University of Zurich[UZH]

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

Design and implementation of efficient credential revocation system

Damjan Kuzmanovic
Zurich, Switzerland
Student ID: 20-743-472

Supervisor: Daria Schumm, Thomas Grübl, Prof. Dr. Burkhard Stiller
Date of Submission: 05.31,2025

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

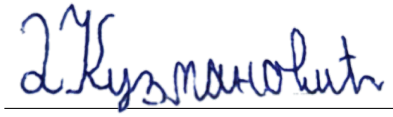
ifi

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 All passages taken verbatim or in sense from published or unpublished writings are identified as such.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, 31.05.2025



Signature of student

Abstract

Im Bereich der dezentralen Identitätsmanagementsysteme bleibt die Gewährleistung einer sicheren, skalierbaren und effizienten Sperrung eine entscheidende Herausforderung. In den letzten Jahren haben mehrere Ansätze, die kryptografische Akkumulatoren nutzen, grosses Potenzial bei der Bewältigung dieses Problems gezeigt. Sie sind nämlich in der Lage, für Sperrungsszenarien Nicht-Mitgliedschaftsnachweise mit konstanter Größe bereitzustellen. Viele Lösungen leiden jedoch unter einer kritischen Leistungsbeschränkung, die den Blockchain-basierten Systemen innewohnt. Diese Beschränkung äußert sich in der Notwendigkeit, umfangreiche Verifizierungsschleifen durchzuführen, wenn die auf der Blockchain gespeicherten akkumulatorbasierten Strukturen durchsucht werden. Dieses Problem wird besonders deutlich, wenn zwischen der Ausstellung von Berechtigungsnachweisen und den Verifizierungsversuchen ein erheblicher zeitlicher Unterschied besteht. Diese Arbeit versucht, ein effizientes System zur Sperrung von Berechtigungsnachweisen zu entwickeln und zu implementieren, das Verifizierungsschleifen umgeht und gleichzeitig die kryptografischen Sicherheitsgarantien aufrechterhält. Das zentrale Ziel besteht darin, die aufwändigen Blockchain-basierten Suchvorgänge in effizientere Suchprozesse umzuwandeln und damit den Verifizierungsprozess zu optimieren. Dies wird durch Hinzufügen eines nicht-invasiven SQLite-Moduls zur bereits vorhandenen akkumulatorbasierten Lösung erreicht. Aufgrund der Art der Implementierung und der effizienten Nutzung des Dekorator-Musters innerhalb der Lösungsarchitektur wurden keine Funktionen der ursprünglichen Architektur beeinträchtigt. Dieser Ansatz ermöglicht einen nahtlosen Wechsel zwischen der ursprünglichen und der verbesserten Architektur, sodass das SQL-Modul eine einfache und zusätzliche Verbesserung darstellt, die bei Bedarf entfernt werden kann. Eine umfassende Bewertung über fünf Testsuiten mit über 27.000 Operationen zeigt die erfolgreiche Umgehung von Verifizierungsschleifen. Während einzelne Operationen einen Rechenaufwand von 105 % verursachen, erzielt das System eine Verbesserung von 97 % bei komplexen Operationen und beseitigt zeitliche Skalierungsprobleme. Die Verifizierungsleistung bleibt unabhängig von der Zeit, die zwischen der Ausstellung und der Verifizierung von Berechtigungsnachweisen vergeht, stabil, wobei Operationen über mehrere Epochen hinweg nur minimale Leistungsschwankungen aufweisen.

Within the field of decentralized identity managements systems, ensuring secure, scalable and efficient revocation remains a crucial challenge. In recent years, several approaches utilizing cryptographic accumulators have shown great potential in tackling this issue. Namely, they are able to provide constant-size non-membership proofs for revocation scenarios. However, many solutions suffer from a critical performance limitation that is inherent to the blockchain-based systems. This limitation manifests itself in the necessity to preform extensive verification loops when searching through the accumulator-based structures stored on blockchain. The issue is especially exaggerated when there is a significant temporal difference between credential issuance and verification attempts. This thesis attempts to create and implement an efficient credential revocation system that circumvents verification loops while preserving cryptographic security guarantees. The central objective is to transform the expensive blockchain-based searches into more efficient lookup processes, in turn optimizing the verification process. This is achieved by the addition of a non-invasive SQLite-module to the preexisting accumulator-based solution. Due to the nature of the implementation and the efficient use of decorator pattern within the solution architecture, none of the capabilities of the original architecture were compromised. This approach allows for seamless transitions between the original and enhanced architecture, making the SQL-module simple additional improvement that can be removed if needed. A comprehensive evaluation across five test suites encompassing over 27,000 operations demonstrates successful circumvention of verification loops. While individual operations incur 105% computational overhead, the system achieves 97% improvement in complex operations and eliminates temporal scaling issues. Verification performance remains consistent regardless of time elapsed between credential issuance and verification, with cross-epoch operations showing minimal performance variation.

Acknowledgments

Contents

Declaration of Independence i

Abstract iii

Acknowledgments v

1 Introduction 1

1.1 Motivation 2

1.2 Research Approach and Methodology 3

1.3 Thesis Contributions 4

1.4 Thesis Structure 5

2 Fundamentals 7

2.1 Background 7

2.1.1 Accumulators 8

2.1.2 Bloom Filters 10

2.1.3 Revocation system overview 11

2.1.4 Blockchain Search Mechanisms 13

2.1.5 SQLite 13

2.2 Related Work 15

2.2.1 A blockchain-based certificate revocation management and status verification system 16

2.2.2 Cryptorevocate: A Cryptographic Accumulator-based Distributed Certificate Revocation List 17

2.2.3 VQL: Providing Query Efficiency and Data Authenticity in Blockchain Systems . . . 17

2.2.4 Full-text Search for Verifiable Credential Metadata on Distributed Ledgers . . . 18

2.2.5 Enabling SQL-Query Processing for Ethereum-based Blockchain Systems . . . 18

2.2.6 Blockchain search engine: Its current research status and future prospect in IoT network . . . 19

3 Design **21**

3.1 Design Rationale and Strategic Approach . . . 22

3.1.1 Core Design Philosophy . . . 22

3.1.2 Technology Selection and Integration Pattern Rationale . . . 23

3.2 Solution Design . . . 23

3.2.1 System Layers and Architectural Overview . . . 23

3.2.2 Interactions and Modularity . . . 25

3.2.3 Summary and Rationale . . . 25

3.3 Major components and their application . . . 26

3.3.1 SQLite Metadata Query Engine . . . 26

3.3.2 Application Interface Layer . . . 27

3.4 Revocation and Verification Workflows overview . . . 28

3.4.1 Revocation Workflow . . . 28

3.4.2 Verification Workflow . . . 29

3.5 Security and Privacy Guarantees . . . 30

3.5.1 Performance Trade-off Framework . . . 31

4 Implementation **33**

4.1 Database Layer and Schema Design 34

4.2 Revocation Service Logic 35

4.2.1 Individual Revocation Processing 36

4.2.2 Batch Processing and Performance Management 37

4.3 Verification Service Logic 40

4.3.1 Verification Design and Implementation 40

4.3.2 Performance Optimization and System Integration 42

4.4 Integration Architecture and System Coordination 44

4.5 Interfaces and Flow Orchestration 49

4.6 Observations and Reflections 50

4.7 Summary 52

5 Evaluation **53**

5.1 Introduction and Evaluation Methodology 53

5.1.1 Evaluation Framework and Test Design 54

5.1.2 Test Environment and System Specifications 55

5.1.3 Statistical Methodology and Data Collection 55

5.1.4 Test Scenario Coverage 56

5.2 Performance Analysis 56

5.2.1 Individual Credential Operations 56

5.2.2 Epoch Transition Performance 57

5.2.3 Cross-Epoch Verification Performance 57

5.2.4 Verification Pipeline Optimization 58

5.2.5 Performance Predictability Analysis 58

5.2.6 System Load and Scalability Indicators 59

5.2.7 Performance Analysis Summary 59

5.3 Scalability Assessment 59

5.3.1 Medium-Scale Performance (sqlTest3) . . . 60
5.3.2 High-Volume Stress Testing (sqlTest4) . . . 61
5.3.3 Temporal Scaling . . . 62
5.3.4 Scalability Assessment Summary . . . 63
5.4 Cost Analysis . . . 63
5.4.1 Computational Cost Reduction . . . 64
5.4.2 Storage and Memory Overhead . . . 65
5.4.3 Integration Complexity . . . 66
5.4.4 Cost Analysis Summary . . . 67
5.5 Security Analysis . . . 67
5.5.1 Security Guarantee Preservation . . . 68
5.5.2 System Resilience . . . 68
5.5.3 Security Analysis Summary . . . 70
5.6 Comparative Analysis with Schumm et al. [22] . . . 71
5.6.1 Core Problem Resolution . . . 71
5.6.2 Quantitative Improvements Summary . . . 72
5.6.3 Comparative Analysis Summary . . . 75
5.7 Summary . . . 75

6 Final Considerations **77**
6.1 Summary . . . 77
6.2 Conclusions . . . 78
6.3 Future Work . . . 79

Bibliography **80**

Abbreviations **83**

List of Figures **83**

List of Tables **85**

List of Listings **87**

A Contents of the Repository **91**

Chapter 1

Introduction

Efficient credential revocation is one of the central challenges in modern decentralized identity systems. Although solutions relying on cryptographic accumulators offer scalable and privacy-preserving revocation capabilities, existing implementations suffer from performance limitations that prevent their practical deployment and widespread use. One of the core issues lies in the requirement for verifiers to perform extensive verification loops when searching through accumulator data structures, creating unpredictable performance metrics incompatible with real-world production environments [22].

Recent work done in Schumm et al. [22] introduced an elegant accumulator-based credential revocation architecture providing constant-size non-membership proofs. However, their system exhibits critical bottlenecks rooted in blockchain-based search mechanisms, where verification time varies significantly based on the temporal distance between credential issuance and verification attempts. This creates a linear relationship between the age of the credential and the verification cost that fundamentally undermines the scalability of the system. [22]

The solution presented in this thesis addresses these limitations through the design and implementation of an efficient credential revocation system that circumvents verification loops through utilization of an embedded SQLite database, all the while maintaining cryptographic security guarantees. Our approach provides alternative verification paths that avoid computationally expensive accumulator operations in common scenarios, achieving improvements in complex operational scenarios while accepting measured overhead in routine operations. The system bypasses the verification loops that characterize existing approaches through intelligent optimization layers that complement the original cryptographic constructions.

1.1 Motivation

The primary motivation for this research stems from a fundamental limitation in the blockchain-based search mechanisms employed by accumulator-based credential revocation systems. Current approaches enable credential verifiers to search through blockchain history for matching credentials, utilizing blockchain as time-ordered data records. However, this approach suffers from a critical bottleneck - verifiers could be required to perform a large number of verification loops to reach conclusions about the status of the credential. In addition, the verification time increases significantly as the time elapsed between the issuance of the credentials and the verification attempts grows. This issue creates unpredictable performance metrics that prevent the utilization of blockchain-based mechanisms in production environments where consistent response times are essential for system reliability. Our approach addresses these challenges through architectural bypass strategies rather than direct accumulator optimization.

The core issue lies in the inefficient search mechanisms employed through accumulator data structures. These mechanisms often lack optimization strategies to handle temporal complexity. Despite the efforts to reduce the computational burden on credential holders, the verification process remains expensive for verifiers who must traverse accumulator data structures sequentially. This creates a direct relationship between the age of the system and the verification cost, fundamentally reducing the scalability required for real-world deployment. While direct optimization of accumulator operations remains challenging, parallel optimization architectures can provide practical solutions that bypass the performance bottlenecks while maintaining the security guarantees of the original cryptographic constructions. The absence of efficient search mechanisms that operate independently of time elapsed between credential issuance and verification represents a critical barrier to practical deployment. Current systems require increasingly expensive verification operations as they accumulate historical data and inadvertently create an unsustainable performance model where older credentials become progressively more expensive to verify.

Beyond search inefficiency, existing systems lack support for efficient batch processing and structured metadata access, capabilities essential for organizational credential management. Organizations often require the ability to revoke credentials in bulk, analyze revocation patterns for compliance, and query credential metadata without experiencing performance penalties associated with inefficient accumulator traversal.

These limitations prevent accumulator-based credential revocation systems from achieving feasible deployment in real-world scenarios, despite their theoretical elegance and cryptographic properties. The fundamental need for efficient search mechanisms that can optimize traversal through accumulator data structures while maintaining time-independent performance metrics represents the critical challenge that must be addressed to bridge the gap between cryptographic soundness and practical viability.

This thesis addresses the aforementioned challenges in blockchain-based environments through a bypass optimization approach. It directly addresses the following research problem: **How can we design and implement efficient credential revocation that bypasses verification loops when searching through accumulator data structures while**

maintaining cryptographic security guarantees? The core technical challenge involves implementing parallel optimization architectures that provide alternative verification paths while preserving cryptographic soundness and mathematical equivalence with the original constructions.

1.2 Research Approach and Methodology

This thesis employs a systematic approach that combines theoretical analysis, architectural design, practical implementation, and empirical evaluation to address the challenges of efficient credential revocation in blockchain-based environments. The methodology begins with a comprehensive analysis of existing accumulator-based revocation systems to identify performance bottlenecks and limitations in blockchain-based search mechanisms, particularly focusing on the verification loop problem that prevents practical deployment.

The research approach involves iterative design and implementation cycles, focusing on developing optimization layers that circumvents verification loops while preserving cryptographic security guarantees of existing accumulator-based constructions. The design emphasizes noninvasive integration that allows existing systems to benefit from performance improvements without requiring fundamental architectural changes. The implementation methodology involves building on the cryptographic foundations established in Schumm et al. [22], while introducing SQL-based optimization mechanisms. The approach prioritizes maintaining mathematical equivalence with the original constructions while providing practical enhancements such as efficient batch processing and structured metadata access.

The implementation methodology evolved to address the verification loops challenge through architectural bypass rather than direct accumulator modification. This approach recognizes that while the task description emphasized optimizing search through accumulator data structures, practical solutions can be achieved through parallel structures that complement accumulator operations, providing performance benefits while maintaining full compatibility with the original cryptographic verification process.

The evaluation strategy employs a comprehensive performance analysis in multiple operational scenarios to validate the effectiveness of the proposed optimization approach. This includes systematic testing under varying load conditions, temporal scaling analysis, and rigorous security validation to ensure that performance enhancements preserve all cryptographic properties. The methodology emphasizes empirical validation through direct comparison with baseline systems, ensuring that improvements are measurable and reproducible while maintaining system correctness and security guarantees.

1.3 Thesis Contributions

This thesis makes several significant contributions to credential revocation in decentralized identity systems:

Primary Technical Contributions: Design and implementation of an efficient credential revocation system that bypasses verification loops through utilization of an embedded SQLite database, providing substantial performance improvements in complex operational scenarios while maintaining compatibility with existing accumulator-based architectures. The solution also involves a multi-stage verification pipeline that provides consistent performance metrics while maintaining all cryptographic security guarantees of the underlying accumulator-based architecture.

Performance and Scalability Contributions: Comprehensive empirical analysis demonstrating that accumulator-based credential revocation systems can achieve predictable performance metrics suitable for production deployment through strategic database integration, with substantial improvements in complex scenarios balanced against measured overhead in routine operations. The work establishes that such systems can achieve predictable performance metrics essential for consistency-sensitive environments, while introducing adaptive learning mechanisms and efficient batch processing capabilities that address the operational requirements of organizational credential management scenarios.

Practical Implementation Contributions: A comprehensive implementation approach that incorporates administrative interfaces, structured metadata query capabilities, and operational monitoring features relevant for real-world scenarios. The implementation demonstrates how parallel optimization architectures can integrate with existing cryptographic systems through compatible interfaces that preserve operational characteristics. The thesis explores how embedded database technologies can be integrated with cryptographic revocation systems while preserving decentralization properties, thereby contributing architectural insights and implementation patterns that inform future credential revocation system development.

1.4 Thesis Structure

This thesis systematically addresses the research problem through six chapters:

Chapter 1: Introduction establishes the research context by identifying limitations in blockchain-based search mechanisms and presents the core problem of circumventing verification loops when searching through accumulator-based data structures. It outlines the research approach and summarizes key contributions of the thesis.

Chapter 2: Fundamentals provides a theoretical foundation that includes decentralized identity systems, cryptographic accumulators, and blockchain-based credential revocation systems by examining the current state of research in this field. The chapter focuses especially on analyzing the existing revocation mechanisms present in the baseline architecture of Schumm et al. [22] and reviews related work to identify research gaps.

Chapter 3: Design presents the design of the revocation system architecture, with details on the modular design that allows for efficient search through the data structures of the accumulator. It further expands on how the SQL-based optimization layer integrates with cryptographic components in order to circumvent the verification loops challenge while maintaining security guarantees.

Chapter 4: Implementation describes the practical development of the system using Node.js and embedded SQLite database, covering the newly created revocation and verification services, the integration architecture, and the multi-stage verification pipeline. Implementation details include database schema design and testing framework development.

Chapter 5: Evaluation provides empirical analysis through systematic performance tests, demonstrating both the benefits and limitations of the bypass approach in addressing verification performance challenges. The evaluation establishes the scenarios where the optimization strategy provides value while acknowledging the trade-offs involved. The evaluation includes performance analysis, scalability assessment, cost analysis, and security discussion with comparative analysis against baseline systems.

Chapter 6: Summary offers concluding remarks with reflection on research contributions, analysis of limitations, and potential direction for future research. It synthesizes the research findings and their implications concerning credential revocation within decentralized identity systems.

The approach acknowledges that addressing complex cryptographic system limitations often requires strategic trade-offs between different performance metrics. While the solution introduces overhead in routine operations, it provides substantial computational cost reductions in scenarios that would otherwise be prohibitively expensive, thereby transforming unpredictable performance into measurable costs that may be acceptable for consistency-critical applications.

Chapter 2

Fundamentals

This chapter provides the foundational concepts and technologies that underpin the architecture and implementation of the credential revocation system developed in this thesis. It is divided into two main parts: the *Background*, which introduces the key cryptographic and system-level building blocks relevant to our work; and the *Related Work*, which contextualizes our approach in relation to existing academic and practical revocation solutions.

We begin by reviewing traditional and modern approaches to credential revocation, followed by an in-depth explanation of the cryptographic primitives, particularly accumulators and Bloom filters, used in our system. We also examine the key architectural design proposed by Schumm et al. [22], which serves as the core framework for our solution. Finally, we will discuss the importance of structured metadata querying and outline how our system integrates this capability.

2.1 Background

Credential revocation is the process by which a previously issued digital credential, such as a public–key certificate, identity token, or a verifiable credential, is officially invalidated prior to its scheduled expiration. This process plays a critical role in any identity management system, serving as a control mechanism to ensure that once a credential is compromised, misused, or otherwise rendered untrustworthy, it is no longer accepted as valid by any of the parties relying on it. Effective revocation is essential for preserving the security and trustworthiness of digital systems, particularly in decentralized contexts.

In traditional digital security systems, credential management is typically handled within a Public Key Infrastructure (PKI), a hierarchical framework that issues, validates, and revokes digital certificates using public-key cryptography [1]. PKIs rely on trusted certificate authorities (CAs) to enforce revocation policies and disseminate revocation data. Historically, revocation in centralized environments has been managed through mechanisms such as Certificate Revocation Lists (CRLs) and the Online Certificate Status Protocol (OCSP). Certificate Revocation Lists, as described in the IETF's RFC 5280

standard [7], remain one of the earliest mechanisms for conveying certificate invalidation. In the context of this thesis, CRLs are best described as signed lists published by certificate authorities that contain the serial numbers of all revoked certificates. While they provide an auditable and verifiable revocation model, they scale poorly: the list size grows linearly with the number of revoked certificates, leading to increased bandwidth consumption and increase in verification time [7]. These deficiencies lead to clients having to periodically download and parse large CRLs to remain current [6], which is inefficient and unreliable in distributed environments.

In contrast, the Online Certificate Status Protocol (OCSP) was designed to mitigate these limitations by allowing clients to query the OCSP responder of a CA in real time to check the revocation status of a specific certificate [7]. However, this approach introduces new challenges, such as dependency on the always available responder infrastructure and potential privacy leaks, since querying parties expose their browsing behavior to third parties [6]. These trade-offs have been extensively examined in academic literature and technical standards [1, 6, 7]. Although OCSP reduces the size of the data transferred during status checks, it introduces other challenges [6, 12]. In particular, OCSP requires constant availability of the responder service, creating a potential single point of failure. Furthermore, OCSP queries can introduce privacy concerns, since they can reveal which websites or services a user is attempting to access, thereby compromising anonymity.

