University of Brasília

Computer Science Department

DNS-Nostr-Tokens: A Decentralized Domain Naming System Built on Bitcoin and Nostr

Emanuel Medeiros and Luigi Minardi

Monograph submitted in partial fullfilment of the requirements to the Computer Engineering Program at University of Brasilia

Advisor

Prof. José Edil Guimarães

Brasília

2025

University of Brasília

Computer Science Department

DNS-Nostr-Tokens: A Decentralized Domain Naming System Built on Bitcoin and Nostr

Emanuel Medeiros and Luigi Minardi

Monograph submitted in partial fullfilment of the requirements to the Computer Engineering Program at University of Brasilia

Prof. José Edil Guimarães (Advisor)
ENE/UnB

Prof. João José Costa Gondim
ENE/UnB

Prof. Cláudia Jacy Barenco Abbas
ENE/UnB

Prof. Guilherme Novaes Ramos
Coordinator of Computer Engineering Program at University of Brasilia

Brasília, July 07, 2025

Dedicated To

The following dedication is written by Emanuel.

To my beloved wife Raquel, whose love, patience, and encouragement sustained me throughout this journey. To my parents, Julie and Marcio, who instilled in me the values of perseverance and curiosity that shaped my path. To my grandfather, Silva Filho, and to my grandmother, Lindalva, who not only inspired me with their wisdom and strength, but also helped raise me and supported me with unwavering care. This work is as much theirs as it is mine.

The following dedication is written by Luigi.

For Ana Clara, my beloved bride, whose support was my foundation.

For my daughter, Rafaela Minardi, who has been my greatest incentive.

And for my parents, Irineu Maia and Elaine Maia, with my deepest love and gratitude.

Acknowledgement

We would like to express our deepest gratitude to our professor, Edil Medeiros, for introducing us to Bitcoin and for guiding us with patience, rigor, and insight throughout this work. His teaching not only provided the technical foundation but also inspired us to pursue the broader vision behind decentralized systems.

We are also proud to acknowledge the Clube Bitcoin UnB, which we co-founded together with our colleague Luis Schwab. The club became our first platform to organize study sessions, share knowledge, and build a community around Bitcoin on campus. In parallel, we recognize the BitDevs Brasília community, which encouraged us to dive even deeper into the ecosystem through open discussions, critical debates, and practical exchanges.

Finally, we would like to highlight the crucial support of **VinteUm**, whose initiative and resources made both the creation of BitDevs Brasília and the Clube Bitcoin UnB possible. Their contribution laid the foundation for the communities that shaped our journey and ultimately inspired the research presented in this work.

Abstract

Traditional DNS systems depend on hierarchical, centralized trust models that introduce single points of failure, censorship risks, and dependence on central authorities. Although attempts to decentralize DNS have emerged, many struggle with usability and coordination. This project proposes a decentralized DNS service where subdomain ownership is tied to the Bitcoin blockchain and DNS records are published as Nostr events on relays. Ownership is cryptographically proven through Bitcoin transactions, with control over a subdomain linked to possession of a specific UTXO, functioning similarly to a transferable NFT. DNS metadata - including IP addresses, TXT records, and service declarations - is distributed via the Nostr protocol, making records tamper-resistant and censorship-resilient. Authenticity is ensured by correlating Nostr signatures with the associated UTXO holders. This approach, termed DNS-Nostr, offers a user-controlled, censorship-resistant DNS model that blends Bitcoin's immutability with Nostr's real-time communication capabilities, potentially laying the groundwork for decentralized websites, identity systems, and service directories based on Bitcoin-native ownership.

Keywords: DNS, Nostr, Bitcoin, blockchain

Resumo

Os sistemas tradicionais de DNS dependem de modelos de confiança hierárquicos e centralizados, o que introduz pontos únicos de falha, riscos de censura e dependência de autoridades centrais. Embora existam iniciativas para descentralizar o DNS, muitas enfrentam desafios relacionados à usabilidade e coordenação. Este projeto propõe um serviço de DNS descentralizado, onde a posse de subdomínios é ancorada na blockchain do Bitcoin e os registros DNS são publicados como eventos Nostr em relays. A propriedade é comprovada criptograficamente por meio de transações em Bitcoin, sendo que o controle de um subdomínio está vinculado à posse de um UTXO específico, funcionando de forma semelhante a um NFT transferível. Os metadados dos registros DNS — como endereços IP, registros TXT e declarações de serviços — são distribuídos via o protocolo Nostr, tornando-os resistentes à adulteração e à censura. A autenticidade é garantida ao vincular as assinaturas dos eventos Nostr aos detentores dos respectivos UTXOs. Essa abordagem, chamada DNS-Nostr, oferece um modelo de DNS controlado pelo usuário, resistente à censura, que combina a imutabilidade do Bitcoin com as capacidades de comunicação em tempo real do Nostr, podendo servir como base para sites descentralizados, sistemas de identidade e diretórios de serviços com propriedade nativa em Bitcoin.

Palavras-chave: DNS, Nostr, Bitcoin, blockchain

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imagine the internet as a giant city. Every house (website) has a unique address (an IP address) — a long string of numbers that can be hard to remember and even harder to type. This is where the Domain Name System (DNS) comes in. It acts like the city's phone book, translating user-friendly domain names (like `example.com` or `google.com`) into the complex IP addresses that computers understand. [1]

This system is crucial because it allows us to navigate the internet using words and phrases we can remember, rather than memorizing long strings of numbers. Additionally, DNS offers another advantage: IP addresses can change over time. By acting as an alias for computers providing services, DNS allows the link between a domain name and the service's IP address to be updated without affecting how users access it. So, even if the "house" (server) moves locations (changes its IP), users can still find it using the familiar domain name they've always known.

The fundamental design goals of the DNS, as outlined in RFC 1034 [2], emphasize consistency and broad utility. The primary objective was to create a consistent naming space for referring to various resources, ensuring that names were not tied to specific network identifiers, addresses, or routes. This design choice allowed names to remain stable even as the underlying network infrastructure evolved.

Furthermore, the architects of DNS intended it to be broadly useful, not limited to a single application. As stated in RFC 1034, names should be usable to retrieve host addresses, mailbox data, and other, yet-undetermined types of information. This forward-thinking design ensures that any data associated with a name is tagged with a specific type, allowing queries to be precisely targeted to the desired information. This highlights DNS's role not just as an IP address lookup service, but as a flexible system capable of associating diverse types of information with unique names on the internet.

The DNS is a hierarchical system of databases with a top-down structure, starting from the broadest level and progressively narrowing down to specific hosts. This hierarchy

begins with the DNS root zone, managed by the Internet Assigned Numbers Authority (IANA) [3]. Below the root are the top-level domain names (TLDs), which include generic categories like ".com", ".org", and ".net", as well as two-letter country codes defined by ISO 3166 (e.g., ".fr", ".br", ".us") [4]. Each TLD is administered by a designated entity, which further delegates the management of subdomains — effectively forming a multi-level tree. These administrators play a crucial role in managing portions of the naming system, performing a public service on behalf of the Internet community [5].

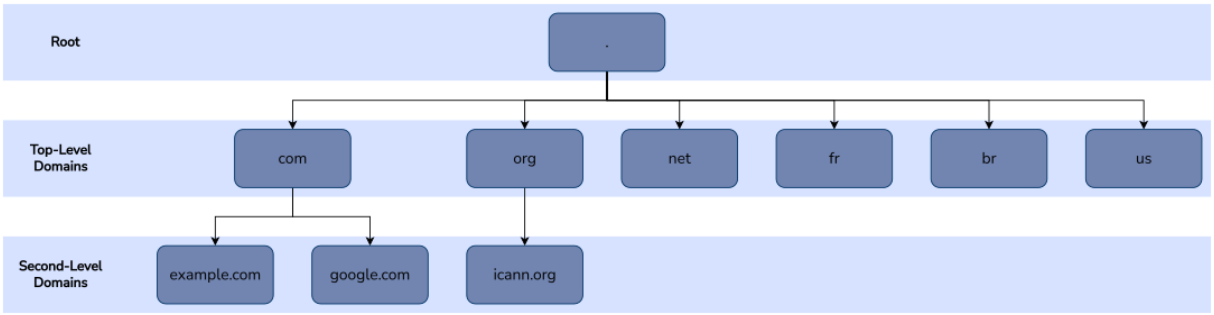

Figure 1.1: Hierarchical structure of the Domain Name System (DNS)

1.1 Problems with DNS

While the Domain Name System (DNS) is foundational to the internet's global functionality, its inherent centralized control points — particularly at the root level — introduce significant political vulnerabilities and risks. The Internet Assigned Numbers Authority (IANA), operating under the Internet Corporation for Assigned Names and Numbers (ICANN), manages the DNS root zone. Although this centralized coordination is broadly deemed indispensable for global internet stability and interoperability, ensuring consistent unique identifiers worldwide [6], it simultaneously creates a crucial point of leverage for political influence.

The primary concern is that IANA's functions represent a single point of failure (SPOF), not necessarily in terms of technical resilience against physical attacks, but critically at the policy and data management layers [7]. This centralized control over the authoritative root zone file — which dictates the internet's top-level structure — exposes the entire system to political pressure, manipulation, and censorship. Such vulnerabilities fundamentally undermine the internet's core principles of openness and neutrality, contributing to the increasing threat of internet fragmentation, often termed the "Splinternet," which jeopardizes global interoperability and communication [8]. The historical role of the U.S. government and ongoing concerns about its influence over ICANN, de-

spite the multi-stakeholder model, further highlight that control in internet governance is a complex sociotechnical and political issue, extending beyond mere technical operations.

A direct manifestation of this political vulnerability is DNS-based blocking and manipulation, a prevalent method of internet censorship. This process interferes with DNS resolution, the initial step in accessing online resources. Authorities can compel Internet Service Providers (ISPs) to configure their DNS servers to prevent access to specific websites — either by failing to return an IP address, redirecting users to government-approved pages, or even presenting deceptive fake versions of the site [9]. ISPs' ability to mandate the use of their designated DNS servers for customer devices provides them with a powerful mechanism for control and censorship.

Real-world incidents vividly illustrate how these centralized control points enable political and state-level interference. For instance, in November 2021, users in Mexico received bogus DNS responses for `whatsapp.net`. This was caused by a BGP (Border Gateway Protocol) route leak, where a network inadvertently or intentionally advertised incorrect routing information. This misdirection diverted DNS queries to a local instance of a root name server located in China. These rerouted queries then passed through middleboxes — network devices that can inspect and alter traffic — which injected fake DNS responses [10]. Similar incidents of BGP route leaks leading to DNS manipulation by state-controlled infrastructure in China were reported in 2010 and 2011, affecting domains like `twitter.com` and `facebook.com` [10]. More recently, in August 2024, Brazil's Supreme Court imposed a ban on X (formerly Twitter), compelling ISPs to block access by configuring their DNS systems to withhold the platform's IP addresses [11] [12].

Beyond direct censorship, the current DNS system also suffers from a significant lack of transparency and accountability. Decisions regarding domain registrations, updates, and resolutions are frequently made within the opaque confines of domain registrars and governing bodies, offering end-users limited insight into these processes. This opacity further underscores the systemic weaknesses in the DNS ecosystem's security, resilience, and trustworthiness.

These examples underscore that despite its technical distribution, the DNS's centralized governance and operational choke points pose significant threats to internet freedom, accessibility, and neutrality, highlighting an urgent need for more resilient and decentralized naming solutions.

1.2 Goal

The primary objective of this work is to propose and demonstrate a decentralized alternative to the traditional Domain Name System (DNS), leveraging the Bitcoin blockchain for

cryptographic ownership guarantees and the Nostr protocol for distributed publication of domain metadata. This system seeks to eliminate central points of control and censorship by introducing a novel architecture wherein subdomain ownership is represented by transferable, verifiable on-chain tokens, and domain records are disseminated through an open, censorship-resistant communication layer.

To achieve this objective, the project defines and implements a custom protocol — DNS-Nostr-Token — that binds a DNS label to a unique Bitcoin UTXO and links it to DNS records published as Nostr events. The scope of the work encompasses the design of the underlying data structures and verification mechanisms, the development of a complete software stack for token creation, validation, and resolution, and the integration of these components into a functioning DNS server capable of responding to real-world queries.

This monograph is structured to provide a comprehensive and technically rigorous account of the proposed system. It begins with a critical analysis of the limitations of current DNS infrastructures, proceeds through an exploration of the relevant cryptographic primitives and protocols, and culminates in the detailed presentation of the implemented architecture. By the conclusion, the work aims to demonstrate not only the feasibility but also the practical viability of a user-controlled, tamper-resistant, and transparent naming system built entirely on open standards and decentralized technologies.

