University of Dublin

Trinity College Dublin

Blockchain Backed DNSSEC

Scarlett Gourley

B.A. (Mod) Integrated Computer Science

Final Year Project May 2018

Supervisor: Dr. Hitesh Tewari

School of Computer Science and Statistics

O'Reilly Institute, College Green, Dublin 2, Ireland

Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

Dublin, May 3 2018

Scarlett Gourley

Permission to Lend

I agree that the Library and other agents of the College may lend or copy this report upon request.

Dublin, May 3 2018

Scarlett Gourley

Abstract

The traditional Domain Name System (DNS) did not include any security details, making it vulnerable to a variety of attacks which were discovered in 1990. The Domain Name System Security Extensions (DNSSEC) attempted to address these concerns and extended the DNS protocol to add origin authentication and message integrity whilst remaining backwards compatible. Yet despite the fact that issues with DNS have been well known since the late 90s, there has been very little adoption of DNSSEC.

This project presents an analysis on why there has been little momentum securing the DNS protocol. As part of this research a new system is proposed using blockchain technology. This system aims to provide the same security benefits as DNSSEC whilst addressing the concerns that led to its slow adoption.

Acknowledgements

Firstly, I would like to thank my supervisor Dr. Hitesh Tewari for his continuous time, guidance, and for being a constant source of ideas and knowledge.

I am eternally thankful to my parents for their unending support and encouragement throughout the years.

I would like to thank Dário for his incredible patience and help over the last four years.

I would also like to acknowledge the many people who helped shape this project.

Acronyms

2LD	Second Level Domain
CA	Certificate Authority
CSK	Combined Signing Key
CSR	Certificate Signing Request
DSA	Digital Signature Algorithm
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DS	Delegation Signer
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
IP	Internet Protocol
KSK	Key Signing Key
NS	Name Server
PKI	Public Key Infrastructure
RSA	Rivest-Shamir-Adleman
RR	Resource Record
TLD	Top Level Domain
ZSK	Zone Signing Key

Contents

Introduction 1

The Domain Name System (DNS) in its original form is insecure. It suffers from a number of protocol attacks. These include DNS spoofing/cache poisoning [1], DNS hijacking [2], and DNS rebinding [3].

A solution to this was the introduction of DNSSEC. However, DNSSEC also suffers from various problems, namely IP fragmentation [4], potential denial of service attacks[5], and complicated key management. This has caused a slow adoption of DNSSEC, with only 4% of second level domains signed [6].

This report aims to outline an alternative security extension set for DNS. Specifically, the proposed system aims to fulfil the following goals:

- Provide the client with origin authentication
- Provide the client with message integrity
- Reduce the size of DNSSEC responses to the traditional DNS packet size
- Reduce number of requests necessary for signature validation
- Simplify the key management

These goals will be revisited at various stages throughout the project, in order to assess if the proposed system achieves these objectives.

1.1 Report Structure

Chapter 2 outlines the background of the project. The background consists of technologies which are necessary building blocks for the rest of this report.

Chapter 3 examines and gives a detailed review on current solutions and technologies either in action or described in a theoretical capacity.

Chapter 4 explains the design of the solution and each of its components.

Chapter 5 describes a partial implementation of the design. Sample output from the system is shown and explained, as well as a comparison to DNSSEC.

Chapter 6 analyses statistics relating to current DNSSEC deployment and the designed system.

Chapter 7 reflects upon the project and touches on several aspects that is open to future research.

Background

2

This chapter aims to give an understanding and background on the technologies related to this project. This includes an overview of how DNS functions today, as well as cryptographic primitives and technologies. A basic understanding of networking fundamentals is assumed of the reader.

2.1 Cryptography

Crytography is a set of tools to ensure secret communication between two parties. In the modern age, it helps to secure systems for ordinary people [7].

2.1.1 Symmetric Cryptography

Symmetric crytography (also known as private key cryptography) is a system where two or more parties have a shared secret key, known as a private key. Using this key, the parties can encrypt and decrypt information. Once the information is encrypted, it can be freely sent over an insecure channel without being compromised [7].

2.1.2 Hash Functions

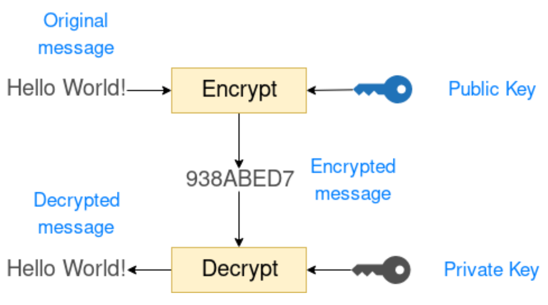
A hash function is a mathematical function that takes a string of arbitrary length and compresses it into a string of fixed length [7]. This is known as a message digest.

A “good” hash functions aims to be collision-resistant or injective [7], i.e. $\forall x, x' \in X, H(x) \neq H(x')$.

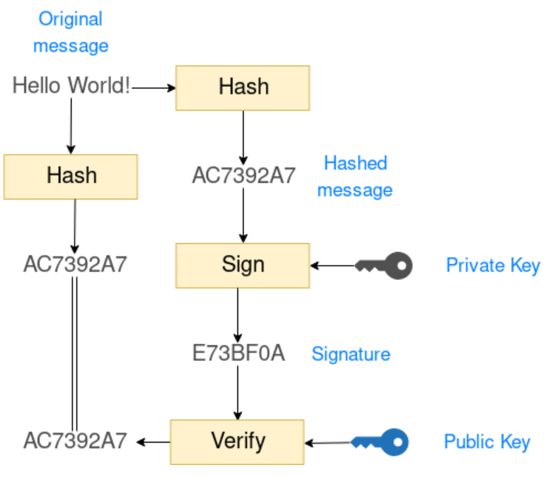
2.1.3 Asymmetric Cryptography

Asymmetric cryptography (also known as public key cryptography) is a system which generates a pair of keys. One of the keys generated is a public key which is freely distributed. The other key is the private key, which is only known to the owner [7].

The strength of an asymmetric crytosystem is the computational effort it takes to find the private key from its paired public key [8]. The two best known and main uses of public key cryptography are encryption and digital signatures. Examples of both processes are depicted in Fig 2.1a and Fig 2.1b respectively.



(a) Encryption and Decryption



(b) Signing and Verifying

Fig. 2.1: Public Key Cryptography

Anyone holding someone's public key can use it to encrypt a message. This renders the message unreadable to everyone except the owner of the corresponding private key, who can decrypt the message [7]. This ensures only the sender and the recipient know the contents of the message (confidentiality).

To create a digital signature, the sender first creates a message digest of the message they wish to send. They follow on by "signing" the message digest with their private key. They send this signature along with the original message. The recipient uses the public key of the sender on the signature, and compares the result to their own hashed version of the message [7]. This allows the recipient to verify that the message came from the sender (authentication), and that the contents have not been altered (integrity).

2.1.4 Merkle Tree

A Merkle tree (or hash tree) is a tree in cryptography where every leaf is the hash of a data block, and every non-leaf is the hash of the concatenation of its leaf nodes [9].

For example, if a non-leaf has two children with values `H(A)` and `H(B)`, then it would have the value `(H(A)|H(B))`. This is illustrated in Fig 2.2.

Only the root of the tree needs to be known to verify that a data block is part of the tree. If the underlying data changes, its hash will also change. This will propagate upwards in the tree, eventually generating a different hash for the root [9].

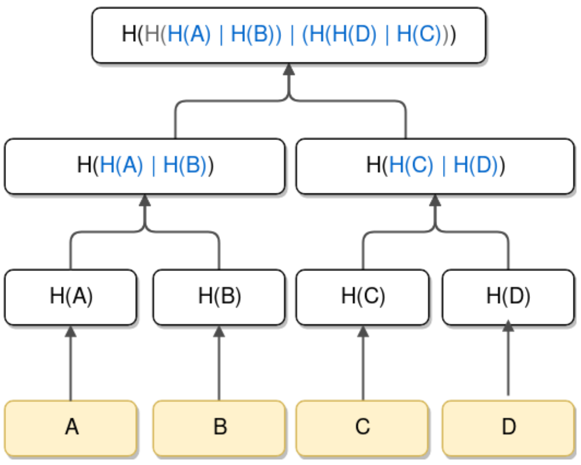


Fig. 2.2: Example Merkle Tree

Merkle trees are useful in peer-to-peer networks to ensure data blocks arrive undamaged and unaltered.

2.2 PKI

A public key infrastructure (PKI) is a system for managing, distributing and revoking digital certificates [10].

A digital certificate is a digitally signed electronic document that contains information on the owner, intended to bind their identity and public key. These are created and issued by trusted third parties, known as Certificate Authorities (CAs) [10].

