Ignacio Miguel Rodríguez

A Novel Approach in Certificate Revocation and Validation Using Bitcoin's UTXO Model

Master's Degree in Computer Security Engineering and Artificial Intelligence

Final Master's Project

Co-Directed by
Dr. Jordi Castellà Roca and Dr. Alexandre Viejo

School of Engineering

UNIVERSITAT ROVIRA i VIRGILI

Tarragona, June 2025

Contents

List of Figures

List of Tables

Resum

En l'era digital actual, la comunicació segura en línia és indispensable, especialment perquè grans volums d'informació sensible es transfereixen diàriament a través d'Internet. La Infraestructura de Clau Pública (PKI) juga un paper clau en la protecció d'aquestes comunicacions en vincular identitats amb claus públiques mitjançant certificats digitals. Tot i això, garantir l'autenticitat d'aquests certificats amb el pas del temps requereix mecanismes de revocació robustos. Els mecanismes tradicionals, com les Llistes de Revocació de Certificats (CRLs) i el Protocol d'Estat de Certificats en Línia (OCSP), presenten reptes importants, incloent-hi problemes d'escalabilitat i privadesa i costos operatius.

En aquest treball, explorem un nou enfocament descentralitzat per a la revocació de certificats aprofitant la cadena de blocs de Bitcoin. En concret, proposem incrustar un identificador únic d'una sortida de transacció no gastada (UTXO) dins d'un camp d'extensió personalitzat del certificat X.509 v3. Un certificat serà vàlid mentre el UTXO corresponent no hagi estat gastat. Per revocar el certificat, el UTXO es gasta mitjançant una transacció estàndard de Bitcoin, la qual pot ser detectada per qualsevol verificador consultant la cadena de blocs. Aquest mètode elimina la dependència de servidors de validació externs, millora la privadesa de l'usuari i vincula la validesa del certificat a una infraestructura pública, verificable, transparent i descentralitzada.

La nostra proposta contribueix a un model de PKI descentralitzat més segur i eficient, abordant les limitacions clau dels sistemes de revocació existents i demostrant la viabilitat de les solucions basades en blockchain en aplicacions de seguretat reals.

Paraules clau— Validació de certificats, mozilla, UTXO, bitcoin, blockchain

Resumen

En la era digital actual, la comunicación segura en línea es indispensable, especialmente debido a que grandes volúmenes de información sensible se intercambian diariamente a través de Internet. La Infraestructura de Clave Pública (PKI) desempeña un papel fundamental en la protección de estas comunicaciones al vincular identidades con claves públicas mediante certificados digitales. Sin embargo, garantizar la autenticidad de estos certificados a lo largo del tiempo requiere mecanismos de revocación robustos. Los enfoques tradicionales, como las Listas de Revocación de Certificados (CRLs) y el Protocolo de Estado de Certificados en Línea (OCSP), presentan desafíos importantes, incluyendo problemas de escalabilidad y privacidad y costes operativos.

En este trabajo, exploramos un enfoque novedoso y descentralizado para la revocación de certificados aprovechando la cadena de bloques de Bitcoin. Específicamente, proponemos incrustar un identificador único de una salida de transacción no gastada (UTXO) en un campo de extensión personalizado del certificado X.509 v3. Un certificado será válido mientras el UTXO correspondiente no haya sido gastado. Para revocar el certificado, el UTXO se gasta mediante una transacción estándar de Bitcoin, lo cual puede ser verificado por cualquier parte interesada consultando la cadena de bloques. Este método elimina la dependencia de servidores de estado externos, mejora la privacidad del usuario y vincula la validez del certificado a una infraestructura pública, transparente y descentralizada.

Nuestra propuesta contribuye a un modelo de PKI descentralizado más seguro y eficiente, abordando las principales limitaciones de los sistemas de revocación existentes y demostrando la viabilidad de soluciones basadas en blockchain en aplicaciones de seguridad del mundo real.

Palabras clave— Validación de certificados, mozilla, UTXO, bitcoin, blockchain.

Abstract

In today's digital era, secure online communication is indispensable, particularly as large volumes of sensitive information are exchanged over the Internet every day. The Public Key Infrastructure (PKI) plays a critical role in safeguarding these communications by binding identities to public keys through digital certificates. However, ensuring the authenticity of these certificates over time requires robust revocation mechanisms. Traditional approaches such as Certificate Revocation Lists (CRLs) and the Online Certificate Status Protocol (OCSP) present significant challenges, including scalability issues, privacy concerns, and operational costs.

In this work, we explore a novel decentralized approach to certificate revocation by leveraging the Bitcoin blockchain. Specifically, we propose embedding a unique identifier of an unspent transaction output (UTXO) into a custom extension field of the X.509 v3 certificate. A certificate will be valid as long as the corresponding UTXO has not yet been spent. To revoke the certificate, the UTXO is spent using a standard Bitcoin transaction, which any verifier can detect by querying the blockchain. This method eliminates reliance on third-party status servers, enhances user privacy, and ties certificate validity to a publicly verifiable, transparent, and decentralized infrastructure.

Our approach contributes to a more secure and efficient, decentralized PKI model by addressing key limitations of existing revocation systems and demonstrating the viability of blockchain-based solutions in real-world security applications.

Keywords— certificate validation, mozilla, UTXO, bitcoin, blockchain.

1 Introduction

In the era of big data and the Internet, people have changed their way of communicating, living, and doing business all over the world. In fact, we couldn't conceive our lives without this new way of engaging with each other. Huge amounts of data are transferred every day through the Internet, much of which is sensitive and requires protection in terms of authenticity, confidentiality, and integrity. To guarantee these three security concerns, TLS (Transport Layer Security) communications are utilized [1]. TLS security is guaranteed through the use of digital certificates that bind a pair of cryptographic keys to an identity, such as a website or a person. Certificates are issued by trusted Certificate Authorities (CA), and its main purpose is to prove that the server is who it claims to be. All this organization is known as Public Key Infrastructure (PKI), which is a framework of authorities, policies, and procedures to manage public-key cryptography. Secure transactions and communications are guaranteed by PKI. With that infrastructure, we can manage, distribute, and revoke certificates effectively.

When a client connects to a server over TLS, the server presents its certificate, which includes its public key and some identifying information such as subject name, serial number, etc. To establish trust, the server participates in a cryptographic protocol using its private key to demonstrate possession of the corresponding public key included in the certificate. The client, in turn, verifies the certificate. This validation includes checking the certificate's signature, its expiration date, and whether it has been revoked or not.

In fact, certificate revocation is crucial to ensure security in PKI since it allows for invalidating compromised or fraudulent certificates before their expiration date. Not being able to revoke such certificates can have extreme consequences, as observed in incidents like the Heartbleed SSL/TLS vulnerability. Thousands of certificates had to be revoked to diminish the risks of attackers exploiting affected servers [2]. In the absence of adequate revocation mechanisms, attackers could impersonate legitimate services, potentially resulting in unauthorized data exposure and harmful code intrusions.

To tackle this problem, several systems to verify whether a certificate has been revoked have been introduced, including Certificate Revocation Lists (CRL) [3], Online Certificate Status Protocol (OCSP) [4], and Certificate Revocation Trees (CRT) [5].

A Certificate Revocation List (CRL) is signed and issued by a CA and contains the serial numbers of all revoked certificates, along with a timestamp as an indicator of its freshness. CRLs are updated regularly, regardless of changes, to mitigate the risk of replay attacks. These lists can be accessed by simply querying a directory. Despite its simplicity, this approach comes at a cost: as CRLs grow over time, the burden on communication between directories and users increases significantly [6], undermining both scalability and efficiency. Moreover, relying on timely updates introduces a critical point of failure, given that an outdated CRL can compromise security.

The Online Certificate Status Protocol (OCSP) is a certificate validation mechanism that was introduced in 1999 by the IETF [4]. The OCSP protocol operates between a client (requester) and a server (responder). The client sends a request with certificate identifiers (e.g., serial numbers), and the server replies with the status of each certificate (marked as good, revoked, or unknown). A good status may also mean the certificate

is not yet valid or has expired. The response includes a timestamp, validity interval, and must be digitally signed. The client then analyzes the response to determine the certificate's status [7]. The main disadvantage of this method is that the OCSP server can become overwhelmed, as it must handle as many queries as the number of connected clients at a specific moment. Moreover, it poses significant privacy concerns, since each query reveals to the server which certificates are being checked. While more privacy-preserving alternatives could be explored, they may not be appealing to certain companies that benefit from monetizing this type of metadata.

Certificate Revocation Trees (CRTs) were introduced by Paul Kocher as an alternative to solve the problems already present in CRLs and other revocation mechanisms [5]. The rationale behind this method is to efficiently represent revocation status using a compact and verifiable data structure. The idea is to divide the list of revoked certificates into a series of contiguous ranges that describe certificate status between serial numbers. For example, suppose a Certificate Authority (CA) has certificates with serial numbers 1, 2, 3, 6, 7, and 8 and wants to revoke certificates 1 and 7. In that case, the ranges (1,6) and (7,8) will be constructed, where (1,6) means certificate 1 is revoked and all certificates with serial numbers greater than 1 and less than 6 are valid; similarly, (7,8) indicates certificate 7 is revoked, and certificates in the interval (7,8) are valid. Each of these ranges is then represented as a leaf node in a Merkle hash tree, along with additional metadata such as the reason and date of revocation. Leaf nodes represent certificate status ranges and are hashed upward to form a root, which is digitally signed by a trusted authority. To check a certificate's validity, a user queries a lightweight Confirmation Issuer, which returns the relevant leaf, an authentication path of hashes, and the signed root. The user verifies the certificate's status by confirming the serial number falls within the leaf's range and reconstructing the root hash. If it matches the signed root and the signature is valid, the certificate's status is confirmed. This process is efficient, scalable, and requires minimal computation and bandwidth, even for very large trees. While the main advantage of CRTs is that they eliminate the need to download the whole CRL for verification, their primary drawback lies in update management, as the entire CRT must be recomputed with any change in the list of revoked certificates. This imposes a significant computational workload on the CRT issuer, especially in the case of frequent revocations.

As seen, each approach comes with its own challenges, such as scalability constraints, financial overhead, and privacy implications. It continues to be really challenging to find a balance between effectiveness and efficiency in PKI security.

1.1 Proposal

In this work, we propose the use of Bitcoin to analyze whether a digital certificate has been revoked or not. Particularly, we plan to use an unspent UTXO, whose identifier is embedded in the `utxoID` custom extension field of the X.509 v3 standard. The web browser validates the digital certificate by verifying the existence of the corresponding UTXO. To revoke the certificate, the UTXO output is spent by generating and broadcasting a transaction using a standard Bitcoin wallet, which effectively invalidates the certificate.

This distributed approach eliminates the bottlenecks inherent in centralized systems such as OCSP by leveraging the peer-to-peer architecture of blockchain networks.

Furthermore, it removes the need to trust third parties, which enhances user privacy.

1.2 Organization of the dissertation

The rest of this dissertation is organized as follows. Section 2 reviews the current state of the art in certificate revocation mechanisms, including blockchain-based proposals, and analyzes their main limitations. Section 3 introduces the fundamentals of blockchain technology with a focus on Bitcoin, detailing the concept of Unspent Transaction Outputs (UTXOs) and their significance within the proposed framework. Section 4 states the framework's requirements based on the previous analysis of the existing literature. Section 5 presents the architectural design of the proposed framework, describing the key entities involved and detailing the steps of the revocation protocol. Section 6 details the implementation process, including the adaptation of the Mozilla Firefox browser to support UTXO-based certificate validation, and provides a brief overview of the Bitcoin network configuration used. Section 7 evaluates the system's deployment and analyzes its security and performance in terms of economic cost, time efficiency, and storage overhead. It also explores the feasibility of the proposal as a viable alternative to existing revocation mechanisms. Finally, Section 8 summarizes the main contributions of this work, reflects on its impact, and outlines potential directions for future research.

2 State of the art

The current X.509 standard comes with significant drawbacks in terms of certificate revocation methods [8]. Over the years, several alternative approaches have been proposed to address these limitations. Broadly, these approaches can be divided into two categories: those that do not rely on blockchain technologies (e.g., enhanced OCSP [9], short-lived certificates [10]), and those that leverage blockchain as a foundational element.

Recent advancements in blockchain technology have led to the emergence of a number of promising proposals. Fundamental properties of blockchain, such as decentralization, integrity, and non-repudiation, make it particularly well-suited for secure certificate revocation. As a result, our research focuses on blockchain-based approaches, which can themselves be further divided into two subcategories: proposals that introduce entirely new certificate standards, and those that remain compliant with the X.509 standard.

2.1 New certificate standard proposals

The contributions in this category put their efforts into the usage of the blockchain not only as a storage system for the newly designed certificates, but also as a platform to execute revocation processes. Some relevant proposals that fall within this category are [11–19].

Yan et al. propose a PKI system in mobile networks based on BC technology in [11]. In this architecture, a user generates a self-signed certificate and submits it to the BC through a submission node. This node is responsible for verifying and endorsing the certificate. When a user needs to revoke a certificate, she changes the status of that certificate in the ledger to ”revoked”. This proposal is not compatible with the current X.509 standard, and they do not evaluate the economic costs associated to the storage of the certificates on the BC.

CertLedger [13] proposes a decentralized PKI model that leverages blockchain to enhance transparency, revocation, and validation of TLS certificates. By storing certificates and their statuses on-chain and using smart contracts for lifecycle management, CertLedger eliminates the need for centralized Certificate Authorities (CAs) and mitigates risks like split-world attacks and OCSP-based privacy leaks. Clients verify certificates during TLS handshakes using Merkle proofs and blockchain state, without relying on external responders or trust stores. This provides a unified, tamper-resistant validation process. Despite its strengths, CertLedger depends on permissioned blockchain models, raising concerns about centralization and governance. The system also introduces integration challenges with existing PKI infrastructures.

Authors in [19] propose a decentralized digital certificate revocation system based on a consortium blockchain. Their approach addresses the limitations of traditional systems, particularly in environments involving multiple Certificate Authorities. Traditional revocation mechanisms often suffer from issues such as a lack of mutual trust, access instability, and delayed data synchronization among CAs. To overcome these challenges, they propose that multiple CAs collaborate in the management of a distributed CRL and introduce a secret sharing scheme to improve the fault tolerance and ensure the reliability of the maintenance process. Instead of relying on incentives

(as in mechanisms like Bitcoin or Ethereum), their proposed consortium BC relies on punishments, meaning that miners refusing to include a new block will be black-listed (i.e., punished). However, this proposal is not compatible with current X.509 standard implying an overhead for the CAs in the revocation process and in the maintenance of the full ecosystem. Additionally, it lacks implementation of the proposed architecture.

The main drawback of these proposals lies in the difficulty of adoption, given the widespread use and strong establishment of the current X.509 standard.

2.2 Proposals compliant with X.509 standard

In the second category (proposals built on top of the X.509 standard), certificate revocation is achieved by leveraging both the extension fields defined in the X.509 v3 specification and blockchain technology. These solutions maintain compatibility with existing PKI infrastructure while introducing decentralized mechanisms for revocation through custom certificate extensions. Noteworthy schemes within this category are [20–23].

In [20], authors introduce Secure Certificate Revocation as a Peer Service (SCRaaPS), a novel method for supporting X.509 certificate revocation using the blockchain-based Scrybe secure provenance system. SCRaaPS aims to overcome the limitations of traditional revocation methods such as CRLs and OCSP by exploiting the immutability and decentralized nature of Scrybe to store revocation data. Scrybe, along with the use of a Cuckoo filter, replaces OCSP servers by storing the serial number of the revoked certificates along with the entity's public key and the CA's public key. The Cuckoo filter is used to enable efficient local verification with no false negatives, significantly reducing the need for online queries and enhancing scalability and performance. Moreover, the system supports cross-CA revocations and intermediate certificate handling, and is designed to be a suitable replacement for OCSP stapling. However, the paper lacks an implementation of the proposed system and does not explain who is responsible for maintaining the Scrybe blockchain infrastructure. Furthermore, the design does not provide a mechanism for users to revoke their own certificates.

The authors in both [21] and [22] propose a certificate revocation and status revocation scheme that uses blockchain technology to improve the resilience and decentralization of PKI systems. In their solution, they extend the X.509 certificate structure to include a field that specifies the distribution point to which a certificate will be assigned if revoked. Each distribution point is represented by a Bloom filter, which contains the identifiers of revoked certificates. This allows clients to check certificate revocation status locally without relying on centralized CRLs or OCSP responders. The main issue with this proposal is the extra overhead and costs on the CAs. Each time a certificate is revoked, the CA has to recompute the corresponding Bloom filter and perform a new transaction to store it in the BC. Furthermore, servers are required to continuously monitor their CAs' transactions to retrieve updated revocation information relevant to their distribution point, increasing processing and synchronization demands. Additionally, Bloom filters inherently carry a risk of false positives, which means that non-revoked certificates might be misclassified as revoked, requiring clients to perform additional verification steps. This leads to increased complexity on the client side.

Xiaoxue Ge et al. propose CRchain [23], a blockchain-based certificate revocation system designed to address the latency, centralization, and inefficiencies of traditional PKI revocation methods like CRLs and OCSP. Built on Hyperledger Fabric, CRchain uses double Cuckoo filters: one for valid certificates, and one for revoked ones. That enables fast and reliable certificate status checks with minimal false positives. It also introduces a co-controlled revocation mechanism, where both CAs and servers can initiate revocations using shared control keys, thus reducing dependence on the CA and enabling faster response in cases of compromised keys. The system is designed to be scalable and fault-tolerant, offering improved performance in both data transmission size and latency over CRL and other blockchain-based methods. Experiments show CRchain significantly outperforms prior schemes like RSI [21] and CertChain [24] in terms of revocation efficiency and validation speed. Nevertheless, this proposal has several shortcomings. First, although reduced, false positives from Cuckoo filters are not fully eliminated. Second, the introduction of co-controlled keys demands infrastructure changes and additional key management responsibilities for CAs and servers, which may not be easily adoptable in existing PKI systems. Third, CRchain is implemented on a permissioned blockchain (Hyperledger Fabric), limiting its openness and raising concerns about governance and scalability across multiple untrusted domains.

3 Bitcoin blockchain

This section provides an overview of the Bitcoin blockchain technology, tracing its origins and introducing key concepts like accounts, addresses, and transactions. It concludes with an explanation of Unspent Transaction Outputs (UTXOs), a fundamental element for the complete understanding of this proposal.

The concept of blockchain technology was first introduced in 1991 by Stuart Haber and W. Scott Stornetta, who proposed a cryptographic method for time-stamping digital documents to prevent back-dating or forward-dating [25]. This early work put the foundation for further advancements in the field. A significant breakthrough came in 2008 with the publication of a paper by the pseudonymous author Satoshi Nakamoto, which not only formalized the principles of blockchain but also presented Bitcoin, the first decentralized cryptocurrency [26].

A blockchain is a distributed ledger that records transactions in a continuously growing database, shared across a decentralized network. It is composed of blocks, each containing a set of transactions. In many public blockchains, users can append new blocks to the chain, but existing blocks cannot be altered or removed. This immutability ensures that any attempt to tamper with the data would require altering the majority of copies across the network nodes, thereby enhancing the overall security and integrity of the system [27].

To interact with a blockchain network, users must first create an account and fund it with tokens, as executing transactions typically requires a form of digital currency. These tokens can be obtained either through direct purchase with real money or via token faucets. Token faucets offer users a simple way to receive small amounts of cryptocurrency, often without requiring technical knowledge or financial investment. These faucets were commonly used to promote engagement with emerging blockchain networks, although their prevalence has declined in recent years. A typical faucet operates by transferring a small quantity of tokens to a specified user account, often with usage limited to once every 24 or 48 hours. Alternatively, some faucets leverage the computational power of the user's device for mining purposes, distributing tokens as a reward after a certain duration, proportional to the user's contribution.

Some core concepts commonly used within Bitcoin blockchain technology are introduced below:

Wallets (Accounts) Unlike Ethereum, Bitcoin does not have accounts in the same sense. Instead, users interact with the network through wallets, which manage pairs of private and public cryptographic keys. A wallet can generate one or more Bitcoin addresses, which serve as destinations for receiving funds. Wallets are used to store private keys securely and sign transactions that transfer Bitcoin to other addresses.

Addresses A Bitcoin address functions as an identifier, similar to an account number, that is used to receive Bitcoin. It typically consists of 27 to 34 alphanumeric Latin characters, excluding similar-looking ones such as 0, O, or I, to avoid confusion. Bitcoin addresses can also be represented as QR codes for easier use in mobile or point-of-sale transactions [28].

Bitcoin addresses are anonymous and do not contain any information about their owner. Anyone can generate an address for free using Bitcoin software such as Bitcoin Core, which includes a feature to create new addresses. Alternatively, addresses can be obtained through exchanges or online wallet services.

