University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of Privacy-Preserving Mechanisms for Metadata and Public Keys in Blockchain-Based Self-Sovereign Identities

Cedric Egon von Rauscher
Zurich, Switzerland
Student ID: 21-702-105

Supervisor: Daria Schumm, Katharina Müller, Prof. Dr. Burkhard Stiller
Date of Submission: September 1, 2024

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, 

Signature of student

Abstract

Im Zuge der zunehmenden Verbreitung der Blockchain Technologie sind insbesondere im Hinblick auf Metadaten und öffentliche Schlüssel in Self-Sovereign Identity (SSI)-Systemen neue Datenschutzprobleme aufgetreten. Ein zentrales Problem bei blockchain-basierten digitalen Identitätslösungen ist die Unklarheit darüber, ob öffentliche Schlüssel und Metadaten gemäss der Datenschutz-Grundverordnung (DSGVO) als personenbezogene Daten gelten, trotz bereits bestehender Datenschutzrichtlinien. Um diese Lücken zu füllen, untersucht diese Arbeit die Datenschutzimplikationen von Metadaten in SSI-Systemen und schlägt zwei Mechanismen zur Wahrung der Metadatensicherheit vor: Zero-Knowledge-Proofs und Homomorphe Verschlüsselung. Diese Mechanismen wurden als Prototypen im *CredChain*-SSI-System auf der Ethereum-Blockchain implementiert und anschliessend hinsichtlich ihrer Sicherheit, Ressourceneffizienz, Skalierbarkeit und DSGVO-Konformität umfassend evaluiert. Die Ergebnisse zeigen, dass beide Lösungen die Sicherheit von Metadaten verbessern und für den täglichen Einsatz geeignet sind. Allerdings erweist sich der Prototyp mit der homomorphen Verschlüsselung als ressourcenintensiv, und die Speicherung und Übertragung von Daten auf der Blockchain stellt eine monetäre Herausforderung dar.

In the context of rising blockchain technology, new issues with data privacy have surfaced, especially with regard to metadata and public keys in Self-Sovereign Identity (SSI) systems. A problem in blockchain-based digital identity solutions is the ambiguity whether public keys and metadata are considered personal data under the General Data Protection Regulation (GDPR), in spite of established data protection guidelines. In order to fill these gaps, this thesis examines the privacy implications of metadata in SSI systems and proposes two mechanisms for maintaining metadata privacy: Zero-Knowledge Proofs (ZKP) and Homomorphic Encryption (HE). The methodology involves implementing these prototypes within the *CredChain* SSI system on the Ethereum blockchain, followed by a comprehensive evaluation of their security, resource efficiency, scalability, and GDPR compliance. The results show that while both solutions enhance metadata privacy and are feasible for daily use, the HE prototype is resource-intensive, and the current economic feasibility of storing and transmitting data on the blockchain remains a challenge.

Acknowledgments

I would like to begin by expressing my gratitude to my supervisor, Daria Schumm, for her persistent support and constructive feedback throughout the thesis and providing me with her repository *CredChain* for my implementation. Furthermore, I am thankful for my co-supervisor Katharina Müller for final recommendations on the thesis.

I would like to express my gratitude to Prof. Dr. Burkhard Stiller for providing me the privilege of writing my thesis at the Communication Systems Group (CSG) of the University of Zurich.

At last, I would like to thank my family and friends for their support and encouragement, not only in regards to this thesis, but also throughout my entire studies.

Contents

Chapter 1

Introduction

With the emergence of blockchain technology, new data privacy challenges were introduced. Data on the blockchain, as well as metadata generated by the blockchain can be seen by anybody who has an internet connection. This leads to the question whether the data can identify an individual and therefore be considered as personal data, as defined by the General Data Protection Regulation (GDPR). Since public blockchains are immutable and transparent, all data transmitted and stored on them can be collected and potentially linked together. If such linking is successful, the data on the blockchain could be classified as personal data, which, under the GDPR, must be deletable.

1.1 Motivation

Although data protection guidelines exist, the classification of metadata as private or personal data is still up for debate in the real world of data protection and privacy concerns. The ambiguity surrounding the status of metadata with regard to data privacy has left a gap in blockchain-based digital identity solutions, particularly in Self-Sovereign Identities (SSI). As of right now, there is no system in place to deal with removing data from blockchain-based SSI systems [10]. Although data transparency and immutability are guaranteed by the blockchain itself, it is unclear if metadata should be treated with the same level of privacy.

Furthermore, the classification of public keys and whether they should be regarded as personal data are topics of discussion [10]. Public keys are fundamental components of blockchain-based SSIs, working as cryptographic identifiers for the identity holders. However, the lack of a definitive opinion on whether public keys qualify as personal data introduces another layer of complexity to the data privacy discussion. More research is therefore required to fully understand the privacy implications of public keys and metadata. It is essential to identify any potential risks and determine the extent to which personal data may be unintentionally exposed as a result of those components.

1.2 Thesis Goals

This thesis investigates the privacy considerations of metadata and public keys in the context of blockchain-based SSIs. In addition, the tightly coupled question of the deletion of metadata from a ledger is discussed and an extensive analysis of attacks on metadata in SSIs is conducted. In order to contribute to the blockchain and privacy evolution, the thesis proposes two privacy-preserving mechanisms to improve metadata privacy in SSI systems: a Zero-Knowledge Proof (ZKP) and a Homomorphic Encryption (HE) solution, both of which allow the verifying party to verify the metadata without gaining any knowledge about it. Both prototypes are implemented into *CredChain*, a SSI developed for the Ethereum blockchain. The evaluation then compares the implementations in terms of resource usage, gas cost and assesses them for GDPR-compliance and everyday-usability.

1.3 Thesis Outline

The thesis is structured as follows: Chapter 2 introduces the fundamental concepts of blockchain technology and SSI systems which are essential to comprehending the topics of the thesis. Chapter 3 provides an overview and comparison of related work. Chapter 4 covers the design and explanation of the two metadata privacy-preserving prototypes and lists its requirements. Chapter 5 describes the used technologies and the details of the implementation. Chapter 6 is a comprehensive evaluation of the data privacy, computational resource utilization and requirements. Finally, Chapter 7 discusses the findings and summarizes the thesis.

Chapter 2

Background

This chapter provides the theoretical foundation for this work. This section is intended to familiarize the reader with blockchains and public keys, SSIs and metadata thereof, the GDPR and its data protection principles, as well as two privacy-preserving mechanisms, ZKP and HE.

2.1 Blockchain

A blockchain is a system of distributed ledgers that store data in blocks. A block is the storage unit of the blockchain where a group of transactions are recorded and persisted. One-way hash functions cryptographically encode blocks to ensure the integrity and immutability of the historical record, thereby protecting the security of the blockchain. Each block is linked to its predecessor, creating a seamless and secure chain of data custody [2].

In a blockchain, every block typically holds two kinds of information: transactional data and metadata. Transactional data usually includes, among others, the sender and recipient's addresses, the transferred amount, and a digital signature for authentication. Metadata typically includes, among others, the hash of the previous block, the block creation timestamp, and a Merkle root [61]. Other transactional data and metadata attributes vary between the different blockchains.

The inherent nature of blockchains ensures that any changes to data are immediately detectable by other participants in the network. If a block is altered, the hash of that block would no longer match, causing a mismatch in the hash of the subsequent block as well, thus signaling the tampering [39]. This characteristic, known as "immutability", makes public blockchains highly resistant to tampering and fosters trust in the system by ensuring that data can not be altered without detection.

2.1.1 Public Keys

Every user in a blockchain network possesses a unique public and private key. In a blockchain, public and private keys are usually generated using the Elliptic Curve Digital Signature Algorithm (ECDSA), specifically over the secp256k1 curve $y^2 = x^3 + 7$ [11]. When a user creates an address for the first time, his private key is generated which is a cryptographically secured random 256-bit integer. The public key is created by multiplying the generator point on the secp256k1 curve with the 256-bit integer, the private key. To obtain the wallet address, the last 20 bytes of the Keccak-256 hash of the public key is taken. A transaction sender uses his private key to sign a transaction using a digital signature technique. Other users can then confirm the transaction by using the sender's public key [2].

In order to create multiple addresses from one wallet, hierarchical deterministic (HD) wallets are used. Typically, a mnemonic phrase of twelve or 24 words is created as the seed phrase. The seed phrase is used to generate the master private key and a master chain code using the HMAC-SHA512 hash function. Parent keys can extend child keys, which can extend grandchild keys, and so on. This enables the presentation of an HD wallet as having a tree-like structure [17]. To determine new public keys, a Schnorr signature [48] is used that serves as the key for the child node. The public key of the child node is then derived from the parent public key and the signature without revealing the private key of the parent. This approach prevents privilege escalation attacks where once a derived child private key coupled with a parent extended public key at any level of the tree is leaked to the attacker, it will result in the leakage of all the child private keys generated in that HD wallet [17].

2.1.2 Gas Fees

On the Ethereum blockchain, gas fees have to be paid for each transaction to be executed. Higher gas fees incentivise validators to prioritize a transaction over others, due to the higher reward the get upon successfully verifying and adding it to the blockchain. The fees compensate the validators for their computational work and ensure the security and efficiency of the Ethereum network. The base amount of gas units per transaction is 21000 Gwei. 1 Gwei is equal to 0.000000001 Ether (ETH) or 10^{-9} ETH. The base amount is multiplied with the addition of the base network fee and a priority fee. The base fee is calculated by comparing the size of the previously validated block with the target size. The target gas usage per block is set to 50% of the block gas limit, meaning if the gas limit is 30 million gas, then the target gas usage is 15 million. The protocol automatically increases the base fee if the gas in the previous block exceeded the target gas usage and decreases it if less gas was used than the target gas. To have a transaction processed more quickly, a priority fee similar to a "tip" can be added. This incentivises validators to validate these transactions first [20].

When sending data via the Ethereum blockchain, additional gas is required based on the data size. Each byte of non-zero data costs 16 gas whereas each byte of zero data costs 4 gas. To store data on the blockchain, 20000 gas is required to "activate" a 32-Byte

slot. Now, another 20000 gas will be used to update the slot from zero to non-zero data. Changing the data on the slot costs 2900 gas. When clearing a slot, therefore setting the data back to zero, a refund of 4800 gas is given. The cost to read a 32-Byte slot is 2100 gas [57].

2.2 Self-Sovereign Identities

The concept of a SSI is a new decentralized identity management idea that allows entities such as individuals, organizations, and things to have complete control over their digital identity without relying on outside authorities [58]. The elimination of a third-party allows entities to digitally identify and authenticate themselves and rules out the risk of a single point of failure while improving trust, privacy, security, and various other qualities including transparency, persistence, interoperability and minimalization [54]. Cucko et al. defined 20 fundamental guiding principles of an SSI where security, verifiability and authenticity, privacy and minimal disclosure, and ownership and control belong to the most important attributes to be fulfilled [12]. Unlike other systems that depend on a specific identity provider, SSIs are an immutable, secure and independent concept which are separate from centralized services that have the ability to restrict, modify, or delete the identity information [56].

2.2.1 Roles within SSIs

Identity management involves keeping specific attribute data separate and regulating who can access it, which is a necessity in the modern digital landscape. Identity management involves three important roles: identity holder, identity issuer, and credential verifiers.

A person, organization or thing that records a distinguishing attribute linked to particular information within a specific system is defined as an identity holder or a *Subject*. The identity holder stores and manages his credentials in a wallet, having full control of his personal information.

A credential is a provable statement about an aspect of someone's identity information (such as their gender) and evidence (such as birth certificates) connected to that person. The statement is approved and digitally signed by a credential issuer [30]. The holder can decide which attributes he wants to provide as proof to service providers, which is a feature called selective disclosure of credentials.

A credential verifier (also referred to as service provider) is an entity that asks for a particular credential from a trusted issuer and confirms its authenticity through the issuer's signature. When opening a new bank account for example, instead of the bank carrying out the KYC (Know-Your-Customer) process, it could ask the client to send the necessary credentials (i.e. full name, address, birthday etc.), which are then verified by a trusted identity issuer.

An identity issuer is a trusted identity provider that acts as a third party and offers authentication and authorization of credentials to external service providers upon request. It is common for service providers to also serve as identity issuers, since their own identity management systems and databases can be used to authenticate and onboard new users [56]. An example for a identity provider could be a university that issues degrees to students that have completed their studies there. When asked by a service provider for verification of a credential issued by the university itself, they also have the means to verify the authenticity of it.

2.2.2 Digital Identifiers and Verifiable Credentials

A Decentralized Identifier (DID) is a newly developed form of identification that allows for a trustworthy, autonomous digital identity of a user. DIDs are similar to Uniform Resource Identifiers (URIs). They are exclusive references that point to DID documents with the necessary public keys and associated metadata. In contrast to conventional identifiers that are issued and supervised by centralized entities, DIDs are completely managed by the owner of the identity [12]. The owner of the DID is able to make updates or deactivate it without the permission from any central authority and have multiple DIDs dedicated to different digital identities [60].

A Verifiable Credential (VC) is an electronic record that confirms certain statements about a person's identity, like age, nationality, or education. The credentials are issued by a trusted entity (identity issuer) with a digital signature that enables independent verification of its authenticity and integrity. VCs are held and managed by the credential owner and can be confirmed by others (verifiers or service providers) without having to communicate with the issuer directly and therefore, eliminating the third-party between the credential holder and the service provider. The privacy and consent can be maintained by holders presenting proofs from their VCs without disclosing the actual credential, giving them control over the extent of information shared. This selective credential sharing guarantees confidentiality and approval in transactions [35]. The W3C outlines a VC model in a JavaScript Object Notation (JSON) document. This document includes credential metadata such as context, issuance and expiration dates, the issuer's DID, credential claims regarding the subject's attributes, and credential proof with the issuer's digital signature in a JSON web signature, along with the algorithm used [13].

First, a person or organization generates a DID and utilizes it to securely handle access to their digital identity. This DID references a DID document detailing the required public keys and service endpoints for secure communication between the holder and the service provider [12]. A trusted entity issues VCs, like a digital driver's license or university degree, connected to the individual's DID. These VCs have cryptographic signatures, ensuring their immutability and verifiability. In situations where an individual must confirm specific details, like their age or education, to a service provider, they utilize their DID to display the appropriate VC or evidence stemming from it. By verifying the issuer's digital signature against the public key in the DID document, the service provider can confirm the VC's authenticity and the information it includes [30].

The standardized DIDs and VCs guarantee that an individual's digital identity and credentials can be used across various platforms and services, promoting interoperability and security. They are stored in the holder's wallet and are under complete ownership and control of the holder [12]. The basis of cryptography offers a strong level of security and confidence, as only the identity owner has control over their DID and VCs, and any verifier can verify the authenticity of the credentials on their own.

2.2.3 SSI Architecture

The components of an SSI system are arranged in a layered design that is reminiscent of the Transmission Control Protocol (TCP)/Internet Protocol (IP) architecture or the layered Open Systems Interconnection (OSI) model. The architecture outlined by the Hyperledger Aries project is most frequently utilized [22]. Figure 2.1 illustrates the four layers that make up this design. The technologies for expressing and storing credentials are defined in the first, bottom layer. While the Hyperledger Aries project uses the Sovrin Ledger [22] to build this layer, alternative decentralized databases, distributed ledgers, and blockchains can also be utilized. This layer is responsible for specifying DIDs and assigning cryptographic keys to them. Furthermore, communication protocols are added by the second layer to enable safe message exchanges between hubs, wallets, and agents [42].

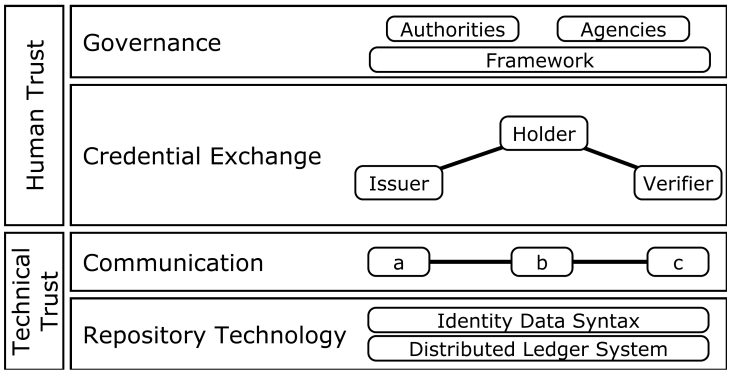


Figure 2.1: SSI layer-architecture adopted from [22].

