigma protocols:

- **Completeness**: If the prover knows α and follows the protocol, then the verification equation always holds:

$$g^s = g^{k+\alpha \cdot e} \tag{2.17}$$
$$= g^k \cdot g^{\alpha \cdot e} \tag{2.18}$$
$$= r \cdot \beta^e \tag{2.19}$$

- **Special Soundness**: Given two accepting transcripts (r, e, s) and (r, e', s') with the same commitment but different challenges, we can extract the witness α as follows:

$$g^s = r \cdot \beta^e \tag{2.20}$$
$$g^{s'} = r \cdot \beta^{e'} \tag{2.21}$$
$$\tag{2.22}$$

  From these equations, we get $g^s = g^{s'} \cdot \beta^{e-e'}$, which implies $g^{s-s'} = \beta^{e-e'}$. Since $\beta = g^\alpha$, we have $g^{s-s'} = g^{\alpha(e-e')}$, thus $s - s' = \alpha(e - e') \mod p$. Since $e \neq e'$, we can compute $\alpha = \frac{s-s'}{e-e'} \mod p$.

- **Special Honest-Verifier Zero-Knowledge**: A simulator for Schnorr's protocol operates as follows:

  1. Given a challenge e, select $s \xleftarrow{R} \mathbb{Z}_p$ randomly.
  2. Compute $r = g^s \cdot \beta^{-e}$.
  3. Output the transcript (r, e, s).

  This simulated transcript is identically distributed to a real transcript from an honest execution of the protocol, thus providing perfect special honest-verifier zero-knowledge.

In our construction, we will use Schnorr's protocol as building blocks to create the foundational zero-knowledge-protocol of the signature.

2.5.4 From Interactive to Non-Interactive Proofs

For practical applications like digital signatures, non-interactive proofs are required. The Fiat-Shamir heuristic transforms interactive Sigma protocols into non-interactive proofs by replacing the verifier's random challenge with a hash of the statement and commitment:

1. The prover computes the commitment a as in the interactive protocol.
2. Instead of receiving a random challenge from the verifier, the prover computes $e = H(x \| a)$, where H is a cryptographic hash function, treated as a random oracle, x is the statement being proven, and $\|$ denotes concatenation.
3. The prover computes the response z as in the interactive protocol, using the computed challenge e.
4. The proof consists of (a, z). Note that the challenge e is not included since it can be recomputed by the verifier.
5. The verifier accepts if and only if the verification equation of the original Sigma protocol holds with respect to the statement x, the commitment a, the recomputed challenge $e = H(x \| a)$, and the response z.

When applying the Fiat-Shamir transform to create a digital signature scheme, the message m to be signed is typically included in the hash computation: $e = H(x\|a\|m)$.

The security of the Fiat-Shamir heuristic is analyzed in the random oracle model, where the hash function is modeled as truly random. In this model, the non-interactive proof inherits the zero-knowledge and soundness properties of the original interactive protocol.

2.5.5 Zero-Knowledge Proofs for Pedersen Commitments

Pedersen commitments allow a prover to commit to a value without revealing it, while still being able to open the commitment later to prove its contents. Zero-knowledge proofs for Pedersen commitments enable proving statements about committed values without revealing the values themselves.

Pedersen Commitments

A Pedersen commitment to a value $\alpha \in \mathbb{Z}_p$ is computed as:

$$C = g^{\alpha} \cdot h^{\beta} \tag{2.23}$$

where g and h are generators of a group $\mathbb{G}$ of prime order p, and $\beta \in \mathbb{Z}_p$ is a random value chosen by the committer. The Pedersen commitment scheme has two important properties:

- **Unconditionally Hiding**: The commitment C reveals no information about the committed value α (if β is chosen uniformly at random).
- **Computationally Binding**: The committer cannot open the commitment to a different value $\alpha' \neq \alpha$ (under the discrete logarithm assumption).

Proving Knowledge of a Pedersen Commitment

Given a Pedersen commitment $C = g^{\alpha} \cdot h^{\beta}$, a prover may want to convince a verifier that they know the values α and β without revealing them. This can be accomplished using a Sigma protocol similar to Schnorr's identification protocol:

1. **Commitment**: The prover selects random values $k_{\alpha}, k_{\beta} \xleftarrow{R} \mathbb{Z}_p$ and computes $t = g^{k_{\alpha}} \cdot h^{k_{\beta}}$. The prover sends t to the verifier.
2. **Challenge**: The verifier chooses a random challenge $e \xleftarrow{R} \mathbb{Z}_p$ and sends it to the prover.
3. **Response**: The prover computes $s_{\alpha} = k_{\alpha} + e \cdot \alpha \mod p$ and $s_{\beta} = k_{\beta} + e \cdot \beta \mod p$, and sends (s_{α}, s_{β}) to the verifier.
4. **Verification**: The verifier checks whether $g^{s_{\alpha}} \cdot h^{s_{\beta}} = t \cdot C^{e}$. If the equation holds, the verifier accepts; otherwise, it rejects.

Multi-Base Commitments in BBS

BBS+ signatures [4] and later derivatives such as the Tessaro-Zhu BBS scheme [5] extend the concept of Pedersen commitments to multiple bases. In these schemes, a commitment to a set of messages $(m_1, m_2, \ldots, m_L)$ takes the form:

$$C = g_0 \cdot g_1^{s} \cdot g_2^{m_1} \cdot g_3^{m_2} \cdots g_{L+1}^{m_L} \tag{2.24}$$

where $g_0, g_1, \ldots, g_{L+1}$ are generators of the group, s is a random value, and $m_1, \ldots, m_L$ are the messages to be signed.

Proving knowledge of such a commitment requires an extension of the basic Pedersen commitment proof. This extended protocol will provide the selective disclosure mechanism of BBS, as it allows a prover to demonstrate knowledge of some messages while keeping others hidden.

2.5.6 Applications to BBS Signatures

The zero-knowledge proof techniques discussed form the cryptographic foundation for BBS signatures by enabling their key features:

- **Multi-Message Signing**: BBS allows signing multiple messages simultaneously, with all messages bound into a single signature.
- **Selective Disclosure**: Users can create proofs that reveal only selected messages while keeping others hidden, all without requiring the original issuer's involvement.
- **Unlinkability**: Different presentations of the same credential cannot be linked together, enhancing privacy.

As we will see in the next chapter, BBS signatures use a zero-knowledge proof of knowledge of a q-Strong Diffie-Hellman tuple, combined with proofs over Pedersen-style commitments. The Fiat-Shamir heuristic transforms these into non-interacting proofs which finally can be used in the signature scheme.

Chapter 3

BBS Signature Schemes

As was mentioned in the introduction, traditional signature schemes like ECDSA are effective for verifying the authenticity of digital credentials. However, they fall short in protecting user privacy. They allow different credential presentations to be linked and—if implemented naively—force full disclosure of all signed attributes. This clearly undermines the principles of data minimization and user autonomy.

This chapter presents an alternative approach in the form of BBS signatures, which were designed to address these privacy limitations.

The name BBS derives from the authors of the foundational academic work by Boneh, Boyen, and Shacham, who first described the scheme in 2004 as part of a group signatures protocol [2]. Soon after, Camenisch and Lysyanskaya reformulated the scheme as a standalone signature scheme for anonymous credentials applications [6]. The first provably secure version of BBS signatures was presented by Au, Susilo, and Mu in 2006 (BBS+) [4].

Later research focused on improving the scheme's efficiency and security properties. Most recently, in 2023, Tessaro and Zhu [5] presented significant performance improvements, further shrinking BBS signature sizes while maintaining security guarantees.

This chapter progressively develops BBS signature schemes, beginning with the zero-knowledge protocols that form their cryptographic foundation. We start by examining the original BBS group signature scheme, then analyze its security properties and extend it to the Tessaro-Zhu framework. The Tessaro-Zhu BBS signatures allow for signing multiple messages simultaneously with a single, constant-size signature while supporting selective disclosure of signed attributes through zero-knowledge proofs. The resulting proofs are privacy-preserving: they reveal nothing about the signature or undisclosed messages while guaranteeing authenticity and integrity of the disclosed information.

Throughout this development, we maintain focus on how these constructions address the specific requirements of privacy-preserving electronic identity systems, setting the foundation for the comparative analysis with traditional schemes that follows.

3.1 Building a ZK Protocol for SDH

In this section, we describe a zero-knowledge protocol from the original 2004 work by Boneh, Boyen, and Shacham [2], enabling a prover to demonstrate possession of a valid SDH pair without revealing any secret information. This protocol will be the underlying building block for BBS signatures.

3.1.1 Problem Statement and Setup

We start by defining what we aim to prove in zero-knowledge. Let $\mathbb{G}_1, \mathbb{G}_2$ be SDH-groups with $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_1$ as respective generators. A prover (Alice) has a special pair (A, x) where $A = g_1^{1/(\gamma+x)}$ for some $x \in \mathbb{Z}_p$. The verifier (Bob) knows the public parameters $g_1, g_2, w = g_2^{\gamma}$, but doesn't know γ. Alice wants to convince Bob that she possesses a valid SDH pair without revealing either A or x. This means that:

- Alice must prove knowledge of both A and x without revealing either value
- Bob must be able to verify that A has the specific form $g_1^{1/(\gamma+x)}$
- The protocol must reveal nothing beyond the fact that Alice possesses a valid SDH pair

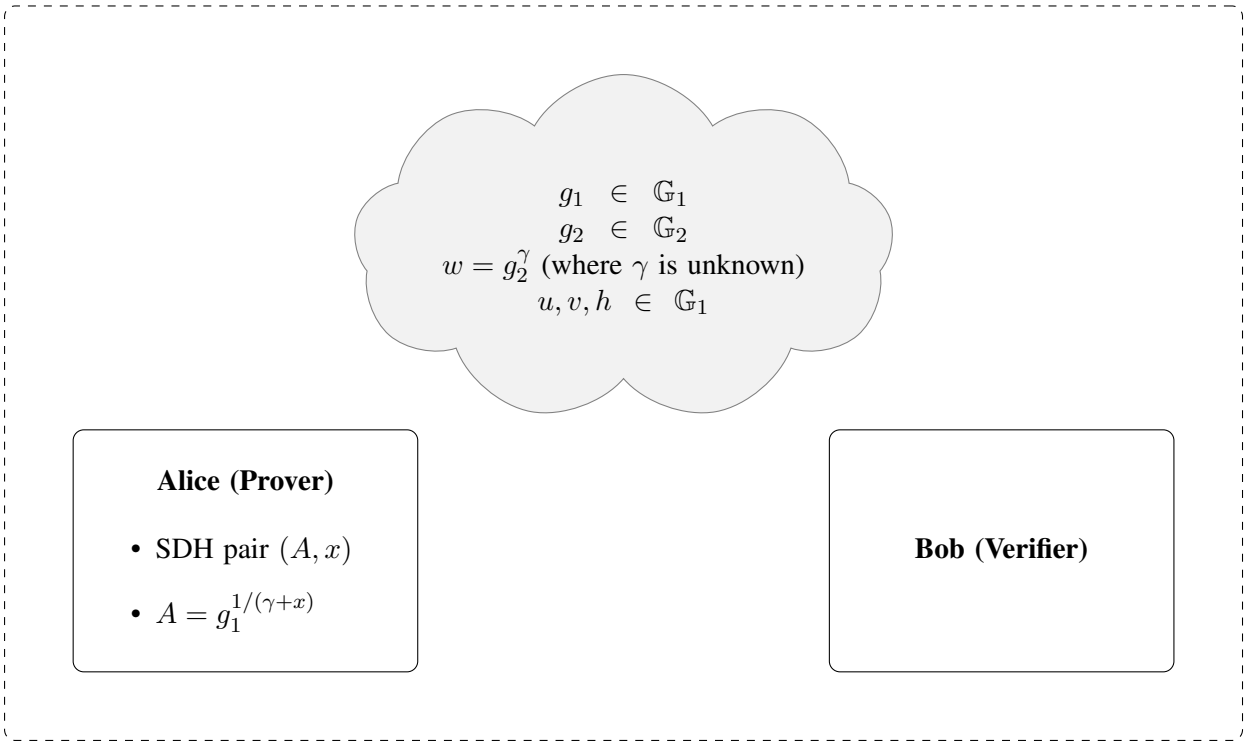


Figure 3.1. Setup of the SDH-Zero-Knowledge Protocol

As mentioned above, Bob needs to be able to verify that A is a valid SDH value without actually seeing A. Informally, Alice needs to "talk" about A without revealing it. Our strategy in this matter will be to hide A via a chosen encryption scheme and then construct the verification around the encrypted value.

Because of its advantages in bilinear groups, Boneh, Boyen and Shacham have decided to use Linear Encryption for this purpose:

$$\begin{aligned} T_1 &= u^{\alpha} \\ T_2 &= v^{\beta} \\ T_3 &= A \cdot h^{\alpha+\beta} \end{aligned} \tag{3.1}$$

where α and β are random values chosen by Alice, and u, v, h are public generators of $\mathbb{G}_1$. As we have seen in the previous chapter, this commitment scheme is:

- **Unconditionally Hiding**: The value A is perfectly masked by the random factors.
- **Computationally Binding**: Alice cannot find different values of A that open to the same commitment.
- **Non-malleable**: Others cannot modify the commitment to create related ones.

3.1.2 Verifying the SDH Relation

If $A = g_1^{1/(\gamma+x)}$, then $A^{(\gamma+x)} = g_1$. Using bilinear pairings this becomes:

$$e(A, g_2^{(\gamma+x)}) = e(g_1, g_2) \tag{3.2}$$

Since $w = g_2^{\gamma}$, we can write:

$$e(A, w \cdot g_2^{x}) = e(g_1, g_2) \tag{3.3}$$

This provides a way to verify the SDH property using bilinear pairings. However, we cannot use this equation directly because A is hidden inside T_3. Instead, we need to derive a relation that works with the hidden A.