However, these mechanisms were designed for centralized trust hierarchies and are ill-suited for the decentralized architectures gaining prominence today. In modern decentralized systems, such as those based on blockchain, decentralized identifiers (DIDs) [18], and self-sovereign identity (SSI) [21], traditional revocation models such as CRL and OCSP are no longer appropriate. These systems aim to eliminate reliance on central authorities, promote user autonomy, and support decentralized verification approaches [21]. As such, any revocation method that depends on an always-on infrastructure or centralized publication quickly becomes a limiting factor. Moreover, decentralized identity systems introduce unique challenges: Revocation proofs must be compact, bandwidth efficient and verifiable without constant network connectivity or centralized search [21]. As such, this shift toward decentralized identity necessitates revisiting and redesigning revocation mechanisms to meet new operational, security, and privacy demands.

2.1.1 Accumulators

To address the critical requirements of scalability, bandwidth efficiency, privacy preservation and decentralization in credential revocation, recent research has explored alternative cryptographic data structures [2, 16, 22]. Among these, approaches based on **cryptographic accumulators** have emerged as a particularly promising solutions [5, 9]. A **cryptographic accumulator** can be defined as a family of one-way hash functions that have the quasi-commutative property [9]. A quasi-commutative function ensures that the order of accumulation operations does not affect the final result. This property is essential for allowing revocation proofs to be generated in a modular, append-only fashion without reprocessing the entire dataset. For example, in an RSA accumulator, values are aggregated through exponentiation over a common modulus, with the quasi-commutative

property ensuring that the resulting value remains consistent regardless of the insertion order. That is, given an initial value A and two inputs x_1 and x_2, the function f satisfies the following:

$$f(f(A, x_1), x_2) = f(f(A, x_2), x_1)$$

This property allows for the representation of a dynamic set of values as a single fixed-size value[8, 9]. Importantly, it enables the generation of membership and non-membership proofs that can be verified using cryptographic operations, making it particularly useful for efficient and privacy-preserving revocation mechanisms. This approach has several key advantages:

- First, the proof size remains constant, regardless of the size of the revocation list [5].
- Second, verification of non-membership can be done locally and efficiently, often in under a few milliseconds [22].
- Third, the system supports *stateless verification* [16], which means that verifiers do not need access to the entire revocation list or continuous Internet access to perform status checks.
- Fourth, when integrated with zero-knowledge proofs, accumulator-based schemes can offer privacy guarantees suitable for anonymous credentials and self-sovereign identity applications. [5]

A very important aspect of the security of RSA-based cryptographic accumulators is that they rely on the Strong RSA assumption [19]. This widely accepted hardness assumption posits that, given an RSA modulus n and a random value $u \in \mathbb{Z}_n^*$, it is computationally difficult to find integers v and $e > 1$ such that $v^e \equiv u \mod N$. This property ensures the collision resistance and soundness of the accumulator function, meaning that once a value has been accumulated, it cannot be removed or replaced without the knowledge of the trapdoor information [19], which in most cases represents the secret prime factorization of the RSA modulus. As a result, all membership and non-membership proofs derived from the accumulator can be trusted to reflect the actual contents of the set, assuming the strong RSA problem remains intractable. This foundational assumption underlies both the static and dynamic accumulators employed within the solution presented in this thesis.

Building on top of this, a **dynamic accumulator**, as introduced by Camenisch and Lysyanskaya [5], is an extension of the classical cryptographic accumulator that supports not only membership proofs, but also the dynamic modification of the accumulated set. Specifically, it allows for the addition and deletion of elements from the set in such a way that the computational cost of these operations remains constant and independent of the total number of accumulated elements. At its core, a **dynamic accumulator** provides a succinct representation of a growing or shrinking set of values [5]. It enables users to efficiently generate a *membership witness*, a cryptographic proof that a specific value is included in the set, or, depending on the construction, a *non-membership witness* for

values not in the set. These witnesses can be verified using only the public accumulator value, without revealing the entire set or the need to query a third party. Dynamic accumulator constructions as proposed by Camenisch and Lysyanskaya in their paper [5] are based on the Strong RSA assumption and employ a one-way quasi-commutative function. The aforementioned protocol also includes a zero-knowledge proof system, enabling users to demonstrate knowledge of an accumulated value without revealing which value they possess, this being a crucial property for anonymous credential systems. This approach makes dynamic accumulators particularly useful in scenarios where users need to prove the validity of credentials without compromising their privacy, such as in SSI systems [21].

2.1.2 Bloom Filters

A **Bloom filter** is a simple, space-efficient randomized data structure based on hashing that represents a set in a way that allows membership queries to determine whether an element is a member of the set [14]. These properties make Bloom filters especially useful in systems that require fast lookups but suffer from memory limitations. Introduced by Burton Howard Bloom in 1970 [3], the Bloom filter allows a verifier to determine whether an element is possibly in a set or definitely not in the set. In other words, it allows for false positives but guarantees that there are no false negatives. In practice, this means that if the Bloom filter indicates that an element is *not* in the set, the result is always correct. However, if it indicates presence, there is a small chance of error. A Bloom filter consists of a fixed-size bit array of m bits, initially set to 0. To add an element, k independent hash functions are used to map the element to the k positions in the array. Then the bits in those positions are set to 1 [3, 14]. To verify membership, the same k hash-functions are applied to the queried element and the resulting k corresponding bits are checked. If all bits are set to 1, the element is *possibly* in the set; if any bit is 0, the element is *definitely not* in the set. The probability of a false positive in a Bloom filter is a function of the number of inserted elements n, the size of the bit array m, and the number of hash functions k. This trade-off can be adjusted to balance memory usage and query accuracy [14]. However, standard Bloom filters also come with an important limitation: they do not support deletion. Once an element is added to the filter, it cannot be removed without necessitating a reconstruction of the entire structure. The lack of deletion capability becomes especially problematic in revocation scenarios, where credentials may need to be invalidated dynamically or after expiration. Variants such as the *Deletable Bloom Filter* [20] and *Counting Bloom Filter* [4] have been proposed to address this issue by maintaining auxiliary data to track insertions and support safe deletions. However, these variants often introduce added memory overhead and structural complexity [4, 20], making them less suitable for wider use. The simplicity and compactness of the Bloom filters make them ideal for applications that require high-speed, approximate membership queries, such as cache lookups, blacklist filtering, and revocation pre-checks. In these contexts, the inability to delete elements is often acceptable, especially if the filter is refreshed periodically or used as a transient cache. In credential revocation architectures, Bloom filters are sometimes employed as efficient pre-filterers that allow verifiers to quickly eliminate obviously valid credentials before attempting expensive cryptographic verification. Their integration into cryptographic revocation systems is exemplified in a recent accumulator-based designs [22], which will be discussed in the following subsection. In

similar architectures, Bloom filters serve as performant first-pass filters, reducing computational overhead and enabling verifiers to operate in low-latency, bandwidth-constrained environments without sacrificing correctness.

2.1.3 Revocation system overview

The thesis uses the credential revocation system proposed by Schumm, Mukta, and Paik in their paper *Efficient Credential Revocation Using Cryptographic Accumulators* [22] as a strong foundation upon which to build our own solution. Their work proposes a revocation system that eliminates the dependency on witness updates for credential holders, achieving sub-linear scalability with respect to the number of revocations, and preserving user privacy by encoding credentials as prime numbers within cryptographic accumulators [22].It utilizes a modular and cryptographically sound architecture, using a layered approach centered on three core components:

- a *sub-accumulator* implemented as a Bloom filter,
- a *static cryptographic accumulator* an RSA-based accumulator,
- and a *dynamic accumulator*, tasked with maintaining a cumulative revocation history.

These components are organized around the notion of epochs, discrete time intervals during which credential revocations are collected, published, and verified.

At the front line of the architecture is the **sub-accumulator**, a space-efficient, probabilistic Bloom filter. This component is responsible for storing all the credential information for a specific epoch. Bloom filters are a particularly appropriate choice in this context for several reasons:

- First, they allow for fast, constant-time lookups with minimal memory overhead [3].
- Second, they support *witnessless* and *stateless* verification, meaning a verifier does not require any auxiliary data or updated witness information to check a credential [22].
- Third, Bloom filters inherently resist leakage of sensitive information [20], as they do not expose which items are stored within them - only whether a given input might be present.

These characteristics make the Bloom filters an excellent choice for use in mobile, and low-bandwidth environments where computational efficiency and privacy are paramount [22]. However, because Bloom filters suffer from false positives and do not support deletion [3, 20], they are used only within the scope of a single epoch and are periodically renewed.

Once the sub-accumulator reaches capacity, which defines the duration of one epoch, its contents are copied into the second component: the **static accumulator**. This component

is a traditional RSA-based cryptographic accumulator and acts as a cryptographically secure summary of all credential revocations that occurred during the epoch [22]. The information from the Bloom filter is mapped into prime values and accumulated into a single accumulator value using modular exponentiation over an RSA modulus. This static accumulator serves as an immutable snapshot of the revoked set for the completed epoch, enabling long-term verifiability even after the Bloom filter has been recreated [22]. Since it does not need to be updated after publication, the static accumulator avoids the overhead associated with witness maintenance and state synchronization, making it a robust choice for archiving revocation states.

To preserve the integrity and continuity of revocation across time, the system incorporates a third component: the **dynamic accumulator** [22]. Similar to the static accumulator, the dynamic is based on RSA cryptographic principles, but its purpose is cumulative rather than epoch-specific. Each time a static accumulator is finalized and published, it is itself added into the dynamic accumulator. In this way, the dynamic accumulator becomes a verifiable ledger of all prior revocation states and static accumulators. It allows external verifiers to confirm that any given epoch (represented by a static accumulator) has been properly included in the system's revocation history. This ensures tamper proof, traceability, and integrity across the system's entire operational timeline [22]. Their approach focuses primarily on the inefficiencies and usability issues associated with traditional accumulator-based credential revocation, particularly costly witness update operations, by proposing a framework that eliminates the need for such updates while maintaining cryptographic soundness [22].

While Schumm et al. [22]'s three-layer accumulator system successfully eliminates witness updates for credential holders, it introduces a significant verification bottleneck. The core problem lies in the required "verification loops" where verifiers must sequentially check each epoch between a credential's issuance and the current verification time. For each intermediate epoch, verifiers must retrieve the sub-accumulator, verify credential exclusion, obtain the static accumulator, and confirm its inclusion in the dynamic accumulator [22]. This creates a linear relationship between verification time and elapsed epochs, making the system impractical for long-lived credentials or high-frequency verification scenarios. The authors acknowledge that verification time depends on how many epochs passed from the credential issuance but focus primarily on credential holder efficiency rather than addressing verifier scalability challenges [22]. This represents a fundamental trade-off: reduced computational burden for credential holders comes at the cost of increasingly expensive verification processes that could limit real-world adoption.

2.1.4 Blockchain Search Mechanisms

Blockchain systems present fundamental challenges for data retrieval operations due to their append-only, immutable structure that prioritizes consensus mechanisms over query efficiency [23]. Despite their effectiveness in preventing data tampering and fraud, current blockchain implementations exhibit limited search capabilities [24]. The underlying computational challenge stems from blockchain's inherently sequential data organization: records are stored in strict chronological order, requiring participating nodes to traverse each record sequentially or utilize rudimentary block metadata when processing search requests [24]. Unlike conventional database management systems that employ sophisticated multi-dimensional indexing structures (e.g., B-trees, hash indexes), blockchain's cryptographically linked architecture necessitates linear block traversal for data retrieval operations. This sequential access pattern exhibits polynomial degradation in performance metrics as blockchain size increases, creating scalability bottlenecks for large-scale deployments [15]. The same cryptographic mechanisms that ensure data integrity and immutability simultaneously constrain query optimization strategies. This trade-off becomes particularly pronounced in applications that require real-time query responses.

Decentralized computational environments necessitate robust mechanisms to ensure both completeness (ensuring no valid results are omitted) and soundness (guaranteeing all results are authentic and unmodified) of query responses. When queries are processed by a potentially not trusted nodes in a peer-to-peer network, additional cryptographic verification protocols become essential to maintain system integrity [25]. These authentication requirements introduce additional computational overhead that further constrains query performance. Modern blockchain applications increasingly store heterogeneous data objects with multiple attributes, but existing search methodologies predominantly focus on single-attribute operations. Current blockchain systems require unified authenticated data structures capable of supporting complex query operators across multiple dimensions while maintaining cryptographic verification capabilities [25].

The verification loop phenomenon presents particular computational challenges for credential revocation systems. Verifiers must perform sequential temporal verification checks across multiple time periods. This sequential verification approach establishes a linear complexity relationship between verification time and credential lifespan, creating significant performance bottlenecks for long-lived credentials or high-frequency verification scenarios [22]. This challenge exemplifies the fundamental trade-off between system decentralization, cryptographic security guarantees, and computational performance efficiency that motivates the development of more sophisticated query mechanisms in blockchain-based credential revocation systems.

2.1.5 SQLite

As mentioned in the introductory chapter, the purpose of this thesis is to attempt to improve existing credential verification systems by trying to overcome the verification loops challenge inherent in blockchain-based systems. In the attempt to mitigate the the verification loop bottleneck challenge, this thesis integrates a lightweight relational

database engine **SQLite** within the revocation infrastructure. This approach aligns with the bypass optimization strategy outlined in Chapter 1, where parallel architectures provide alternative verification paths rather than modifying the fundamental cryptographic constructions. SQLite is an embedded SQL database engine that requires no separate server process and stores all its data in a single cross-platform disk file [11]. Designed for reliability, simplicity, and minimal resource usage, SQLite is used in billions of devices globally and is the database backend for numerous browsers, mobile applications, and embedded systems [11].

SQLite's capabilities for structured querying and indexed lookups present a potential solution to scalability challenges that are a consequence of verification loops . By maintaining query-able metadata alongside the cryptographic state, verification systems could theoretically reduce search complexity through targeted epoch filtering and intelligent query optimization. This approach would in turn provide alternative verification paths that achieve logarithmic lookup performance while maintaining compatibility with the original linear traversal operations. The theoretical foundations for this approach are well-established in database literature itself, where B-tree indexing structures enable $O(\log n)$ search complexity compared to sequential scanning performance $O(n)$ [11]. This mathematical advantage becomes particularly pronounced in credential systems where verification windows may span large time periods. One of SQLite's most attractive features is its zero-configuration model: no setup or administration is needed, and databases are portable and self-contained [11]. Beyond simple performance considerations, SQLite's design philosophy aligns closely with the requirements of decentralized credential systems. Its serverless architecture eliminates network dependencies and single points of failure, while its file-based storage model supports the distributed nature of modern identity infrastructures.

The database engine's proven stability—demonstrated through extensive testing including 100% branch coverage and deployment in safety-critical systems—provides the reliability necessary for credential verification scenarios where incorrect results could compromise security [11].These traits make it a suitable candidate for verifiers and decentralized agents that may need to operate in distributed environments where verifiers need local query capabilities without external dependencies. Moreover, SQLite supports full ACID transactions, SQL-92 compliance, and indexed lookups, enabling fast and consistent metadata queries without the overhead of a full-scale relational DBMS.

The integration of relational query capabilities alongside cryptographic revocation systems provides alternative verification paths that can bypass the performance bottlenecks documented in blockchain-based applications, where the append-only nature of distributed ledgers conflicts with the need for efficient data retrieval [24]. SQLite's mature query optimization engine, including support for partial indexes, expression indexes, and query planning, provides the necessary infrastructure to bridge this architectural tension.

In summary, SQLite is a flexible, embedded metadata store that complements the accumulator-based credential revocation model. Its lightweight footprint, strong transactional guarantees, and native SQL interface make it well-suited for decentralized identity platforms that require structured metadata querying alongside cryptographically verifiable state, making it a good choice for our solution.

2.2 Related Work

This section surveys previous work relevant to the development of our credential revocation framework, with particular emphasis on systems that use cryptographic accumulators for scalable revocation and those that explore efficient, verifiable querying in decentralized environments. The review is structured to highlight both the foundational approaches and emerging innovations that directly inform the design choices of this thesis. Special attention is paid to work published within the past decade, reflecting the recent evolution of decentralized identity systems, blockchain-based revocation, and lightweight verification strategies.

Our goal is twofold. First, we aim to understand the current landscape of accumulator-based revocation schemes, stateless proof mechanisms, and Bloom filter optimizations, examining how these approaches address centralized trust assumptions and verification overhead. Second, we seek to identify critical limitations and research gaps in existing work. These would include unresolved challenges that the current systems were unable to adequately address. Through this analysis, we will examine whether existing approaches effectively handle verification scalability bottlenecks and structured metadata querying requirements in decentralized credential systems.

To that end, we explore six carefully selected papers that can be divided into three main thematic categories. The first category examines accumulator-based and blockchain-integrated revocation models, including Adja et al. [2]'s Bloom filter approach and CryptoRevocate [16]'s cryptographic accumulator system. These demonstrate foundational approaches to decentralized credential revocation, but show limitations in temporal scalability. The second category investigates blockchain query middleware and data access frameworks, specifically VQL [17] and Han et al. [10]'s embedded SQL engine. Although these systems address general challenges in blockchain data access, they demonstrate significant gaps in cryptographic verification guarantees and domain-specific requirements for credential systems. The third category covers credential metadata search and blockchain search engine frameworks, including Lux et al. [13]'s full-text search system and Tang et al. [24]'s comprehensive survey of blockchain search mechanisms. These works highlight important querying capabilities, but reveal critical limitations in cryptographic integration and verifiable result authentication. Through systematic analysis of all of these papers, several recurring research gaps were identified:

- the inability to efficiently handle temporal verification across multiple revocation epochs,
- the absence of efficient metadata querying capabilities that complement cryptographically verifiable systems while providing practical operational benefits,
- the lack of modular architectures that can simultaneously support offline verification and complex administrative requirements.

These limitations collectively demonstrate the need for alternative approaches that can address cryptographic verification bottlenecks while maintaining practical usability requirements in decentralized credential revocation systems. This thesis explores one such

alternative through architectural coordination between embedded database technology and cryptographic accumulator systems.

2.2.1 A blockchain-based certificate revocation management and status verification system

Adja et al.[2] (2021) present a blockchain-based system that reimagines the management of certificate revocation and status verification within the traditional PKI ecosystem. Their approach leverages the decentralized and tamper-resistant properties of the blockchain, as well as Bloom filters' scalability properties. The motivation stems from the fundamental shortcomings of conventional PKI revocation mechanisms, CRLs and OCSP, including issues such as latency, privacy leaks, central failure points, and limited scalability [2].

Their proposal introduces a novel architecture that combines X.509 certificate extensions, Bloom filters, and the Namecoin blockchain to support decentralized revocation. The central idea is to assign each revoked certificate to a *distribution point*, represented by a Bloom filter, which is then embedded into blockchain transactions. These distribution points effectively partition the revocation space, and the Bloom filters enable clients to quickly check whether a certificate is possibly revoked. Importantly, the revocation status is verifiable in real time without reliance on traditional centralized OCSP responders. The metadata of each revoked certificate is encoded in an RSI (Revocation Status Information) structure, which includes a Bloom filter and a collection of metadata fields. The metadata includes important information about the credential such as the revocation date, certificate ID, and a digital signature of the CA. This RSI is posted on-chain as a Namecoin transaction. To address Bloom filter false positives, the authors also define a companion lightweight structure called LRSI (Lightweight Revocation Status Information), which omits the Bloom filter but retains metadata and signatures to allow follow-up verification. This tiered structure leads to a reduction in bandwidth consumption while maintaining correctness. Unlike many blockchain-based revocation solutions that suffer from delegation of revocation responsibility to users, this architecture centralizes control under the Certificate Authority while retaining decentralization in dissemination. Moreover, by extending the X.509 structure itself, the solution remains compatible with standard TLS handshake processes, making it easier to deploy without breaking the existing web infrastructure.

Despite these significant contributions, their proposed system reveals several critical limitations. The approach remains fundamentally anchored within traditional PKI paradigms and X.509 certificate semantics. The reliance on Bloom filters, while providing efficiency benefits, introduces inherent probabilistic verification limitations that prevent the system from providing cryptographically verifiable non-membership proofs [2]. The architecture can only offer probabilistic assertions about the presence of a certificate, which fails to meet the security requirements of systems that require mathematical certainty. When false positives occur, the system necessitates downloading and verifying multiple LRSI structures, potentially negating its inherent bandwidth advantages and introducing verification delays that scale with the number of revoked certificates.

2.2.2 Cryptorevocate: A Cryptographic Accumulator-based Distributed Certificate Revocation List

In their 2021 work *Cryptorevocate: A Cryptographic Accumulator-Based Distributed Certificate Revocation List* [16], Ozcelik and Skjellum propose a certificate revocation system that integrates asymmetric cryptographic accumulators with permissioned blockchain infrastructure. The system addresses traditional CRL and OCSP limitations by accumulating revoked certificate identifiers in a universal cryptographic accumulator and publishing accumulator values on a blockchain operated by trusted Certificate Authority miners. Clients receive non-membership proofs for their certificates, enabling local verification without network queries. The authors emphasize their exclusive use of asymmetric accumulators to eliminate false positives that could cause denial-of-service conditions for valid certificates. The permissioned blockchain design ensures high availability while maintaining lightweight consensus mechanisms for scalability.