Chapter 2

Redesigning DNS Through Bitcoin and Nostr Protocols

2.1 How does DNS work?

As defined by RFC 1034 [2], the DNS system comprises three major elements: the Domain Name Space and Resource Records, Name Servers, and Resolvers. Together, these components establish and manage a consistent naming space for referring to various resources, allowing for the retrieval of associated information without names being rigidly tied to specific network identifiers or addresses.

The domain name space forms a crucial part of this structure, serving as a hierarchical, tree-structured system where each node or leaf conceptually names a set of information. This space and its associated resource records define the types of data that can be linked to a domain name. When a query is initiated, it targets a specific domain name and requests a particular type of resource information. For instance, the Internet commonly uses domain names to identify hosts, and queries for `address resources` (A records) return the corresponding IP addresses. This flexible design allows DNS to store various kinds of information beyond host addresses, such as mailbox data (MX records) or descriptive text (TXT records), all tagged by their specific type, allowing for precise queries.

Name servers interact with this namespace; they are specialized programs that maintain information about the tree's structure and associated data. While a name server can cache information from any part of the domain tree, each typically holds complete and definitive information for a specific subset of the domain space. A name server is considered authoritative for these particular portions of the namespace. This authoritative information is organized into distinct units called zones, which can be automatically distributed among multiple name servers to provide redundant service and ensure data

availability. Critically, name servers can also provide pointers to other name servers, guiding a resolver toward the authoritative source for information not held locally.

Finally, resolvers act as intermediaries between user programs and name servers. These are typically system routines designed to extract information from name servers in response to client requests. A resolver's primary role is to access at least one name server and directly answer a query from its cached data or pursue it by following referrals to other name servers in the DNS hierarchy. This design means that users interact with the DNS system indirectly through their local resolver, abstracting away the complex process of traversing the name server network.

These three components correspond to different perspectives within the domain system. From a user's point of view, the DNS is a unified, accessible tree where information can be requested from any part via a simple call to a local resolver. For the resolver, the system appears as a collection of potentially many name servers, each holding a piece of the overall domain tree. The name server itself consists of distinct, local information sets called zones, which it must periodically update from the master copies while concurrently processing queries from resolvers.

When you type a domain name into your browser, a series of steps occur leveraging these components:

1. **Request Initiation:** Your computer, acting as a DNS client, sends a query to a DNS resolver, often provided by your internet service provider (ISP) [13].

2. If the local DNS resolver does not have the requested information in its cache, it initiates a recursive query process on behalf of the DNS client. It begins this process by contacting one of the root name servers. The recursive resolver is preconfigured with IANA's "root hints file," which contains the names and IP addresses of the authoritative name servers for the root zone.

3. **TLD Name Servers:** The resolver then queries the appropriate TLD name server, which directs it to the authoritative name servers for the specific domain (e.g., `"example.com"`).

4. **Authoritative Name Server:** Finally, the resolver queries the authoritative name server, which holds the actual DNS records (such as the IP address) for that domain.

5. **Response and Caching:** The answer is returned to your resolver, then to your computer, and is often cached along the way for faster future lookups.

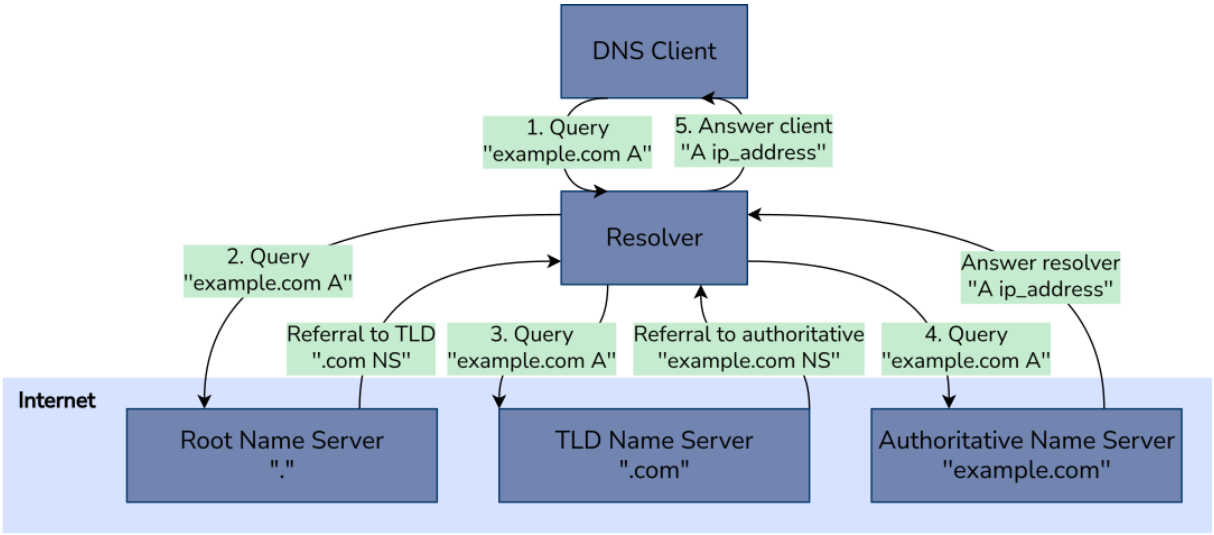

Figure 2.1: Comprehensive overview of the DNS lookup mechanism. The figure illustrates the recursive query initiated by a DNS Client to a Resolver (Step 1), which subsequently performs iterative queries: first to the Root Name Server for TLD referral (Step 2), then to the TLD Name Server for authoritative server referral (Step 3), and finally to the Authoritative Name Server to obtain the 'A' record (IP address) for "example.com" (Step 4). The resolved IP address is then returned to the client (Step 5).

Crucially, for a domain owner to make their website or service accessible, they must publish their DNS records with their chosen DNS service provider (often a registrar or a specialized DNS host). These providers typically store the zone information, including all associated resource records (e.g., A, AAAA, CNAME, MX, TXT records), in master files [14]. As defined in RFC 1035, these master files are text files that contain resource records (RRs) in a standardized text format, serving as the primary means to define a zone's contents for a name server. The owner must make any changes or updates to these records through this provider, which then propagates the updates to the global DNS system.

To illustrate the structure of a typical DNS master file, consider the following example for `example.com`:

```
; Start of Authority (SOA) Record
@       IN SOA (
                    ns.icann.org.       ; Primary name server
                    noc.dns.icann.org.  ; E-mail of the administrator
                    2025011653          ; Serial Number
                    7200                ; Refresh
```

```
                    3600                ; Retry
                    1209600             ; Expire
                    3600                ; Minimum TTL
                )

; Address (A) Records
@       IN A    23.192.228.94   ; example.com maps to this IP address
www     IN A    201.30.129.137  ; www.example.com maps to this IP address
mail    IN A    96.7.128.198    ; mail.example.com maps to this IP address

; IPv6 Address (AAAA) Records
@       IN AAAA     2600:1408:ec00:36::1736:7f24
www     IN AAAA     2600:1408:3A00:21::173e:2e65

; Mail Exchanger (MX) Records
@       IN MX   10 mail

; Canonical Name (CNAME) Record
ftp     IN CNAME    example.com.

; Text (TXT) Records
@       IN TXT      "_k2n1y4vw3qtb4skdx9e7dxt97qrmmq9"
@       IN TXT      "v=spf1 -all"
_dmarc  IN TXT      "v=DMARC1;p=reject;sp=reject;adkim=s;aspf=s"
```

2.2 The Case for Blockchain-Based Naming Systems

To address the inherent vulnerabilities and centralized control points of the traditional Domain Name System (DNS), blockchain-based naming systems (BDNS) offer a paradigm shift. This approach leverages blockchain technology to architect a more resilient, secure, and censorship-resistant internet naming infrastructure, freeing the system from its reliance on central authorities such as ICANN, domain registries, and registrars [15].

The core tenets of blockchain — decentralization, immutability, and cryptographic security — directly counteract the weaknesses of the current DNS. By recording and replicating information across a distributed network of nodes, a BDNS eliminates the single points of failure that make the traditional system susceptible to technical outages and political pressure [16]. This distributed architecture not only enhances resilience against

attacks like Distributed Denial of Service (DDoS), it also makes the data exceptionally difficult to falsify [15]. Consequently, it becomes nearly impossible for any single entity to block, seize, or censor a domain name through administrative or legal coercion — a direct countermeasure to the censorship mechanisms prevalent today [15].

Furthermore, using an immutable public ledger enhances the security and transparency of the entire domain management lifecycle. Every registration and modification is recorded as a cryptographically secured transaction, creating a tamper-proof and auditable ownership history. BDNS allows network participants to independently validate domain ownership and data integrity without relying on a central intermediary [16].

The unique, verifiable, and user-controlled digital ownership model finds a natural and robust technical implementation in Non-Fungible Tokens (NFTs). An NFT is a unique cryptographic token on a blockchain that represents ownership of a specific asset, such as a domain name [17]. The "non-fungible" property is critical: just as every domain name (e.g., `example.com`) is unique, each NFT is distinct and cannot be interchanged on a one-to-one basis, making them ideally suited to represent exclusive domain ownership [18]. This model transforms domain ownership from a temporary lease, contingent on periodic renewals with a registrar, into a persistent digital asset fully controlled by the user. This shift to actual ownership is a foundational principle of the proposed solution [17].

While numerous blockchains can support NFTs, Bitcoin presents a particularly compelling foundation for a global naming system. Launched in 2009 by Satoshi Nakamoto, Bitcoin's defining feature is its unparalleled security, which stems directly from its Proof of Work (PoW) consensus mechanism [19]. The immense and continuously accumulating computational work required to validate transactions makes Bitcoin's blockchain extraordinarily resistant to attack and manipulation, providing the robust foundation of trust necessary for critical infrastructure like DNS [20].

Moreover, Bitcoin's long-standing market dominance and vast network effect signal its stability and long-term viability — crucial attributes for a service intended to be permanent and globally accessible [21]. By leveraging Bitcoin's native security, this project proposes a system where domain ownership is anchored directly on-chain. The following sections will detail how specific protocols built upon Bitcoin can enable the inscription of these domain NFTs, establishing ownership with an unprecedented level of immutability and censorship resistance, thereby forming the core of the DNS-Nostr architecture.

2.3 How Bitcoin Works?

To understand how domain ownership can be anchored in Bitcoin, one must first grasp its fundamental architecture, which differs significantly from modern blockchains. The primary distinction lies in Bitcoin's intentionally limited scripting capabilities. Unlike Ethereum, Bitcoin does not feature Turing-complete smart contracts. A Turing-complete system can, with enough resources, compute anything computable, allowing developers to write code for virtually any arbitrary state transition function [22]. On Ethereum, this enables complex, stateful smart contracts to manage an entire collection of NFTs — dictating their properties, minting rules, and ownership transfers within a sophisticated program. On the other hand, developers must creatively combine simpler base components to create a similar system on Bitcoin.

We will explore this from the top down: first, the overall structure of the blockchain ledger; second, the UTXO set that represents ownership; third, the transactions that update this state; and finally, the Script language that enforces the ownership rules.

2.3.1 The Blockchain Ledger and Proof of Work

The Bitcoin blockchain is a public, ordered, back-linked list of transaction blocks. It is often visualized as a vertical stack, where each new block is placed on top of the previous one, creating a chain that extends back to the first block, known as the Genesis block. This structure gives rise to terms like "height," which refers to a block's position in the chain relative to the Genesis block.

The integrity and immutability of the blockchain are ensured through cryptography. Each block contains a header with critical metadata, including a timestamp, a single cryptographic hash (known as the Merkle root) that serves as a compact summary of all transactions within that block, and, most importantly, a cryptographic hash of the previous block's header. This "previous block hash" acts as a pointer, linking each block to its parent. It is also a key input for calculating the current block's hash. As a result, any alteration to a historical block would change its hash, invalidating the "previous block hash" pointer in the subsequent block and causing a cascading invalidation all the way to the tip of the chain.

The diagram below illustrates this structure, showing how Block 2 contains a hash pointing to Block 1, which in turn points to Block 0, thereby forming a cryptographic chain.