Starting with the basic SDH verification equation, we expand it to work with our commitment scheme:

$$e(A, w \cdot g_2^{x}) = e(g_1, g_2) \tag{3.4}$$

$$\tag{3.5}$$

Since $T_3 = A \cdot h^{\alpha+\beta}$, we want to express this in terms of T_3. We can rewrite the equation as:

$$e(T_3/h^{\alpha+\beta}, w \cdot g_2^{x}) = e(g_1, g_2) \tag{3.6}$$

However, directly computing $T_3/h^{\alpha+\beta}$ would reveal α and β. Instead, we use the properties of bilinear pairings to rearrange this relation:

$$e(T_3, w \cdot g_2^{x})/e(h^{\alpha+\beta}, w \cdot g_2^{x}) = e(g_1, g_2) \tag{3.7}$$

Using bilinear properties, we expand this further:

$$e(T_3, w) \cdot e(T_3, g_2^{x})/(e(h, w)^{\alpha+\beta} \cdot e(h, g_2^{x})^{\alpha+\beta}) = e(g_1, g_2) \tag{3.8}$$

This can be rewritten as:

$$e(T_3, w) \cdot e(T_3, g_2)^{x}/(e(h, w)^{\alpha+\beta} \cdot e(h, g_2)^{x(\alpha+\beta)}) = e(g_1, g_2) \tag{3.9}$$

To simplify the exponents, we introduce helper values $\delta_1 = \alpha x$ and $\delta_2 = \beta x$. This gives us:

$$e(T_3, g_2)^{x} \cdot e(h, w)^{-\alpha-\beta} \cdot e(h, g_2)^{-\delta_1-\delta_2} = e(g_1, g_2)/e(T_3, w) \tag{3.10}$$

3.1.3 Proving Knowledge of the Commitment Values

Alice must prove that she knows the values α, β, x, δ_1, and δ_2 that satisfy five key relations. Notice the similarity to the previously discussed Schnorr Protocol 2.5.3.

$$u^{\alpha} = T_1 \tag{1}$$

$$v^{\beta} = T_2 \tag{2}$$

$$e(T_3, g_2)^{x} \cdot e(h, w)^{-\alpha-\beta} \cdot e(h, g_2)^{-\delta_1-\delta_2} = e(g_1, g_2)/e(T_3, w) \tag{3}$$

$$T_1^{x} \cdot u^{-\delta_1} = 1 \tag{4}$$

$$T_2^{x} \cdot v^{-\delta_2} = 1 \tag{5}$$

Relations (1) and (2) prove knowledge of α and β used in the commitment. Relation (3) verifies that the committed value A satisfies the SDH relation. Relations (4) and (5) ensure that $\delta_1 = \alpha x$ and $\delta_2 = \beta x$ are correctly formed, binding the commitment values to the SDH exponent x.

3.1.4 The Sigma Protocol

We now describe a Sigma protocol (commit-challenge-response), as introduced by Boneh, Boyen, and Shacham [2], to prove knowledge of values satisfying the five relations above:

Commitment Phase

Alice selects random values $r_\alpha, r_\beta, r_x, r_{\delta_1}, r_{\delta_2} \xleftarrow{R} \mathbb{Z}_p$ and computes:

$$R_1 = u^{r_\alpha} \tag{3.11}$$
$$R_2 = v^{r_\beta} \tag{3.12}$$
$$R_3 = e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}} \tag{3.13}$$
$$R_4 = T_1^{r_x} \cdot u^{-r_{\delta_1}} \tag{3.14}$$
$$R_5 = T_2^{r_x} \cdot v^{-r_{\delta_2}} \tag{3.15}$$

Alice sends $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ to Bob.

Challenge Phase

Bob sends a random challenge $c \xleftarrow{R} \mathbb{Z}_p$ to Alice.

Response Phase

Alice computes and sends back the responses:

$$s_\alpha = r_\alpha + c\alpha \tag{3.16}$$
$$s_\beta = r_\beta + c\beta \tag{3.17}$$
$$s_x = r_x + cx \tag{3.18}$$
$$s_{\delta_1} = r_{\delta_1} + c\delta_1 \tag{3.19}$$
$$s_{\delta_2} = r_{\delta_2} + c\delta_2 \tag{3.20}$$

Verification

Bob verifies the following equations:

$$u^{s_\alpha} \stackrel{?}{=} T_1^c \cdot R_1 \tag{3.21}$$
$$v^{s_\beta} \stackrel{?}{=} T_2^c \cdot R_2 \tag{3.22}$$
$$e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \stackrel{?}{=} (e(g_1, g_2)/e(T_3, w))^c \cdot R_3 \tag{3.23}$$
$$T_1^{s_x} \cdot u^{-s_{\delta_1}} \stackrel{?}{=} R_4 \tag{3.24}$$
$$T_2^{s_x} \cdot v^{-s_{\delta_2}} \stackrel{?}{=} R_5 \tag{3.25}$$

If all equations hold, Bob accepts the proof; otherwise, he rejects.

3.1.5 Security Analysis

The protocol satisfies the three key properties of zero-knowledge proofs:

Completeness

If Alice honestly follows the protocol and knows a valid SDH pair (A, x), she can always satisfy the verification equations. For example, for the first verification equation:

$$\begin{aligned} u^{s_\alpha} &= u^{r_\alpha + c\alpha} && (3.26)\\ &= u^{r_\alpha} \cdot u^{c\alpha} && (3.27)\\ &= u^{r_\alpha} \cdot (u^{\alpha})^{c} && (3.28)\\ &= u^{r_\alpha} \cdot T_1^{c} && (3.29)\\ &= R_1 \cdot T_1^{c} && (3.30) \end{aligned}$$

Similar calculations show that all verification equations will be satisfied.

Special Soundness

Let's recall that for a given protocol, special soundness is achieved by extracting the witness from two accepting transcripts with the same commitments but different challenges $c \neq c'$.

The shown SDH-protocol satisfies special soundness. From the two transcripts

$$(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}) \quad (3.31)$$
$$(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c', s'_\alpha, s'_\beta, s'_x, s'_{\delta_1}, s'_{\delta_2}) \quad (3.32)$$

with the same commitments but different challenges $c \neq c'$, we can extract the witness $(\alpha, \beta, x, \delta_1, \delta_2)$ in the following way:

From the first verification equation (3.21) for both transcripts, we get:

$$u^{s_\alpha} = T_1^{c} \cdot R_1 \quad (3.33)$$
$$u^{s'_\alpha} = T_1^{c'} \cdot R_1 \quad (3.34)$$

Dividing these equations, we get:

$$u^{s_\alpha - s'_\alpha} = T_1^{c-c'} \quad (3.35)$$
$$(3.36)$$

Since $T_1 = u^\alpha$, this gives:

$$u^{s_\alpha - s'_\alpha} = u^{\alpha(c-c')} \quad (3.37)$$
$$(3.38)$$

This implies $s_\alpha - s'_\alpha = \alpha(c - c')$, from which we can extract $\alpha = \frac{s_\alpha - s'_\alpha}{c-c'}$. Similarly, from equation two (3.22), we obtain $\beta = \frac{s_\beta - s'_\beta}{c-c'}$. Notice that T_1 and T_2 are now given by $T_1 = u^\alpha$ and $T_2 = v^\beta$.

Now consider equation four (3.24). Dividing the two instances, we get:

$$T_1^{s_x - s'_x} = u^{s_{\delta_1} - s'_{\delta_1}} \quad (3.39)$$

Substituting $T_1 = u^\alpha$ gives:

$$u^{\alpha(s_x - s'_x)} = u^{s_{\delta_1} - s'_{\delta_1}} \tag{3.40}$$

This implies $\alpha(s_x - s'_x) = s_{\delta_1} - s'_{\delta_1}$. Since we already know α, we can use this equation together with the similar equation derived from (3.25):

$$\beta(s_x - s'_x) = s_{\delta_2} - s'_{\delta_2} \tag{3.41}$$

To extract x, we examine the third verification equation (3.23) for both transcripts. After dividing and simplifying, we get:

$$e(T_3, g_2)^{s_x - s'_x} \cdot e(h, w)^{-(s_\alpha - s'_\alpha) - (s_\beta - s'_\beta)} \cdot e(h, g_2)^{-(s_{\delta_1} - s'_{\delta_1}) - (s_{\delta_2} - s'_{\delta_2})} = \left(\frac{e(g_1, g_2)}{e(T_3, w)}\right)^{c - c'} \tag{3.42}$$

Substituting our known values:

$$e(T_3, g_2)^{s_x - s'_x} \cdot e(h, w)^{-\alpha(c - c') - \beta(c - c')} \cdot e(h, g_2)^{-\alpha(s_x - s'_x) - \beta(s_x - s'_x)} = \left(\frac{e(g_1, g_2)}{e(T_3, w)}\right)^{c - c'} \tag{3.43}$$

Simplifying:

$$e(T_3, g_2)^{s_x - s'_x} \cdot e(h, w)^{-(\alpha + \beta)(c - c')} \cdot e(h, g_2)^{-(\alpha + \beta)(s_x - s'_x)} = \left(\frac{e(g_1, g_2)}{e(T_3, w)}\right)^{c - c'} \tag{3.44}$$

This equation can be solved for $s_x - s'_x$, giving us:

$$x = \frac{s_x - s'_x}{c - c'} \tag{3.45}$$

With α, β, and x extracted, we can also calculate $\delta_1 = \alpha x$ and $\delta_2 = \beta x$.

Finally, we can recover the SDH pair (A, x) by computing $A = T_3 / h^{\alpha + \beta}$ and verifying that $e(A, w \cdot g_2^x) = e(g_1, g_2)$, confirming that $A = g_1^{1/(\gamma + x)}$.

The successful extraction of all these values from two valid transcripts proves that our protocol satisfies the special soundness property.

Zero-Knowledge

To prove the zero-knowledge property, we need to construct a simulator that can generate valid-looking transcripts without knowing the witness. The simulator:

1. Chooses random values $T_1, T_2, T_3 \in \mathbb{G}_1$
2. Selects a random challenge $c \xleftarrow{R} \mathbb{Z}_p$
3. Chooses random responses $s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2} \xleftarrow{R} \mathbb{Z}_p$
4. Computes commitments R_1, R_2, R_3, R_4, R_5 that make the verification equations hold:

$$R_1 = u^{s_\alpha} \cdot T_1^{-c} \tag{3.46}$$
$$R_2 = v^{s_\beta} \cdot T_2^{-c} \tag{3.47}$$
$$R_3 = e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot (e(g_1, g_2) / e(T_3, w))^{-c} \tag{3.48}$$
$$R_4 = T_1^{s_x} \cdot u^{-s_{\delta_1}} \tag{3.49}$$
$$R_5 = T_2^{s_x} \cdot v^{-s_{\delta_2}} \tag{3.50}$$

5. Outputs the transcript $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$

The simulated transcript is computationally indistinguishable from a real interaction, assuming the Decision Linear assumption holds in $\mathbb{G}_1$.

3.1.6 Non-Interactive Version via Fiat-Shamir

For practical use, we convert this interactive protocol to a non-interactive one using the Fiat-Shamir heuristic. The prover:

1. Computes $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ as in the interactive protocol
2. Computes the challenge as $c = H(T_1 \| T_2 \| T_3 \| R_1 \| R_2 \| R_3 \| R_4 \| R_5 \| M)$, where H is a cryptographic hash function treated as a random oracle and M is an optional message to be signed
3. Computes responses $(s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ as in the interactive protocol
4. The signature/proof consists of $(T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$

3.2 Constructing BBS

We now describe the construction of the full BBS signature scheme as it was originally introduced by Boneh, Boyen, and Shacham [2]. The scheme consists of three main algorithms: $\mathsf{KeyGen}(\mathsf{n})$, $\mathsf{Sign}(\mathsf{gpk}, \mathsf{gsk[i]}, \mathsf{M})$, and $\mathsf{Verify}(\mathsf{gpk}, \mathsf{M}, \sigma)$.

3.2.1 System Setup

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups of prime order p with an efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ such that $\psi(g_2) = g_1$ where g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$. The system selects additional random generators $h, u, v \in \mathbb{G}_1$ such that their relative discrete logarithms are unknown, i.e., no one knows values $a, b \in \mathbb{Z}_p^*$ such that $u^a = v$ or $u^b = h$.

3.2.2 Key Generation

Since BBS was originally conceived as a group signature scheme, we will introduce it as such. However, we will not explore the workings and implications of these group-signature aspects too deeply since they are not directly relevant to our E-ID context.

The KeyGen algorithm (Algorithm 1) shown below takes an additional parameter: the number n of members of the group. Moreover, the BBS-KeyGen-algorithm generates three types of keys: a group public key (gpk) used for signature verification, individual secret signing keys (gsk[i]) for each group member, and a special tracing key (gmsk) held by the group manager. This tracing key enables the group manager to identify which group member created a particular signature.

Notice how the KeyGen algorithm creates an SDH tuple (A_i, x_i) for each user, where $A_i = g_1^{1/(\gamma + x_i)}$ serves as a signature on the user's secret identifier x_i.

3.2.3 Signature Generation

In the Sign algorithm (Algorithm 2) shown below, the signer creates a Linear Encryption of their SDH component A_i in the form of (T_1, T_2, T_3), computes commitments $(R_1, R_2, R_3, R_4, R_5)$ for the zero-knowledge proof, and then uses the hash function to generate a challenge c. These steps are identical to the established SDH-Protocol with the exception that a message M is additionally given to the hash function. The responses $(s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ complete the proof, allowing verification without revealing the SDH pair.

Algorithm 1 $\mathsf{KeyGen}(n)$

Generate system parameters:

$g_2 \xleftarrow{R} \mathbb{G}_2$	// Random generator
$g_1 \leftarrow \psi(g_2)$	// Mapped generator in $\mathbb{G}_1$
$h \xleftarrow{R} \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$	// Random non-identity element
$\xi_1, \xi_2 \xleftarrow{R} \mathbb{Z}_p^*$	// Secret values for tracing
$u, v \in \mathbb{G}_1, (u^{\xi_1} = v^{\xi_2} = h)$	// Derived elements for linear encryption

Generate master secret and public key:

$\gamma \xleftarrow{R} \mathbb{Z}_p^*$	// Master secret for key issuance
$w \leftarrow g_2^{\gamma}$	// Public commitment to γ

Generate user keys:

For $i = 1$ to n :

$x_i \xleftarrow{R} \mathbb{Z}_p^*$	// User's secret identifier
$A_i \leftarrow g_1^{1/(\gamma + x_i)}$	// SDH signature component

Return:

$\mathsf{gpk} = (g_1, g_2, h, u, v, w)$	// Group public key
$\mathsf{gmsk} = (\xi_1, \xi_2)$	// Group manager's key for tracing
$\mathsf{gsk}[i] = (A_i, x_i)$ for $i \in \{1, \ldots, n\}$	// User private keys

3.2.4 Signature Verification

The Verify algorithm (Algorithm 3) provides the counterpart to signing. It allows any party with the group public key to validate a signature without learning the signer's identity. The verifier recomputes the expected commitment values $(R_1, R_2, R_3, R_4, R_5)$ using the provided signature components and then confirms that these values, when hashed with the message, produce the challenge value c included in the signature. If the verification equation holds, it confirms that the signer possesses a valid SDH pair issued by the group manager, without revealing which specific pair was used.