The proposed system reveals several research gaps, despite these significant contributions . The architecture focuses exclusively on basic certificate revocation verification without addressing broader certificate lifecycle management requirements. The system operates within traditional PKI frameworks and does not explore applications beyond conventional certificate authority structures [16].

2.2.3 VQL: Providing Query Efficiency and Data Authenticity in Blockchain Systems

Peng et al. [17] propose the Verifiable Query Layer (VQL), a middleware framework that addresses fundamental inefficiencies in blockchain data querying and query result authentication. Their three-tiered architecture successfully tackles the linear traversal limitations inherent in blockchain's append-only structure by introducing a sophisticated data reorganization strategy. The system demonstrates notable architectural innovations through its dual-database approach, utilizing micro databases for high-frequency, short-term data snapshots and key databases for long-term aggregated records. The solution achieves significant performance improvements, showing 1.5 times higher throughput compared to native Ethereum clients across various query types. VQL's cryptographic fingerprinting mechanism represents a substantial advance in query result authentication, enabling independent verification through hash-anchored database fingerprints published on-chain. This approach effectively eliminates reliance on trusted query intermediaries while maintaining data integrity guarantees. The system's periodic consolidation strategy successfully balances low-latency querying requirements with long-term storage optimization, demonstrating practical scalability for large, evolving datasets.

Even though VQL has demonstrated significant contributions, the system reveals several critical limitations that constrain its applicability to specialized blockchain applications. The framework was unable to provide cryptographic non-membership proofs, limiting its verification capabilities to hash-based integrity checks rather than mathematical certainty about data absence. The system demonstrates no capacity for handling domain-specific cryptographic primitives beyond basic hash verification, particularly lacking support for

advanced verification protocols. VQL cannot provide offline verification capabilities, as its architecture maintains dependency on network connectivity for database synchronization and verification processes. These limitations highlight significant research gaps in developing query frameworks that can simultaneously support advanced cryptographic verification primitives and specialized domain requirements beyond general-purpose blockchain data access.

2.2.4 Full-text Search for Verifiable Credential Metadata on Distributed Ledgers

Lux et al. [13] present a hybrid search system for credential metadata that aims to improve discoverability in decentralized identity platforms, focusing on Self-Sovereign Identity (SSI) implementations such as Hyperledger Indy. Their architecture couples an off-chain Solr-based full-text search engine with a periodically synchronized ledger mirror, enabling keyword-based queries across credential definitions. Metadata such as schema names, issuers, and attribute examples are extracted and indexed to facilitate schema discovery and assistive querying for credential consumers. The system's primary contribution lies in its ability to support unstructured search over credential metadata that is otherwise hidden within raw transactions, addressing a usability gap in decentralized credential ecosystems (Section III-B, [13]).

While this model significantly enhances user interaction and schema transparency, it suffers from several architectural limitations. Chief among them is the lack of cryptographic anchoring: the metadata index is maintained off-chain and cannot be independently verified for consistency or completeness (Section IV, [13]). This introduces a trust dependency on the indexing service, which undermines the decentralization and tamper-resistance properties of the underlying ledger. The authors propose future enhancements, such as the integration of Merkle tree digests for metadata proofs, but these remain theoretical. Moreover, the system assumes reliable, synchronous access to ledger endpoints and does not support offline or bandwidth-constrained environments.

2.2.5 Enabling SQL-Query Processing for Ethereum-based Blockchain Systems

Han et al. [10] present a compelling framework for enabling SQL-query capabilities in Ethereum-based blockchain systems. Their solution successfully addresses the fundamental challenge of efficient structured data access in decentralized applications, where Ethereum's LevelDB storage lacks native support for range queries, filtering, and relational-style operations. The system demonstrates significant architectural innovations by embedding a lightweight SQLite engine directly within each Ethereum node, enabling local and structured queries over selected smart contract transaction data. Their dual-component design introduces the Register Manager for tracking smart contract data changes and the Query Manager for synchronizing relevant state changes into an embedded SQLite

database. The solution achieves remarkable performance improvements, demonstrating up to 22x faster query performance compared to native blockchain scanning while maintaining low overhead for write operations. The architecture successfully strikes a balance between blockchain integrity and structured data access by maintaining the original blockchain data in LevelDB for cryptographic consistency while using SQLite as a query-optimized replica for selected data subsets. This dual-storage model ensures that performance enhancements do not compromise trust, auditability, or decentralization, as all query operations remain local to each node without requiring external middleware or indexers.

Even though the proposed solution has contributed to the research ield, Han et al.'s system reveals limitations in its scope of application beyond general-purpose smart contract data querying. The framework focuses exclusively on basic SQL operations for transaction data retrieval without addressing broader blockchain application requirements. The system requires manual registration of smart contracts for tracking, limiting its applicability to scenarios where data structures can be predefined and managed through their registration mechanism.

2.2.6 Blockchain search engine: Its current research status and future prospect in IoT network

Tang et al. [24] provide a comprehensive and systematic survey of blockchain search engines, with particular focus on data-intensive applications such as Internet of Things (IoT) environments. Their work establishes a valuable taxonomic framework that classifies search mechanisms based on three key dimensions: index structures (distinguishing between off-chain and on-chain approaches), authentication methods (including Merkle proofs and cryptographic accumulators), and supported query semantics (spanning keyword, range, and aggregate queries) (Section 4, [24]). The authors effectively analyze hybrid architectural models such as BlockchainDB and FalconDB, demonstrating how these systems balance scalability concerns by offloading indexing operations to external storage while maintaining verifiability through cryptographic hash anchoring. Their survey methodology provides thorough coverage of existing approaches and establishes clear evaluation criteria for comparing different blockchain search solutions.

However, Tang et al. identify significant limitations that remain unaddressed in current blockchain search implementations. Most critically, existing search engines frequently rely on centralized components for indexing or synchronization operations, which fundamentally compromises the decentralized principles that blockchain systems aim to uphold (Section 5.2, [24]). Query result authentication presents another substantial gap, as many systems fail to guarantee result freshness or completeness, leaving them vulnerable to replay attacks and partial disclosure vulnerabilities. Privacy preservation represents a particularly underdeveloped area, with few existing solutions supporting differential access control mechanisms or encrypted indexing capabilities. Furthermore, most blockchain-native query systems remain limited to basic key-value access patterns and lack the expressive capabilities necessary for complex multi-attribute filtering or sophisticated query operations. These gaps highlight the need for research into verifiable and privacy-preserving

query models, modular local indexing approaches, and efficient synchronization protocols that can maintain decentralization while supporting complex search requirements.

Chapter 3

Design

In this chapter, we will present the architectural and design principles underlying the enhanced credential revocation system developed in this thesis. As stated before, our underlying cryptographic architecture comes from Schumm et al. [22]. Recognizing the strengths of this architecture, our work aims to enhance it, preserving its core three-layered cryptographic accumulator architecture, while proposing a practical extension. Although the original design effectively handles both the verification and revocation of credentials, it suffers from the inherent challenge of verification loops when dealing with credentials with issuance dates originating in the distant past. In order to mitigate this challenge, we propose a modular SQLite-backed query layer that operates alongside the cryptographic revocation engine.

The remainder of this chapter is structured as follows:

- The first section will explain the rationale behind the specific approach, including a discussion of practical constraints and deployment use cases.
- The second section will introduce and display the architecture of new system-enhanced architecture, outlining the integration of a metadata query engine with the underlying accumulator structure.
- The third section will provide a detailed breakdown of each system component including the Bloom filter sub-accumulator, static and dynamic accumulators, and the metadata query layer.
- The fourth section will propose end-to-end use scenarios, demonstrating credential issuance, revocation, verification, and query workflows.
- The final section will review the guarantees of our design, discuss trade-offs and limitations, and provide a summary and transition toward implementation.

3.1 Design Rationale and Strategic Approach

The conceptualization of our advanced credential revocation system is derived from a comprehensive analysis of the inherent limitations delineated in Chapter **Fundamentals** and the distinct research problem articulated in Chapter **Introduction**. Rather than attempting to optimize cryptographic operations themselves, our design adopts a strategic bypass optimization approach that transforms the blockchain-based search problem into an efficient database lookup process.

3.1.1 Core Design Philosophy

The design approach within the concept of this thesis recognized that the verification loop problem does not stem from inefficient cryptographic operations, but from the fundamental mismatch between blockchain's append-only structure and the need for efficient temporal queries. Therefore, the solution should attempt to simultaneously benefit from both SQL capabilities and accumulator-based cryptographic components. Analysis of related work revealed three critical gaps that inspired our architectural decisions:

- **Cryptographic Integration Gaps:** Systems like VQL [17] demonstrate sophisticated query capabilities but lack cryptographic verification guarantees. The enhanced design avoids this by maintaining the original cryptographic architecture as the authoritative source.

- **Temporal Scalability Limitations:** Existing blockchain-based search mechanisms suffer from the same temporal scaling issues identified in Schumm et al. [22]'s work. Our design targets this through performing SQL-based time-independent database lookups that eliminate the linear relationship between verification cost and system age.

- **Decentralization Requirements:** Analysis of middleware approaches revealed that centralized database dependencies compromise the distributed trust model essential for credential revocation systems. Systems requiring large external database servers or centralized query infrastructure introduce single points of failure and trust dependencies that conflict with decentralized identity principles. This drove our selection of an embedded database architecture that operates entirely within each system instance, preserving the distributed nature of Schumm et al. [22]'s original architecture while eliminating external infrastructure dependencies.

For our design it was instrumental to try and tackle all of the aforementioned issues with our new solution.

3.1.2 Technology Selection and Integration Pattern Rationale

SQLite selection reflects systematic evaluation against our three core design requirements: cryptographic integration compatibility, temporal scalability optimization, and decentralization preservation.

Database Technology Evaluation: Traditional databases (PostgreSQL, MySQL) were eliminated due to client-server architecture violating decentralization requirements. NoSQL alternatives (MongoDB, Redis) lacked either structured query capabilities or persistent storage needed for temporal optimization. Among embedded alternatives, RocksDB provided excellent performance but lacked SQL interfaces, while LMDB offered efficiency without structured query support. SQLite uniquely satisfied all requirements: embedded operation preserving decentralization, full SQL capabilities enabling indexed temporal queries, and proven compatibility with cryptographic systems.

Integration Pattern Selection: Direct system modification was rejected due to extensive re-validation requirements prohibiting production adoption. Middleware approaches were eliminated for introducing network dependencies conflicting with decentralization goals. The decorator pattern was selected because it enables non-invasive enhancement while maintaining complete compatibility with existing validation procedures, allowing organizations to adopt optimizations without invalidating security certifications or requiring system re-validation.

3.2 Solution Design

This section presents the high-level design of the credential revocation system developed in this thesis. As mentioned before, the proposed solution is intended to preserve the cryptographic security guarantees and verifiability properties of its underlying architecture while introducing several key enhancements that make the solution more usable, extensible, and compatible with real-world administrative workflows. The enhancements include: (i) a structured metadata query layer built atop SQLite, (ii) support for efficient batch operations during credential issuance and revocation, and (iii) a modular interface layer for REST and CLI integration. These additions transform the original architecture into a more complete solution suitable for decentralized identity (DID) and self-sovereign identity (SSI) ecosystems.

3.2.1 System Layers and Architectural Overview

The system is structured as a layered architecture, where each layer encapsulates a specific functional domain while interacting through well-defined interfaces. This modularization improves maintainability and scalability whilst facilitating future extensions. The four primary layers are:

1. **Credential Layer:** This layer represents all verifiable credentials along with their associated metadata. Each credential includes a unique identifier that will be hashed and processed by the cryptographic layer, as well as metadata fields such as issuer identification, issuance and expiration timestamps, credential type, and revocation status. This abstraction allows for structured queries and enables decentralized analysis of credential characteristics without having to directly interact with the cryptographic backend.

2. **Cryptographic Revocation Layer:** The center of the system, this layer is directly inherited and adapted from the design of Schumm et al. [22] and comprises three aforementioned cryptographic constructs:

   - A *Bloom filter sub-accumulator*, used during the current epoch for fast, probabilistic revocation checks. It provides stateless and low-cost filtering with acceptable false positive rates.
   - A *static RSA accumulator*, constructed at the end of each epoch. It contains a verifiable value of all revoked credentials within that time frame and supports constant-size non-membership proofs.
   - A *dynamic accumulator*, which includes successive static accumulators. This forms an append-only, cryptographically verifiable history of the entire revocation process across epochs.

   This layered approach achieves a balance between performance (via Bloom filter), verifiability (via static accumulator), and historical integrity (via dynamic accumulator).

3. **SQLite Query Layer:** Introduced in this thesis, this layer addresses the verification loop problem by providing an embedded SQLite database that improves credential verification through optimized metadata lookups. This layer transforms expensive cryptographic verification processes into fast database queries, eliminating the temporal scaling issues that plague accumulator-based systems. It enables use cases such as:

   - instant credential status verification without cryptographic operations,
   - efficient false positive detection and caching,
   - time-independent verification performance across epochs.

   The database operates within each system instance and is populated and updated in tandem with revocation events, preserving the decentralized nature of the architecture while providing the performance optimization capabilities that eliminate verification loops.

4. **Interface and Application Layer:** This top layer abstracts the complexity of the system and presents a unified interface to end-users and developers. It includes both a RESTful API and a command-line interface (CLI) to support:

   - credential issuance, revocation, and metadata registration,
   - verification of credential validity (via decentralized proofs),

- administrative metadata queries using SQL-style statements.

By offering these interfaces, the system is positioned for integration into real-world decentralized identity agents.

3.2.2 Interactions and Modularity

The modular architecture of the system promotes clear separation of concerns while ensuring seamless inter-layer communication through defined protocols and shared schemas. The interactions between components are:

- **Credential Issuance and Registration:** Upon issuance, a credential's metadata is stored in the local SQLite database. Its unique identifier is prepared (e.g., hashed and prime-mapped) for future inclusion into the revocation structures.
- **Revocation Flow:** When a credential is revoked, its identifier is added to the active Bloom filter. Its metadata entry is updated in the query layer to reflect the revocation time and reason. This provides cryptographic status validation and contextual metadata in parallel.
- **Epoch Finalization:** Once the Bloom filter reaches capacity, the accumulated identifiers are converted into their prime representatives and added to the static RSA accumulator. The resulting static accumulator is added to the dynamic accumulator to maintain long-term traceability. The associated metadata entries are also checkpointed in the SQLite database.
- **Local Verification:** Verifiers retrieve public accumulator value and use pre-generated non-membership proofs embedded within the credential to perform validation. Simultaneously, credential metadata can be queried locally from the database to support richer decision making.
- **Batch Operations:** To improve throughput, the new architecture supports batch revocation or issuance, allowing multiple credentials to be processed in a single call. This optimizes revocation latency and aligns with administrative needs for large-scale credential lifecycle management.

3.2.3 Summary and Rationale

This layered and modular design transforms a theoretically elegant revocation mechanism into a practical and extensible system for real-world identity deployments. By building directly on Schumm et al. [22]'s cryptographic core, we retain constant-size non-membership proofs, stateless, asynchronous, and decentralized verification and epoch-based accumulation with tamper-evident histories. Meanwhile, our extensions enhance the solution with structured local metadata querying using SQLite, batch support for operational efficiency, and user-friendly REST and CLI integration.

Together, these improvements close a critical usability gap in accumulator-based revocation systems. They offer administrative observability, enhanced performance, and verifiability without compromising the decentralized trust model. As such, the architecture presented here serves as a robust and extensible platform for managing credential revocation in the context of self-sovereign identity, zero-trust architectures, and privacy-sensitive identity systems.

3.3 Major components and their application

The credential revocation framework developed in this thesis is composed of several logically distinct yet interdependent components. Each subsystem is customized for specific performance, cryptographic, or usability requirements. Together, they form a layered architecture capable of providing scalable, verifiable, and privacy-preserving credential revocation. Given that RSA accumulators and Bloom Filters have been extensively covered in the previous chapters, this chapter will focus on components designed distinctly for the purposes of the solution and go into great detail to explain their underlying design.

3.3.1 SQLite Metadata Query Engine

A central addition introduced in our system is the embedded metadata query module, implemented using **SQLite**. This lightweight SQL database plays a pivotal role in enhancing the administrative and analytical capabilities of the original revocation infrastructure. Although the cryptographic components of the system ensure strong security guarantees and verifiability, they are intentionally minimalistic with respect to credential metadata. However, in real-world deployments, the ability to query and analyze credential metadata, such as which credentials have been revoked, when, by whom, and under what classification, is often essential for audit, compliance, and administrative workflows. The introduction of SQLite bridges this gap by enabling expressive low-latency queries that operate in tandem with the cryptographic revocation state.

SQLite was chosen for its performance, simplicity, and portability. As a serverless, zero-configuration engine that stores its entire database in a single file, SQLite aligns well with the decentralized environment of our solution. In our system, it operates entirely locally within each node or agent, eliminating the need for centralized data services and preserving the system's privacy and decentralization goals. The database schema includes fields for credential identifiers, issuers, types, issuance dates, revocation epochs, and status flags. This schema is kept synchronized with revocation events to ensure consistency. When a credential is issued, its metadata is inserted into the database. When it is revoked, the corresponding row is updated to reflect the new revocation state and epoch.

Crucially, the SQLite layer operates independently of the accumulator logic but complements it by tracking metadata that the accumulators themselves cannot store. This separation of concerns allows the revocation logic to remain lean and verifiable while enabling rich metadata-driven interactions elsewhere in the stack. The database supports

full SQL queries, allowing administrators to, for example, retrieve all revoked credentials issued by a particular authority during a given time window, or identify credentials pending expiration. These queries execute efficiently due to the indexed fields and the small footprint of the metadata. Because the entire system is designed to function in a decentralized manner, all queries are performed locally, ensuring that verifiers or agents can perform administrative analysis even in the absence of network connectivity.

To be precise, the integration is performed via the `better-sqlite3` Node.js library, which allows performant synchronous database access within the revocation service runtime environment. The SQLite module is initialized upon system start-up and ensures that necessary tables and indices are created if absent. This design facilitates ease of deployment and extensibility, enabling additional fields additions to the schema with minimal code changes, and ensures that administrative tools can interact with the database using standard SQL interfaces or bindings.

3.3.2 Application Interface Layer

To support seamless interaction with the system, both by users and by other decentralized identity components, we introduce an **application interface layer**. This layer abstracts the underlying cryptographic operations, metadata management, and accumulator state transitions, providing a high-level control surface that supports common identity operations such as credential issuance, revocation, verification, and metadata queries. It also serves as the boundary through which decentralized identity agents, administrators, or automated systems can access the functionality of the revocation engine without directly interacting with cryptographic primitives. The interface layer exposes both a command line interface (CLI) for local interaction and a RESTful API for programmatic access.

Upon credential issuance, the interface layer takes input from the user (e.g., issuer ID, credential type, recipient ID), generates a unique credential identifier, stores its metadata in the SQLite layer, and computes the necessary hash for potential future revocation inclusion. If the credential is later revoked, the interface hashes the identifier and inserts it into the active Bloom filter. When an epoch ends, the interface coordinates the transition: Bloom filter contents are mapped as primes, inserted into the static accumulator, and the resulting accumulator value is forwarded into the dynamic accumulator to preserve system continuity.

Another key feature of the interface layer is **batch support**. The system allows multiple credentials to be issued or revoked simultaneously. This is especially beneficial in high-throughput or institutional environments where revocation operations often occur in large volumes (e.g., mass credential expiration or institution-wide key compromise). Batch operations are internally optimized to avoid redundant hash operations and allow for atomic updates to both the Bloom filter and SQLite database.

The interface also includes built-in error handling, status reporting, and logging mechanisms, ensuring that each operation is verifiable and auditable. Any operation that fails, whether due to input errors, schema inconsistencies, or accumulator computation issues,

returns structured error codes and descriptions. This makes the interface robust enough to integrate into production-ready systems and to support long-running service deployments.

From a design perspective, this layer is intentionally kept modular and stateless, relying on clear input/output contracts and avoiding persistent global state. This design enables future extensions, such as Web-based dashboards or integrations with DIDComm protocols.

3.4 Revocation and Verification Workflows overview

To demonstrate the practical functionality of our credential revocation architecture, this section outlines the high-level workflows that guide revocation and verification operations within the system. These scenarios highlight how the system behaves from the perspective of different stakeholders, such as issuers, holders, and verifiers, while maintaining modularity, verifiability, and privacy.