There are four Bitcoin address formats. Although their purpose is the same (receiving and sending Bitcoin), they differ in structure, features, and compatibility between wallets and platforms.

- **P2PKH (Pay-to-Public-Key-Hash)**: These are the original Bitcoin address format and begin with the number 1 (e.g.,`1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa`). They are widely supported but are considered less efficient in terms of data usage, which often results in higher transaction fees.
- **P2SH (Pay-to-Script-Hash)**: Introduced in 2015, these addresses start with 3 (e.g., `3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy`). They allow for more advanced transaction types, such as multi-signature wallets or time-locked transfers. In this format, the bitcoin is sent to a hash of a script, and the recipient must fulfill that script's conditions to spend the funds.
- **Bech32 (SegWit)**: These addresses start with `bc1` and were launched in 2017 as part of the Segregated Witness (SegWit) upgrade. They are more space-efficient, which leads to lower fees and faster confirmations. Bech32 is also critical for enabling second-layer solutions like the Lightning Network.
- **Taproot (bc1p)**: Introduced in 2021, Taproot addresses also use the Bech32 format but start with `bc1p`. They improve privacy and efficiency by allowing complex spending conditions (e.g., multi-signatures or smart contracts) to appear indistinguishable from simple transactions on the blockchain.

Note that on the Bitcoin testnet, the prefix `b` in Bech32 and Taproot addresses is replaced with `t`. This helps distinguish testnet addresses from those used on the main Bitcoin network.

A Bitcoin address is derived from a public key using a series of hashing operations. First, the public key is hashed with SHA-256, and then the result is hashed again with RIPEMD-160. The resulting hash is then prefixed and suffixed with specific metadata and checksum data, and finally encoded using Base58Check [29].

Example of public key:

`04b0bd634234abbb1ba1e986e884185c8b24a17b2b2039f14813f6f84dd2b8b0e9`

SHA-256:

`0f9c95eaa7b5f8dbbd1efcb3e7bb879f3dd7c9ef3ceac1711a7e9be081a2bde7`

RIPEMD-160:

`6fe28c0ab6f1b372c1a6a246ae63f74f5e5fc6c4`

Bitcoin address (Base58Check):

`1PMycacnJaSqwwJqjawXBErnLsZ7RkXUAs`

Fees Bitcoin transactions are not measured in "gas" like Ethereum, but they do require transaction fees to incentivize miners. The fee is typically calculated based on the transaction's size in bytes and the current demand on the network. Although there is no minimum fee, transactions with higher fees are usually prioritized by miners, especially when the network is congested.

If a fee is too low, the transaction may risk being delayed or not included at all. Wallets often suggest appropriate fees based on network conditions to ensure confirmation within a reasonable time lapse.

3.1 UTXOs

When a Bitcoin transaction is broadcast, it consumes previous outputs (as inputs) and generates new outputs. The consumed inputs become spent, while the new outputs remain unspent until used in future transactions. These unspent outputs are known as Unspent Transaction Outputs (UTXOs). The UTXO set, which holds all currently unspent outputs, contains all the necessary data to validate new transactions without needing to analyze the entire blockchain [30].

A UTXO consists of two fields: the transaction ID (`txid`) and the output index (`vout`), and it holds a specific amount of BTC. Figure 1 illustrates how UTXOs are created and consumed in a Bitcoin transaction between two wallets. To make a payment, the sender's wallet selects UTXOs whose total value covers the intended amount. In this example, two UTXOs totaling 5.2 BTC are used to pay 5.1 BTC. The transaction generates two new UTXOs: one representing the payment and another as change, which is returned to the sender after subtracting the miner's fee.

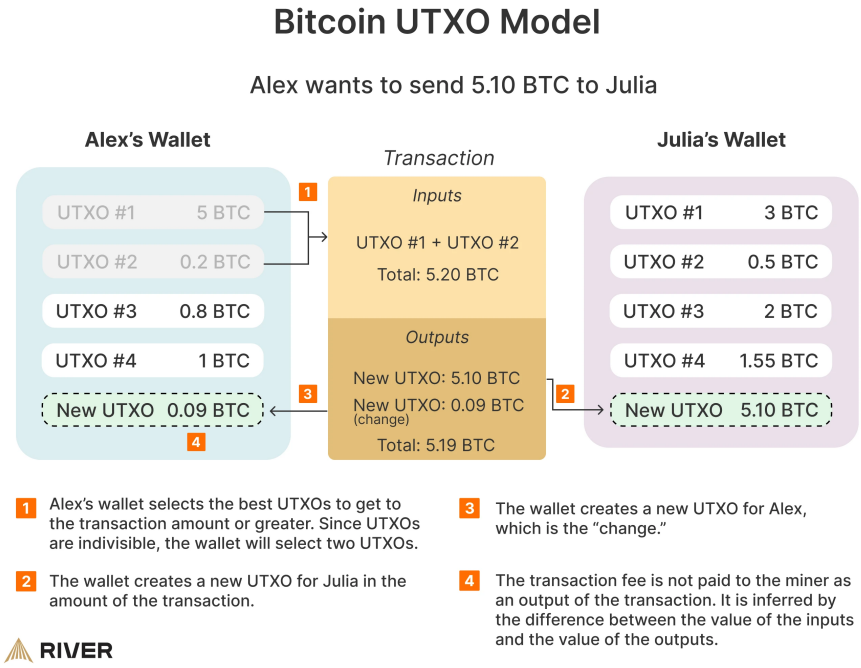

Figure 1: UTXO Model in a Bitcoin Transaction. Illustration of the creation and consumption of UTXOs in a payment. Source: https://river.com/learn/bitcoins-utxo-model/

4 Requirements

Based on the analysis of the existing literature (Section 2) and with the aim of addressing current shortcomings, several key requirements for the system have been identified. These requirements are divided into two main categories: functional and non-functional. Each category is detailed in the following sections.

4.1 Functional requirements

R1: *Support the issuance of valid certificates fully compliant with the X.509 v3 standard.*

Proposals that do not adhere to the X.509 v3 standard are difficult to integrate into the existing secure Internet infrastructure. Building on this standard ensures broad compatibility and facilitates adoption within current TLS ecosystems.

R2: *Enable validation of custom certificates directly by the web browser during the TLS handshake.*

The browser must transparently and automatically verify the validity of certificates customized with our proposal. It should provide appropriate feedback to the user (such as flagging the website as secure or insecure based on the certificate's revocation status), thereby enhancing security and user trust.

R3: *Allow certificate revocation.*

To ensure the integrity and trustworthiness of the system, it must support the ability to revoke certificates that are no longer valid or have been compromised. Revocation should be timely, reliable, and verifiable, so that revoked certificates are not mistakenly accepted as valid during the TLS handshake.

R4: *Permit servers to revoke their own certificates autonomously (thus reducing risks when keys are compromised).*

Many existing proposals do not grant revocation power to the certificate holder (i.e., the server). Empowering servers to perform timely, independent revocation, without relying on the CA, can significantly improve security in scenarios where private keys are compromised.

4.2 Non-functional requirements

4.2.1 Security and privacy requirements

R5: *Ensure that certificate verification is secure and that the TLS connection guarantees no revoked certificate can be mistakenly accepted as valid.*

The cornerstone of any revocation system lies in its ability to provide accurate and trustworthy information regarding the revocation status of certificates. The overall security of the system hinges on ensuring that no revoked certificate is erroneously accepted as valid during the TLS handshake, thereby maintaining the integrity of secure communications.

R6: *Enhance user privacy by avoiding the tracking risks associated with traditional OCSP systems.*

Current systems like OCSP can leak information about a user's browsing activity (such as visited websites and search interests), thereby compromising privacy. To mitigate these risks, our proposal avoids reliance on third-party servers by leveraging local nodes, which eliminates the need to query public RPC endpoints that could log user behavior.

R7: *Provide resilience against single points of failure to guarantee system availability and robustness.*

Traditional OCSP systems rely on centralized architectures, where certificate status queries are sent to a single trusted server. This central point can become a bottleneck or target for attacks (e.g., denial-of-service (DoS)), which may prevent timely responses and lead to revoked certificates being mistakenly accepted as valid. To address this critical vulnerability, our proposal leverages decentralized architectures, eliminating the single point of failure and enhancing the resilience and reliability of the revocation mechanism.

4.2.2 Design and implementation requirements

R8: *Leverage a decentralized and trustworthy architecture for certificate status verification.*

The system should rely on decentralized infrastructures that are publicly verifiable, tamper-resistant, and resilient to single points of failure. Such architectures enhance transparency, improve security, and reduce reliance on centralized authorities.

4.2.3 Usability requirements

R9: *Offer users the option to seamlessly switch between traditional and new certificate validation methods within the browser.*

Making it as easy as toggling a checkbox to switch between traditional and new revocation mechanisms will encourage user adoption and acceptance of the proposal.

4.2.4 Performance requirements

R10: *Ensure that the monetary costs associated with certificate revocation remain reasonable.*

Revocation mechanisms should not impose excessive financial burdens on users or certificate authorities, making the system economically sustainable and accessible.

R11: *Guarantee practical and efficient times for certificate issuance, validation, and revocation.*

The system must operate with response times that are suitable for real-world use, ensuring that users and servers experience minimal delays during certificate issuance, validation, and revocation processes.

5 Design and architecture

In this section, we present an overview of the system's architecture. First, a high-level diagram that illustrates the main components of the system and how they interact with each other is presented. Following the architectural overview, the key entities and components that constitute the system are described. Finally, the protocols that govern the system's functionalities are specified step-by-step. These include the certificate issuance, validation, and revocation.

5.1 General structure

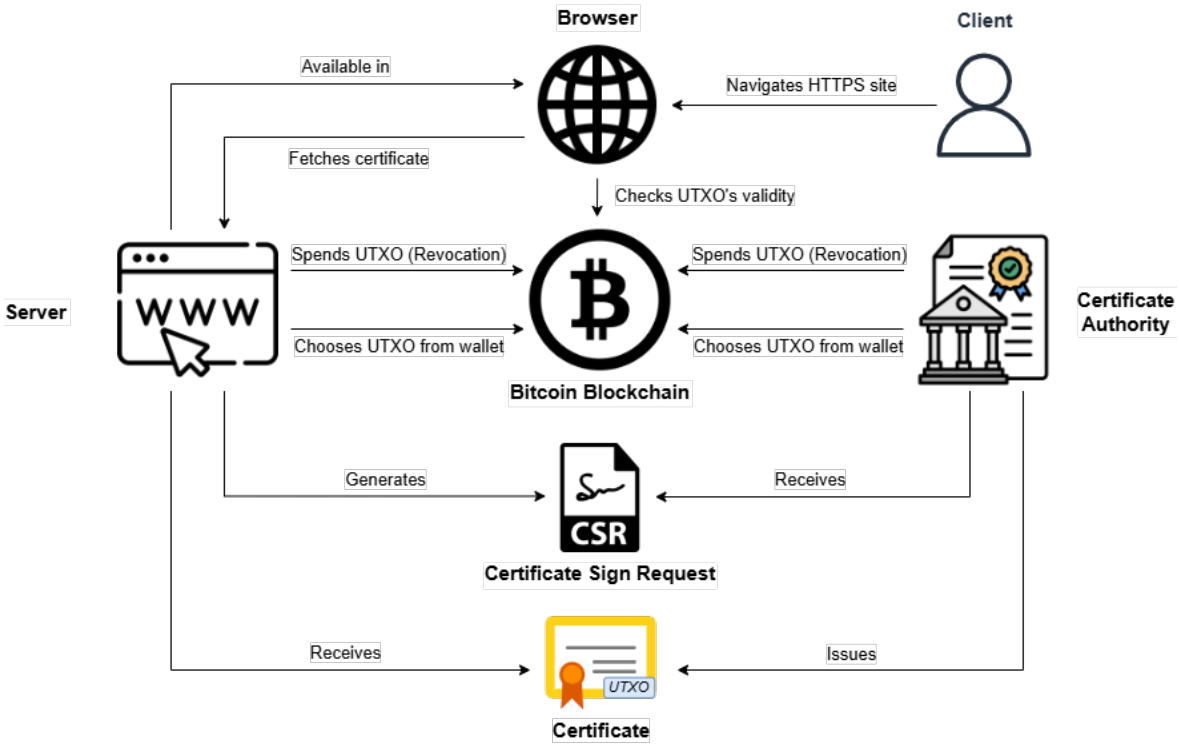

Figure 2: Overall architecture of the framework.

The overall architecture of the proposed system, depicted in Figure 2, comprises five principal entities and two key objects:

- **Bitcoin Blockchain**: Serves as the decentralized infrastructure for a secure and privacy-enhanced certification revocation mechanism based on UTXOs.
- **Browser**: Performs certificate validation when the *Client* visits websites (*Servers*) that present a certificate containing a UTXO extension.
- **Certificate Authority (CA)**: Is responsible for issuing certificates that include a valid UTXO as a custom extension, based on the information provided in the Certificate Signing Request (CSR). Additionally, the *CA* is tasked with broadcasting a Bitcoin transaction to spend a specific UTXO to revoke the associated certificate, if revocation is to be done by the *CA*.
- **Client**: Acts as the typical client in SSL/TLS-secured communications. The *Client* browses the Internet through the *Browser* and accesses websites (hosted by *Servers*).

- **Server**: Represents any website requiring a valid server certificate. It generates a CSR and submits it to the *CA* to obtain a web certificate that includes a UTXO-based extension. If the *Server* intends to retain the ability to revoke the certificate, it must include the UTXO during the CSR generation. To revoke the certificate at any point, the *Server* must then broadcast a transaction that spends the associated UTXO.
- **Certificate Signing Request (CSR)**: A request generated by the *Server* when a new certificate is needed. It contains details such as certificate constraints, intended usages, and optionally a UTXO identifier specified by the *Server*.
- **Certificate with UTXO**: An X.509 v3 certificate extended with a custom field, `utxoID`, which embeds both the `txid` and `vout` of the chosen UTXO. It is issued by the *CA* based on the *CSR* and is structured to meet the requirements of a valid server certificate.

R8: *Leverage a decentralized and trustworthy architecture for certificate status verification.*

Proof: To fulfill this requirement, our system leverages the Bitcoin blockchain as a decentralized and tamper-resistant infrastructure. By associating each certificate with a specific UTXO, the revocation status can be verified locally and securely through a query to a trusted Bitcoin node. This eliminates the need to rely on centralized revocation responders and ensures that verification is consistent, transparent, and resistant to manipulation or censorship.

In this system, the browser is seamlessly connected to a pruned Bitcoin node, enabling local download and verification of blockchain data upon startup. This approach offers strong security guarantees by eliminating the need to rely on potentially untrusted or malicious third-party block explorers. Additionally, it improves performance, as querying external explorers would require contacting multiple sources to ensure the integrity and authenticity of the information. By handling data locally, the system achieves both greater trust and faster response times.

However, this setup does require additional storage. Fortunately, storing the entire blockchain is not essential for our purposes, as we are primarily interested in the UTXO set, which is the data queried during certificate issuance, validation, and revocation. This focus significantly reduces storage requirements. A detailed analysis of storage costs is provided in Section 7.3.3.

5.2 Issuance

The generation of a UTXO-based certificate involves both the *Server* and the *Certificate Authority (CA)*. Steps 1 and 2 in the process can be carried out by either the *Server* or the *CA*, depending on which party is designated to perform revocation later. To abstract this decision, a shared procedure `ChooseUTXO()` has been defined, as illustrated in Figure 3.

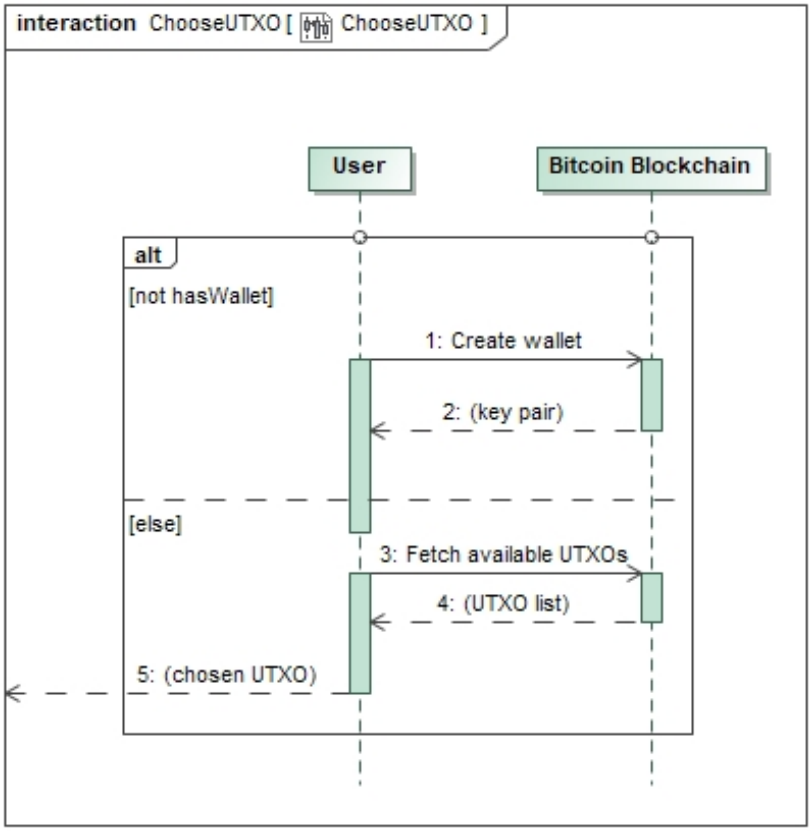

Figure 3: ChooseUTXO sequence diagram.

The certificate generation procedure (Figure 4) has the following steps:

1. User [1] U creates a wallet with a private/public key pair (or uses a previous one).
2. U chooses a UTXO from his wallet. The UTXO is defined by the transaction identifier `txid`, and the output index `vout`. Although the value in BTC of such UTXO does not affect the correctness of the protocol, a UTXO with a small amount of BTC is recommended in order to decrease the revocation cost.
3. U generates a Certificate Signing Request (CSR). They must include a custom extension named `utxoID`, which will contain both fields (`txid`, `vout`) of the chosen UTXO.
4. U sends the CSR to the certification authority (CA).
5. The CA checks that the information inside the CSR is valid. More specifically, it verifies that the selected UTXO has not been spent yet and that such UTXO belongs to U, making sure that they will be able to revoke it.
6. The CA issues a digital certificate according to the x509 v3 standard.
7. The CA finally sends the digital certificate to U.

[1] **Note:** To maintain generality, the term *User* is used throughout this section to refer to either the *Server* or the *CA*, depending on which entity is responsible for revocation.

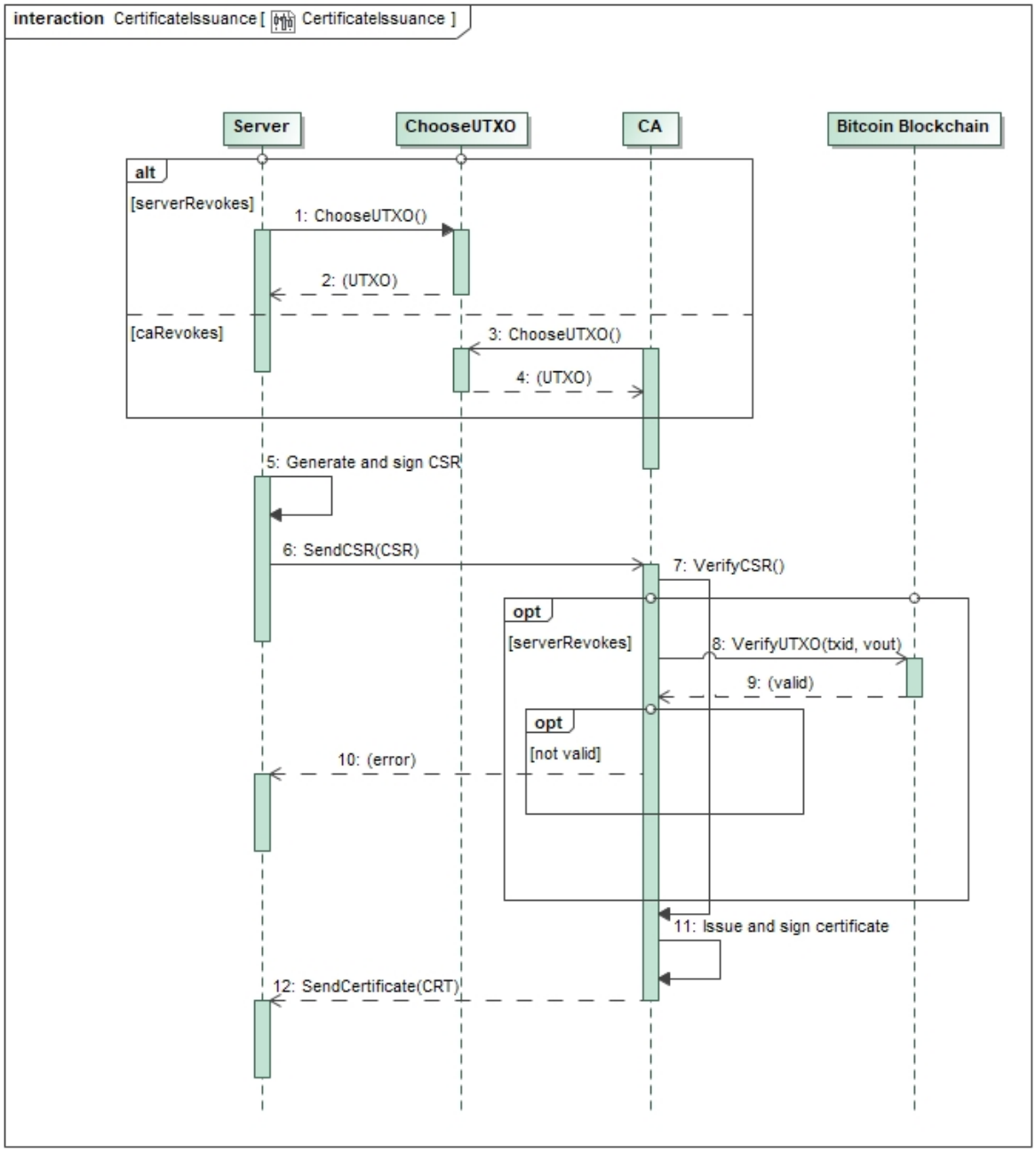

Figure 4: Certificate Issuance protocol sequence diagram.