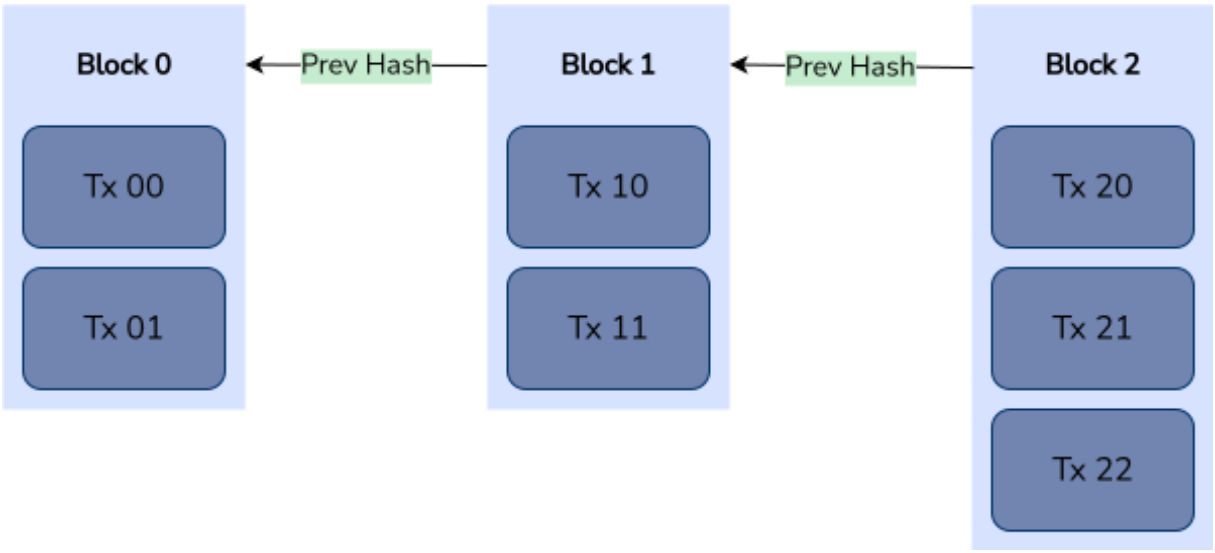

Figure 2.2: Blockchain Structure diagram

This chain reaction makes the blockchain's history extremely difficult to recompute. To successfully change a past block, an attacker must redo the proof-of-work for that block and all subsequent blocks. Bitcoin's consensus mechanism, Proof of Work (PoW), makes this computationally and economically impractical. To produce a new valid block, network participants known as "miners" select a set of unconfirmed transactions and compete to find a valid hash for the new block's header that meets a specific difficulty target. This process involves repeatedly hashing the block header with a different random value (called nonce) until a valid solution is found. The network automatically adjusts this difficulty target approximately every two weeks to ensure that, on average, a new block is added to the chain on average every 10 minutes.

The existence of a long chain of blocks, each secured by intensive computational work, demonstrates that the transaction history is authentic and cannot be altered without redoing an impractically large amount of work. Miners are rewarded to perform this resource-intensive task. The first transaction in every block is a special type called the coinbase transaction, created by the miner. It serves two purposes: issuing newly generated bitcoins (known as the block subsidy) and collecting transaction fees from all other transactions in the block. The block subsidy is a fixed amount that decreases over time, halving approximately every four years (or every 210,000 blocks). This mechanism controls the issuance rate of new bitcoins and ensures that the total supply will never exceed 21 million.

This system creates a market for block space. Each Bitcoin block has a limited capacity, currently capped at 4 million weight units — a more precise metric that replaced the

traditional byte count. Because this space is scarce, users who want their transactions confirmed quickly must include a transaction fee. Miners, acting in their own economic interest, will prioritize transactions offering higher fees to maximize the total reward from the coinbase transaction. This dynamic creates a competitive fee market where users effectively bid for inclusion in the next block. It forms a key part of Bitcoin's long-term economic security model, ensuring the network remains sustainable even after the block subsidy eventually phases out.

2.3.2 State and Transactions

In Bitcoin, the current state of the blockchain is represented by the Unspent Transaction Output (UTXO) set. A UTXO denotes a discrete amount of bitcoin locked to a specific condition — typically a public key derived from a private key — that determines who is authorized to spend it. This set serves as a global, publicly verifiable ledger of all spendable bitcoin in circulation. Possessing the private key necessary to satisfy a UTXO's conditions effectively grants ownership of those funds within the Bitcoin network [23].

To transfer ownership of bitcoins on the network, users create transactions — signed data structures that serve as requests to update the UTXO set. Simply put, a transaction performs two main functions:

1. It consumes one or more existing UTXOs as inputs. These are the funds being spent. Once consumed, these UTXOs are removed from the global UTXO set, as they are no longer considered "unspent."

2. It creates one or more new UTXOs as outputs. These new UTXOs represent the transferred value now locked to new owners and are added to the global UTXO set.

The following diagram illustrates a single transaction. It consumes two existing UTXOs (worth 5 BTC and 3 BTC) for a total input of 8 BTC. It then creates two new UTXOs as outputs: one for 6 BTC and another for 1 BTC. The total output value is 7 BTC. The 1 BTC difference between inputs and outputs is not lost; the miner who includes this transaction in a block can claim it as the transaction fee.

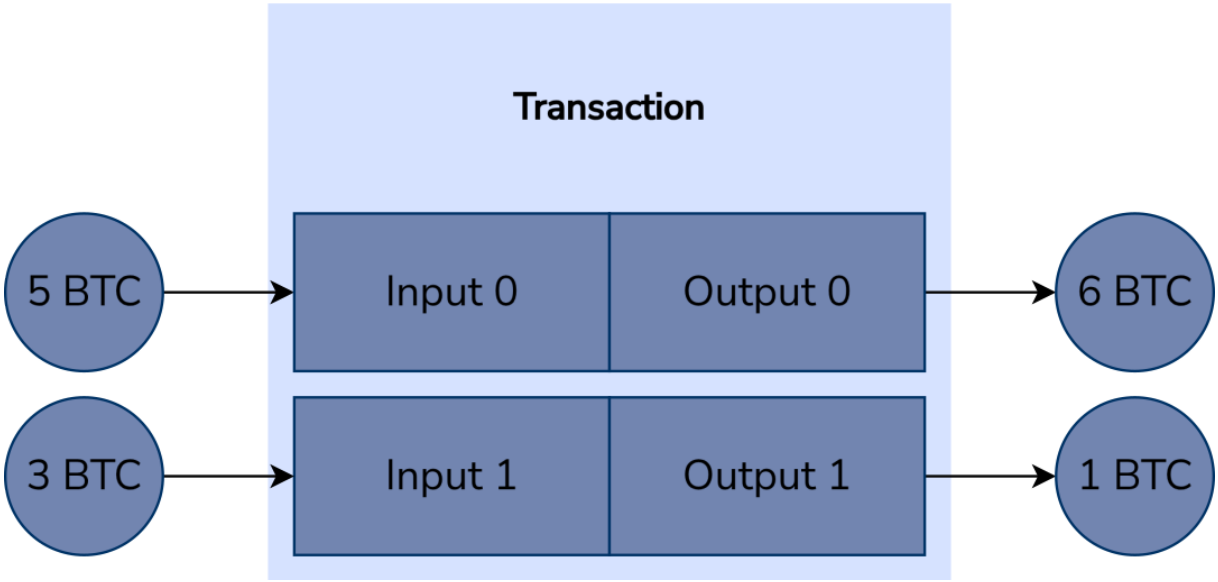

Figure 2.3: Bitcoin transaction diagram

This process forms a continuous "transaction chain," where the outputs of one transaction become potential inputs for future transactions. Each transaction must include a digital signature that proves ownership of the inputs being spent, which any node on the network can independently validate [23]. The diagram below illustrates this chain of dependency: an initial 60 BTC UTXO is spent in Transaction 0, creating two new outputs. The first output (10 BTC) is later consumed as the input for Transaction 1, which creates a 9 BTC output that then becomes the input for Transaction 2. This example demonstrates how value flows from one transaction to the next over time.

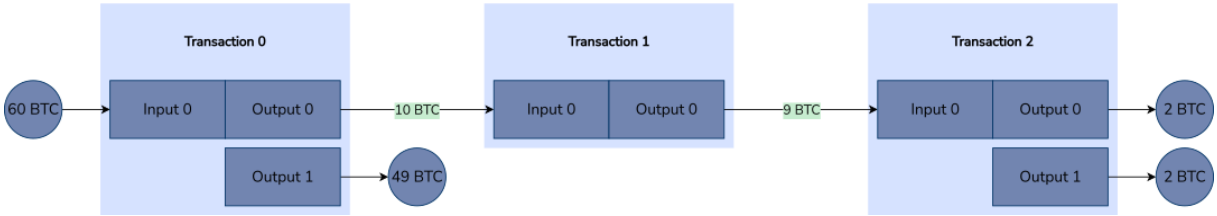

Figure 2.4: Unspent Transaction Output dependency diagram

A transaction is considered final only once it has been included in a valid block and added to the blockchain; this process is known as transaction confirmation. The position of the transaction in the blockchain strictly determines the order in which transactions officially affect the ledger. This order is defined first by the height of the block containing the transaction, and then by the transaction's specific index (its position) within that

block. This system ensures that every node in the network processes the history of state changes in the same sequence.

A transaction fee is a crucial element for getting a transaction confirmed. It is implicitly calculated as the sum of a transaction's inputs minus the sum of its outputs. Miners are incentivized to maximize their profit, so they prioritize transactions offering higher fees relative to their size. This means users must be mindful of their transaction's size, as larger transactions generally require higher total fees to remain competitive. The primary factors affecting transaction size are the number of inputs (which typically contribute the most), the number of outputs, and the complexity of the Script code within them.

As shown, every UTXO has a permanent and unique position on the blockchain that allows it to be precisely identified. This position is defined by the combination of the transaction ID (commonly known as txid) of the transaction that created it and its output index within that transaction's list of outputs (often denoted as vout, starting from 0). This (txid, vout) pair uniquely points to a specific spendable output in the entire blockchain history. Alternatively, once a transaction is confirmed, its UTXOs can also be located by their physical position in the ledger: their block height (commonly known as blockheight), their index within that block (known as blockindex), and their specific output index, forming the triplet (blockheight, blockindex, vout).

2.3.3 Bitcoin Script: Enforcing Ownership Rules

The "lock" on a UTXO is a small program written in Bitcoin's Script language. Each UTXO contains a locking script (technically called the scriptPubKey) that specifies the conditions required to spend it. To spend a UTXO, a new transaction must include a corresponding unlocking script (or scriptSig) in its input that satisfies these conditions.

Bitcoin's scripting language, Script, is intentionally non-Turing-complete. It lacks loops and complex state-handling capabilities, making it more predictable, secure, and less prone to complicated bugs. Consequently, creating an NFT-like system on Bitcoin is more challenging. Deploying a single, all-encompassing contract to manage a domain registry is impossible. Instead, the desired functionality must be constructed by cleverly combining Bitcoin's simpler base components — transactions and UTXOs — to represent the state and ownership of unique digital assets.

Bitcoin Script is a stack-based language. Operations execute sequentially, pushing data onto and popping data off a data structure called a stack. A typical legacy script, known as "Pay-to-Public-Key-Hash" (P2PKH), locks a UTXO to the hash of a public key. The locking script looks like this:

```
OP_DUP
```

```
OP_HASH160
OP_PUSH <PUBLIC_KEY_HASH>
OP_EQUALVERIFY
OP_CHECKSIG
```

To spend this UTXO, the owner must provide an unlocking script that contains their full public key and a valid digital signature for the transaction. A corresponding unlocking script might look like this:

```
OP_PUSH <SIGNATURE>
OP_PUSH <PUBLIC_KEY>
```

To validate the transaction, the Bitcoin network executes the unlocking script followed immediately by the locking script. The transaction is considered valid if, at the end of this execution, the top item on the stack is TRUE — represented by any "truthy" value, that is, any nonzero and non-empty value.

Let's walk through the P2PKH validation process step-by-step:

1. **Initial State:** The process begins with an empty stack.
2. **Execute Unlocking Script:** The items from the unlocking script are pushed onto the stack in order: first the `SIGNATURE`, followed by the `PUBLIC_KEY`.
3. **Execute Locking Script:** The opcodes from the locking script are executed sequentially:
    - **`OP_DUP`:** Duplicates the top item on the stack (the `PUBLIC_KEY`).
    - **`OP_HASH160`:** Pops the top item `PUBLIC_KEY`, hashes it, and pushes the resulting `RESULT_PUBLIC_KEY_HASH` back onto the stack.
    - **`OP_PUSH <PUBLIC_KEY_HASH>`:** Pushes the public key hash specified in the original locking script onto the stack for comparison.
    - **`OP_EQUALVERIFY`:** Pops the top two items (the two public key hashes) and compares them. If they are not equal, the script fails immediately. If they are equal, both are removed, and execution continues. This step verifies that the provided public key matches the one required by the lock.
    - **`OP_CHECKSIG`:** Pops the final two items (`PUBLIC_KEY` and `SIGNATURE`) and verifies that the signature is valid for this transaction using the provided public key. If valid, it pushes TRUE onto the stack.