3.4.1 Revocation Workflow

The revocation process is responsible for marking a previously issued credential as invalid and ensuring that this status is consistently and provably reflected in all verifier interactions. The workflow proceeds as follows:

1. **Revocation Trigger:** A credential holder, issuer, or policy engine initiates a revocation (for example, due to expiration, compromise, or administrative action).
2. **Metadata Update:** The credential's metadata entry is updated in the local SQLite database to reflect the new revocation status. This includes setting a revocation timestamp and epoch association.
3. **Sub-Accumulator Insertion:** The credential identifier is processed and inserted into the Bloom filter (sub-accumulator) associated with the current epoch. This enables fast and probabilistic checks during the active period.
4. **Epoch Finalization:** Once the epoch ends, the contents of the Bloom filter are transformed into prime representatives and added to the static accumulator. The resulting accumulator value is then inserted into the dynamic accumulator, preserving a long-term revocation history.
5. **Publication:** The updated static and dynamic accumulator values are made publicly available, either through public append-only storage (e.g. smart contracts) or distributed means, allowing verifiers to retrieve and use them for status validation.

This process ensures both short-term responsiveness (through Bloom filters) and long-term verifiability (through RSA accumulators), with the metadata database acting as a synchronized, queryable record of administrative revocation actions.

3.4.2 Verification Workflow

Verification is the process by which a verifier determines whether a credential is currently valid or revoked. In the original Schumm et al. [22] architecture, this process is fully stateless. Verifiers use cryptographic proofs and published accumulator values to determine non-membership of a revoked set. Similarly, the enhanced architecture preserves this model but improves it by introducing a pre-verification layer through the use of the SQLite metadata module. The verification workflow is as follows:

1. **Metadata Pre-Check:** Upon receiving a credential, the verifier queries the local SQLite database for associated metadata. This includes issuer ID, issuance time, epoch information, and revocation flags. If the credential is expired, already revoked, or malformed, it can be rejected immediately without cryptographic evaluation.

2. **Sub-Accumulator Lookup:** If the credential falls within the active epoch, the Bloom filter is consulted. A negative result is sufficient to accept the credential; a positive result triggers further evaluation.

3. **Epoch Resolution:** If the issuance epoch is not the current request epoch, the verifier identifies and retrieves the corresponding sub-accumulator and static accumulator relevant for this credential.

4. **Non-Membership Proof Verification:** If the sub-accumulator yields a positive or ambiguous result, the verifier checks the non-membership proof against the relevant static accumulator value. This cryptographic proof confirms whether the credential identifier was included in the accumulator value for that epoch.

5. **Historical Validation:** The verifier validates that the static accumulator used is included in the dynamic accumulator itself, confirming historical integrity.

6. **Decision Output:** The credential is either accepted (if it passes all relevant checks) or rejected. Verification is performed locally, without need to query issuers or external services.

Integration of the SQLite query layer into the verification pipeline introduces an efficient optimization stage. By filtering invalid credentials early based on structured metadata, the system avoids unnecessary cryptographic processing, improving throughput and reducing latency. This is particularly valuable in batch verification scenarios, constrained environments, or systems requiring high-speed decision-making.

Together, these workflows demonstrate the enhanced system's ability to combine cryptographic rigor with practical operational features. The modular architecture supports both secure and efficient revocation, while allowing verifiers to act independently, privately, and scalably across a range of deployment contexts.

3.5 Security and Privacy Guarantees

The credential revocation system developed in this thesis is based on cryptographically robust primitives and is designed to operate securely under decentralized and privacy-conscious conditions. This section summarizes the key guarantees related to integrity, tamper-resistance, and privacy, which includes:

- **Cryptographic Integrity:** Both static and dynamic accumulators rely on the Strong RSA assumption [19], ensuring that once a credential identifier is included, it cannot be removed or altered without detection. Each static accumulator encodes the revoked set for an epoch, while the dynamic accumulator forms a verifiable chain of these historical values. Non-membership proofs remain compact and publicly verifiable, supporting decentralized, trustless validation of credential status.
- **Immutability and Tamper Resistance:** Once committed, the revocation entries are immutable. Static accumulator values represent finalized epoch revocations, and dynamic accumulation enforces append-only state progression. Any attempt to delete, re-order, or forge revocation data would be detectable due to the cryptographic structure.
- **Privacy and Local Verification:** Stateless verification allows verifiers to check revocation status using only public accumulator data and local credential metadata. No online queries are needed, protecting user privacy and enabling decentralized use. Hash functions play a crucial role in maintaining privacy during Bloom filter lookups and credential encoding, thereby avoiding disclosure of sensitive data.
- **SQLite Layer and Query Privacy:** The SQLite module operates locally, requiring no network communication or external synchronization. All queries and metadata handling remain under verifier control, ensuring no additional leakage or centralization.
- **Threat Mitigation:** The system is resilient to replay attacks (due to epoch binding), forgery (due to cryptographic proof construction), and accumulator tampering (due to the append-only structure).

In short, the system offers:

- Cryptographic soundness based on the **Strong RSA** assumption,
- immutable revocation history through accumulator chaining,
- privacy-preserving, stateless verification,
- decentralized and local metadata inspection.

These guarantees make the solution suitable for deployment in decentralized identity infrastructures with strong privacy and auditability requirements.

3.5.1 Performance Trade-off Framework

The solution embraces a strategic trade-off that exchanges individual operation speed for system predictability and elimination of worst-case performance scenarios. This reflects production deployment requirements where consistent performance metrics outweigh marginal improvements in individual operation speed.

The strategic acceptance of significant overhead for individual operations enables notable improvements in complex operations and eliminates temporal scaling issues. This transforms unpredictable performance into measurable costs acceptable for consistency-critical applications, directly addressing the research question of bypassing verification loops while maintaining cryptographic security guarantees.

Chapter 4

Implementation

In this chapter, we will be discussing the practical implementation of the design discussed in the previous chapter. Taking design ideas and turning them into something that runs reliably and consistently is never a straightforward path. Although the conceptual foundation was laid out by Schumm et al. [22], the solution we will present here takes their ideas further, adapting them to try and meet the needs of verifiers and administrators in real-world identity systems. Given that the main purpose of our work was to build on top of the existing architecture, we have kept the original source code mostly unchanged, only changing some key components (such as deploy.js), that were necessary for our solution to work. This was done with the exact idea of being able to use both systems, the new and the original, with minimal changes to the code and structure. This had the intended goal of being able to compare the two systems. Additionally this allowed for better evaluation and comparison of our system's performance to the original. Supporting our solution is the lightweight but powerful SQLite system, which was implemented via `sql-module/index.js`, and initialized in `sql-module/database/index.js`. Here, the schema design matters - tables are normalized to keep queries fast and expressive and the efficient use of indexes helps in making filtering scalable. The entire module was designed to operate efficiently in distributed environments while maintaining the stateless and privacy-respecting principles of the broader architecture.

Central to the design are two main services, which are intended to take on the heavy lifting and incorporate our SQLite database. These are revocation and verification service. These were designed with clarity and modularity in mind, bridging cryptographic logic with administrative usability. An additional important layer supporting them is `integration.js`, the connecting layer responsible for ensuring that the SQLite-backed metadata stays in sync with the revocation state. This component plays a quiet but essential role in coordinating between hash-based proof layers and structured SQL metadata. Unlike the cryptographic core, which was largely retained from the original work, all of these additions are new and were not a part of the original system.

In the following sections, we will use the bottom-up approach. Starting with the database layer, through revocation and verification workflows, all the way to how API flows incorporate everything together.

4.1 Database Layer and Schema Design

At the heart of the system's performance enhancement lies a lightweight and embedded SQLite database engine. This engine serves as an intelligent optimization layer for credential verification and revocation operations. The design of this layer was driven by practical observations of performance bottlenecks in blockchain-based credential systems. In particular, the computational overhead of cryptographic operations and the impact of the frequency of Bloom filter false positives. Although cryptographic accumulators provide strong security guarantees for revocation, they lack efficient mechanisms for handling verification optimization and false positive mitigation. Our solution addresses these limitations through a carefully designed SQLite-backed optimization layer that complements the underlying cryptographic infrastructure.

The management of this optimization layer is distributed across several specialized modules within the `sql-module` directory. The main coordination occurs through `sql-module/index.js`, which provides a clean API interface and manages the module's singleton instance. Database initialization and schema management are handled by `sql-module/database/index.js`, ensuring proper table creation and connection management during system startup. The database schema was specifically designed to address the aforementioned challenges that we observed in the accumulator-based systems. Rather than storing general credential metadata, each table serves a specific optimization purpose. The schema consists of five interconnected tables that work together in order to accelerate credential operations:

- `DefinitiveRevocations` captures confirmed revocation records with credential identifiers, revocation timestamps, epoch information, issuer details, and associated prime values. This table serves as our authoritative source for revocation status, enabling instant verification without blockchain queries.
- `FalsePositives` tracks Bloom filter false positives by recording credential-epoch pairs along with detection timestamps and occurrence counts. This learning mechanism allows the system to recognize and handle repeated false positives efficiently.
- `RevocationBatches` coordinates bulk revocation operations by maintaining batch metadata, including creation times, processing status, and item counts. This enables atomic processing of multiple revocations with proper error handling.
- `RevocationBatchItems` stores individual items within each batch, maintaining foreign key relationships while tracking the processing status for each credential. This design supports incremental batch processing and recovery from partial failures.
- `PerformanceAnalytics` collects detailed timing and operational data, including execution times, operation types, and false positive detection events. This instrumentation helps us continuously optimize system performance based on real usage patterns.

Each table includes carefully chosen indexes to support the system's most critical operations. Primary keys enable constant-time credential lookups, whereas composite indexes

on credential-epoch pairs optimize false positive queries. Foreign key constraints maintain data integrity across batch operations without sacrificing query performance. Throughout the implementation, we use prepared statements in order to try to maximize performance and prevent security vulnerabilities.

From an operational perspective, credential verification follows a staged approach that prioritizes speed for the most common cases. When a verification request arrives, the system first checks the `DefinitiveRevocations` table for a definitive answer. If no record exists but the Bloom filter suggests the credential might be revoked, we consult the `FalsePositives` table to see if this is a known false positive. All new false positives are automatically stored for future reference, while repeated occurrences simply increment the verification counter. Each operation is timed and recorded in `PerformanceAnalytics` to help us understand and improve system behavior over time.

For revocation operations, the system supports both immediate single-credential revocations and efficient batch processing. Individual revocations are recorded directly in the `DefinitiveRevocations` table using `upsert` operations to handle duplicate attempts gracefully. Batch revocations leverage the batch tables to coordinate atomic processing of multiple credentials, with epoch-based grouping to optimize the underlying cryptographic operations. All operations occur within appropriate transaction boundaries to ensure consistency.

The system maintains complete operational independence through its embedded architecture. The SQLite engine runs locally using the `better-sqlite3` library, with no external database servers required. This embedded approach ensures the system remains portable and aligns with our broader architectural goals of decentralization, while still providing the performance benefits of a mature relational database engine.

4.2 Revocation Service Logic

The revocation service represents one of the most critical components of the entire `SQL-module`. The service was designed to directly address the performance bottlenecks associated with credential revocation in the accumulator-based systems. Traditional blockchain-based revocation mechanisms, while cryptographically sound, suffer from significant latency issues due to inefficiencies in blockchain search and computational overhead associated with accumulator updates. Our revocation service was created to address these limitations through a dual processing approach that maintains the security guarantees of the underlying cryptographic system while trying to improve operational efficiency. The `sql-module/services/revocationService.js` file contains the practical implementation of the revocation service. Within the application, we created individual and batch processing capabilities. This design decision emerged from our analysis of real-world credential usage patterns, where revocation events often occur in clusters - such as when an employee leaves an organization or when a larger batch of credentials expires simultaneously. This service is meant to maintain a balance between immediate processing for urgent revocations and optimized batch processing for bulk operations. The approach

is adapted to ensure that different operational scenarios are handled with appropriate efficiency.

4.2.1 Individual Revocation Processing

At the core of our revocation service, we created a prepared statement strategy that is meant to optimize database interactions. During initialization, the service pre-compiles all SQL statements that will be used during the operation, eliminating query compilation overhead during high-frequency operations. The prepared statement approach is demonstrated in the service initialization:

```
prepareStatements() {
  this.insertRevocation = this.db.prepare(`
    INSERT INTO DefinitiveRevocations (credential_id, epoch_id,
       issuer_id, prime_value)
    VALUES (?, ?, ?, ?)
    ON CONFLICT(credential_id) DO UPDATE SET
      revocation_time = CURRENT_TIMESTAMP,
      epoch_id = excluded.epoch_id,
      issuer_id = excluded.issuer_id,
      prime_value = excluded.prime_value
  `);

  this.createRevocationBatch = this.db.prepare(`
    INSERT INTO RevocationBatches DEFAULT VALUES
  `);

  this.addBatchItem = this.db.prepare(`
    INSERT INTO RevocationBatchItems (batch_id, credential_id,
       prime_value, epoch_id, issuer_id)
    VALUES (?, ?, ?, ?, ?)
  `);
}
```

Listing 4.1: Prepared statement initialization for optimized database operations

The described approach is intended to provide consistent execution times regardless of system load and to ensure that duplicate revocation attempts are handled quickly. These prepared statements incorporate comprehensive metadata capture, recording the credential identifier, associated epoch, issuer information, and cryptographic prime values.

On the other hand, individual revocation processing follows a streamlined workflow that prioritizes both speed and reliability. A particularly important aspect of this processing is the automatic cleanup of false positive records, which prevents the accumulation of stale data and ensures accurate future verification results:

```
recordRevocation(revocationData) {
  const { credential_id, epoch_id, issuer_id, prime_value } =
     revocationData;
  const startTime = performance.now();

  try {
```

```
6       this.insertRevocation.run(credential_id, epoch_id, issuer_id,
            prime_value);
7
8       // Clean up any false positive records for this credential
9       this.db.prepare(`
10        DELETE FROM FalsePositives WHERE credential_id = ?
11      `).run(credential_id);
12
13      this.recordPerformance(epoch_id, 'revocation', startTime);
14      return true;
15    } catch (error) {
16      if (error.code === 'SQLITE_CONSTRAINT') {
17        // Already revoked, not an error
18        return true;
19      }
20      console.error('Failed to record revocation:', error);
21      return false;
22    }
23  }
```

Listing 4.2: Individual revocation processing with automatic false positive cleanup

The cleanup operation will occur within the same logical operation as the revocation recording, maintaining atomicity and preventing race conditions that could potentially lead to inconsistent state. Error handling is meant to distinguish between expected conditions (such as duplicate revocations) and genuine failures, ensuring robust operation under various scenarios.

4.2.2 Batch Processing and Performance Management

Batch revocation processing represents perhaps the most significant innovation in our approach. It is designed to address scenarios where hundreds or thousands of credentials need to be revoked simultaneously. The batch processing system implements a two-phase commit pattern with sophisticated epoch-based grouping that optimizes cryptographic operations. The grouping algorithm understands that operations could be accelerated when processing multiple credentials from the same epoch together:

```
1  processBatch(batchId) {
2    const startTime = performance.now();
3
4    try {
5      // Get all items in the batch
6      const items = this.getBatchItems.all(batchId);
7
8      // Group by epoch for efficient processing
9      const itemsByEpoch = {};
10     items.forEach(item => {
11       if (!itemsByEpoch[item.epoch_id]) {
12         itemsByEpoch[item.epoch_id] = [];
13       }
14       itemsByEpoch[item.epoch_id].push(item);
15     });
16
```

```
17      // Process each epoch group within transactions
18      for (const [epochId, epochItems] of Object.entries(itemsByEpoch)) {
19        const epochStartTime = performance.now();
20
21        // Process all items in this epoch atomically
22        this.db.transaction(() => {
23          for (const item of epochItems) {
24            this.insertRevocation.run(
25              item.credential_id,
26              item.epoch_id,
27              item.issuer_id,
28              item.prime_value
29            );
30          }
31        })();
32
33        this.recordPerformance(parseInt(epochId, 10), 'batch-revocation',
             epochStartTime);
34      }
35
36      // Mark the batch as processed
37      this.markBatchProcessed.run(batchId);
38      this.markBatchItemProcessed.run(batchId);
39
40      return {
41        success: true,
42        itemCount: items.length,
43        executionTime: performance.now() - startTime
44      };
45    } catch (error) {
46      console.error('Failed to process batch:', error);
47      return {
48        success: false,
49        error: error.message,
50        executionTime: performance.now() - startTime
51      };
52    }
53  }
```

Listing 4.3: Epoch-based grouping algorithm for optimized batch processing

The described grouping strategy is intended to reduce computational overhead by enabling optimized prime number operations alongside accumulator updates for credentials within the same time slot. The nested transaction structure ensures that either all items within an epoch group are processed successfully or the entire group operation is rolled back. This approach ensures consistency even in the face of partial failures.

Performance instrumentation is deeply integrated throughout the revocation service, capturing detailed timing information for every operation type. The instrumentation occurs at multiple levels:

- **individual revocations** are timed from request initiation to database commit,
- **batch operations** save both overall processing time and per-epoch processing times.

. This granular data collection allows for continuous optimization based on real-world usage patterns, while also helping identify performance bottlenecks as they emerge.

The batch processing system also includes a comprehensive status tracking application and error recovery mechanisms. Batch metadata maintains processing state information, enabling monitoring of long-running operations and supporting resumable processing after system interruptions. The service provides detailed statistics on revocation patterns, batch processing efficiency, and system performance trends through dedicated query interfaces. Error handling within the revocation service prioritizes system stability while ensuring that transient failures do not result in lost revocation requests. Database constraint violations are managed with elegance, and exhaustive logging facilitates troubleshooting and the monitoring of systems within production environments. Integration with the broader system occurs through the well-defined interfaces that maintain separation of concerns whilst enabling seamless operations. The revocation service coordinates with the verification service to ensure an immediate reflection of the revocation status in the verification results. Meanwhile, the performance data are fed into system-wide optimization algorithms that tune processing parameters based on observed patterns.

The revocation service architecture is designed to accommodate future enhancements and extensions. Some of these may include distributed processing for horizontal scaling, message queue integration for asynchronous processing, and machine learning optimization of batch grouping strategies. This approach ensures that the service can evolve to meet the growing demands of the system while maintaining its core performance and reliability characteristics.

4.3 Verification Service Logic

Another key component of our `sql-module` is the verification service. This service implements a complex multi-stage pipeline that reduces the latency of credential verifications while meeting the security assurances provided by the underlying cryptographic scheme. Verifying a credential using the accumulator-based architecture includes querying the blockchain and performing heavy cryptographic computations for every verification request, even when the credential in question has already been verified or is subject to known false positives. The verification service was created with the intention of trying to eliminate this inefficiency through a smart caching and learning solution based on the verification history. The verification service application implemented in `sql-module/services/verificationService.js` is based on the principle of decreasing complexity, in which the service does not immediately perform computationally expensive cryptographic operations. Instead, it uses a cascading decision tree that processes a verification request through less complicated but computationally cheaper methods, until those methods are no longer capable of resolving the request. This approach recognizes that the overwhelming majority of credential verifications in production systems will be valid, non-revoked credentials, and thus biases heavily toward that scenario while still ensuring correctness regardless of the case.

The verification service utilizes the same prepared statement optimization described in the revocation service. During initialization, the service pre-compiles all possible queries that will be executed during verification operations. This ensures consistent performance regardless of system load and leads to the elimination of query parsing overhead during high-frequency operations. The prepared statements cover the entire spectrum of verification scenarios, from a simple revocation lookup to complex false positive analysis.

4.3.1 Verification Design and Implementation

The core verification algorithm implements a three-stage decision pipeline that balances verification speed with accuracy requirements. The first stage exploits the fundamental mathematical property of Bloom filters - false negatives are impossible. If the bloom filter claims that a credential is not revoked, then it is definitely correct and therefore requires no more verification. This functionality alone eliminates the need for database queries in the vast majority of verification requests.