5.3 Validation

The certificate validation procedure (Figure 5) has the following steps:

1. Any entity may present the certificate in front of the validation authority *VA*. Particularly, in this architecture, the *VA* is represented by both the *Browser* and the *Bitcoin Blockchain*.

2. The *Browser* obtains `txid` and `vout` from the certificate (i.e., parses the UTXO) and queries the *Bitcoin Blockchain* to check if the UTXO is still spendable.

   (a) If the output is spendable, the certificate is valid. Therefore, the *Browser* will mark the *Server's* website as secure and valid to the *Client*.

(b) If the output has been spent, the certificate has been revoked and therefore it is invalid. This way, a secure connection cannot be guaranteed. The *Browser* should mark the browsing as insecure and communicate to the *Client* that the certificate has been revoked.

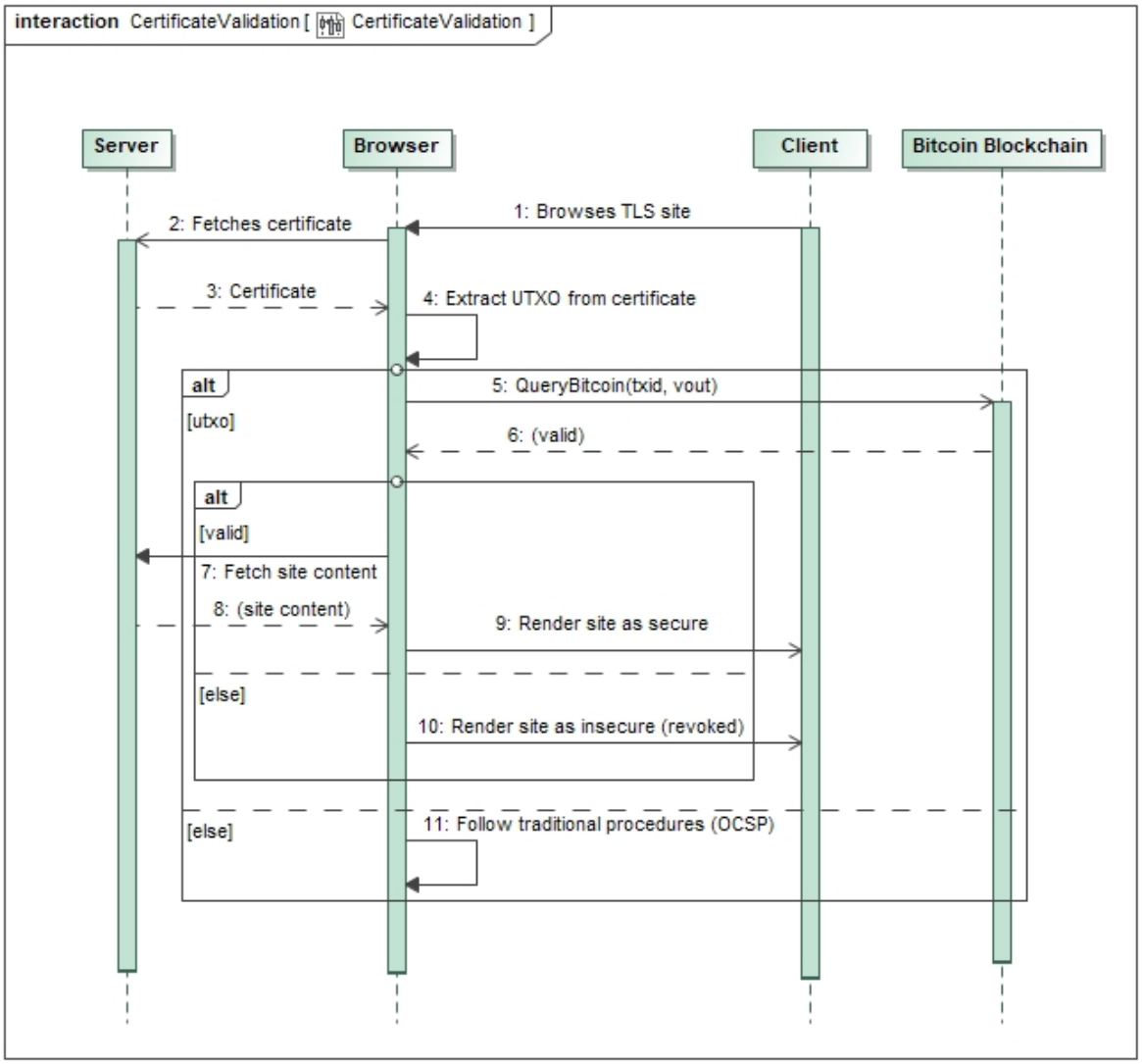

Figure 5: Certificate Validation protocol sequence diagram.

5.4 Revocation

The certificate revocation procedure, illustrated in Figure 6, consists of the following steps, which are carried out by the *User* entity responsible for revocation. This entity can either be the *Server* itself or the *CA*, depending on whose UTXO was embedded into the certificate. The process is identical for both cases:

1. The user (U) extracts the transaction identifier `txid` and the output index `vout` from the certificate's custom extension field, which was embedded at issuance time.

2. *U* then uses their Bitcoin wallet to broadcast a transaction that spends the referenced UTXO. This transaction is propagated through the Bitcoin network and, once included in a block, marks the associated certificate as revoked.

It is important to note that the *CA* must keep a registry of UTXOs it has assigned to certificates that are still active. This measure prevents accidental reuse of the same UTXO across multiple certificates or attempts to revoke a certificate with an already-spent output. A more detailed discussion on how UTXOs can be efficiently managed and recycled to optimize system resources is provided in Section 7.

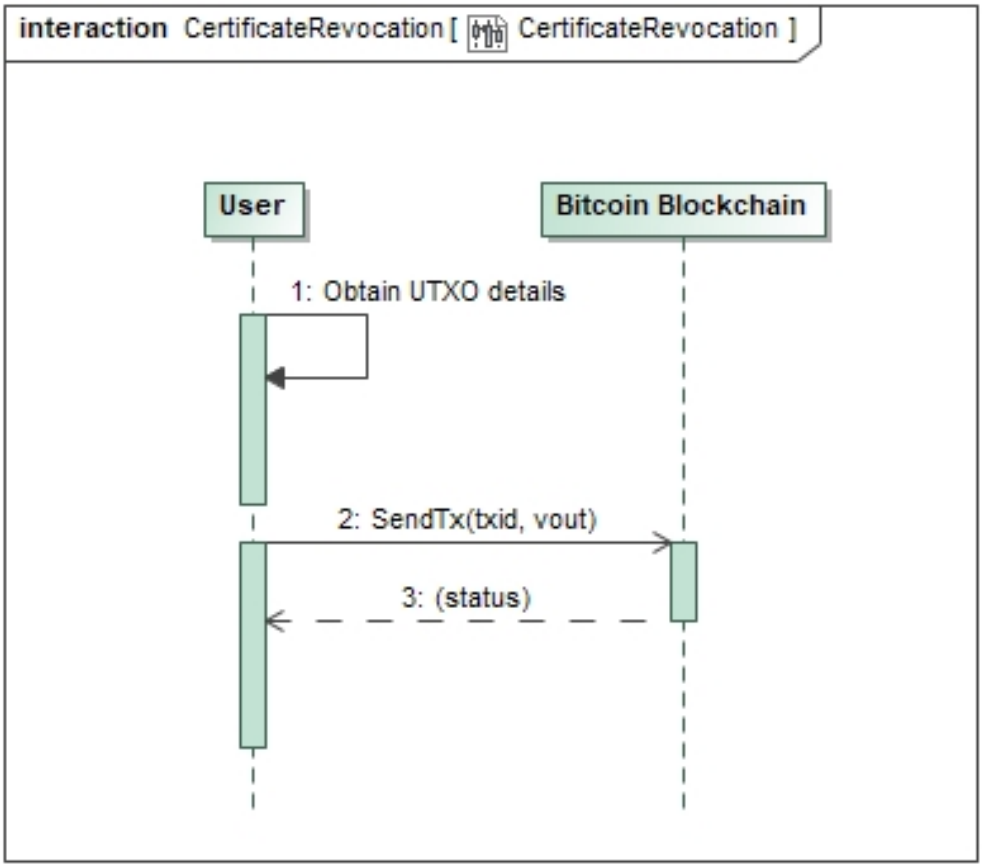

Figure 6: Certificate Revocation protocol sequence diagram.

6 Implementation

This section describes the key steps taken to implement the system introduced in Section 5. In particular, it details the modifications made to the Mozilla Firefox browser to support UTXO-based certificate validation, as well as the selected Bitcoin architecture used to allow this functionality. Additionally, it also provides the main steps to generate Certificate Signing Requests (issued by Servers) and the associated Certificates (issued by Certificate Authorities).

6.1 Mozilla Firefox fork

A crucial part in guaranteeing the security of the PKI infrastructure is proper certificate validation. In most cases, this validation logic is built directly into the browser. To support our proposed system, we chose to modify the source code of Mozilla Firefox, as its open-source nature enables direct changes to the certificate validation process. Note also that, throughout this document, the process of determining whether a certificate is revoked may be referred to interchangeably as certificate validation or certificate verification. In particular, Mozilla developers refer to this process as certificate verification, so we will adopt the same terminology in the implementation section.

To modify Mozilla Firefox's source code, we first downloaded the necessary developer tools for Windows and cloned the repository as instructed in the Firefox Source Docs[2]. After making the necessary modifications, the source code has to be compiled using the `./mach build` command, and ran afterwards (if no error is found) using the `./mach run` command from within the working directory. The following subsections detail the specific modules that have been modified, with a focus on the certificate verification logic, the communication with the Bitcoin blockchain, and the updates made to Mozilla's GUI to enable or disable custom preferences related to UTXO-based validation. The complete modified code can be found in this GitHub project: https://github.com/imiguelrodriguez/utxo-based-mozilla/tree/utxo-mods.

6.1.1 Certificate verification

Most of the modifications were applied to the certificate verification mechanism within the security module, as it constitutes the core logic of the project. Initially, reverse engineering was required to precisely locate the code section responsible for certificate verification. Historically, Firefox relied on the NSS (Network Security Services) library for this purpose. However, recent versions (more precisely from version number 31) now utilize a dedicated module called `certverifier` for certificate verification [31], although it still partially relies on some data structures provided by NSS.

To support the validation of certificates extended with a custom Bitcoin-based revocation mechanism, the core function `CertVerifier::VerifyCert` of the security module was modified. This function is responsible for validating a certificate at the time of the TLS handshake and is part of Mozilla's certificate verification engine.

The implemented logic is conditionally activated via a customized Mozilla user preference `security.bitcoin.utxo.enabled` (further described in Section 6.1.3). When

[2] **Quick guide for Mozilla Firefox contributors:** https://firefox-source-docs.mozilla.org/contributing/contribution_quickref.html#firefox-contributors-quick-reference

this preference is enabled, the certificate's extensions are scanned for a custom-defined OID (`1.3.112.4.30.1270`)[3]. According to the OID repository, this identifier is associated with key revocation, making it an appropriate choice for our use case. In the context of our implementation, this OID is used to embed a reference to a Bitcoin UTXO directly within the certificate, enabling decentralized validation of its revocation status. Conversely, when the preference is disabled, the custom validation logic is bypassed and standard OCSP-based revocation checking is performed instead.

- The function iterates through the list of certificate extensions. When the custom OID is found, the corresponding extension is decoded as an ASN.1 `SEQUENCE` containing two elements: a transaction ID (`txid`) encoded as an `OCTET STRING`, and a transaction output index (`vout`) encoded as an `INTEGER`. Note that this encoding matches the one in the configuration file when issuing CSR and certificates (explained in Appendix D), ensuring correct parsing of the UTXO.
- These fields are extracted using a custom ASN.1 decoding template. The `txid` is converted into a hexadecimal string for RPC compatibility, and the `vout` is parsed as an unsigned integer.
- Once extracted, the function invokes the helper `QueryBitcoinTxOut(txid, vout)` function to query a local Bitcoin node via RPC, checking whether the corresponding UTXO is still unspent. More details on the implementation of the communication with the Bitcoin node are given in Section 6.1.2.
- An auxiliary helper function (`parseJSONresponse(std::string& response)`) parses the node's response. If the node returns `null`, it indicates the UTXO has been spent, which the system interprets as a certificate revocation. In that case, the verification function returns a revoked status, interrupting the connection with an appropriate error (`SEC_ERROR_REVOKED_CERTIFICATE`).

```
if (Json::parseFromStream(...)) {
// Check if the result is null
  if (jsonResponse["result"].isNull()) {
    MOZ_LOG(gCertVerifierLog, LogLevel::Error, ("UTXO has been spent..."));
    return Result::ERROR_REVOKED_CERTIFICATE;
   } else {
      MOZ_LOG(gCertVerifierLog, LogLevel::Debug, ("UTXO is unspent...\n"));
      return Result::Success;
    }
  } else {
    MOZ_LOG(gCertVerifierLog, LogLevel::Error, ("Failed to parse JSON);
    return Result::FATAL_ERROR_INVALID_ARGS;
    }
}
```

- Extensive logging was added using `MOZ_LOG` to provide debugging and traceability throughout the parsing, querying, and decision process.

[3] **Key revocation OID:** https://oid-base.com/get/1.3.112.4.30.1270

This integration demonstrates how Mozilla's verification engine can be extended to support decentralized revocation logic using blockchain data, while preserving compatibility with existing certificate structures through the use of standard-compliant custom OIDs.

To support the use of this new custom OID for UTXO-based revocation, several modifications were made to the NSS (Network Security Services) codebase. These changes ensure that the new OID can be recognized, parsed, and handled properly within the certificate verification process.

- In `nss/lib/util/secoidt.h`, the identifier `SEC_OID_UTXO = 390` was added to the `SECOidTag` enumeration. This value corresponds to the next available tag in the enumeration and uniquely identifies the new OID within the NSS internal system.
- In `nss/lib/util/secoid.c`, the byte sequence representing the OID was defined as a constant array using hexadecimal values:

  ```
  CONST_OID utxo[] = { 0x2B, 0x70, 0x04, 0x1E, 0x89, 0x76 };
  // OID 1.3.112.4.30.1270
  ```

  The encoding follows the DER (Distinguished Encoding Rules) format for object identifiers.
- In the global `oids` array (also in `secoid.c`), a new entry was added corresponding to the custom UTXO OID. The entry uses the `OD` macro, which simplifies the definition of OID descriptors within NSS. The added line is:

  ```
  OD(utxo, SEC_OID_UTXO, "UTXO", CKM_INVALID_MECHANISM,
  INVALID_CERT_EXTENSION),
  ```

  Here, the `OD` macro is defined as follows:

  ```
  #define OD(oid, tag, desc, mech, ext)
  { OI(oid), tag, desc, mech, ext }
  ```

  This macro constructs an entry in the OID descriptor table, where:

  - `OI(oid)` expands the encoded OID bytes,
  - `tag` is the corresponding `SECOidTag` identifier (`SEC_OID_UTXO`),
  - `desc` is a human-readable description (`"UTXO"`),
  - `mech` specifies the associated cryptographic mechanism (`CKM_INVALID_MECHANISM`),
  - `ext` indicates whether the OID is a recognized certificate extension (here set to `INVALID_CERT_EXTENSION` as it's custom-defined).

  This registration enables NSS to correctly recognize the new OID as a valid extension when parsing certificates.

6.1.2 Blockchain communication

To interact with the Bitcoin blockchain and query UTXO status, two source files were added to the `certverifier` module: `BitcoinHandler.cpp` and `BitcoinHandler.h`. These files implement a minimal RPC client that communicates with a local Bitcoin node using sockets over TCP.

The primary function responsible for this communication is defined as:

```
nsresult QueryBitcoinTxOut(const std::string& txid,
const std::string& vout, std::string& response);
```

This function performs the following sequence of steps:

1. **Initialize the Winsock environment:** Using `WSAStartup`, the function prepares the Windows networking stack to perform socket operations.
2. **Create and connect the socket:** A TCP socket is created and connected to the local Bitcoin RPC server, typically running on `127.0.0.1` and port `18332` (default for testnet).
3. **Construct the RPC request:** A JSON-RPC 1.0 request is built for the `gettxout` method, inserting the parsed `txid` and `vout` values. The request is structured as:

   ```
   {
     "jsonrpc": "1.0",
     "id": "curltext",
     "method": "gettxout",
     "params": ["<txid>", <vout>, true]
   }
   ```

4. **Add authentication:** RPC access requires Basic HTTP authentication. The function encodes the `username:password` pair chosen by the user in Base64 using a custom `base64_encode` utility function.
5. **Send the HTTP request:** The function assembles the full HTTP POST request with proper headers, including `Content-Length` and `Authorization`, then sends it through the socket.
6. **Receive and store the response:** The function reads the response from the Bitcoin node into a buffer and stores it in the `response` output parameter. This response is later parsed by the certificate verification logic to determine whether the referenced UTXO is still unspent.
7. **Cleanup:** Finally, the socket is closed and Winsock is cleaned up using `closesocket` and `WSACleanup`, respectively.

To support integration with the Mozilla build system, both source files were registered in the `moz.build` file under the `certverifier` module using:

```
EXPORTS += [
    "BitcoinHandler.h",
    ...
]
UNIFIED_SOURCES += [
    "BitcoinHandler.cpp",
    ...
]
```

This component plays a central role in enabling decentralized revocation by retrieving the live status of the UTXO embedded in the certificate. If the UTXO has been spent, the certificate is considered revoked.

6.1.3 Modifications in GUI

R9: *Offer users the option to seamlessly switch between traditional and new certificate validation methods within the browser.*

Proof: When users interact with our system, they should be able to indicate whether they prefer to use it or not, much like Mozilla allows users to enable or disable OCSP. To facilitate this, a new checkbox was added to the "Privacy & Security" panel within the browser's Preferences settings, inside the section devoted to certificates. This checkbox allows users to toggle between UTXO-based certificate validation and the traditional OCSP mechanism (Figure 7). When the UTXO option is enabled, the OCSP checkbox is automatically disabled, and vice versa, ensuring that only one validation method is active at a time.