Algorithm 2 $\mathsf{Sign}(\mathsf{gpk}, \mathsf{gsk}, M)$

Input:

$\mathsf{gpk} = (g_1, g_2, h, u, v, w)$ // Group public key
$\mathsf{gsk} = (A, x)$ // User's private key
$M \in \{0,1\}^*$ // Message to sign

Compute first-round values:

$\alpha, \beta \xleftarrow{R} \mathbb{Z}_p$ // Random blinding factors
$T_1 \leftarrow u^{\alpha}$ // First component of linear encryption
$T_2 \leftarrow v^{\beta}$ // Second component
$T_3 \leftarrow A_i \cdot h^{\alpha+\beta}$ // Third component
$\delta_1 \leftarrow x_i\alpha, \delta_2 \leftarrow x_i\beta$ // Helper values

Compute commitments:

$r_\alpha, r_\beta, r_x, r_{\delta_1}, r_{\delta_2} \xleftarrow{R} \mathbb{Z}_p$ // Random values for commitments
$R_1 \leftarrow u^{r_\alpha}$
$R_2 \leftarrow v^{r_\beta}$
$R_3 \leftarrow e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}}$
$R_4 \leftarrow T_1^{r_x} \cdot u^{-r_{\delta_1}}$
$R_5 \leftarrow T_2^{r_x} \cdot v^{-r_{\delta_2}}$

Compute challenge:

$c \leftarrow H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5) \in \mathbb{Z}_p$ // Fiat-Shamir challenge

Compute responses:

$s_\alpha \leftarrow r_\alpha + c\alpha$
$s_\beta \leftarrow r_\beta + c\beta$
$s_x \leftarrow r_x + cx_i$
$s_{\delta_1} \leftarrow r_{\delta_1} + c\delta_1$
$s_{\delta_2} \leftarrow r_{\delta_2} + c\delta_2$

Return:

$\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ // The group signature

Algorithm 3 $\mathsf{Verify}(\mathsf{gpk}, M, \sigma)$

Input:

$\mathsf{gpk} = (g_1, g_2, h, u, v, w)$	// Group public key
$M \in \{0,1\}^*$	// Message
$\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$	// Group signature

Re-derive verification values:

$\tilde{R}_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c}$
$\tilde{R}_2 \leftarrow v^{s_\beta} \cdot T_2^{-c}$
$\tilde{R}_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\delta_1}}$
$\tilde{R}_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\delta_2}}$
$\tilde{R}_3 \leftarrow e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot (e(T_3, w)/e(g_1, g_2))^c$

Verify challenge:

Check if $c \stackrel{?}{=} H(M, T_1, T_2, T_3, \tilde{R}_1, \tilde{R}_2, \tilde{R}_3, \tilde{R}_4, \tilde{R}_5)$	// Validate signature

Return:

ACCEPT if check succeeds; REJECT otherwise	// Verification result

3.2.5 Signature Length and Efficiency

A BBS signature consists of three elements of $\mathbb{G}_1$ (the values T_1, T_2, T_3) and six elements of $\mathbb{Z}_p$ (the values $c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}$). Following the original parameters suggested by Boneh, Boyen, and Shacham [2], we take p to be a 170-bit prime and use a group $\mathbb{G}_1$ where each element is 171 bits, resulting in a total signature size of 1533 bits (192 bytes). However, in modern settings it is common to use prime order groups of at least 256 bits (leading to larger signatures) in order to achieve approximately 128-bit security.

Creating a BBS signature requires eight multi-exponentiations and no pairing computation, since for the signer all pairing operations can be cached or computed without evaluating a pairing.

Verifying a BBS signature requires computing six multi-exponentiations and one pairing computation.

3.2.6 Security Properties

The BBS signature scheme provides the following security guarantees:

- **Unforgeability**: Without knowing a valid SDH pair (A, x), it is computationally infeasible to forge a valid signature, assuming the q-Strong Diffie-Hellman assumption holds.
- **Full Anonymity**: The scheme provides anonymity for signers—a signature does not reveal which member created it. This property holds assuming the Decision Linear assumption is hard in $\mathbb{G}_1$.
- **Full-Traceablility**: All signatures trace to a member of the signing party. This property holds even if multiple members and the group manager collude.
- **Exculpability (Non-frameability)**: No entity, including the group manager, can forge signatures on behalf of honest group members. This is a consequence of the full-traceability property.

3.3 BBS in the E-ID Context

3.3.1 Unlinkability in BBS

The BBS signature scheme, as constructed above, provides the property of unlinkability, which ensures that two signatures created by the same user cannot be linked together. This stems from the use of fresh randomness in every signature generation process.

In particular, each signature includes a randomized linear encryption of the user's secret key component A_i, formed as:

$$(T_1, T_2, T_3) = (u^{\alpha}, v^{\beta}, A_i \cdot h^{\alpha+\beta}),$$

where $\alpha, \beta \in \mathbb{Z}_p$ are freshly chosen for every signature. Since the zero-knowledge proof encapsulates knowledge of the SDH tuple (A_i, x_i) without revealing it, and the randomizers (α, β) are not reused, the resulting signatures are computationally unlinkable under the Decision Linear assumption.

However, the signature scheme introduced above is not yet suitable for our authentication use case. To see why, we have to look again at the role distribution.

Why the raw BBS Group-Signature Model does not fit an E-ID Credential System

In an E-ID scenario the roles are unambiguous: The issuer (e.g. the government) signs a vector of attributes for the holder (e.g. citizen Alice) who later proves possession of that signature—ideally selectively disclosing just the attributes a verifier requests—and doing so unlinkably across multiple sessions.

However, the original BBS group signatures follow a different trust model:

1. A *group manager* issues to each member i a private signing key $\mathsf{gsk}[i] = (A_i, x_i)$ which enables that member to create fresh group signatures on any message.
2. A separate *opening authority* (holding a tracing key) can later deanonymise a given group signature.

When we map those roles directly to the E-ID-model, this gives us two unsatisfactory designs:

(i) Issuer as group manager, citizens as members. Alice would receive a valid SDH pair (A, x). Because BBS lets any member sign arbitrary messages, Alice could now mint new credentials—attributes the issuer never endorsed. This violates the credential model.

(ii) Issuer as only member, single group key shared. Here the government keeps the only SDH pair and signs every citizen's document. Each signature σ is static. When Alice shows it twice, the verifier sees the same tuple, so presentations are trivially linkable. Issuing a fresh signature per session would restore privacy but is infeasible in practice.

We conclude that the original BBS group-signature scheme grants members signing power or forces linkability. Therefore, we need to implement some alteration to the original group signature scheme in order to properly obtain unlinkablility and selective disclosure:

- The issuer signs all credentials once, producing a modified SDH-pair (A, x) tied to those values.
- The holder can only re-randomise that signature—never forge new attributes—while proving knowledge in zero-knowledge and revealing any chosen subset.
- There is no tracing key, no Linear-Encryption layer, and security rests solely on the q-SDH assumption.

This transformation eliminates the flaws above and aligns BBS with the privacy and integrity requirements of modern E-ID systems.

3.3.2 Tessaro-Zhu BBS-Signatures

In this section we now give an altered and improved version of BBS signatures based on the CFRG draft [7], which in turn is based on the work of Tessaro and Zhu [5]. The presented BBS scheme turns an issuer's one-time credential into a compact, unlinkable presentation that reveals only the attributes a verifier requests and eliminates the above identified issues. We describe the following core operations:

1. **Key generation** (4) is run once by the credential issuer. It outputs both a signing key-pair and a publicly verifiable generator set that determines which attributes can later be disclosed.
2. **Credential issuance** (5) produces a *core BBS signature*$\sigma = (A, e)$ on a vector of messages $\boldsymbol{m}$. This signature is sent to (and stored by) the holder together with the clear-text attributes.
3. **Proof generation** (7) allows the holder to re-randomise σ and prove, in zero knowledge, that the re-randomised value authenticates *all* attributes while revealing only a subset.
4. **Proof verification** (8) confirms the proof and returns a simple VALID/INVALID decision to the relying party.

Key generation

The key-generation algorithm (Algorithm 4) remains similar to the original construction and follows section 3.4 from the draft. The issuer samples a fresh secret scalar $\gamma \leftarrow \mathbb{Z}_p^*$, computes the public key $w = g_2^{\gamma} \in \mathbb{G}_2$, and derives $\mathcal{G}$ for the chosen attribute length L. The triple $(w, \mathcal{G}, L)$ is published. The secret key γ is kept private and never leaves the issuer.

Algorithm 4 $\mathsf{CoreKeyGen}(L)$

Input:
$L \in \mathbb{N}$ // number of messages per credential

Generate secret / public key:
$\gamma \xleftarrow{R} \mathbb{Z}_p^*$ // Secret key
$w \leftarrow g_2^{\gamma}$ // Public key in $\mathbb{G}_2$

Derive message generators:
$(H_1, \ldots, H_L) \leftarrow \mathsf{CreateGenerators}(L)$

Return:
$\gamma,\ w,\ \mathcal{G} = (H_1, \ldots, H_L)$ // Secret key, public key, generators

Credential Issuance

Algorithm 4 follows draft section 3.6.1. The algorithm takes the before generated public and private keys, generators as well as a message vector $\boldsymbol{m}$ of length L. In the E-ID setting, the messages correspond to the verifiable credentials of the credential holder—date-of-birth, surname, municipality etc. This operation is thus called once per citizen.

The algorithm operates analogous to the second half of the original BBS KeyGen-algorithm (Algorithm 1). The signature-tuple (A, e) is the analog to the 'user key' and constitutes a valid q-SDH pair with the important distinction that A now incorporates a Pedersen-style commitment to $\boldsymbol{m}$. This (computationally) binds the message-vector to the signature while (unconditionally) hiding individual attributes under the (Elliptic Curve) Discrete Logarithm Assumption. Note that the signature-tuple (A, e) consists of an element in $\mathbb{G}_1$ and a scalar and is thus constant in size.

Algorithm 5 CoreSign$(\gamma, w, \mathcal{G}, \boldsymbol{m})$

Input:

$\gamma \in \mathbb{Z}_p^*$ // Issuer's secret key

$w = g_2^{\gamma} \in \mathbb{G}_2$ // Public key

$\mathcal{G} = (H_1, \ldots, H_L)$ // Message generators

$\boldsymbol{m} = (m_1, \ldots, m_L) \in \mathbb{Z}_r^L$ // Messages (attributes)

e value:

$e \xleftarrow{R} \mathbb{Z}_p^*$ // deterministically chosen in practice

Commitment:

$$B \leftarrow g_1 \prod_{i=1}^{L} H_i^{m_i}$$

Signature element:

$A \leftarrow B^{\frac{1}{\gamma+e}} \quad \in \mathbb{G}_1$ // Inversion mod r

Return:

$\sigma = (A, e)$ // Tessaro–Zhu BBS signature

The below verification algorithm (Algorithm 6) checks the validity of a given signature-tuple (A, e) over the plaintext-messages. In the E-ID context however, a credential holder would always present a ZKPoK of a signature tuple using the given CoreProofGen algorithm (Algorithm 7).

Algorithm 6 CoreVerify$(w, \sigma, \mathcal{G}, \boldsymbol{m})$

Input: $w \in \mathbb{G}_2,\ \sigma = (A, e),\ \mathcal{G},\ \boldsymbol{m}$

1. Rebuild B as in Alg. 5

2. Pairing check:

Accept iff $e(A,\ w \cdot g_2^{e}) \ = \ e\big(B,\ g_2^{-1}\big)$

Proof Generation

The following CoreProofGen-algorithm (Algorithm 7) is based on Appendix B of Tessaro and Zhu [5]. To enable unlinkability, the credential holder blinds her signature using the randomly chosen pair of scalars (r_1, r_2):

$$\bar{A} = A^{r_1 r_2}, \quad D = B^{r_2}, \quad \bar{B} = D^{r_1}\,\bar{A}^{-e}.$$

Because $e(\bar{A},\, w) = e(\bar{B},\, g_2)$, $(\bar{A}, \bar{B})$ is a fresh SDH tuple under the same public key w. Thus every run of the algorithm produces completely new group elements, which destroys any link the verifier could draw between sessions. The holder then follows through with the rest of the ZKP as is described in the algorithm. Notice how the size of proof π is dependent on the number of hidden attributes $|\bar{\mathcal{R}}|$.