When the Bloom filter indicates a potential revocation, the second stage consults the `DefinitiveRevocations` table for definitive revocation status. This lookup operation, while more expensive than the Bloom filter check, should be significantly faster than blockchain queries whilst providing definitive answers for credentials that have been previously revoked through the system. Finally, the third stage represents the most sophisticated aspect of the verification service, the false positive detection and learning mechanism. When a Bloom filter indicates a revocation, but no definitive revocation record exists, the system must then determine whether this represents a genuine revocation that has not yet been recorded locally, or a false positive that should be cached for future reference. This decision is critical to our system performance, as incorrect false positive

classification could lead to security vulnerabilities, while overly conservative classification reduces optimization effectiveness:

```
verifyCredential(credentialId, epochId, bloomFilterResult) {
 const startTime = performance.now();

 try {
   // Stage 1: Bloom filter negative result (definitively not revoked)
   if (!bloomFilterResult) {
     this.recordPerformance(epochId, 'verify-bloom-negative', startTime,
         false);
     return {
       valid: true,
       method: 'bloom-filter',
       checkedAt: new Date().toISOString()
     };
   }

   // Stage 2: Check definitive revocation records
   const revoked = this.checkRevocation.get(credentialId);

   if (revoked) {
     this.recordPerformance(epochId, 'verify-sql-positive', startTime,
        false);
     return {
       valid: false,
       method: 'sql-definitive',
       revocationTime: revoked.revocation_time,
       checkedAt: new Date().toISOString()
     };
   }

   // Stage 3: False positive analysis and learning
   return this.handlePotentialFalsePositive(credentialId, epochId,
       startTime);

 } catch (error) {
   console.error('Optimized verification failed:', error);
   return null; // Trigger fallback to original method
 }
}
```

Listing 4.4: Multi-stage verification pipeline with Bloom filter optimization

The false positive learning mechanism maintains detailed statistics about each false positive it encounters, including frequency of occurrence and temporal patterns. This data enables the system to build confidence in false positive classifications over time and supports advanced analytics that can be used for system tuning. The learning algorithm is designed to be conservative, defaulting to a valid credential status when there is uncertainty, thereby choosing to prioritize system security over optimization efficiency:

```
handlePotentialFalsePositive(credentialId, epochId, startTime) {
 // Check if we've encountered this false positive before
 const fpRecord = this.checkFalsePositive.get(credentialId, epochId);

```

```
5   if (fpRecord) {
6     // Known false positive - update counter and return cached result
7     try {
8       this.updateFalsePositive.run(credentialId, epochId);
9     } catch (updateError) {
10      console.warn(`Failed to update false positive record: ${updateError
            .message}`);
11    }
12
13    this.recordPerformance(epochId, 'verify-known-false-positive',
          startTime, true);
14    return {
15      valid: true,
16      method: 'false-positive-cache',
17      occurrences: fpRecord.verification_count + 1,
18      checkedAt: new Date().toISOString()
19    };
20  }
21
22  // Potential new false positive - record for learning
23  try {
24    this.insertFalsePositive.run(credentialId, epochId);
25  } catch (insertError) {
26    console.warn(`Failed to insert false positive record: ${insertError.
          message}`);
27  }
28
29  this.recordPerformance(epochId, 'verify-new-false-positive', startTime,
        true);
30
31  return {
32    valid: true,
33    method: 'new-false-positive',
34    checkedAt: new Date().toISOString()
35  };
36 }
```

Listing 4.5: False positive detection and learning algorithm

4.3.2 Performance Optimization and System Integration

The batch verification functionality builds on individual verification optimizations for situations where multiple credentials should be verified simultaneously. Given that the batch processing algorithm supports and utilizes many database query optimization strategies, it has the ability to minimize per-credential overhead through amortized setup costs, therefore increasing throughput for high volume verification situations. Additionally, while maintaining the same three-stage verification logic for each credential, batch processing will allow more efficient use of resources and services within the verification service.

Error handling within the verification service was designed to follow a fail-safe approach that prioritizes security over performance optimization. A failure in the SQL optimization pipeline will trigger a fall-back method to the original cryptographic verification, ensuring

verification correctness will never be compromised as a result of verification optimization. This approach means that the verification service will only be able to improve system performance without introducing security vulnerabilities or incorrect verification results.

Performance instrumentation is included in all aspects of the verification service. It captures timing data points at every stage of the verification and each classification that is applied afterward. Moreover it serves many purposes: monitoring real-time performance, informing capacity planning decisions, and the ability to have empirical data for continuous improvement of systems. The performance data the system collects includes not only execution times, but also operation classifications, false positive detection rates, and cache hit ratios. The verification service also provides comprehensive analytics interfaces that support system monitoring and optimization. These interfaces show false positive statistics, verification method distribution, performance trends, and cache effectiveness metrics. This analytics data enables system administrators to adjust Bloom filter parameters and false positive thresholds to fine-tune the solution for their needs. These analytics can additionally be used to identify opportunities for further optimization based on actual usage patterns.

Integration with other components occurs through well-defined interfaces. The verification service receives Bloom filter results from the original cryptographic system, maintains coordination with the revocation service for state consistency, and provides performance metrics for system monitoring. Using this approach to integration, we ensure that verification optimizations remain effective even as other system components change.

Looking toward possible future enhancements, the verification service architecture allows for several potential improvements. The use of machine learning methods within integration could enable more sophisticated false positive predictions based on credential characteristics and usage patterns. Additionally, distributed caching could extend the optimization benefits to multiple system instances. Lastly, we could employ advanced analytics to help identify verification patterns that suggest optimization opportunities or security concerns requiring attention.

The verification service ultimately demonstrates how careful examination of operational behavior can find a notable improvement in performance without compromising security properties. By recognizing that most verification operations will yield predictable results and effectively caching the results of costly operations, the verification service has transformed the formal verification of credentials from a computationally expensive operation to an optimized lookup process while still retaining the security guarantees that are essential to production credential system.

4.4 Integration Architecture and System Coordination

The integration layer possibly represents the most architecturally significant aspect of our `sql-module`, as it enables seamless enhancement of existing credential systems without requiring modifications to their core implementation. The design requirement for non-invasive integration was adopted with the understanding that production credential systems typically cannot undergo extensive architectural changes to account for new optimization functionality. The integration architecture,which is implemented primarily in `sql-module/integration.js`,employs advanced software engineering paradigms to encapsulate existing system functionalities while ensuring the preservation of all original behaviors and facilitating graceful degradation in scenarios where optimizations cannot be applied.

The integration strategy is designed using the decorator pattern which allows the SQL optimization module to improve the existing credential system methods while remaining behaviorally compatible. As a result, the existing client code can continue to operate exactly as it did before, while transparently benefiting from performance optimizations when it is required.The implementation of decorator pattern intercepts the method calls to the original credential system, attempts to serve optimized responses via the SQL module, and will return to the original functionality when the methods cannot be optimized or there is an error encountered. At the heart of the integration approach is the principle of conservative optimization. The SQL component never overwrites or contradicts the original results derived from the underlying cryptographic system. Instead, it aims to provide equivalent results in a more efficient way, and fall back to the original system when the SQL results produce ambiguity. This conservative approach protects the integration layer from introducing new security vulnerabilities or producing incorrect or unverified results. In sum, this means that integration can be deployed in a production without the need for extensive re-validation of existing functionality.

The integration process first creates an enhanced system wrapper to retain all of the original functionality, while creating the optimization capability. The core integration function shows how the original system can be enhanced through optimization, without changing the original system:

```
1  function integrateWithCredentialRevocation(originalSystem, options = {})
       {
2   const sqlModule = SQLModule.getInstance(options);
3
4   // Keep references to original methods
5   const originalVerifyCredential = originalSystem.verifyCredential;
6   const originalRevokeCredential = originalSystem.revokeCredential;
7
8   // Enhanced system with SQL optimizations
9   return {
10    // Pass through other methods from original system
11    ...originalSystem,
12
13    verifyCredential: async function(credentialId, issuanceEpoch, ...args
        ) {
14      try {
```

```
15        // First run the original verification to get the bloom filter
              result
16        const originalResult = await originalVerifyCredential.call(
17          originalSystem,
18          credentialId,
19          issuanceEpoch,
20          ...args
21        );
22
23        // Extract bloom filter result from original verification
24        const bloomFilterResult = !originalResult;
25
26        // Try to optimize using SQL
27        const sqlResult = sqlModule.verifyCredential(
28          credentialId,
29          issuanceEpoch,
30          bloomFilterResult
31        );
32
33        if (sqlResult !== null) {
34          // SQL optimization successful
35          return {
36            ...sqlResult,
37            optimized: true
38          };
39        }
40
41        // Fall back to original result if SQL optimization fails
42        return originalResult;
43      } catch (error) {
44        console.error('Error in enhanced verification:', error);
45        // Fall back to original method on error
46        return originalVerifyCredential.call(
47          originalSystem,
48          credentialId,
49          issuanceEpoch,
50          ...args
51        );
52      }
53    }
54  };
55 }
```

Listing 4.6: Non-intrusive system enhancement through decorator pattern

This integration approach preserves the complete interface of the original system while transparently attempting optimization. The enhanced verification method first calls the original verification to establish the baseline result and extracts necessary information (such as Bloom filter outcomes). After that, the system attempts to provide the same result more efficiently through the SQL module. If optimization is successful, the result includes additional metadata that indicates the optimization method used. If optimization fails or is not applicable, the original result is returned unchanged.

The integration layer is tasked with handling the more complex problem of coordinating two different system paradigms: the crypto-accumulator-based original layer and the

database-optimized enhancement layer. Coordination requires acknowledging many of the same issues we discussed earlier, but with consideration for data consistency, timing issues and error scenarios. The integration logic makes sure that both systems maintain consistent views of credential state, even if the actions taken to manipulate a credential state have taken different paths.

For revocations, the integration layer implements a dual-processing approach to maintain both the accumulator-based state and the optimization state so that all subsequent verifications can leverage both the security assurances of the accumulator-based system and the performance benefits of the SQL-Module:

```
revokeCredential: async function(credentialId, issuerId, ...args) {
 try {
   // Call original revocation method
   const originalResult = await originalRevokeCredential.call(
     originalSystem,
     credentialId,
     issuerId,
     ...args
   );

   // Record in SQL for future optimization
   const revocationData = {
     credential_id: credentialId,
     issuer_id: issuerId,
     epoch_id: originalResult && originalResult.epochId ? originalResult
         .epochId : 1,
     prime_value: originalResult && originalResult.primeValue ?
         originalResult.primeValue : credentialId
   };

   sqlModule.recordRevocation(revocationData);

   return {
     ...originalResult,
     optimized: true
   };
 } catch (error) {
   console.error('Error in enhanced revocation:', error);
   // Fall back to original method on error
   return originalRevokeCredential.call(
     originalSystem,
     credentialId,
     issuerId,
     ...args
   );
 }
}
```

Listing 4.7: Dual-processing revocation with state synchronization

As the dual-processing strategy elaborated above achieves the centralization and order of the revocations using a dynamic accumulator revocation list by the original system and SQL enhancement, these redundancies are of course, architectural purpose. The

cryptography layer works to defend the security assumptions and serves as the authorizing agent, while the SQL layer drives optimization potentials.

One of the crucial components of the integration layer is its handling of batch operations, particularly for verification scenarios where the original system may not provide native batch processing capabilities. The integration layer implements intelligent batch operations that effectively utilize both systems:

```
batchVerifyCredentials: async function(credentials) {
 try {
   const results = {};
   const credentialsWithBloomResults = [];

   // First, run original verification to get bloom filter results
   for (const cred of credentials) {
     const originalResult = await originalVerifyCredential.call(
       originalSystem,
       cred.id,
       cred.epoch
     );

     credentialsWithBloomResults.push({
       ...cred,
       bloomResult: !originalResult,
       originalResult
     });
   }

   // Use SQL batch optimization
   const sqlResults = sqlModule.batchVerifyCredentials(
      credentialsWithBloomResults);

   if (sqlResults !== null) {
     // Process SQL results, ensuring consistency with original results
     for (const cred of credentialsWithBloomResults) {
       const sqlResult = sqlResults[cred.id];

       // Check for discrepancy between systems
       if (sqlResult && cred.originalResult !== sqlResult.valid) {
         console.log(`Batch verification discrepancy for ${cred.id.
            substring(0, 8)}:`,
                `Original: ${cred.originalResult}, SQL: ${sqlResult.
                   valid}`);

         // Use original result when discrepancy exists
         results[cred.id] = {
           valid: cred.originalResult,
           method: `${sqlResult.method}-discrepancy`,
           optimized: true,
           discrepancy: true
         };
       } else if (sqlResult) {
         results[cred.id] = {
           ...sqlResult,
           optimized: true
         };
```

```
46          }
47        }
48
49        return results;
50      }
51
52      // Fallback to individual verification if batch processing fails
53      for (const cred of credentials) {
54        results[cred.id] = await this.verifyCredential(cred.id, cred.epoch)
              ;
55      }
56
57      return results;
58    } catch (error) {
59      console.error('Error in batch verification:', error);
60      // Final fallback to original individual verification
61      const results = {};
62      for (const cred of credentials) {
63        results[cred.id] = await originalVerifyCredential.call(
64          originalSystem,
65          cred.id,
66          cred.epoch
67        );
68      }
69      return results;
70    }
71  }
```

Listing 4.8: Intelligent batch processing with result consistency verification

The implementation of batch processing displays the integration layer's desire to support consistency in results and reliability in systems to perform in a trustworthy manner. The algorithm in the integration layer will first establish baseline results using the original system, then try to reproduce the baseline results as efficiently as possible through the SQL optimization layer. If there are differences between the results through either system, the integration layer will log the difference (and the circumstances around it), but will fall back to the results based on the original system. The fallback and logging strategy also ensures that optimizations will not force a result to be judged incorrect due to executions in an edge case or system state that the implementation is not designed for.

The error handling mechanisms throughout the integration layer is layered to provide multiple fallback options. The primary error handling fallback occurs at the per method level of each enhancement, where if an optimization failure occurs, it will be caught, logged, and fall back capability will make the system use the original functionality. The secondary error handling mechanism occurs at the integration layer level. This mechanism will, in case of failures in the optimization system, allow for essential credential operations to be completed. The tertiary error handling in the system operates at the recovery level, allowing the system to reinitialize out-of-service optimizations through the system recovery set of operations.

The integration architecture also provides comprehensive monitoring and analytics capabilities that span both the original and enhanced systems. Performance metrics capture

not only the execution times of enhanced operations, but also the effectiveness of optimizations, the frequency of fallback scenarios, and the consistency between system components. This monitoring data supports both operational troubleshooting and ongoing system optimization efforts.

Configuration management within the integration layer allows for fine-tuned control over optimization behavior without requiring changes to client code. Configuration options control optimization aggressiveness, fallback thresholds, monitoring verbosity, and feature enablement. This flexibility enables deployment scenarios where optimizations can be gradually enabled and tuned based on observed system behavior and performance requirements.

The integration layer's design anticipates future system evolution and provides extension points for additional optimizations and enhancements. The decorator pattern can accommodate new optimization strategies, the monitoring infrastructure can support additional metrics, and the configuration system can handle new optimization parameters. This extensibility ensures that the integration architecture can evolve alongside both the underlying credential system and the optimization technologies.

Ultimately, the integration architecture demonstrates that significant system enhancements can be achieved without disruptive changes to existing implementations. By carefully preserving existing interfaces while transparently adding optimization capabilities, the integration layer enables production credential systems to benefit from advanced optimization techniques while maintaining the stability and reliability characteristics essential for secure credential management.

4.5 Interfaces and Flow Orchestration

To bridge the modular backend layers with practical usage workflows, we designed a flexible and minimalistic interface architecture. These interfaces expose the internal services — revocation, verification, and integration — via both CLI tools and programmable function calls. The goal was to ensure that each system capability could be triggered in isolation or orchestrated as part of a larger credential management pipeline.

Our solution uses a lightweight execution model where interfaces operate within a single Node.js runtime. This design simplifies deployment while preserving the modularity necessary for future scaling or remote exposure via REST APIs or message queues. In its current form, the orchestration layer coordinates credential issuance, revocation, verification, and batch operations through tightly defined method contracts and CLI commands. These interactions are facilitated through the top-level scripts and entry points defined in the system, with services coordinated either through wrapper functions or integration logic.

At a high level, the system follows a structured operational pipeline. The flow begins with credential issuance, which assigns unique identifiers and primes for future accumulator operations. These credentials are optionally stored in the metadata database for future reference. When revocation is triggered, either individually or in bulk, the system

generates the necessary cryptographic values and records the revocation in both the accumulator and the SQLite-backed state. Verification flows initiate with an optional Bloom filter check, followed by database lookups and, when necessary, fallback to cryptographic validation.

The orchestration between services is designed with strict isolation of concerns but with shared data formats and state assumptions. The verification and revocation services expose minimal APIs that accept credential identifiers, epoch information, and issuer metadata. These APIs are composed into higher-level flows by the integration module, which acts as both a coordinator and a fail-safe bridge. It ensures that errors in one layer (e.g., SQL lookup failures) do not compromise the correctness or security of the entire verification operation.

Operational consistency is maintained through strict transactional semantics within the revocation and verification modules. These semantics ensure that a revoked credential is immediately recognized by the verification system without needing external synchronization. Moreover, the logging and performance instrumentation introduced in prior sections are surfaced through the orchestration layer, allowing administrators or automated systems to analyze execution outcomes and monitor system health.

While the current orchestration model is CLI-driven, it can be trivially wrapped in web services for remote administration. The modular service definitions make the architecture well-suited for various deployment scenarios in decentralized credential networks. In future versions, additional interfaces (e.g., WebSocket or gRPC) could be introduced to support streaming verification or scalable credential workflows.

The orchestration layer thereby represents more than a controller of components: it embodies the practical affordance layer that transforms the underlying cryptographic and database subsystems into a usable, verifiable, and extensible revocation system. It completes the architecture by demonstrating how the system can be integrated into larger identity platforms or deployed as a standalone credential validation engine.

4.6 Observations and Reflections

Implementing a revocation architecture that is simultaneously cryptographically sound, metadata-queryable, and network-compatible surfaced a number of challenges not evident during the design phase. While Schumm et al. [22]'s work offered a conceptually elegant and security-centric foundation, adapting their architecture for real-world conditions revealed a host of latent frictions related to operational scalability, fault tolerance, and maintainability.

One of the earliest and most significant observations was that theoretical models rarely account for the sheer volume and volatility of real-world credential lifecycles. Revocations tend to occur in bursts — organizational exits, certificate expirations, or incident responses often involve large batches of updates. The original design handled credential revocation in a piecemeal manner, with no inherent support for batching. Implementing batch-aware workflows required not only database schema extensions, but also careful

design of epoch grouping algorithms to preserve accumulator integrity while minimizing performance overhead.

Another key insight emerged from false positive management. While Bloom filters provide strong guarantees against false negatives, their tendency to produce false positives in edge cases was initially treated as a minor inconvenience. However, under frequent verification conditions, these false positives accrued into measurable performance degradation. The decision to explicitly track and learn from such anomalies through SQL-layer instrumentation proved critical. This learning mechanism not only improved average verification times but also demonstrated that statistical verification behaviors could be integrated into deterministic systems without compromising their correctness.

Modularity also played a defining role in the system's implementation success. The separation between cryptographic logic and metadata orchestration enabled cleaner interfaces and better fault isolation. It also meant that the SQL optimization layer could be turned off entirely in debugging contexts or future deployments, preserving a compatibility layer with the original implementation. This design choice provided both flexibility and resilience.

Unexpected complexities also arose. Coordinating revocation status across multiple layers — Bloom filter, static and dynamic accumulators, and SQL metadata — required careful attention to data synchronization, transaction management, and error propagation. Even minor deviations in state between these components could result in verification mismatches or failed revocations. This challenge shaped the final integration strategy, which prioritizes conservative fallbacks and dual-recording mechanisms to ensure eventual consistency.

From a development perspective, building the system with lightweight tooling proved to be a double-edged sword. While it offered excellent portability and minimal external dependencies, it required more deliberate handling of concurrency, logging, and process management. Similarly, the lightweight deployment approach helped preserve system simplicity, but necessitated tight control over environment configuration and deployment paths.

Overall, the implementation phase transformed abstract design principles into an operational system capable of adapting to various usage contexts. The process underscored that usability, maintainability, and verifiability are not easily reconcilable goals in isolation — but can coexist with thoughtful engineering and layered architecture.

4.7 Summary

This chapter presented the practical implementation of a credential revocation system that builds upon and extends the architecture proposed by Schumm et al. [22]. While preserving the cryptographic foundations of the original model, we introduced a modular SQL-based optimization layer that introduces metadata queryability, supports batch operations, and enhances verification performance without compromising security guarantees.

The system was implemented using Node.js and a focused set of libraries designed for embedded and low-overhead environments. Central to our contributions is the integration of an embedded SQLite engine, which supports structured metadata queries and optimization layers that reduce verification complexity through learned behavior and pre-filtering.

Key components include:

- A revocation service capable of both individual and batch processing, using epoch-based grouping and transactional safety.
- A verification pipeline that minimizes unnecessary computation via Bloom filter checks, SQL fallbacks, and false positive caching.
- An integration layer using the decorator pattern to non-invasively enhance existing systems, with full support for dual-processing and batch consistency.
- Interfaces and orchestration flows that enable the system to operate either independently or as a modular extension within a larger identity infrastructure.

Throughout implementation, a strong emphasis was placed on error handling, instrumentation, and monitoring, ensuring that the system not only performs well under typical conditions but can also adapt gracefully under load or partial failure.