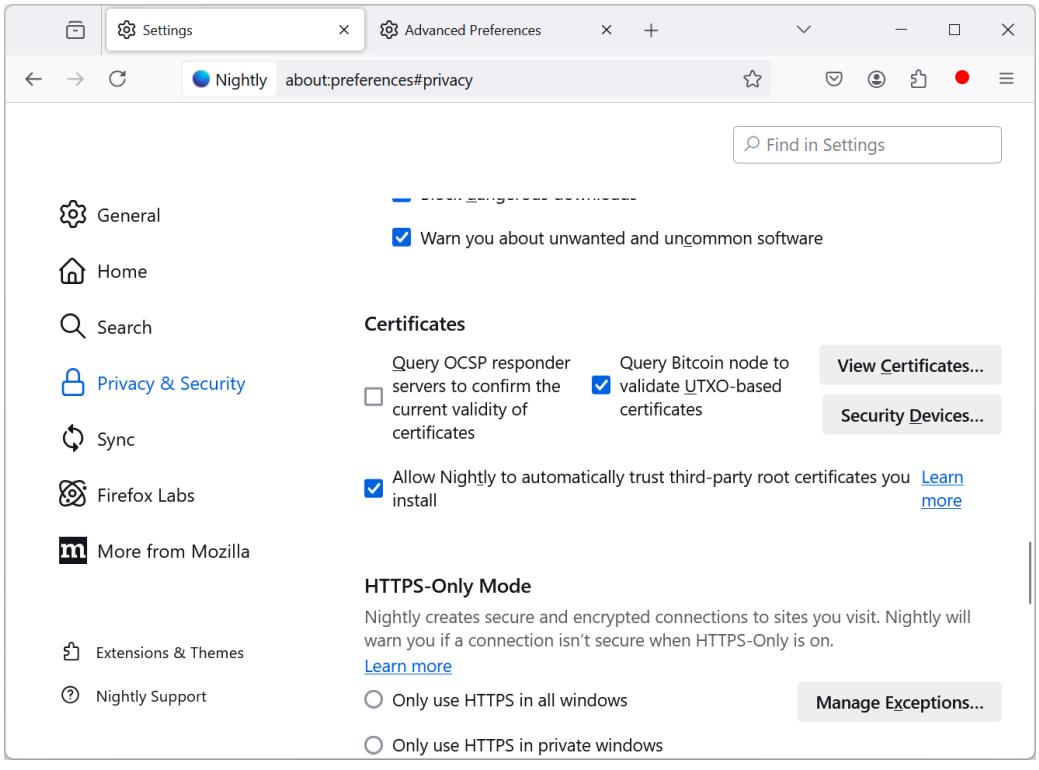

Figure 7: UTXO and OCSP certificate validation options in the Privacy & Security section of Firefox's Preferences.

This toggle updates a preference flag named `security.bitcoin.utxo.enabled` (Figure 8), which is later checked in the `VerifyCert.cpp` file during certificate validation. If the flag is enabled, the browser executes the logic related to UTXO-based validation. Otherwise, the default OCSP validation procedure is followed.

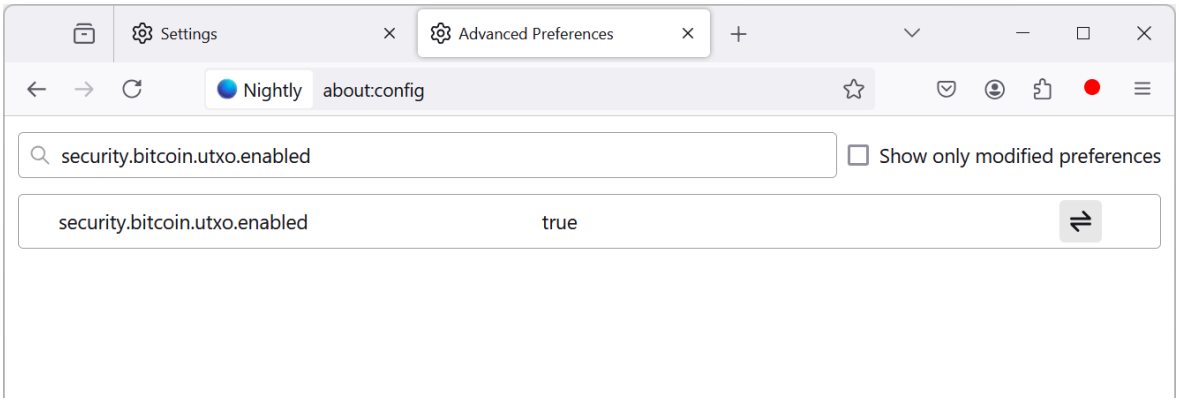

Figure 8: The `security.bitcoin.utxo.enabled` flag in `about:config`, which determines whether UTXO-based validation is active.

To implement the checkbox that allows users to enable or disable UTXO-based certificate validation from Firefox's settings interface, several modifications were required across different components of the browser. The purpose of each change is outlined below:

- **Preference definition:** In `modules/libpref/init/StaticPrefList.yaml`, the preference `security.bitcoin.utxo.enabled` is defined as a boolean. This allows the preference to be accessed directly from the C++ backend, for example in the `VerifyCert.cpp` file, where the custom logic for UTXO validation is triggered depending on this flag.

  ```
  # Bitcoin validation behavior:
  # false: do not validate using Bitcoin, true: otherwise
  - name: security.bitcoin.utxo.enabled
    type: RelaxedAtomicBool
    value: false
    mirror: always
  ```

- **Default value:** In `modules/libpref/init/all.js`, the default value of the preference is set to `true`. This ensures that the behavior is initially active unless the user disables it manually.

  ```
  pref("security.bitcoin.utxo.enabled", true);
  ```

- **User interface binding:** In `browser/components/preferences/privacy.inc.xhtml`, a new checkbox element is added to the Certificates section, referencing the `security.bitcoin.utxo.enabled` preference. This element is what users interact with in the Privacy & Security panel.

  ```
  <checkbox id="enableUTXO" data-l10n-id="certs-enable-utxo"
  preference="security.bitcoin.utxo.enabled" flex="1"/>
  ```

- **Localization:** The label for the checkbox is added in `browser/locales/en-US/browser/preferences/preferences.ftl`, allowing proper internationalization and user-friendly text for the checkbox.

```
certs-enable-utxo =
    .label = Query Bitcoin node to validate UTXO-based certificates
    .accesskey = U
```

- **Read/Write functions:** Functions such as `readEnableUTXO()` and `writeEnableUTXO()` are implemented in `browser/components/preferences/privacy.js` to handle the dynamic state and behavior of the checkbox. These functions synchronize the UI with the internal preference and also ensure that enabling one mechanism (e.g., UTXO) disables the other (e.g., OCSP), maintaining mutual exclusivity. Additionally, `readEnableOCSP()` and `writeEnableOCSP()` were modified for the same reason.

6.2 Public Key Infrastructure (PKI)

This section outlines the deployment of the Public Key Infrastructure, describing the implementation of all involved entities and components: the Certificate Authority, the Server, the Certificate, and the Certificate Signing Request.

6.2.1 Server and CSR

The server (i.e., any website that needs a certificate) plays a critical role in the architecture, as it is responsible for presenting a UTXO-based certificate to clients via the browser during the TLS handshake. To support HTTPS, we employed Apache HTTP Server, which allows for flexible configuration of certificates and secure communication protocols. Specifically, Apache provides configuration directives that can be used to specify the file paths to the server's certificate, the corresponding private key, and the issuing Certificate Authority certificate, so that the chain of certification can be built.

In our implementation, the server uses a certificate that includes a custom extension referencing a Bitcoin UTXO. This certificate is generated by submitting a specially crafted Certificate Signing Request to the CA. Once issued, the certificate is installed on the server and referenced in Apache's SSL configuration (via `SSLCertificateFile`, `SSLCertificateKeyFile`, and `SSLCertificateChainFile` directives in the `ssl.conf` file).

A Certificate Signing Request (CSR) is like a recipe sent to the Certificate Authority (CA), containing the essential information required for generating a digital certificate [32]. It marks the initial step in obtaining a certificate. The CSR includes the applicant's public key, domain name, organization details, and location. This data is used by the CA to create and sign the certificate, enabling secure and trusted communications over protocols like TLS. The associated private key remains confidential (through password-based encryption) and is never shared with the CA.

To generate a valid CSR, we make use of the `OpenSSL` command-line tool on Linux. A detailed, step-by-step guide for generating a CSR using `OpenSSL` is provided in Appendix C. By default, the CSR includes standard server certificate extensions, such as Subject Alternative Name (SAN) and Extended Key Usage (EKU), when revocation

control is delegated to the Certificate Authority. However, if the server is responsible for revoking its own certificate, the UTXO information must be embedded directly within the Certificate Signing Request (CSR). This is achieved by defining a custom OID for UTXO embedding in the `opensslPKI.cnf` file, which can be specified as an input when generating CSRs or certificates using `OpenSSL` (see Appendix D for details). The CSR will be sent to the CA as a `.csr` file in order to obtain the corresponding certificate.

6.2.2 CA and Certificate

The Certificate Authority is responsible for assessing the validity and correctness of the received CSR and issuing a certificate in response. That involves verifying that all information included in the CSR is accurate and genuinely associated with the requesting server. The issued certificate should contain the same information as the original CSR. However, the CA may modify the custom extensions if necessary, particularly if the requested extensions are incorrect or require adjustment.

In the case that the CA is responsible for revocation, it should also embed the corresponding UTXO in the certificate at this stage using custom extensions. As done for the CSR, our implementation leverages the `OpenSSL` library to issue the custom certificate. For a detailed explanation of the steps involved in signing and issuing a certificate from a CSR, refer to Appendix C. It is the responsibility of the Certificate Authority (CA) to maintain a record of all issued certificates in order to track used and unused UTXOs. This ensures that no more than one certificate is issued for the same UTXO, preventing potential complications during revocation, where other certificates could be mistakenly revoked.

6.3 Bitcoin Blockchain

To provide the maximum degree of security and privacy, we decided to run a local copy of a Bitcoin node alongside the modified Mozilla Firefox browser. This can be achieved by installing the official Bitcoin Core [4] software. Bitcoin Core enables the full download of the blockchain and supports the creation and management of multiple wallets, which can be used to send and receive transactions. Moreover, it automatically performs block validation as new blocks are downloaded. An overview of the Bitcoin Core GUI can be seen in Figure 9. As is common in many blockchain-based projects [33], the proof-of-concept is tested using the Bitcoin testnet rather than the mainnet. This approach avoids real monetary costs during testing while providing a safer and more flexible environment for experimentation.

To fund wallets in the testnet, users typically rely on faucets. These are services that distribute small amounts of cryptocurrency, often in exchange for computational effort or simply offered altruistically. These resources are usually rate-limited by IP address or time intervals to prevent abuse.

In this project, two Bitcoin testnet3 faucets were utilized: Coinfaucet[5] and UO1's Bitcoin Testnet faucet[6]. Particularly, these tools provide small amounts of Bitcoin

[4] **Bitcoin Core:** https://bitcoin.org/en/bitcoin-core/
[5] **Coinfaucet:** https://coinfaucet.eu/en/btc-testnet/
[6] **UO1's Bitcoin Testnet faucet:** https://bitcoinfaucet.uo1.net/

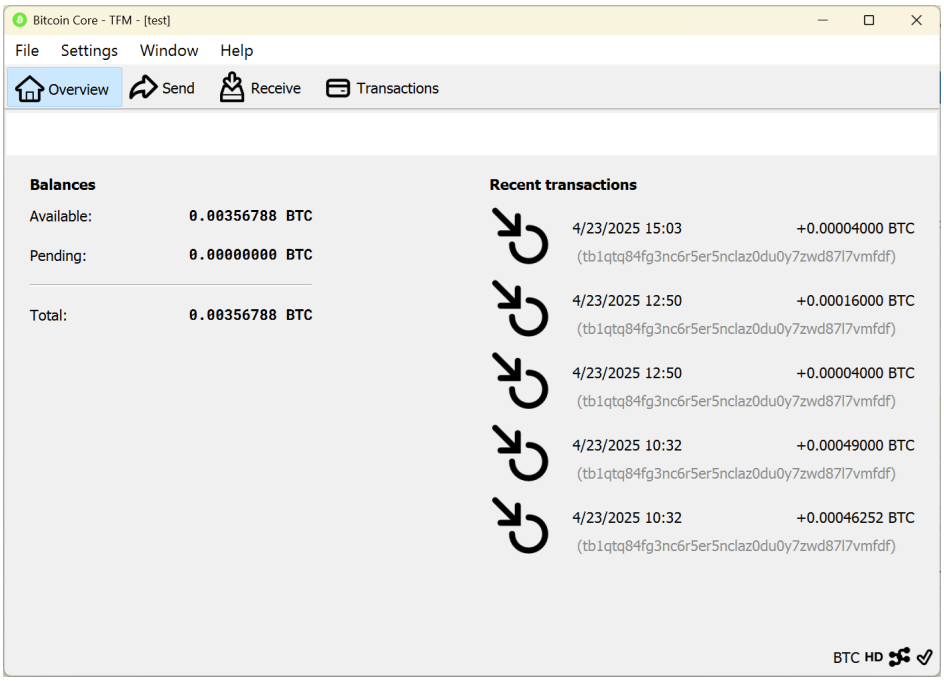

Figure 9: Bitcoin Core GUI

testnet tokens (e.g., 0.00004 BTC) to user accounts without the need for mining or technical effort. They are truly valuable for developers testing applications in the Bitcoin testnet environment. To prevent abuse and ensure fair distribution, these faucets implement IP tracking and other security measures to restrict repeated requests from the same user. Examples of such faucets are shown in Figures 10 and 11.

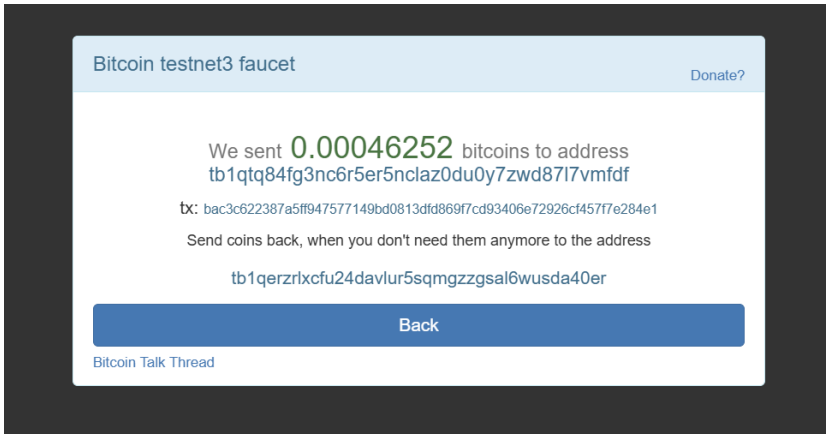

Figure 10: CoinFaucet sending a small amount of testnet BTC to the developer's wallet.

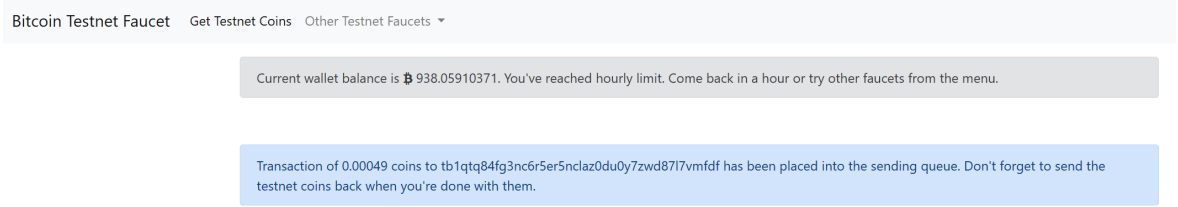

Figure 11: UO1 Faucet puts into the sending queue 0.00049 testnet BTC to be transferred to the developer's wallet.

7 Evaluation

This section evaluates the system's performance and overall quality by examining economic, temporal, and storage-related costs, as well as its security and practical viability as a potential replacement for traditional revocation methods like CRLs and OCSP. We start by describing the testing environment, then we analyze the associated costs and security implications. Finally, we assess the suitability of the proposed system as an alternative to existing revocation mechanisms.

7.1 Testing environment

To test the correct functionality of the system, we conducted tests using a Kali Linux virtual machine. Two Apache servers were configured: one hosting a valid UTXO-based certificate and the other using a revoked UTXO-based certificate. An outline of the described testing environment is shown in Figure 12. As described in Section 6, certificates were generated using `OpenSSL`. During testing, the Bitcoin Core software was also running, as it is necessary for enabling communication from the browser to the blockchain to perform certificate validation.

As a preliminary step, we evaluated how the Mozilla browser handles the validation of OCSP-based certificates. To do this, we set up an OCSP server and deployed the same Apache server configuration (one with a valid certificate and one with a revoked certificate). This allowed us to observe the behavior of both Apache and Mozilla under standard OCSP validation. Further details on this setup can be found in Appendix A.

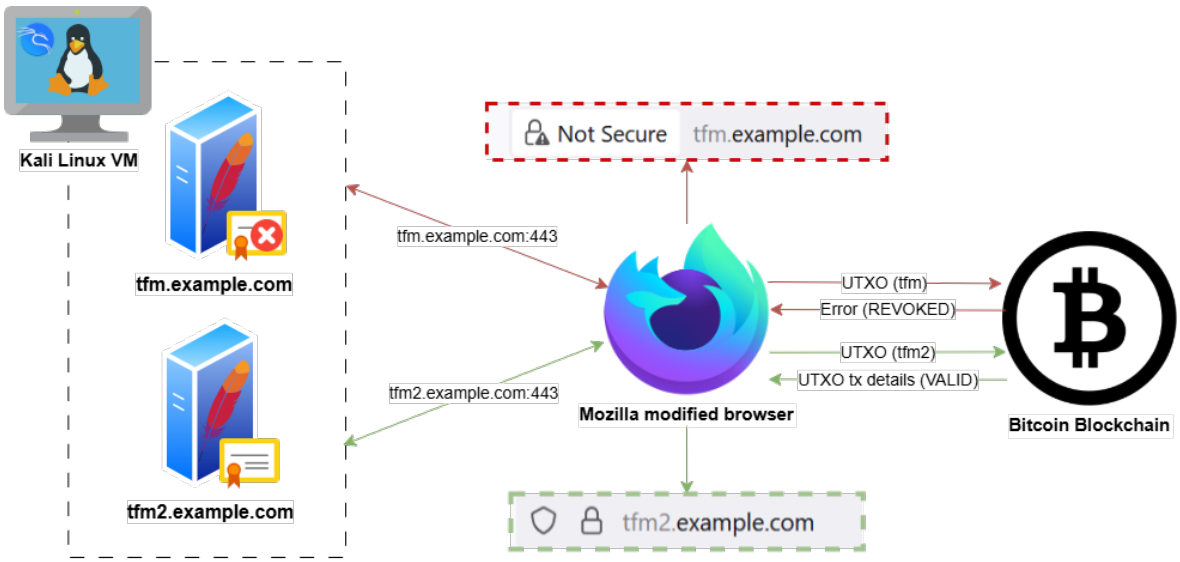

Figure 12: Testing environment overview. Two Apache servers are configured: one hosts a revoked certificate (`tfm.example.com`), and the other serves a valid certificate (`tfm2.example.com`). The modified version of the Mozilla browser queries the Bitcoin blockchain using the UTXO information. Based on the revocation status, the browser marks `tfm.example.com` as insecure (revoked certificate) and `tfm2.example.com` as secure (valid certificate).

Next, we prove the functional requirements stated in Section 4.1.

R1: *Support the issuance of valid certificates fully compliant with the X.509 v3 standard.*

Proof: The certificate generation process employs `OpenSSL`'s `x509` module, which ensures that all certificates issued follow the X.509 v3 standard. The inclusion of a UTXO reference is achieved through a custom extension defined by a unique OID (`1.3.112.4.30.1270`, associated with key revocation processes). This extension embeds a reference to a specific transaction output on the blockchain, allowing later verification or revocation. Importantly, the X.509 v3 standard explicitly supports the use of custom extensions, meaning that our addition does not compromise compliance or interoperability. For implementation details on how the UTXO is embedded, refer to Appendix D.

R2: *Enable validation of custom certificates directly by the web browser during the TLS handshake.*

Proof: A modified Mozilla-based browser has been implemented to support the validation process. It extracts the UTXO from the certificate's custom extension and queries the Bitcoin blockchain to determine whether the referenced UTXO is still unspent (indicating validity) or spent (indicating revocation).

To observe log messages related to certificate verification in Mozilla Firefox, the following command should be run in the shell: `MOZ_LOG="certverifier:5"`. Then, the browser can be launched and directed to the two test sites shown in Figure 12.

The first site (`tfm.example.com`) is flagged as insecure (see Figure 13). Specifically, Firefox returns the `SEC_ERROR_REVOKED_CERTIFICATE` error, indicating that the website's certificate has been revoked and the connection cannot be trusted.