The lower layers one and two provide the technical basis and are used as the foundation to establish organizational and human trust at layers three and four. The creation, sharing, and presentation of VCs are outlined in layer three. In layer four, the goal is to allow these credentials to be utilized in more than one specific setting, which spans a governance structure facilitating the cooperation and acceptance of credentials between organizations [42].

2.2.4 Metadata in SSI

Different components in SSIs produce metadata. Metadata is defined as "data about [other] data" [26]. The most crucial issued metadata lies in the VCs. The metadata

```
{
  "id": "http://example.ch/credentials/123",
  "type": ["VerifiableCredential", "StudentCredential"],
  "issuer": "did:example:rstuvwxyz0987654321",
  "issuanceDate": "2024-05-14T18:29:04Z",
  "credentialSubject": {
    "id": "did:example:1234567890abcdefghij",
    "name": "Lisa",
    "enrolledStudentAt": "University of Zurich",
    "subject": "Computer Science"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2024-05-14T18:29:04Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:example:rstuvwxyz0987654321#key-1",
    "jws": "pYw8XNi1...Cky6Ed="
  }
}
```


Figure 2.2: A simple VC [51].

includes an ID for the VC, the DID of the issuer, date of creation, validity period, proof link and the credential's type [52]. A credential subject's metadata usually consists of the DID of the subject, the name and other necessary attributes that are being transmitted to the verifier i.e. the subject and the enrolled university, as seen in Figure 2.2. The proof makes the VC immutable by employing digital signatures guaranteeing the authenticity and integrity of a credential subject, which is often referred to as the claim of the VC. This confirms that the user (*Subject*) has signed them with the metadata and their private key derived from a public/private keypair which is mapped to the issuer DID. Therefore, the integrity and authenticity of the VC is ensured [60].

2.3 The GDPR

The General Data Protection Regulation (GDPR) aims to protect individual privacy and personal data. It establishes fundamental principles and legitimate grounds for processing personal data, alongside outlining specific rights for individuals. The scope of the GDPR is limited to personal data; non-personal data is not covered by it. This segment provides insights into a selection of these components that are crucial to understand the challenges public blockchain systems face in complying to GDPR requirements. These problems also account for SSIs.

2.3.1 Personal Data

Article 4(1) [7] of the GDPR defines 'personal data' as follows:

"any information relating to an identified or identifiable natural person ('data subject')".

Instead of just 'data', the word 'information' is used, indicating that the data must contain some informational value. The distinction between information and data is not always clear. The definition of a 'data subject' is further expanded as:

> *"an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person".*

Legal entities and deceased persons are excluded from the GDPR, as personal data must relate to a 'natural person'. Additionally, it suggests that a single point of information would not be considered personal data. However, information combined with other information can be regarded as personal data if the combination infers the identity of the data subject. To conquer this problem, data pseudonymisation can be applied which is defined as:

> *"the processing of personal data in such a manner that the personal data can no longer be attributed to a specific data subject without the use of additional information, provided that such additional information is kept separately and is subject to technical and organisational measures to ensure that the personal data are not attributed to an identified or identifiable natural person".*

Based on Recital 26 [45], pseudonymised data can be regarded as personal data if the information on how the pseudonymisation was carried out is known:

> *"Personal data which have undergone pseudonymisation, which could be attributed to a natural person by the use of additional information should be considered to be information on an identifiable natural person."*

Furthermore, the recital states that:

> *"to ascertain whether means are reasonably likely to be used to identify the natural person, account should be taken of all objective factors, such as the costs of and the amount of time required for identification, taking into consideration the available technology at the time of the processing and technological developments".*

The key criterion is whether the means to identify an individual from the data are not just possible but reasonably probable under typical conditions. It is unclear what 'reasonably likely' means, as it heavily depends on the context. Furthermore, it can never be assessed with certainty what data is already available or what data may be released in the future, which makes the risk of re-identification through data linking unpredictable. Figure 2.3 visualizes the considerations in order to identify if data can be regarded personal or not.

2.3.2 Data Protection Principles

The GDPR establishes strict guidelines for data controllers and processors in order to ensure that personal data is treated in a fair and lawful manner. Article 5 [8] sets seven data protection principles to achieve this goal: (1.1) *Lawfulness, fairness and transparency*

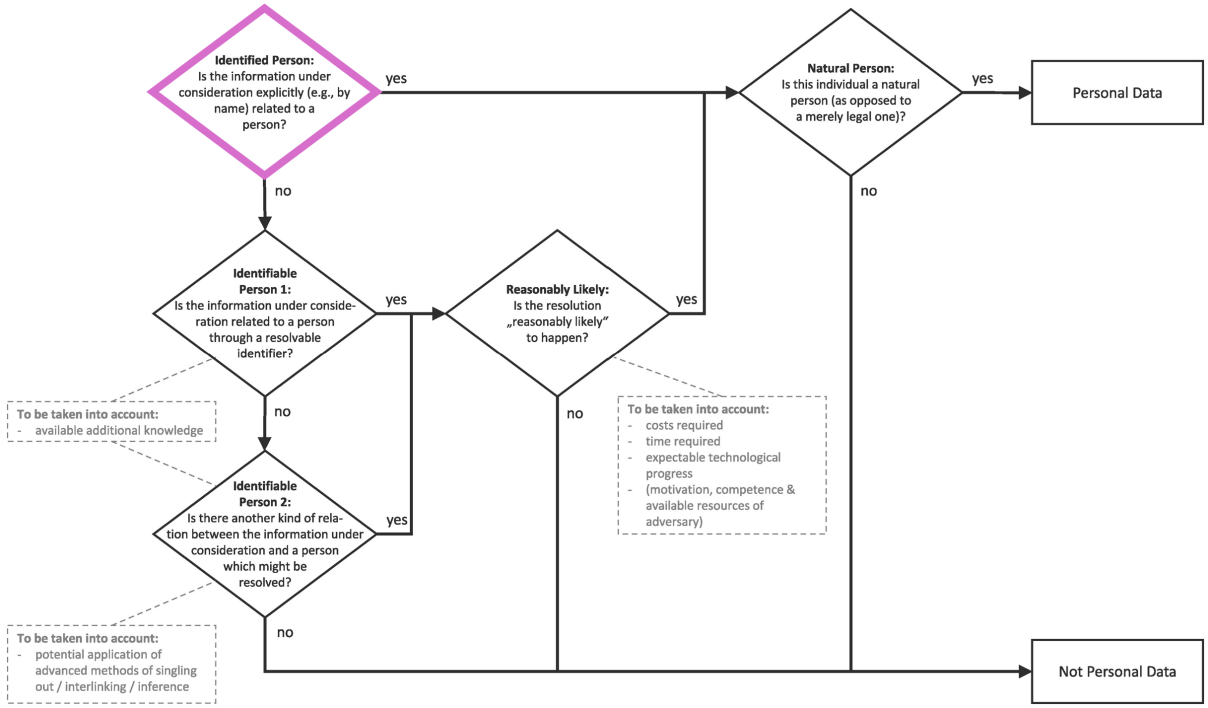


Figure 2.3: Assessment scheme for person-relatedness of data under the GDPR adopted from [19].

requires the data subject to consent to its data being stored and processed in a lawful way. Under 'fair' conditions, data must not be processed in an unreasonably harmful, unexpected or misleading manner to the data subject [25]. Additionally, clarity in the processing techniques is essential to be provided openly. (1.2) *Purpose limitation* indicates that personal data is only "collected for specified, explicit and legitimate purposes and not further processed". (1.3) *Data minimisation* demands data to be minimised to the relevant part and only include data that is "necessary in relation to the purposes for which they are processed". (1.4) *Accuracy* implies that the data must be up-to-date and replaced by old or inaccurate data. (1.5) *Storage limitation* mandates that data must be deleted when it is no longer needed and should not be kept for longer than is necessary for the purposes during which the data is processed. The storage period must be limited to a strict minimum. (1.6) *Integrity and confidentiality* requires the data to be stored and processed with proper security measures in order to protect the personal data. (1.7) *Accountability* demands a party to assume responsibility for the data storage and processing, as well as having appropriate measures and documentation in order to establish compliance.

Data subjects are given several rights in the GDPR. In order to comply with the GDPR, controllers and processors must both fulfill certain obligations and duties regarding these rights of data subjects. (2.1) *Right to erasure*, or right to be forgotten (RTBF), requires controllers to delete data in specific cases. According to Article 6(1) [9], data must be deleted when it is not necessary anymore for the purpose it was collected. Furthermore, data subjects have the right to ask for the removal of any connected personal data, as stated in Article 17 [4]. (2.2) *Right to rectification* (RTR) gives data subjects the right, as stated in Article 16 of the GDPR [3], to request that incomplete or erroneous personal data is updated to be accurate. As a result, this right is closely related to the previously

discussed GDPR accuracy principle (1.4). (2.3) *Right to be informed and right to access* mandates that data controllers notify data subjects about how their personal data is processed and stored. The right of access broadens this right, which enables people to request access to their personal data and obtain comprehensive information about the lawful processing of their data and how it is handled. (2.4) *Right to object and automated decision making* gives data subjects the ability to object to the processing of their personal data. Additionally, Article 22 [5] entitles data subjects to not have their decisions made exclusively on the basis of automated processing. (2.5) *Right to data portability* grants the data subject to move, copy and transfer personal data securely and efficiently from one electronic processing system to another, without compromising its usability. Furthermore, data subjects may request a provision of their personal data in a standardised readable computer format. (2.6) *Right to restrict data processing* allows data subjects in certain situations to request the suppression or restriction of their personal data. Situations that apply are inaccurate or incomplete personal data, unlawful processing, when the data subject needs the personal data to establish, exercise or defend a legal claim, and finally, if the data subjects have disapproved to processing their personal data.

(3) In Article 25 [6], the GDPR provides clear guidelines for the protection of personal data, including *data protection by design and by default*. Data protection *by default* requires controllers to implement appropriate organizational and technical measures designed to enforce data protection principles effectively [10]. This should be done in a manner ensuring that the data processing safeguards the rights of the data subjects. The measures may include pseudonymising personal data as soon as possible to minimize risks. Protection *by design* requires that only personal data which are necessary for each specific purpose of the processing are processed. This applies to the amount of data collected, the extent of their processing, the duration of their storage, and their accessibility.

2.4 Zero-Knowledge Proof

By default data is stored on the blockchain in plaintext [50]. The blockchain is a transparent database, where anybody can see any sort of transactional data or metadata. This data could be regarded as personal data and therefore the concept of the public blockchain goes against the GDPR guidelines. A Zero-Knowledge Proof (ZKP) enables writing only a proof of knowledge on the blockchain instead of the underlying data itself [27].

The concept of a ZKP is explained in the following example: Person A (prover) wants to prove person B (verifier) that he knows the (secret) Personal Identification Number (PIN) of his phone and therefore can unlock it without exposing the PIN to person B. First, A shows his phone to B in the locked state. He then enters the PIN code without person B seeing it and unlocks the phone. Now, A shows the phone in the unlocked state to B and therefore proved that he indeed knows the PIN code to his phone. It is possible, that person A just got lucky and typed in the correct PIN number. To rule out this case, we assume that the numbers on the keyboard of the phone change every time A types a digit, making it more difficult to enter the PIN by guessing. B asks A to unlock his phone another 20 times and B is now very certain that A indeed knows the PIN code of his phone with B having no knowledge of the actual PIN code.

The most common type of a ZKP is a zk-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge). They allow a prover to convince a verifier that the prover knows a secret without revealing the actual data about that secret. It consists of four main algorithms [31]:

1. Setup: This algorithm generates the initial public parameters that will be used to generate the keys and create the proof.
2. Key Generation: The proving key and the verification key are generated based on the setup parameters.
3. Proof: The proof is created with the proving key, a public statement and a secret.
4. Verification: Using the verification key, the public statement and the proof are checked for their validity. The output of the function is `true` or `false`.

The prover executes steps 1 to 3 and the verifier executes step 4. Regarding security requirements, zk-SNARKS satisfy the following fundamental properties of ZKPs:

- *Completeness*: If the statement is true, then the prover will convince the verifier.
- *Soundness*: It is not possible for the prover to generate a proof for the verifier without a private key. Furthermore, it is not possible for the prover to generate a proof without knowing the secret.
- *Zero-Knowledge*: The secret of the prover is protected, since it is not possible for the verifier to calculate or access the private key of the prover from the public statement.

In comparison to other privacy-preserving mechanisms, ZKPs are relatively time efficient and do not require a trusted third party, therefore allowing for peer-to-peer verification [31].

2.5 Homomorphic Encryption

Homomorphic Encryption (HE) allows calculations to be performed directly on encrypted data without the need for decryption first. The fundamental idea is to perform mathematical operations such as addition, multiplication or any other arbitrary function on encrypted data, which leads to the same result as the computation on the original, unencrypted data and encrypting it at the end. By preserving data encryption throughout the computational process, HE guarantees that sensitive data remains protected from unauthorized access and data breaches, thus complying with the GDPR requirements [43]. Figure 2.4 illustrates a real-world use case of HE.

Ideal encryption algorithms eliminate any relationships between plaintext and the corresponding ciphertext, therefore only making it possible to determine the value of the plaintext with the decryption key. However, plaintexts and ciphertexts must be related

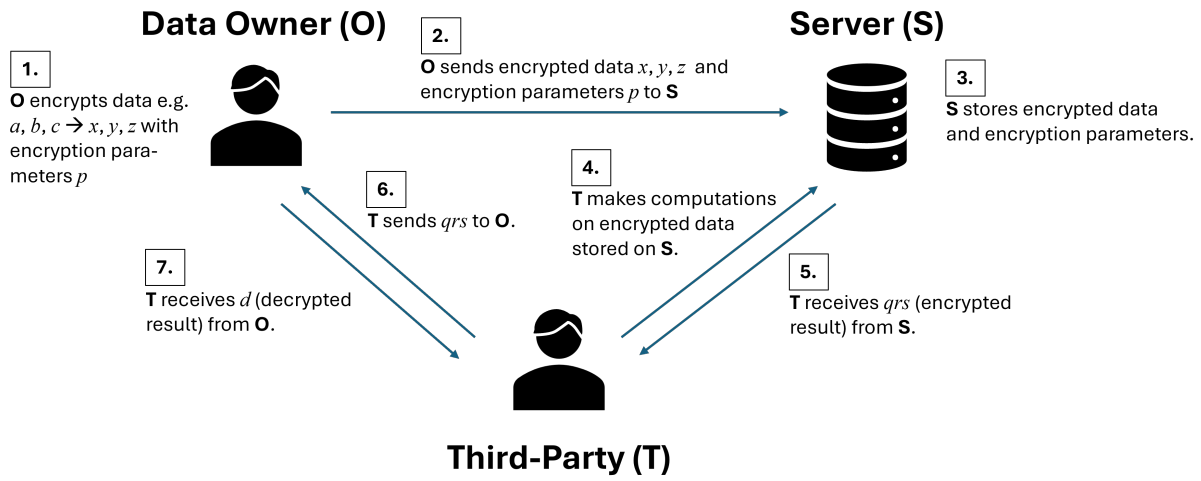


Figure 2.4: Basic application of HE where a third-party can only make use of the decrypted result and never knows anything about the underlying data.

in order to be able to execute mathematical operations on homomorphically encrypted data. The challenge is to implement the relationships such that an observer can not infer these, otherwise the encryption would be compromised. There exist three HE schemes, each varying in features, limitations and efficiency [1]:

- **Partially Homomorphic Encryption (PHE)** allows for a single mathematical function (addition or multiplication) to be performed on encrypted data an infinite number of times. The limitation comes with the benefit of efficiency. Since only one mathematical operation can be performed, PHE schemes like the Paillier cryptosystem are designed, such that the introduced noise by the encrypted calculation remains manageable. Thus, they do not require resource-intensive noise reduction algorithms.
- **Somewhat Homomorphic Encryption (SHE)** enables addition and multiplication operations on encrypted data, however only for a limited number of times, because every calculation adds a specific amount of noise into the ciphertext. The exact number of operations that can be performed before hitting the noise limit depends on the encryption parameters. Stricter encryption parameters i.e. larger key sizes, smaller modulus, or higher security levels reduce the number of possible calculations. Tighter security parameters usually involve more complex algebraic structures that cause noise to accumulate more rapidly with each operation. Also, additions increase the amount of noise linearly, while multiplications often induce noise at an exponential rate.
- **Fully Homomorphic Encryption (FHE)** is the most versatile scheme and allows for an infinite number of additions and multiplications of ciphertexts. In order to handle the exponentially growing amount of noise, different methods i.e. bootstrapping or relinearization are employed after a homomorphic computation. Noise-management algorithms however, are very computation- and storage-heavy and produce the most additional overhead out of the HE schemes [36].