Algorithm 7 $\mathsf{CoreProofGen}(w, \sigma, \mathcal{G}, \boldsymbol{m}, \mathcal{R})$

Input:
$w,\ \sigma = (A, e),\ \mathcal{G} = (H_1, \ldots, H_L)$,
$\boldsymbol{m} = (m_1, \ldots, m_L) \in \mathbb{Z}_r^L,\ \mathcal{R} \subseteq \{1, \ldots, L\}$ // Indices to disclose

Randomisers:
$(r_1, r_2, e', r_1', r_3') \xleftarrow{R} \mathbb{Z}_r^5, \qquad (m_j')_{j \in \bar{\mathcal{R}}} \xleftarrow{R} \mathbb{Z}_r^{|\bar{\mathcal{R}}|}$ // $\bar{\mathcal{R}} = \{1..L\} \setminus \mathcal{R}$

Blind signature:
$\bar{A} \leftarrow A^{r_1 r_2}, \quad B \leftarrow g_1 \prod_{i=1}^{L} H_i^{m_i}, \quad D \leftarrow B^{r_2}, \quad \bar{B} \leftarrow D^{r_1}\,\bar{A}^{-e}$

Commitments:
$T_1 \leftarrow \bar{A}^{e'}\, D^{r_1'}, \qquad T_2 \leftarrow D^{r_3'} \prod_{j \in \bar{\mathcal{R}}} H_j^{m_j'}$

Fiat–Shamir challenge:
$c \leftarrow \mathsf{ProofChallengeCalculate}(\bar{A}, \bar{B}, D, T_1, T_2, \mathcal{R}, \boldsymbol{m}_{\mathcal{R}})$

Responses:
$\hat{e} = e' + e\,c, \qquad \hat{r_1} = r_1' - r_1\,c, \qquad \hat{r_3} = r_3' - r_2^{-1} c,$
$\hat{m}_j = m_j' + m_j\,c \quad (\forall j \in \bar{\mathcal{R}})$

Return:
$\pi = \big(\bar{A}, \bar{B}, D,\ \hat{e}, \hat{r}_1, \hat{r}_3, (\hat{m}_j)_{j \in \bar{\mathcal{R}}}, c\big)$ // Zero-knowledge proof

Verification (Verifier side)

CoreProofVerify is based on section 3.6.4 of the draft. In a first step, the commitments T_1 and T_2 need to be recomputed from the responses packed into π. If the prover cheated in any of the response scalars, (T_1, T_2) will not line up with the original commitments. The verifier then recomputes $c = \mathsf{ProofChallengeCalculate}(\bar{A}, \bar{B}, D, T_1, T_2, \mathcal{R}, \boldsymbol{m}_{\mathcal{R}})$. The recomputed challenge has to equal the challenge given with the proof, this ties the whole transcript to a single hash invocation - the standard soundness argument of the Fiat-Shamir heuristic. Finally, the verifier checks

$$e(\bar{A},\, w)\; e(\bar{B},\, g_2^{-1}) \stackrel{?}{=} 1_{G_T}.$$

Because $(\bar{A}, \bar{B})$ form an SDH pair under w, the product of pairings indeed collapses to the identity element of G_T.

Algorithm 8 $\mathsf{CoreProofVerify}(w, \pi, \mathcal{G}, \boldsymbol{m}_{\mathcal{R}}, \mathcal{R})$

Input:

$w,\ \pi = (\bar{A}, \bar{B}, D, \hat{e}, \hat{r}_1, \hat{r}_3, (\hat{m}_j)_{j\in\bar{\mathcal{R}}}, c),\ \mathcal{G},\ \boldsymbol{m}_{\mathcal{R}},\ \mathcal{R}$

1. Recompute T_1 and T_2 :

$T_1 \ = \ \bar{B}^{\,c}\ \bar{A}^{\,\hat{e}}\ D^{\,\hat{r}_1}$

$B_v \ = \ g_1 \prod_{i\in\mathcal{R}} H_i^{\,m_i}$

$T_2 \ = \ B_v^{\,c}\ D^{\,\hat{r}_3}\ \prod_{j\in\bar{\mathcal{R}}} H_j^{\,\hat{m}_j}$

2. Recompute challenge:

$c \leftarrow \mathsf{ProofChallengeCalculate}(\bar{A}, \bar{B}, D, T_1, T_2, \mathcal{R}, \boldsymbol{m}_{\mathcal{R}})$

Reject if $c \neq c$

3. Pairing check:

Accept iff $e(\bar{A},\ w)\ e(\bar{B},\ g_2^{-1}) = 1_{G_T}$

Return: VALID if all checks pass, else INVALID

Conclusion

In this chapter we have moved from the original Boneh-Boyen-Shacham group signature scheme to a selective-disclosure BBS scheme that satisfies the core requirements of a digital identity system:

- A one-time credential issuance phase produces an constant size signature (A, e) bound to a vector of credentials.
- Holders can later create compact, unlinkable proofs that reveal only the attributes a verifier requests - no online interaction with the issuer is required.
- Security rests on the q-SDH and ECDL assumptions; anonymity and unforgeability are formally proven in the Tessaro-Zhu model.

Together, these properties demonstrate that BBS signatures provide a privacy-preserving alternative to traditional signature schemes such as ECDSA. The next chapter will elaborate on the use of these schemes in real-world scenarios and extensively compare both approaches.

Chapter 4

Comparing BBS and SD-JWT/ECDSA

The Swiss Federal Office of Information Technology, Systems and Telecommunication (FOITT) is responsible for developing and operating the Trust Infrastructure and mobile app (wallet) for the Swiss electronic identity[8]. The following table 4.1 outlines the technology stack selected for the public beta version of the Swiss E-ID currently in use:

Aspect	**Selected Technology**
Identifiers	W3C Decentralized Identifiers v1.0 did:tdw/did:webvh v0.3 (as DID Method)
Status Mechanisms	Token Status List draft 3
Trust Protocol	Swiss Trust Protocol version 0.1 (based on VCs)
Communication Protocol	OID4VCI – draft 13 (for credential issuance) OID4VP – draft 20 (for credential verification)
Payload Encryption	JWE[1] (as proposed by the communication protocol)
VC-Format & Signature-Scheme	SD-JWT VC – draft 4 SD-JWT draft 10 ECDSA P-256
Device Binding Scheme	**Hardware-based** device binding (depending on capabilities provided by Android or Apple mobile devices) **Software-based**[1] device binding implemented by wallets
VC appearance	Overlays Capture Architecture (for visualization of VCs)[1]

Table 4.1. Selected technology and references for the public beta version currently in use [9]

We notice their choice of SD-JWT in combination with the ECDSA signature-scheme for *VC-Format & Signature-Scheme*. This chapter will give an overview of this approach in section 4.1, outline the implied advantages (section 4.2) and disadvantages (section 4.3) and compare it to a possible approach using BBS signatures (sections 4.4-4.5).

4.1 Overview of SD-JWT with ECDSA Signatures

The JSON Web Token (JWT) defines a format for representing claims (VCs for example) to be transferred between two parties. JWTs are compact, URL-safe and enable a standardized way of digitally signing said claims in order to ensure integrity and authenticity [10]. However, in the E-ID context it makes sense to use the Selective Disclosure JWT (SD-JWT) format. SD-JWT is an extension of the standard JWT to support revealing only a certain subset of claims on demand. This format can be used with "traditional" cryptography (e.g. no zero knowledge cryptography) and thus ensures compatibility with existing infrastructure and hardware. In principle, SD-JWT can be used in tandem with any valid signature scheme. In the following sections however, we assume SD-JWT is combined with ECDSA

signatures since this is the approach taken by the swiss E-ID program.

In SD-JWT, sensitive data fields are not embedded in cleartext. Instead, each selectively-disclosable claim (VC) is replaced by a cryptographic digest in the signed JWT payload. This digest is typically created using the salted hash of the corresponding claim. The issuer computes a uniqe salt per claim and includes the hash of the claim value (plus salt) in the JWT. The holder receives, alongside the token, the original values and salts for those claims. Later, when presenting credentials, the holder can choose which claims to disclose by providing the plaintext value and its salt for each revealed field. The verifier, who sees the issuer's signed token with hashed claims, can hash the disclosed value (using the provided salt) and check that it matches the digest in the JWT. If the match succeeds, the verifier is convinced the revealed data was part of the original signed credential while the unrevealed fields remain hashed [11].

4.2 Advantages of SD-JWT with ECDSA Signatures

There are several reasons for which the FOITT might have chosen the combination of SD-JWTs and ECDSA signatures. A major reason is to maximize compatibility and ease of adoption. SD-JWT was explicitly designed to use well established, traditional cryptography and integrate cleanly into systems that already use JWTs like the OpenID ecosystem. The scheme benefits form existing libraries and experties by using such established technologies.

Moreover, ECDSA signatures are widely supported, including by secure elements in modern smartphones and smartcards. This means signing keys can be stored and used in tamper-resistant hardware. In practice, national digital ID programs can leverage the devices of citizens to perform ECDSA signatures since those chips commonly support P-256 elliptic curve operations.

Additionally, government regulators tend to prefer well-audited algorithms. ECDSA is approved by the SOG-IS (Senior Officials Group - Information Systems Security) Crypto Evalutation Scheme, which sets the benchmark for approved cryptographic algorithms in European government and high-assurance contexts [12]. Furthermore, the Digital Identity Wallet of the European Union (EUDI Wallet) already adopts SD-JWT as VC-Format in tandem with ECDSA signatures [13].

In summary, the SD-JWT/ECDSA design prioritizes simplicity and interoperability. This makes it the pramatic choice for early deployments of digital credentials.

4.3 Privacy Vulnerabilities in SD-JWT/ECDSA

While SD-JWT/ECDSA adds selective disclosure capability to JWTs, this approach has significant privacy limitations. The primary concern is linkablility: if a user presents the same SD-JWT credential multiple times, those presentations can potenitally be linked to each other. For example, suppose Alice uses her national E-ID (implemented as an SD-JWT) at a bank and later at a telecom provider. Even if she only discloses the necessary claims to each party, the underlying signed JWT (including all the hashed claims she chose not to reveal) will be the same in both transactions. If these verifiers (or and outside observer who sees both) compare the tokens, they would notice the identical hash values and metadata and realize it is the ssame credential and thus the same user.

This vulnerability is inherent: the token carries static, repeated information. The very feature that makes SD-JWT easy to implement—reusing a single signed structure—creates a privacy weakness when that structure is reused across multiple contexts.

The consequences of this are not theoretical. In practice, linkable credentials allow different services to correlate user activity, even without cooperation from the user. Over time, such linkability can lead to the creation of detailed behavioral profiles, covering where and when the user has authenticated, what claims were presented, and potentially why. This undermines one of the core promises of self-sovereign digital identity: that individuals remain in control not just of what they share, but also of when, where, and with whom they share it [14]. Unlinkability is therefore not a luxury feature. It is essential for preserving privacy in a system where users regularly present credentials to different entities.

A simple mitigation to the linkability problem would be to avoid reusing the same tokens too many times. For example, an issuer could provide the holder with multiple copies of a credential. Each copy is a seperate SD-JWT token with different salts. When a holder presents her credentials to a verifier, she (or her wallet-application) can choose one of the tokens uniformly at random. With this approach, correlating presentations becomes more difficult since the probability of colluding verifiers to see the same token becomes smaller. However, this complicates the credential management (the wallet must track multiple versions of what is logically the same credential) while still not totally solving the linkability problem. A report of the Dutch government highlights this issue in the following paragraph:

> This solution [issuing multiple tokens] is not ideal: it introduces extra load for attestation issuers as well as additional complexity in the software. When using ordinary signature schemes such as ECDSA to sign the attestations, we see however no practical alternative. In the long term more sophisticated cryptographical schemes such as Idemix or BBS+ can help. These schemes offer multishow unlinkability out of the box in a more elegant way. However as these schemes currently do not meet the requirements of the ARF [Architecture and Reference Framework], they are considered out of scope for now. [15]

Another issue is over-disclosure of information. Over-disclosure describes the scenario where the verifier learns more information than is necessary. The selective disclosure ability of SD-JWTs is based on the all-or-nothing principle: a holder either reveals an entire claim or withholds it (in which case only its hash is shown). There is no built-in way to prove statements about a hidden value (such that a number lies within a range) without revealing the value itself. For example, we consider a common scenario where a verifier ony needs to know if the holder is over 18 years old. With an SD-JWT credential, unless the issuer had the foresight to include a boolean claim like "isOver18:true", the holder would have to disclose their full birth date to prove adulthood. This pattern can repeat itself for several attributes (disclosing a full home address when only a city is needed, revealing an exact salary when only a range is needed etc.) and is a direct consequence of SD-JWTs lack of proof capability.

If implemented naively, there exists also a third danger beyond linkability and over-disclosure: Because an SD-JWT token includes hashes of all selectively disclosable fields, the mere presence and structure of those hashes can leak information to a verifier. For example, the verifier can tell which claims exist in the credential and how many there are even if their values remain hidden. In some cases, this structural knowledge alone could be sensitive. For instance, let us consider that a credential contains an array of the names of the holder's children. Even if the holder does not disclose that list, the verifier might register three hashed entries correspondig to "children" and deduces that the holder has three children. Without countermeasures such as decoy values and padding the SD-JWT format will leak information based on the document structure. These countermeasures add complexity and size.

The next sections show how a BBS signature suite eliminates these three leakage vectors by construction.

4.4 BBS Signatures as a Privacy-Preserving Alternative

As we have seen in the previous chapters, BBS signatures provide native unlinkable selective disclosure: an issuer signs multiple attributes once while the holder can later derive fresh zero knowledge proofs that reveal only a chosen subset of those attributes to a verifier. Because the verifier must know (i) which individual messages were covered by the BBS signature, (ii) which of those messages the holder has chosen to reveal in the current presentation, and (iii) the plaintext value of each revealed message so the proof can be checked, the surrounding credential container needs dedicated structure for these elements. Plain containers such as JWTs lack those structures, so they cannot natively support BBS-style multi-message schemes. For this reason, recent standardization work has begun packaging zero-knowledge-based schemes—such as BBS signatures—into JSON friendly container formats like JSON Web Proofs [16]. This development makes a replacement of current-in-use VC-formats such as SD-JWT more realistic. This section introduces one of the container options, explains how the BBS cryptosuite

removes the privacy limitations identified in section 4.3 and sketches how a Swiss or EU E-ID wallet could bind BBS proofs to the secure hardware that today only speaks ECDSA.

4.4.1 JSON Web Proof (JWP)

The JSON Object Signing and Encryption (JOSE) specification provides a standardized representation of digital content using JSON (JavaScript Object Notation) data structures. Within the original JOSE framework, the JSON Web Signatures (JWS) format has the responsibility of enveloping and signing one single payload such as a JWT. The IETF JSON Web Proof draft defines a multi-payload succesor to JWS (JSON Web Signatures) that natively protects multiple payloads [16]. JSON Web Proofs are designed to replace existing implementations using JWTs with minimal new code.

There are also other specified container formats such as the Data-Integrity Proof Container by W3C [17]. This format was created for linked-data credentials such as JSON-LD. However, since the E-ID Program makes use of the OpenID-Ecosystem (see table 4.1), we found JWPs a more relevant container-format.

4.4.2 How BBS Eliminates SD-JWT Privacy Gaps

Unlinkability. Each BBS proof includes fresh randomness and a verifier nonce, so two presentations derived from the same credential are computationally indistinguishable - even if the same claims are revealed both times.