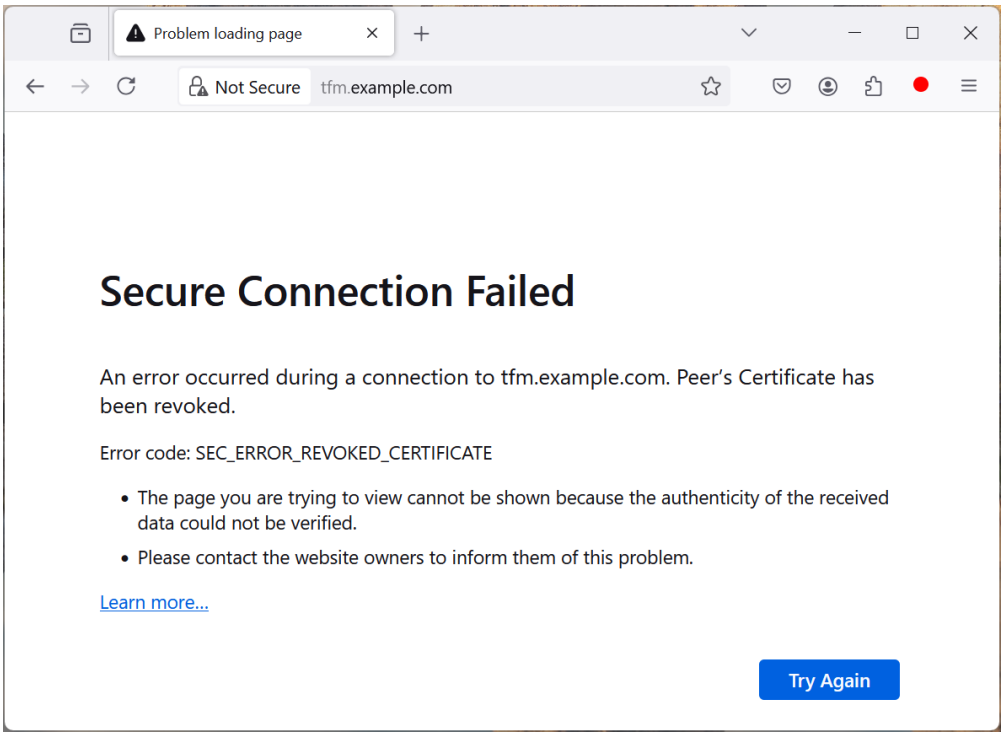

Figure 13: `tfm.example.com` server, which holds a revoked certificate. Mozilla flags the communication as insecure (see lock with warning in the search tab).

This behavior occurs because, although Mozilla successfully identifies and parses the custom UTXO extension embedded in the certificate (as shown in the logs in Figure 14), the subsequent query to the local Bitcoin node (see the Wireshark capture in Figure 15) returns `null`. This response indicates that the referenced UTXO has already been spent, and consequently, the certificate must be considered revoked.

Figure 14: Logs showing the verification process of the certificate for the website `tfm.example.com`. First, the UTXO-related OID is detected and parsed to extract the `txid` and `vout`. Then, a query is issued to the local Bitcoin node, whose response confirms that the referenced UTXO has been spent, leading to the certificate being marked as revoked and invalid.

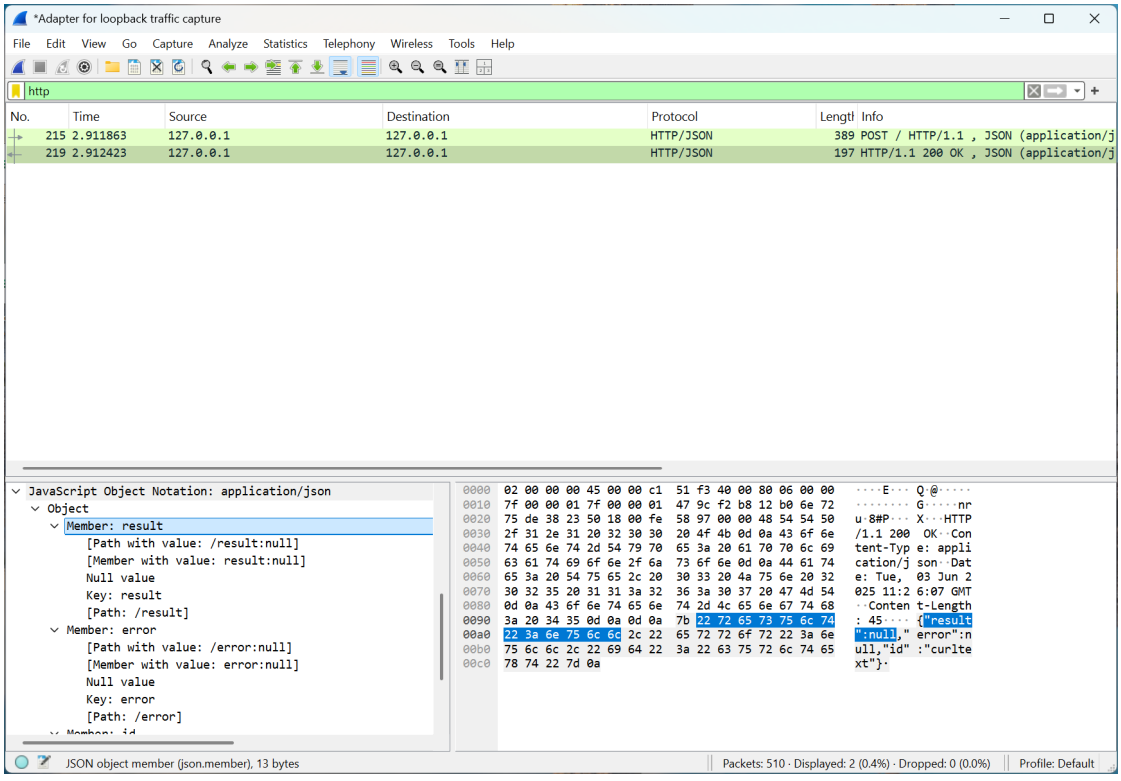

Figure 15: Wireshark capture highlighting the RPC response from the Bitcoin node. The response indicates that the queried UTXO is `null`, confirming that it has already been spent and, therefore, that the associated certificate must be considered revoked.

The second site (`tfm2.example.com`), in contrast, is marked as secure (see Figure 16). The steps followed by the Mozilla browser are the same as in the other case: identifying and parsing the custom UTXO OID embedded in the certificate (Figure

17) and querying the Bitcoin node about the associated UTXO (Figure 18). These results demonstrate that the modified Mozilla browser correctly validated the custom certificate and responded accordingly by establishing a secure connection.

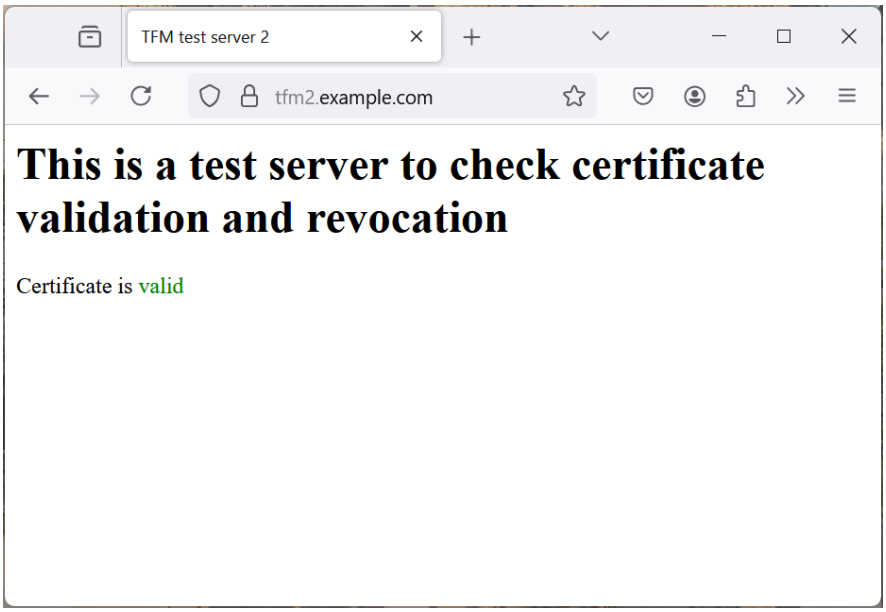

Figure 16: `tfm2.example.com` server, which holds a valid certificate. Mozilla flags the communication as secure (see lock in the search tab).

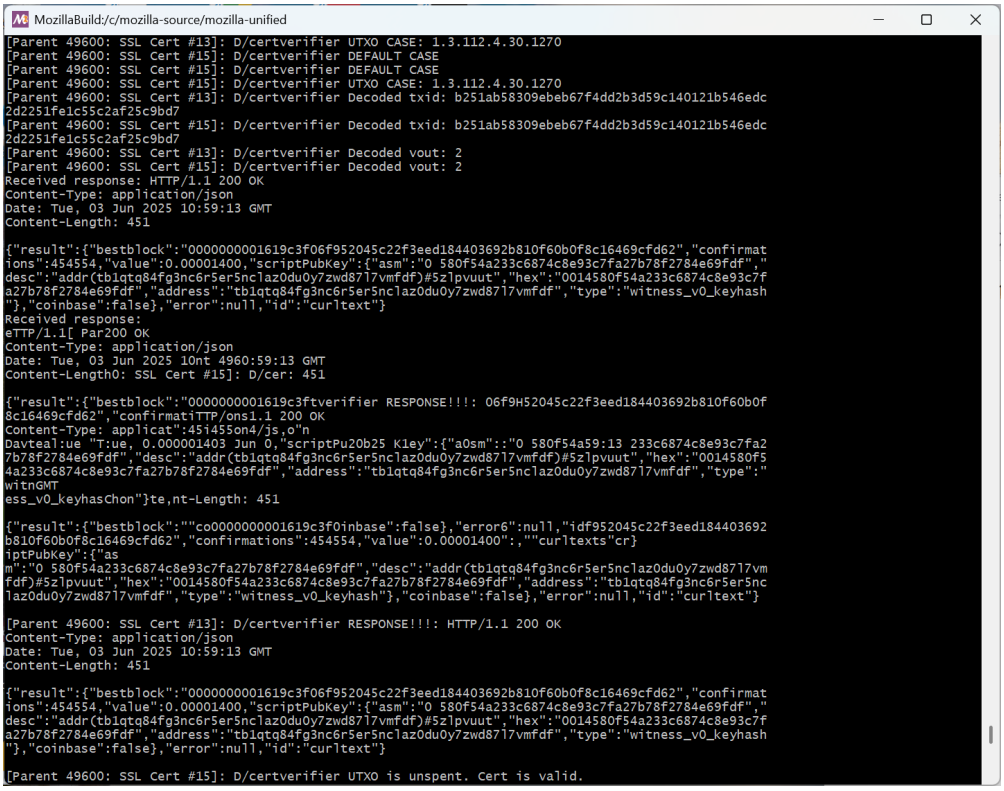

Figure 17: Logs related to the verification of the certificate of website `tfm2.example.com`. First, the UTXO-related OID is identified. Then it is parsed to obtain `txid` and `vout`. After that, a query is sent to the Bitcoin node, and a response with the UTXO details is obtained, indicating that the certificate has not been revoked and therefore it is valid.

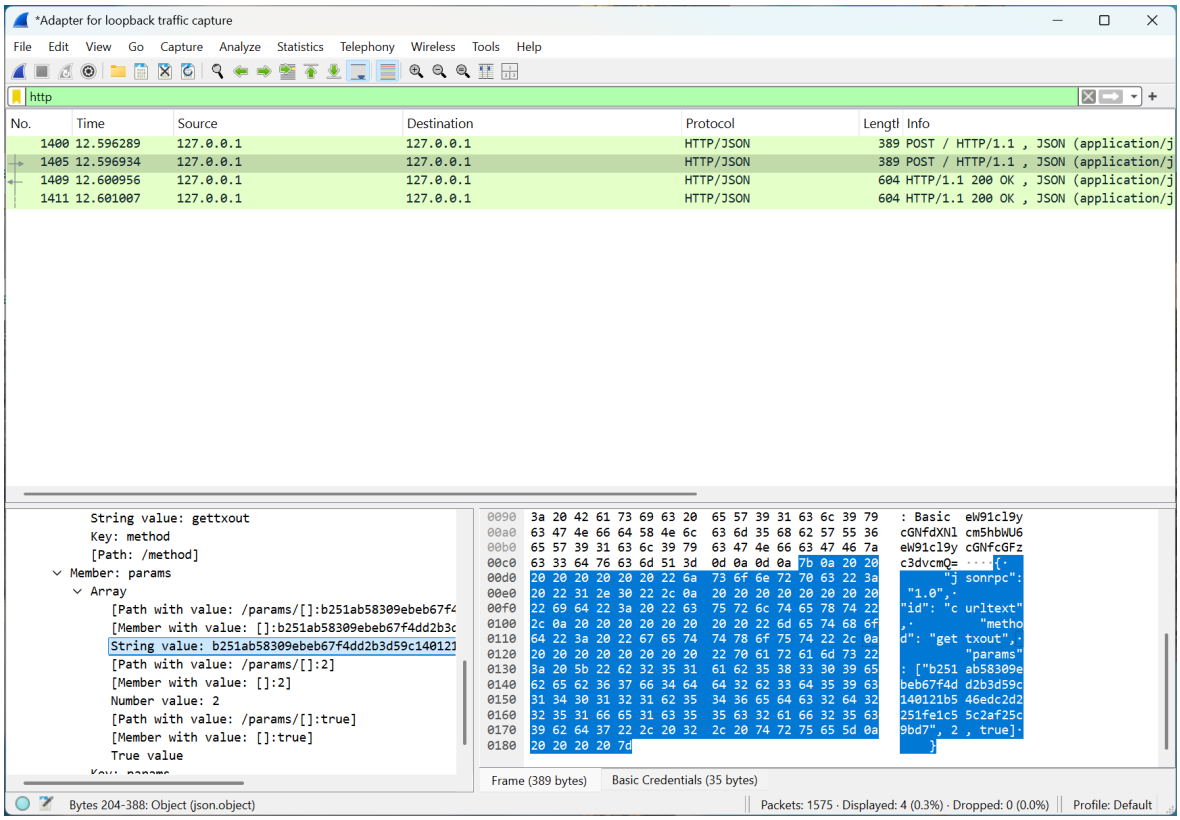

Figure 18: Wireshark capture highlighting the RPC call issued by the Mozilla browser to the Bitcoin node. The capture shows the use of the `gettxout` method, along with the specified `txid` and `vout` parameters used to query the status of the corresponding UTXO.

R3: *Allow certificate revocation.*

Proof: Revocation is implemented by spending the UTXO associated with the certificate. Once the transaction is confirmed on the Bitcoin network, any client validating the certificate will detect the spent status and mark the certificate as revoked. This can be directly done using the Bitcoin core command line.

1. **Identify the UTXO:** the `listunspent` command will show a list of all UTXO belonging to the wallet, as shown in Figure 19.

2. **Create the raw transaction:** with the command

   ```
   createrawtransaction "[{\"txid\":\"<TXID>\",\"vout\":<VOUT>}]"
   "{\"<destination_address>\":<amount>}"
   ```

   If we choose the first UTXO shown in Figure 19, we should execute the following (Figure 20 shows the output of this command):

   ```
   createrawtransaction "[{\"txid\":\"e9eb0ce1acac9a33bede58d3235e14
   dde6065d5f862baad1c856aa97b07302fb\",\"vout\":1}]"
   {\"tb1qyg7dqrh2dyg7743jg5pvsh9reljn5uwsh8yg27\":0.00015}"
   ```

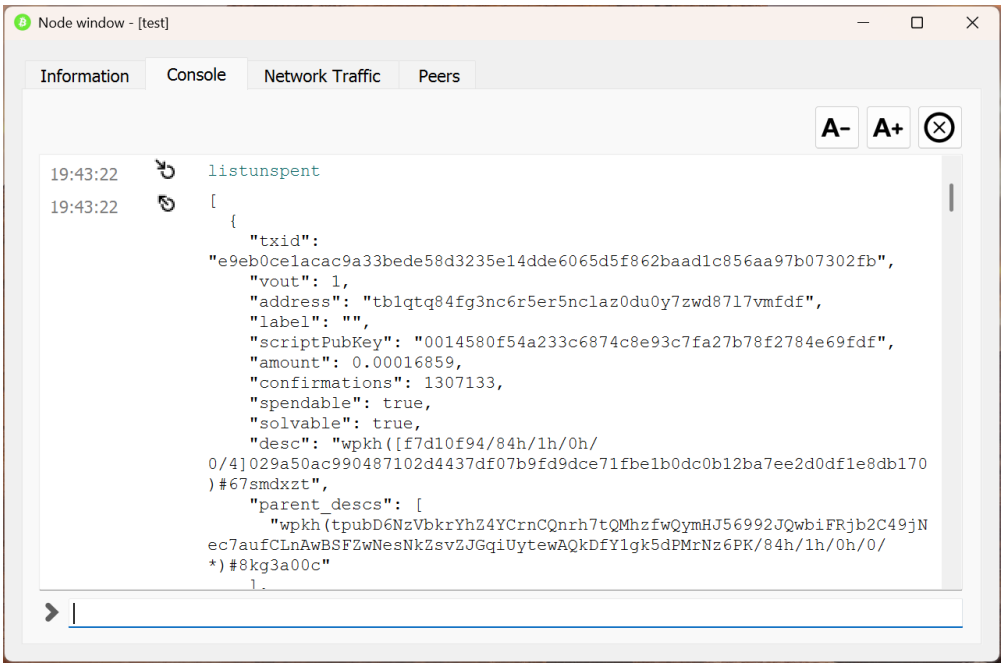

Figure 19: List of available UTXOs.

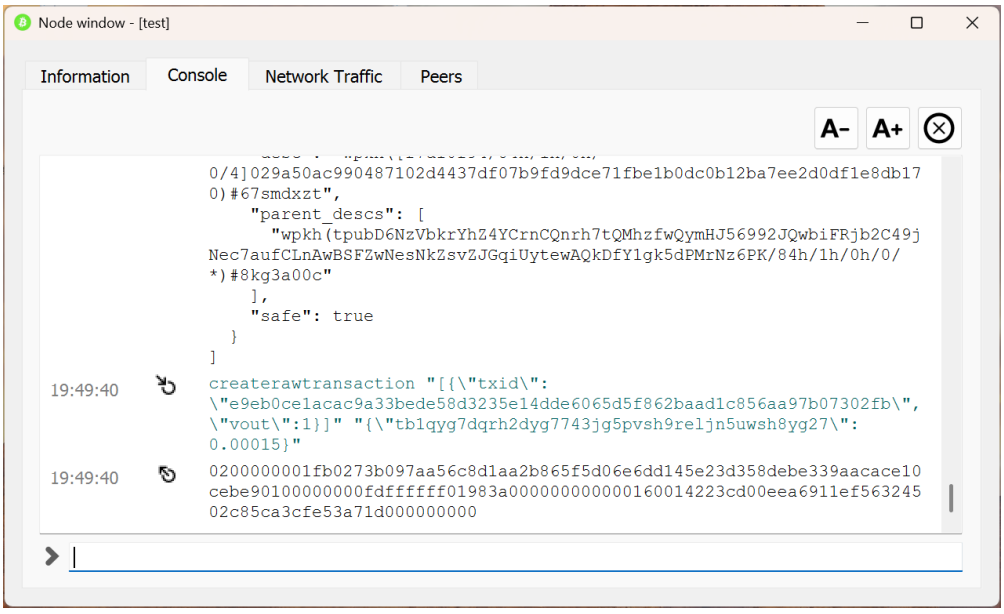

Figure 20: Creation of the raw transaction. UTXO's `txID` and `vout` are specified first. Then, the destination address (generated for the CA wallet) and an amount similar to the UTXO are specified.

3. **Sign the transaction:** running the command `signrawtransactionwithwallet "<rawtx>"`, where `rawtx` must be substituted by the hexadecimal output shown in Figure 20. The output of the signing operation is depicted in Figure 21.

   ```
   signrawtransactionwithwallet "0200000001fb0273b097aa56c8d1aa2b865f5d
   06e6dd145e23d358debe339aacace10cebe90100000000fdffffff01983a00000000
   0000160014223cd00eea6911ef56324502c85ca3cfe53a71d000000000"
   ```

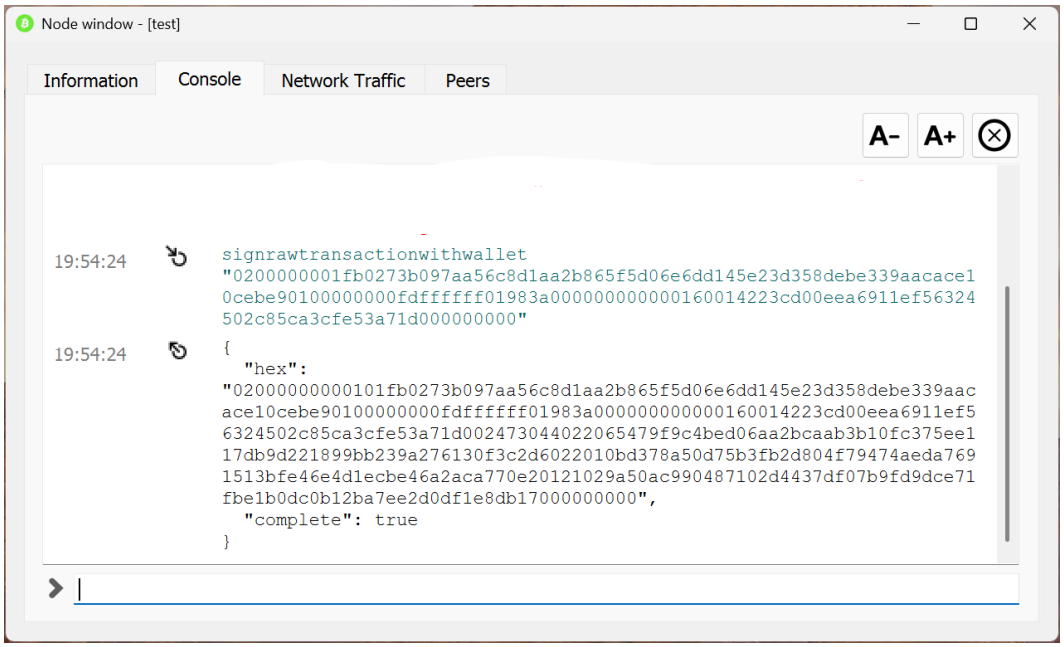

Figure 21: Signature of the raw transaction with a Bitcoin wallet. The hexadecimal string of the raw transaction has to be specified.