Chapter 3

Related Work

This chapter provides an overview and analysis of related work that has been conducted on the privacy consideration of metadata and public keys. It further outlines possible attacks on metadata, mitigation strategies thereof and vulnerabilities of SSI systems. The chapter concludes with arguments for metadata and public keys being regarded as personal data and specifies the research gap that this thesis aims to fill.

3.1 Metadata and the GDPR

The GDPR significantly heightened the standards for privacy and data protection across Europe, posing important questions about the nature and treatment of metadata under these new regulations. Transactional data and public keys which can directly or indirectly identify a person may be seen as personal data. This could occur through various methods, such as patterns of activity, linking a public key to an identity through additional data, or associating it with real-world entities, such as through a KYC process. Encrypted data might still be considered personal under the GDPR if the encryption key is accessible because it could potentially be decrypted and reveal personal information [10]. The main challenges for metadata in complying with the GDPR laws are the following: First, the RTBF is hard to achieve due to the immutable nature of a blockchain, where data once written can not be altered or deleted. The GDPR requires a hard-deletion of the affected data [60]. Gonçalves et al. proposed the solution of storing personal data in the InterPlanetary File System (IPFS), which allows for modification and deletion, instead of on the blockchain [32]. Although the hashes are stored on the blockchain, they can not be linked to the personal data on the IPFS if they have been deleted there. The second difficulty is to protect and process personal data lawfully and transparently.

A possible solution to tackle the pseudonymisation and anonymisation problem is to utilise ZKPs, which make it harder to link data back to an individual. The downside of them is the high computational power, which the blockchain does not provide yet. Therefore, ZKPs are computed off-chain.

3.2 Attacks to Extract Metadata in SSI

Stokkink et al. [53] criticise current SSI approaches for their lack of network-level privacy, arguing that without addressing it, SSIs can not fully ensure user privacy. Their proposed solution, TrustChain IDentity (TCID), meets seven functional requirements to provide these desired system properties: peer-to-peer communication, authenticity and integrity of messages, network-level anonymisation, decentral synchronization of published information, credential disclosure to identify users and accountability of subjects. It demonstrates that while TCID introduces greater latency due to network-level anonymisation, it still remains practical for use. The analysis emphasizes the need for broader focus in SSI research beyond data disclosure protocols to include network-level anonymity for effective privacy preservation. In the research, they do not detail specific attacks that directly exploit metadata on SSI platforms but emphasize general privacy concerns associated with digital identities. It underlines the importance of network-level anonymity and highlights the potential vulnerabilities that arise without it, suggesting that without measures to mask the source-destination of messages, users' identities could be compromised through traffic analysis. [53]. This concern indirectly relates to metadata privacy, as metadata (such as communication patterns and transaction times) could be used in such analyses. The TCID system's emphasis on anonymisation shows an approach to mitigate potential attacks that could exploit metadata to undermine privacy.

Maesa et al. explore how ZKPs can protect attribute privacy but still leave room for metadata attacks if not properly encrypted or isolated [13]. Metadata can be subject to different attacks i.e. replay attacks and reusing proofs that were generated in the past but related to expired values of the attributes of the identity holder. These could be for example revoked values from VCs that are not valid anymore. To prevent fraudulent proof attempts, it is crucial to have a reliable way to link the private attribute values to the specific individuals they belong to. An immutable method, like hashing, can be used to securely and efficiently establish this link, ensuring that the relationship between the identity holder and their attributes can not be tampered with.

An evaluation of potential attack surfaces was carried out by Naik et al. [38]. The evaluation distinguishes three potential attacks on an SSI system: faking an identity, identity theft and Distributed Denial of Service (DDoS) attacks. A user might unintentionally expose credentials without realizing the possible risks to their privacy. Even if the user thinks he is anonymous, linking attacks can collect information from credential presentations to identify individuals by analyzing background data. Personal information that is commonly gathered can be collected under various pseudonyms (DIDs), i.e. VCs can link pseudonyms by matching data sets.

Naik et al. outlined three potential attacks on SSIs and created an attack tree for each of them [37]. In order to produce fake credentials from an issuer, the attacker must either acquire administrator credentials (through malware or social engineering) or store fake credential information (by infiltrating the provider's server or altering data). To get fake credentials, a person can create fake credentials, pretend to be the issuer by spoofing, alter legitimate credentials, or steal them. Three forms of spoofing include: posting a fake issuer DID on the SSI network, making a duplicate issuer host, or decreasing trust in

the real issuer through Sybil or Eclipse attacks. In order to alter credentials, the attacker must either acquire the private key of the issuer (by gaining access to the issuer's host and finding the private key through social engineering or malware) or sign the modified credential in the wallet. In order to steal credentials, the attacker must either take a wallet (such as stealing a phone or attacking the user's cloud storage) or pretend to be the user (by obtaining and requesting credentials). Naik et al. [37] further discuss the danger of acquiring personal information by highlighting three vectors: accessing the user's wallet without permission, background data attacks and credential creep. In order to gain unauthorized access to the wallet, the attacker must acquire the user's login information (through malware or social engineering) and gain access to the wallet (either by stealing the phone or obtaining remote access). When performing a background data attack, the attacker has to acquire a confidential dataset and link data through a verifier request. Credential creep can happen when extra information or user profiling is requested, such as multiple verifier requests for credentials and linking DIDs to identify the user.

Le et al. conducted an in-depth cyber risk analysis highlighting different types of external threats that could pose risk to metadata in SSI systems [28]. In order to achieve faster accessibility, wallets may store some sensitive information in plaintext on the user's device. Usually, tag names, record IDs and values are always encrypted, except when a certain prefix is added to the name of the tag value. This is usually done to speed up complex database searches. Also, should the Android or iOS device be rooted respectively jailbroken, attackers may implement malicious code altering the execution of data and therefore extract sensitive information [59].

Grüner et al. [24] evaluated the attack surface of SSIs by applying the STRIDE model to seven SSI components: user agent, VC store, organizational agent, trust, data store, verifiable data registry and communication channels. STRIDE abbreviates for Spoofing, Tampering, Repudiation, Information conflict of interest, Denial of Service, and Elevation of Privilege. They identified 35 SSI-specific threats and 15 protection measures and conclude that the user agent, organisational agent and the verifiable data registry are the most vulnerable to potential threats. Spoofing, denial, and repudiation of identity actions pose significant risks to both the user and the organizational agent.

Fröhlich et al. developed a user-centered threat model for cryptocurrency users, which outlines different forms of attacks on cryptocurrency users in general [23]. Fraudulent wallet software may send upon address generation the private key to the attacker giving him access to the wallet, all funds and all the data which may include personal data. In addition, the wallet provider could implement backdoors granting secret illicit access to data and funds. Keyloggers installed by third-party apps or malicious wallet apps can gather the user's login credentials and send them to the attacker. Moreover, unintended smart contract vulnerabilities may lead to the loss of funds or an exploit of personal data. These threats could also pose risk to metadata being accessed in an unauthorized way and stolen, which, however, is not mentioned in the research. SSIs are not mentioned specifically, but the outlined threats i.e. a keylogger may gain unauthorized access of the attacker to personal data stored in the SSI application.

Pöhn et al. systematically analyzed and categorized related work in the field of security threats concerning SSIs [42]. They analysed threats regarding the blockchain and

evaluated if these threats can also apply for SSIs. They identified that only a handful of blockchain threats affect SSIs. Potential SSI-specific threats were discovered on the human layer (i.e. social engineering, human errors and wallet threats), data layer (i.e. trusted third parties not following the laws) and network layer (i.e. Sybil and Eclipse attacks) of the blockchain. They conclude that network layer threats to SSI can be mitigated through effective governance. Furthermore, the analysis of threats to SSIs is still evolving, and the identified threats tend to be broad and not specific to any particular implementation.

Table 3.1 summarizes the research papers that have been found concerning and mentioning metadata within SSIs, personal data issues of the GDPR, attacks on and vulnerabilities of SSIs, as well as outlined solutions to mitigate the threats. No research paper mentioned the consideration of metadata being regarded as private data nor specific attacks on metadata within a blockchain-based SSI system.

Table 3.1: Related work comparison.

Research Paper	**Metadata in Blockchains**	**Metadata in SSI**	**GDPR**	**SSI attacks and vulnerabilities**	**Mitigation strategies**	**Metadata is private data**	**Specific attacks on SSI-metadata**
Yildiz et al. [60]		✓	✓	✓	✓		
Belen-Saglam et al. [10]	✓		✓		✓		
Gonçalves et al. [32]	✓		✓		✓		
Stokkink et al. [53]		✓		✓	✓		
Maesa et al. [13]		✓		✓	✓		
Le et al. [28]		✓		✓	✓		
Pöhn et al. [42]		✓		✓	✓		
Naik et al. [38]				✓	✓		
Naik et al. [37]				✓			
Grüner et al. [24]				✓	✓		

3.3 Public Keys and the GDPR

Public keys are visible on the blockchain to anyone. There has been a consensus in the literature for public keys being considered as personal data, when associated with other information, as public keys are often utilized to infer the origin of transactions [10]. For example, when a user posts his public key online to receive donations, the public key could be linked to his real-world identity especially if he doesn't use a pseudonym. The question however, still remains when public keys turn into personal data. For example, when transferring the ownership of a house using the blockchain, the neighbor suddenly could associate the public key containing the ownership of the house with the person who owns it [18]. However, it is possible that the public key has not had any other direct or indirect links to real-world entities before the transfer of ownership of the house. Furthermore, regulatory requirements force centralized exchanges to enforce KYC and Anti-Money Laundering (AML) obligations on their clients which lead to the disclosure of real-world identities connected to public keys [14].

Additionally, when a natural person uses the same public key for several transactions, it may be possible to analyse patterns which could identify the subjects behind the public key [47]. However, a public key is not always linked to a natural person, as legal entities also can issue public keys. Since the GDPR does not account for legal entities, public keys would not be considered personal data [44]. When a key can not be reasonably linked or does not belong to a natural person, it is anonymous [16].

3.4 Deletion of Metadata on Blockchains

In SSIs, metadata is generated upon execution of a transaction i.e. when issuing a VC. The concern is whether the metadata can directly or indirectly infer a person's identity and therefore be considered as personal data. According to the GDPR, there must exist a possibility to delete personal data. This is not possible on a public permissionless blockchain due to its immutability. However, when using a consortium (permissioned) blockchain where a selected group of participants have privilege rights, it is possible to implement the deletion feature. The same possibility exists for private blockchains where one single entity has full control of the blockchain [10]. The biggest problem of the latter two blockchains is the missing trust and integrity, which goes against the nature of a fully transparent, trusted and tamper-proof blockchain system.

When looking at the issuance date of a VC in the case of a university degree, as shown in Figure 2.2, it is unlikely to link the metadata to a natural person. The issuer, in this case the university, will most probably issue the VCs of the obtained degrees for all students on the same day. With such a large number of VCs containing the same issuance date, the privacy of credential owners is not threatened.

Assuming that the issuance date was linked to a natural person's identity and that the deletion of metadata on a public blockchain were possible, then naturally, the owner of the VC or another party will delete that specific metadata from the blockchain, therefore

exercising the RTBF. However, this would render his VC invalid, since the issuance date of the VC is an essential part of the whole credential. Credential verifiers would not accept the VC when presented to them for verification, therefore leading the credential owner to ask for a new VC from the issuer.

3.5 Considerations of Metadata and Public Keys Being Personal Data

Research has shown that it is not entirely certain from the beginning, which and whether metadata in SSIs or public keys are considered as personal data. The consensus is that as soon as metadata and public keys have been generated and are visible on the blockchain, they are not considered as personal data. The reason lies in the definition of personal data by the GDPR that a single point of information would not be considered as personal data. Furthermore, it is generally not possible to know the exact moment, when metadata or public keys become personal data. The most probable reason for them to be labelled as personal data is through linkage, e.g. when a public key is being assigned via KYC to a natural person registered at a central exchange. The metadata generated from transactions from that public key will therefore also be considered as personal data.

Assuming that metadata in SSI and public keys are considered personal data under the GDPR, the outlined key regulations (1.1) to (1.7) and (2.1) to (2.6) must be adhered, as described in Section 2.3.2. Since metadata on SSI are usually stored off-chain on the user's personal device, the user has the ability to manage, process and delete his personal data according to the GDPR regulations.

Public keys on the other hand, are stored on-chain and can be seen by anyone without a considerable amount of effort and cost. Therefore, it remains debatable whether public keys are considered personal data. When a user utilises the blockchain for the first time and assuming he creates a public key without deriving it from another public key, the created public key can not be linked to another public key. This scenario also rules out that the user did create an address via a third-party central exchange that requires KYC. In that case the public key can not be linked to a real person.

In conclusion, it is assumed that metadata and public keys become personal the instant it is feasible to identify the natural person behind them. Therefore they are initially, when generated, not regarded as personal data and will remain non-personal until the linkage to a real-world identity. The important concern is whether a particular dataset can be linked with other datasets from the standpoint of availability rather than technological feasibility. Therefore, data in the present is not considered legally qualified based on the possibility that the entity in possession of the dataset may obtain or have access to additional information in the future that, when combined, may enable identification.

3.6 Problem Statement

This literature review has shown that no research has been done on whether metadata in SSIs is considered to be personal data according to the GDPR. Most of the papers mention that especially the RTBF or the RTR are essential for blockchain systems to be GDPR-compliant which is hardly possible to achieve due to the blockchain's immutable nature. Metadata was never mentioned in connection to the GDPR and if, it did not cover the main part of the research and usually was brought up in the introduction or alongside other topics in the research section.

Furthermore, no research was found that specifically mentions attacks on metadata within blockchain-based SSIs. Attacks on SSIs mostly concern the application as a whole or faking, modifying or stealing identities and credentials. Metadata in general is affected indirectly and usually at risk due to user behaviour i.e. jailbraking or rooting the smartphone and therefore reducing the security of the device allowing for unauthorized installation of keyloggers or other malware that exploit metadata.

Since the deletion of metadata on a public blockchain is not possible, it must be ensured that metadata can not reveal the identity of a natural person. Even if SSI-metadata is stored on the user's device, it is essential to protect the privacy of metadata to prevent future linkage and therefore avoid metadata turning into personal data. This thesis seeks to fill the research gap as presented in Table 3.1 and provide additional security for specific metadata in blockchain-based SSI systems with the proposal of two privacy-preserving mechanisms which are described in the next chapter.

Chapter 4

Design

This chapter presents the design of two privacy-preserving mechanisms that ensure metadata privacy within blockchain-based SSI systems. First, the situation and the underlying platform are explained on which the two prototypes are designed. The next part illustrates the communication and calculation sequences between the involved parties. Finally, the prototype requirements are listed and additional security considerations for SSI applications are mentioned that help protect the privacy of metadata.

4.1 Situation to Evaluate

A student graduates from a university with his bachelor's degree. He receives it in the form of a credential from the university (issuer). When applying for a job, the company wants to know when the university degree has been issued. The company demands an "up-to-date" software engineering degree. Since this data could potentially be considered personal, the system is designed to be GDPR-compliant to prevent future linkage to a real-world identity. Furthermore, the design aims to minimize any additional overhead. In the best case, the company (verifier) should not know the underlying timestamp of the student's degree issuance date.