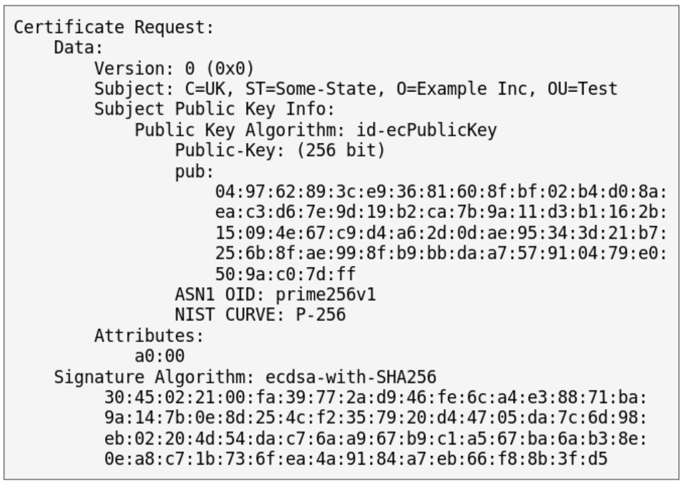
2.2.1 X.509

X.509 is a digital certificate standard that is used in many internet protocols, such as HTTPS [11]. Its PKI consists of [12]:

CA	Certificate Authority
RA	Registration Authority
CRL issuer	A system which generates and signs CRLs
Repository	A system for storing and distributing certificates and CRLs

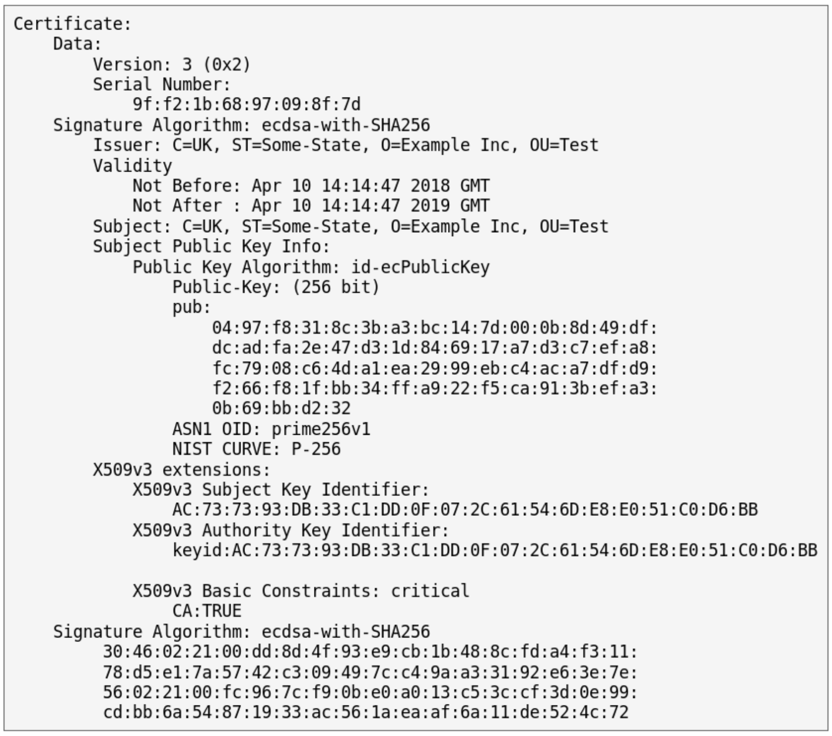
In order for an entity to receive an X.509 certificate, they must create a Certificate Signing Request (CSR) with their information, which includes their identity and their public key. This CSR gets sent to a Registration Authority [12]. A Registration Authority is approved by a CA to accept, reject and revoke certificates [12][13]. If the CSR is accepted, an X.509 certificate signed by the CA is returned to the entity [12].

An example CSR and certificate is shown in Fig 2.3 and 2.4 respectively.



```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=UK, ST=Some-State, O=Example Inc, OU=Test
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:97:62:89:3c:e9:36:81:60:8f:bf:02:b4:d0:8a:
                    ea:c3:d6:7e:9d:19:b2:ca:7b:9a:11:d3:b1:16:2b:
                    15:09:4e:67:c9:d4:a6:2d:0d:ae:95:34:3d:21:b7:
                    25:6b:8f:ae:99:8f:b9:bb:da:a7:57:91:04:79:e0:
                    50:9a:c0:7d:ff
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            a0:00
    Signature Algorithm: ecdsa-with-SHA256
         30:45:02:21:00:fa:39:77:2a:d9:46:fe:6c:a4:e3:88:71:ba:
         9a:14:7b:0e:8d:25:4c:f2:35:79:20:d4:47:05:da:7c:6d:98:
         eb:02:20:4d:54:da:c7:6a:a9:67:b9:c1:a5:67:ba:6a:b3:8e:
         0e:a8:c7:1b:73:6f:ea:4a:91:84:a7:eb:66:f8:8b:3f:d5
```

Fig. 2.3: Example X.509 CSR with `openssl`



```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            9f:f2:1b:68:97:09:8f:7d
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=UK, ST=Some-State, O=Example Inc, OU=Test
        Validity
            Not Before: Apr 10 14:14:47 2018 GMT
            Not After : Apr 10 14:14:47 2019 GMT
        Subject: C=UK, ST=Some-State, O=Example Inc, OU=Test
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:97:f8:31:8c:3b:a3:bc:14:7d:00:0b:8d:49:df:
                    dc:ad:fa:2e:47:d3:1d:84:69:17:a7:d3:c7:ef:a8:
                    fc:79:08:c6:4d:a1:ea:29:99:eb:c4:ac:a7:df:d9:
                    f2:66:f8:1f:bb:34:ff:a9:22:f5:ca:91:3b:ef:a3:
                    0b:69:bb:d2:32
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                AC:73:73:93:DB:33:C1:DD:0F:07:2C:61:54:6D:E8:E0:51:C0:D6:BB
            X509v3 Authority Key Identifier:
                keyid:AC:73:73:93:DB:33:C1:DD:0F:07:2C:61:54:6D:E8:E0:51:C0:D6:BB

            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: ecdsa-with-SHA256
         30:46:02:21:00:dd:8d:4f:93:e9:cb:1b:48:8c:fd:a4:f3:11:
         78:d5:e1:7a:57:42:c3:09:49:7c:c4:9a:a3:31:92:e6:3e:7e:
         56:02:21:00:fc:96:7c:f9:0b:e0:a0:13:c5:3c:cf:3d:0e:99:
         cd:bb:6a:54:87:19:33:ac:56:1a:ea:af:6a:11:de:52:4c:72
```

Fig. 2.4: Example X.509 Certificate with `openssl`

Once a certificate is returned to the owner, it can be used to establish a secure connection using their public key. In order to verify the certificate's public key, the CAs public key that signed the certificate must also be validated. This verification process is continued up until a trusted certificate, which is stored on the client's machine. Once a trusted signature is identified, it can be believed that the public key contained in the original certificate belongs to the certificate owner [10][12].

2.3 DNS

The Domain Name System (DNS) is a hierarchical and decentralised naming service, which translates human readable, more readily memorable names into IP addresses. It is composed of name servers which contains information for a particular DNS zone, and resolvers which extract information from name servers for a client. A DNS zone is a portion of the domain name space in the DNS which is managed by a specific organisation or administrator [14].

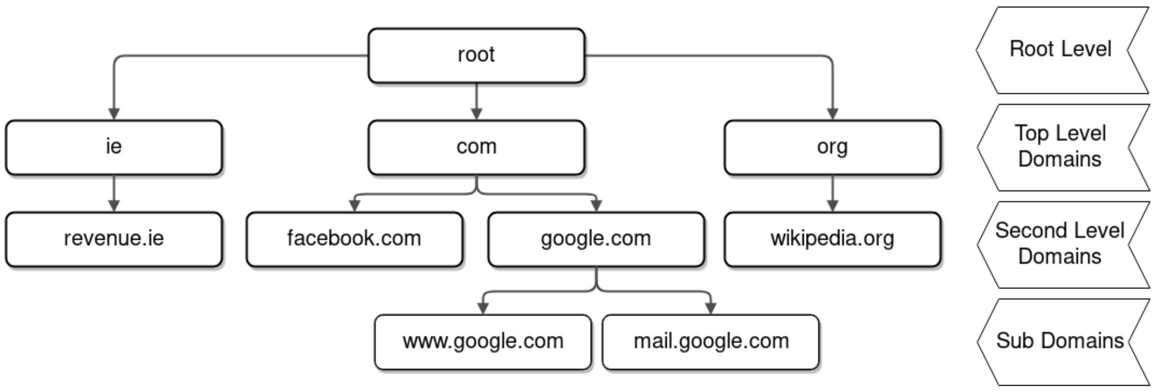


Fig. 2.5: Example hierarchy

Name servers are structured in a tree, with the root name servers at the top, followed by the TLDs. An example hierarchy is shown in Fig 2.5. Most TLD name registry operators allow the public to register second level domains.

Whilst the primary concern for DNS is to map domain names to IP addresses, it supports a number of records. The primary ones are as follows [15] [16]:

A	Maps a domain name to an IPv4 address[1]
AAAA	Maps a domain name to an IPv6 address[2]
NS	Points to an authoritative name server for a domain
CNAME	Allows the specification of an alias
MX	Points to a mail server for a domain

Name servers respond to queries with the appropriate RR. If it is configured as authoritative, then the RRs for its particular zone is always interpreted as correct [14].

The DNS also contains resolvers, which are split into two different categories; stub and recursive. A stub resolver simply forwards a query to a recursive resolver. This recursive resolver will perform multiple queries on the hierarchy of name servers in order to determine an appropriate response [14].

Recursive resolvers can cache records from authoritative name servers until a particular expiration date. This is to reduce traffic and improve speed of responses. These records are seen as non-authoritative.

2.3.1 Look Ups

A domain name is structured from the chain of names in the name server tree. For example, if `example` was registered under the `com` TLD it would be identified as `example.com`. A DNS lookup follows this chain from a root server to the requested zone's name server.

A lookup example is as follows [15]:

1. A client begins a lookup for an IP address for `google.com` by sending a query to a stub resolver.
2. The stub resolver forwards the query to a recursive, caching DNS resolver.
3. The DNS resolver checks its cache for `google.com`. If it is not found it requests a lookup from one of the 13 root name servers.
4. The root name server will send back a NS RR for the `com` authoritative name server. Another lookup would need to commence in order to find out the address of this name server, so the response will also contain a "glue record" which contains an A RR.
5. The DNS resolver requests a lookup for `google.com` from the `com` authoritative name server.
6. The `com` name server responds with the NS RR for the `google.com` authoritative name server, along with the "glue record".
7. The DNS resolver requests a lookup for `google.com` from the `google.com` authoritative name server.
8. The `google.com` name server responds with an A RR, containing the IPv4 address for `google.com`.
9. The DNS resolver updates its cache with the A RR and forwards the response to the stub resolver.
10. The stub resolver forwards the response back to the client.