Minimal disclosure. BBS lets the holder omit entire attributes at will, so no hash placeholders appear in the presentation. That already avoids the "all-hashes-are-visible" fingerprint of SD-JWT. However, we have still an attribute-level reveal: proving "date of birth ≥ 18" without showing the exact date needs an additional predicate proof. Since BBS already employs zero-knowledge technology, the addition of predicate proofs such as Bulletproofs [18] should be feasable in the future.

No structural metadata leak. Since undisclosed statements are simply absent from the proof, the verifier cannot infer how many hidden fields exist or whether an array has one or ten elements.

4.4.3 Binding BBS Proofs to Secure Hardware

Current smartphone secure elements (Apple S Secure Enclave, Android StrongBox) expose ECDSA / EdDSA but not the bilinear-pairing operations that BBS requires [19][20]. As a consequence, a wallet must generate the BBS proof in application software while relying on the secure element only for a lightweight possession test. The standard design signs the verifier's nonce with a device-resident ECDSA key; the wallet transmits that one-time signature together with the BBS proof. Because the nonce is unique per session, the ECDSA artefact cannot be reused as a cross-site identifier, so unlinkability is preserved while the verifier still learns that the proof came from a device holding the certified key [13].

Hardware vendors have not announced pairing support on consumer-grade SEs, so near-term deployments must either keep BBS computations in software or use less powerful schemes without pairings.

4.4.4 Summary

JSON Web Proof and W3C Data-Integrity containers let BBS credentials inhabit the same JWT-centric ecosystem as SD-JWT while closing the privacy gaps analysed earlier. They achieve cryptographic unlinkability, attribute-level disclosure, and no structural leaks at the cost of requiring pairing-friendly crypto libraries and (for now) a software implementation outside secure elements. Section 4.5 quantifies these trade-offs against SD-JWT/ECDSA.

4.5 Comparative Analysis of BBS versus SD-JWT/ECDSA

To better illustrate the differences between the two approaches, the following table summarizes some of the important aspects of design, privacy and applicability for SD-JWT / ECDSA versus JWP credentials signed with BBS signatures.

Aspect	**SD–JWT / ECDSA**	**BBS in JSON (JWP or Data Integrity)**
Selective disclosure mechanism	**Salted-hash commitments:** issuer signs a JWT where each disclosable claim is replaced by a salted hash; holder later releases values and salts; verifier recomputes hashes	**Zero-knowledge proof** derived from a multi-message BBS signature; holder reveals only chosen attributes and proves the rest without exposing them
Unlinkability across presentations	**No** – reusing the same token leaks stable hashes and header metadata	**Yes** – every proof is freshly randomised and contains no persistent identifier
Over-disclosure risk	**High** – all-or-nothing per claim	**Low** – per-attribute; predicate proofs (e.g. range) possible
Structural metadata leak	**Yes** – hashes for each hidden claim expose count/order	**No** – unrevealed attributes are absent from the proof
Typical signature size	64 B (ECDSA P-256)	80 B (BBS on BLS12-381)
Typical presentation size	$\sim$ 64 B base + $\sim$ 48 B $\times$ attribute [1]	272 B base + 32 B $\times$ hidden attribute
Public-key size (compressed)	33 B (SEC 1 compressed form)	96 B (one G_2 element on BLS12-381)
Native secure-element support	**Yes** – widely available on iOS/Android	**No** – pairings not in SE; hybrid binding via ECDSA is possible

Table 4.2. Key differences between SD–JWT/ECDSA and BBS credentials in JSON containers.

Note: More detailed performance evaluations are beyond the scope of this chapter and will be presented in the following chapter. The comparative remarks above focus on structural and qualitative differences; Chapter 5 will provide quantitative metrics to complement this analysis.

4.6 Conclusion and Outlook

The comparison between SD-JWT/ECDSA and BBS does not cconclude in a clear "winner". Instead, it highlights a trade-off between short-term feasibility and long-term privacy goals. SD-JWT enables some privacy gains while using only established and implemented cryptography. This minimal-change path explains its adoption in the first national and European wallet pilots.

BBS signatures demonstrate what becomes possible when the format itself is designed around zero-knowledge proofs. They remove the threat of correlation-attacks and enable holders to withhold any attribute - potentially even proofing any predicate over an attribute. However, the practical use of BBS requires currently non-existing hardware support and more established standardization and audits.

In the near term, hybrid implementations are thinkable where hardware-binding is achieved through supported cryptography and the identification-operations are executed with more advanced cryptography like BBS. The maturation of supporting infrastructure and standards may thus lead to a progression from SD-JWT implementations to the use of zero-knowledge based schemes like BBS.

[1] See Section 5.4.6 for a more detailed size discussion.

Chapter 5

A Practical Implementation of BBS Signatures

The preceding chapters established a theoretical foundation of BBS signatures and highlighted their advantages over traditional approaches in electronic identity systems. While these advantages are compelling in theory, the practical implementation depends critically on the computational feasibility—in particular on resource-constrained devices such as smartphones that are the primary platform for digital identity applications.

This chapter presents a practical implementation of the BBS signature scheme. The implementation was designed to evalute the real-world viability for electronic identity systems and serves as a proof-of-concept. We assess its computational performance, scalability characteristics and operational feasibility within the constraints of mobile devices.

5.1 Scope and Limitations

As stated above, the primary objective of this implementation is to evaluate the practical feasibility of BBS signatures in privacy-preserving digital identity systems. To that end, the project operates within several important limitations that define its scope:

- **Specification Compliance:** Our implementation is based on the IETF BBS Signatures draft specification [7] but does not claim full compliance with all specification details. Certain simplifications have been made for clarity and educational value, and the implementation would require additional hardening for production deployment.

- **Security:** In our understanding, the implementation is cryptographically sound. However, it may not perfectly implement constant-time operations or resist side-channel attacks which would be essential for production systems but beyond the scope of this feasibility study.

- **Limited Platform Testing:** The given performance evaluation focuses on desktop benchmarking with mobile projections based on published hardware benchmarks. Direct mobile device testing lies outside the current scope but represents an important direction for future work.

5.2 Technology Stack and Architecture

The technology stack for our implementation was selected with three pragmatic considerations in mind: (i) availability of mature, open-source components for pairing-based cryptography, (ii) adequate performance for our target use cases, and (iii) existing expertise with the chosen tools. Table 5.1 summarizes the resulting stack.

Component	Technology
Programming Language	C++17 (https://en.cppreference.com/w/cpp/17)
Cryptographic Library	MIRACL Core (https://github.com/miracl/core)
Elliptic Curve	BLS12-381
Build System	CMaake 3.10+
Compiler	GCC 12.2.0
Operating System	Linux (6.1.0-32-amd64)
Development Environment	Unix-like system
Random Number Generation	MIRACL Core RNG
Pairing Implementation	Optimal Ate pairing (MIRACL Core)
Hash Functions	SHA-256 (via MIRACL Core)
Testing Framework	Custom test suite
Benchmarking	Custom performance measurement

Table 5.1. Complete Technology Stack for BBS+ Implementation

C++17. C++17 offers a balance between low-level control and modern language-features such as smart pointers, lambda expressions and full object-oriented programming support.

MIRACL Core. MIRACL Core satisfies the requirements of BBS signatures by providing:

- built-in support for the BLS12-381 curve and the optimal-ate pairing needed by the IETF draft;
- a compact, header-only codebase under the permissive Apache 2.0 licence;
- constant-time scalar multiplication and field arithmetic operations;
- support for both x86-64 and ARMv8 architectures;
- active maintenance.

The remaining technology choices followed naturally from the selection of programming-language and cryptographic library.

5.2.1 Project Structure

The project structure adheres to common C++ conventions and keeps third-party code separated from the thesis-related source files. The full implementation is hosted in a private Git repository at the University of Bern (https://gitlab.inf.unibe.ch/crypto-students/2025.bsc.lukas.leuba/-/tree/73e5a41f7035e0206e7bd3ac44850b369283a66c/src). Access is currently restricted to the crypto research group and the author.

- **include/** - public headers
  - credential.h
  - credential_system.h
  - bbs_plus.h
  - benchmark.h
  - debug_utils.h
- **src/** — implementation files
  - bbs_plus.cpp
  - benchmark.cpp
  - debug_utils.cpp
- **test/** — unit-test drivers
- **examples/** — minimal end-to-end demo
- **benchmark/** — standalone micro-benchmark harness
- **lib/** — MIRACL Core build artefacts

5.2.2 Software Architecture

The implementation relies on a relatively small set of classes and clear responsibilities:

Credential layer. BBSPlusCredential holds the attribute list and signature tuple (A, e). BBSPlusProof stores selective-disclosure proofs and the indices of disclosed attributes. Both classes conform to the abstract interfaces in credential.h, potentially allowing for alternative credential types to be plugged in later.

System layer. BBSPlusCredentialSystem manages key generation, credential issuance, verification, proof creation, and proof verification. It tracks the last OperationMetrics object (timings, sizes, disclosure count) and a full history for later analysis.

Support layer. benchmark.cpp wraps each public operation in high-resolution timers and aggregates results.debug_utils.* offers optional verbose logging controlled by a compile-time flag. These helpers are compiled only for test and benchmark targets, so no overhead leaks into release builds.

Below we give a simplified UML class diagram in Figure 5.1 in order to visualize the credential and system layers; the support layer is omitted for clarity.

5.2.3 Security Level

The cryptographic strength of the implemented scheme is determined almost entirely by the pairing-friendly curve BLS12-381 used in all group operations. BLS12-381 was initially intended to offer approximately 128 bits of security, which would be equivalent to the security of the widely used EdDSA signature algorithm executed on the ed25519 curve. However, recent analyses place its discrete-log hardness at $\approx$117–120 bits [1].

In practical terms this shortcoming is not consequential: a 117-bit attack cost is still far beyond the capabilities of current or foreseeable hardware [22].

[1] See this article by Michael Scott, current Chief Cryptographer at MIRACL [21, §2], the curve choice discussion in the BBS draft [7, §6.6], and NCC Group's Zcash review [22, §4, "Curve BLS12-381 Security"]

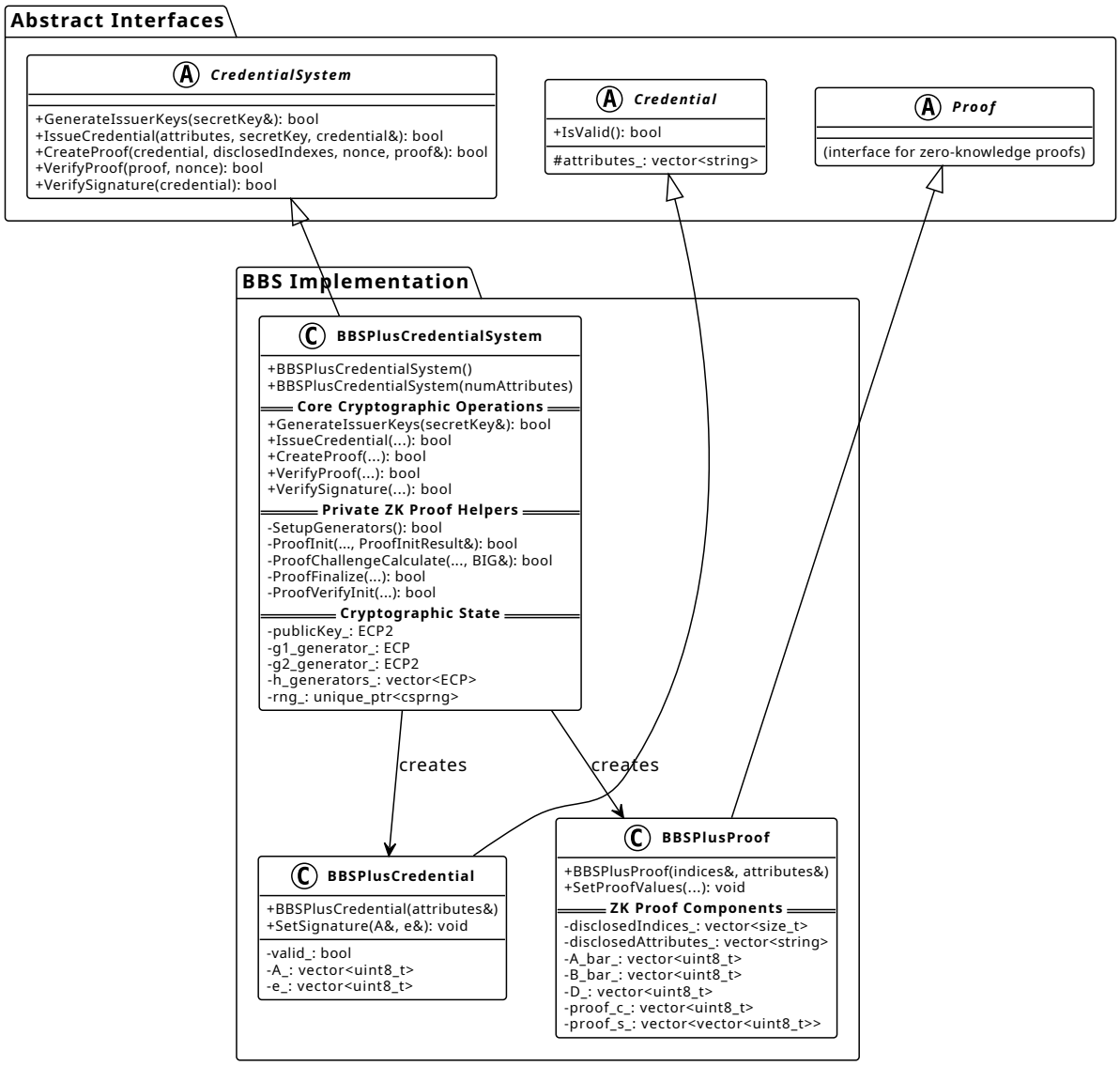


Figure 5.1. UML Class Diagram: `BBSPlusCredential` holds the attribute list and signature tuple (A, e). `BBSPlusProof` stores selective-disclosure proofs and the indices of disclosed attributes. Both classes conform to the abstract interfaces in `credential.h`, potentially allowing for alternative credential types to be plugged in later. `BBSPlusCredentialSystem` manages the core operations of key generation, credential issuance, verification, proof creation, and proof verification. It tracks the last `OperationMetrics` object (timings, sizes, disclosure count) and a full history for later analysis.