The calculation of the task is as follows: The company requires the degree to have been issued after Friday, January 1st 2010 00:00:00 which corresponds to the Unix timestamp of 1262304000. The student's degree was issued at Unix timestamp 1500000000 which dates to Friday, July 14th 2017 02:40:00. To determine whether the degree was issued after the company's specified date, we subtract the issuance date from the company's threshold date. If the result is negative, then the degree was issued after the threshold date, otherwise it was issued before.

4.2 Architecture

CredChain is the underlying blockchain-based SSI platform architecture used in this paper, which was developed by a team from UNSW Engineering at the University of Sydney. *CredChain* allows for secure creation, sharing, verification and revocation of credentials. The user can maintain control over their personal data, such as through selective disclosure, therefore reducing the risks associated with traditional centralized identity systems [40].

The underlying platform for *CredChain* is Hardhat, a local Ethereum network designed for smart contract development. Hardhat was developed by the Nomic Foundation and operates on TypeScript, which allows for an easy integration of comprehensive TypeScript and JavaScript libraries [21]. It includes extensive debugging and performance measurement functions which will be used in Chapter 6 for evaluation.

4.3 Proposed Privacy-Preserving Mechanisms

In the context of SSI systems, maintaining the privacy and security of sensitive metadata while ensuring compliance with regulatory standards is crucial. The two GDPR-compliant privacy-preserving mechanisms ZKP and HE have been chosen to enhance the privacy of the student's bachelor's degree issuance date, in order to protect the underlying data which is being transmitted between the two parties.

The two prototypes focus on preserving the privacy of the issuance date field in the credential, as it could be linked to a real-world entity when combined with other information. Other metadata such as credential ID, DID of the issuer, validity period, proof link or credential's type have a significantly lower or no chance of identifying the person who owns the VC. The credential ID is the identifier of the credential itself and not directly tied to the individual's identity. The DID of the issuer identifies the credential issuer, in this case the university which is not a natural person. The validity period just indicates the timeframe during which the credential is valid. The proof link verifies the authenticity of the credential but does not contain personal details about the holder. Finally, the credential type does not directly identify the credential holder.

These metadata attributes become potentially identifying when linked with additional specific personal data. However, the question remains how much effort and data is needed to link the metadata to a natural person in the future. Therefore, the prototypes focus on the privacy-preservation of the issuance date, which has among the other metadata the highest chance of being linked to a natural person.

4.3.1 Zero-Knowledge Proof

With ZKPs, an entity can cryptographically demonstrate to another that he is aware of a piece of information without disclosing the real underlying information. There is no

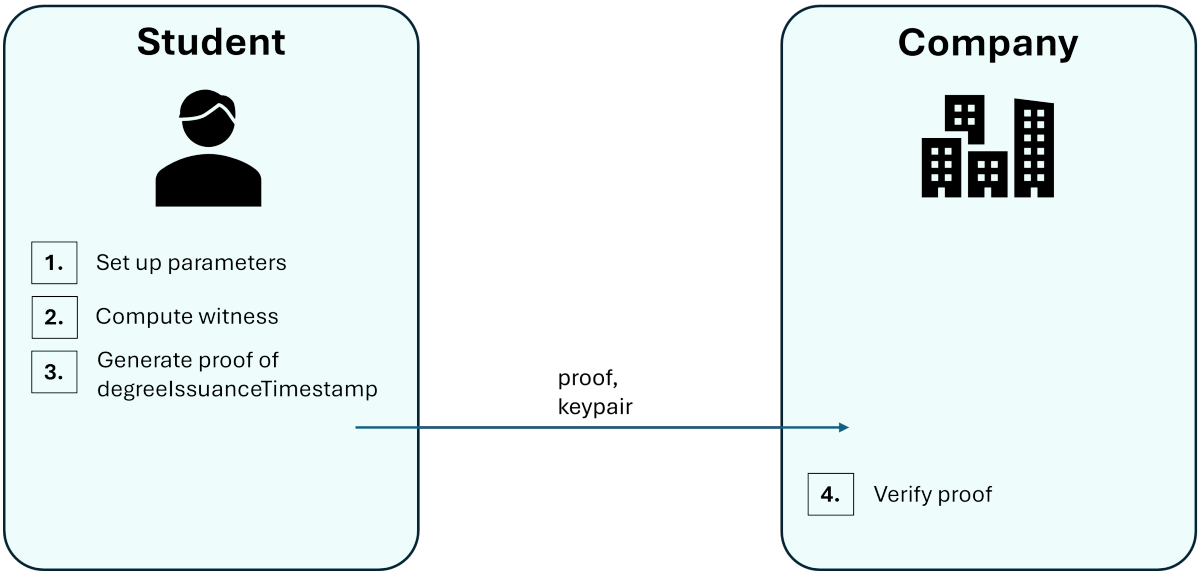


Figure 4.1: Design of a privacy-preserving solution for metadata in SSIs using a ZKP.

need for a trusted third party to verify the knowledge of the entity issuing the proof and therefore it is a peer-to-peer privacy-preserving solution. Although ZKPs already have been implemented in SSI systems to present credentials by not revealing the underlying data [34], they have not been used as mechanisms to protect the privacy of metadata.

Figure 4.1 illustrates the design and sequence of functions of the ZKP-solution: The student sets up the parameters in order to generate the ZKP proof. After the setup, the witness is computed by checking if the issuance date is larger than the threshold date, which would lead to the boolean output of `true`. Next, the proof that the degree was issued after the threshold date is created without revealing the actual issuance date. The student then sends the proof and the keypair which was generated during the setup to the company. The company verifies the proof using the keypair and therefore confirms or denies that the student's issuance date is smaller or larger than the threshold date specified by the company.

4.3.2 Homomorphic Encryption

The design of the HE-solution with its data transmission and calculation sequences is illustrated in Figure 4.2: The company sets and provides the encryption parameters. After the initial setup, the company encrypts the threshold date and sends it with the generated public key and the other encryption parameters to the student. The student loads the received encryption parameters in order to encrypt his issuance date and perform the calculation correctly. After the retrieving of the setup data is successful, the student encrypts the issuance date and calculates the difference between the issuance date and the threshold date. The obtained encrypted result is sent with the encrypted issuance date back to the company. To verify that both parties calculated correctly, the company performs the same calculation again with the received encrypted issuance date. Finally,

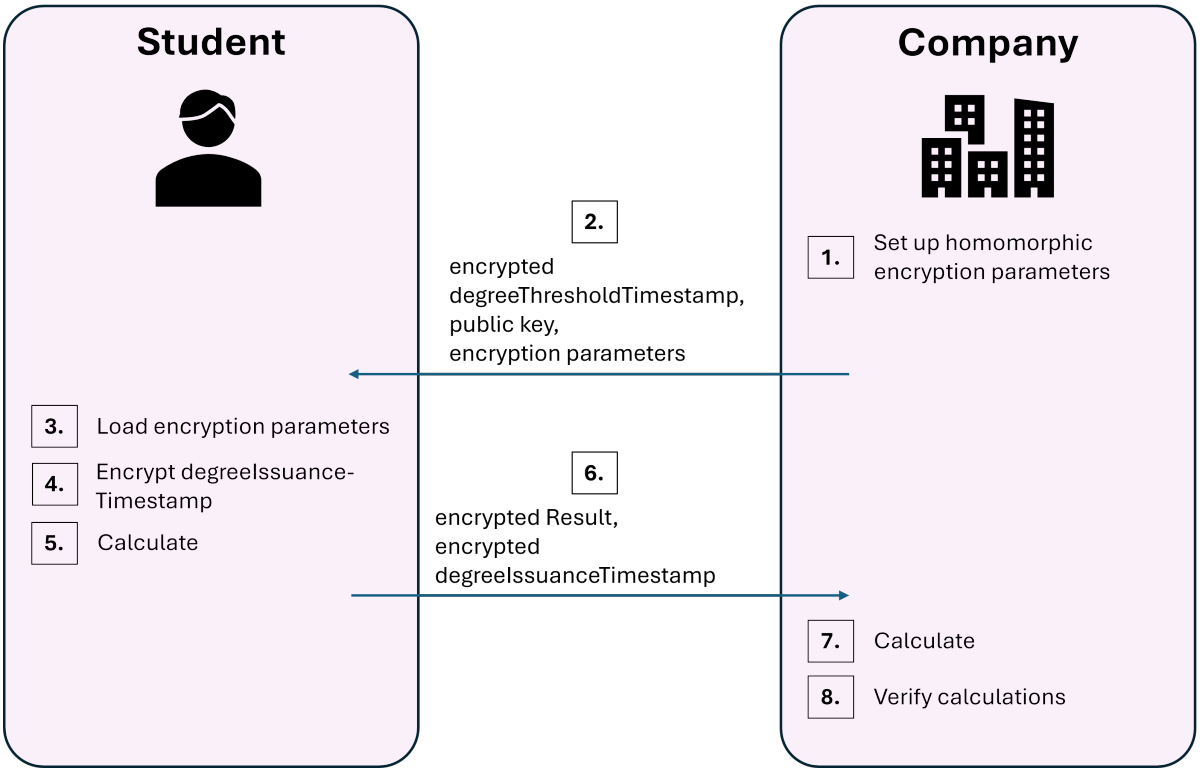


Figure 4.2: Design of a privacy-preserving solution for metadata in SSIs using HE.

it verifies whether its result matches with the calculation result sent by the student and therefore outputs the boolean values `true` or `false`.

4.3.3 Requirements

The following ten requirements outline the essential criteria for achieving a robust and GDPR-compliant privacy-preserving solution in SSI environments for both mechanisms, ZKP and HE.

Privacy and Security:

R1. Privacy: The specific metadata (issuance date) must not be revealed to the company.

R2. Integrity: Ensure that the encrypted data and results are not tampered with and validate the integrity of the encrypted computations.

R3. Security: Decryption keys must not be generated.

R4. Peer-to-peer architecture: The solution must operate in a peer-to-peer manner without the involvement of a trusted third party.

R5. GDPR-compliance: The privacy-preserving mechanism must comply with the regulatory requirements.

Performance and Efficiency:

R6. Additional overhead minimization: Both parties have to be able to encrypt, compute or verify the calculations or proofs in a timely manner.

R7. Sufficient computational resources: The prototype uses ordinary CPU and memory resources.

R8. Cost-efficiency: Minimization of gas (Gwei) consumption.

Compatibility and Usability:

R9. Interoperability: Ensure compatibility with existing systems and environments.

R10. Usability: The user should be able to easily generate and verify proofs.

4.4 Additional Security Considerations

To mitigate the risks of attacks on or leakage of metadata, as described in Chapter 3.2 the following procedures should be considered:

SC1. Jailbreak/root detection: To prevent malicious attackers to gain potential access to data within the SSI application, the application should detect whether the device it is running on has been jailbroken or rooted. Attackers may have implemented malicious code that could change data execution and extract sensitive information. The user should be warned that the data may be at risk by using the application on his rooted device.

SC2. Integrated keyboard: The application should provide its own keyboard to prevent spying of the phone's manufacturer or other keyloggers.

SC3. Encrypted database: The data generated by the SSI should be securely stored in an encrypted database on the user's device, rather than in individual files within the file system.

SC4. Network level security: In order to reduce the risk of metadata getting exploited during transmission, a VPN (Virtual Private Network) should be used to encrypt the data and tunnel it to protect unauthorized entities to gain access to it.

SC5. Public key privacy: The application should generate a new public key for each credential it sends to another entity in order to prevent future linking.

These additional security considerations are not implemented in the two prototypes, as they are security measures that the SSI application itself should consider and provide.

Chapter 5

Implementation

This chapter provides a guide through both privacy-preserving prototypes proposed in the previous chapter. The first section outlines the technologies, libraries, and explanations used in the project. The second section details the implementation of the ZKP solution, while the third section covers the implementation of the HE solution.

5.1 Tech-Stack and Technologies

The implementation in this section was done using a modified version of *CredChain* [49] as template, which is a SSI system developed for the Ethereum blockchain. Unused files were removed and the repository was renamed to *CredChain-Privacy* [55] containing the two implemented privacy-preserving prototypes.

5.1.1 Languages

The development of *CredChain* involved the use of multiple programming languages and technologies to create a robust and secure SSI platform. The core implementation which includes a frontend and backend component was developed in JavaScript. The smart contracts were written in Solitidy, the standard language for smart contract development on the Ethereum Blockchain. Hardhat is the Ethereum development environment used in *CredChain* which uses TypeScript as its programming language. The two privacy-preserving prototypes for metadata (ZKP and HE) are developed in JavaScript, ensuring easy integration and compatibility with the existing system. The benchmark evaluation scripts are written in JavaScript, while the graphing scripts of the result were coded using Python.

5.1.2 Libraries

Due to the open-source nature of blockchain-based applications, likewise open-source libraries were chosen to provide anybody with transparent code.

5.1.2.1 ZoKrates-js

For the implementation of a ZKP in *CredChain*, the ZoKrates JavaScript library zokrates_js [15] is utilized. ZoKrates is a toolkit that allows to integrate zkSNARKs on the Ethereum blockchain using JavaScript. The library is light-weight enabling the integration directly in the frontend of an application.

5.1.2.2 node-SEAL

The implementation of HE into *CredChain* was done utilizing node-SEAL [29], which is a JavaScript wrapper for Microsoft's Simple Encrypted Arithmetic Library (SEAL) [33]. node-SEAL provides an easy-to-use interface to the SEAL library, enabling to perform encrypted arithmetic operations such as addition and multiplication directly within JavaScript applications.

5.1.3 Explanations

The following terms will occur frequently throughout this thesis. For clarity, they are explained:

- Prover: The student will be referred to as the "prover" who has to prove that the issuance date of his degree is greater than the specified threshold date defined by the company.
- Verifier: The company will be referred to as the "verifier" with the task to verify the claim made by the prover (student).
- `degreeIssuanceTimestamp`: The issuance date of the student's (prover's) bachelor's degree in form of a Unix timestamp.
- `degreeThresholdTimestamp`: The threshold date set by the company (verifier) for the bachelor's degree to be accepted in form of a Unix timestamp.

5.2 Zero-Knowledge Proof Prototype

The implementation of the ZKP prototype involves two main stages: generating the ZKP by the student and verifying the ZKP by the company. This process ensures the privacy and integrity of sensitive information, such as the issuance date of a degree, without revealing the actual value of `degreeIssuanceTimestamp` to the verifier. This solution assumes that the user already knows the `degreeThresholdTimestamp`, which has been sent to him beforehand.

5.2.1 Prover Side: Generating the ZKP

After initialization of the zokrates_js library, the *source* code used to create the proof is compiled. The function in Listing 5.1 takes two integers as input. The field degreeTimestamp corresponds to `degreeIssuanceTimestamp` and the field thresholdTimestamp to `degreeThresholdTimestamp`. The return value of the function is a boolean: `true`, if the degreeTimestamp is valid and larger than the thresholdTimestamp, otherwise `false`. The function also handles negative Unix timestamps which correspond to dates before January 1st, 1970 00:00:00.

```
1 const source = `
2     def main(private field degreeTimestamp, field thresholdTimestamp) ->
          bool {
3         return degreeTimestamp > thresholdTimestamp;
4     }
5 `;
6 const artifacts = zokratesProvider.compile(source);
```

Listing 5.1: ZoKrates program to validate the degree year.

The next phase consists of setting up the parameters that generate the cryptographic keys for the proof. The setup function on line two in Listing 5.2 generates a keypair specifically tailored to the provided *source* program in Listing 5.1. This setup phase requires initial parameters that must be kept secret to ensure the security of the ZKP system and returns two keys:

- Proving Key: Used by the prover (student) to generate a proof that a certain computation was done correctly without revealing the actual inputs or computation details.
- Verification Key: Used by the verifier (company) to check the validity of the proof generated by the proving key. The verifier can be certain that the computation was performed correctly if the proof is valid.