2.3.2 Attacks

Due to the lack of security mechanisms in the DNS, there are a number of protocol attacks that exist.

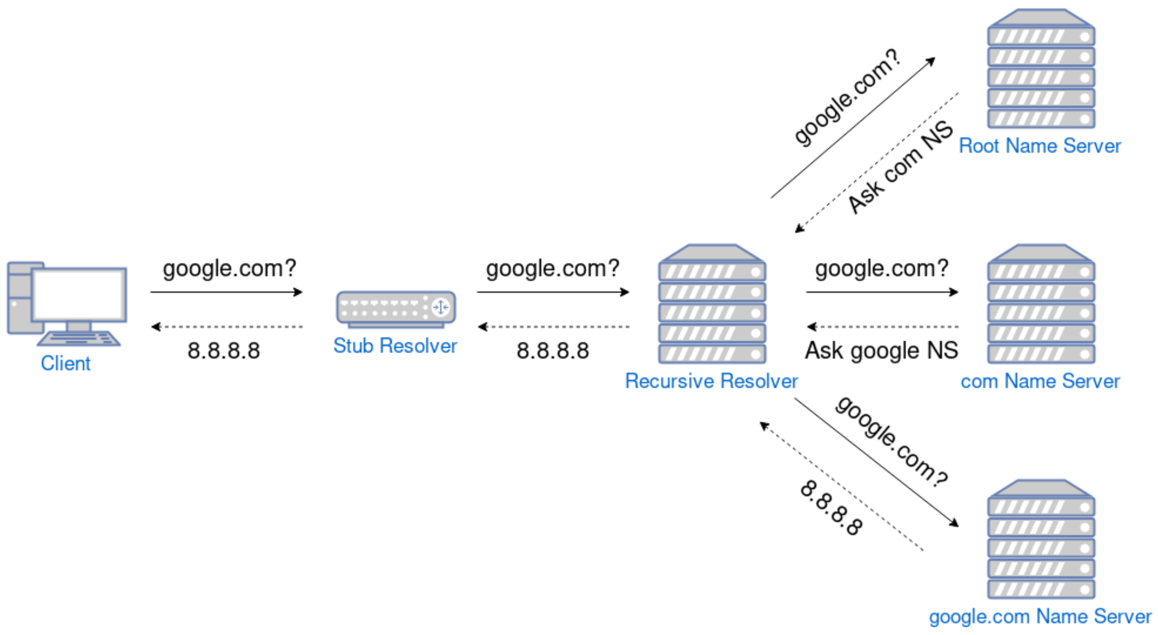


Fig. 2.6: Example Lookup

2.3.2.1 Spoofing/Cache Poisoning

A DNS message has four sections to it [14]:

Question	The query
Answer	The RRs directly responding to the query
Authority	The RRs describing other authoritative servers
Additional	Some RRs which might be helpful

A spoofing or cache poisoning attack involves sending incorrect information in the authority or additional section which the resolver subsequently caches [1]. For example, the “glue” A RR contained in the additional section could be a forged IP address redirecting all traffic that uses that name server to a malicious server.

2.3.2.2 Hijacking

DNS hijacking is a type of attack which redirects a client’s DNS queries to a different domain name server [2]. There are two methods of doing this.

One method of DNS hijacking is when a computer’s DNS settings are changed so that a trusted resolver is no longer used [2]. This causes forged IP addresses to be returned to the client.

The second is when a website is broken in to and their IP addresses are changed [2]. If this happens then during a DNS look-up their domain will resolve to a malicious website instead.

2.3.2.3 Rebinding

A rebinding attack abuses the DNS by mapping a domain name to the target's IP address. This allows a malicious website to abuse the same-origin policy[3] and attack other machines on the target's network [3].

2.4 DNSSEC

The Domain Name System Security Extensions (DNSSEC) is a set of security extensions which provides origin authentication and message integrity. It does this by using public key based digital signatures. It does not increase availability or grant message confidentiality.

To facilitate security in the DNS, some new RR were added [19]:

RRSIG	Contains a digital signature
DNSKEY	Contains a signing key
DS	Contains the hash of a DNSKEY
NSEC and **NSEC3**	Maps a denial of existence to a domain range

RRs are grouped together to form RRsets. The RRset is digitally signed to form an RRSIG record [19]. Hence, anytime a request is made for a domain's RR (e.g. `A`) all the RRs for of that type for the aforementioned domain must also be requested in order to create the RRset to verify the RRSIG.

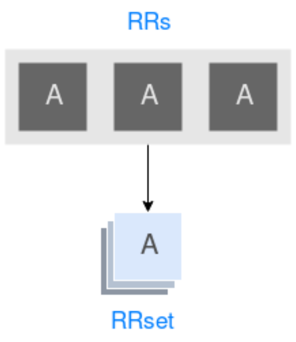


Fig. 2.7: RRset

Each zone has a Zone Signing Key (ZSK) pair. The private key portion is used to digitally sign the RRsets. The public portion is stored as a DNSKEY record in the name server. This DNSKEY also needs signing in order to prove its integrity. Each zone also has a Key Signing Key (KSK), which is used to sign the RRset of DNSKEY records. The public portion of the KSK is also stored as a DNSKEY record [19].

[3]A websites script can only access data if the location of that data is of the same origin as the script

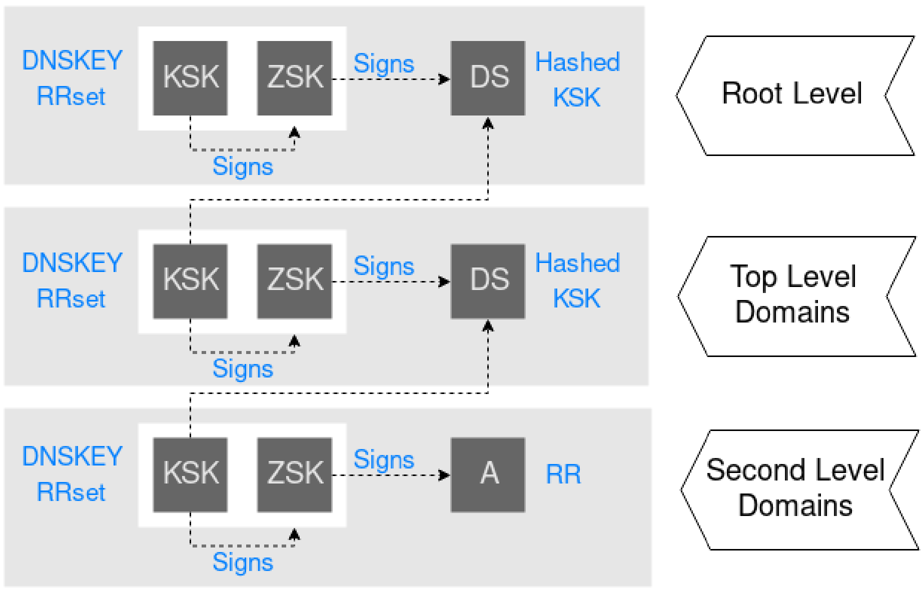


Fig. 2.8: Chain of Trust

A DS RR is created by generating the hash of the public portion of the KSK. This hash is added to the parent zone (e.g. `google.com` would place their DS RR in the `com` zone). The parent would sign these DS RRs with their ZSK. This carries up to the root server [19]. Fig 2.8 contains a visualisation of this trust chain.

A root signing ceremony is held where individuals come together and sign the root DNSKEY RRset in a highly public and audited way. The RRSIG generated can be used to verify the root server's KSK and ZSK. This allows us to trust the root key, and creates a chain of trust for which we can validate any DNSSEC enabled zone.

2.4.1 Denial of Existence

Since every record must be signed, each query that asks for a domain name that does not exist must also get a signed response. It would require an infinite amount of disk space to store an RRSIG for every possible domain name within a zone. Instead, NSEC RRs explicitly map a range of domains that do not exist [19]. For example, in the `com` zone there might be two domains lexicographical side by side, `a.com` and `c.com`. A request for `b.com` would get an NSEC record that says, "No domains exist between `a.com` and `c.com`".

Being able to get a range of domains that do not exist makes it trivial to compile a list of every domain that does exist by collecting the complete set of NSEC RRs. This is known as zone walking or zone enumeration. NSEC3 RRs store the hash of the domain names that do not exist rather than the domain name [20].

2.4.2 Algorithms

There are a number of digital signature algorithms and hash function combinations supported and standardised for DNSSEC. Table 2.1 lists all the algorithms which have been standardised and support zone signing.

Algorithm	Signature Algorithm	Hash Function	Mnemonic
3	DSA	SHA-1	DSA
5	RSA	SHA-1	RSASHA1
6	DSA	SHA-1	DSA-NSEC3-SHA1
7	RSA	SHA-1	RSASHA1-NSEC3-SHA1
8	RSA	SHA-256	RSASHA256
10	RSA	SHA-512	RSASHA512
12	GOST R 34.10-2001	GOST R 34.11-94	ECC-GOST
13	ECDSA Curve P-256	SHA-256	ECDSAP256SHA256
14	ECDSA Curve P-384	SHA-384	ECDSAP384SHA384
15	Ed25519	SHA-256	ED25519
16	Ed448	SHA-256	ED448

Tab. 2.1: DNSSEC Algorithms with Zone Signing Support [21]

The signature algorithm specifies how to produce the key pairs, and how to perform signature creation and validation. It is responsible for the creation of information stored in DNSKEY and RRSIG RRs. The hash function is used to produce the message digest which is subsequently signed.