5.3 Benchmark Methodology

Having laid out the technical background of the implementation, we now turn our attention to its practical evaluation. We conducted a series of performance benchmarks under controlled conditions. The methodology outlined in this section details the used test environment and the chosen performance metrics.

5.3.1 Test Environment

Our performance evaluation was conducted on a modern desktop system. The key specifications of the test platform are as follows:

- **Processor:** 13th Generation Intel Core i5-1340P (16 cores, x86_64 architecture)
- **Memory:** 32 GB system RAM
- **Operating System:** Linux 6.1.0-37-amd64
- **Compiler:** GCC 12.2.0

This desktop environment provides a stable and reproducible baseline for performance measurement. However, in a practical scenario the main deployment targets would be generally less powerful mobile devices. To translate the results to mobile platforms, we apply performance scaling factors based on published benchmarks.

5.3.2 Metrics Collected

For our analysis we decided to track three main metrics:

Operation Duration: We measure the wall-clock time required for each BBS operation using high-resolution timing. Each measurement represents the complete operation including all necessary cryptographic computations, memory allocations, and data serialization. Timing measurements exclude system initialization and random number generator seeding.

Data Size: We record the serialized size of cryptographic objects such as credentials, proofs, and key material. These measurements inform storage and bandwidth requirements. All sizes represent the binary encoding used in our implementation, which differs slightly from standardized formats as we describe in section 5.4.4.

Scalability: For each operation, we track the number of total attributes and disclosed attributes to analyze scaling behavior. This allows for performance prediction for credentials of arbitrary size and disclosure patterns.

Each given time-measurement is the mean of multiple repetitions in order to exclude one-time system-anomalies.

5.4 Benchmark Results

This section presents the performance evaluation results of the BBS scheme implementation across multiple dimensions: basic operation performance, attribute count scalability, and selective disclosure characteristics. All measurements were conducted on the test platform described in Section 5.3.1 with each data point representing the mean of multiple iterations.

Operation	Duration (ms)	Output Size (bytes)
Key Generation	3.07	97 (public), 48 (private)
Credential Issuance	2.47	97 (credential)
Proof Creation	5.91	579 (proof)
Proof Verification	6.31	-

Table 5.2. Basic BBS operation performance (10 attributes, 5 disclosed). Output sizes refer to the serialized size of each operation's main result: the public and private keys for key generation, the credential for issuance, and the selective disclosure proof for proof creation.

5.4.1 Basic Operation Performance

Table 5.2 presents the fundamental performance characteristics of our BBS implementation for a representative scenario involving 10 attributes with 5 disclosed (50% selective disclosure rate). These measurements are averages of 5 repetitions.

These results demonstrate that all BBS operations complete within single-digit milliseconds on desktop hardware. The total time for a complete proof-based authentication cycle (proof creation + verification) is approximately 12.22ms, which falls well within acceptable user experience thresholds for interactive applications [23] [24].

5.4.2 Attribute Count Scalability

Figure 5.2 shows the performance characteristics of all BBS operations as the number of attributes increases from 1 to 100. For this evaluation, we used a 50% selective disclosure rate (half of the attributes disclosed) to represent an a privacy-preserving scenario. Each datapoint represents an average of 5 repetitions.

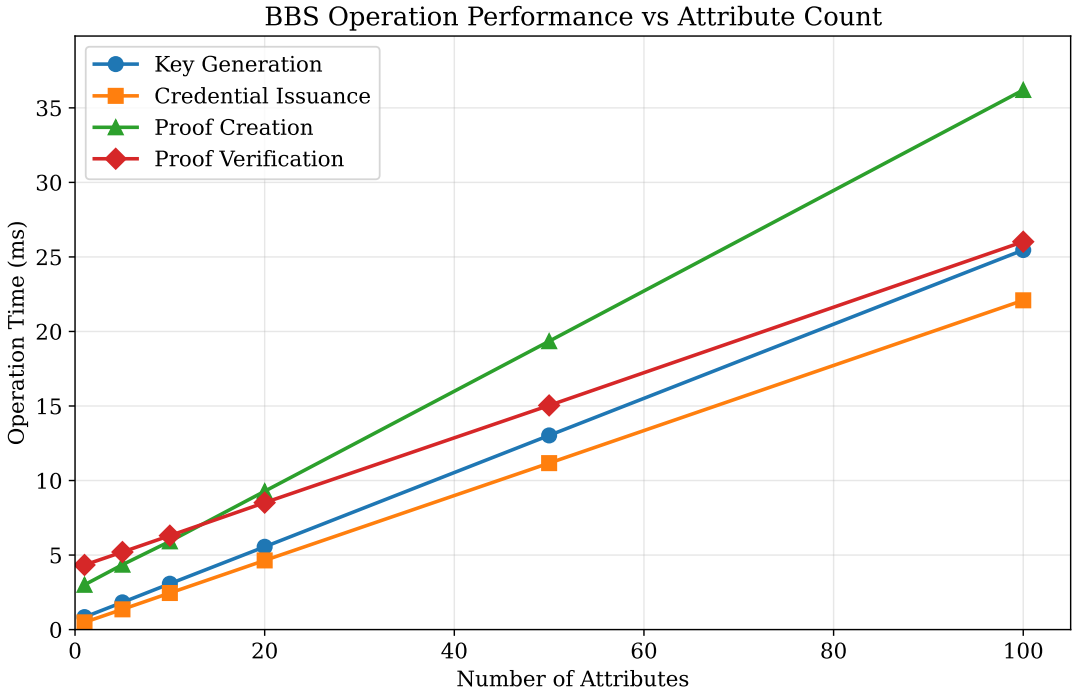


Figure 5.2. BBS Operation Performance vs Attribute Count

The results reveal clear linear scaling behavior for all operations:

- **Key Generation**: Scales linearly from 0.84ms (1 attribute) to 25.5ms (100 attributes)

- **Credential Issuance**: Grows from 0.49ms (1 attribute) to 22.1ms (100 attributes)
- **Proof Creation**: Increases from 3.0ms (1 attribute) to 36.2ms (100 attributes)
- **Proof Verification**: Ranges from 4.3ms (1 attribute) to 26.0ms (100 attributes)

Proof creation consistently requires the highest computational effort, which is expected given the complex zero-knowledge proof generation process. However, even for large credentials with 100 attributes, all operations complete within 40ms, demonstrating decent scalability for practical identity applications.

Figure 5.3 shows the corresponding growth in proof size as the number of attributes increases.

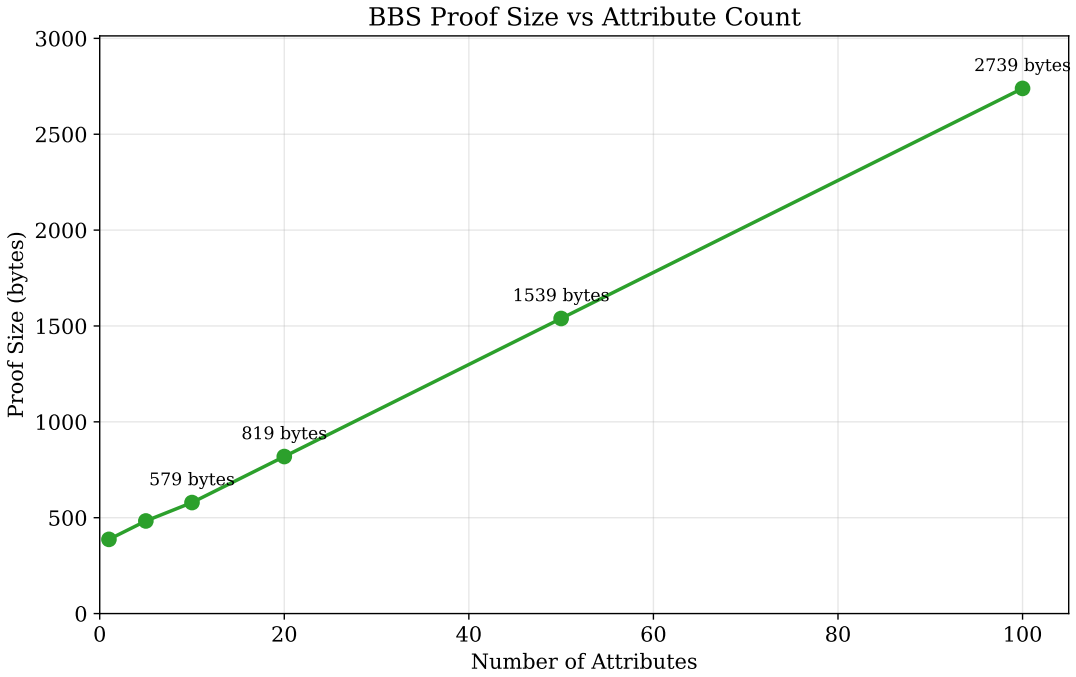


Figure 5.3. BBS Proof Size vs Attribute Count

Proof size exhibits linear scaling, growing from 387 bytes for a single attribute to 2,739 bytes for 100 attributes. This represents an average of approximately 24 bytes per additional attribute.

A typical physical ID card contains around ten attributes, which means that only a subset of the results shown above are directly relevant in current real-world scenarios. However, in a digital E-ID context, it is plausible that credentials will contain a larger number of attributes. Unlike in physical documents, adding additional fields in digital credentials has virtually no cost in terms of space or logistics. As a result, the number of attributes may grow over time to support more complex or multi-purpose use-cases such as health-insurance status, vaccination history, student status etc. In such scenarios, the scaling behavior of the underlying cryptographic schemes becomes increasingly important.

5.4.3 Selective Disclosure Performance

To evaluate the impact of selective disclosure on performance, we measured both operation duration and proof size across different disclosure percentages using a fixed credential size of 20 attributes. Figure 5.4 presents the relationship between disclosure percentage and computational performance.

The results show an expected asymmetry between proof creation and verification:

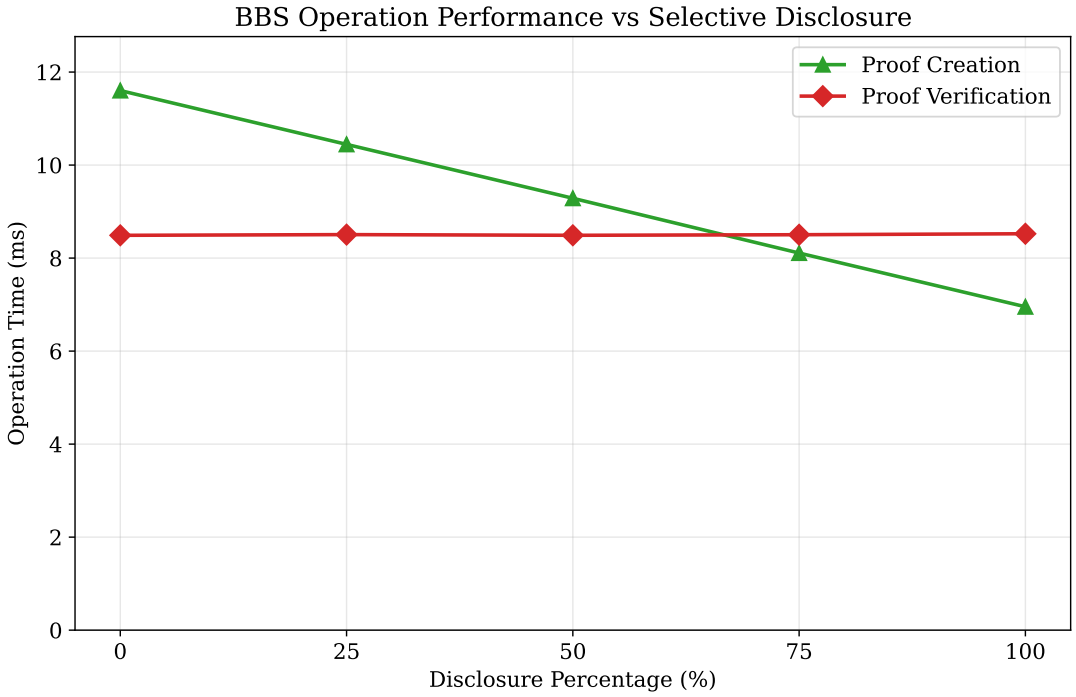


Figure 5.4. BBS Operation Performance vs Selective Disclosure

- **Proof Creation**: Shows a clear inverse relationship with disclosure percentage, decreasing from 11.6ms (0% disclosure) to 7.0ms (100% disclosure). This behavior reflects the reduced zero-knowledge computation effort required when more attributes are revealed (and thus less undisclosed attributes are part of the proof).
- **Proof Verification**: Remains relatively constant at approximately 8.5ms regardless of disclosure percentage, confirming that verification complexity is primarily determined by the overall proof structure rather than the disclosure rate.

Figure 5.5 demonstrates the corresponding impact on proof size.

Proof size shows a strong negative correlation with disclosure percentage, decreasing from 1,299 bytes (0% disclosure) to 339 bytes (100% disclosure). This substantial variation—nearly a 4x difference—reflects the fundamental trade-off between privacy and efficiency in zero-knowledge proof systems. Hidden attributes require cryptographic commitments and zero-knowledge proofs, while disclosed attributes can be transmitted as plaintext values.

5.4.4 Size Expectations and Encoding Caveats

For this variant of the BBS signature scheme according to the IETF draft 08, serialized objects have fixed sizes that follow directly from the group and scalar encodings: a signature is the concatenation of one compressed G_1 element (48 B) and one Fr scalar (32 B), which results in 80 bytes in total. A zero-knowledge proof with no hidden attributes serialises three G_1 points and four scalars, i.e. $3{\times}48 + 4{\times}32 = 272$ bytes, and each additional undisclosed attribute adds exactly one scalar (+32 B).

The prototype presented here reports 97 B for signatures and 339 B + 48 B per hidden attribute for proofs. This deviation is caused by two features of MIRACL core:

1. **Group points.** MIRACL's `ECP_toOctet(..., true)` prepends a 1-byte format tag to the 48-byte compressed x-coordinate, producing 49-byte elements.
2. **Scalars.** `BIG_toBytes(...)` always emits the full limb width of the field (48 B for BLS12-381), whereas the subgroup order fits in 32 bytes.