The setup and keypair generation is followed by the computation of the witness. The function computeWitness() takes the compiled ZoKrates *source* program (artifacts), the `degreeIssuanceTimestamp` and the `degreeThresholdTimestamp` as inputs. Now, the ZoKrates runtime executes the zkSNARK circuit using the provided inputs. The output of the witness contains three components:

- Input values: Artifacts, `degreeIssuanceTimestamp` and `degreeThresholdTimestamp`.
- Intermediate values: values generated by intermediate calculations.
- Output value: the boolean value indicating whether the degree issuance timestamp is valid according to the threshold.

In this case there are no intermediate values as the *source* code contains exactly one operation which is the comparison of the degreeTimestamp to the thresholdTimestamp resulting in a boolean value. Otherwise, each intermediate step can be verified and thus it is evident to the verifier how the result and each step in between was achieved. Here, even without intermediate values the witness still encapsulates the necessary information to prove the correctness of the computation. It shows that:

1. The prover knows the values for degreeTimestamp (`degreeIssuanceTimestamp`) and thresholdTimestamp (`degreeThresholdTimestamp`).
2. The prover can demonstrate that degreeTimestamp is greater than thresholdTimestamp without revealing the underlying values of both timestamps.

The witness includes all the data required to prove that the computation was done correctly without revealing the underlying input values to the verifier and the validity of the statement that the degree issuance date is valid. It is essentially preparing the data for the next step, the generation of the proof.

```
1 // Setup phase
2 const keypair = zokratesProvider.setup(artifacts.program);
3
4 // Compute witness
5 const { witness } = zokratesProvider.computeWitness(artifacts, [
     degreeIssuanceTimestamp, degreeThresholdTimestamp]);
6
7 // Generate proof
8 const proof = zokratesProvider.generateProof(artifacts.program, witness,
      keypair.pk);
```

Listing 5.2: Setup followed by witness computation and proof generation.

The last step on the prover's side is to generate the proof with the proving key. The generateProof() function uses the compiled *source* program, the created witness and the proving key as input and executes the circuit. This step involves the recalculation of constraints defined in the *source* program. Each constraint evaluation step represents a recalculation, and the total number of recalculations depends on the number of constraints. In this case there is exactly one constraint which evaluates whether the degreeTimestamp is greater than the thresholdTimestamp, therefore resulting in one recalculation being made. Finally, the proving key is used to encode this information into a proof, ensuring that the proof can be verified independently without revealing the actual input values.

In the end the user has the cryptographic proof of the witness and the verification key which he sends to the verifier. This proof can be verified using the verification key, ensuring the computation's integrity and privacy.

5.2.2 Verifier Side: Verifying the ZKP

The verifier receives the proof and the verification key from the prover. Before he can start verifying he also initializes the zokrates_js library. The verification process checks the validity of the proof against the given verification key.

The code in Listing 5.3 interprets the result of the proof verification. The proof includes, among other fields, two "input" values, one being the hexadecimal value of the `degreeThresholdTimestamp` and the other being the hexadecimal representation of the output of the *source* program which is `true` or `false`. The `degreeIssuanceTimestamp` of the prover is not known to the verifier. Since the boolean result of the *source* program is always found in `proof.inputs[1]`, we can hardcode the message according to the hexadecimal output. The output "1" states it is true that the `degreeIssuanceTimestamp` is larger than the `degreeThresholdTimestamp` whereas "0" means false.

```
1  const isVerified = zokratesProvider.verify(vk, proof);
2
3  // Interpret the result based on the inputs
4  const result = proof.inputs[1];
5  if (result === '0
       x0000000000000000000000000000000000000000000000000000000000000001') {
6      console.log('\tThe Issuance Date is after the Threshold Date.');
7  } else if (result === '0
       x0000000000000000000000000000000000000000000000000000000000000000') {
8      console.log('\tThe Issuance Date is before the Threshold Date and
           therefore INVALID.');
9  } else {
10     console.log('\tUnexpected result in proof inputs.');
11     return false;
12 }
13
14 return isVerified;
```

Listing 5.3: Verification and interpretation of the proof.

In general, the proof itself is always valid, unless it has been tampered with. In that case the witness can not be reconstructed using the verification key thus making the proof invalid. In the rare case that the result of the *source* program is something other than '0' or '1', the function is set to return `false`, indicating the same scenario as an invalid proof.

5.2.3 Summary

The integration of a ZKP using the ZoKrates library in *CredChain* provides a powerful solution for secure and private verification of metadata. The approach shows that a complex cryptographic mechanism can easily be integrated into an existing SSI platform. There is no need for a trusted third-party and the verifier can prove the witness of the prover without knowing the actual data. Furthermore, the usage of zkSNARKs do not require an interactive communication between the two entities. With only one interaction, the verifier can be certain of the integrity and authenticity of the provided proof.

5.3 Homomorphic Encryption Prototype

The second prototype utilises HE and contains three main stages: Parameter setup by the company (verifier), calculation by the student (prover), and recalculation and verifi-

cation by the company (verifier). As with the previous prototype, the verifier does not know the underlying value of the student's `degreeIssuanceTimestamp`. Additionally, in this solution, the prover does not know the underlying value of the company's `degreeThresholdTimestamp`.

5.3.1 Verifier Side: Encryption Parameters Setup

Upon initializing the node-SEAL library, the first step is to set the encryption parameters which will be used by the verifier and the prover to encrypt their secret data, which the other entity should not know. Listing 5.4 outlines the various parameters required to set the security properties for the encryption, which will be needed for the calculations at a later stage. The solution uses the standard parameters recommended by the documentation [46]. Choosing larger integers for the encryption settings such as the polynomial modulus degree or the coefficient modulus bit sizes increases the security and allows for more complex computations. The trade-off lies in exponentially larger memory usage and calculation duration.

```
1  const schemeType = seal.SchemeType.bfv;
2  const securityLevel = seal.SecurityLevel.tc128;
3  const polyModulusDegree = 4096;
4  const bitSizes = [36, 36, 37];
5  const bitSize = 20;
6
7  const parms = seal.EncryptionParameters(schemeType);
8
9  // Set the PolyModulusDegree
10 parms.setPolyModulusDegree(polyModulusDegree);
11
12 // Create a suitable set of CoeffModulus primes
13 parms.setCoeffModulus(seal.CoeffModulus.Create(polyModulusDegree,
       Int32Array.from(bitSizes)));
14
15 // Set the PlainModulus to a prime of bitSize 20.
16 parms.setPlainModulus(seal.PlainModulus.Batching(polyModulusDegree,
       bitSize));
17
18 const context = seal.Context(parms,    // Encryption Parameters
19     true,                               // ExpandModChain
20     securityLevel                       // Enforce security level
21 );
```

Listing 5.4: HE parameters setup.

The type of HE used is the Brakerski/Fan-Vercauteren (BFV) scheme as defined in the first line of Listing 5.4. This scheme is usually classified as SHE which allows a limited number of addition and multiplication operations. SHE is ideal for this prototype, because it provides a balance between usability and cryptographic strength.

After the configuration of the encryption parameters according to the scheme type, the expansion of the modulus chain and enforcement of security level, the encryption context is created from which the following components are generated in Listing 5.5:

- Encoder: The batch encoder is used to encode plaintext values, i.e. the `degreeThresholdTimestamp`.
- Key Generator: The key generator creates public and private keys. This implementation does not create a private key to decrypt the encrypted result of the calculation in order to preserve the privacy of the underlying value of the calculated result.
- Public Key: The public key is used to encrypt the encoded plaintext values.
- Encryptor: The encryptor encrypts the encoded plaintext values in context using the generated public key.

```
1  const encoder = seal.BatchEncoder(context);
2  const keyGenerator = seal.KeyGenerator(context);
3  const publicKey = keyGenerator.createPublicKey();
4  const encryptor = seal.Encryptor(context, publicKey);
5
6  // Encode the numbers
7  const plainText = encoder.encode(Int32Array.from([
       degreeThresholdTimestamp]));
8
9  // Encrypt the PlainTexts
10 const cipherText = encryptor.encrypt(plainText);
11
12 // Create the JSON object
13 const companySetupData = {
14     parms: parms.save(),
15     publicKey: publicKey.save(),
16     cipherTextThreshold: cipherText.save(),
17 };
18
19 return companySetupData;
```

Listing 5.5: HE component generators.

The final step is to save the encryption parameters, the public key and the encrypted `degreeThresholdTimestamp` into a JSON object. This JSON object is returned by the verifier and passed (or sent) to the prover who will use the parameters for the upcoming calculation.

5.3.2 Prover Side: Calculation on Encrypted Data

The prover receives the JSON object from the verifier and loads the three data components. First, the encryption parameters are loaded from which the encryption context is created. This step is important to ensure that the encryption format and homomorphic calculation are consistent with those of the verifier. It is essential that both parties encrypt and calculate in the same manner to be assured that the calculation has been done correctly. Using the context from the setup, the prover generates its own batch encoder and encryptor. Additionally, a new evaluator is initialized which will perform the homomorphic operation. Subsequently, the public key and the encrypted `degreeThresholdTimestamp` from the verifier are loaded using the verifier's context as shown in Listing 5.6.

```
// Load the context with saved parameters
const parmsFromFile = seal.EncryptionParameters();
parmsFromFile.load(setupData.parms);

const contextFromFile = seal.Context(parmsFromFile, true, securityLevel);

const encoder = seal.BatchEncoder(contextFromFile);
const encryptor = seal.Encryptor(contextFromFile, publicKeyFromFile);
const evaluator = seal.Evaluator(contextFromFile);

const publicKeyFromFile = seal.PublicKey();
publicKeyFromFile.load(contextFromFile, setupData.publicKey);

const cipherTextFromFile = seal.CipherText();
cipherTextFromFile.load(contextFromFile, setupData.cipherTextThreshold);
```

Listing 5.6: Loading the HE setup parameters from the verifier.

The next step is to encode and encrypt the prover's `degreeIssuanceTimestamp` by using the encoder, the encryptor and the public key from the verifier. To be able to perform the calculation, the cipherTextResult object must be initialized. Now that both values are in cipher text form, the homomorphic operation can be executed. The `degreeIssuanceTimestamp` (= cipherText) is subtracted from the `degreeThresholdTimestamp` (= cipherTextFromFile). The sub() function of the evaluator performs the calculation on the encrypted values and stores result in the cipherTextResult variable as seen in Listing 5.7.

The encrypted `degreeIssuanceTimestamp` and the encrypted result are packaged into a JSON object, which is returned by the prover. The JSON object is then sent back to the verifier, who will perform the same calculation again. For debugging purposes, this data is written to a file.

```
// Encode the numbers
const plainText = encoder.encode(Int32Array.from([
    degreeIssuanceTimestamp]));

// Encrypt the PlainTexts
const cipherText = encryptor.encrypt(plainText);

// Subtract A from B and store it in cipherTextResult
const cipherTextResult = seal.CipherText();
evaluator.sub(cipherTextFromFile, cipherText, cipherTextResult);

// Create the JSON object
const studentData = {
    cipherTextIssuanceDate: cipherText.save(),
    cipherTextResult: cipherTextResult.save()
};

// Save the results to file
fs.writeFileSync('studentData.json', JSON.stringify(studentData));

return studentData;
```

Listing 5.7: Performing the HE calculation.

5.3.3 Verifier Side: Recalculation and Verification

After the prover performed the homomorphic calculation on the two encrypted values and returned them to the verifier, the verifier performs the same calculation again to verify that he gets the same result and that the data has not been tampered with during the transmission. First, the encrypted result needs to be loaded. The cipherTextResultStudent object is initialized and then retrieved from the JSON object. The built-in load() function of the node-SEAL library automatically checks the integrity of the ciphertext. If the format of the ciphertext is incompatible with the given context, it throws an error indicating that the data may have been modified or that different encryption parameters have been used. The error is caught in the try-catch block which leads the verifier to return `false` (Listing 5.8).

```
1 const cipherTextResultStudent = seal.CipherText();
2 try {
3     cipherTextResultStudent.load(context, studentData.cipherTextResult);
4 } catch (error) {
5     // Error occurs for "wrong" format, meaning if the result has been
          altered
6     console.log("\tIncompatible cipher text format.");
7     return false;
8 }
```

Listing 5.8: Verifier loading the encrypted result from the prover.

If the encrypted result is successfully loaded, the verifier constructs the evaluator based on the context from its initial setup phase. It then initializes the cipherTextResult object which will store the outcome of the encrypted calculation. The same homomorphic calculation is performed again by subtracting the prover's encrypted `degreeIssuanceTimestamp` (= cipherTextIssuanceDateStudent) from the verifier's encrypted `degreeThresholdTimestamp` (= cipherTextThresholdDate).

After the calculation, the prover's result and the verifier's result in form of ciphertexts are converted to strings. Listing 5.9 shows a simple string comparison to check if they are identical. If the results are equal, it means the prover's encrypted result is valid and has not been modified. Otherwise, it indicates potential tampering. Finally, the verifier returns the boolean value whether the result is valid or not.

```
1  // Company performs the same computation
2  const evaluator = seal.Evaluator(context);
3  const cipherTextResult = seal.CipherText();
4  evaluator.sub(cipherTextThresholdDate, cipherTextIssuanceDateStudent,
       cipherTextResult);
5
6  // Convert ciphertexts to strings for comparison
7  const companyResultString = cipherTextResult.save();
8  const studentResultString = cipherTextResultStudent.save();
9
10 // Compare the company's computed encrypted result with the student's
       encrypted result
11 const isResultValid = companyResultString === studentResultString;
12
13 if (isResultValid) {
```

```
14      console.log("\tEncrypted results identical");
15  } else {
16      console.log("\tEncrypted results NOT identical");
17  }
18
19  return isResultValid;
```

Listing 5.9: Recalculation and verification of results.

5.3.4 Summary

This prototype shows that two independent entities can agree on knowledge of one party without knowing anything about the data used for the calculation. Both the prover and the verifier transmit encrypted data and perform a calculation on these. Using the node-SEAL library, it is straightforward to implement a complex privacy-preserving mechanism into an already existing SSI platform. A trusted third-party is therefore not needed to oversee the integrity of the calculation and encrypted values, since the verifier can detect whether the encrypted values have been modified.

Chapter 6

Evaluation

This chapter evaluates the two implemented prototypes. Following a description of the measurement setup, the analysis focuses on the security, privacy, and interception implications of the data, as well as its compliance with the GDPR. The next section measures the used computational resources and the scalability aspect. Finally, the defined requirements are summarized and evaluated.

6.1 Setup

The two prototypes were evaluated on a Windows 11 64-bit computer with an Intel Core i7-1165G7 at 2.8 GHz base frequency and 16 GB of RAM. The Central Processing Unit (CPU) contains 4 physical cores and 8 logical cores, allowing for hyper-threading. The measurements were done in the ”Performance-Mode” which provides a maximum clock speed of 4.2 GHz. To measure the duration, CPU and memory occupation during the test runs, the in-built Node.js library perf_hooks [41] was used.

6.2 Evaluation Criteria

To ensure compliance with the GDPR, the two prototypes are thoroughly evaluated in terms of security, data privacy, and the potential for data interception during transmission and its consequences. Given that enhanced security and privacy often come at the cost of performance, the evaluation also covers resource usage and concludes with an analysis of scalability.

6.2.1 Security and Data Privacy

When generating a proof and verification key for the same underlying statement, they will never be identical. During proof generation, ZKPs incorporate randomness ensuring

each proof and verification key to be unique. By including nonces or random numbers, it is practically infeasible to link multiple proofs and verification keys together.

In HE, the encryption process of plaintexts follows a similar structure using nonces resulting in different ciphertexts for the same underlying data. Encrypting a plaintext multiple times will always produce a different ciphertext, because of the randomness used for the encryption process. Likewise, it is not possible to gain knowledge about the underlying data and encryption relations between different ciphertexts, therefore ensuring data privacy.

6.2.2 Data Interception

Data is being transmitted between the prover and the verifier in both prototypes. This section evaluates the risks of them being intercepted and altered by an attacker.