Algorithms supporting zone signing have their number stored in the “Algorithm Number” field of DNSKEY, RRSIG and DS RRs [19].

A number of hash functions are supported, outlined in Table 2.2. These are used to produce the information stored in DS RRs. This number is stored in the “Digest Number” field of DS RRs [19].

Value	Hash Function
1	SHA-1
2	SHA-256
3	GOST R 34.11-94
4	SHA-384

Tab. 2.2: DNSSEC DS Hash Functions [22]

2.4.3 Problems

DNSSEC suffers from a few issues which can lead to accessibility problems.

2.4.3.1 IP Fragmentation

Traditional DNS messages are restricted to 512 bytes [15]. With the burden of signatures and DNSKEY RRs in DNSSEC, this restriction has been lifted [23]. However, the minimum IP reassembly buffer size must be 576 bytes or more [24]. This includes the IP header and UDP header, which are a maximum of 60 bytes and 8 bytes respectively. Normally an IP header does not reach this maximum, so accounting for an IP header of 56 bytes is seen as sufficient. A 512 byte UDP payload is seen as the largest safe UDP packet size.

If a packet exceeds 576 bytes, the IP layer is under no obligation to transfer it in a single datagram and thus can fragment the message into various parts to be reassembled later. This is known as IP fragmentation and can cause internet zones to become unreachable [4].

2.4.3.2 Denial of Service

A denial of service attack intends to swarm a machine with so much traffic it is too busy to process normal network traffic [25]. An amplification attack is a type of denial of service attack. It operates by an attacker initiating a series of small queries which result in a number of significantly larger queries being sent to a target [5].

DNS queries are very small, whereas a DNSSEC responses can be quite large due to the inclusion of signatures. This makes DNSSEC open to amplification attacks.

2.4.3.3 Complicated Key Management

Due to the fact that DNSSEC has a complicated key structure, it can be difficult and tedious to correctly configure a zone. If a zone is not correctly configured then it can seem as if an attack is happening which may lead clients to reject the resolution.

2.5 Blockchain

A blockchain is an append only chain of blocks, where each block is connected by containing the hash of the previous block's header [26]. Each block pointing to its predecessor forms an immutable chain. If a block were to be altered its hash would also change, which would alter the next block, and so on.

Each block has a header, which contains a timestamp, the previous block header's hash, the merkle root, and an optional nonce. The rest of the block is transaction data, which a merkle tree is generated from [26].

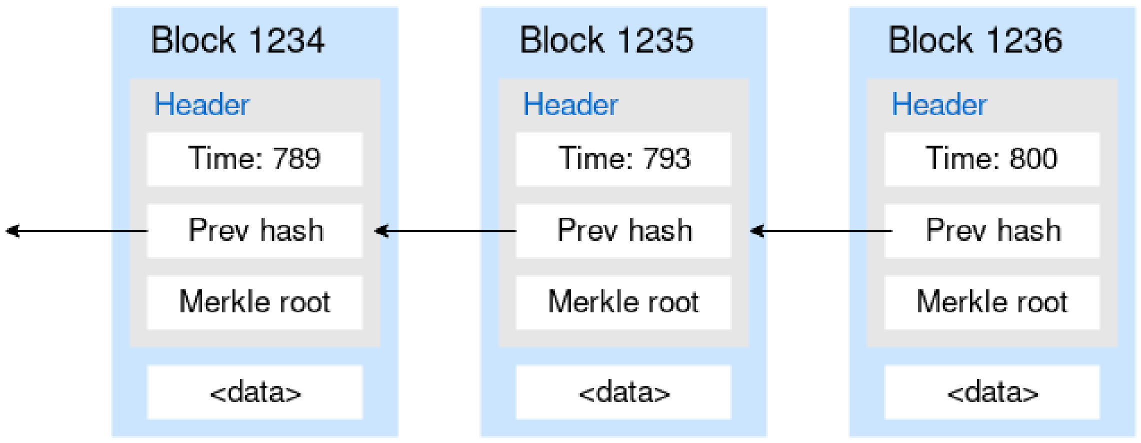


Fig. 2.9: Blockchain

The decision to only include the merkle root in the header was to save disk space [26]. If a malicious user were to attempt to alter any transactions the change would propagate upwards and change the merkle root, also changing the hash of the header.

Blockchains are typically managed on a peer-to-peer network, where each node stores the state of the blockchain. The true state of the blockchain is seen as the chain with the largest number of blocks.

2.5.1 Cryptocurrency

A cryptocurrency is a digital asset, where transactions of this asset are secured using digital cryptography to validate transfers [26]. The decentralised use of this asset was first achieved by the Bitcoin network [27].

2.5.2 Mining

In order to mitigate the ability to change the history of the blockchain, it must be time consuming to create a block. Ideally, it should be a system which is costly to produce, but easy for others to verify that it satisfies specific requirements. This is known as a proof of work system.

A "miner" who is attempting to add a block to the blockchain must attempt to find a nonce in the block header, which when hashed produces a value with a predetermined amount of leading zeros [26][28]. The difficulty can be adjusted by increasing the number of leading zeroes necessary [26].

In order to incentivise mining, a specific amount of a blockchains cryptocurrency may be granted to the miner [26]. A transaction fee is also often used.

2.5.3 Adding

The process of adding a transaction to the blockchain operates as follows [26]:

1. Each transaction is broadcasted to all nodes in the network
2. Nodes will collect transactions into blocks
3. If a proof of work scheme is being used, the mining process will begin
4. Once a valid block is formed, a node will broadcast this block to the entire network
5. Other nodes will check this block to ensure it is valid
6. If it is accepted, the next block will be appended to the end of this one, otherwise it will be ignored

2.5.4 Attacks

Blockchains are susceptible to a number of attacks.

2.5.4.1 Denial of Service

If a lot of traffic is sent to a node, it could keep the node so busy it cannot read and process normal network traffic. This is known as a denial of service attack [25].

A denial of service attack becomes more difficult with more active nodes on a network.

2.5.4.2 51%

A malicious user could create a transaction to an online merchant, and once the asset arrives they could go back to their original transaction and change the block, creating a “fork” (or “double spending”). If they were able to make this chain longer than the original chain be redoing all of the blocks that followed, they could rewrite the history of the blockchain. This is known as a majority attack or a 51% attack.

To mitigate this, transactions must be publicly announced and a consensus mechanism is necessary [26]. Mining largely alleviates this issue as a lot of computing power is needed in order to perform it [26].

2.5.4.3 Sybil

An attacker could fill the network with nodes that they control, making it likely that a client would connect to this node [29]. This leads to issues where an attacker can choose to only relay specific traffic, which leaves a client open to double spend attacks.

Restricting outbound connections can reduce the likelihood of this attack.

State of the Art 3

This chapter assesses a variety of technologies which range from security solutions for DNS, alternative DNS protocols, and PKI proposals.

3.1 DNSCurve

DNSCurve is a new security protocol proposed for DNS. It encrypts and authenticates DNS packets sent between resolvers and authoritative servers using keys which were established with elliptic curve cryptography [30].

NS records are prepended with `uz5` and a base32 encoding of the name server's public key [30]. This public key is used to establish the connection between the resolver and name server.

DNSCurve provides confidentiality, integrity and availability [30]. However, it does not provide authentication. An example lookup using this protocol is as follows [31]:

1. The recursive resolver performs a normal resolution up until the `org` name server and asks the `org` name server for a `dnscurve.org` A record
2. The `org` name server responds with the NS and "glue" RRs for `dnscurve.org`
3. The recursive resolver notices the magic string `uz5` at the start of the name server entry and extracts the base32 encoded public key
4. The recursive resolver calculates a 96-bit nonce and sends the name server
   - Its public key
   - Its nonce
   - A "crytographic box" containing the query, made up of the resolver's private key, name server's public key, and the nonce
5. The name server unpacks this "cryptographic box", extracting the query that it needs to respond to
6. The name server calculates its own 96-bit nonce and sends the resolver
   - The client's nonce
   - Its nonce

    - A “crytographic box” containing the response, made up of the resolver’s public key, name server’s private key, and the new nonce