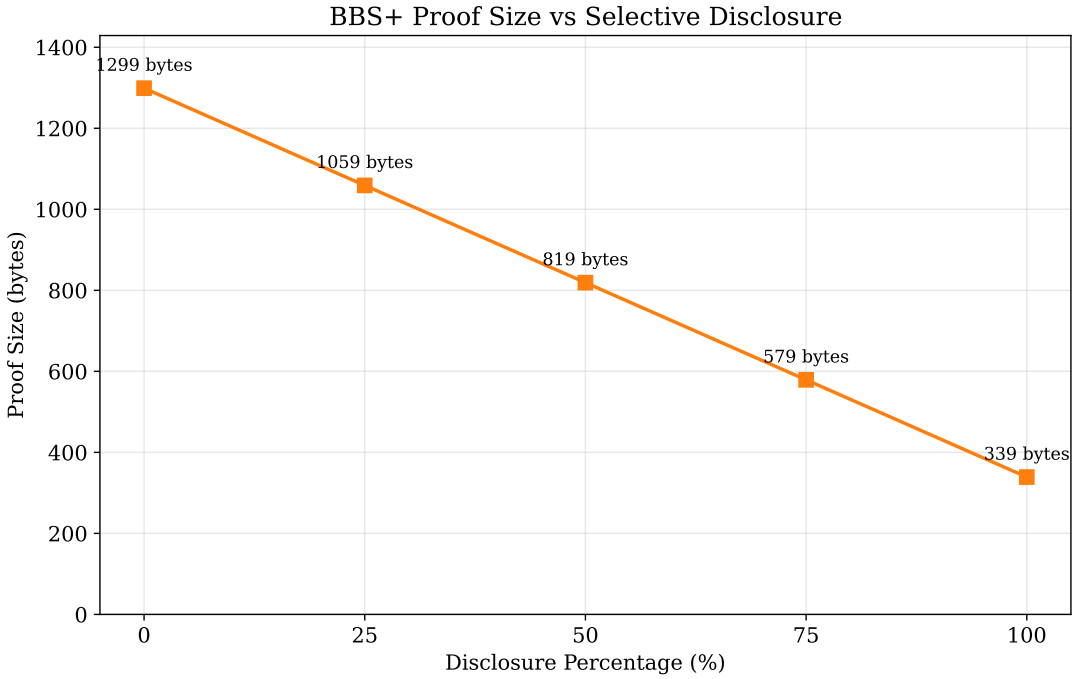


Figure 5.5. BBS Proof Size vs Selective Disclosure

After mitigating these two encoding artefacts, the implementation would produce the expected 80 B signatures and $272 + 32u$ B proofs, where u is the number of hidden attributes, matching the size figures used in Table 4.2. However, we decided not to implement these mitigations since the lightly inflated object-sizes do not affect the scaling behavior and feasablility analysis in a significant way.

5.4.5 Mobile Performance Projections

To estimate how the BBS implementation would perform on smartphone hardware, we project performance based on general-purpose CPU benchmarks comparing the Intel Core i5-1340P—used in our test system—to a representative mobile processor, the Qualcomm Snapdragon 8 Gen 2. This comparison is based on the cross-platform benchmark data from the Geekbench suite, which offers standardized single-core and multi-core performance scores for both platforms [25] [26].

According to publicly available Geekbench 6 results, the Intel Core i5-1340P achieves approximately 2.1× higher multi-core scores than the Snapdragon 8 Gen 2, and about 1.2× higher single-core scores [25]. We adopt a conservative scaling factor of 2× to project performance on mobile hardware. This projection reflects a balanced view between single-threaded and multi-threaded execution characteristics.

The resulting performance estimates for a modern mobile device are summarized below, using the same benchmarked scenario of a 10-attribute credential with 50% selective disclosure:

- **Key Generation:** ≈ 6.1 ms
- **Credential Issuance:** ≈ 4.9 ms
- **Proof Creation:** ≈ 11.8 ms
- **Proof Verification:** ≈ 12.62 ms
- **Round-Trip** (proof creation + verification) ≈ 24.44 ms

These projected mobile performance characteristics remain well within acceptable bounds for interactive user experiences, supporting the practical viability of BBS signatures for smartphone-based digital identity applications [23] [24].

To additionally support our mobile performance projections, we also conducted the same benchmark suite on a Raspberry Pi 4 Model B, which features a quad-core ARM Cortex-A72 CPU. The specification of the Raspberry Pi 4 is highly comparable to a low-end smartphone. The results show that a 10-attribute credential with 50% selective disclosure requires approximately 15.0 ms for key generation, 11.9 ms for credential issuance, 28.3 ms for proof creation, and 30.8 ms for proof verification. These figures align well with our mobile projections and reinforce the conclusion that BBS signatures remain computationally feasible even on low-power ARM devices.

5.4.6 Performance Comparison with SD-JWT-ECDSA

This section compares our BBS benchmark results against the established SD-JWT-ECDSA approach in terms of cryptographic performance and data size overhead.

Cryptographic Performance. ECDSA operations are highly efficient compared to BBS: a 2021 paper by Martin Koppl et al. found that all ECDSA-P256 operations perform consistently under the 2 ms mark (Intel i7-7700, 3.0-3.8 GHz CPU, Windows 10, python) [27]. On even more modern hardware, these runtimes would likely fall well below 1 ms. In addition to key generation, signing, and verification, SD-JWT with ECDSA involves hashing of credential claims, which has negligible runtime even on constrained devices. In contrast, BBS proof generation and verification involves pairing-based zero-knowledge proofs. Our measurements show 10–20 ms per proof operation on desktop and 2–3 times longer on projected mobile hardware (see Section 5.4.5). This computational overhead represents the main trade-off for achieving cryptographic unlinkability.

Transmission Size. An SD-JWT includes one 32-byte digest and a 16-byte salt per claim, in addition to a 64-byte ECDSA signature. This yields a typical overhead of $\sim$48 B per claim [11]. For a 20-attribute credential, this results in $\sim$1 kB of signed metadata. In comparison, a BBS signature is constant-size, while the selective disclosure proof grows with the number of hidden attributes. Our implementation yields proofs of 339–1299 B for 0–20 hidden attributes (Section 5.4.3). When many attributes are disclosed, BBS proofs can be more compact than SD-JWT disclosures. When most attributes are hidden, the size is comparable.

Conclusion. From a performance standpoint, the primary disadvantage of BBS signatures compared to ECDSA lies in the cost of cryptographic operations, where ECDSA is roughly an order of magnitude faster. However, in terms of transmission size the schemes are largely comparable. Importantly, despite the relative slowdown, BBS operations still complete within tens of milliseconds which is well within acceptable bounds for modern desktop and mobile hardware. This means that while BBS introduces additional computational overhead, it remains comfortably feasible for deployment in practical E-ID scenarios.

5.5 Summary and Conclusion

This chapter presented an implementation and performance evaluation of the BBS signature scheme as specified in the current IETF draft. Our benchmark results show that all operations complete within tens of milliseconds on modern desktop hardware. Under increased attribute numbers, both the runtime and proof size scale linearly. Even under mobile performance projections, total authentication cycles remain below common interactivity thresholds.

In comparison to SD-JWT with ECDSA, BBS signatures trade off higher cryptographic cost for stronger privacy guarantees. While ECDSA-based solutions achieve lower runtimes, BBS signatures offer unlinkability and flexible selective disclosure with manageable overhead.

These results confirm that BBS signatures are not only theoretically attractive but also practically viable for real-world deployment in privacy-preserving digital identity systems.

Chapter 6

Conclusion

This thesis aimed to explore the BBS signature scheme as a privacy-enhancing alternative to the traditional ECDSA-based digital signatures used with the SD-JWT credential format. After introducing the cryptographic foundations of BBS, we derived a non-interactive zero-knowledge proof of knowledge for q-SDH tuples and embedded this construction into a group signature scheme. We altered this original group-signature scheme in order to better meet the demands of electronic identity systems. This lead to the modern BBS signature scheme. In a detailed comparison, we evaluated BBS signatures against SD-JWT, both in terms of theoretical privacy guarantees and practical performance. We found that BBS inherently provides unlinkability and selective disclosure. These features eliminate the observed strucural fingerprinting and over-disclosure risks that remain in SD-JWT.

These theoretical promises needed to prove feasible in practice. We addressed this feasablility analysis through a practical implementation and accompanying benchmark study. On a modern desktop CPU, full proof generation and verification complete in under 20 ms per presentation, and remain below 60 ms under our applied mobile projections. The runtime of all operations scaled linearly with the number of attributes. Proof size grows also linearly with the number of hidden attributes—from 339 B to about 1.3 kB, which is comparable to or smaller than the SD-JWT metadata overhead for typical credential sizes. This predictable scaling facilitates bandwidth and storage planning. We concluded that BBS provides unlinkability and selective disclosure at a cost that is acceptable for interactive user experiences.

Despite its technical merits, BBS signatures are not currently adopted by leading European E-ID initiatives, including the Swiss E-ID. We perceived two major reasons for this circumstance: (i) the lack of standardization and audit maturity for BBS, and (ii) the absence of native hardware support on current mobile devices. These limitations, however, are not intrinsic. As the cryptographic landscape evolves, and as infrastructure and standards catch up, a gradual shift toward zero-knowledge-based schemes seems both realistic and desirable.

Future work could explore the theoretical and practical integration of predicate proofs to enable more expressive privacy-preserving claims such as proving age ranges without revealing raw values. This would make such BBS-based identity systems adhere even closer to the data-minimization principle.

In conclusion, this thesis demonstrated that BBS signatures provide a viable path to a future where citizens control exactly *what* they reveal, *when* and to *whom*—the foundation of a trustworthy digital public infrastructure.

Appendix: Raspberry Pi 4 Benchmark Figures

This appendix provides the same set of performance plots shown in the main evaluation (Section 5.4), but generated on a Raspberry Pi 4 Model B (ARM Cortex-A72, 1.5 GHz, 4 cores, 8 GB RAM). These figures help illustrate the performance characteristics of BBS signatures on a constrained, ARM-based device.

- **Figure 1**: Runtime scaling of all BBS operations as a function of attribute count (1–100 attributes, 50% disclosure).
- **Figure 2**: Proof size growth with increasing attribute count.
- **Figure 3**: Runtime variation of proof creation and verification as disclosure percentage increases (0–100%, 20 attributes).
- **Figure 4**: Proof size variation as a function of disclosure percentage (0–100%, 20 attributes).

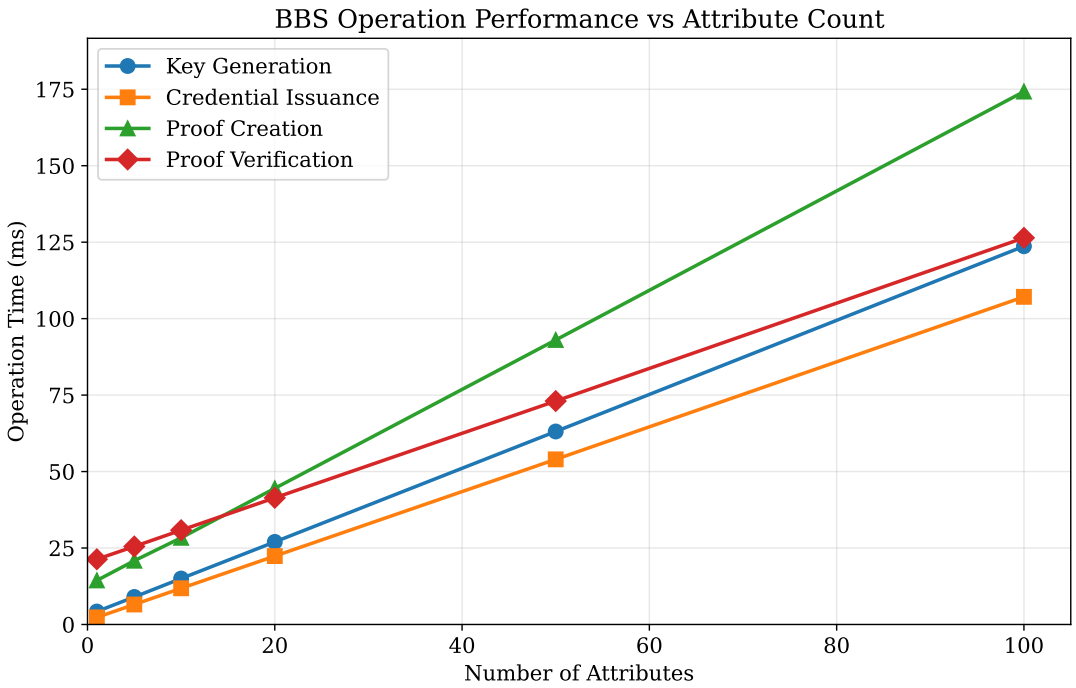


Figure 1. BBS Operation Performance vs Attribute Count (Raspberry Pi 4)

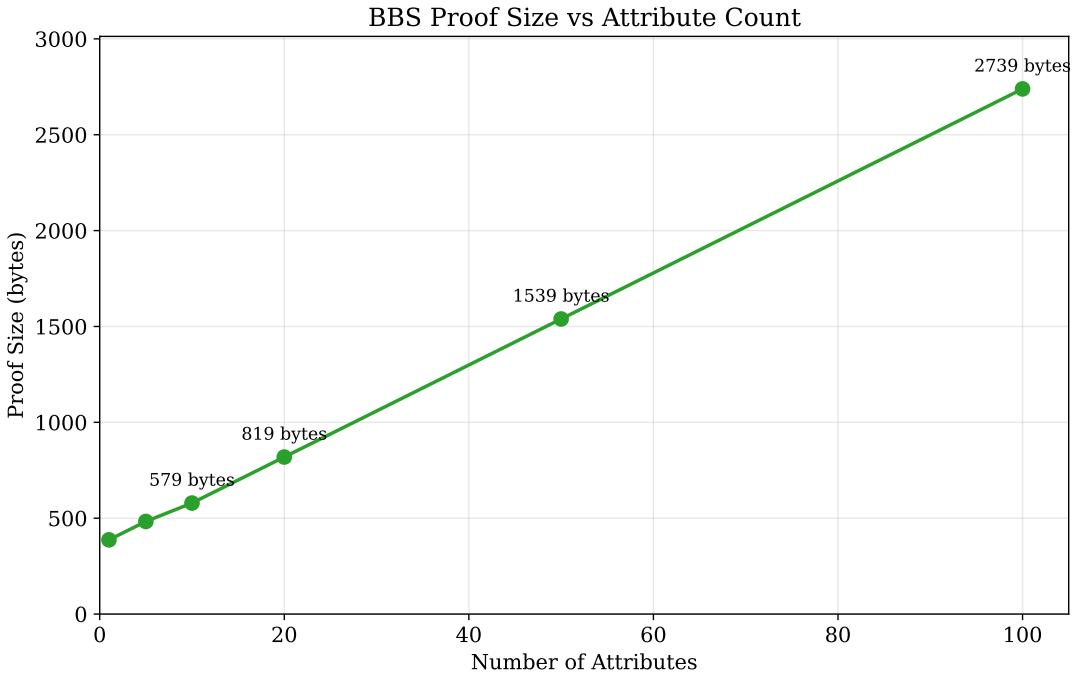


Figure 2. BBS Proof Size vs Attribute Count (Raspberry Pi 4)