This entire security model is grounded in the principles of asymmetric cryptography, with the `OP_CHECKSIG` opcode serving as the pivotal mechanism that enforces this foundation. It leverages digital signatures to authenticate ownership. During script execution, `OP_CHECKSIG` uses the public key and the digital signature to mathematically verify that the signature could only have been produced by the holder of the private key corresponding to the `PUBLIC_KEY` used in the transaction. This mechanism allows the spender to prove ownership of the UTXO without ever revealing their private key on the network, securely authorizing the transfer of value. The deterministic and minimal nature of Bitcoin Script ensures that every node in the network can independently validate the transaction and reach the same conclusion about its legitimacy.

Chapter 3

Name-Token Protocol

One of the primary challenges of tokenizing assets on Bitcoin is establishing a stable and persistent identifier. Bitcoin's architecture is built around Unspent Transaction Outputs (UTXOs), which are inherently ephemeral — each UTXO is consumed and destroyed when it is spent. This transient nature makes it difficult to create a digital token that can be reliably tracked, updated, or revoked over time without the use of a specialized protocol.

To address this limitation, this project introduces Name-Token, a novel protocol designed to manage unique names and associate them with arbitrary data directly on the Bitcoin blockchain. The protocol treats each unique name as a distinct digital asset, represented by a single UTXO. Ownership and updates are handled entirely through Bitcoin transactions, allowing the full lifecycle of each Name-Token to be immutably recorded on-chain. This approach provides a stable, verifiable identifier — a critical requirement that is otherwise difficult to fulfill within Bitcoin's native constraints.

A concise set of rules governs the creation, management, and validity of Name-Tokens. These principles ensure each name remains unique and establish a predictable lifecycle for every asset by leveraging the native structure of Bitcoin transactions.

1. **Unique Representation:** Each Name-Token is represented by a single, distinct Unspent Transaction Output (UTXO), which embodies its existence and current state.
2. **Inscribed Metadata:** All relevant data — including the token's label and associated parameters — is permanently inscribed on the blockchain.
3. **State Verification via UTXO Scan:** The set of valid Name-Tokens is derived by scanning the Bitcoin UTXO set for outputs that match the protocol's inscription format.

4. **First-is-Root Uniqueness:** Only one Name-Token can be valid for a given label at any time. The "First-is-Root" rule dictates that the earliest confirmed token for a label is the authoritative one. Any attempt to mint another token with the same label is invalid unless all previous tokens with that label have first been revoked.

5. **Continuity via Same-Index Chain:** A Name-Token's continuity is preserved by maintaining its position within the transaction. To update a token, the current UTXO must be spent as an input, and the new version must appear in the same output index of the resulting transaction.

To fully define the protocol, it is necessary to expand on the technical specifications and practical implications of each core principle. The following subsections describe how these rules are implemented in practice and how they interact to form a coherent, verifiable system for managing names on the Bitcoin blockchain.

3.1 Unique Representation

The core design principle of the Name-Token protocol is to encapsulate the entire existence and state of a token within a single, distinct Unspent Transaction Output (UTXO). This approach guarantees that each name is uniquely identifiable and can be independently updated, transferred, or revoked without ambiguity. Ownership is unambiguous: control of the private key required to spend the UTXO equates to control over the Name-Token. This UTXO acts as the definitive on-chain proof of both ownership and the current state of the name.

3.2 Inscribed Metadata

For a Name-Token to be truly functional, it must carry relevant data. Bitcoin's blockchain has a long history of being utilized beyond simple payments, with early innovators embedding data within transaction outputs to harness the network's unparalleled security for diverse applications like digital notary services.

While various methods have emerged, a common approach for embedding data involves an `OP_RETURN` output. This opcode creates a provably unspendable output designed solely to hold arbitrary data. However, the Name-Token protocol employs a more sophisticated "envelope" technique, leveraging `OP_FALSE`, `OP_IF`, and `OP_ENDIF` opcodes. This specific structure enables the inscription of metadata directly into the locking script of a standard, spendable transaction output. The critical innovation here is that the enclosed opcodes are never executed because the data is encapsulated within an `OP_IF` block preceded

by OP_FALSE. Standard Bitcoin nodes, therefore, treat the script as valid but ignore its encapsulated content, maintaining full compatibility with the network. Conversely, Name-Token-aware software can readily parse and interpret this embedded data.

This ingenious envelope effectively splits the locking script into two conceptual parts:

1. **Name-Token Envelope:** This is the OP_FALSE OP_IF ... OP_ENDIF segment that contains the inscribed metadata.
2. **Non-Dead-Code Locking Script:** This is the standard Bitcoin script that follows the OP_ENDIF and defines the actual spending conditions for the UTXO. For instance, it could be a simple Pay-to-Public-Key-Hash (P2PKH) script requiring a signature from the owner's private key. The presence of the Name-Token inscription does not prevent the UTXO from being spent; it merely adds metadata.

The general structure of the Name-Token inscription within the locking script is as follows:

```
OP_FALSE
OP_IF
    OP_PUSH "name"
    OP_PUSH <LABEL>
    OP_NOP
    OP_PUSH <SECTION_PROTOCOL>
    OP_PUSH <ARGUMENT_0>
    ...
OP_ENDIF
# [Non-dead-code locking script follows here, e.g., P2PKH script]
```

Each line within the OP_FALSE OP_IF ... OP_ENDIF block serves a specific purpose, designed for extensibility and multi-protocol support. The initial OP_PUSH "name" acts as a magic number or protocol identifier, signaling that the subsequent data should be interpreted according to the Name-Token specification. The <LABEL> defines the specific name of the Name-Token, providing the actual naming component. The OP_NOP is a crucial separator, allowing the protocol to delineate distinct metadata sections. This separation is particularly powerful for accommodating multiple protocols; the <SECTION_PROTOCOL> identifier, followed by its <ARGUMENT_0> and potentially more arguments, specifies how different applications building on top of Name-Tokens should interpret the following data. This modular design ensures that the same underlying Name-Token UTXO can carry different types of metadata, relevant to various decentralized applications, without breaking compatibility.

For a Name-Token UTXO with this inscription to be spent, an unlocking script must be provided that satisfies the non-dead-code locking script portion following the `OP_ENDIF`.

For example, if the non-dead-code locking script is a standard P2PKH script (requiring a public key and a signature), an input unlocking script for a transaction spending this Name-Token UTXO would look like this:

```
OP_PUSH <SIGNATURE>
OP_PUSH <PUBLIC_KEY>
```

3.3 State Verification via UTXO Scan

The complete, current state of all Name-Tokens can be derived directly from Bitcoin's live UTXO set. This is a powerful feature, as it means the protocol's state is embedded within Bitcoin's own state, simplifying the process of locating Name-Tokens on the blockchain. An application can retrieve the data associated with a name simply by finding its corresponding UTXO.

It is critical to note that not all UTXOs are Name-Tokens. To identify a Name-Token, a client or indexer must scan the UTXO set and filter for outputs whose locking scripts contain valid inscriptions that conform to the protocol's `OP_FALSE OP_IF ...OP_ENDIF` structure. The protocol ignores any UTXO that does not match this format.

3.4 First-is-Root Uniqueness

This principle dictates that while multiple Name-Tokens may exist for a given label, only one can be considered the valid, authoritative token at any given time. The valid token is unequivocally the one minted at the earliest chronological point — that is, the one confirmed at the lowest block height. For instance, if Alice mints a Name-Token for "blog" in block 800,000, and Bob subsequently attempts to mint a Name-Token for "blog" in block 800,100, only Alice's token is recognized as valid. Although Bob's token is recorded on the blockchain, it is considered an invalid mint attempt under this rule.

Crucially, the "First-is-Root" rule introduces a notion of age or seniority to Name-Tokens: older, earlier-minted tokens take precedence over newer ones. Bob's otherwise invalid "blog" token could only become valid if Alice's token — and any other "blog" tokens minted between blocks 800,000 and 800,100 — were first explicitly revoked or spent. This rule ensures a clear, unambiguous, and unforgeable ownership hierarchy, leveraging the Bitcoin blockchain's immutable timestamping and ordered structure to establish global uniqueness for each Name-Token label.

3.5 Continuity via Same-Index Chain

The Name-Token protocol ensures each token's stable and auditable history by enforcing the "same-index chain" (or "input/output chain") rule. This mechanism establishes a robust positional link between an input and its corresponding output within a transaction, enabling the token's lifecycle to be transparently tracked across the Bitcoin blockchain.

This principle closely aligns with the fundamental concept of the Bitcoin transaction chain, wherein the output of one transaction becomes the input of a subsequent one. The Name-Token protocol extends this model by introducing a crucial positional constraint. To update a token's inscribed data or transfer its ownership, the current Name-Token UTXO must be used as an input in a new transaction. Critically, a new UTXO containing the updated inscription must be created in an output at the same index as the consumed input. For example, if a token's UTXO is consumed as the second input (at `input[1]`), the updated token must be inscribed in the second output (at `output[1]`) of the new transaction. This "same-index" rule serves as a stable identifier, consistently linking a token's state across an indefinite number of transactions. A transfer of ownership is simply a specific type of update in which the new output is locked to a new owner's public key.

Given that the principle of **State Verification via UTXO Scan** dictates that only the latest valid Name-Token inscription present in the UTXO set is considered active, a token is effectively "revoked" if its UTXO is spent and a corresponding valid Name-Token inscription with the same label is not created at the same output index. This mechanism ensures that a token's active status is always either explicitly maintained or explicitly terminated.

The diagram below illustrates several scenarios of Name-Token lifecycle management within a single transaction, emphasizing the application of the "same-index chain" principle:

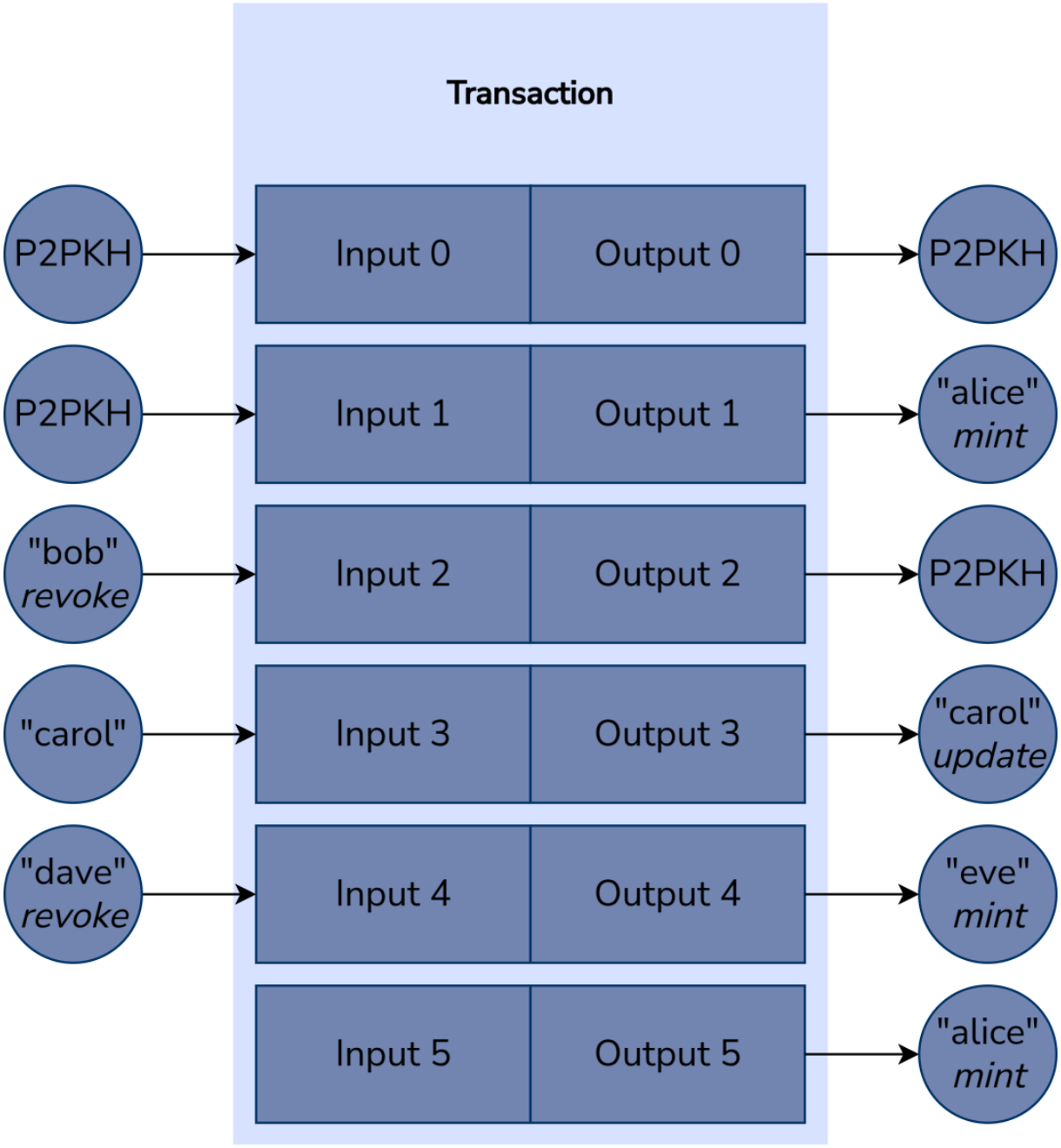

Figure 3.1: Same-index chain principle diagram

The central transaction box in the diagram represents a single Bitcoin transaction containing multiple inputs and outputs. Each link conceptually pairs an input with the output at the same index, visually illustrating the "same-index chain" rule. Breaking down the example:

- **Link 0 (Standard Bitcoin Transfer):** A standard Pay-to-Public-Key-Hash (P2PKH) UTXO is consumed as Input 0, and a new P2PKH UTXO is created as Output 0.