Zero-Knowledge Proof

The prover sends the proof and the verification key to the verifier. If the proof gets intercepted, the malicious actor can not find out the underlying value of the metadata, since the proof alone is insufficient to reveal any private information about the metadata. The interception of the verification key does also not provide any information about the underlying metadata. If the attacker intercepts the proof and the verification key, it is still infeasible to gain knowledge of the underlying data. Also, he can not derive any additional information from having both.

The risk of a replay attack by sending the proof and verification key to another verifier is reduced as well, since the proof proves that the `degreeIssuanceTimestamp` is larger than the `degreeThresholdTimestamp`. The threshold may vary for different verifiers or not even be part of the proof at all. To increase the security, TLS/SSL communication channels should be used which protect the data during transit on the network layer [53].

Altering one or both parts of the data would result in the verifier being unable to verify the proof and the verifier would know that the data has been tampered with.

Homomorphic Encryption

The first data transmission package consists of the encryption parameters and the encrypted `degreeThresholdTimestamp` which the verifier sends to the prover. Upon interception of the encryption parameters, the security is not compromised. Essentially, no decryption key is even created, therefore leaving no possibility for the verifier, prover or anyone else to decrypt the data, since only the public key is being shared. The second data transmission package contains the homomorphically calculated result and the prover's `degreeIssuanceTimestamp`, which are both encrypted.

When intercepting data and sending an altered version to the next entity, the following situations will occur:

- Encryption parameters: First, the prover would either encrypt his data in the wrong format or won't be able to encrypt the data at all, because the corrupted parameters can not be read correctly. Secondly, the calculation will result in an error since the verifier's encrypted `degreeThresholdTimestamp` will not be compatible with the prover's encrypted data, since it has been encrypted with different parameters. As long as a strong HE scheme is used, such as BFV or similar, the intercepted data remains secure. This is typically ensured by default.

- `degreeThresholdTimestamp`, `degreeIssuanceTimestamp` and encrypted result: If the attacker exchanges these data points with its own correctly encrypted values, the prover would calculate with wrong values and thus yield a wrong result. However, the verifier re-calculates it using his original `degreeThresholdTimestamp` and would therefore realise that his result is not equal to the prover's result.

One possible way to mislead the verifier would require the following circumstances to apply: The attacker knows what type of content and the context thereof in the data transmission packages. We assume he knows that the prover must prove to the verifier that his issuance date is after a specific threshold date. In that case, when knowing the encryption parameters and the `degreeThresholdTimestamp` intercepted from the first data transmission package, he can create and encrypt his own issuance date, execute the calculation and send it to the verifier. In this highly unlikely scenario, the verifier could be fooled. Otherwise it is practically infeasible to deceive the verifier. To mitigate the risk of an attacker to intercept useful data, TLS/SSL protocols should be used to protect the data during the transmission on the network layer [53].

6.2.3 GDPR-Compliance

By using the ZKP or the HE prototype, users and organisations can process and verify data while ensuring privacy and security, therefore making both prototypes GDPR-compliant. With HE, computations can be performed on encrypted data. The sensitive personal data does not have to be decrypted and therefore remains private and secure. Although HE allows for creating a decryption key, the prototype does not create it leaving the result of the computation encrypted. The ZKP protocol disallows the generation of a decryption key from the beginning. Furthermore, the users can select which part of data they want to send to the other entity. The generated data is not stored on-chain, but locally at one of the two parties in encrypted form and not revealing anything about the underlying data itself. Finally, the fundamental feature of the *CredChain* environment allows the user to revoke any credential sent to a verifier.

6.2.4 Computational Resources

The values for the performance of the prototypes result from running them 100 times consecutively. When measuring the CPU usage for the functions, approximately 40-60% of the test runs yielded the value 0. In that case, the value has been removed from the

dataset and the remaining non-zero CPU measurements have been used in the graph. Possible reasons for the zero measurements include asynchronous function calls, which might prevent the CPU measurement from capturing the actual time spent accurately, especially during short measurement periods. Another cause could be the operating system's process scheduler, which dynamically assigns processes among the cores influencing the CPU measurements.

The performance of both prototypes are presented side by side in a box-and-whisker plot, which contains the following components:

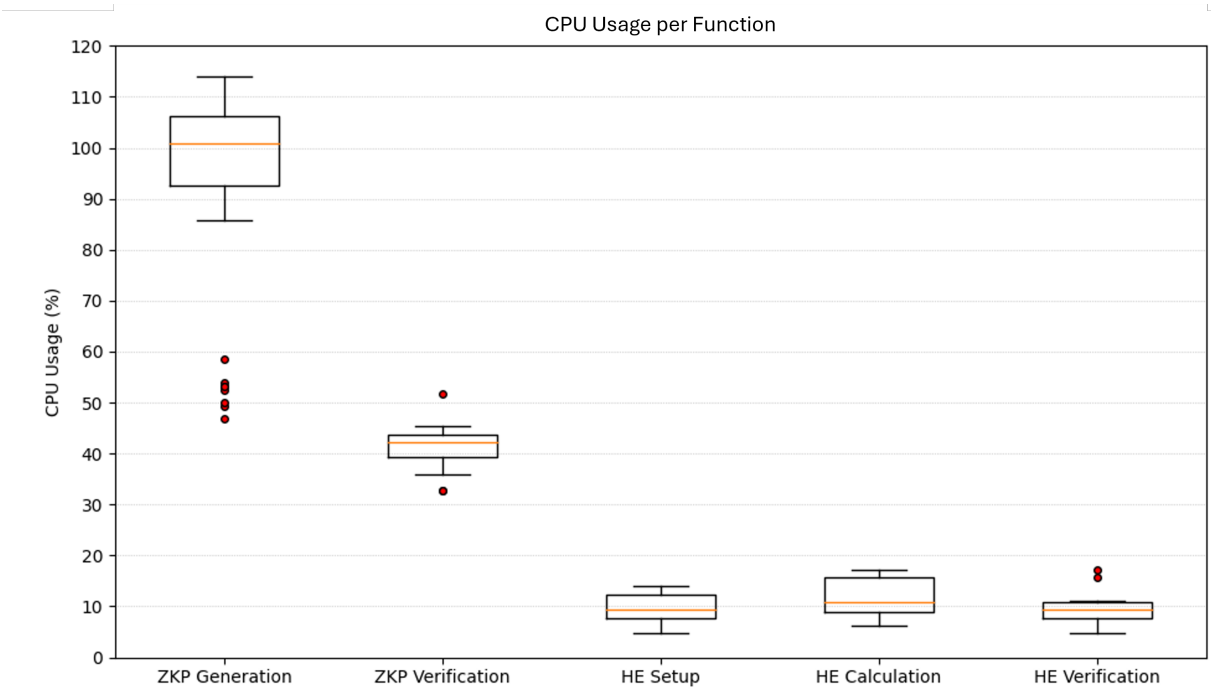
- *Box*: The main rectangular box of the plot shows the interquartile range of the dataset, which lies between the first quartile and the third quartile containing the middle 50% of the data.
- *Median Line*: The orange line within the box represents the median of the data. If the median is exactly in the middle of the box, the data is distributed symmetrically. Otherwise, it indicates skewness.
- *Whiskers*: These lines extending the box at the top and bottom show the smallest and largest values within 1.5 times the interquartile range from the lower and upper quartiles.
- *Outliers*: Data points outside the whiskers are considered outliers and marked as red dots.

Figures 6.1, 6.2 and 6.3 contain the graph and a table underneath with the corresponding measured values of the respective functions outlined on the x-axis.

6.2.4.1 CPU Usage

The ZKP prototype clearly uses more CPU resources in comparison to the HE prototype, as shown in Figure 6.1. The reason for that are computationally intensive mathematical operations, especially when generating the proof like elliptic curve operations, polynomial evaluations and interpolations, bilinear pairings, witness computation and more. All of these are necessary to ensure the security and correctness of the generated proofs. Surprisingly, the ZKP proof generation takes up about 8 times the amount of resources in comparison to the HE calculation. It turns out that a simple one-operation computation on encrypted values is efficient. On the other hand, the ZKP proof verification is designed to be efficient and is less CPU-intensive than the ZKP proof generation. However, on average it still uses 4 times as much processing power as the HE verification.

The majority of the CPU usage values for the ZKP proof generation exceed 100% of one single core. This is possible since every physical core can handle two threads, i.e. the maximum measured value of 114.1% shows that the proof generation used 100% of one thread and 14.1% of another thread. It is also possible that the load is being split more equally among the threads. With the median being above 100% it signifies that



MAX	114.10 %	51.60 %	14.00 %	17.20 %	17.10 %
AVG	94.10 %	41.12 %	9.60 %	11.74 %	9.62 %
MEDIAN	100.75 %	42.15 %	9.40 %	10.95 %	9.40 %
MIN	46.95 %	32.80 %	4.70 %	6.30 %	4.70 %

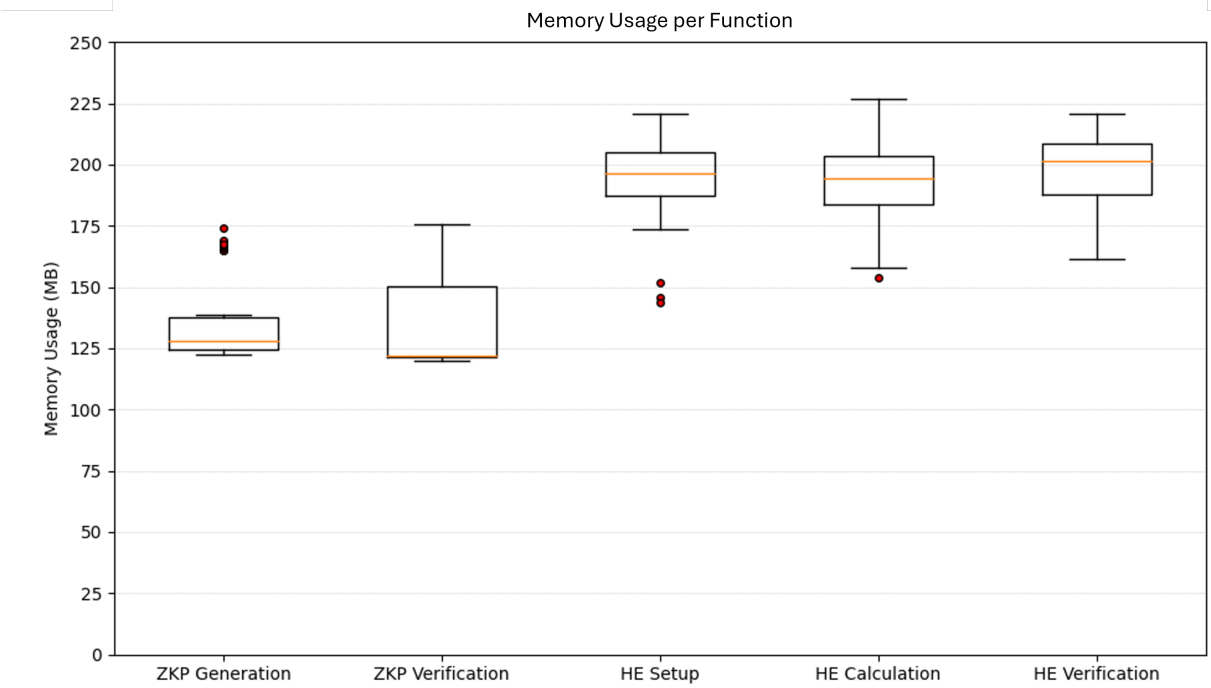
Figure 6.1: CPU usage comparison between the ZKP and HE prototype functions.

most of the time more than one thread was used implying the proof generation to be computationally heavy.

The noticeable outlier values for the ZKP proof generation could have occurred due to different factors. A likely reason may be the interruption of system tasks that temporarily take away CPU cycles during the proof generation. Also, in the Node.js environment, garbage collection processes may start during program execution that run for a longer period of time in combination with high CPU and memory usage. This clean-up process uses CPU that temporarily reduces the resource for generating a proof.

6.2.4.2 Memory Usage

Figure 6.2 visualizes a significantly smaller difference in memory usage in comparison to the CPU usage between the two prototypes. It is noticeable that the median of both ZKP functions but especially verifying the ZKP are at the bottom of the range, indicating that they mostly occupy a similar memory amount. Although both have rather high values that are far away from the median, most test runs used memory amounts near the minimum measurement value. In contrast, the memory measurements for the HE functions are distributed symmetrically, as indicated by the median being in the middle of the box.



MAX	174.14 MB	175.48 MB	220.52 MB	227.04 MB	220.78 MB
AVG	136.33 MB	132.87 MB	193.68 MB	193.06 MB	197.59 MB
MEDIAN	128.16 MB	121.77 MB	196.41 MB	194.53 MB	201.56 MB
MIN	122.55 MB	120.03 MB	143.84 MB	153.80 MB	161.72 MB

Figure 6.2: Memory usage comparison between the ZKP and HE prototype functions.

The HE prototype generally uses about 60% more memory than the ZKP prototype. One reason lies in the size of the ciphertexts. The plaintext `degreeThresholdTimestamp` has ten characters, while its ciphertext contains around 118'000 characters. Calculating with values of this size requires more storage. Another reason would lie in storing the accumulated intermediary results. Furthermore, noise reduction would be applied which is also memory-heavy. In this case, since the single homomorphic operation does not store any intermediary results, this reason does not apply. Although the ZKP prototype also generates intermediate data during the proof generation, its data is more compact in comparison to the data in the HE prototype.

Outlier data points could be due to interference from other processes running during the measurement period. The grouping of these outliers, specifically in the ZKP Generation column suggests that these runs might have encountered other processes which used up a larger amount of memory than the usual system processes. Alternatively, memory-intensive processes may have terminated during the run, temporarily reducing memory consumption for the HE Setup measurement.

6.2.4.3 Estimated Gas Costs

Both prototypes need to send and store data. The measurements assume an ETH price of 2600 United States Dollars (USD) and a base network fee of 4 Gwei. The values can

be obtained through the following formulas:

- **Gas used (Gas)**: In Table 6.1 the gas used does not account for storing the data on-chain. Therefore, the formula $totalGas = baseTransactionCost + gasForData$.

  Table 6.2 considers storing the data on-chain, leading the calculation of gas used to be $totalGas = baseTransactionCost + gasForData + numberOfStorageSlots * (storageCostPerSlot + sLoadGasCost)$.

  The composition of the variables is described in Section 2.1.2.

- **Gas cost (ETH)**: $gasCostETH = gasPriceETH * totalGas$ where $gasPriceETH$ is the base network fee.

- **Gas cost (USD)**: $gasCostUSD = gasCostETH * priceETH$ where $priceETH$ is the price of one ETH in USD.

Table 6.1 illustrates the cost difference for sending the produced data to the other entity. The data generated in the HE prototype is significantly larger than that of the ZKP prototype. The HE setup parameters and ciphertexts are very long strings, resulting in a combined storage requirement that is approximately 236 times greater than that of the ZKP proof and verification key. Also in monetary terms, the two data transmissions of the HE prototype are 114 times more expensive than the two ZKP prototype data transmissions. While it would cost 0.033875 ETH or 88.08 USD to send the generated HE data, it would only cost 0.000299 ETH or 0.77 USD to send the produced ZKP data.

Table 6.1: Estimated gas usage when sending data via the Ethereum blockchain.

	HE setup data	**HE calc data**	**ZKP verification key**	**ZKP proof**
Data Size (KB)	289.96	230.56	1.45	0.75
Gas used (Gas)	4715192	3753672	43444	31200
Gas cost (ETH)	0.018861	0.015014	0.000174	0.000125
Gas cost (USD)	49.04	39.04	0.45	0.32

When comparing the gas usage and gas cost of sending data in Table 6.1 to the numbers in Table 6.2 it is clearly visible that storing data on the blockchain is expensive. The HE prototype stores around 520 Kilobytes (KB) with a cost of over 1.5 ETH or 3916 USD, which is 229 times more expensive than sending and storing 2.2 KB of ZKP data on-chain for a cost of 0.006574 ETH or 17.09 USD.