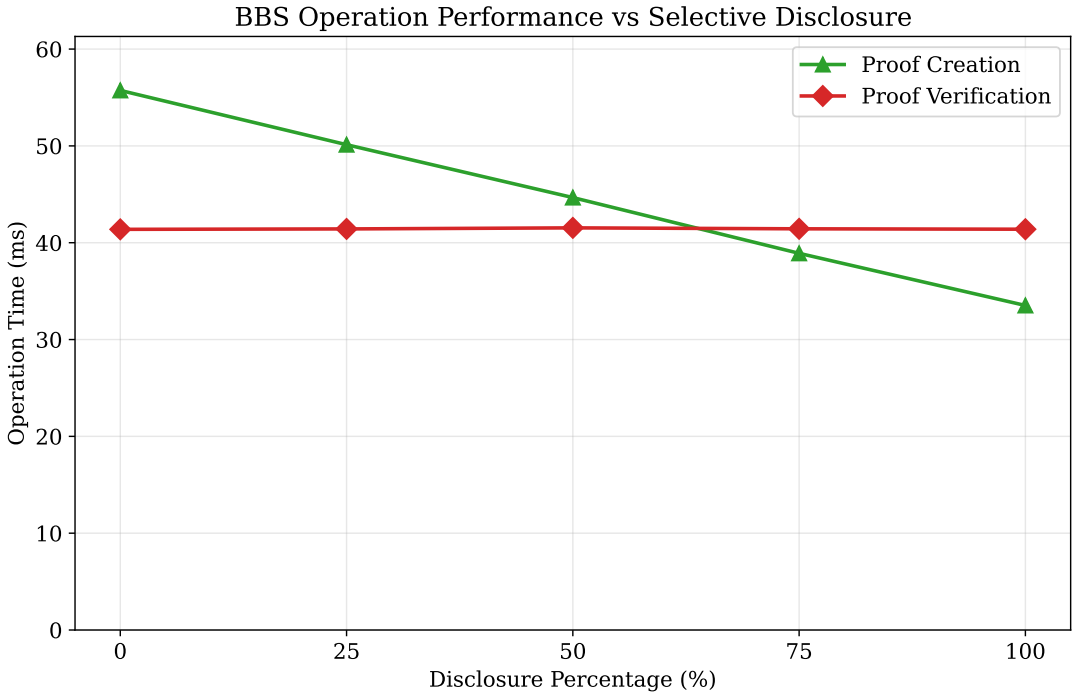


Figure 3. BBS Operation Performance vs Selective Disclosure (Raspberry Pi 4)

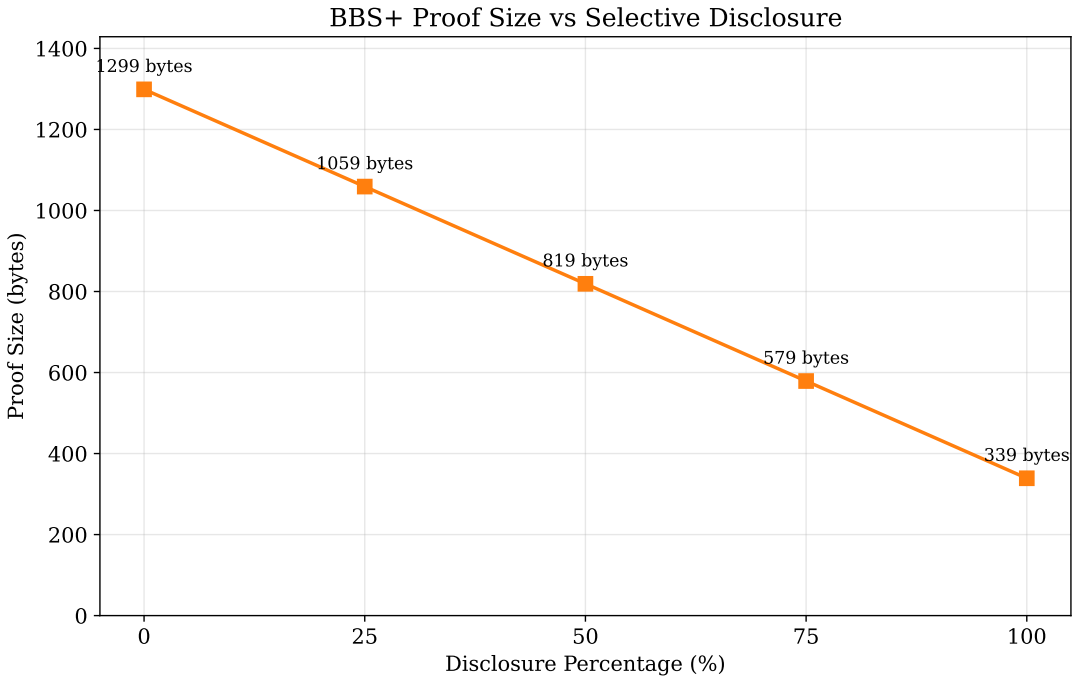


Figure 4. BBS Proof Size vs Selective Disclosure (Raspberry Pi 4)

Bibliography

[1] Dan Boneh and Victor Shoup, *A Graduate Course in Applied Cryptography*, en, 0.6. Unpublished manuscript, 2023. Available: https://toc.cryptobook.us/.

[2] Dan Boneh, Xavier Boyen, and Hovav Shacham, "Short Group Signatures," in *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, M. K. Franklin, Ed., ser. Lecture Notes in Computer Science, vol. 3152, Springer, 2004, pp. 41–55. DOI: 10.1007/978-3-540-28628-8_3. Available: https://doi.org/10.1007/978-3-540-28628-8%5C_3.

[3] Nigel P. Smart, *Cryptography Made Simple* (Information Security and Cryptography), en. Cham: Springer International Publishing, 2016, Also available as eBook: 978-3-319-21936-3, ISBN: 978-3-319-21935-6. DOI: 10.1007/978-3-319-21936-3. Available: http://link.springer.com/10.1007/978-3-319-21936-3.

[4] Man Ho Au, Willy Susilo, and Yi Mu, "Constant-Size Dynamic k-TAA," en, in *Security and Cryptography for Networks*, D. Hutchison, T. Kanade, J. Kittler, *et al.*, Eds., vol. 4116, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 111–125, ISBN: 978-3-540-38080-1. DOI: 10.1007/11832072_8. Available: http://link.springer.com/10.1007/11832072_8.

[5] Stefano Tessaro and Chenzhi Zhu, "Revisiting BBS Signatures," in *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, C. Hazay and M. Stam, Eds., ser. Lecture Notes in Computer Science, vol. 14008, Springer, 2023, pp. 691–721. DOI: 10.1007/978-3-031-30589-4_24. Available: https://doi.org/10.1007/978-3-031-30589-4%5C_24.

[6] Jan Camenisch and Anna Lysyanskaya, "Signature Schemes and Anonymous Credentials from Bilinear Maps," in *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, M. K. Franklin, Ed., ser. Lecture Notes in Computer Science, vol. 3152, Springer, 2004, pp. 56–72. DOI: 10.1007/978-3-540-28628-8_4. Available: https://doi.org/10.1007/978-3-540-28628-8%5C_4.

[7] Tobias Looker, Vasilis Kalos, Anew Whitehead, and Mike Lodder, "The BBS Signature Scheme," Internet Engineering Task Force, Internet Draft draft-irtf-cfrg-bbs-signatures-08, Mar. 2025, Num Pages: 119. Available: https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures-08 (visited on 05/26/2025).

[8] Federal Office of Information Technology, Systems and Telecommunication, *Digital identity e-ID*. Available: https://www.eid.admin.ch/en (visited on 05/26/2025).

[9] Federal Office of Information Technology, Systems and Telecommunication, *Swiyu technical documentation*, en-US. Available: https://swiyu-admin-ch.github.io/ (visited on 05/26/2025).

[10] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," en, RFC Editor, Tech. Rep. RFC7519, May 2015, RFC7519. DOI: 10.17487/RFC7519. Available: https://www.rfc-editor.org/info/rfc7519 (visited on 05/26/2025).

[11] Daniel Fett, Kristina Yasuda, and Brian Campbell, "Selective Disclosure for JWTs (SD-JWT)," Internet Engineering Task Force, Internet Draft draft-ietf-oauth-selective-disclosure-jwt-19, May 2025, Num Pages: 97. Available: https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt (visited on 05/26/2025).

[12] SOG-IS Crypto Working Group, "SOG-IS Crypto Evaluation Scheme: Agreed Cryptographic Mechanisms Version 1.3," SOG-IS, Tech. Rep., Feb. 2023. Available: https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf.

[13] European Commission, *Architecture and Reference Framework (ARF) - EUDI Wallet v1.2.0*, Nov. 2023. Available: https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/1.1.0/arf/ (visited on 05/29/2025).

[14] Alexander Mühle, Andreas Grüner, Tatiana Gayvoronskaya, and Christoph Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2018.10.002. Available: https://www.sciencedirect.com/science/article/pii/S1574013718301217.

[15] Ministry of the Interior and Kingdom Relations, "NL Wallet Design Considerations," Government of the Netherlands, Tech. Rep., May 2023. Available: https://files.dyne.org/eudi/NL-Wallet-Design-v1.0.3.pdf.

[16] David Waite, Michael B. Jones, and Jeremie Miller, "JSON Web Proof," Internet Engineering Task Force, Internet-Draft draft-ietf-jose-json-web-proof-09, Apr. 2025, Work in Progress, 30 pp. Available: https://datatracker.ietf.org/doc/draft-ietf-jose-json-web-proof/09/ (visited on 05/26/2025).

[17] Digital Bazaar and W3C Verifiable Credentials Working Group, "Verifiable credentials data integrity 1.0," W3C, Recommendation, 2023. Available: https://www.w3.org/TR/vc-data-integrity/ (visited on 05/26/2025).

[18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More," en, in *2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA: IEEE, May 2018, pp. 315–334, ISBN: 978-1-5386-4353-2. DOI: 10.1109/SP.2018.00020. Available: https://ieeexplore.ieee.org/document/8418611/.

[19] Apple Inc., "Apple Platform Security," en, Dec. 2024. Available: https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf (visited on 05/30/2025).

[20] Android Developers, *Android Keystore System — StrongBox KeyMint Secure Element*, en, 2025. Available: https://developer.android.com/privacy-and-security/keystore (visited on 05/30/2025).

[21] Michael Scott, *Pairing Implementation Revisited*, Publication info: Preprint. MINOR revision., 2019. Available: https://eprint.iacr.org/2019/077.

[22] Thomas Pornin, Aleks Kircanski, Mason Hemmel, *et al.*, "Zcash Overwinter Consensus and Sapling Cryptography Review," en, NCC Group Security Services, Inc., Tech. Rep., Jan. 2019. Available: https://www.nccgroup.com/media/v1kkxeae/_ncc_group_zcash2018_public_report_2019-01-30_v13.pdf.

[23] Jakob Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[24] Jakob Nielsen, *Response Times: The Three Important Limits*, Published: Nielsen Norman Group article, 1993. Available: https://www.nngroup.com/articles/response-times-3-important-limits/ (visited on 06/01/2025).

[25] Geekbench Browser, *Intel core i5-1340p processor benchmark results*, https://browser.geekbench.com/v6/cpu/baseline/1929095, 2025. (visited on 06/01/2025).

[26] CPU-Monkey, *Qualcomm snapdragon 8 gen 2 vs intel core i5-1340p (geekbench 5)*, https://www.cpu-monkey.com/en/compare_cpu-qualcomm_snapdragon_8_gen_2-vs-intel_core_i5_1340p, 2025. (visited on 06/01/2025).

[27] Martin Koppl, Dmytro Siroshtan, Milos Orgon, *et al.*, "Performance Comparison of ECDH and ECDSA," in *2021 2nd International Conference on Electronics, Communications and Information Technology (CECIT)*, Dec. 2021, pp. 825–829. DOI: 10.1109/CECIT53797.2021.00149. Available: https://ieeexplore.ieee.org/document/9742212/.

Erklärung

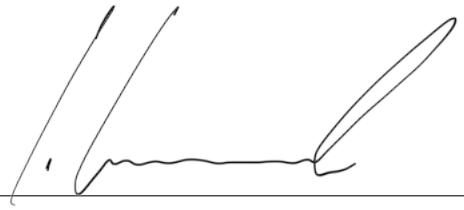
Erklärung gemäss Art. 30 RSL Phil.-nat. 18

Ich erkläre hiert, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Für die Zwecke der Begutachtung und der Überprüfung der Einhaltung der Selbständigkeitserklärung bzw. der Reglemente betreffend Plagiate erteile ich der Universität Bern das Recht, die dazu erforderlichen Personendaten zu bearbeiten und Nutzungshandlungen vorzunehmen, insbesondere die schriftliche Arbeit zu vervielfältigen und dauerhaft in einer Datenbank zu speichern sowie diese zur Überprüfung von Arbeiten Dritter zu verwenden oder hierzu zur Verfügung zu stellen.

Bern, 7. Juli 2025

Ort/Datum



Unterschrift