This represents a typical Bitcoin transfer with no Name-Token involved at this index.

- **Link 1 (Minting "alice"):** A standard P2PKH UTXO is consumed as Input 1. Output 1 then creates a new Name-Token, "alice," indicating its initial minting. This is a fresh inscription at a specific index.
- **Link 2 (Revoking "bob"):** The Name-Token "bob" is consumed as Input 2. However, Output 2 creates a P2PKH UTXO without a corresponding Name-Token inscription at this index. This signifies that "bob" has been explicitly revoked or ended.
- **Link 3 (Updating "carol"):** The existing Name-Token "carol" is consumed as Input 3. Critically, Output 3 then inscribes a Name-Token at the same index with the same label but with updated metadata. This demonstrates how a token's data or ownership can be modified while maintaining its identity through the index.
- **Link 4 (Reassignment/New Mint):** An existing Name-Token "dave" is consumed as Input 4. Interestingly, Output 4 creates a new Name-Token, "eve." While "dave" is effectively revoked at this index, a completely new Name-Token, "eve," is minted in its place. This illustrates how a spent token's index can be reused for a new token.
- **Link 5 (Another Mint):** This shows an output creating a new Name-Token "alice," without a corresponding input from a Name-Token. This would represent a direct minting of a new "alice" token. This token, though, is provably invalid since there is already a minted "alice" token in previous links of the same transaction.

3.6 How does Nostr work?

Given the Name-Tokens protocol, a naive approach to associating domain names with DNS would be to place the binary representation of the records in the Name-Token inscription. RFC 1035 [14] itself defines the binary format for DNS records, which domain owners could use to create a Name-Token inscription like this (using a human-readable format for clarity):

```
OP_FALSE
OP_IF
    OP_PUSH "name"
    OP_PUSH "blog"
    OP_NOP
```

```
    OP_PUSH "dns"
    OP_PUSH "@      IN A    023.192.228.084"
    OP_PUSH "www    IN A    201.030.129.137"
    OP_PUSH "main   IN A    096.007.128.198"
OP_ENDIF
# [Non-dead-code locking script follows here, e.g., P2PKH script]
```

However, this approach has several limitations. First, using the Bitcoin blockchain for extensive data storage unrelated to payments is contentious, as it increases the storage burden on all full nodes. Second, adding such a large amount of data to a transaction significantly increases its size, thereby raising the transaction fee required to mint and update the Name-Token.

Nostr provides an elegant off-chain solution to this problem. Standing for "Notes and Other Stuff Transmitted by Relays," Nostr is a simple, open protocol designed to create a global, censorship-resistant social network. It is not a peer-to-peer system and does not have its own blockchain. Instead, it relies on a straightforward client-relay architecture and employs the same public-key cryptography model used by Bitcoin [24].

The diagram below illustrates the fundamental workflow of the Nostr protocol:

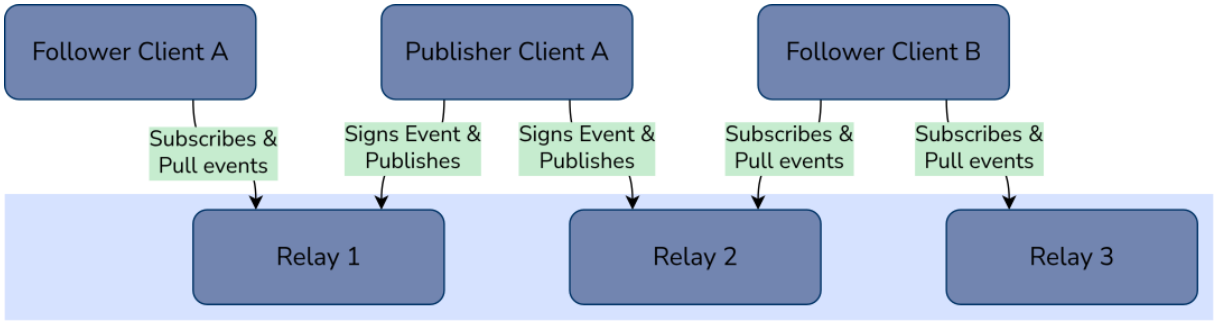

Figure 3.2: Nostr protocol fundamental workflow

In Nostr, the core components consist of Clients and Relays:

- **Clients:** These are the applications with which users interact. To publish content, a user's client creates an "event" (e.g., a text note or, in our case, DNS records) and signs it with the user's private key. The corresponding public key serves as the user's identity.
- **Relays:** These are simple servers that receive events from clients, store them, and forward them to other subscribed clients. Relays do not communicate with each other.

As shown in the diagram, a user on Client A can publish their signed event to any relays they choose, such as Relay 1 and Relay 2. Other users, like the one on Client B, can connect to those same relays to retrieve the event. This architecture provides strong censorship resistance: if a single relay blocks a user, that user can simply publish their event to another relay. The ability of relays to charge fees creates a financial incentive for operators, ensuring there will always be someone willing to host a relay, thereby further strengthening the network's resilience against censorship. Since every event is cryptographically signed, its authenticity can be verified by anyone, making the data tamper-proof regardless of which relay it is retrieved from.

3.7 DNS-Nostr-Token Protocol

The DNS-Nostr-Token Protocol is a specific application built upon the general Name-Token framework. It defines a dedicated section within the Name-Token inscription to link a domain name directly to DNS records stored on the Nostr network. This approach separates the on-chain proof of ownership from the off-chain, easily updatable DNS data.

The core mechanism is straightforward: the protocol maps a registered label (the domain name) to a specific Nostr public key. This key is then used to query Nostr relays for the latest text note published by its owner. The content of this Nostr note is expected to be a standard DNS master file, containing all relevant DNS records (A, AAAA, MX, TXT, etc.) for that domain.

A UTXO inscribed according to this specification is referred to as a DNS-Nostr-Token. The inscription uses "dns-nostr" for the `<SECTION_PROTOCOL>` field, and the first argument, `<ARGUMENT_0>`, is the associated Nostr public key.

```
OP_FALSE
OP_IF
    OP_PUSH "name"
    OP_PUSH <LABEL>
    OP_NOP
    OP_PUSH "dns-nostr"
    OP_PUSH <NOSTR_PUBLIC_KEY>
OP_ENDIF
# [Non-dead-code locking script follows here, e.g., P2PKH script]
```

For a Name-Token to be considered a valid DNS-Nostr-Token, it must adhere to specific rules that ensure compatibility with the global DNS system:

1. **Label Compliance:** The label must comply with DNS naming conventions as defined in RFC-1034 [2]. Specifically, it must begin with a letter, must not end with a hyphen, and may contain only lowercase letters (a–z), digits (0–9), and hyphens. It must be no more than 63 characters long. The lowercase requirement ensures consistency, as DNS is case-insensitive.
2. **Valid Public Key:** The `<NOSTR_PUBLIC_KEY>` field must contain a correctly formatted Nostr public key, as specified in NIP-01 [25].

A token may qualify as a valid Name-Token under the general protocol, but it will be considered an invalid DNS-Nostr-Token if it fails to meet either of the two criteria above.

It is also important to note that DNS records retrieved from Nostr may vary across different relays. Under this protocol, the responsibility for publishing and maintaining accurate DNS records lies entirely with the owner of the DNS-Nostr-Token. The owner must ensure that a valid DNS master file is published to the relays and kept consistently up to date.

Chapter 4

System Architecture and Implementation

This section describes the practical implementation of the DNS-Nostr Protocol. The system architecture is built around three primary, interoperable components: the DNS-Nostr Wallet, the Name-Token Repository, and the DNS-Nostr Server.

Together, these components leverage the security of the Bitcoin network for ownership and the flexibility of the Nostr network for data publication. The Wallet serves as the user-facing tool, the Repository functions as the blockchain indexer, and the Server provides the public-facing resolution service.

The DNS-Nostr Wallet serves as the user's interface for minting Name-Tokens on the Bitcoin network and for publishing the corresponding DNS records to Nostr relays. The Name-Token Repository is a backend service that continuously scans the Bitcoin blockchain to identify and validate all Name-Tokens, maintaining a clean, queryable database of valid entries. The DNS-Nostr Server acts as the resolver interface: it receives queries from standard DNS clients, uses the Repository to locate the correct Nostr public key for the requested name, and retrieves the most recent DNS records from Nostr to construct and return a valid DNS response.

The open-source implementation of the system is available in the project's GitHub repository: `https://github.com/luigiminardim/dns-nostr-tokens`.

4.1 DNS-Nostr Wallet

The DNS-Nostr Wallet is a client-side application that enables users to create and manage decentralized domain labels. Its core function extends beyond standard Bitcoin wallet capabilities, focusing on handling the full lifecycle of Name-Tokens and their associated DNS-Nostr inscriptions.

The wallet's features offer a comprehensive user experience, supporting both standard Bitcoin operations and the specialized capabilities required by the Name-Token protocol:

- **Standard Bitcoin Functions:** Supports basic operations such as sending and receiving bitcoins.
- **Name-Token Management:** Manages the complete lifecycle of DNS-Nostr Tokens:
  - **Create:** Mints new tokens on the Bitcoin blockchain to represent subdomain labels.
  - **Transfer:** Transfers ownership of Name-Tokens to other Bitcoin addresses.
  - **Revoke:** Permanently removes a Name-Token from the active set by breaking its same-index chain.
  - **Nostr Integration:** Publishes user-provided DNS records as signed text notes to Nostr relays, using the public key embedded in the token's inscription.

All token operations involve crafting a Bitcoin transaction in which the output's locking script (scriptPubKey) is a standard Pay-to-Public-Key-Hash (P2PKH) script, prefixed by the DNS-Nostr Token inscription envelope. This structure ensures that the output remains spendable under Bitcoin's consensus rules while embedding the required protocol data directly into the transaction.

```
# Inscription Envelope
OP_FALSE
OP_IF
    OP_PUSH "name"
    OP_PUSH <LABEL>
    OP_NOP
    OP_PUSH "dns-nostr"
    OP_PUSH <NOSTR_PUBLIC_KEY>
OP_ENDIF
# Standard P2PKH Locking Script
OP_DUP
OP_HASH160
OP_PUSH <PUBLIC_KEY_HASH>
OP_EQUALVERIFY
OP_CHECKSIG
```

4.1.1 Key Management

To enhance both usability and security, the wallet adopts hierarchical deterministic (HD) key derivation for its key management. This approach streamlines the backup process by requiring the user to secure only a single master key (denoted as `m`), which is typically derived from a mnemonic seed phrase. From this master key, the wallet can deterministically generate the full set of required cryptographic keys, thereby eliminating the need to back up individual keys and reducing the risk of key loss.

A segregated key management scheme is employed to minimize the risk of inadvertently spending valuable Name-Token UTXOs and to isolate Nostr-related activities from other wallet operations. This separation is enforced through the use of distinct hierarchical deterministic (HD) derivation paths for each key type, ensuring clear operational boundaries and enhancing overall wallet safety.

- **Standard UTXOs:** For general-purpose Bitcoin transactions, including fee payments and transfers, the wallet adheres to the BIP 84 standard [26], which defines key derivation for native SegWit (bech32) addresses. This ensures broad compatibility with modern Bitcoin infrastructure and promotes efficient transaction processing.
- **Name-Token UTXOs:** To manage Name-Tokens, the wallet employs a customized derivation path based on BIP 44 [27], introducing a critical modification at the "coin type" or "chain" level. Instead of a conventional numeric identifier, it uses a constant derived from the ASCII encoding of the string `"name"` (0x6E616D65 in hexadecimal). This deliberate deviation renders Name-Token UTXOs effectively invisible to standard wallets, thereby offering a built-in safeguard against accidental spending.
- **Nostr Keys:** For managing Nostr key pairs used to sign and publish DNS records, the wallet follows a custom derivation scheme inspired by NIP-06 [28]. To ensure strict segregation from other Nostr-related operations, the same ASCII-derived constant (0x6E616D65) is again used at the chain level. To guarantee that each Name-Token corresponds to a unique Nostr key, the final index of the derivation path is determined by a truncated (31-bit) double SHA-256 hash of the Name-Token label. This approach preserves compatibility with non-hardened derivations, maintaining flexibility while ensuring determinism and uniqueness.