7. The resolver unpacks the box and forwards the response back to the client

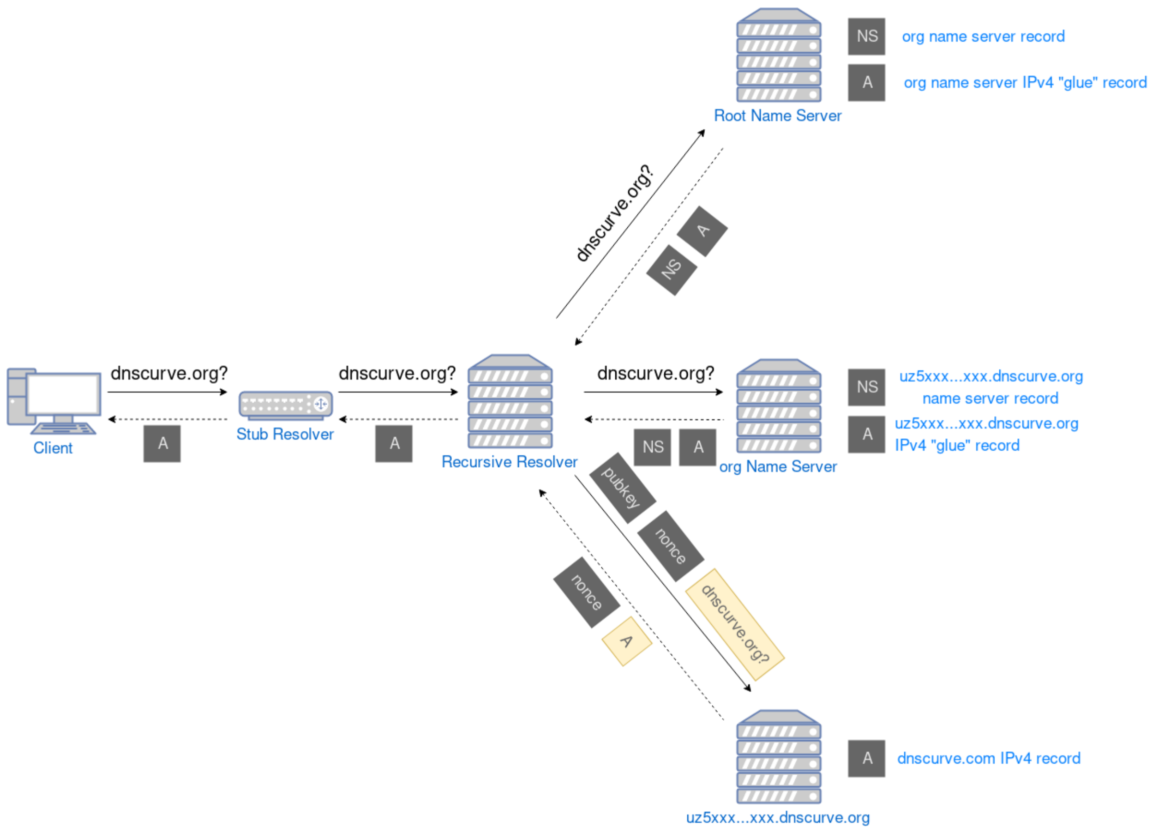


Fig. 3.1: DNSCurve Lookup

3.1.1 Discussion

DNSCurve fills a different void than what DNSSEC does and are both compatible with one another. Although both attempt to provide message integrity, DNSCurve’s method suffers from a number of problems.

Firstly, since messages are encrypted and decrypted on the fly, each name server must store the private key. If a name server has multiple IP addresses then the same private key must be stored on all of them as the public key is encoded into the NS RR.

There is also the issue that the TLD name server could encode the name server record with a rogue public key (or no public key at all) and perform a man in the middle attack on the network traffic. For example, `org` could change the public key for `dnscurve.org` and then intercept the traffic between the resolver and `dnscurve.org`.

If there was a trust chain from the root server, it could be verified that the TLD name server is who it claims to be, i.e. what DNSSEC does.

3.2 DNS-based Authentication of Named Entities

DNS-based Authentication of Named Entities (DANE) is a protocol which uses DNSSEC to bind X.509 certificates to DNS names [32]. It is a way for a client to authenticate to a server without the use of a CA. Certificates from CAs are currently used in protocols such as TLS [33].

DANE specifies a "TLSA" RR, which describes constraints on the certificate to be found at a specific domain name [32]. These records are placed at a subdomain which is generated as follows [32]:

1. An underscore, followed by a port number (e.g. `_443`)
2. An underscore, followed by a transport layer protocol name (e.g. `_tcp`)
3. Optionally, a subdomain name (e.g. `www`)

For example, a TLSA RR may be stored at `_443._tcp.www.example.com` and specifies constraints on the certificate found at port 443 (HTTPS), using the TCP transport layer protocol, at the domain name `www.example.com` [32].

A TLSA RR contains a number of fields specifying the type of certificate, as well as either the certificate data or the hash of this data depending on the criteria for matching [32].

3.2.1 Discussion

DANE serves a number of use cases, such as [34]:

- Specifying CA constraints, such as who their CA is
- Validating that the certificate is the correct one, and not a fake one deployed by an insecure CA
- The choice to not use a CA at all, but for a client to still be able to validate the entity
- Delegating services to other DNS servers whilst still controlling which certificates are used
- Encryption over email

This provides us with certificate integrity, certificate authentication and confidentiality. It is used alongside DNSSEC to provide origin authentication and message integrity for the TLSA RRs.

3.3 Ethereum Name Service

Ethereum Name Service (ENS) is a naming system built on top of the Ethereum blockchain. The original use was to translate human readable and memorable names to Ethereum addresses [35]. It was designed with flexibility in mind, and has expanded to also host DNS domains [36].

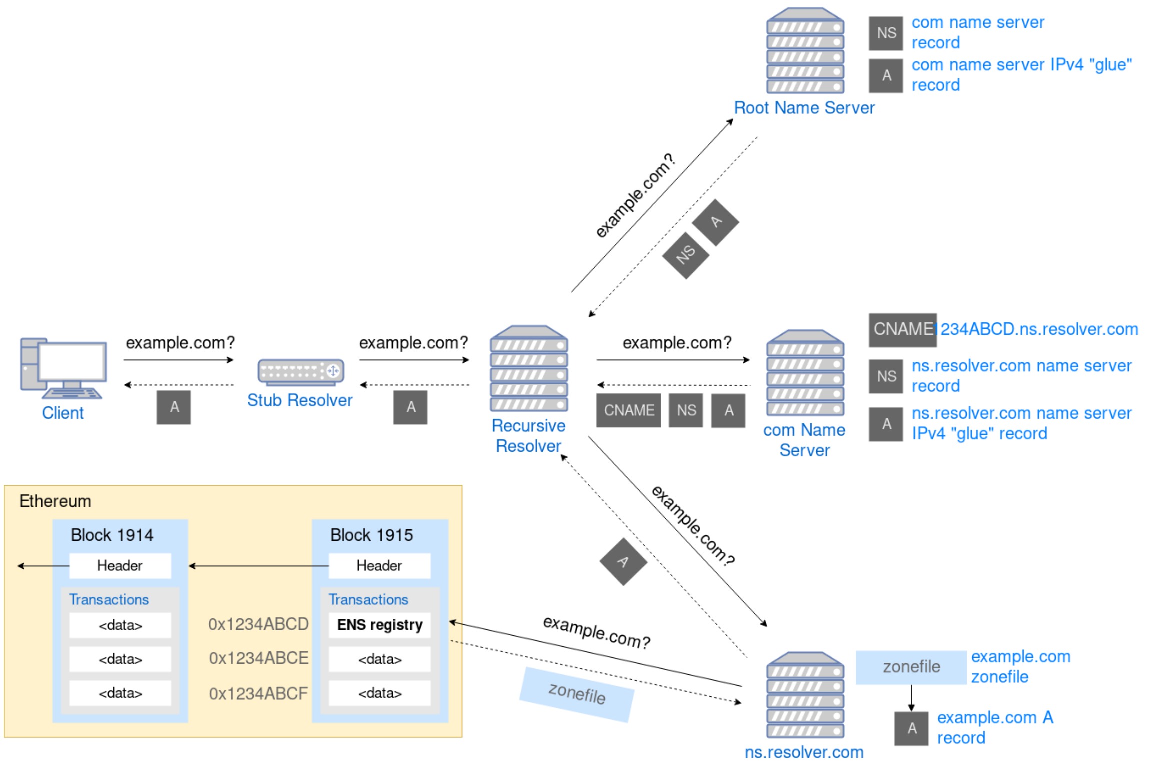


Fig. 3.2: ENS to DNS Lookup

Suppose an ENS resolver was at the DNS address `ns.resolver.com`, and an ENS registry[4] was at the ethereum address `0x1234ABCD`. A client begins a lookup for `example.com`, which is configured as a CNAME for `0x1234ABCD.ns.resolver.com`. A resolution would look like this [36]:

1. The recursive resolver performs a normal resolution up until the `com` name server, which tells it `example.com` is a CNAME for `1234ABCD.ns.resolver.com`, `ns.resolver.com` is a name server, and the A record for `ns.resolver.com`
2. The recursive resolver requests the A RR for `example.com` at `ns.resolver.com`

[4]A database of all domain names and associated information

3. The ENS resolver repeats the lookup for `example.com`, eventually getting the same result as the resolver and recognises the ethereum address `1234ABCD`
4. The ENS resolver queries the ENS registrar at the ethereum address `1234ABCD` for the `example.com` zone file
5. The ENS resolver extracts the A RR and forwards it back to the recursive resolver
6. The recursive resolver forwards the response back to the client

3.3.1 Discussion

ENS has a very convoluted lookup process, requiring the ENS resolver to repeat the lookup in order to get the address of the ENS registry. This at least doubles the number of lookups performed.

ENS provides integrity to a zone file from a given ENS registry, as the immutable state of a blockchain prevents tampering. However, it suffers from similar issues to DNSCurve, in which the registry address is encoded in the name of the name server. This means that a name server may not be who it claims to be, as origin authentication is not provided. A rogue name server could encode an alternative ENS registry address and spoof the zone files for the domain being requested. ENS is fully compatible with DNSSEC though, which would help this issue.

3.4 Namecoin

Namecoin is a blockchain application which contains its own cryptocurrency and allows users to create arbitrary key-value pairs [37]. It contains several predefined namespaces with a variety of use cases, such as the **d/** namespace for domain names in the `.bit` TLD [37].

A domain name is registered in a two-step process. First, it is pre-ordered within a Namecoin transaction in order to prevent a race where other users attempt to steal the name whilst the transaction is being confirmed. After a wait period the domain name registration can be finalised [37].

Since these domain names are not registered within the typical DNS hierarchy, other methods of retrieving their IP address must be used.

Web Proxy	Requires no set up, as you simply tunnel through another server which will perform the resolution for you. However, this can be insecure, as all data is tunnelling through a remote server [38].
Browser Plugin	Very easy to set up, and can either be used with a public or local DNS server for resolutions [38].
Public DNS Server	Requires minimal set up. Once again, this requires resolution through a remote server, and so can be insecure [38].
Local DNS Server	Traffic remains local and private using this method making it the most secure of the options. It requires downloading the entire state of the blockchain and keeping it readily updated, which takes up a lot of disk space [38].