4. **Broadcast the transaction:** executing the command `sendrawtransaction "<signed_hex>"`, see Figure 22.

```
sendrawtransaction "02000000000101fb0273b097aa56c8d1aa2b865f5d06e6d
d145e23d358debe339aacace10cebe90100000000fdffffff01983a000000000000
160014223cd00eea6911ef56324502c85ca3cfe53a71d002473044022065479f9c4
bed06aa2bcaab3b10fc375ee117db9d221899bb239a276130f3c2d6022010bd378a
50d75b3fb2d804f79474aeda7691513bfe46e4d1ecbe46a2aca770e20121029a50a
c990487102d4437df07b9fd9dce71fbe1b0dc0b12ba7ee2d0df1e8db17000000000"
```

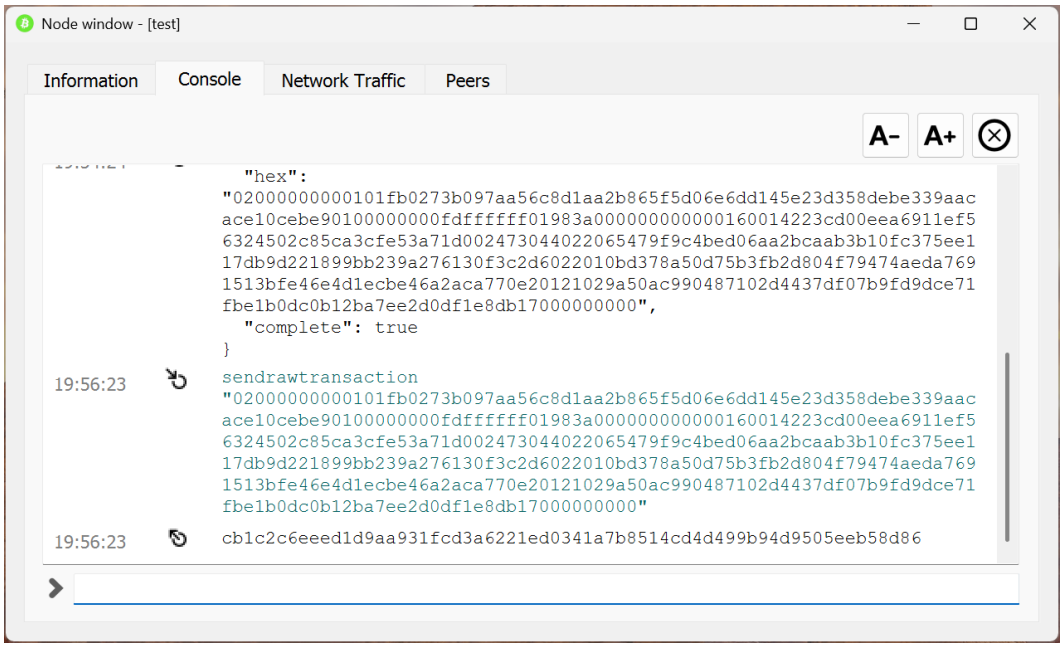

Figure 22: Broadcast of the signed transaction with a Bitcoin wallet.

Lastly, after broadcasting the transaction, we can confirm that the UTXO has been successfully spent by inspecting the account balance. The red negative values shown in Figure 23 clearly illustrate the expenditure of the UTXO.

Figure 23: Balance of the CA wallet in BTC and recent transactions. Recent transactions confirm that the UTXO has been spent and send back to the CA wallet.

R4: *Permit servers to revoke their own certificates autonomously (thus reducing risks when keys are compromised).*

Proof: Because each certificate is associated with a unique UTXO controlled by the entity owning the certificate, the server can unilaterally revoke it by spending the UTXO. This avoids reliance on centralized certificate revocation infrastructure and allows for immediate mitigation in the event of key compromise.

7.2 Security

In this section, we analyze and demonstrate how the system meets its defined security and privacy requirements.

R5: *Ensure that certificate verification is secure and that the TLS connection guarantees no revoked certificate can be mistakenly accepted as valid.*

R6: *Enhance user privacy by avoiding the tracking risks associated with traditional OCSP systems.*

R7: *Provide resilience against single points of failure to guarantee system availability and robustness.*

Proof: While maintaining a local node can be demanding in terms of storage requirements (as discussed in Section 7.3.3), it offers the highest levels of privacy and security by eliminating the need to rely on third parties such as block explorers. It also results in faster response times.

Using the Bitcoin blockchain inherently addresses the single point of failure problem present in traditional OCSP systems, which rely on centralized servers that can be overwhelmed or attacked, potentially allowing revoked certificates to be accepted erroneously. The decentralized nature of the Bitcoin network distributes trust across numerous nodes, ensuring availability and robustness even under attack.

Moreover, since queries are made locally to the Bitcoin node, user privacy is significantly enhanced compared to OCSP, which requires contacting a third-party server that may log browsing activity and thus expose user profiles. This eliminates tracking risks, preserving user anonymity and confidentiality during certificate validation.

To achieve a similar level of trustworthiness and security without the burden of running a local node (and therefore saving disk space), it would be necessary to query multiple remote Bitcoin nodes or block explorers. This redundancy ensures that the result (e.g., whether a specific UTXO has been spent) can be verified by majority consensus, thus reducing the risk of relying on a potentially compromised or inaccurate source.

Aside from the risks inherent to the blockchain (such as 51% attacks and sybil attacks), the main security vulnerability in this system arises during periods of node desynchronization. Certificates that were issued and revoked under the previous synchronization period will remain verifiable without compromising security. However, if the Bitcoin node is not fully updated, there is a risk that a revoked certificate may still be accepted as valid during the desynchronization window. Therefore, it is the user's responsibility to be cautious when browsing while the node is still syncing. In Section 7.3.2, we analyze best- and worst-case scenarios for certificate revocation, showing that a revoked certificate may appear valid for up to 10 minutes after revocation. This time corresponds to the average time for a block to be confirmed on the Bitcoin blockchain.

7.3 Costs

In this section, we examine the inherent costs of our implementation, beginning with economic expenses and finishing with time-related matters.

7.3.1 Economic costs

One of the key advantages of this system lies in its cost distribution across different certificate lifecycle operations. Validation, being the most frequent operation, is entirely free. Any user or application can verify the validity of a certificate by checking the status of the corresponding UTXO on the blockchain, without incurring any cost.

Certificate issuance is also effectively "free" in terms of transaction fees, as it does not require broadcasting a transaction to the network. The only requirement is the commitment of a UTXO, which remains unspent for as long as the certificate is valid. This UTXO acts as a cryptographic proof of existence and validity.

Here, we do not discuss the initial step of setting up a CA, which would be responsible for obtaining a large set of UTXOs (which inherently incurs the initial inversion cost). Estimating the initial number of UTXOs needed to cover the certificate demand is not easy, although insights are given in Section 7.4, where we discuss the feasibility of this system as a replacement of traditional revocation mechanisms.

The only operation that incurs a cost is revocation. Revoking a certificate requires spending the associated UTXO, which means broadcasting a transaction to the Bitcoin network. The cost of revocation depends on multiple factors, including the value of the UTXO, network congestion, and the current transaction fee rate. Therefore, a balance must be found between revocation speed and cost efficiency. For urgent revocations (such as in the case of compromised keys), higher fees may be necessary to ensure inclusion in the next mined block. Conversely, if revocation is not time-critical, lower fees can be used to reduce expenses.

Notably, if an issued certificate has not been revoked and simply expires (i.e., the

certificate turns invalid because of its expiration date), the CA can reuse its associated UTXO to issue a new certificate. This approach helps reduce the cost associated with acquiring new UTXOs. Since CAs can recover their UTXOs after revocation (by spending them to another address they control) they do not need to acquire new UTXOs for every certificate. Instead, new UTXOs only need to be obtained periodically to cover transaction fees, while reclaimed or unused UTXOs can be reused for issuing future certificates.

R10: *Ensure that the monetary costs associated with certificate revocation remain reasonable.*

Proof: As of 2015, over 8% of freshly issued certificates were revoked [34]. This implies that a Certificate Authority would only need to maintain revocation infrastructure for a small subset (approximately 8%) of the total certificates it issues (noting that this percentage reflects a general average across all CAs). As of June 2025, the average Bitcoin transaction fee is approximately 0.000014 BTC, equivalent to $1.42[7]. Assuming a CA issues 1 million certificates per year, the estimated cost of revoking 8% of them (based on the Bitcoin fee) can be calculated as:

$$\text{Transaction fee (in \$)} \times \text{Certificates issued per year} \times \text{Revocation rate} =$$

$$\$1.42 \times 1{,}000{,}000 \times 0.08 = \$113{,}600$$

In comparison, the Google Cloud Certificate Authority Service (Enterprise tier) charges $36,700 per month for issuing 1 million certificates[8], totaling $440,400 per year. Therefore, the proposed system could reduce operational costs to approximately 25% of those incurred using traditional cloud-based CA services.

A practical strategy to diminish the expenditure is to use UTXOs with minimal value for certificate issuance, thereby minimizing the financial impact of revocation. In this project, we have utilized Bitcoin Testnet faucets, as described in Section 6.3, which typically distribute UTXOs with very small amounts of BTC. These small-value UTXOs are ideal for testing and also serve as a model for real-world deployment. In practice, Certificate Authorities and end-users should deliberately source low-value UTXOs to reduce Bitcoin expenditure during revocation. However, users with the ability of revocation must be aware that lower boundaries apply when choosing a UTXO. Mainly, it is not possible to spend a UTXO that is smaller than the transaction fee. Even though one can construct such a transaction, it will be considered invalid and rejected by Bitcoin nodes. Let's consider:

$$\text{Input value} \geq \text{Output value} + \text{Transaction fee}$$

[7] **Bitinfocharts (Bitcoin Transaction Fee)**: https://bitinfocharts.com/comparison/bitcoin-transactionfees.html

[8] **Google Cloud CA pricing:** https://cloud.google.com/certificate-authority-service/pricing

So, if the value of the UTXO is less than the minimum required fee, the result would be a transaction with a negative fee (which is invalid), or a transaction with zero or near-zero output, which may also be rejected for being considered "dust" (too small to be useful).

As a final remark, all monetary costs are bounded by current network congestion and cryptocurrency values, meaning that prices can increase or decrease along with these conditions. Even if the incurred cost went up due to said conditions, users might be willing to pay a higher price for a privacy-preserving system that, unlike OCSP, does not perform tracking. Especially considering that this amount covers the entire duration of the certificate.

7.3.2 Time costs

This section evaluates the time costs incurred by the system. First, we analyze the local Bitcoin node update time, followed by issuance and validation times, and finally, we study revocation time.

Update time To assess the update-related costs, we registered how long it takes to update the Bitcoin blockchain state. This includes both downloading blocks from the peer-to-peer network and validating them, as both processes occur simultaneously.

Table 1 presents the average update durations for various desynchronized intervals. These intervals represent the amount of time the Bitcoin node has been inactive (for example, when the device is turned off or the browser is closed). Each time the validation browser is reopened, the local node must catch up with the network. The typical startup time of the Bitcoin node (excluding block download and validation) is approximately 14.90 seconds, which has been subtracted from each measurement to isolate the actual update time.

Node desync time	Avg update time (s)	Avg - startup time (s)
1 h	22.33	7.44
2 h	35.00	20.10
3 h	42.67	27.77
4 h	50.00	35.10
5 h	53.67	38.77
6 h	60.33	45.44
7 h	71.67	56.77
8 h	75.67	60.77
9 h	77.00	62.10
10 h	80.00	65.10
15 h	90.00	75.10
24 h	121.67	106.77
48 h	270.00	255.10
72 h	346.67	331.77
1 week	707.33	692.44
2 weeks	1498.00	1483.10

Table 1: Average update time and adjusted update time (after subtracting average opening time of 14.90 s) for different desynchronization moments.

Figure 24 provides a clearer visualization of the data, revealing an almost perfectly linear trend. The fitted line follows the equation $y = 4.31x + 16.96$, with a correlation coefficient of $R^2 = 0.998$, indicating an excellent fit.

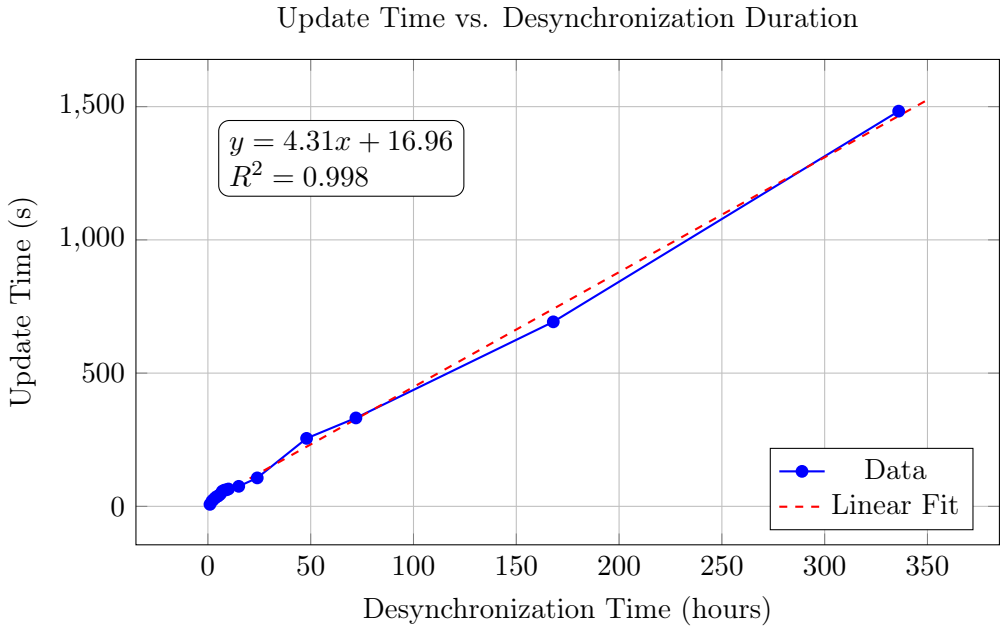

Figure 24: Update time as a function of node desynchronization. Blue points represent the data collected by measuring update times. The dashed red line shows a linear fit with $R^2 = 0.998$.

It is important to note that once synchronization is complete, the Bitcoin node continues to update its state in real time, as long as the browser and the node interface remain active, providing a continuous and up-to-date validation status.

Issuance and Validation Both the certificate issuance and validation processes are almost instantaneous and therefore negligible. Validation, in particular, involves only a simple RPC call to the local Bitcoin node. While maintaining a local node can be demanding in terms of storage requirements (as discussed in Section 7.3.3), it offers the highest levels of privacy and security by eliminating the need to rely on third parties such as block explorers. It also results in faster response times, as answers from different sources don't have to be aggregated.

Revocation In the case of certificate revocation, a small delay may be introduced due to the underlying blockchain confirmation process. Specifically, once the UTXO associated with a certificate is spent to revoke it, the revocation transaction must be included in the next mined block. The best-case scenario occurs when the next block is mined immediately after broadcasting the transaction, ensuring near-instantaneous inclusion in the blockchain. On the other hand, the worst-case scenario happens if a block has just been confirmed right before broadcasting the revocation transaction, resulting in a delay of approximately 10 minutes (the average Bitcoin block generation time[9]) during which verification of the certificate might still yield a valid result instead of revoked.

[9]**Bitinfocharts (Average Bitcoin Block Time):** https://bitinfocharts.com/comparison/bitcoin-confirmationtime.html

Despite this potential delay, this blockchain-based revocation approach is faster than traditional Certificate Revocation Lists (CRLs), which suffer from poor update frequency and latency. Compared to the Online Certificate Status Protocol (OCSP), it may be somewhat slower, since OCSP only requires one query to a trusted server. Although in this blockchain model, verification also requires a single query to the local Bitcoin node, we do have to consider also the time for the blockchain to update and synchronize.

R11: *Guarantee practical and efficient times for certificate issuance, validation, and revocation.*

Proof: In addition to the above analysis, we can conclude that under typical daily conditions, the update time is practical, with synchronization delays generally under two minutes after 24 hours of continuous operation (see Table 1). Furthermore, issuance and validation are instantaneous.

7.3.3 Storage costs

As of 2025, the full Bitcoin mainnet blockchain has grown to approximately 650 GB, effectively doubling in size over the past four years (see trend in Figure 25). Storing the entire blockchain on most devices (especially mobile or wearable ones) is impractical. Therefore, we chose to use a pruned version of the Bitcoin blockchain. For testing purposes, the Bitcoin testnet was used; however, the mainnet should be employed during the production phase of the proposal. This pruned version retains only the most recent portion of the blockchain data, specifically the last few hundred megabytes. The minimum pruning target allowed by Bitcoin Core is 550 MB [35], which excludes the additional space required for the block index and UTXO databases. This minimum ensures that at least the latest 288 blocks (equivalent to about two days, assuming 10 minutes per block) are stored on disk. Occasionally, disk usage may temporarily exceed this target to maintain those 288 blocks. Moreover, since only the UTXO set is essential for our purposes, we can disregard storing full blocks and instead keep just the UTXO set, which is around 12 GB in size (see Figure 26). Then, the required storage for the system follows:

$$\text{size}(\text{UTXO\_set}) + \min(\text{pruning\_size}) = 12\,\text{GB} + 550\,\text{MB} \approx 12.5\,\text{GB}$$

While the storage requirements of the system may initially seem high for portable devices, both PCs and mobile devices are increasingly equipped with larger storage capacities. For example, the average storage space for desktop and laptop computers ranges from 512 GB to 1 TB. In this context, the system would occupy approximately 1.25% to 2.4% of the total disk space. These percentages represent a reasonable trade-off given the benefits of decentralization and privacy preservation.

For portable devices such as smartphones and tablets, average storage ranges from 128 GB to 256 GB. In this case, the system would consume between 4.8% and 9.7% of the available space. Although this is a more significant portion, it may still be acceptable for users who prioritize the system's privacy and decentralization advantages over storage constraints.

It is also important to note that we do not include the storage required for the modified browser, as we assume that all target devices already have a functioning web browser installed by default.

Although the growth in blockchain size is evident in Figure 25, it does not directly impact the system, as only a subset of block data is stored. In contrast, the UTXO set experienced exponential growth from 2023 to 2024, followed by a decline over the past year (Figure 26). As a result, the system must adapt its storage requirements to reflect the current state of the blockchain.