The table below summarizes the distinct hierarchical deterministic key derivation paths employed by the wallet for each functional purpose, illustrating the systematic segregation of key types within the wallet architecture:

Description	Coin	Account	Chain	Address	Path
Standard UTXO	Bitcoin	First	External	First	`m/84'/0'/0'/0/0`
Standard UTXO	Bitcoin	First	External	Second	`m/84'/0'/0'/0/1`
Standard UTXO	Bitcoin	First	Change	First	`m/84'/0'/0'/1/0`
Name-Token UTXO	Bitcoin	First	Name-Token	First	`m/44'/0'/0'/0x6E616D65/0`
Nostr Key	Nostr	First	Name-Token	hash31("alice")	`m/44'/0'/0'/0x6E616D65/0xBD306425`

Table 4.1: Wallet Derivation Paths

4.1.2 Token Creation

To mint a new DNS-Nostr-Token, the wallet constructs and broadcasts a Bitcoin transaction that links a chosen label to a Nostr public key. It selects one or more standard UTXOs with a total value sufficient to cover both the transaction fee and the dust limit for the new token output. The transaction's first output is a P2PKH Name-Token containing the complete DNS-Nostr token inscription. A second output is typically included to return the remaining bitcoins to the user's wallet as change.

After building the transaction, the wallet uses the user's selected local file — containing their DNS records in standard master file format — signs it with the corresponding Nostr private key, and publishes it as a text note to the user-configured Nostr relays.

4.1.3 Token Revocation

Revoking a Name-Token is a straightforward process that involves spending it without creating a replacement. The wallet consumes the Name-Token UTXO and may include additional UTXOs to cover the transaction fee. The resulting transaction contains no special outputs — the entire value of the inputs (minus the fee) is sent to a standard change address in the user's wallet. Since no new Name-Token is created at the same output index, the token is effectively burned and removed from the active set.

4.2 Name-Token Repository

The Name-Token Repository is a crucial backend component that provides a queryable interface to the current state of all valid Name-Tokens. To accomplish this, it functions as

a specialized indexer, continuously monitoring the Bitcoin blockchain, validating transactions against protocol rules, and maintaining a local database of results.

The repository persists its internal state in a database that maps each valid label to its current on-chain data, including its UTXO identifier (outpoint) and inscription content. This design allows the service to resume operation after a restart without needing to re-scan the entire blockchain from the beginning.

4.2.1 Indexing Process and Fork Resistance

The integrity of the repository's database is paramount. Since the Bitcoin blockchain operates under probabilistic finality, the indexing process must be resilient to chain reorganizations (forks). To mitigate the risk of processing data from transient, non-canonical chains, a block is only indexed after it reaches a minimum confirmation depth, denoted as `MIN_NUMBER_CONFIRMATIONS` (typically set to 6). By enforcing this confirmation delay, the system ensures it operates solely on blocks that are part of the stable, canonical history of the ledger.

At the core of the repository lies a state transition indexing algorithm that executes periodically (e.g., every 10 minutes) to update its internal state in accordance with newly confirmed blocks. This background process operates on a fixed schedule, systematically processing incoming blocks and reflecting their effects within the local database. The procedure follows these steps:

1. **Initialization:** Retrieve the next blockchain position to be processed, identified by block height, block index, and input/output index.
2. **Verify Confirmation Depth:** Check whether the block at the specified height has reached the minimum number of confirmations. If not, terminate the current execution cycle and resume once the block is sufficiently confirmed.
3. **Fetch Block Data:** Retrieve the block data at the specified block height.
4. **Process State Transitions:** For each input/output link in each transaction within the block, apply the Name-Token protocol rules to determine if a token is being minted, updated, or revoked. Then, commit all resulting changes in a single atomic database transaction, including:
    - Inserting a new record for any minted token.
    - Updating the record for any modified token.
    - Deleting the record for any revoked token.
    - Updating the scan position pointer to the next input/output link.

5. **Checkpoint Block Completion:** Save the next block height as the new starting position for subsequent processing.
6. **Loop:** Repeat the entire process beginning from Step 1 to continue processing the next block.

The atomicity of the database operations in Step 4 is crucial for maintaining data consistency. The repository ensures a consistent state by committing all state changes — including inserts, updates, and deletions — alongside advancing the scan position pointer within a single, indivisible transaction. This approach prevents failure scenarios, such as a crash occurring after data is updated but before the scan position is saved, which could otherwise lead to incorrect reprocessing or data loss upon restart.

The block checkpoint in Step 5 serves as an important performance optimization. While the atomic updates in Step 4 ensure fine-grained recovery, saving a checkpoint after processing an entire block allows the indexer to resume from the beginning of the next block, rather than reprocessing transactions within a partially completed block. This significantly reduces recovery time following a shutdown.

4.2.2 Database Schema

SQLite was selected as the underlying database for this implementation. The schema is designed to store both the scanner's state and the indexed collection of Name-Tokens, organizing data to enable efficient lookups and updates.

- **Table State:**
  - `block_height`: The height of the block currently being scanned.
  - `tx_index`: The index of the transaction within the block.
  - `vout_index`: The index of the output within the transaction.
- **Name-Token Set:**
  - `label`: The unique name of the token, stored as a byte string.
  - `mint_blockheight`: The block height where the token was first created.
  - `mint_tx_index`: The index of the transaction where the token was first created within that block.
  - `mint_vout`: The output index (vout) where the token was first created.
  - `last_txid`: The transaction ID (txid) of the token's current UTXO, stored as a byte string.

  - `last_vout`: The output index (vout) of the token's current UTXO.
  - `inscription_content`: JSON string containing the raw data from the token's inscription, which may include protocol-specific sections and arguments.

To ensure fast queries, the database employs two critical indexes:

- An index on `(label, mint_blockheight, mint_tx_index, mint_vout)`: This enables quick lookups to find the oldest — and thus valid — token for a given label, enforcing the "First-is-Root" rule.
- An index on `(last_txid, last_vout)`: This is essential for the scanning algorithm to rapidly identify which Name-Token is being spent when processing transaction inputs.

4.3 DNS-Nostr Server

The DNS-Nostr Server functions as an intermediary component that integrates the conventional DNS infrastructure with the decentralized ownership records maintained on the Bitcoin blockchain.

In an ideal scenario, every computer would operate its own Bitcoin node alongside a background instance of the Name-Token Repository, enabling each machine to resolve names independently in a truly peer-to-peer manner. However, to ensure widespread adoption and maintain compatibility, the system is designed to integrate seamlessly with the existing DNS hierarchy. This integration is facilitated by permitting independent domain owners to deploy the DNS-Nostr Server as an authoritative server for their specific subdomains.

This model adheres to a principle akin to Nostr's decentralization: just as anyone may operate a Nostr relay, any domain owner is empowered to run a DNS-Nostr Server. This approach fosters a resilient, distributed network of independent resolvers, eliminating reliance on a single central authority.

An operator of an existing domain, such as `dns.app`, can delegate a subdomain — for example, `nostr.dns.app` — to their DNS-Nostr Server by adding two records to the primary domain's master file. Assuming the DNS-Nostr Server is hosted at IP address `164.41.102.70`, the configuration would be as follows:

```
nostr               IN NS   dns-nostr-server.dns.app.
dns-nostr-server    IN A    164.41.102.70
```

The NS (Name Server) record delegates all resolution requests for `*.nostr.dns.app` to the server `dns-nostr-server.dns.app`. The A record maps this server name to its

corresponding IP address. Consequently, any DNS query for a subdomain ending with `.nostr.dns.app` will be directed to the DNS-Nostr Server for resolution.

The DNS-Nostr Server functions as an authoritative DNS server for its delegated subdomains. Upon receiving a query, it initiates a deterministic resolution process to identify and return the appropriate resource records.

For example, consider a query for the A record `blog.alice.nostr.dns.app`. The server executes the following steps:

1. **Isolate the Label:** The query is parsed to identify the user-registered label — in this case, `alice`.
2. **Query the Repository:** The server consults its local Name-Token Repository to locate the valid Name-Token associated with the label `alice`.
3. **Extract the Public Key:** If a valid token is found, the corresponding Nostr public key is extracted from the token's inscription data.
4. **Fetch from Nostr:** The server connects to a set of Nostr relays and retrieves the most recent text note published by the associated public key.
5. **Parse Records:** The retrieved note, expected to be in standard DNS master file format, is parsed to extract the requested resource record (e.g., the A record for the name `blog`).
6. **Respond to Client:** The extracted record is formatted into a valid DNS response and returned to the client that issued the original query.

The server is capable of resolving all standard DNS record types, including `A`, `AAAA`, `CNAME`, `NS`, `MX`, and `TXT`. The complete resolution workflow is illustrated in the diagram below.

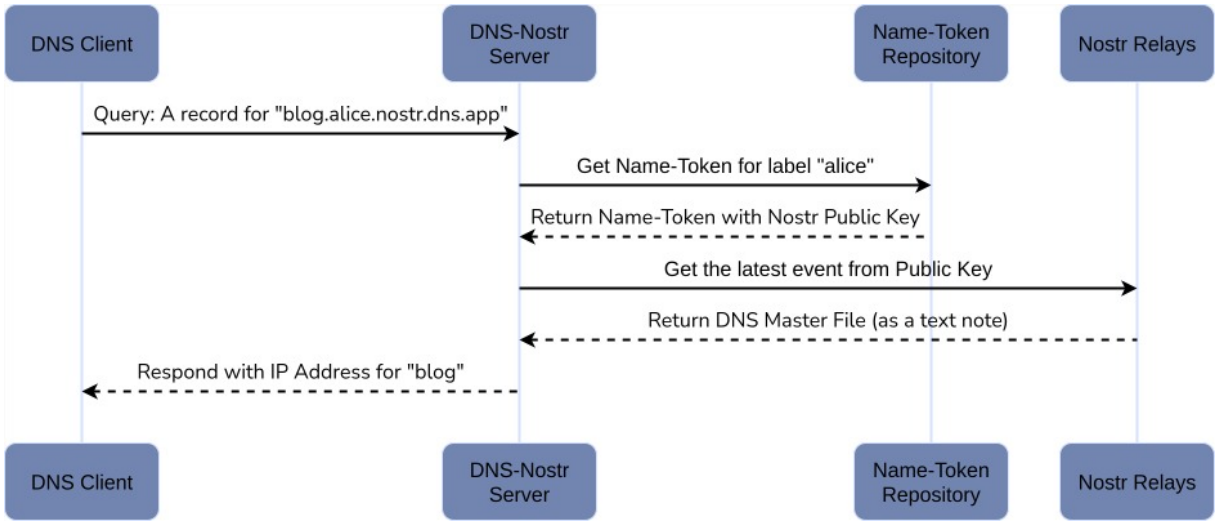

Figure 4.1: Query resolution diagram

To better illustrate the differences between the conventional DNS infrastructure and the proposed DNS-Nostr Server, Table 4.2 provides a side-by-side comparison of their main characteristics.

Aspect	Traditional DNS	DNS-Nostr Server
Governance	Root zone managed by ICANN/IANA, with registries and registrars controlling domain ownership.	Ownership anchored in Bitcoin UTXOs; no central authority or registrar.
Trust Model	Relies on hierarchical authorities and delegated registries.	Relies on cryptographic proof of UTXO ownership and signed Nostr events.
Censorship Risk	High — governments can pressure registries/ISPs to block or seize domains.	Low — resistant to seizure, since control is purely key-based and records are relayed openly.
Update Speed	Fast — registrar/registry changes propagate within minutes/hours.	Slow for ownership changes (Bitcoin confirmations ∼10 min/block); Nostr record updates are near-instant but depend on relay availability.
Cost	Recurring fees to registrars; administrative overhead.	On-chain transaction fees for mints/transfers; relay usage often free or low-cost.
Scalability	Globally scalable, supported by decades of infrastructure.	Limited by Bitcoin throughput and the need for custom indexers; performance depends on relay distribution.
Resolution	Based on hierarchical query to root, TLD, and authoritative servers.	Direct mapping from Name-Token to Nostr public key, then record retrieval from relays.
Transparency	Decisions on domain allocation often opaque to end users.	Fully auditable — all ownership proofs and updates are publicly verifiable on-chain.