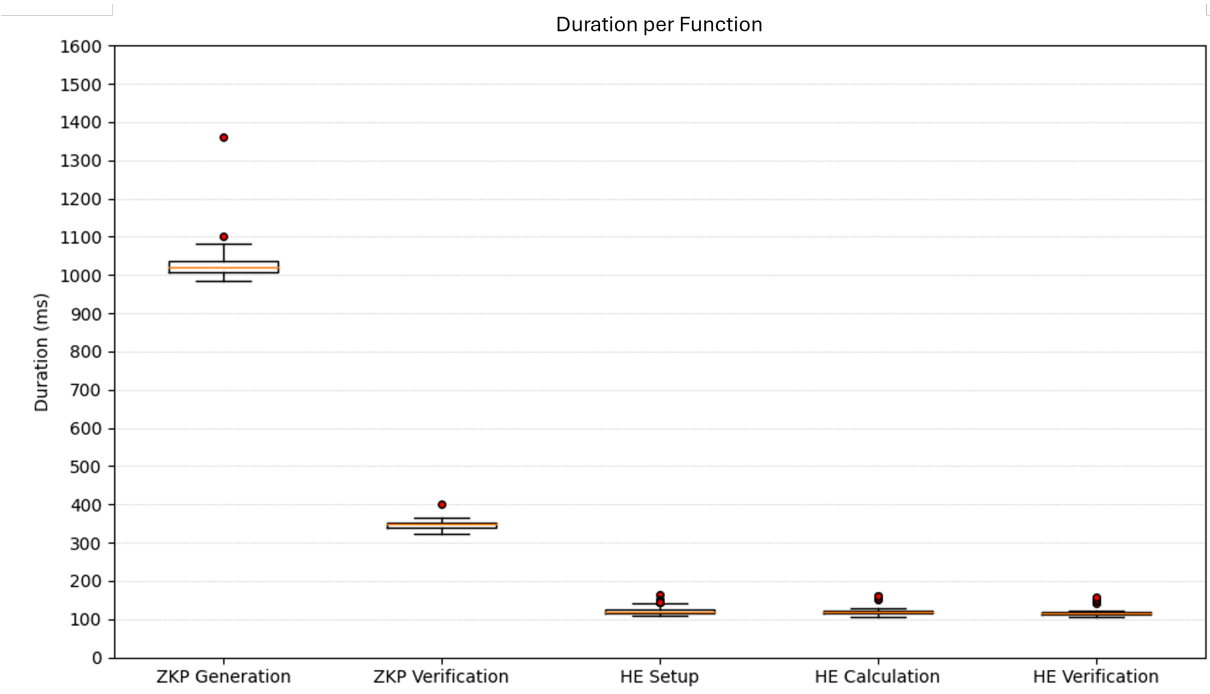
Table 6.2: Estimated gas usage for storing data on the Ethereum blockchain.

	HE setup data	**HE calc data**	**ZKP verification key**	**ZKP proof**
Data Size (KB)	289.96	230.56	1.45	0.75
Gas used (gas)	209781092	166807472	1082144	561600
Gas cost (ETH)	0.839124	0.667229	0.004328	0.002246
Gas cost (USD)	2181.72	1734.8	11.25	5.84

This gas estimation clearly shows that it is not economically feasible to store generated data on-chain on a regular basis. This explains why most data stored on-chain is in plaintext, since its encrypted form would drive up the costs massively. The measurement indicates that storing encrypted ciphertexts from the HE prototype is much more expensive than storing a ZKP proof and its verification key. The gas estimation made it clear that the HE prototype uses much more gas in comparison to the ZKP prototype for storing and sending metadata and preserving its privacy. In general, credentials within SSIs are transmitted through Hypertext Transfer Protocol Secure (HTTPS) or the Decentralized Identifier Communication (DIDComm) protocol. Furthermore, the two resource-intensive prototypes perform all computations off-chain, since these large-scale computations are infeasible to be carried out on a blockchain at the moment.

6.2.4.4 Time Measurement

Figure 6.3 visualizes the difference in execution duration for the individual prototype functions. The ZKP proof generation takes about 1 second on average and approximately 0.35 seconds to verify the proof. Multiple iterations of complex mathematical operations to generate the proof require time and CPU resources to be computed, which explains the large difference in execution time. In contrast, the HE prototype contains two single-operation calculations on encrypted data, which are rather light-weight on a relative scale.



	ZKP Generation	ZKP Verification	HE Setup	HE Calculation	HE Verification
MAX	1360.15 ms	400.03 ms	163.49 ms	161.68 ms	157.38 ms
AVG	1031.78 ms	347.23 ms	123.34 ms	121.50 ms	120.76 ms
MEDIAN	1020.65 ms	347.93 ms	118.36 ms	118.77 ms	116.61 ms
MIN	983.88 ms	323.80 ms	108.83 ms	107.36 ms	105.57 ms

Figure 6.3: Time measurement of the ZKP and HE prototype functions.

The three HE functions average around 0.12 seconds to complete. Surprisingly, setting up the HE parameters yielded the highest maximum, average and minimum values, even if no calculations are performed in that function.

The generated additional time overhead for the ZKP prototype averages at 1.37 seconds, which is 3.7 times more than the average time of the HE prototype taking 0.36 seconds to complete. These measurements only include the computation, writing and reading of data locally and do not factor in the time to transmit the data to the other entity.

6.2.5 Scalability

When comparing the scalability aspect of the two prototypes, the memory consumption comparison stands out. Figure 6.4 shows the memory consumption of 1000 consecutive iterations of both prototypes. There are four major observations:

1. Both ZKP function calls use the same amount of memory over time, but the values oscillate significantly between two ranges that gradually increase over time. There are more measurements on the upper orange-purple line of points that range from around 160 to 195 Megabytes (MB). Approximately every third measurement yielded a lower value ranging from 120 to 140 MB represented by the lower orange-purple line of points.

2. The first functions calls of the HE prototype utilize around 55 MB of memory. Over time, the memory usage increases linearly at an approximate rate of 70 to 80 MB per 100 prototype iterations.

3. After approximately 330 iterations, the memory utilization of the HE functions decline and stagnate for 30 to 50 iterations. This is most likely due to the garbage collection process, which reclaims memory that is not used anymore. Garbage collection usually gets triggered after a certain memory threshold is crossed. Another potential reason could be memory buffers that fill up and get flushed. As shown in Figure 6.1, the HE prototype stores around 520 KB of data in each iteration. After 330 subsequent iterations, an additional memory of 171 MB (= 520 KB * 330) is occupied, thus triggering the clean up and memory optimization process.

4. During the first 50 iterations, both prototypes achieve different memory measurements in comparison to further measurements. While the ZKP functions' values are more scattered, the HE function calls clearly utilize less memory than subsequent function calls. This is due to the "warm-up" phase where the system does not know exactly how much memory to allocate for the function calls. Memory pools might not be fully utilized, therefore leading to an initial lower memory usage. After the first HE garbage collection process at iteration 330, the measurement values are much more uniform with less scattering. This further indicates that the system tries to allocate only as much memory as needed.

The memory usage evaluation has highlighted the difference between both prototypes. When evaluating CPU and duration values, no outstanding observations occurred. The

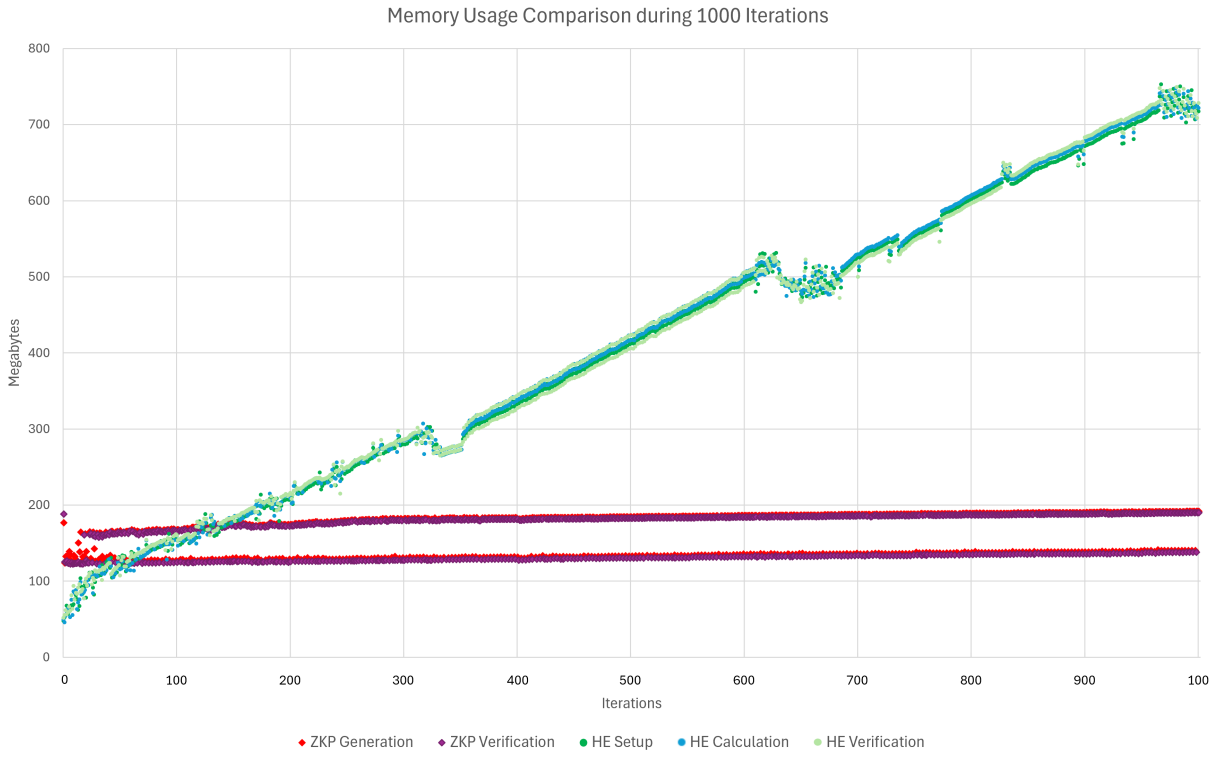


Figure 6.4: Memory usage of 1000 iterations for the ZKP and HE prototype.

CPU usage and time duration remained constant throughout the iterations, as visualized in Figures 6.1 and 6.3.

The scalability evaluation has shown that the ZKP prototype has a lower and more stable memory usage compared to the HE prototype due to the generation and storage of large ciphertexts. This makes the ZKP prototype more suitable for memory-constrained devices.

6.3 Requirements Evaluation

This section assesses whether the requirements defined in Section 4.3.3 have been fulfilled according to the evaluation of both prototypes.

Privacy and Security

The verifier (company) does not know the underlying value of the student's (prover) issuance date, therefore keeping the metadata private as required in ***R1***. The HE prototype takes this a step further by ensuring that the prover does not know the threshold date defined by the verifier, creating a zero-knowledge scenario for both parties. For ***R2***, both prototypes have checks implemented that verify the integrity of the received data. Should they fail, then the prototype automatically returns `false` indicating that there was a mistake when verifying the data meaning the data has been modified during transmission or by the other party. The security requirement ***R3*** is satisfied, because both prototypes

incorporate randomness in the encryption of the data. It is infeasible to figure out the relation between a plaintext and its corresponding ciphertext. Furthermore, the transmitted data is not at risk when being intercepted, because the attacker can not harm anyone. They can just verify the ZKP proof or the HE calculation. Although HE allows the creation of a decryption key, the prototype does not generate one. Both prototypes strictly communicate peer-to-peer and do not require a trusted third party to verify the data, therefore adhering to ***R4***. The implementations are GDPR-compliant, since the metadata is being encrypted and the verifier has zero-knowledge of the underlying data. In addition, it is not possible to link proofs and ciphertexts, thus satisfying ***R5***.

Table 6.3: Evaluation of the prototype requirements.

	Description	**ZKP**	**HE**
R1	**Privacy**: Demonstrated in Sections 5.2.1, 5.3.2 and 5.3.3.	✓	✓
R2	**Integrity**: Demonstrated in Sections 5.2.2 and 5.3.3.	✓	✓
R3	**Security**: Explained in Section 6.2.1.	✓	✓
R4	**Peer-to-peer architecture**: Demonstrated in Figures 4.1 and 4.2.	✓	✓
R5	**GDPR-compliance**: Explained in Section 6.2.3.	✓	✓
R6	**Additional overhead minimization**: Shown in Figure 6.3.	✓	✓
R7	**Sufficient computational resources**: Shown in Figures 6.1 and 6.2.	(✓)	✓
R8	**Cost-efficiency**: Estimated in Table 6.1 and 6.2.	✓	✗
R9	**Interoperability**: Explained in Section 5.1.	✓	✓
R10	**Usability**: Explained in Sections 5.2.3 and 5.3.4.	✓	✓

Performance and Efficiency

R6 is met although the ZKP prototype takes significantly longer to complete than the HE prototype. The additional overhead of about 1.3 seconds is acceptable, especially for single tasks that happen on a daily basis. The HE prototype proved to be fast and CPU efficient. While the ZKP proof generation needed by far the most CPU resources and pushed the processor to its limits, it still fulfills ***R7***. The memory usage for both solutions was rather low, again making the solutions ideal for everyday devices such as smartphones and computers. However, the gas consumption of the HE prototype when sending the generated data on-chain is too high, thus leading to large monetary costs for both parties. In contrast to the ZKP prototype it does not satisfy ***R8***. However, it is still expensive to send the ZKP data via the blockchain. The evaluation therefore shows that the data should not be sent and stored on-chain and rather be transmitted via traditional protocols like HTTPS or DIDComm.

Compatibility and Usability

Both prototypes are implemented on the already existing SSI platform *CredChain*, which was built for the Ethereum blockchain. The solutions are coded in JavaScript, which supports an easy web integration across all platforms and browsers. Moreover, the ZKP and

HE libraries are open-source, therefore satisfying ***R9***. At last, the usability requirement ***R10*** is achieved by allowing the user to easily create a ZKP or execute a homomorphic calculation with one click of a button.

Additional Security Considerations

The two prototypes generate, store and transmit data that can be intercepted. Since ***R1*** and ***R3*** are fulfilled, there is no need to store the data in an encrypted database as proposed in ***SC3***, because the generated data already is encrypted or if intercepted, can not be used on its own by the attacker. Furthermore, the usage of a VPN as recommended in ***SC4*** is not necessary because it would encrypt already encrypted data. In addition, if the encrypted data is intercepted, the attacker can not use it to his advantage, nor can he try to decrypt or learn something about the data.

Chapter 7

Final Considerations

This final chapter is divided into two parts. The first part summarizes and reflects on the work completed, while the second part outlines potential future research directions to further advance the findings and address the remaining challenges.

7.1 Summary

This thesis contributed to the classification debate of metadata in blockchain-based SSI and public keys as personal data. In conclusion, metadata is classified as non-personal data, as it consists of data that does not identify a natural person. Furthermore, metadata is essential to verify the integrity and credibility of an issued credential in a SSI. Therefore they should not be deleted. Additionally, the GDPR only requires that personal data must be deletable from a system. In SSIs, metadata is stored on the user's personal devices and therefore the user has full control of his data and can, if really wanted, delete the metadata. Public keys are also considered non-personal data when generated and only turn into personal data as soon as they can be linked to a real-world identity i.e. through KYC or AML procedures from centralised exchanges. The same principle applies to metadata when it is associated with a real-world identity, rendering it personal data the moment the identification occurs.

The literature review analysed different forms of attacks on metadata in SSIs. In conclusion, the attacks do not directly target the metadata but rather rely on bad user behaviour such as rooting or jailbraking their device, on which the SSI application runs. The attacks focus on stealing or altering credentials but not specifically on metadata. This thesis lists security considerations that SSI applications should implement to prevent these dangers and also specifies the necessary requirements for metadata privacy-preserving mechanisms.