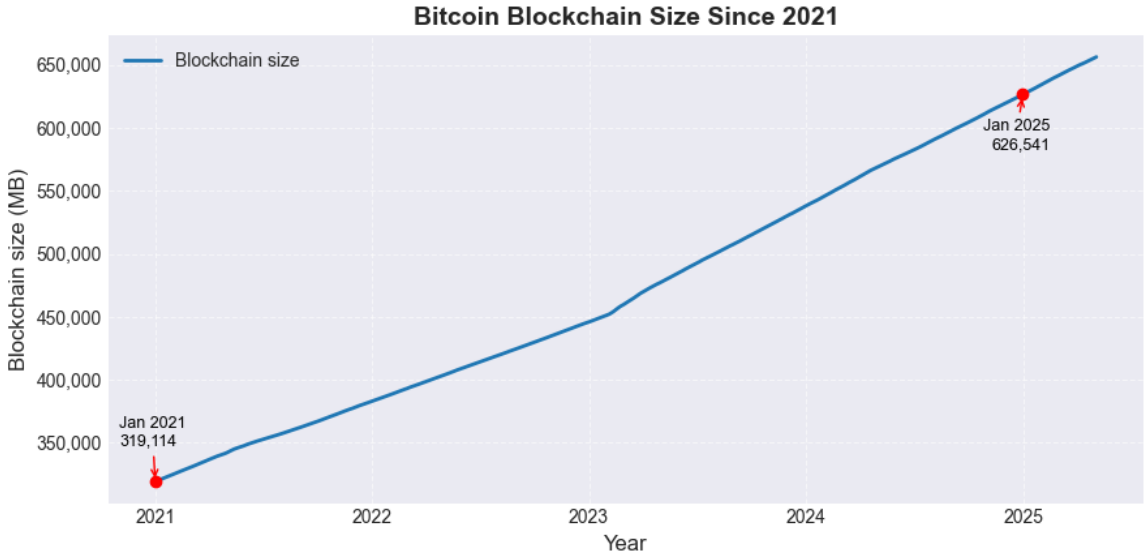

Figure 25: Bitcoin mainnet blockchain size growth from January 2021 to June 2025. Source: https://www.blockchain.com/explorer/charts/blocks-size

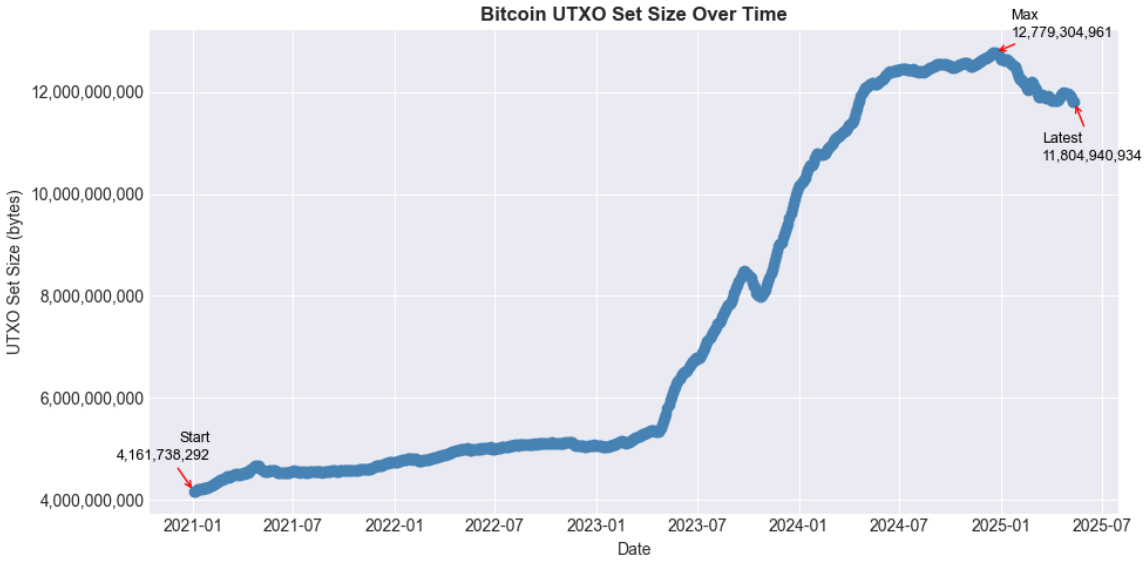

Figure 26: Growth trend of the Bitcoin UTXO set size over time in mainnet (from January 2021 to June 2025). Source: https://statoshi.info/d/000000009/unspent-transaction-output-set

7.4 UTXO-based revocation as a replacement for current systems

To evaluate the feasibility of this system as a potential alternative to existing revocation mechanisms (such as CRL and OCSP), it is essential to compare the number of websites that provide secure connections over TLS with the number of available Bitcoin UTXOs, in order to determine whether the latter could meet the demand. In 2021, over 46 million websites used SSL by default [10]. However, with the exponential growth of the Internet and its increasing focus on security, this number has surged to more than 302 million as of January 2025 (more than double the current number of available UTXOs [11]).

In 2021, Bitcoin had approximately 84.3 million UTXOs, which could have supported SSL for every secure website at the time, given that this number was nearly twice the amount of SSL-enabled domains. Today, despite Bitcoin's increased adoption, there are only around 175 million UTXOs (refer to Figure 27 to observe the trend), which is sufficient to cover just over half of the current SSL-enabled websites.

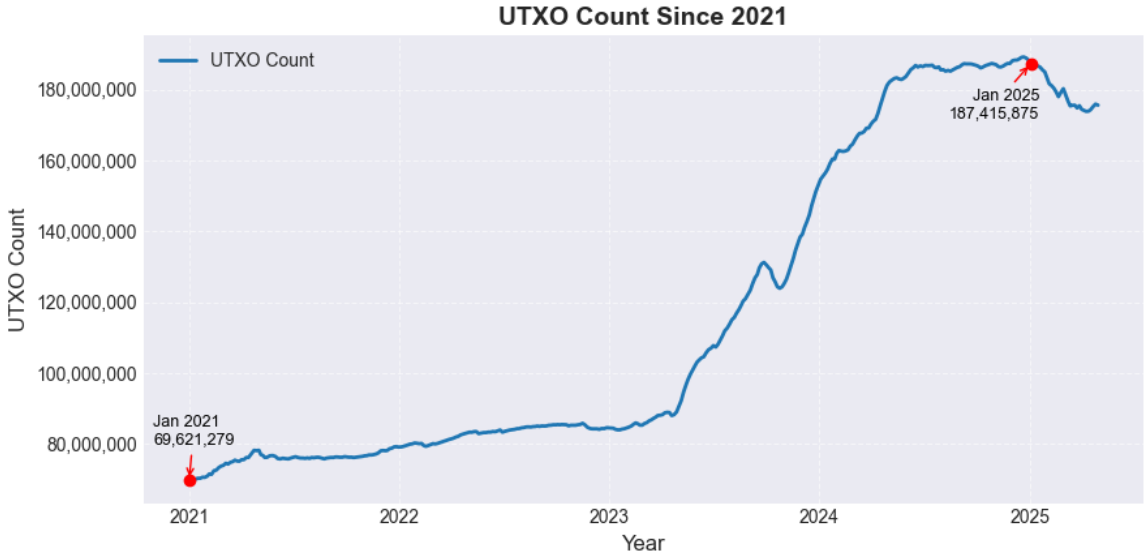

Figure 27: Number of unspent transaction outputs (UTXOs) in the Bitcoin mainnet from 2021 to 2025, shown in millions.

This suggests that while the proposed system could serve as a viable solution for securing a significant portion of the Internet, it is not yet capable of fully replacing traditional SSL infrastructure. Therefore, it should be considered a complementary alternative rather than a full-scale replacement.

Alternatively, a Certificate Authority could issue additional transactions to subdivide its UTXO into multiple smaller ones. This approach offers two main benefits: it helps address potential UTXO shortages and reduces the cost of certificate revocation. However, it also introduces a drawback. There must be a minimum value threshold for acceptable UTXOs to ensure that their value does not fall below the transaction fee required by miners, which has already been discussed in Section 7.3.1.

[10] **SSL stats (2021):** https://serpwatch.io/blog/ssl-stats/
[11] **SSL stats (January 2025):** https://sslinsights.com/ssl-certificates-statistics/

8 Conclusions

In this work, we have proposed and implemented a novel certificate revocation mechanism based on Bitcoin's UTXO model, embedded directly into the X.509 v3 certificate standard. Our approach addresses several key shortcomings of traditional systems like CRLs and OCSP, particularly regarding privacy, decentralization, and resilience against single points of failure.

Using the decentralized infrastructure of the Bitcoin blockchain, the proposed system allows certificate revocation to be enforced by spending an associated UTXO. This removes the reliance on trusted third-party responders and enhances user privacy by enabling local verification through a Bitcoin node. Our implementation within Mozilla Firefox demonstrates that it is possible to embed and validate UTXO references during the TLS handshake, offering a seamless experience comparable to traditional validation mechanisms.

Evaluation results show that the system performs well in terms of security, privacy, and time efficiency. Storage requirements, while non-trivial, are manageable, especially when using pruned nodes and relying solely on the UTXO set. Moreover, monetary costs for revocation remain low when using UTXOs with minimal value and given the fact that UTXO costs will only need to be covered equally to the transaction fees, as UTXOs will be recycled.

Ultimately, the results validate the feasibility of a blockchain-based revocation model that is both privacy-respecting and compatible with existing X.509 PKI infrastructure.

8.1 Future work

While the presented system proves functional and secure, there are several areas where further development and research could enhance its capabilities:

- **Integration with other browsers:** Although the current implementation targets Mozilla Firefox, extending compatibility to other browsers (e.g., Chromium-based) would significantly broaden the system's applicability and facilitate real-world adoption.
- **System optimization:** While pruned nodes effectively reduce storage overhead, further efforts are needed to optimize synchronization times and storage burdens, especially in low-resource environments such as mobile and embedded devices.
- **Bilateral revocation:** This implementation focuses on unilateral revocation, where either only the user or only the CA can revoke the certificate, depending on which party's UTXO is embedded. Introducing bilateral revocation (allowing both the user and the CA to revoke independently) could improve protocol robustness. This would require some form of cooperation between both parties (e.g., shared secrets or joint authorization), making it particularly valuable in scenarios where mutual agreement is essential to prevent unauthorized revocations.
- **Automatic UTXO and certificate management:** Implementing automated UTXO recycling mechanisms would mitigate the risk of UTXO exhaustion and

simplify revocation processes for both servers and CAs. Additionally, developing tooling to support automated CSR generation and certificate issuance would improve system usability and facilitate adoption by non-expert users.

8.2 Reflections on the ethical and deontological aspects

The development of this project, which integrates computer security techniques and interacts with decentralized blockchain technologies, raises several ethical and deontological considerations that must be acknowledged and addressed.

- **Transparency:** The system aims to enhance transparency in certificate revocation by anchoring revocation information on a public and auditable blockchain. This allows any user or verifier to independently confirm the status of a certificate, removing opaque or centralized decision-making mechanisms.
- **Privacy:** One of the core motivations of the proposal is to improve user privacy compared to traditional mechanisms like OCSP, which may leak browsing behavior to third parties. By relying on local validation through a Bitcoin node, the system avoids exposing personal data to external servers, reinforcing respect for user privacy.
- **Responsibility and Security:** Developers of this kind of security infrastructure carry the responsibility to ensure that it functions reliably and does not unintentionally compromise user safety. This includes making users aware of potential risks, such as operating with an unsynchronized node, as well as applying best practices to secure private keys and blockchain access.
- **Fairness:** Although the system does not involve AI or machine learning components that could introduce algorithmic bias, fairness is still relevant in ensuring that all entities (CAs, users, and servers) have equal capabilities to revoke and validate certificates. The future extension to support bilateral revocation enhances this principle by promoting shared control.
- **Social Impact:** By removing central points of failure and increasing control over digital identities, the system empowers individuals and reduces dependency on centralized authorities. This decentralization can contribute to more democratic and resilient digital infrastructures.
- **Sustainability:** In terms of environmental impact, the use of Bitcoin (particularly its proof-of-work consensus mechanism) is a known source of high energy consumption. While this project leverages the Bitcoin blockchain for revocation, it does so in a lightweight manner, querying only the UTXO set and requiring minimal transactions. Moreover, the possibility of using pruned nodes significantly reduces storage and processing requirements.

In summary, this project has been developed with a strong ethical foundation, aiming to enhance transparency, privacy, and decentralization while being mindful of environmental impact and user autonomy.

References

[1] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. URL https://www.rfc-editor.org/info/rfc8446.

[2] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, page 475–488, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450332132. doi: 10.1145/2663716.2663755. URL https://doi.org/10.1145/2663716.2663755.

[3] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008. URL https://www.rfc-editor.org/info/rfc5280.

[4] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Dr. Carlisle Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, June 2013. URL https://www.rfc-editor.org/info/rfc6960.

[5] Paul C. Kocher. On certificate revocation and validation. In Rafael Hirchfeld, editor, *Financial Cryptography*, pages 172–177, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-53918-6.

[6] M. Naor and K. Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications*, 18(4):561–570, 2000. doi: 10.1109/49.839932.

[7] Petra Wohlmacher. Digital certificates: a survey of revocation methods. In *Proceedings of the 2000 ACM Workshops on Multimedia*, MULTIMEDIA '00, page 111–114, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581133111. doi: 10.1145/357744.357892. URL https://doi.org/10.1145/357744.357892.

[8] Alexander Yakubov, Wazen M. Shbair, Anders Wallbom, David Sanda, and Radu State. A blockchain-based pki management framework. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6, 2018. doi: 10.1109/NOMS.2018.8406325.

[9] Hong-Sheng Huang, Cheng-Che Chuang, Jhih-Zen Shih, Hsuan-Tung Chen, and Hung-Min Sun. An enhanced online certificate status protocol for public key infrastructure with smart grid and energy storage system, 2024. URL https://arxiv.org/abs/2409.10929.

[10] Emin Topalovic, Brennan Saeta, Lin-Shung Huang, Collin Jackson, and Dan Boneh. Towards short-lived certificates. Web, 2012.

[11] Junzhi Yan, Xiaoyong Hang, Bo Yang, Li Su, and Shen He. Blockchain based pki and certificates management in mobile networks. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1764–1770, 2020. doi: 10.1109/TrustCom50675.2020.00242.

[12] A. Yakubov, W. Shbair, A. Wallbom, D. Sanda, and R. State. A blockchain-based pki management framework. In *The First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) colocated with IEEE/IFIP NOMS*, Taipei, Taiwan, 2018.

[13] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar. Certledger: A new pki model with certificate transparency based on blockchain. *Computers & Security*, 85: 333–352, 2019. URL https://www.sciencedirect.com/science/article/pii/S0167404818313014.

[14] A. Garba, Q. Hu, Z. Chen, and M. R. Asghar. Bb-pki: Blockchain-based public key infrastructure certificate management. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 824–829, 2020.

[15] J. Zhao, Z. Lin, X. Huang, Y. Zhang, and S. Xiang. Trustca: Achieving certificate transparency through smart contract in blockchain platforms. In *2020 International Conference on High Performance Big Data and Intelligent Systems (HPBDIS)*, pages 1–6, 2020.

[16] A. Gayathiri, J. Jayachitra, and S. Matilda. Certificate validation using blockchain. In *2020 7th International Conference on Smart Structures and Systems (ICSSS)*, pages 1–4, 2020.

[17] B. Liu, L. Xiao, J. Long, M. Tang, and O. Hosam. Secure digital certificate-based data access control scheme in blockchain. *IEEE Access*, 8:91751–91760, 2020.

[18] M. Toorani and C. Gehrmann. A decentralized dynamic pki based on blockchain. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21)*, pages 1646–1655, New York, NY, USA, 2021. Association for Computing Machinery. doi: 10.1145/3412841.3442038.

[19] Aisong Zhang and Xinxin Ma. Decentralized digital certificate revocation system based on blockchain. *Journal of Physics: Conference Series*, 1069(1):012125, aug 2018. doi: 10.1088/1742-6596/1069/1/012125. URL https://dx.doi.org/10.1088/1742-6596/1069/1/012125.

[20] Sai Medury, Anthony Skjellum, Richard R. Brooks, and Lu Yu. Scraaps: X.509 certificate revocation using the blockchain-based scrybe secure provenance system. In *2018 13th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 145–152, 2018. doi: 10.1109/MALWARE.2018.8659364.

[21] Yves Christian Elloh Adja, Badis Hammi, Ahmed Serhrouchni, and Sherali Zeadally. A blockchain-based certificate revocation management and status verification system. *Computers Security*, 104:102209, 2021. ISSN 0167-4048. doi:

https://doi.org/10.1016/j.cose.2021.102209. URL https://www.sciencedirect.com/science/article/pii/S016740482100033X.

[22] Qurotul Aini, Eka Purnama Harahap, Nuke Puji Lestari Santoso, Siti Nurindah Sari, and Po Abas Sunarya. Blockchain based certificate verification system management. *APTISI Transactions on Management*, 7(3):184–193, Nov. 2022. doi: 10.33050/atm.v7i3.1922. URL https://ijc.ilearning.co/index.php/ATM/article/view/1922.

[23] Xiaoxue Ge, Liming Wang, Wei An, Xiaojun Zhou, and Benyu Li. Crchain: An efficient certificate revocation scheme based on blockchain. In Yongxuan Lai, Tian Wang, Min Jiang, Guangquan Xu, Wei Liang, and Aniello Castiglione, editors, *Algorithms and Architectures for Parallel Processing*, pages 453–472, Cham, 2022. Springer International Publishing. ISBN 978-3-030-95388-1.

[24] Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, and Ruiying Du. Certchain: Public and efficient certificate audit based on blockchain for tls connections. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 2060–2068, 2018. doi: 10.1109/INFOCOM.2018.8486344.

[25] S. Haber and W. S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3:99–111, 1991. doi: 10.1007/BF00196791. URL https://doi.org/10.1007/BF00196791.

[26] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL https://www.bitcoin.org/bitcoin.pdf. Email: satoshin@gmx.com.

[27] Ethereum. What is ethereum?, 2024. URL https://ethereum.org/en/what-is-ethereum/. Accessed: 2025-04-23.

[28] Bitcoin Wiki contributors. Bitcoin address – bitcoin wiki, 2024. URL https://bitcoinwiki.org/wiki/bitcoin-address#What.27s_in_a_Bitcoin_address. Accessed: 2025-04-23.

[29] Strike Team. What is a bitcoin address?, 2024. URL https://strike.me/learn/what-is-a-bitcoin-address/. Accessed: 2025-04-23.

[30] Sergi Delgado-Segura, Cristina Pérez-Solà, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí. Analysis of the bitcoin utxo set. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *Financial Cryptography and Data Security*, pages 78–91, Berlin, Heidelberg, 2019. Springer Berlin Heidelberg. ISBN 978-3-662-58820-8.

[31] Mozilla Security Engineering Team. Exciting updates to certificate verification in gecko, April 2014. URL https://blog.mozilla.org/security/2014/04/24/exciting-updates-to-certificate-verification-in-gecko/. Accessed: 2025-06-01.

[32] Ricky Publico in GlobalSign. What is a certificate signing request (csr)?, 2024. URL https://www.globalsign.com/en/blog/what-is-a-certificate-signing-request-csr. Accessed: 2025-05-22.

[33] Cristòfol Daudén-Esmel, Jordi Castellà-Roca, Alexandre Viejo, and Ignacio Miguel-Rodríguez. Multi-platform wallet for privacy protection and key recovery in decentralized applications. *Blockchain: Research and Applications*, 6(1):100243, 2025. ISSN 2096-7209. doi: https://doi.org/10.1016/j.bcra.2024.100243. URL https://www.sciencedirect.com/science/article/pii/S2096720924000563.

[34] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. An end-to-end measurement of certificate revocation in the web's pki. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, page 183–196, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450338486. doi: 10.1145/2815675.2815685. URL https://doi.org/10.1145/2815675.2815685.

[35] Bitcoin Core Developers. Bitcoin core 0.11.0 release notes – block file pruning. https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md#block-file-pruning, 2015. URL https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md#block-file-pruning. Accessed: 2025-06-02.

Appendix A: OCSP server configuration

This appendix constitutes the steps to configure an OCSP server. The OCSP server was created to test the behaviour of the Mozilla Firefox browser in terms of certificate validation before applying modifications to allow UTXO-based validations.

1. **Generate keys and root CA certificate** (refer to Appendix B)
2. **Create CSR and Certificate for a testing server** (refer to Appendix C).
3. **Create and activate OCSP server**. To do so, we must specify `extendedKeyUsage =OCSPSigning`, which is already defined in the `[v3_OCSP]` section of the `opensslPKI.cnf` file (Appendix D).

```
openssl req -new -nodes -out ocspSigning.csr -keyout \
    ocspSigning.key
openssl ca -keyfile rootCA.key -cert rootCA.crt -in \
    ocspSigning.csr -out ocspSigning.crt -config opensslPKI.conf
openssl ocsp -index PKI/index.txt -port 8080 -rsigner \
    ocspSigning.crt -rkey ocspSigning.key -CA TrustURV.crt \
    -text -out log.txt &
```

4. **Access the testing server's main page from Mozilla**. Figure 28 shows that the testing server's page is accessible and has passed the OCSP validation.