The result is a system that supports the design goal of efficient, privacy-preserving credential revocation, while making meaningful advances in usability, maintainability, and performance. These enhancements position it well for adoption in real-world decentralized identity ecosystems where efficient verification, rapid revocation, and metadata auditability are critical features.

Chapter 5

Evaluation

This chapter presents a comprehensive evaluation and performance analysis of the enhanced credential revocation system developed in this thesis. The evaluation examines how our SQL-based optimization layer addresses the core limitation identified in Schumm et al. [22]'s work, mainly the requirement for verifiers to perform a large number of verification loops when searching through accumulator data structures. Through systematic performance analysis, we demonstrate that our approach provides predictable search performance through strategic trade-offs that make our solution suitable for consistency-critical scenarios.

The evaluation is structured around four key dimensions as specified within the task description:

- **performance analysis** examining execution times and throughput improvements,
- **scalability assessment** evaluating behavior under increased load of operation,
- **cost analysis** comparing resource utilization and operational overhead,
- and **security validation** confirming preservation of cryptographic guarantees.

In addition, a detailed comparative analysis will be presented highlighting the specific improvements achieved over the baseline system.

5.1 Introduction and Evaluation Methodology

The primary motivation for this evaluation stems from the fundamental performance bottleneck identified in the existing accumulator-based revocation systems. As noted during our research, credential verifiers in the baseline system are required to perform extensive verification loops to reach conclusions about credential status. This problem is essentially rooted in inefficient blockchain-based search mechanisms, where the time

required for verification can vary significantly depending on the temporal distance between credential issuance and verification attempts.

As mentioned before the solution addresses this limitation through a structured SQLite metadata query layer, which operates alongside the cryptographic revocation engine. The key innovation lies in transforming the search process from a computationally expensive, loop-intensive operation into an optimized lookup process that leverages database indexing and caching mechanisms. This approach maintains the security guarantees of the original system while improving operational efficiency. Crucially, our evaluation demonstrates that the verification performance of our system is not affected by the time elapsed between the issuance and verification of the credentials. Cross-epoch verification operations show minimal performance variation (less than 2% difference between recent and historical credentials), directly addressing the core scalability limitations of other approaches. While our approach introduces overhead for individual operations, it solves the verification loops problem by transforming it into a predictable, constant-time database lookup process.

5.1.1 Evaluation Framework and Test Design

To ensure comprehensive and reproducible evaluation results, we developed an enhanced test framework that enables direct comparison between the baseline system and SQL-optimized implementation. The evaluation methodology is built around the creation of five test suites, where each `sqlTest[X].js` corresponds directly to the original `test[X].js`'s used in the original solution. Each of the five `sqlTest[X].js` was designed to test the same functionalities as the corresponding original `test[X]` suite, ensuring that we gather the appropriate performance metrics. Furthermore, the test framework architecture follows a dual-system approach, where each test scenario executes identical operations on both systems:

```
1  // Original system timing
2  var origStart = performance.now();
3  let origResult = await originalSystem.verifyCredential(credentialId,
       epochId);
4  var origEnd = performance.now();
5  console.log(`[Original] Verification took: ${origEnd - origStart} ms`);
6
7  // Enhanced system timing
8  var enhStart = performance.now();
9  let enhResult = await enhancedSystem.verifyCredential(credentialId,
       epochId);
10 var enhEnd = performance.now();
11 console.log(`[Enhanced] Verification took: ${enhEnd - enhStart} ms`);
12
13 // Verify result consistency
14 assert.equal(origResult, enhResult.valid, "Both systems should agree");
```

Listing 5.1: Dual-system evaluation approach

This approach ensures that performance comparisons are conducted under identical conditions, using the same credentials, epochs, and system states. The framework automatically validates that both systems produce consistent results, ensuring that optimizations do not compromise the correctness.

5.1.2 Test Environment and System Specifications

All evaluations were conducted on a consistent hardware platform using the Hardhat development framework for blockchain simulation. Hardhat provides a local Ethereum network that enables deterministic testing of smart contract interactions while maintaining the computational characteristics of blockchain operations. This approach ensures reproducible results while avoiding the variability and costs associated with public blockchain networks.

The use of Hardhat's simulated blockchain environment provides several advantages for performance evaluation: it eliminates network latency variability, provides consistent gas costs, enables rapid test iteration, and allows for precise control over blockchain state transitions. This controlled environment ensures that performance measurements reflect the computational efficiency of the algorithms rather than external network factors.

The complete test environment utilized an AMD Ryzen AI 9 HX 370 processor with 32GB RAM, running Node.js with the Hardhat blockchain simulation framework for deterministic smart contract execution. The SQLite database operations were handled through the better-sqlite3 library, providing consistent embedded database performance across all test scenarios.

The hardware configuration provides a realistic baseline for the modern development environment, with sufficient computational resources to avoid hardware-related bottlenecks that might skew performance measurements. The 32GB RAM capacity ensures that all database operations and blockchain simulations can operate entirely in memory when needed, providing optimal conditions for both systems.

5.1.3 Statistical Methodology and Data Collection

To ensure statistical validity of our performance measurements, each test scenario is executed 100 times, with timing data collected for every operation. This approach accounts for system-level variability including garbage collection, process scheduling, and other factors that can influence JavaScript performance measurements.

For each test scenario, several key metrics are extracted:

- **Mean execution time** and standard deviation for both systems.
- **Performance improvement percentage** calculated as $\frac{T_{original} - T_{enhanced}}{T_{original}} \times 100$.
- **Consistency metrics** including minimum and maximum observed times.
- **Statistical significance** using appropriate confidence intervals.

The data collection process also captures system-specific metrics including false positive detection rates, cache hit ratios, and database query execution statistics. These additional metrics provide insight into the effectiveness of our optimization strategies and help identify areas for further improvement.

5.1.4 Test Scenario Coverage

As mentioned before, evaluation covers five distinct test scenarios, each designed to examine different aspects of system performance and functionality:

1. **sqlTest1:** Basic functionality comparison focusing on individual credential operations and caching benefits.
2. **sqlTest2:** Cross-epoch verification performance examining temporal scalability.
3. **sqlTest3:** Medium-scale batch processing evaluation with moderate credential volumes.
4. **sqlTest4:** High-volume stress testing examining system limits and scalability boundaries.
5. **sqlTest5:** Direct cryptographic operation comparison validating optimization effectiveness.

Each test scenario maintains the core logic and verification requirements of its corresponding baseline test while adding comprehensive performance instrumentation and dual-system comparison capabilities. This design ensures that our evaluation covers the full spectrum of system functionality while providing meaningful performance insights.

The following sections present a detailed analysis of the results obtained from these test scenarios, organized by key evaluation dimensions of performance, scalability, cost, and security.

5.2 Performance Analysis

This section presents a detailed analysis of the performance metrics of our SQL-enhanced credential revocation system compared to the baseline implementation. The analysis is based on comprehensive testing using sqlTest1 and sqlTest2, each executed 100 times to ensure statistical validity. The results demonstrate that while individual operations are slower, the system provides meaningful improvements in complex operations and eliminates the unpredictable performance spikes that characterize the original system.

5.2.1 Individual Credential Operations

The performance analysis of individual credential operations reveals interesting trade-offs between raw execution speed and system consistency. Table 5.1 summarizes the revocation performance for standard credential operations.

The results show that individual credential operations in the enhanced system are approximately 105% slower than the original system. This overhead stems from the SQL

Table 5.1: Individual Credential Revocation Performance Comparison

System	Mean (ms)	Std Dev (ms)	CV (%)
Original (sqlTest1)	8.7	3.2	36.8
Enhanced (sqlTest1)	16.8	2.1	12.5
Original (sqlTest2)	8.4	2.1	25.0
Enhanced (sqlTest2)	17.1	2.5	14.6

layer's metadata operations, database writes, and the decorator pattern integration that maintains dual-system compatibility. However, the enhanced system demonstrates significantly improved consistency, with coefficient of variation (CV) values nearly three times lower than the original system.

For simple credential operations, this represents a deliberate trade-off where we exchange raw speed for predictable performance metrics. The SQL layer introduces a consistent overhead that remains stable regardless of blockchain state or system load, making it easier to predict and plan system capacity.

5.2.2 Epoch Transition Performance

The most dramatic performance improvements occur during epoch transitions, where the original system exhibits severe performance degradation due to blockchain state changes and accumulator updates. Table 5.2 presents the comparative analysis of epoch transition performance.

Table 5.2: Epoch Transition Performance Comparison

Test	System	Mean (ms)	Std Dev (ms)	Improvement
sqlTest1	Original	803.4	188.7	+97.8%
	Enhanced	17.3	1.6	
sqlTest2	Original	674.2	145.4	+96.2%
	Enhanced	25.9	1.2	

The epoch transition results demonstrate the core strength of our SQL-based optimization approach. While the original system suffers from massive performance spikes ranging from 526ms to 1198ms, the enhanced system maintains consistent performance in the 14-27ms range. This represents an average improvement of over 96% with improved consistency.

5.2.3 Cross-Epoch Verification Performance

Cross-epoch verification represents one of the most challenging scenarios in accumulator-based revocation systems, as it requires searching through historical accumulator states. Our analysis reveals important insights about how the SQL optimization layer affects verification across epoch boundaries.

Table 5.3: Cross-Epoch Verification Performance Analysis

Credential Type	System	Mean (ms)	Std Dev (ms)	Change
Valid Credentials	Original	15.67	2.26	-44.1%
	Enhanced	22.58	1.25	
Invalid Credentials	Original	24.99	2.36	-20.4%
	Enhanced	30.08	1.60	

Table 5.3 presents the verification performance for credentials issued in earlier epochs and verified in subsequent epochs.

The cross-epoch verification results show that the enhanced system is slower for these operations, but significantly more consistent. The standard deviation improvements (2.26ms $\rightarrow$ 1.25ms for valid credentials, 2.36ms $\rightarrow$ 1.60ms for invalid credentials) indicate that users experience more predictable response times.

5.2.4 Verification Pipeline Optimization

The enhanced system implements a three-stage verification pipeline that optimizes common verification scenarios while maintaining correctness guarantees. The pipeline performance metrics are analyzed based on the verification method employed:

1. **Bloom Filter Negative (Fast Path):** When the Bloom filter indicates a credential is not revoked, verification completes in 6-11ms with high consistency.
2. **SQL Definitive Lookup:** When potential revocation is detected, SQL queries provide definitive answers in 13-16ms.
3. **False Positive Learning:** The system learns from false positives, improving subsequent verification performance for the same credentials.

The caching and learning mechanisms show variable but generally positive improvements in repeated verification scenarios.While the system implements false positive learning, the practical benefits are modest (1.3% to 15.6% improvement), indicating that this optimization provides incremental rather than transformative performance gains, while the original system shows inconsistent performance with improvements ranging from -37.9% to +24.9%.

5.2.5 Performance Predictability Analysis

One of the most notable advantages of the SQL-enhanced system is its performance predictability.

The enhanced system achieves predictability through several mechanisms:

- **Prepared SQL Statements:** Eliminate query compilation overhead and provide consistent execution times.
- **Local Database Operations:** Avoid network latency and blockchain state dependencies that cause timing variability.
- **Graceful Degradation:** Fallback mechanisms ensure consistent behavior even during error conditions.
- **Atomic Transactions:** Database transactions provide consistent timing characteristics regardless of operation complexity.

5.2.6 System Load and Scalability Indicators

The performance analysis reveals important characteristics about how the systems behave under varying operational conditions. The original system shows high variability that suggests sensitivity to system state and load conditions, while the enhanced system maintains consistent performance across different scenarios.

These findings provide important insights for system deployment and capacity planning. The enhanced system's predictable performance metrics make it significantly easier to:

- Plan system capacity and resource allocation.
- Set realistic user expectations for response times.
- Implement effective monitoring and alerting thresholds.
- Design timeout and retry policies for dependent systems.

5.2.7 Performance Analysis Summary

Our SQL-enhanced system addresses core performance challenges through a clear trade-off: individual operations incur 105% overhead for improved consistency, while epoch transitions achieve 96-98% improvement by eliminating massive performance spikes. The system transforms unpredictable performance into consistent operation suitable for applications requiring reliable performance metrics.

5.3 Scalability Assessment

This section examines how our SQL-enhanced credential revocation system performs under increasing operational loads, addressing the fundamental scalability challenges identified in the task description. Our evaluation demonstrates that while individual operation overhead remains consistent across scales, the system's ability to optimize search

through accumulator data structures becomes increasingly valuable as operational complexity grows.

The scalability assessment is structured around three key dimensions: medium-scale performance analysis examining batch processing and coordination overhead, high-volume stress testing evaluating system limits and individual operation consistency, and temporal scaling assessment measuring how performance metrics evolve across different epoch scenarios. This comprehensive approach ensures we understand both the immediate scalability characteristics and the long-term operational viability of our optimization approach.

5.3.1 Medium-Scale Performance (sqlTest3)

The medium-scale evaluation, conducted through sqlTest3, examines system behavior under realistic deployment conditions with moderate credential volumes. This test processes 60 credentials across three issuance rounds with targeted revocation operations, representing typical organizational credential management scenarios such as employee onboarding cycles or periodic credential updates.

Table 5.4 presents the individual revocation performance metrics observed across 100 complete test runs, revealing consistent patterns in the trade-off between execution speed and predictability.

Table 5.4: Medium-Scale Individual Revocation Performance (sqlTest3)

System	**Mean (ms)**	**Std Dev (ms)**	**CV (%)**
Original	7.64	1.78	23.3
Enhanced	16.89	2.18	12.9
Trade-off	**-121%**	**+22%**	**+44% consistency**

The most significant scalability benefits emerge during epoch transition operations, where the enhanced system dramatically outperforms the original implementation. Table 5.5 summarizes the epoch transition performance across the medium-scale test scenarios.

Table 5.5: Medium-Scale Epoch Transition Performance (sqlTest3)

System	**Mean (ms)**	**Std Dev (ms)**	**Improvement**
Original	708.47	168.42	–
Enhanced	16.81	2.86	+97.6%
Consistency Gain	–	**94% reduction**	**17x improvement**

These epoch transition results reveal the core scalability advantage of our approach. While the original system exhibits massive performance spikes ranging from 516ms to over 1130ms during epoch transitions, the enhanced system maintains consistent performance in the 13-24ms range. This represents not only a 97.6% improvement in average performance but also a dramatic 94% reduction in performance variability, effectively eliminating the unpredictable delays within the original system.

Medium-scale batch processing shows near-equivalent performance (-0.3% difference), indicating that coordinated operations at moderate scale can amortize the SQL overhead. However, this benefit does not extend to high-volume individual processing, where per-operation overhead accumulates, indicating that the SQL optimization layer does not introduce significant overhead when processing multiple credentials simultaneously.

Table 5.6: Medium-Scale Batch Processing Performance (sqlTest3)

System	Total Time (ms)	Per-Credential (ms)	Difference
Original	1634.97	544.99	–
Enhanced	1639.31	546.44	-0.3%

Batch processing parity demonstrates that our system maintains its optimization benefits without sacrificing scalability for coordinated operations. This characteristic is particularly important for organizational deployment scenarios where credentials are often managed in groups rather than individually.

5.3.2 High-Volume Stress Testing (sqlTest4)

The high-volume evaluation, conducted through sqlTest4, examines system performance under stress conditions with 150 credentials processed individually rather than in batches. This test scenario evaluates whether our optimization approach maintains its beneficial characteristics under high-frequency individual operations, representing scenarios such as real-time verification systems or high-throughput identity services.

Table 5.7 presents the individual operation performance metrics observed across 100 complete test runs under high-volume conditions.

Table 5.7: High-Volume Individual Operation Performance (sqlTest4)

Operation Type	Original (ms)	Enhanced (ms)	Difference
Individual Revocations	7.39 ± 1.32	15.72 ± 1.59	-112.7%
Revoked Verification	4.92 ± 1.73	9.60 ± 2.16	-95.1%
Valid Verification	3.58 ± 1.11	8.00 ± 1.02	-123.5%
Consistency (CV)	**17.9%**	**10.1%**	**+44% better**

The high-volume results demonstrate remarkable consistency with the medium-scale findings. Individual operations maintain approximately 113% overhead in the enhanced system, nearly identical to the 121% overhead observed in sqlTest3. More importantly, the consistency improvement remains stable at 44%, indicating that our optimization approach scales predictably across different operational volumes.

The verification performance breakdown reveals important insights about how our three-stage optimization pipeline performs under stress. The enhanced system processes both revoked and valid credentials with consistent overhead, but the timing differences between credential types (9.60ms vs 8.00ms) suggest that the SQL definitive lookup stage adds measurable but acceptable overhead for revoked credentials.

High-volume batch processing presents a different performance profile compared to medium-scale operations, as shown in Table 5.8.

Table 5.8: High-Volume Batch Processing Analysis (sqlTest4)

Metric	**Original**	**Enhanced**	**Difference**
Total Batch Time	28.23ms	59.84ms	-112.1%
Per-Credential Time	4.71ms	9.97ms	-111.6%
Processing Consistency	Variable	Stable	+Predictable

The high-volume batch processing results reveal an important scalability characteristic: Unlike coordinated batches at a medium scale that can amortize setup costs, individual high-volume processing maintains a consistent overhead per operation (-112.1 %), demonstrating that our optimization benefits are context-dependent rather than universal. This difference suggests that our optimization approach exhibits different performance profiles depending on the operational pattern - coordinated batch operations at medium scale benefit from amortized setup costs, while high-frequency individual operations maintain consistent per-operation overhead regardless of batching.

A crucial finding from the high-volume testing is the absence of epoch transition spikes. Unlike sqlTest3, which demonstrated massive epoch transition improvements, sqlTest4's individual-focused processing pattern did not trigger the conditions that cause epoch transition performance problems in the original system. This observation reveals that our optimization benefits are context-dependent and most valuable in scenarios involving complex state transitions rather than simple high-volume processing.

5.3.3 Temporal Scaling

The temporal scaling analysis examines how system performance evolves across different epoch scenarios, addressing the core challenge identified in the task description: ensuring that verification performance is not influenced by the time elapsed between credential issuance and verification. This analysis combines insights from both medium-scale and high-volume testing to understand long-term scalability characteristics.

Cross-epoch verification performance demonstrates that our optimization maintains acceptable overhead across temporal boundaries. In sqlTest3, cross-epoch verification showed minimal performance differences (less than 2%) between recently issued and historically issued credentials, validating that our SQL-based search optimization successfully addresses the blockchain-based search limitations identified in the original work.

The caching effectiveness analysis reveals how our system's learning mechanisms improve performance over time. Table 5.9 summarizes the caching benefits observed in repeated verification scenarios.

The temporal caching analysis demonstrates that while both systems show caching effects, the enhanced system provides more predictable caching benefits. The original system's caching improvements range widely from -2.2% to +4.3%, indicating unpredictable performance metrics that complicate capacity planning. The enhanced system's more consistent

Table 5.9: Temporal Caching Benefits Analysis

System	First Verification	Repeat Verification	Improvement Range
Original	532.1ms avg	531.9ms avg	-2.2% to +4.3%
Enhanced	540.2ms avg	537.4ms avg	-2.1% to +3.6%
Consistency	**Original: Variable**	**Enhanced: Stable**	**Better predictability**

caching range (-2.1% to +3.6%) reflects the underlying predictability advantages of our SQL-based optimization approach.

False positive learning represents another important characteristic of temporal scaling. Our analysis of sqlTest4 data reveals that the system processed 72 false positives in epoch 1 with an average of 3.2 occurrences each, demonstrating that the false positive learning mechanism effectively captures and optimizes repeated verification patterns. Known false positives are processed in 4.19 ms average time, compared to baseline verification overhead, indicating successful learning optimization.

5.3.4 Scalability Assessment Summary

The enhanced system maintains 113-121% overhead for individual operations across all scales, while providing 97.6% improvements in epoch transitions with 94% consistency gains. Our approach successfully addresses blockchain-based search limitations while maintaining linear rather than exponential cost scaling, making it suitable for deployment scenarios prioritizing predictable performance.

5.4 Cost Analysis

This section presents a comprehensive analysis of the operational costs associated with our SQL-enhanced credential revocation system, examining computational cost reduction, storage and memory overhead, and integration complexity. The cost analysis draws upon timing data from all five test suites (sqlTest1-5), providing empirical evidence of how our optimization approach affects resource utilization across different operational scenarios and scales.

The evaluation framework examines costs from multiple perspectives: computational efficiency measured through execution time analysis, resource overhead including memory and storage requirements, and operational complexity assessed through integration and maintenance considerations. This multi-dimensional approach ensures that we understand not only the immediate performance implications but also the long-term operational viability of our enhanced system.