Table 4.2: Traditional DNS vs. DNS-Nostr Server

4.4 System Workflow Diagram

The diagram below presents the complete system architecture, highlighting the interaction between core components responsible for enabling both name creation and resolution. The

overall process is divided into two distinct workflows: token creation and data publication, and DNS resolution.

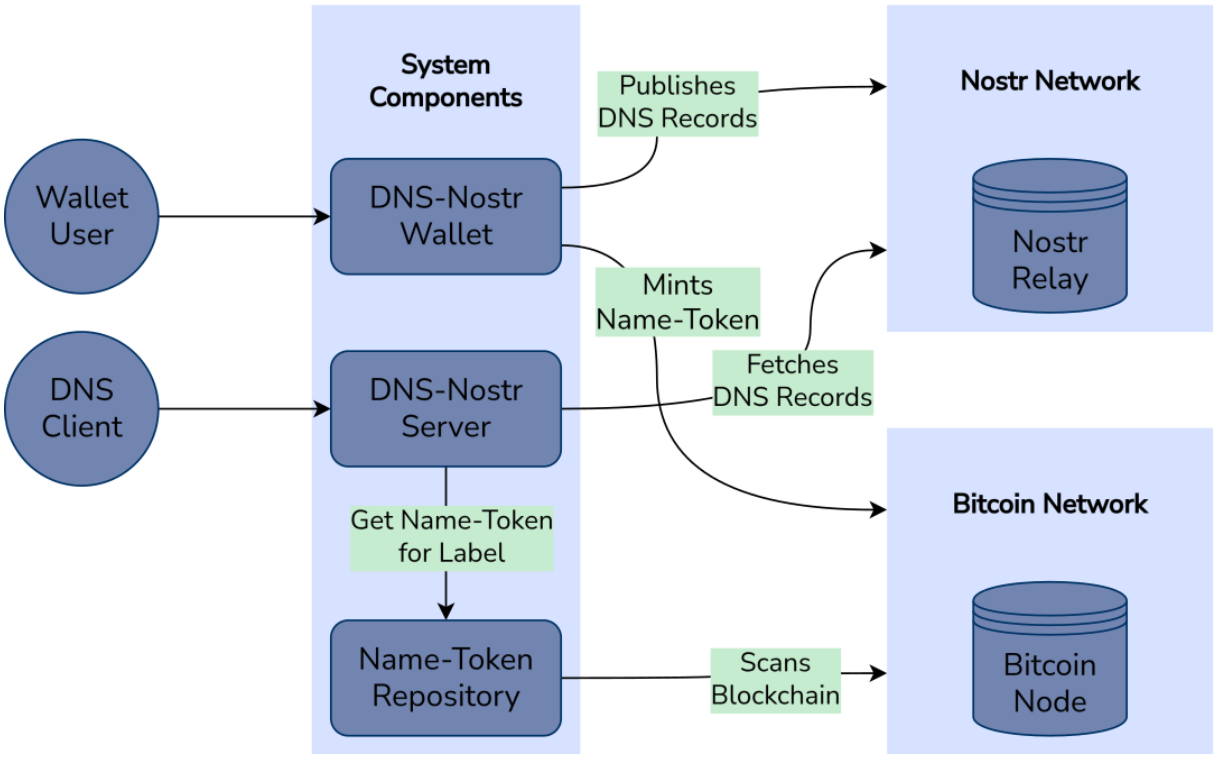

Figure 4.2: System workflow diagram

4.4.1 Token Creation and Data Publication

This workflow outlines the process by which a user registers a new name and binds it to corresponding DNS data.

1. **Minting the Name-Token:** The user interacts with the DNS-Nostr Wallet to construct and broadcast a Bitcoin transaction. This transaction includes a specially formatted inscription that mints a new Name-Token on the Bitcoin blockchain, thereby permanently associating a chosen label with a Nostr public key.
2. **Publishing DNS Records:** Using the same wallet, the user prepares their DNS records in the standard master file format, signs them with their Nostr private key, and publishes the resulting event to the Nostr network.
3. **Indexing the Name-Token:** Once the minting transaction receives sufficient confirmations, the Name-Token Repository — operating independently — detects the new token, validates it against the protocol rules, and persists the label and corresponding Nostr public key in its local database.

4.4.2 DNS Resolution

This workflow outlines the process through which the system resolves a DNS query initiated by an end-user.

1. **DNS Query:** A standard DNS client issues a query for a specific subdomain (e.g., `blog.alice.nostr.dns.app`) directed to the DNS-Nostr Server.
2. **Name Lookup:** Upon receiving the query, the server extracts the registered label (in this case, `alice`) and queries its internal Name-Token Repository to obtain the corresponding Nostr public key.
3. **Data Retrieval:** Using the resolved public key, the server connects to one or more Nostr relays to retrieve the most recent event containing DNS records published by that key.
4. **DNS Response:** The server parses the retrieved DNS master file to locate the requested record (e.g., the A record for `blog`), formats it into a standard DNS response, and returns it to the querying client.

4.5 Implementation and Proof of Concept

To validate the feasibility of the DNS-Nostr-Token protocol, we developed a working prototype composed of the three components previously described: the DNS-Nostr Wallet, the Name-Token Repository, and the DNS-Nostr Server. This section presents the implementation details and experimental validation through a series of tests and screenshots of the system in operation.

4.5.1 Wallet Operations

The DNS-Nostr Wallet was implemented as a command-line interface (CLI) that allows users to manage both standard Bitcoin funds and Name-Tokens. The wallet supports operations such as checking balance, creating new DNS-Nostr-Tokens, updating metadata, transferring ownership, and revoking tokens.

Figure 4.3 shows the wallet balance operation, confirming that the prototype can interact with Bitcoin test funds. Subsequently, Figure 4.4 demonstrates the creation of a DNS-Nostr-Token named `luigi`, where the transaction ID (TXID) and the generated Bitcoin Script with the Name-Token envelope are displayed. At this point, the corresponding DNS zone file is also signed with the Nostr private key and published to the configured relay.

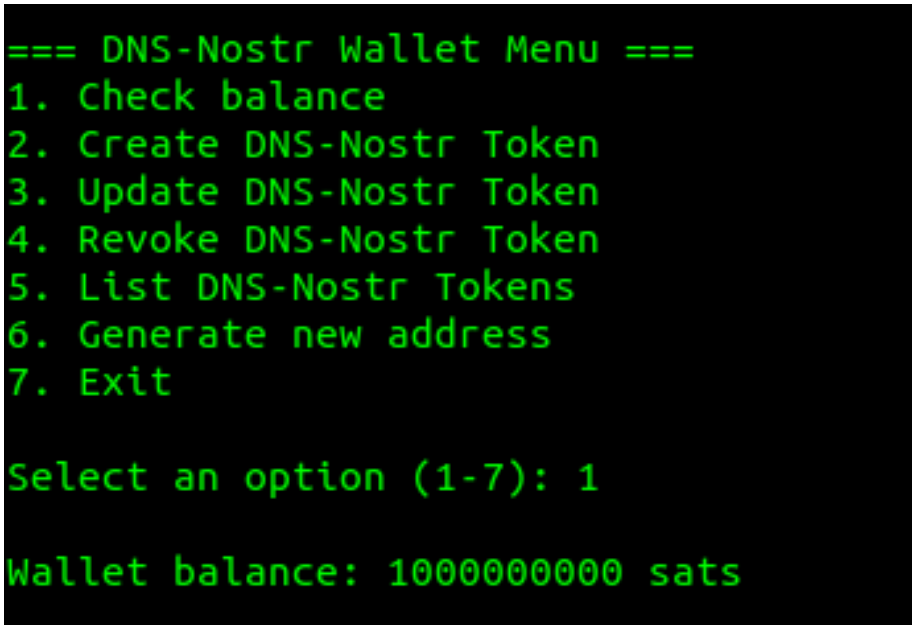

Figure 4.3: Wallet balance check showing available test funds.

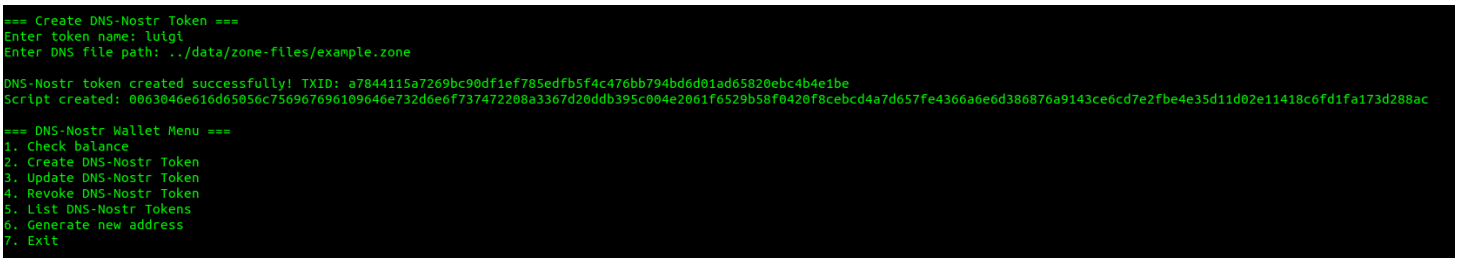

Figure 4.4: Creation of the `luigi` DNS-Nostr-Token.

4.5.2 Blockchain Synchronization

The Name-Token Repository runs as a background service that continuously scans the Bitcoin blockchain to identify and validate Name-Token inscriptions. The repository enforces the protocol rules, including *First-is-Root* uniqueness and *Same-Index Chain* continuity, storing valid tokens in a local SQLite database.

As shown in Figure 4.5, the repository iterates through block heights and applies updates whenever a Name-Token event is detected. This process ensures that the system maintains an up-to-date, fork-resistant view of all valid tokens.

Figure 4.5: Continuous synchronization of blocks and protocol rule validation.

4.5.3 Nostr Integration

Once a Name-Token is created, its associated DNS records are published to the Nostr network as signed events. This guarantees authenticity and censorship resistance, since each record is cryptographically tied to the token's public key.

Figure 4.6 illustrates the use of the NostrDebug tool to verify that the DNS zone file was correctly propagated through a relay. The content includes traditional DNS records (`A`, `AAAA`, `MX`, `TXT`, `CNAME`), demonstrating that the system supports the standard record set while ensuring verifiable authorship.

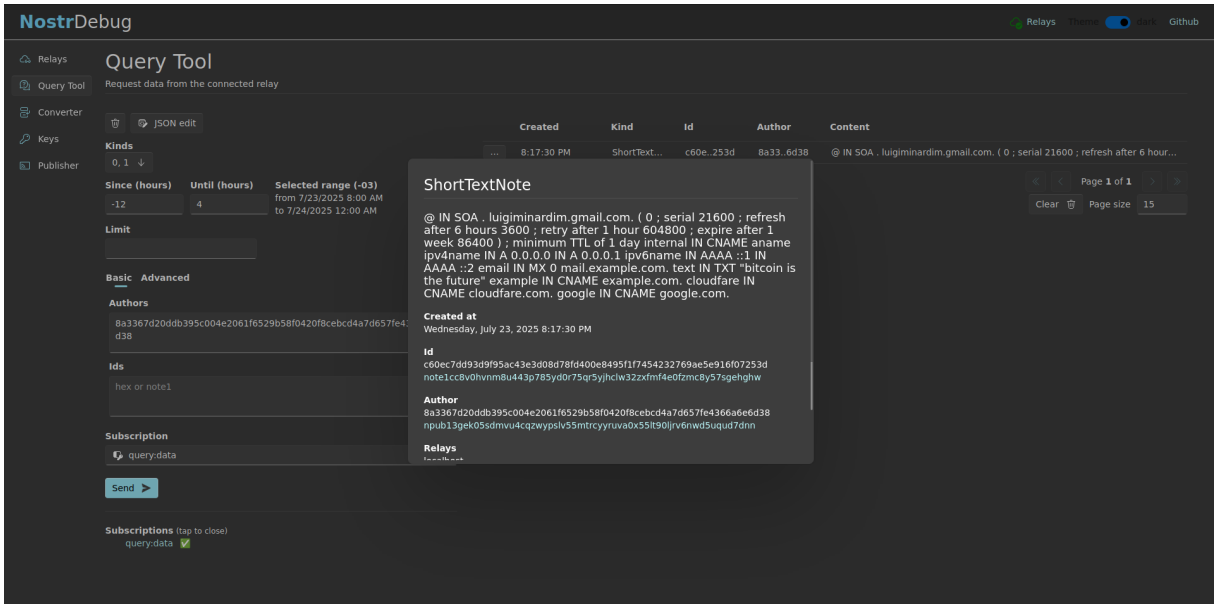

Figure 4.6: Verification of published DNS records via NostrDebug.