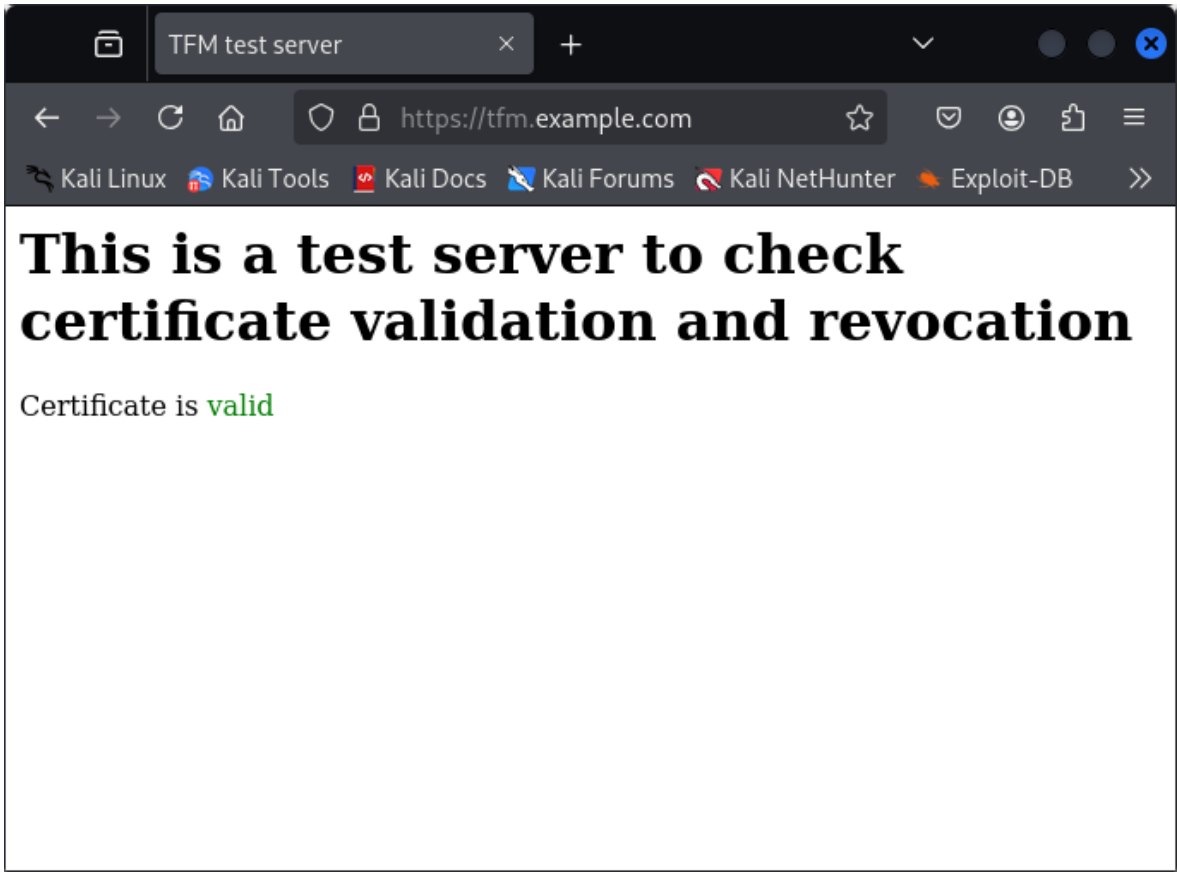

Figure 28: Testing server landing page. A secure connection is provided.

More precisely, Figure 29 shows the packets captured by the Wireshark tool that correspond to the OCSP request and response exchanges used to validate the certificate. Specifically, the response from the OCSP server indicates a status of `good`, confirming that the certificate has not been revoked.

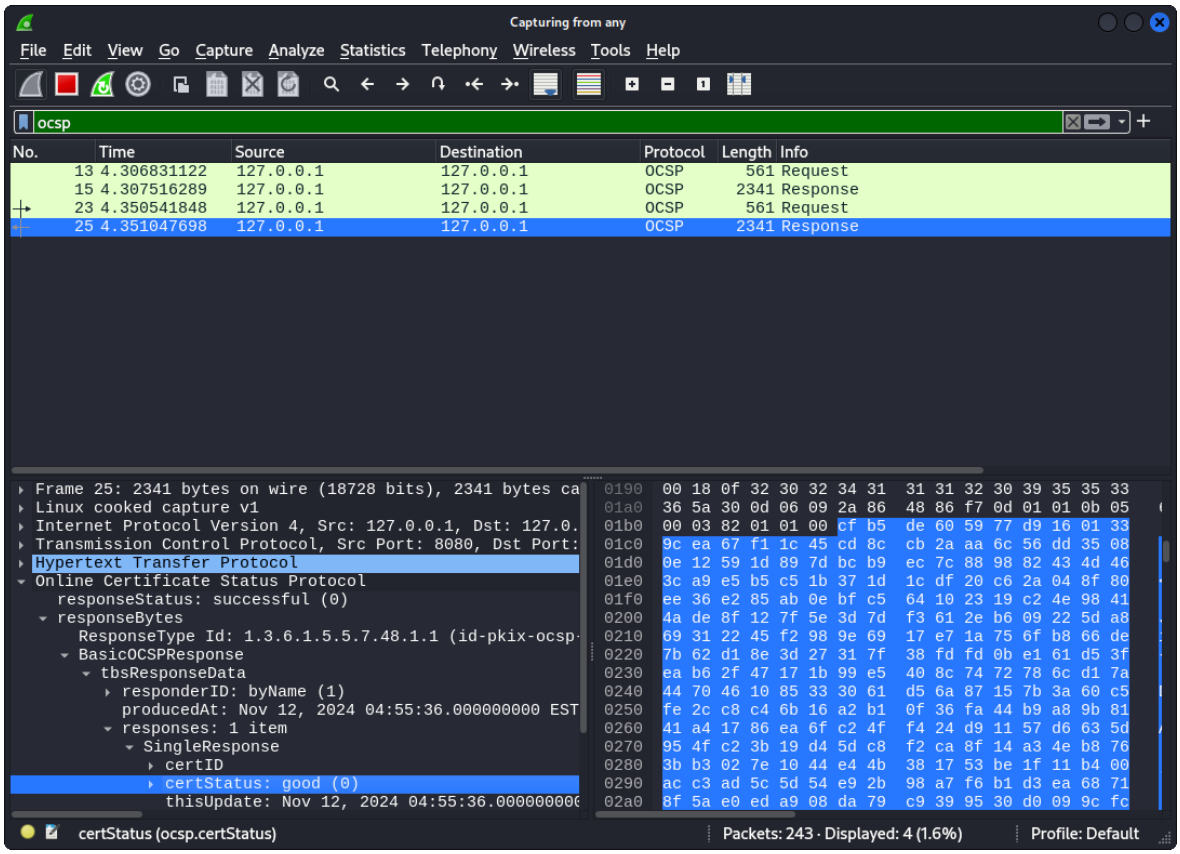

Figure 29: Wireshark capture of the OCSP protocol during the validation of a valid testing certificate. The highlighted section shows the OCSP server's response, indicating a `good` status, meaning the certificate is valid and has not been revoked.

5. **Revoke Server certificate**. This way, we will be able to test the behavior of Mozilla Firefox and the OCSP server when dealing with a revoked certificate.

   ```
   openssl ca -config opensslPKI.cnf -revoke tfm.crt
   ```

6. **Reload OCSP Server**

   ```
   openssl ocsp -index PKI/index.txt -port 8080 -rsigner \
       ocspSigning.crt -rkey ocspSigning.key -CA TrustURV.crt \
       -text -out log.txt &
   ```

7. **Reaccess the testing server's main page from Mozilla**. In Figure 30, the browser flags the website as revoked, indicating that a secure connection cannot be established. This behavior is further confirmed by the Wireshark capture in Figure 31, where the highlighted OCSP response explicitly shows that the certificate has been revoked.

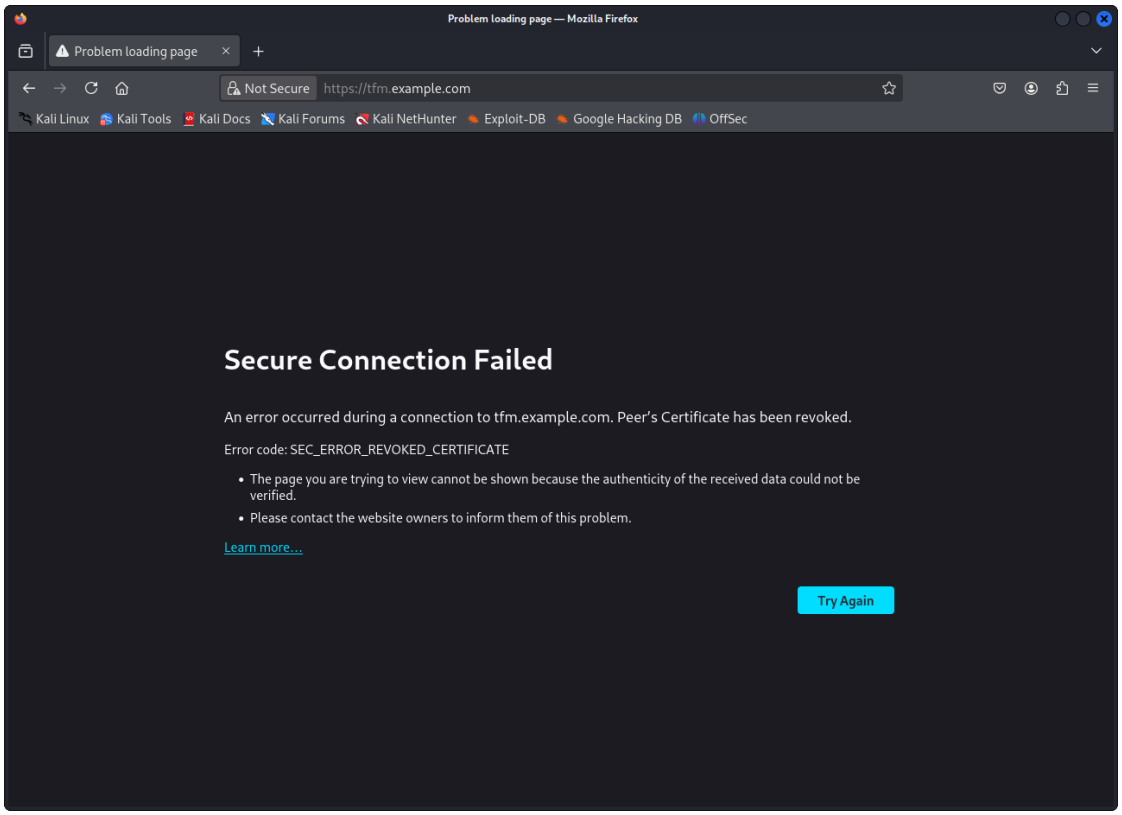

Figure 30: Testing server landing page. Secure connection failed due to revoked certificate.

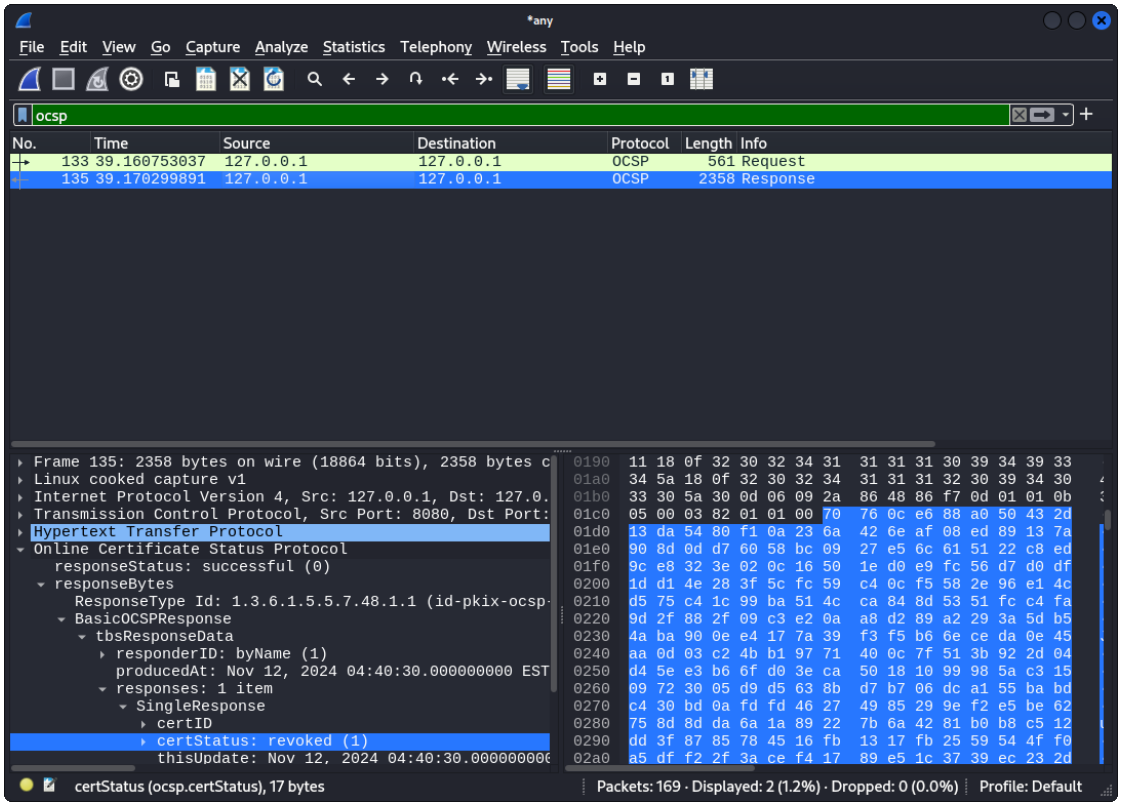

Figure 31: Wireshark capture of the OCSP protocol during the validation of a revoked testing certificate. The highlighted section shows the OCSP server's response, indicating a `revoked` status, meaning the certificate is invalid.

Appendix B: CA configuration

1. **Create a private key for the CA**

   We will encrypt the key with AES-256 for additional protection. If someone gains access to the key, they could create trusted certificates. Although a stronger password is advised, here for educational purposes, we will use `Abcd1234@#`.

   ```
   CANAME=TrustURV

   # Optional: create and enter the CA directory
   mkdir $CANAME
   cd $CANAME

   # Generate AES-256 encrypted private key (4096 bits)
   openssl genrsa -aes256 -f4 -passout pass:Abcd1234@# -out TrustURV.key 4096
   ```

2. **Create a Certificate for the CA**

   This will be the root certificate (and thus self-signed). The certificate will be valid for 730 days (2 years). You can either fill in the fields interactively or pass them as arguments.

   ```
   # Interactive mode
   openssl req -x509 -new -nodes -key $CANAME.key \
     -sha256 -days 1826 -out $CANAME.crt

   # Or non-interactive, providing subject details
   openssl req -new -sha256 -x509 -key $CANAME.key \
   -out $CANAME.crt -days 730 -passin pass:Abcd1234@# -config \
   opensslPKI.cnf -extensions v3_ca -subj \
   "/C=ES/ST=Catalonia/L=Tarragona/O=Universitat Rovira i Virgili/ \
   OU=DEIM/CN=TFM Bitcoin Certificate Revocation"
   ```

   The `opensslPKI.cnf` file has information about the extensions that are necessary for the certificate. A detailed view of this file is given in Appendix D.

3. **Import Root CA Certificate into Mozilla Firefox**

   Since the root CA certificate is self-signed, we will have to import it directly as a trusted entity in the browser. Use the browser's certificate manager (under Settings → Privacy & Security → Certificates) to import the CA certificate you just created.

Appendix C: CSR and Certificate issuance

C.1 Creating the Certificate Signing Request (CSR)

1. **Create a private key for the server**

```
MYCERT=tfm
openssl genrsa -aes256 -passout pass:Abcd1234@# -out $MYCERT.key 2048
```

2. **Create a Certificate Signing Request (CSR) for the Web Server**

```
openssl req -new -sha256 -config opensslPKI.cnf -key $MYCERT.key -out
$MYCERT.csr -passin pass:Abcd1234@#
```

3. **Create a v3 extension file for SAN (Subject Alternative Names)**

   The necessary information for a certificate to be valid for use within servers can be introduced directly in the `opensslPKI.cnf` configuration file.

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = tfm.example.com
IP.1 = 192.168.0.1
```

C.2 Certificate issuance by the CA

Here, we indicate the necessary extensions indicated in the `opensslPKI.cnf` configuration file under the id `usr_crt`. For more information on extensions, please refer to Appendix D.

```
openssl ca -config opensslPKI.cnf -extensions usr_cert -out \
$MYCERT.crt -passin pass:Abcd1234@# -infiles $MYCERT.csr
```

Appendix D: Certificate issuance configuration file

The following configuration file is used to generate self-signed certificates (e.g., for root CAs), Certificate Signing Requests (CSRs, typically for servers), and server certificates issued by CAs. Each certificate type is associated with a dedicated section in the file, which can be customized to include or remove extensions based on the required key usages and certificate purposes.

The main innovation in this configuration lies in the definition of a custom object identifier (OID) within the `utxo_section`. This OID allows a UTXO to be embedded directly into both the CSR and the resulting certificate. To include it, one simply specifies the OID and a reference to the corresponding section, for example:

```
1.3.112.4.30.1270 = ASN1:SEQUENCE:utxo_section
```

```
#
# OpenSSL example configuration file.

# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = .
#RANDFILE               = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:

[ utxo_section ]
# In this section, we must specify the details of the UTXO in
txid = FORMAT:HEX,OCTETSTRING:\
b251ab58309ebeb67f4dd2b3d59c140121b546edc2d2251fe1c55c2af25c9bd7
vout = INTEGER:2

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

####################################################################
[ ca ]
default_ca      = CA_default
        # The default ca section

####################################################################
[ CA_default ]

dir             = ./PKI # Where everything is kept
certs           = $dir/certs            # Where the issued certs are \
    kept
crl_dir         = $dir/crl              # Where the issued crl are \
    kept
database        = $dir/index.txt        # database index file.
new_certs_dir   = $dir/newcerts         # default place for new certs.
```

```
certificate     = $certs/TrustURV.crt # The CA certificate
serial          = $dir/serial           # The current serial number
crl             = $dir/crl.pem          # The current CRL
crlnumber       = $dir/crlnumber        # The CRL number
private_key     = $dir/private/TrustURV.key # The private key
RANDFILE        = $dir/private/randFile # private random number file

x509_extensions = usr_cert              # The extentions to add to \
    the cert

# Extensions to add to a CRL. Note: Netscape communicator chokes on \
    V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crl_extensions        = crl_ext

default_days    = 180                   # how long to certify for
default_crl_days= 30                    # how long before next CRL
default_md      = sha256                   # which md to use.
preserve        = yes           # keep passed DN ordering

copy_extensions = copy

# A few difference way of specifying how similar the request should \
    look
# For type CA, the listed attributes must be the same, and the \
    optional
# and supplied fields are just that :-)
policy          = policy_match

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

####################################################################
[ req ]
default_bits            = 2048
default_keyfile         = privkey.pem
distinguished_name      = req_distinguished_name
attributes              = req_attributes
x509_extensions = v3_ca # The extensions to add to the self-signed \
```

```
    cert

# This sets a mask for permitted string types. There are several \
    options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString.
# utf8only: only UTF8Strings.
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: current versions of Netscape crash on BMPStrings or \
    UTF8Strings
# so use this option with caution!
string_mask = nombstr

req_extensions = v3_req # The extensions to add to a certificate \
    request


[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = ES
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Catalunya

localityName                    = Locality Name (eg, city)
localityName_default            = Tarragona

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Universitat Rovira i Virgili

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, \
    company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, \
    section)
organizationalUnitName_default  = DEIM

commonName                      = Common Name (eg, YOUR name)
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_max                = 50

# SET-ex3                       = SET extension number 3

[ req_attributes ]
challengePassword               = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName                = An optional company name

[ usr_cert ]
```

```
# These extensions are added when 'ca' signs a request.

# requires this to avoid interpreting an end-user certificate as a CA.

basicConstraints=CA:FALSE

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
authorityInfoAccess = OCSP;URI:http://127.0.0.1:8080/

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
subjectAltName=@alt_names

# Custom OID extension for UTXO-based validation
1.3.112.4.30.1270 = ASN1:SEQUENCE:utxo_section

keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage= serverAuth, clientAuth, codeSigning, emailProtection
#extendedKeyUsage= clientAuth, emailProtection


# Here are some examples of the usage of nsCertType. If it is omitted
# The certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
 nsCertType                      = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
nsCertType = client, server, email

# This will be displayed in Netscape's comment listbox.
nsComment                       = "TFM"


[ alt_names ]

DNS.1 = tfm.example.com
IP.1 = 127.0.0.1

[ v3_OCSP ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = OCSPSigning

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
1.3.112.4.30.1270 = ASN1:SEQUENCE:utxo_section
```

```
[ easy ]

basicConstraints = critical, CA:TRUE
keyUsage = keyCertSign, cRLSign
subjectKeyIdentifier = hash

[ v3_ca ]
# Extensions for a typical CA


subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
basicConstraints = critical,CA:TRUE

# Key usage: this is typical for a CA certificate. However since it \
    will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
keyUsage =  keyCertSign
extendedKeyUsage =serverAuth, OCSPSigning

# Some might want this also
nsCertType = sslCA, emailCA
```