To enhance metadata privacy, this thesis proposed two prototypes that can be integrated into a SSI system. The ZKP and the HE solution ensure that the credential-verifying party does not know anything about the underlying metadata. Furthermore, the solutions demonstrated that the transmitted data is safe from interception during transmission. The attacker can also not learn anything about the underlying data, since the transmission

data is encrypted by default through the ZKP and HE mechanisms. The evaluation has further shown, that although both solutions are computationally intensive, they are still scalable in terms of computational resources and ideal for daily usage on devices such as smartphones or personal computers. Additionally, the evaluation of gas usage revealed that the implementations, particularly the HE prototype, are not economically viable for sending and storing the generated data on the blockchain.

7.2 Future Work

The two implemented privacy-preserving mechanisms are prototypes and still have to be implemented and tested for real-world usage. In order for both implementations to fully utilize the blockchain, the cost of sending and especially storing data on the blockchain must decrease. Therefore, it would be of interest to explore and develop other privacy-preserving solutions that generate less data which has to be sent, as the on-chain transmission and storage costs are still very high, even with a low base network fee.

At the moment, FHE is still in its early stages and requires further development to complete complex computations within a reasonable timeframe. The implemented HE prototype is fast, because only a single mathematical operation is being executed. However, when increasing the security parameters, the calculation time will rise exponentially. Likewise, this thesis did not evaluate the performance of different security parameter settings.

Lastly, there is a crucial need in clarification from GDPR regulators concerning the treatment of metadata and data on the blockchain. As blockchain technology continues to evolve and integrate with various industries, the ambiguity surrounding how metadata such as inter alia, public keys and timestamps should be classified under GDPR becomes increasingly problematic. Future work by GDPR authorities should focus on providing clear guidelines on whether metadata should be considered personal data and, if so, how it should be managed in a blockchain context, where data is inherently immutable.

Bibliography

[1] Abbas Acar et al. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation". In: *ACM Comput. Surv.* 51.4 (2018).

[2] Andreas M Antonopoulos and David A Harding. *Mastering bitcoin*. 2023.

[3] *Article 16 GDPR - Right to rectification*. 2024. URL: https://gdpr-info.eu/art-16-gdpr/.

[4] *Article 17 GDPR - Right to erasure ('right to be forgotten')*. 2024. URL: https://gdpr-info.eu/art-17-gdpr/.

[5] *Article 22 GDPR - Automated individual decision-making, including profiling*. 2024. URL: https://gdpr-info.eu/art-22-gdpr/.

[6] *Article 25 GDPR - Data protection by design and by default*. 2024. URL: https://gdpr-info.eu/art-25-gdpr/.

[7] *Article 4 GDPR - Definitions*. https://gdpr-info.eu/art-4-gdpr/. 2024.

[8] *Article 5 GDPR - Principles relating to processing of personal data*. 2024. URL: https://gdpr-info.eu/art-5-gdpr/.

[9] *Article 6 GDPR - Lawfulness of processing*. 2024. URL: https://gdpr-info.eu/art-6-gdpr/.

[10] Rahime Belen-Saglam et al. "A systematic literature review of the tension between the GDPR and public blockchain systems". In: *Blockchain: Research and Applications* 4.2 (2023), p. 100129.

[11] Daniel R. L. Brown. *SEC 2: Recommended Elliptic Curve Domain Parameters*. Version 2.0. 2010. URL: https://www.secg.org/sec2-v2.pdf.

[12] Špela Cucko et al. "Towards the Classification of Self-Sovereign Identity Properties". In: *IEEE Access* 10 (2022), pp. 88306–88329.

[13] Damiano Di Francesco Maesa et al. "Self sovereign and blockchain based access control: Supporting attributes privacy with zero knowledge". In: *Journal of Network and Computer Applications* 212 (2023), p. 103577.

[14] Diogo Duarte. "An introduction to Blockchain technology from a legal perspective and its tensions with the GDPR". In: *Cyberlaw Journal of the Cyberlaw Research Centre of the University of Lisbon School of Law-CIJIC* (2019).

[15] Jacob Eberhardt and the ZoKrates team. *ZoKrates - a toolbox for zkSNARKs on Ethereum*. https://github.com/Zokrates/ZoKrates/tree/develop/zokrates_js. 2024.

[16] N. Eichler et al. *Blockchain, Data Protection, and the GDPR*. Technical Report VR 36105 B 27/661/52176. German Blockchain Association (Bundesblock), 2018.

[17] Chun-I Fan et al. “Secure hierarchical bitcoin wallet scheme against privilege escalation attacks”. In: *International Journal of Information Security* 19 (2020), pp. 245–255.

[18] Michèle Finck. “Blockchains and data protection in the European Union”. In: *Eur. Data Prot. L. Rev.* 4 (2018), p. 17.

[19] Michèle Finck and Frank Pallas. “They who must not be identified—distinguishing personal from non-personal data under the GDPR”. In: *International Data Privacy Law* 10.1 (Mar. 2020), pp. 11–36.

[20] Ethereum Foundation. *Gas and Fees*. 2024. URL: `https://ethereum.org/en/developers/docs/gas/`.

[21] Nomic Foundation. *Hardhat: Flexible. Extensible. Fast. Ethereum development environment for professionals*. Mar. 2020. URL: `https://hardhat.org/`.

[22] Sovrin Foundation. *Sovrin Glossary V3*. Dec. 2019. URL: `https://docs.google.com/document/d/1gfIz5TT0cNp2kxGMLFXr19x1uoZsruUe_0glHst2fZ8/`.

[23] Michael Froehlich, Philipp Hulm, and Florian Alt. “Under pressure. A user-centered threat model for cryptocurrency owners”. In: *Proceedings of the 2021 4th International Conference on Blockchain Technology and Applications*. 2021, pp. 39–50.

[24] Andreas Grüner et al. “Analyzing and comparing the security of self-sovereign identity management systems through threat modeling”. In: *International journal of information security* 22.5 (2023), pp. 1231–1248.

[25] Information Commissioner’s Office. *Lawfulness, Fairness and Transparency - A Guide to the Data Protection Principles*. 2024. URL: `https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/data-protection-principles/a-guide-to-the-data-protection-principles/the-principles/lawfulness-fairness-and-transparency/`.

[26] Farzaneh Karegar et al. “Opportunities and Challenges of CREDENTIAL”. In: *Privacy and Identity Management. Facing up to Next Steps: 11th IFIP WG 9.2, 9.5, 9.6/11.7, 11.4, 11.6/SIG 9.2.2 International Summer School, Karlstad, Sweden, August 21-26, 2016, Revised Selected Papers*. Ed. by Anja Lehmann et al. 2016, pp. 76–91.

[27] Nataliia Kulabukhova. “Self-sovereign identity as trusted root in knowledge based systems”. In: *Computational Science and Its Applications–ICCSA 2020: 20th International Conference, Cagliari, Italy, July 1–4, 2020, Proceedings, Part III 20*. Springer. 2020, pp. 14–24.

[28] Anhtuan Le, Gregory Epiphaniou, and Carsten Maple. “A comparative cyber risk analysis between federated and self-sovereign identity management systems”. In: *Data & Policy* 5 (2023), p. 38.

[29] Jorge Lesmes and the Microsoft SEAL team. *node-SEAL - A JavaScript library for homomorphic encryption*. `https://github.com/s0l0ist/node-seal`. 2024.

[30] Yue Liu et al. “Design Pattern as a Service for Blockchain-Based Self-Sovereign Identity”. In: *IEEE Software* 37.5 (2020), pp. 30–36.

[31] Duc Anh Luong and Jong Hwan Park. “Privacy-preserving blockchain-based healthcare system for IoT devices using zk-SNARK”. In: *IEEE Access* 10 (2022), pp. 55739–55752.

[32] Ricardo Martins Gonçalves, Miguel Mira da Silva, and Paulo Rupino da Cunha. “Implementing GDPR-Compliant Surveys Using Blockchain”. In: *Future Internet* 15.4 (2023), p. 143.

[33] *Microsoft SEAL (release 4.1)*. `https://github.com/Microsoft/SEAL`. Jan. 2023.
[34] Cristina Vilchez Moya et al. "Implementation and Security Test of Zero-Knowledge Protocols on SSI Blockchain". In: *Applied Sciences* 13.9 (2023).
[35] Rahma Mukta et al. "Blockchain-Based Verifiable Credential Sharing with Selective Disclosure". In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020, pp. 959–966.
[36] Kundan Munjal and Rekha Bhatia. "A systematic review of homomorphic encryption and its contributions in healthcare industry". In: *Complex & Intelligent Systems* 9.4 (2023), pp. 3759–3786.
[37] Nitin Naik, Paul Grace, and Paul Jenkins. "An Attack Tree Based Risk Analysis Method for Investigating Attacks and Facilitating Their Mitigations in Self-Sovereign Identity". In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2021, pp. 1–8.
[38] Nitin Naik et al. "An evaluation of potential attack surfaces based on attack tree modelling and risk matrix applied to self-sovereign identity". In: *Computers & Security* 120 (2022), p. 102808.
[39] Satoshi Nakamoto. "Bitcoin: a peer-to-peer electronic cash system". In: (2008).
[40] Martin Neil. *CredChain: Take control of your own digital identity... and keep that valuable Bitcoin password safe*. Mar. 2021. URL: `https://www.unsw.edu.au/newsroom/news/2021/03/credchain--take-control-of-your-own-digital-identity-and-keep-th`.
[41] Node.js. *Performance Hooks API Documentation*. 2024. URL: `https://nodejs.org/api/perf_hooks.html`.
[42] Daniela Pöhn, Michael Grabatin, and Wolfgang Hommel. "Analyzing the Threats to Blockchain-Based Self-Sovereign Identities by Conducting a Literature Survey". In: *Applied Sciences* 14.1 (2023), p. 139.
[43] Kaledio Potter, Dylan Stilinski, and Selorm Adablanu. *Homomorphic Encryption for Secure Cloud Computing*. Tech. rep. EasyChair, 2024.
[44] Francesco Rampone. "Data protection in the Blockchain environment: GDPR is not a hurdle to permissionless DLT solutions". In: *Ciberspazio e diritto* 19.61 (2018), pp. 457–20.
[45] *Recital 26 of GDPR - Not Applicable to Anonymous Data*. `https://gdpr-info.eu/recitals/no-26/`. 2023.
[46] s0l0ist. *node-SEAL Documentation*. 2024. URL: `https://s0l0ist.github.io/node-seal/`.
[47] Dominik Schmelz et al. "Towards Using Public Blockchain in Information-Centric Networks: Challenges Imposed by the European Union's General Data Protection Regulation". In: *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*. 2018, pp. 223–228.
[48] Claus-Peter Schnorr. "Efficient signature generation by smart cards". In: *Journal of cryptology* 4 (1991), pp. 161–174.
[49] Daria Schumm. *credchain*. 2024. URL: `https://github.com/schummd/credchain`.
[50] Rakesh Shrestha and Shiho Kim. "Integration of IoT with blockchain and homomorphic encryption: Challenging issues and opportunities". In: *Advances in computers*. Vol. 115. 2019, pp. 293–331.
[51] Manu Sporny et al. *Decentralized Identifiers (DIDs) v1.0*. World Wide Web Consortium, W3C Recommendation. July 2022.

[52] Quinten Stokkink and Johan Pouwelse. "Deployment of a Blockchain-Based Self-Sovereign Identity". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 1336–1342.

[53] Quinten Stokkink et al. "A Truly Self-Sovereign Identity System". In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. 2021, pp. 1–8.

[54] Sofia Terzi et al. "Securing Emission Data of Smart Vehicles with Blockchain and Self-Sovereign Identities". In: *2020 IEEE International Conference on Blockchain (Blockchain)*. 2020, pp. 462–469.

[55] Cedric Von Rauscher. *credchain-privacy*. 2024. URL: `https://github.com/cedric-vr/credchain-privacy`.

[56] Kai Wagner et al. "Self-sovereign identity: A position paper on blockchain enabled identity and the road ahead". In: *Blockchain Bundesverband* (2018), pp. 1–56.

[57] Gavin Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. 2024. URL: `https://ethereum.github.io/yellowpaper/paper.pdf`.

[58] Jie Xu et al. "An Identity Management and Authentication Scheme Based on Redactable Blockchain for Mobile Networks". In: *IEEE Transactions on Vehicular Technology* 69.6 (2020), pp. 6688–6698.

[59] Wenbo Yang et al. "Security analysis of third-party in-app payment in mobile applications". In: *Journal of Information Security and Applications* 48 (2019), p. 102358.

[60] Hakan Yildiz et al. "Toward Interoperable Self-Sovereign Identities". In: *IEEE Access* 11 (2023), pp. 114080–114116.

[61] Zibin Zheng et al. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends". In: *2017 IEEE International Congress on Big Data (BigData Congress)*. 2017, pp. 557–564.

Abbreviations

BFV	Brakerski/Fan-Vercauteren
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DID	Decentralized Identifier
DIDComm	Decentralized Identifier Communication
ECDSA	Elliptic Curve Digital Signature Algorithm
ETH	Ether (Cryptocurrency)
FHE	Fully Homomorphic Encryption
GDPR	General Data Protection Regulation
HD	Hierarchical Deterministic
HE	Homomorphic Encryption
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
IPFS	InterPlanetary File System
JSON	JavaScript Object Notation
KB	Kilobyte
KYC	Know-Your-Customer
MB	Megabyte
PHE	Partially Homomorphic Encryption
PIN	Personal Identification Number
RTBF	Right to be Forgotten
RTR	Right to Rectification
SSI	Self-Sovereign Identity
SHE	Somewhat Homomorphic Encryption
TCP	Transmission Control Protocol
USD	United States Dollar
VC	Verifiable Credential
VPN	Virtual Private Network
ZKP	Zero-Knowledge Proof
zk-SNARK	Zero-Knowledge Succinct Non-Interactive Argument of Knowledge

List of Figures

List of Tables

Listings

Appendix A

Repository on GitHub

The GitHub repository containing the implementation of the two metadata privacy-preserving prototypes can be found under the following URL:

`https://github.com/cedric-vr/credchain-privacy`

The repository contains:

- **HE prototype**
- **ZKP prototype**
- **CPU, duration and memory measurement functions**
- **Gas estimation function**
- **Data plot generator**

A.1 Installation

The following installation and operation guide can also be found directly on the GitHub page. First, we need to compile and deploy smart contracts to the testnet. To do this, make sure all dependencies are installed in your environment or install them using:

```
1 npm install
```

To deploy, start the hardhat node with:

```
1 npx hardhat node
```

Open another terminal window and deploy the contracts:

```
1 npx hardhat run --network localhost scripts/deploy.js
```

A.2 Operation

This repository includes two metadata privacy-preserving prototypes to calculate and verify that a university degree has been issued after a specific date (Unix timestamp). Run the commands in the upmost folder of the repository.

Zero-Knowledge Proof (ZKP)
Run the following command to test the ZKP prototype:

```
1 npx hardhat test test/testZKP.js
```

Homomorphic Encryption (HE)
Run the following command to test the HE prototype:

```
1 npx hardhat test test/testHE.js
```

Evaluation

After running the above two privacy-preserving mechanism tests, you can run the following commands to evaluate the performance.

Gas Estimation

This outputs the estimated gas cost of sending and storing the files which were generated during the ZKP and HE process. By default, an ETH price of 2600 USD and a network fee of 4 Gwei is used.

```
1 node utilities/gasEstimator.js
```

Use flags to modify one or all of the parameters for the gas estimation calculation. To calculate the gas cost of storing the data on the blockchain, append the `-storeOnChain` flag.

```
1 node utilities/gasEstimator.js --ethPrice=3200 --gasPriceGwei=40 --
    storeOnChain
```

Duration, CPU- and RAM-Usage

Evaluate the benchmarks over a specified number of function runs. By default, it will run 50 times for the ZKP and the HE functions.

```
1 node evaluation/generateBenchmarks.js
```

Enter any integer to specify the number of runs for both prototypes.

```
1 node evaluation/generateBenchmarks.js 100
```

Plotting Data

This project uses Python 3.11.9 to plot data. It is recommended to use Python version 3.11 or newer to avoid any compatibility issues. Make sure that the libraries `matplotlib`, `pandas`, and `numpy` are installed. Use the following command to plot the generated benchmark data. The plots are saved as .PNG files in the `evaluation` folder.

```
1 python evaluation/generateGraphs.py
```