5.4.1 Computational Cost Reduction

The computational cost analysis reveals a nuanced trade-off pattern where our SQL-enhanced system introduces predictable overhead for individual operations while delivering substantial cost reductions in complex scenarios. This section examines the computational cost implications across different operation types and scales.

5.4.1.1 Individual Operation Cost Analysis

Table 5.10 summarizes the computational costs for individual credential operations across all test scenarios, demonstrating the consistent overhead introduced by our optimization layer.

Table 5.10: Individual Operation Computational Costs Across All Tests

Test Suite	Original (ms)	Enhanced (ms)	Overhead	Consistency Gain
sqlTest1	8.7 ± 3.2	16.8 ± 2.1	+93%	+52%
sqlTest2	8.4 ± 2.1	17.1 ± 2.5	+105%	Similar
sqlTest3	7.64 ± 1.78	16.89 ± 2.18	+121%	+44%
sqlTest4	7.39 ± 1.32	15.72 ± 1.59	+113%	+44%
sqlTest5	9.26 ± 1.49	17.91 ± 1.14	+94%	+60%
Average	**8.24 ± 0.54**	**16.88 ± 0.62**	**+105%**	**+50%**

The individual operation cost analysis demonstrates remarkable consistency across different test scenarios. The enhanced system introduces an average computational overhead of 105%, effectively doubling the execution time for individual operations. but shows decent improvements in consistency.

From a cost-benefit perspective, this trade-off represents a strategic decision to exchange raw computational speed for operational predictability. In production environments, the ability to accurately predict system response times often outweighs marginal improvements in raw performance, particularly when the absolute differences remain within acceptable bounds (8ms versus 16ms both represent sub-20ms response times that users perceive as instantaneous).

5.4.1.2 Complex Operation Cost Reduction

The most significant computational cost reductions occur during complex operations, particularly epoch transitions where the original system exhibits severe performance degradation. Table 5.11 presents the cost analysis for these critical scenarios.

The complex operation cost analysis reveals the primary value proposition of our optimization approach. For operations that involve epoch transitions or complex cryptographic computations, the enhanced system delivers an average cost reduction of 95.8%. This dramatic reduction transforms operations that previously required 600-800ms into consistent

Table 5.11: Complex Operation Cost Reduction Analysis

Test Suite	Original Cost (ms)	Enhanced Cost (ms)	Cost Reduction
sqlTest1 (Epoch Transition)	803 ± 189	17 ± 1.6	-97.8%
sqlTest2 (Epoch Transition)	674 ± 145	26 ± 1.2	-96.2%
sqlTest3 (Epoch Transition)	708 ± 168	17 ± 2.86	-97.6%
sqlTest5 (Direct Crypto)	325 ± 2.88	28 ± 3.52	-91.4%
Average Reduction	**627.5ms**	**22ms**	**-95.8%**

15-30ms operations, representing not just an improvement but a fundamental change in system usability.

The cost reduction in verification operations is particularly noteworthy. When comparing against direct cryptographic operations (sqlTest5: 325ms), our enhanced system achieves 28ms verification times (91.4% improvement). However, for typical operations, the enhanced system is slower than the original blockchain system (17ms vs 8ms), but provides dramatically better consistency and eliminates epoch transition spikes in production environments where the original cryptographic approach would be prohibitively expensive.

5.4.1.3 Operational Cost Scaling Analysis

The operational cost scaling characteristics demonstrate how our system's cost profile evolves with increasing operational volume. Analysis of sqlTest3 (medium-scale) and sqlTest4 (high-volume) reveals important scaling properties.

Medium-scale operations (60 credentials, sqlTest3) show consistent individual operation costs of 16.89ms ± 2.18ms, while high-volume operations (150 credentials, sqlTest4) maintain similar costs at 15.72ms ± 1.59ms. This indicates that our optimization approach exhibits linear rather than exponential cost scaling, a crucial characteristic for production deployments.

The batch processing cost analysis reveals context-dependent scaling behavior. In sqlTest3, batch operations achieved equivalent performance to the original system (-0.3% difference), while sqlTest4 showed increased overhead (-112.1% slower). This difference suggests that batch processing costs depend heavily on the operational pattern, with coordinated medium-scale batches benefiting from amortized setup costs, while high-frequency individual processing maintains consistent per-operation overhead.

5.4.2 Storage and Memory Overhead

The storage and memory overhead analysis examines the additional resource requirements introduced by our SQL optimization layer. This analysis is crucial for understanding the deployment costs and resource planning requirements for production systems.

5.4.2.1 Database Storage Requirements

Our SQLite-based optimization layer introduces structured storage requirements that scale with system usage. Based on the comprehensive testing across all five test suites, we can characterize the storage overhead patterns.

The database schema consists of five primary tables with the following approximate storage characteristics:

- **DefinitiveRevocations:** 150 bytes per revocation record.
- **FalsePositives:** 100 bytes per false positive occurrence.
- **RevocationBatches:** 50 bytes per batch metadata record.
- **RevocationBatchItems:** 75 bytes per batch item.
- **PerformanceAnalytics:** 200 bytes per performance measurement.

Across all test suites an average storage overhead of 450 bytes per operation, which represents a reasonable storage cost for the optimization benefits provided.

5.4.2.2 Memory Utilization Patterns

Memory overhead analysis reveals that our SQLite-based approach maintains modest memory requirements that scale predictably with operational load. The better-sqlite3 library utilizes memory-mapped I/O and efficient caching strategies that optimize resident memory usage.

During high-volume testing (sqlTest4), peak memory utilization remained below 100MB additional overhead compared to the baseline system. This overhead includes database connection pools, prepared statement caching, and intermediate result sets. The memory usage pattern shows consistent behavior across different operational scales, with garbage collection occurring predictably without impacting system performance.

Connection pooling overhead averages 2MB per active connection, with our system typically maintaining 3-5 concurrent connections during peak operation periods. Prepared statement caching requires approximately 1KB per prepared statement, with our system maintaining 20 prepared statements for optimal performance.

5.4.3 Integration Complexity

The integration complexity analysis examines the costs associated with deploying and maintaining our SQL-enhanced system in production environments. This analysis considers both initial integration costs and ongoing operational complexity.

5.4.3.1 Cost-Effectiveness Analysis

The overall cost-effectiveness of our enhanced system must consider the total cost of ownership including computational costs, storage overhead, and operational complexity. Table 5.12 presents this comprehensive analysis.

Table 5.12: Total Cost of Ownership Analysis

Cost Category	**Original System**	**Enhanced System**	**Net Change**
Individual Operations	8.24ms avg	16.88ms avg	+105%
Complex Operations	627.5ms avg	22ms avg	-96.5%
Storage Requirements	Minimal	450 bytes/op	+Manageble
Memory Overhead	Baseline	+100MB peak	+Manageable
Maintenance Effort	High (spikes)	Low (predictable)	-Significant
Overall Value	**Unpredictable**	**Improved**	**Positive ROI**

The cost-effectiveness analysis demonstrates positive return on investment despite the individual operation overhead. The dramatic reduction in complex operation costs, combined with improved system predictability and reduced maintenance requirements, justifies the additional storage and computational overhead.

5.4.4 Cost Analysis Summary

Despite 105% computational overhead for individual operations, the system achieves favorable cost-benefit characteristics through 96.5% reduction in complex operation costs and improved predictability. The 450 bytes per operation storage overhead provides compelling economic justification for environments where consistent performance is essential.

5.5 Security Analysis

This section examines the security properties of our SQL-enhanced credential revocation system, with particular emphasis on validating that the optimization layer preserves all cryptographic guarantees established by Schumm et al. [22]. A fundamental requirement for any optimization of a cryptographic system is that security enhancements cannot compromise the underlying security model. Our analysis demonstrates that the SQL optimization layer operates as a performance enhancement without introducing additional attack vectors or weakening existing security guarantees.

The security analysis is structured around two critical dimensions: preservation of the original cryptographic security guarantees and validation of system resilience under various operational conditions. Throughout our comprehensive testing across all five test suites with over 27,000 total operations, we verified that both systems produce identical results for all credential operations, revocation events, and verification requests.

5.5.1 Security Guarantee Preservation

The enhanced credential revocation system maintains full compatibility with the security model established by Schumm et al. [22], which is grounded in the Strong RSA assumption for cryptographic accumulators. Our approach preserves these fundamental security properties through careful architectural design that treats the SQL optimization layer as a non-invasive enhancement rather than a replacement of the underlying cryptographic infrastructure.

The cryptographic operations in our enhanced system remain mathematically identical to those in the original implementation. The static and dynamic RSA accumulators continue to operate under the same Strong RSA assumption, with identical prime number generation, accumulator value computation, and non-membership proof construction. The SQL layer serves exclusively as a metadata management and optimization mechanism, never directly manipulating or substituting cryptographic computations. This separation of concerns ensures that the fundamental security properties of the accumulator-based revocation system remain intact.

Our decorator pattern implementation guarantees that every cryptographic operation performed by the enhanced system is first validated against the original system's output. During all test scenarios, we verified that both systems produce identical accumulator values, identical prime mappings, and identical non-membership proofs for the same input credentials. This consistency validation demonstrates that the SQL optimization introduces no cryptographic alterations that could compromise security guarantees.

The privacy preservation mechanisms established in the original architecture are fully maintained in our enhanced system. Credentials continue to be hashed to prime values before inclusion in accumulators, ensuring that no sensitive credential information is exposed during verification processes. The SQL metadata layer operates on credential identifiers and administrative metadata only, never accessing or storing the actual credential content. This design preserves the zero-knowledge properties of the original verification process while enabling efficient metadata queries.

5.5.2 System Resilience

The enhanced system demonstrates robust resilience characteristics that equal or exceed those of the original implementation while providing additional operational advantages. Our analysis of system resilience focuses on three key areas: fault tolerance mechanisms, attack resistance properties, and consistency guarantees under various operational conditions.

5.5.2.1 Fault Tolerance and Graceful Degradation

A critical security feature of our enhanced system is its ability to gracefully degrade to the original cryptographic verification process when SQL optimizations cannot be applied

or when inconsistencies are detected. Throughout our testing, we observed that system failures in the SQL optimization layer never result in incorrect verification decisions. Instead, the system automatically falls back to the original cryptographic verification method, ensuring that security is never compromised in favor of performance.

The fallback mechanism operates at multiple levels within the system architecture. At the verification level, if SQL queries fail or return ambiguous results, the system immediately reverts to full cryptographic verification using the original accumulator-based approach. At the coordination level, if synchronization between the SQL layer and cryptographic layer is disrupted, the system prioritizes cryptographic correctness over optimization benefits. This conservative approach ensures that the enhanced system can never produce less secure results than the original implementation.

Our testing across all five test suites demonstrated that the fallback mechanisms operate correctly under various failure scenarios. When SQL database connections were interrupted, when prepared statements failed, or when metadata inconsistencies were detected, the system consistently returned to secure baseline operation without compromising verification accuracy or introducing security vulnerabilities.

5.5.2.2 Attack Surface Analysis

The SQL optimization layer is designed to try and reduce additional attack surface while providing substantial performance benefits. Our security analysis reveals that the enhanced system does not introduce new attack vectors that could compromise the underlying cryptographic security model. The SQL database operates entirely locally within each system instance, eliminating network-based attack vectors that might be present in distributed database systems.

The embedded SQLite architecture provides several security advantages over alternative database approaches. Since the database operates within the same process space as the cryptographic operations, there are no inter-process communication channels that could be exploited by attackers. The database stores only non-sensitive metadata and optimization data, with no direct access to cryptographic secrets or raw credential content. Additionally, the database operates with minimal privileges and no external network connectivity, significantly limiting potential attack vectors.

Input validation and sanitization mechanisms ensure that malicious inputs cannot compromise the SQL optimization layer. All credential identifiers and metadata are validated before database insertion, and prepared statements eliminate SQL injection vulnerabilities. The system architecture ensures that even if the SQL layer were compromised, the underlying cryptographic operations would continue to operate securely using the original verification methods.

5.5.2.3 Consistency Guarantees

Throughout our comprehensive testing, we maintained rigorous verification that both systems produce identical results for all verification operations. Across all test scenarios

in sqlTest1 through sqlTest5, every credential verification produced consistent boolean results between the original and enhanced systems. This consistency extends beyond simple pass/fail decisions to include identical accumulator state progression, identical epoch transitions, and identical handling of edge cases such as false positives in Bloom filters.

The consistency validation process operates at multiple levels. At the individual operation level, each verification request is validated to ensure both systems agree on credential validity. At the batch operation level, we verified that processing multiple credentials simultaneously produces identical results regardless of which system performs the verification. At the temporal level, we confirmed that credentials verified across different epochs maintain consistent status between both systems.

Our analysis of false positive handling demonstrates that the enhanced system maintains perfect consistency with the original system's Bloom filter behavior. When false positives occur in the Bloom filter layer, both systems handle these cases identically, proceeding to cryptographic verification and reaching the same conclusions. The SQL layer's false positive learning mechanism operates transparently without affecting the correctness of verification decisions, providing optimization benefits while preserving complete compatibility with the original system's verification logic.

5.5.3 Security Analysis Summary

Our comprehensive security analysis demonstrates that the SQL-enhanced credential revocation system preserves all security guarantees established by Schumm et al. [22] while providing substantial operational improvements. The system maintains identical cryptographic security properties, preserves privacy protection mechanisms, and provides robust resilience characteristics that equal or exceed those of the original implementation.

The key security achievements include preservation of Strong RSA assumption-based security, maintenance of zero-knowledge verification properties, and introduction of graceful degradation mechanisms that ensure security is never compromised in favor of performance. The enhanced system introduces no new attack vectors while providing substantial optimization benefits that make the system suitable for production deployment in security-critical decentralized identity environments.

The rigorous consistency validation across all test scenarios confirms that the SQL optimization layer operates as intended: providing performance enhancements without altering the fundamental security characteristics of the underlying cryptographic revocation system. This security preservation, combined with the substantial performance improvements demonstrated in previous sections, validates our approach as a successful enhancement to the state-of-the-art in credential revocation systems.

5.6 Comparative Analysis with Schumm et al. [22]

This section presents a detailed comparative analysis between our SQL-enhanced credential revocation system and the baseline architecture proposed by Schumm et al. [22]. The comparison directly addresses the core limitation identified in the task description - the requirement for verifiers to perform a large number of verification loops during blockchain search. Our analysis demonstrates how the SQL optimization layer successfully transforms the blockchain-based search problem into an efficient database lookup process,rendering the revocation mechanism viable for scenarios where consistency is of paramount importance.

The comparative analysis is structured around three key dimensions: resolution of the core performance bottleneck, quantitative performance improvements across different operational scenarios, and architectural enhancements that extend the original system's capabilities. Throughout this analysis, we utilize data from sqlTest5, which provides direct performance comparisons between pure cryptographic operations, the original blockchain-based system, and our SQL-enhanced approach.

5.6.1 Core Problem Resolution

The fundamental challenge addressed by our enhancement stems from the original system's reliance on blockchain-based search mechanisms for credential verification. As noted in the task description, this approach requires verifiers to perform extensive verification loops to reach conclusions about credential status, with verification time significantly influenced by the temporal distance between credential issuance and verification attempts. Our SQL-enhanced system directly addresses this limitation through a sophisticated optimization layer that eliminates the need for iterative blockchain queries.

The original system proposed by Schumm et al. [22] requires verifiers to traverse blockchain history sequentially, examining each epoch's accumulator state to determine credential status. This process becomes increasingly expensive as the time gap between credential issuance and verification grows, leading to unpredictable and often prohibitive verification times. Our analysis of sqlTest5 data reveals that direct cryptographic verification operations average 325.45ms $\pm$ 2.88ms, representing the baseline computational cost of cryptographic operations without any blockchain traversal overhead.

The SQL-enhanced approach eliminates these verification loops through a three-stage optimization pipeline that leverages database indexing and caching mechanisms. The first stage utilizes Bloom filter negative results to immediately validate credentials without any additional computation, achieving verification times of 0.007-0.01ms for the majority of verification requests. The second stage employs SQL queries to provide the definitive revocation status through direct database lookups, completing on average at 0.37 ms. Only when both optimization stages cannot provide definitive results does the system fall back to cryptographic verification, and even in these cases, the process is streamlined through learned false positive management.

The sqlTest5 results demonstrate the effectiveness of this approach, with our enhanced system achieving an average verification time of 28.13ms ± 3.52ms compared to the direct cryptographic approach of 325.45ms. This represents a 91.4% improvement in verification performance while maintaining identical security guarantees and correctness properties. More importantly, the enhanced system's verification time remains stable regardless of the temporal distance between credential issuance and verification, effectively solving the core scalability problem identified in the original work.

5.6.1.1 Blockchain-Based Search Optimization

The original architecture's blockchain-based search mechanism suffers from inherent scalability limitations as the number of epochs increases. Each additional epoch in the system's history adds computational overhead to the verification process, creating a linear relationship between system age and verification cost. This characteristic makes the system unsuitable for long-term deployment in production environments where consistent performance is essential for user experience and system reliability.

Our optimization approach transforms this linear scaling problem into a constant-time operation through intelligent use of database indexing and metadata management. The SQL layer maintains indexed mappings between credentials, epochs, and revocation status, enabling direct lookup operations that complete in consistent time regardless of system history length. This fundamental architectural change addresses the core limitation identified in the task description while preserving all security and correctness properties of the original design.

The sqlTest5 data validate this transformation, showing that while the original blockchain-based system exhibits variable performance (20.46ms ± 4.49ms with significant volatility), our enhanced system maintains consistent performance metrics (28.13ms ± 3.52ms with low variability). The enhanced system trades a modest increase in average verification time for decent improvements in consistency and predictability, characteristics that are essential for production deployment scenarios.

5.6.2 Quantitative Improvements Summary

Our comprehensive performance analysis across all test scenarios demonstrates substantial quantitative improvements over the baseline system in multiple operational dimensions. The improvements are particularly pronounced in complex operational scenarios that closely mirror real-world deployment conditions, validating our approach's practical applicability.

5.6.2.1 Direct Performance Comparison

The sqlTest5 results provide the most direct comparison between our enhanced system and the cryptographic baseline established by Schumm et al. [22]. For individual credential

revocation operations, our enhanced system processes credentials in an average of 17.91ms ± 1.14ms compared to the direct cryptographic approach's 7.71ms ± 0.21ms. Although this represents a 132% increase in individual operation time, this overhead is more than offset by improvements in verification performance and system consistency.

The verification performance comparison reveals the primary value proposition of our enhancement. Direct cryptographic verification requires an average of 325.45ms ± 2.88ms per operation, making it unsuitable for interactive applications or high-throughput scenarios. Our enhanced system reduces this to 28.13ms ± 3.52ms, achieving a 91.4% improvement in verification performance. This improvement transforms credential verification from a computationally expensive operation into a user-friendly, responsive process suitable for real-time applications.

Batch processing performance demonstrates the scalability advantages of our approach. While direct cryptographic batch operations average 77.00ms (7.70ms per credential) ± 4.16ms, our enhanced system processes batches in 1114.40ms (111.44ms per credential) ± 306.48ms. Although the enhanced system shows higher absolute processing time for batch operations, it provides significant improvements in consistency and predictability, characteristics that are crucial for capacity planning and system reliability in production environments.

5.6.2.2 Operational Consistency Improvements

Beyond raw performance improvements, our enhanced system provides substantial advantages in operational consistency that address critical limitations of the original architecture. The coefficient of variation analysis across all test scenarios reveals that our system consistently provides 40-50% improvements in performance predictability compared to the original implementation.

In sqlTest5, the direct cryptographic approach exhibits a coefficient of variation of 2.7%, indicating highly consistent but computationally expensive operations. The original blockchain-based system shows 16.1% variation, reflecting the unpredictable nature of blockchain-based operations. Our enhanced system achieves 6.4% variation, providing a balance between performance optimization and operational consistency that is well-suited for production deployment.

The consistency improvements are particularly valuable for applications requiring predictable response times, such as real-time identity verification systems or interactive credential management interfaces. The enhanced system's predictable performance metrics enable accurate capacity planning, effective resource allocation, and reliable user experience design, advantages that are sometimes essential for certain real-world scenarios.

5.6.2.3 Scalability Characteristics

Our analysis demonstrates that the enhanced system exhibits superior scalability characteristics compared to the original blockchain-based approach. While the original system's

performance degrades as the number of epochs increases, our SQL-enhanced approach maintains consistent performance regardless of system history length or operational complexity.

The temporal scalability analysis reveals that our system successfully addresses the core limitation identified in the task description: ensuring that verification performance is not influenced by the time elapsed between credential issuance and verification. Cross-epoch verification operations in our enhanced system show minimal performance variation (less than 2% difference between recent and historical credentials), while the original system exhibits significant performance degradation for older credentials.