3.4.1 Discussion

Namecoin offers a secure way of retrieving DNS information. With a local DNS server it provides origin authentication, message integrity, confidentiality and availability, as all requests are being handled from your local copy of the blockchain. If a remote server is used and trusted, origin authentication and message integrity are provided. However, a number of issues do exist which are outlined below.

There is no authority over what name a user can buy, so a lot of large organisations would be unable to get their name as there is no way to handle disputes. This means well known and commonly used websites, such as `google` and `facebook`, would have to register a different name under Namecoin.

A mobile device would also be limited in its options for resolution solutions which involve downloading the entire blockchain, as disk space is much more limited. Some of the more insecure options would have to be used. This leaves a client open to responses being spoofed.

The most important factor in blockchain security is the mining power. If a single party operates the majority of the mining power, then the history of the blockchain can be rewritten. It was discovered in 2014 that Namecoin consistently had a single party which controlled over 51% of the network (up to 75%) [39].

The less nodes a blockchain has, the more susceptible it is to a denial of service attack. As of the 4th of April 2018, bitcoin had 132 times as many nodes as Namecoin [40].

3.5 Blockstack

Blockstack is a decentralised PKI system, built on top of bitcoin [39]. It attempts to make its implementation agnostic to the underlying blockchain by using a series of layers [39]:

Blockchain Layer	This is part of the control plane. Transactions store an encoding of Blockstack instructions. It also acts as a consensus mechanism.
Virtualchain Layer	This is part of the control plane. New operations are defined here, abstracted away from the blockchain layer. These operations are encoded as metadata in the lower blockchain layer.
Routing Layer	Routes operations processed from the lower layers to the storage layer. The routing information is defined in "zonefiles", identical to DNS zonefiles. A hash of these files is stored in the control plane to validate their integrity.
Storage Layer	The data for the PKI is stored here. Storing the data outside of the blockchain allows for a variety of storage mediums and scalability. The integrity of this information can be validated in the control plane.

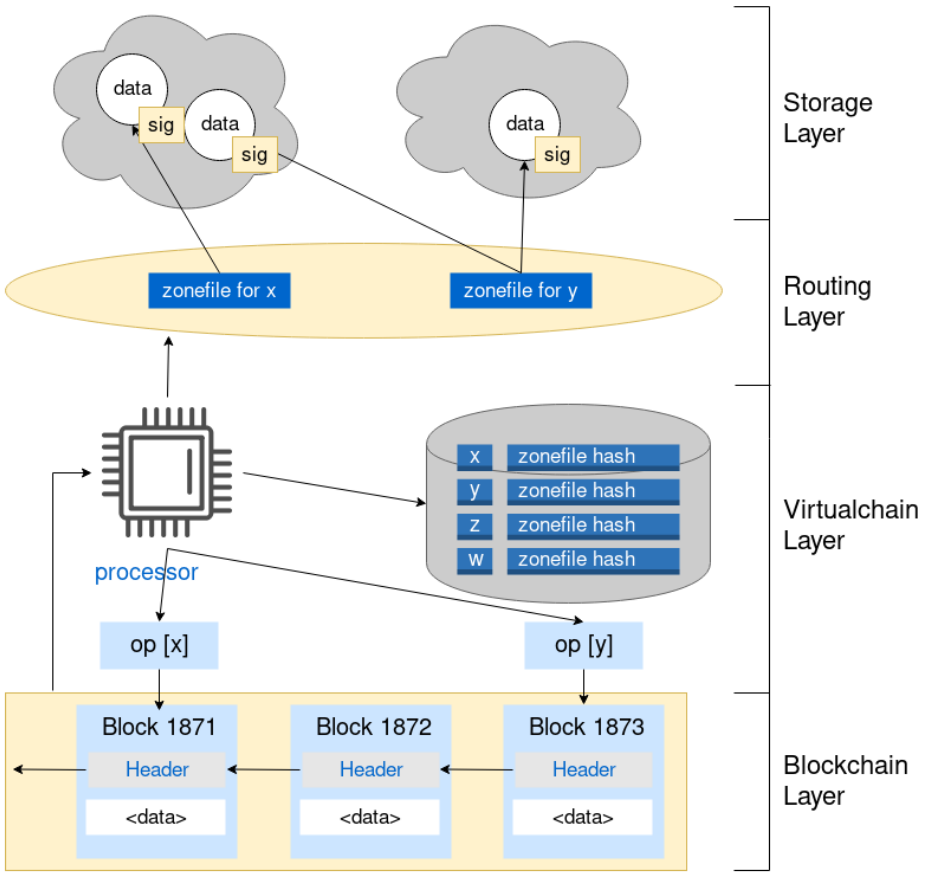


Fig. 3.3: Blockstack's Architecture

These tiers together form a naming system, using a similar protocol to Namecoin where pre-ordering and registering is a two step process [39].

3.5.1 Discussion

Blockstack operates a complete naming system where namespaces can be registered. Like Namecoin, they operate a DNS service, and also a browser to resolve the registered domain names [41]. Due to it being agnostic to any particular blockchain, it is able to move around to whichever is the most secure at the time. This makes a lot of its security dependent on whatever blockchain it is using. As bitcoin is currently the most widely used blockchain it is also the most secure [40].

Like Namecoin, any user can buy any domain name as there is no dispute authority. This means organisations will have difficulty registering their trademarked and copyrighted names, which would lead to little adoption of this application as a large scale solution for DNS security.

3.6 A Distributed X.509 Public Key Infrastructure Backed by a Blockchain

Bastian Fredriksson's thesis, A Distributed X.509 Public Key Infrastructure Backed by a Blockchain, discusses a theoretical method of improving identity retention in the current X.509 infrastructure [42]. The belief behind a blockchain being an appropriate method of backing certificates came down to the following properties of blockchains and peer-to-peer networks [42]:

- Immutability
- Consensus Protocol
- Mining Incentive
- Transparency
- Scripting Capabilities
- Availability

When an entity creates a CSR for a certificate, and once the CA has approved the request, the certificate is sent back to the customer. In this report, the CA would also create an "account" on the blockchain with the parameters specified in the certificate. This account is a container for a blockchain entity, acting as a trust anchor for the PKI. When the certificate is renewed, the account is updated [42].

3.6.1 Discussion

The report investigates a theoretical way of turning an existing PKI into a distributed version. The benefits of this, such as the ease of certificate revocation and the ability for an entity to attest to the validity of their certificate, are discussed in the thesis [42].

The solution allows for a way of getting certificate parameters from a secure trust store which can then be used to validate an organisation and provide secure communications. It provides origin authentication, integrity and confidentiality. Due to the nature of a peer-to-peer network, its availability relies on how many nodes exist in the network.

However, this is an extremely convoluted method of backing the X509 PKI. Solutions such as DANE do this in a much simpler manner, and grant a domain owner more control.

3.7 X509 Cloud

X509Cloud is a framework which enables organisations to freely issue certificates to individuals within their organisation, in order to provide authentication and secure communications for these users [43].

The organisation operates as a subordinate CA, and only signs certificates for users which will be operating under that organisation's network or domain. For example, if Alice wanted a certificate as a student at Trinity College, she could request a certificate that can be used on Trinity's network, which would be signed by Trinity. She would not be able to use this certificate to authenticate at another college, as it would not be signed by that other college.

The certificate is not only sent back to the individual, but it is also broadcasted on the X509Cloud blockchain network [43]. Each CA participating in the system will verify the certificate is valid. Once the transaction is validated it is grouped with other transactions into a block which is also be broadcasted. If the block is accepted, the next bundle of transactions is validated.

As a blockchain is an append only ledger, the most recent certificate for an entity is seen as the valid version. This allows the blockchain to also operate as a CRL [43].[5]

[5]A list of certificates that have been revoked before their expiration date and should not be trusted [12]

3.7.1 Discussion

X509Cloud offers a very simple solution which solves several issues. It allows organisations to distribute certificates to those within it in a convenient and secure way. It also solves the issues associated with CRLs, where it can take time for the changes to propagate through the network. Lastly, it creates an always up to date data store of the current X509 certificates that are in use by those participating in the system.

As a certificate can be securely received and trusted, this system provides origin authentication, integrity and confidentiality. Similar to Bastian Fredriksson's thesis, its availability is dependent on the number of nodes active in the network.

System Design

4

This chapter aims to provide a solution to the security issues that DNS suffers from whilst maintaining the goals described in section 1. Technologies outlined in section 2 are built upon and combined with ideas and research discussed in section 3 to form the inspiration behind the design of the system.

The aim is to create a DNS security extension, which does not suffer from common issues which arise in DNSSEC deployment. Various iterations of the design will be discussed, as well as an in depth assessment of a permission based blockchain application.

4.1 MultiChain

A number of blockchains exist, from pioneers such as bitcoin to more general purpose implementations. Systems that implement a naming system or PKI have already been discussed in sections 3.4 and 3.5, and are not appropriate as there is no authority on who gets to own a particular name.

MultiChain is a general purpose blockchain that solves problems of mining, privacy and openness with permissions [44].

4.1.1 Permissions

A number of permissions exist in MultiChain [45]:

`connect`	connect to other nodes
`send`	sign inputs to transactions
`receive`	appear in outputs of transactions
`issue`	create new native assets
`create`	create new streams
`mine`	mine blocks
`activate`	alter `connect`, `send` and `receive` permissions for other users
`admin`	change all permissions for other users

General permissions can also be set with MultiChain. For example, `anyone-can-connect` might be enabled, allowing anyone to view the blockchain [45].

4.1.2 Mining Diversity

In a traditional blockchain system a proof-of-work algorithm is used in order to stop miners dominating the miner pool, preventing double spend attacks.