4.5.4 DNS Server Resolution

The DNS-Nostr Server functions as a bridge between traditional DNS clients and the decentralized data published through Bitcoin and Nostr. When a query is received, the server locates the corresponding Name-Token in the repository, retrieves the latest zone file from Nostr, and responds with a standard DNS answer.

Figure 4.7 shows the resolution of a TXT record for the subdomain `text.luigi.nostr.dns.name`, where the client successfully receives the response `"bitcoin is the future"`. This validates that the system is compatible with conventional DNS tools such as `dig`, while introducing a decentralized ownership and publication layer.

```
emanuks@emanuks:~/Documents/UnB-FinalProject-DnsOverBitcoin/dns_nostr_server$ dig @0.0.0.0 -p 1053 text.luigi.nostr.dns.name. TXT

; <<>> DiG 9.18.30-0ubuntu0.22.04.2-Ubuntu <<>> @0.0.0.0 -p 1053 text.luigi.nostr.dns.name. TXT
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57050
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1232
;; QUESTION SECTION:
;text.luigi.nostr.dns.name.     IN      TXT

;; ANSWER SECTION:
text.luigi.nostr.dns.name. 86400 IN     TXT     "bitcoin is the future"

;; Query time: 4 msec
;; SERVER: 0.0.0.0#1053(0.0.0.0) (UDP)
;; WHEN: Wed Jul 23 20:18:43 -03 2025
;; MSG SIZE  rcvd: 88
```

Figure 4.7: Successful DNS resolution using the DNS-Nostr Server.

4.5.5 Demonstration Workflow

The proof of concept can be summarized in the following workflow:

1. The wallet creates a new DNS-Nostr-Token and publishes DNS records to Nostr.
2. The repository scans the blockchain, validates the token, and updates its database.
3. The DNS-Nostr Server receives a query, resolves the owner's Nostr key, and fetches the corresponding zone file.
4. The client obtains a valid DNS response (e.g., TXT record), demonstrating end-to-end functionality.

This prototype confirms the viability of the DNS-Nostr-Token protocol and illustrates how Bitcoin and Nostr can be combined to implement a decentralized, censorship-resistant naming system that remains compatible with the existing DNS infrastructure.

Chapter 5

Conclusion: A Foundational Step Towards a Decentralized Web

This project confronts the fundamental political and structural vulnerabilities embedded within the traditional Domain Name System. The proposed DNS-Nostr-Token protocol offers a compelling alternative by directly addressing the core issue of centralized control. By anchoring domain ownership to the Bitcoin blockchain, the protocol displaces the politically mediated governance of centralized authorities such as ICANN with a permissionless, cryptographically verifiable, and censorship-resistant framework. In this paradigm, control over a domain name is not granted by a registrar but inherently derived from the cryptographic ownership of a private key — returning authority to the individual rather than institutional intermediaries.

Despite its advantages in decentralization and censorship resistance, the DNS-Nostr-Token system faces important limitations. Ownership updates anchored in Bitcoin inherit its slow confirmation times and fee volatility, making frequent changes costly and delayed. Off-chain reliance on Nostr relays introduces risks of inconsistency, censorship, or downtime, while indexers must handle the computational burden of scanning the blockchain and resolving potential reorgs. Furthermore, the system is vulnerable to attacks such as mempool front-running, key compromise, and relay manipulation. These challenges highlight that while technically feasible, the proposed architecture still requires performance optimizations, robust key management practices, and multi-relay verification strategies to ensure secure and reliable adoption at scale.

Other decentralized DNS initiatives, such as Unstoppable Domains or the Ethereum Name Service (ENS), demonstrate both the potential and the challenges of blockchain-based naming systems. These solutions benefit from faster confirmation times and integrated smart contracts, but they often depend on alternative chains with weaker security guarantees than Bitcoin and require centralized gateways for practical usability. A key

lesson is the importance of seamless browser integration: today, most decentralized domain systems function only through extensions, custom resolvers, or browser partnerships. For DNS-Nostr-Token, achieving native browser support will be crucial to adoption, since requiring users to install plugins or configure custom resolvers significantly limits reach. Future work should therefore include collaboration with browser vendors, development of lightweight resolver libraries, and integration paths similar to how HTTPS certificates and DNSSEC became standardized in mainstream browsers.

A crucial challenge for any decentralized naming and publishing system is preventing its misuse for harmful or illegal purposes. Unlike traditional DNS, where registrars and centralized operators can suspend or seize domains under clear legal orders, DNS-Nostr shifts responsibility to cryptographic ownership, making direct intervention more difficult. To address this, the protocol must balance censorship resistance with safeguards. Possible mitigations include developing community-run blacklists that resolvers can voluntarily adopt, implementing relay-level filtering policies, and encouraging browsers to integrate opt-in reputation systems for labels and records. Rather than enabling unilateral takedowns, these mechanisms distribute moderation to multiple layers - relays, resolvers, and clients - ensuring that abusive material can be mitigated without reinstating a single point of control. This layered approach preserves the core principle of decentralized ownership while still providing pathways to protect users and comply with societal norms.

Nevertheless, while the protocol successfully addresses the critical issue of centralized root management, its practical implementation introduces a distinct set of architectural trade-offs and operational challenges. These limitations delineate important avenues for future research and development.

5.1 Remaining Challenges and Future Work

The current system, while a functional proof-of-concept, highlights several areas requiring further innovation:

1. **The Trust–Verification Dilemma:** In an ideal decentralized architecture, users would independently verify the blockchain state by running their own full Bitcoin nodes. However, the computational and storage demands of this approach render it impractical for most users. As a result, reliance on third-party DNS-Nostr Servers becomes necessary, reintroducing a trust dependency — albeit shifted from centralized root authorities to server operators — thereby partially undermining the protocol's trustless design.

2. **Economic Incentives and Name Squatting:** The "First-is-Root" ownership paradigm, while conceptually simple, does not inherently address the persistent issue of domain squatting. It creates incentives for speculative mass registration of names without any intent for legitimate use. One potential mitigation strategy in future protocol iterations involves introducing a renewal mechanism that requires periodic on-chain transactions to retain ownership. This recurring cost would discourage passive hoarding and encourage active utilization of domain names.

3. **Protocol-Level Security:** The reliance on Bitcoin's fee market introduces a novel vulnerability: since transactions with higher fees tend to be confirmed more quickly, a malicious actor could attempt to hijack an unconfirmed name by broadcasting a duplicate Name-Token transaction with an identical label but a higher fee, thereby front-running the legitimate registrant. A possible defense against this exploit is to decouple the name-token creation into a two-phase transaction process. In the first transaction, only a cryptographic commitment (i.e., a hash) of the desired label is published. The second transaction then reveals the full label along with its associated metadata. This approach ensures that only the initiator of the initial commitment can later reveal and claim the label, effectively preventing hijacking attempts. In this model, rightful ownership is provably established by the party who first published the commitment transaction.

4. **Off-Chain Data Availability:** The protocol's reliance on the Nostr network for DNS record storage introduces a significant off-chain availability challenge. Given the decentralized and unstructured nature of Nostr relays, it is non-trivial for a DNS-Nostr Server to efficiently determine which of the many relays hosts the most recent and authoritative record for a given name. This can result in increased latency and inconsistent resolution outcomes. A promising avenue for future development is the inclusion of relay hints directly within the Name-Token inscription. By allowing users to embed verifiable metadata pointing to preferred or primary relays, the system could bootstrap the data discovery process with a trusted starting point, improving both resolution reliability and performance.

In summary, the DNS-Nostr-Token protocol represents a meaningful and technically robust advancement toward a user-sovereign naming infrastructure for the internet. By anchoring ownership to cryptographic primitives and public blockchains, it offers a compelling alternative to the centralized and politically vulnerable architecture of the traditional DNS. While not a conclusive solution, it marks an important milestone—one that not only validates the feasibility of decentralized naming on top of Bitcoin but also clearly

outlines the future engineering challenges and design considerations necessary to achieve a more open, censorship-resistant, and resilient web.

References

[1] Cloudflare: *What is dns? / how dns works.* https://www.cloudflare.com/learning/dns/what-is-dns. 1

[2] V. Mockapetris, Paul: *Domain names - concepts and facilities.* https://www.ietf.org/rfc/rfc1034.txt, 1987. 1, 5, 26

[3] Authority, Internet Assigned Numbers: *Root zone management.* https://www.iana.org/domains/root. 2

[4] ISO: *Iso 3166 - country codes.* https://www.iso.org/iso-3166-country-codes.html, 1974. 2

[5] Bruce Postel, Jonathan: *Domain name system structure and delegation.* https://www.ietf.org/rfc/rfc1591.txt, 1994. 2

[6] Authority, Internet Assigned Numbers: *Iana - about us.* https://www.iana.org/about. 2

[7] Runyan, Mallory: *A brief review of dns, root servers, vulnerabilities and decentralization.* https://digitalcommons.odu.edu/cgi/viewcontent.cgi?article=1014context=covacci-undergraduateresearch, 2022. 2

[8] Parliament, European: *Internet governance - keeping the internet open, free and unfragmented.* https://tinyurl.com/y7zt35ya, 2024. 2

[9] Wothaya, Jacinta: *What is censorship and what tools can sjos use to bypass restricted content.* https://tatua.digital/services/what-is-censorship-and-what-tools-can-sjos-use-to-bypass-restricted-content/, 2024. 3

[10] Nosyk, Yevheniya; Lone, Qasim; Zhauniarovich Yury; Gañán Carlos H.; Aben Emile; Moura Giovane C.M.; Tajalizadehkhoob Samaneh; Duda Andrzej; Korczyński Maciej: *Intercept and inject - dns response manipulation in the wild.* https://pure.tudelft.nl/ws/portalfiles/portal/151232870/978_3_031_28486_1_19.pdf, 2023. 3

[11] Terceiro, Ivanildo; Bastos, J.P.; Trujillo Mariana: *How brazil's x ban signals growing control over online free speech.* https://reason.org/commentary/how-brazils-x-ban-signals-growing-control-over-online-free-speech/, 2025. 3

[12] Mozelli, Rodrigo: *Como é feito o bloqueio de uma rede social no brasil?* https://olhardigital.com.br/2024/08/29/pro/como-e-feito-o-bloqueio-de-uma-rede-social-no-brasil/, 2024. 3

[13] Droms, Ralph and Steve Alexander: *Dhcp options and bootp vendor extensions.* https://www.ietf.org/rfc/rfc2132.txt, 1997. 6

[14] V. Mockapetris, Paul: *Domain names - implementation and specification.* https://www.ietf.org/rfc/rfc1034.txt, 1987. 7, 23

[15] AFNIC: *Could blockchain (really) replace dns?* https://www.afnic.fr/wp-media/uploads/2024/06/Could-Blockchain-really-replace-DNS-Afnic-Issue-Paper.pdf, 2024. 8, 9

[16] Aniello, Leonardo; Kang, BooJoong; Giamouridis George: *Blockchain-based dns: Current solutions and challenges to adoption.* https://ceur-ws.org/Vol-3791/paper16.pdf, 2024. 8, 9

[17] Sea, Open: *What are nft domain names?* https://opensea.io/learn/nft/what-are-nft-domain-names, 2023. 9

[18] Sea, Open: *What is an nft?* https://opensea.io/learn/nft/what-are-nfts, 2022. 9

[19] Nakamoto, Satoshi: *Bitcoin: A peer-to-peer electronic cash system.* https://bitcoin.org/bitcoin.pdf, 2008. 9

[20] Eck, Van: *Bitcoin vs. ethereum in 2025: Comparison outlook.* https://www.vaneck.com/us/en/blogs/digital-assets/bitcoin-vs-ethereum/, 2025. 9

[21] CoinMarketCap: *Bitcoin dominance.* https://coinmarketcap.com/charts/bitcoin-dominance/. 9

[22] Buterin, Vitalik: *Ethereum: A next-generation smart contract and decentralized application platform.* https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf, 2014. 10

[23] M. Antonopoulos, Andreas and David A. Harding: *Mastering bitcoin, programing the open blockchain.* https://github.com/bitcoinbook/bitcoinbook/, 2023. 12, 13

[24] Fiatjaf: *Nostr - notes and other stuff transmitted by relays.* https://fiatjaf.com/nostr.html, 2019. 24

[25] Fiatjaf: *Nip-01 - basic protocol flow description.* https://github.com/nostr-protocol/nips/blob/master/01.md, 2022. 26

[26] Rusnak, Pavol: *Bip-84.* https://github.com/bitcoin/bips/blob/master/bip-0084.mediawiki, 2017. 29

[27] Rusnak, Pavol and Marek Palatinus: *Bip-44.* https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki, 2014. 29

[28] Fiatjaf: *Nip-06 - basic key derivation from mnemonic seed phrase.* https://github.com/nostr-protocol/nips/blob/master/06.md, 2022. 29