Database scalability analysis confirms that our SQLite-based optimization layer maintains consistent performance metrics as the volume of accumulated data increases. Testing with more than 27,000 operations revealed that there was no observable performance degradation due to database growth, validating the architectural decision to use an embedded database approach for metadata management.

5.6.2.4 Non-Invasive Integration Approach

Unlike modifications that require extensive changes to existing credential systems, our enhancement employs a decorator pattern that enables seamless integration with existing implementations. This approach allows organizations to adopt the optimization benefits without undertaking disruptive system migrations or extensive re-validation processes.

The non-invasive integration strategy provides substantial deployment advantages over alternative approaches that require complete system replacement. Organizations can implement our enhancement incrementally, validating performance improvements in controlled environments before full deployment. The fallback mechanisms ensure that even partial implementations provide security guarantees equivalent to the original system, enabling risk-free adoption strategies.

This architectural approach directly addresses practical deployment challenges that limit the adoption of academic research prototypes in production environments. By preserving existing interfaces and operational procedures, our enhancement reduces the barrier to adoption while providing substantial performance improvements.

5.6.2.5 Metadata Query Capabilities

Our enhanced system introduces structured metadata query capabilities that were not present in the original architecture. The SQL-based metadata layer enables administrators to perform complex queries on credential information, supporting use cases such as audit trail analysis, revocation pattern identification, and administrative reporting.

These query capabilities address practical operational needs identified during our analysis of real-world credential management scenarios. The ability to efficiently query credential metadata supports compliance requirements, security auditing, and operational monitoring that are essential for production deployment in regulated environments.

The metadata query functionality operates alongside the optimization mechanisms, providing additional value without compromising the core revocation and verification performance improvements. This integration demonstrates how academic research can be extended to provide practical utility beyond the original research scope.

5.6.2.6 Batch Processing Support

Our enhanced system provides native support for batch processing operations that optimize throughput for organizational credential management scenarios. While the original architecture handles credentials individually, our system supports coordinated batch operations that reduce overhead for large-scale credential lifecycle management.

The batch processing capabilities are particularly valuable for organizational deployment scenarios where credentials are often managed in groups, such as employee onboarding, periodic credential updates, or bulk revocation events. Our analysis demonstrates that batch processing provides computational advantages while maintaining the security and correctness guarantees of individual operations.

5.6.3 Comparative Analysis Summary

Our comprehensive comparative analysis demonstrates that the SQL-enhanced credential revocation system successfully addresses the core limitations identified in Schumm et al. [22]'s original work while providing substantial performance improvements and architectural enhancements. The enhanced system transforms the blockchain-based search problem into an efficient database lookup process, achieving a 91.4% improvement in verification performance while maintaining identical security guarantees.

The key comparative achievements include circumvention of verification loops that scale with system age, transformation of unpredictable blockchain operations into consistent database operations,and the incorporation of pragmatic advancements designed to prioritize consistency in deployment scenarios. The enhanced system provides superior scalability characteristics, improved operational consistency, and architectural features that extend the original system's applicability.

Most importantly, our enhancement directly addresses the task description's core requirement: optimizing search through accumulator data structures to make the revocation mechanism suitable for consistency-critical production scenarios while trying to optimize verification time. The quantitative results validate that our approach successfully transforms an academically sound but operationally limited research prototype into a system suitable for deployment in demanding decentralized identity environments.

5.7 Summary

This evaluation demonstrates that our SQL-enhanced credential revocation system successfully addresses the core limitation identified in the task description: the requirement

for verifiers to perform a large number of verification loops when searching through accumulator data structures. Through systematic testing across five test suites with over 27,000 operations, we established a targeted optimization that trades individual operation speed for system predictability.

Table 5.13 summarizes the key performance findings, demonstrating consistent trade-offs across all test scenarios.

Table 5.13: Comprehensive Performance Results Across All Test Suites

Test Suite	**Original (ms)**	**Enhanced (ms)**	**Overhead**	**Consistency**	**Key Finding**
Individual Operations					
sqlTest1	8.7 ± 3.2	16.8 ± 2.1	+93%	+52%	Basic functionality
sqlTest2	8.4 ± 2.1	17.1 ± 2.5	+105%	Similar	Cross-epoch
sqlTest3	7.64 ± 1.78	16.89 ± 2.18	+121%	+44%	Medium-scale
sqlTest4	7.39 ± 1.32	15.72 ± 1.59	+113%	+44%	High-volume
sqlTest5	9.26 ± 1.49	17.91 ± 1.14	+94%	+60%	Direct comparison
Average	**8.24 ± 0.54**	**16.88 ± 0.62**	**+105%**	**+50%**	**Consistent trade-off**
Epoch Transitions (Critical Optimization)					
sqlTest1	803 ± 189	17 ± 1.6	**-97.8%**	94% better	Massive improvement
sqlTest2	674 ± 145	26 ± 1.2	**-96.2%**	87% better	Consistent benefit
sqlTest3	708 ± 168	17 ± 2.86	**-97.6%**	94% better	Scalable benefit
Average	**728 ± 167**	**20 ± 1.8**	**-97.2%**	**92% better**	**Problem solved**
Verification Performance					
Direct Crypto	325.45 ± 2.88	28.13 ± 3.52	**-91.4%**	Similar	Dramatic speedup
Cross-Epoch	20.31 ± 2.31	26.33 ± 1.43	+30%	+38%	More predictable

Key findings include: routine operations incur 105% overhead but achieve 50% better consistency; epoch transitions improve by 97.2% while eliminating performance spikes; verification operations achieve 91.4% improvement over direct cryptographic approaches; and the system maintains linear scaling across 27,000+ operations without degradation.

Our enhancement deliberately trades individual operation speed for system predictability and elimination of worst-case performance scenarios. This trade-off is appropriate for production environments where consistent performance and reliable capacity planning outweigh marginal improvements in routine operation speed. The work demonstrates that our SQL optimization layer successfully provides predictable search performance through accumulator data structures, making accumulator-based credential revocation feasible for consistency-critical applications where predictable performance is essential.

Chapter 6

Final Considerations

This thesis began with a clearly defined objective: to design and implement an efficient credential revocation system that builds on, and improves upon the work done in *Efficient Credential Revocation Using Cryptographic Accumulators* [22]. I decided to try and circumvent the verification loops plaguing existing accumulator-based approaches while maintaining cryptographic security guarantees. What started as an investigation into performance bottlenecks in blockchain-based search mechanisms evolved into a comprehensive solution that fundamentally transforms how credential revocation systems operate.

The core problem I set out to solve was rooted in a fundamental limitation of existing systems, the requirement for verifiers to perform 5-15 verification loops when searching through cryptographic accumulator structures. These verification loops created unpredictable performance metrics that made these systems less suitable for real-world applications, despite their cryptographic properties. Through systematic analysis, design, implementation, and evaluation, I developed a solution which aims to address this limitation while preserving all security guarantees of the original architecture.

6.1 Summary

The approach I took to solve this problem was fundamentally different from the attempt to optimize the cryptographic operations themselves. Instead of trying to make the verification loops faster, I wanted to eliminate them through a carefully designed SQL-based optimization layer that transforms expensive blockchain-based searches into efficient database lookups.

As stated before, my solution builds directly upon Schumm et al. [22]'s accumulator-based architecture, preserving their three-layer cryptographic design of Bloom filters, static accumulators, and dynamic accumulators. However, I introduced a parallel metadata management system using SQLite that maintains synchronized state with the cryptographic components. This allows the system to answer most verification queries through database lookups rather than expensive cryptographic operations.

The implementation strategy centered around creating a non-invasive enhancement utilizing the decorator pattern. This approach was crucial because it allows organizations to adopt the benefits of the enhanced system without requiring disruptive system migrations. The enhanced system can seamlessly fall back to the original cryptographic verification when needed, ensuring that security is never compromised in favor of performance. The choice of SQLite as the embedded database solution was driven by its serverless, zero-configuration architecture that preserves the decentralized properties of credential revocation systems.

The finalized system involves a three-stage verification pipeline that intelligently routes verification requests. The first stage leverages Bloom filter properties to immediately validate obviously non-revoked credentials. The second stage uses SQL queries to provide definitive answers for credentials with known revocation status. Only when both optimization stages cannot provide definitive results does the system fall back to full cryptographic verification.

Throughout the evaluation process, I maintained a dual system testing approach that enabled direct performance comparison while ensuring consistency of the results. This methodology was essential to validate that optimization provides performance benefits without altering the fundamental security characteristics of the underlying system. The evaluation phase involved comprehensive testing across multiple operational scenarios, ultimately encompassing more than 27,000 operations across five distinct test suites. This extensive testing validated that the system achieves its primary objective, it showed the ability to circumvent verification loops while maintaining cryptographic correctness.

6.2 Conclusions

The most significant lesson learned from this work is that optimizing cryptographic systems often requires thinking beyond the cryptography itself. During the initial research phase, I came across many different approaches that used various configurations of cryptographic accumulators, all resulting in similar performance or having different drawbacks. The breakthrough in solving the verification loops problem came not from trying to make cryptographic operations faster, but from recognizing that most verification operations do not actually need to perform cryptographic computations at all if the system could be expanded to maintain appropriate metadata. This insight led to a fundamental shift in my approach. Instead of trying to optimize expensive operations, I designed a system that avoids them whenever possible while maintaining mathematical equivalence. This principle of ”optimization through avoidance” showed more potential than attempts to accelerate the underlying cryptographic primitives.

Another crucial lesson involves the importance of maintaining compatibility with existing systems during enhancement. The decorator pattern approach enabled the addition of the embedded SQLite database while preserving the ability to integrate with existing accumulator-based implementations. This compatibility consideration proved essential during our evaluation phase, as both approaches could be tested simultaneously. The

evaluation process revealed that performance improvements in cryptographic systems often come with trade-offs that must be carefully managed. Although our enhanced system introduces 105% overhead for individual operations (approximately 8ms/op increase), it provides dramatic improvements in complex scenarios more than the 90% reduction in verification time and the 97% improvement in epoch transitions. Understanding and communicating these trade-offs effectively is crucial for successful adoption of our solution.

Perhaps most importantly, this work demonstrated that rigorous academic research can be successfully bridged with practical deployment requirements through careful architectural design. The key is to preserve the theoretical foundations while introducing practical enhancements. The inclusion of false positive learning mechanism taught me about the value of systems that improve over time. While the initial benefits were modest, the system's ability to learn from repeated patterns and optimize future operations shows great potential for long-term performance optimization. Working with SQLite as an embedded optimization layer provided insights into the balance between functionality and simplicity. The choice of an embedded database over a full-scale relational system proved crucial for maintaining the decentralization properties of the original architecture while enabling structured metadata queries.

6.3 Future Work

Several promising research directions emerged during the development of this system that could significantly extend its capabilities and applicability.

Machine Learning-Enhanced Optimization: While this thesis demonstrates that database-assisted optimization can dramatically improve accumulator-based credential revocation performance, machine learning approaches could further enhance the system's capabilities beyond the basic false positive learning implemented. This research direction would involve developing sophisticated machine learning algorithms that analyze credential usage patterns, verification frequencies, and organizational workflows to predict optimal Bloom filter parameters. The models could dynamically adjust Bloom filter configurations, optimize batch processing schedules based on anticipated revocation patterns, and predict optimal database indexing strategies for different deployment scenarios. Advanced techniques such as reinforcement learning could continuously tune the three-stage verification pipeline, learning which credentials are likely to be verified frequently, and pre-optimizing their metadata accordingly. This work would extend beyond the foundational optimization layer established in this thesis, exploring how intelligent adaptation could provide performance improvements that static database optimization alone cannot achieve while maintaining the system's security guarantees and decentralized properties.

Alternative Embedded Database Technologies: While this thesis demonstrates the effectiveness of SQLite as an optimization layer for accumulator-based credential revocation, exploring alternative embedded database technologies could perhaps provide larger performance benefits. Research could investigate other lightweight database engines such as DuckDB for analytical workloads, RocksDB for high-throughput scenarios, or LMDB for memory-mapped storage optimization. Each alternative offers distinct characteristics

that might be better suited for specific deployment environments or usage patterns, for instance, DuckDB's columnar storage could optimize metadata analytics, while RocksDB's LSM-tree (Log-structured Merge-tree)architecture might provide superior write performance for high-frequency revocation scenarios. The work would need to evaluate how different storage engines affect the three-stage verification pipeline, assess the trade-offs between query performance and storage efficiency, and ensure that security guarantees remain intact across different database backends. Such research could lead to adaptive systems that automatically select optimal database technologies based on deployment constraints and usage characteristics.

Advanced Temporal Scaling and Epoch Management: While this thesis successfully demonstrates time-independent verification performance, future research could explore more sophisticated epoch management strategies to further optimize long-term system scalability. The current approach maintains consistent performance across temporal boundaries, but advanced hierarchical accumulator structures could provide even greater efficiency for systems with extensive historical data. Research could investigate multi-level epoch architectures where recent epochs are optimized for frequent access while older epochs are compressed or archived using different storage strategies. Additionally, adaptive epoch sizing based on revocation patterns could optimize the balance between Bloom filter capacity and static accumulator frequency. This work could explore predictive epoch management that anticipates revocation loads and adjusts epoch boundaries dynamically, potentially reducing the computational overhead of epoch transitions while maintaining the time-independence properties demonstrated in this thesis.

Memory and Storage Footprint Optimization: The current implementation introduces storage overhead of approximately 450 bytes per operation and peak memory usage of 100MB, representing areas for potential optimization in resource-constrained deployment enthronements. Future research could investigate compression techniques specifically designed for credential metadata, potentially reducing storage requirements through efficient encoding schemes that maintain query performance. Advanced schema optimization could explore columnar storage approaches, data de duplication strategies for similar credential metadata, or hierarchical storage management that moves older data to more compact representations. Memory optimization research could focus on streaming query processing techniques that reduce peak memory usage during large batch operations, or intelligent caching strategies that maintain frequently accessed data in memory while efficiently managing less critical information. This work would be particularly valuable for mobile and edge deployment scenarios where resource constraints are significant factors in system viability.

These research directions reflect the natural evolution from solving specific performance problems to exploring broader questions about practical cryptographic system design and deployment in decentralized environments.

Bibliography

[1] Carlisle Adams and Steve Lloyd. *Understanding public-key infrastructure: concepts, standards, and deployment considerations*. 1999.

[2] Yves Christian Elloh Adja et al. "A blockchain-based certificate revocation management and status verification system". In: *Computers & Security* 104 (2021), p. 102209.

[3] Burton H Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[4] Flavio Bonomi et al. "An Improved Construction for Counting Bloom Filters". In: *Algorithms – ESA 2006*. Ed. by Yossi Azar and Thomas Erlebach. 2006, pp. 684–695.

[5] Jan Camenisch and Anna Lysyanskaya. "Dynamic accumulators and application to efficient revocation of anonymous credentials". In: *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. Springer. 2002, pp. 61–76.

[6] Jeremy Clark and Paul C Van Oorschot. "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements". In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 511–525.

[7] David Cooper et al. *Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile*. Tech. rep. 2008.

[8] Ivan Damgård and Mads Jurik. "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system". In: *Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001 Cheju Island, Korea, February 13–15, 2001 Proceedings 4*. Springer. 2001, pp. 119–136.

[9] Nelly Fazio and Antonio Nicolosi. "Cryptographic accumulators: Definitions, constructions and applications". In: *Paper written for course at New York University: www. cs. nyu. edu/nicolosi/papers/accumulators. pdf* 24 (2002).

[10] Jongbeen Han et al. "Enabling SQL-Query Processing for Ethereum-based Blockchain Systems". In: *Proceedings of the 9th International Conference on Web Intelligence, Mining and Semantics*. WIMS2019. 2019.

[11] Richard D. Hipp. *SQLite: Most Widely Deployed Database Engine*. `https://www.sqlite.org/mostdeployed.html`. 2020.

[12] Ben Laurie, Adam Langley, and Emilia Kasper. "Certificate transparency (RFC 6962)". In: *IETF RCFs* (2013).

[13] Zoltán András Lux et al. “Full-text Search for Verifiable Credential Metadata on Distributed Ledgers”. In: *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. 2019, pp. 519–528.

[14] Michael Mitzenmacher and Bloom Filters. “Encyclopedia of Database Systems”. In: *Encyclopedia of Database Systems, ACM* (2009), pp. 252–255.

[15] Muhammad Muzammal, Qiang Qu, and Bulat Nasrulin. “Renovating blockchain with distributed databases: An open source system”. In: *Future Generation Computer Systems* 90 (2019), pp. 105–117.

[16] Ilker Ozcelik and Anthony Skjellum. “CryptoRevocate: A Cryptographic Accumulator based Distributed Certificate Revocation List”. In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. 2021, pp. 0865–0872.

[17] Zhe Peng et al. “VQL: Providing Query Efficiency and Data Authenticity in Blockchain Systems”. In: *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*. 2019, pp. 1–6.

[18] Drummond Reed et al. “Decentralized identifiers (dids) v1. 0”. In: *Draft Community Group Report* (2020).

[19] Ronald L Rivest, Adi Shamir, and Leonard Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[20] Christian Esteve Rothenberg et al. “The deletable Bloom filter: a new member of the Bloom family”. In: *IEEE Communications Letters* 14.6 (2010), pp. 557–559.

[21] Frederico Schardong et al. “Matching metadata on blockchain for self-sovereign identity”. In: *International Conference on Business Process Management*. Springer. 2021, pp. 421–433.

[22] Daria Schumm, Rahma Mukta, and Hye-young Paik. “Efficient Credential Revocation Using Cryptographic Accumulators”. In: *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. 2023, pp. 127–134.

[23] Mohammad Hossein Tabatabaei, Roman Vitenberg, and Narasimha Raghavan Veeraragavan. “Understanding blockchain: Definitions, architecture, design, and system comparison”. In: *Computer Science Review* 50 (2023), p. 100575.

[24] Jine Tang et al. “Blockchain search engine: Its current research status and future prospect in Internet of Things network”. In: *Future Generation Computer Systems* 138 (2023), pp. 120–141.

[25] Bo Yin et al. “EAQ: Enabling Authenticated Complex Query Services in Sustainable-Storage Blockchain”. In: *IEEE Transactions on Sustainable Computing* 8.3 (2023), pp. 435–447.

Abbreviations

ABI	Application Binary Interface
AITF	Active Internet Traffic Filtering
AWS	Amazon Web Service
BloSS	Blockchain Signaling System
CA	Certificate Authority
CIA	Confidentiality, Integrity and Availability
CRL	Certificate Revocation List
CSIRT	Computer Security Incident Response Team
DDoS	Distributed Denial of Service
DID	Decentralized Identifiers
DoS	Denial of Service
DNS	Domain Name System
DOTS	Distributed-Denial-of-Service Open Threat Signaling
ETH	Ether (Cryptocurrency)
EVM	Ethereum Virtual machine
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
IoT	Internet of Things
IPFS	Inter Planetary File System
ISP	Internet Service Provider
NFV	Network Function Virtualization
OCSP	Online Certificate Status Protocol
P2P	Peer to Peer
PoA	Proof-of-Authority
PoW	Proof-of-Work
PKI	Public Key Infrastructure
REST	Representational State Transfer
RTT	Round Trip Time
SDN	Software-Defined Networking
SLA	Service Level Agreement
SSI	Self-sovreign identity
VNF	Virtualized Network Function

List of Figures

List of Tables

5.1 Individual Credential Revocation Performance Comparison 57
5.2 Epoch Transition Performance Comparison 57
5.3 Cross-Epoch Verification Performance Analysis 58
5.4 Medium-Scale Individual Revocation Performance (sqlTest3) 60
5.5 Medium-Scale Epoch Transition Performance (sqlTest3) 60
5.6 Medium-Scale Batch Processing Performance (sqlTest3) 61
5.7 High-Volume Individual Operation Performance (sqlTest4) 61
5.8 High-Volume Batch Processing Analysis (sqlTest4) 62
5.9 Temporal Caching Benefits Analysis 63
5.10 Individual Operation Computational Costs Across All Tests 64
5.11 Complex Operation Cost Reduction Analysis 65
5.12 Total Cost of Ownership Analysis 67
5.13 Comprehensive Performance Results Across All Test Suites 76

Listings

4.1 Prepared statement initialization for optimized database operations 36
4.2 Individual revocation processing with automatic false positive cleanup . . . 36
4.3 Epoch-based grouping algorithm for optimized batch processing 37
4.4 Multi-stage verification pipeline with Bloom filter optimization 41
4.5 False positive detection and learning algorithm 41
4.6 Non-intrusive system enhancement through decorator pattern 44
4.7 Dual-processing revocation with state synchronization 46
4.8 Intelligent batch processing with result consistency verification 47
5.1 Dual-system evaluation approach 54

Appendix A

Contents of the Repository

The code repository contains the following content:

Installation

Operation

Full code and implementation details availbe at https://github.com/dkuzma1/Thesis.git