Since MultiChain can be configured as private, issues such as these can be resolved. By enforcing mining diversity, a miner is constrained in the number of blocks they can add within a given window [44].

The `mining-diversity` parameter is set between 0 and 1. A value of 0 means anyone can mine at a given stage, whilst a value of 1 means every miner must contribute in a given cycle of all the miners [46]. A value of 0.75 is suggested as a reasonable compromise, meaning 75% of miners must participate [44].

A mining difficulty can be specified [46], although in a private blockchain with mining diversity it is simply a formality and does not add any security [44].

4.1.3 Data Streams

Data streams in MultiChain allow it to be used as a general purpose, append only database. Any number of streams can be created, and each contain an ordered list of items [47].

Each item must be signed by a publisher, contain a key, some data, and miscellaneous information about the transaction and block (such as timestamp, transaction ID, etc.) [47]. This makes it suitable as a key-value or time series database.

4.1.4 Advantages

There are a number of reasons a private blockchain is beneficial over a public blockchain. These include [44][48]:

- Only selected participants are able to write to the blockchain. This makes a sybil attack less likely.
- Mining is only a formality in a private blockchain as it is not necessary from a security perspective. This reduces economic costs from computing power.

- A lot of large scale blockchains such as bitcoin require large transaction fees to get a transaction included in a block. Transaction fees are not necessary in a closed system as there is no associated cost from mining.
- The blockchain is not bloated with irrelevant data. This makes for faster transaction look-ups and less disk space is used storing the blockchain.

This begs the question if a traditional centralised database would be better, as they have significantly better performance and allows for confidentiality [49]. A private blockchain is beneficial because [44][49]:

- Each participant has full control over their assets with their private key, as each transaction performed by a participant is signed. This means information can be shared over multiple boundaries of trust.
- Control is distributed across multiple parties. This decentralisation means a trusted mediator is not necessary, which is what would be needed for a centralised solution.
- Fault tolerance and robustness. If a single node goes down, it is not crucial to the overall operation.

4.2 CSK

A Combined Signing Key (CSK) is what will be used to refer to a DNSSEC configuration which uses a single key. This CSK is simply a KSK which also operates as a ZSK, i.e. the DNSKEY RRset is signed by the same key which signs all other RRsets in that zone.

DNS software for authoritative servers can enable this configuration. A flag is simply used to indicate the KSK signs all RRsets [50].

4.3 Protocol

For each version of the protocol, the recursive resolver is intended to be a node in the blockchain network. When looking up transactions in the blockchain, it is querying its local copy of the blockchain.

To reiterate the goals which will be kept in mind when discussing each versions of the protocol:

- Has the client been provided origin authentication?
- Has the client been provided message integrity?

- Do responses suffer from IP fragmentation?
- Are the number of requests reduced?
- Is the key management simpler?

4.3.1 Version 1

The first version of the protocol involved removing DNSKEY RRs from the DNS hierarchy. Instead, a single key would be introduced, known as a CSK. This would be placed on the blockchain as a key-value pair, where the key is the domain name and the value is the CSK.

Once DNSKEY entries have been removed from the DNS hierarchy there are no DNSKEY RRsets to be signed. As the DNSKEY RRSIG no longer exists there is no need to create a DS RR to be placed in the parent zone, as this is used to create a trust chain of KSKs up to the root server [51].

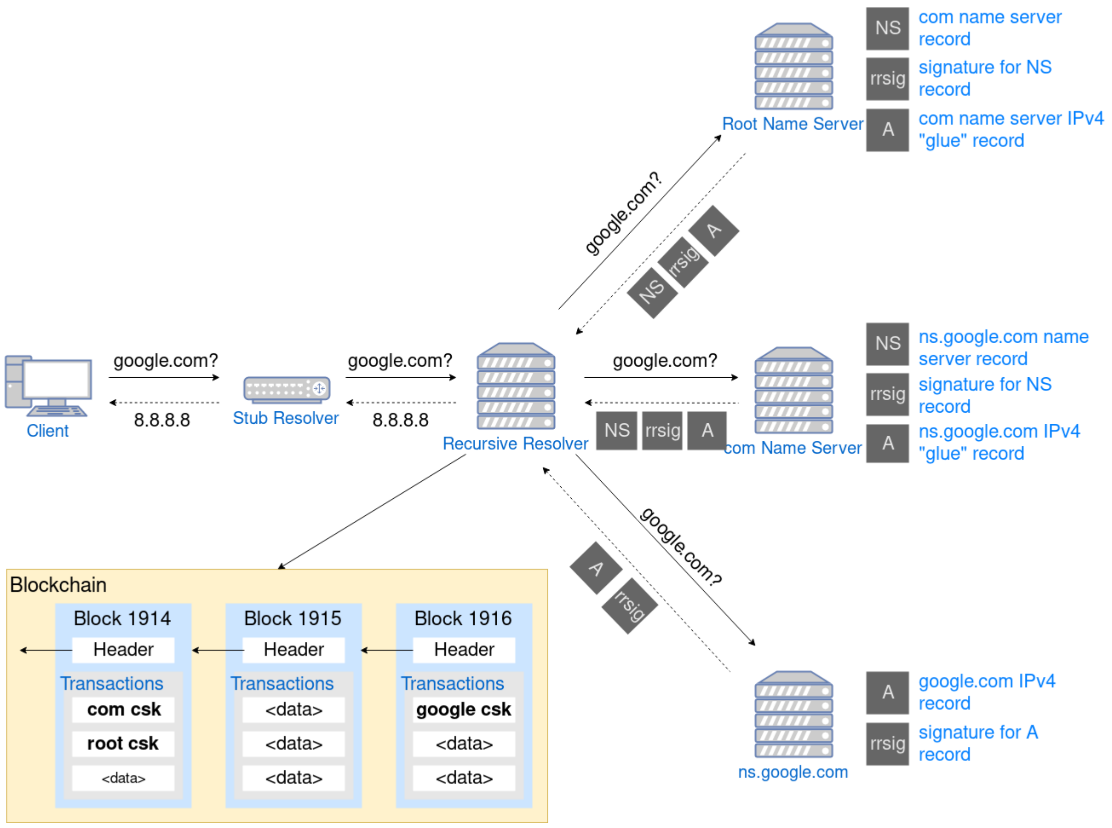


Fig. 4.1: First Design

A lookup would operate as follows:

1. A client initiates a normal lookup for the domain `google.com`, which gets forwarded to a recursive resolver

2. In the case of a cache miss, the resolver will query one of the 13 root name servers for `google.com`
3. The root name server will return the NS RR for `com`, a "glue", and the signatures to validate
4. The recursive resolver will search for the most recent entry of the root CSK in the blockchain, and validate the signatures
5. The recursive resolver will query the `com` name server for `google.com`
6. The `com` name server will respond with the NS RR for `google.com`, a "glue", and the signatures to validate
7. The recursive resolver will search for the most recent entry of the `com` CSK in the blockchain, and validate the signatures
8. The recursive resolver will query the `google.com` name server for the `google.com` A RR
9. The `google.com` name server will respond with the A RR for `google.com` and a signature
10. The recursive resolver will search for the most recent entry of the `google.com` CSK in the blockchain, and validate the signature
11. The verified A RR will be forwarded back to the client

This protocol provides us with origin authentication and message integrity. The number of requests are also reduced. However, the protocol suffers from a number of issues:

- To validate that an entry on the blockchain belongs to an entity, we need a way of getting their public key that they use for their blockchain transactions. This would either involve looking for a certificate, or having it stored somewhere in the DNS hierarchy, which will then need securing. This does not simplify key management.
- It is unclear if IP fragmentation still occurs, as signatures are still returned.

4.3.2 Version 2

The second version of the protocol took a different approach, which removes public keys entirely. Without public keys, DNSKEY RRs do not exist, and DS RRs are not necessary [51]. If there are no keys, then there is no way to sign RRs, which removes RRSIG RRs.

Instead, the hash of an RR would be stored on the blockchain, which is placed there by domain owners. We also introduce a new "BC" RR, which contains the transaction address of the hashed RR being requested. When a request is made, the "BC" RR is sent alongside other RRs.

A lookup looks as follows:

1. A client initiates a normal lookup for the domain `google.com`, which gets forwarded to a recursive resolver
2. In the case of a cache miss, the resolver will query one of the 13 root name servers for `google.com`
3. The root name server will return the NS RR for `com`, a “glue”, and the “BC” RR
4. The recursive resolver will search for transaction address stored in the “BC” RR and validate the NS and A RRs by hashing them and comparing the result with the entry in the blockchain
5. The recursive resolver will query the `com` name server for `google.com`
6. The `com` name server will respond with the NS RR for `google.com`, a “glue”, and the “BC” RR
7. The recursive resolver will search for transaction address stored in the “BC” RR and validate the NS and A RRs by hashing them and comparing the result with the entry in the blockchain
8. The recursive resolver will query the `google.com` name server for the `google.com` A RR
9. The `google.com` name server will respond with the A RR for `google.com` and the “BC” RR
10. The recursive resolver will search for the transaction address stored in the “BC” RR and validate the NS and A RRs by hashing them and comparing the result with the entry in the blockchain
11. The verified A RR will be forwarded back to the client

The protocol provides us with message integrity. The key management is nonexistent, so it is objectively simpler. Messages do not contain signatures, but simply “BC” RRs, and are unlikely to suffer from IP fragmentation. The number of requests are also reduced.

However, this protocol does not grant us origin authentication and without this we cannot guarantee that a record comes from who it says it does. This makes the protocol susceptible to spoofing from a man in the middle attack.

There is also a question of how scalable the blockchain would be in this solution. If every entry from every domain is stored, then this would amount to a lot of information. This would cause the blockchain to become very large, which means it would consume a lot of disk space.

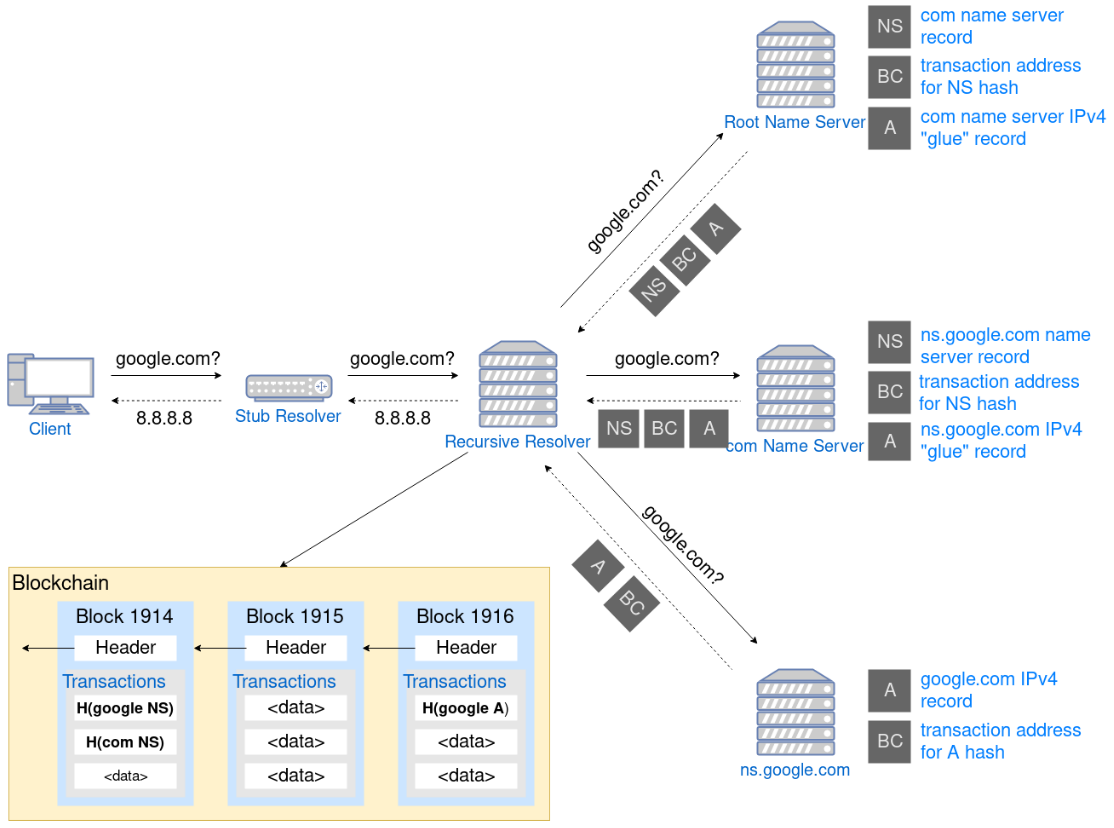


Fig. 4.2: Second Design

Not only would the current state of the DNS hierarchy be stored, but due to the fact the blockchain is immutable every old RR would also be stored. This is a significant security flaw. For example, an attacker could get hold of an old IP address that was used for `google.com`, and spoof the A RR and "BC" RR, directing a client to an incorrect website.

4.3.3 Version 3

For the third version of the protocol, the first version and its associated issues were revisited. Namely, how can we associate a key with an organisation in a way that is trusted?

If the CSK is replaced with a digital certificate that contained the aforementioned key, we can link an identity with the key [10]. A lookup now operates as follows:

1. A client initiates a normal lookup for the domain `google.com`, which gets forwarded to a recursive resolver
2. In the case of a cache miss, the resolver will query one of the 13 root name servers for `google.com`
3. The root name server will return the NS RR for `com`, a "glue", and the signatures to validate

4. The recursive resolver will search for the most recent entry of the root certificate in the blockchain, check the validity of the certificate, and validate the signatures
5. The recursive resolver will query the `com` name server for `google.com`
6. The `com` name server will respond with the NS RR for `google.com`, a “glue”, and the signatures to validate
7. The recursive resolver will search for the most recent entry of the `com` certificate in the blockchain, check the validity of the certificate, and validate the signatures
8. The recursive resolver will query the `google.com` name server for the `google.com` A RR
9. The `google.com` name server will respond with the A RR for `google.com` and a signature
10. The recursive resolver will search for the most recent entry of the `google.com` certificate in the blockchain, check the validity of the certificate, and validate the signature
11. The verified A RR will be forwarded back to the client

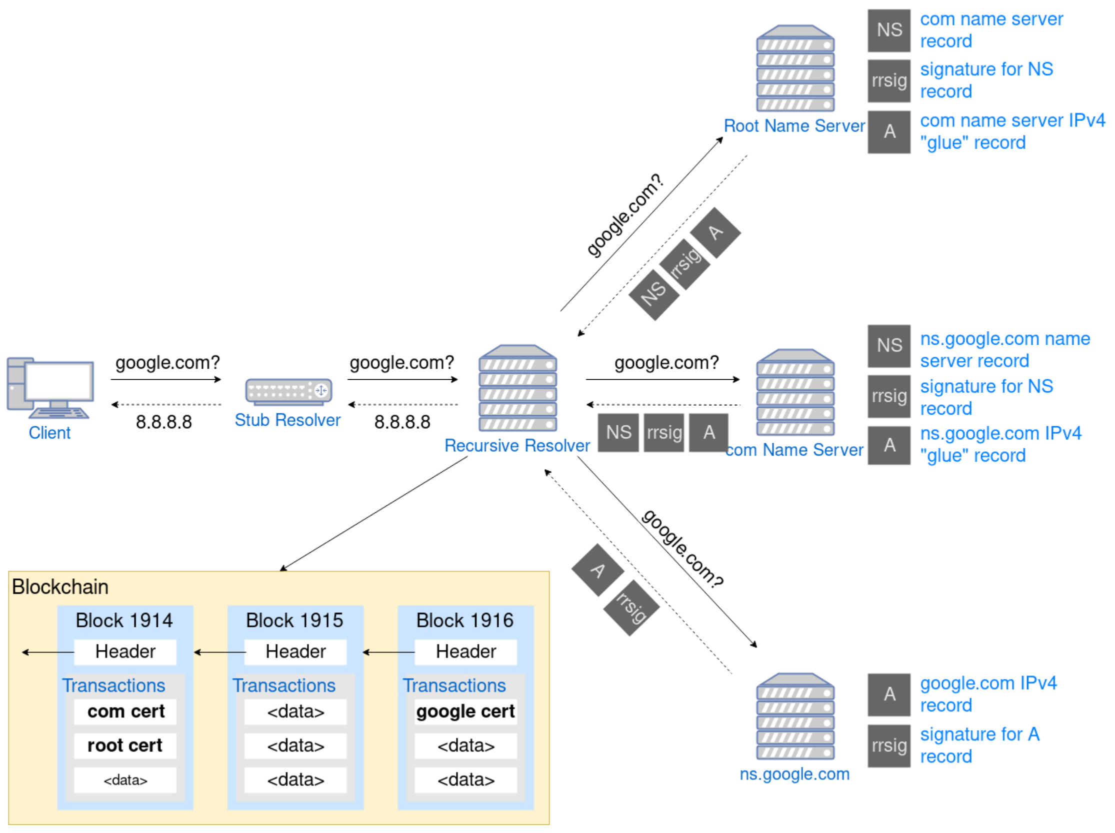


Fig. 4.3: Final Design

It is important to note that at any given stage in the resolution, the blockchain acts as a secure and trusted store. The root KSK is no longer used as a trust anchor, so if the A RR for a TLD name server was cached the resolution could securely start from there. Hence, a look-up where we known the IP address for the `com` name server would look as follows:

1. A client initiates a normal lookup for the domain `google.com`, which gets forwarded to a recursive resolver
2. The resolver checks its cache and finds the A RR for the `com` name server and queries this for `google.com`
3. The `com` name server will respond with the NS RR for `google.com`, a "glue", and the signatures to validate
4. The recursive resolver will search for the most recent entry of the `com` certificate in the blockchain, check the validity of the certificate, and validate the signatures
5. The recursive resolver will query the `google.com` name server for the `google.com` A RR
6. The `google.com` name server will respond with the A RR for `google.com` and a signature
7. The recursive resolver will search for the most recent entry of the `google.com` certificate in the blockchain, check the validity of the certificate, and validate the signature
8. The verified A RR will be forwarded back to the client

Once again, this could simply begin at the `google.com` name server, and shorten the resolution even further. This speeds up the verification process considerably, as each level does not need to be validated.

This protocol provides us with origin authentication and message integrity. The number of requests are also reduced. We reduce key management to a single certificate rather than multiple keys, making it simpler.

However, the question on RRSIGs suffering from IP fragmentation still exists. This will be studied and revisited later in section 6.2.

4.3.4 MultiChain Configuration

Since a certificate needs to be signed by a CA, the blockchain setup is not as simple as allowing domain owners to place certificates on the blockchain. Therefore, the following configuration would be used:

- Root CAs have `admin`, `mine` and `create` permissions
- CAs have `mine` and `create` permissions
- Anyone can `connect`

Adding a certificate to the blockchain begins with MultiChain streams. Each domain would have a stream associated with it, which stores the history of all its certificates. At any given point, the most recent certificate is the valid one. This doubles as a CRL.

An organisation creates a CSR as normal, containing information including their public key which will be the public component of their CSK. This gets sent to a Registration Authority. If it is approved a certificate signed by the CA will be created and a copy will be sent back to the organisation, which can be used for other purposes such as TLS. The CA will also create a transaction to add it to that organisation's stream on the blockchain. This process is illustrated in Fig 4.4.

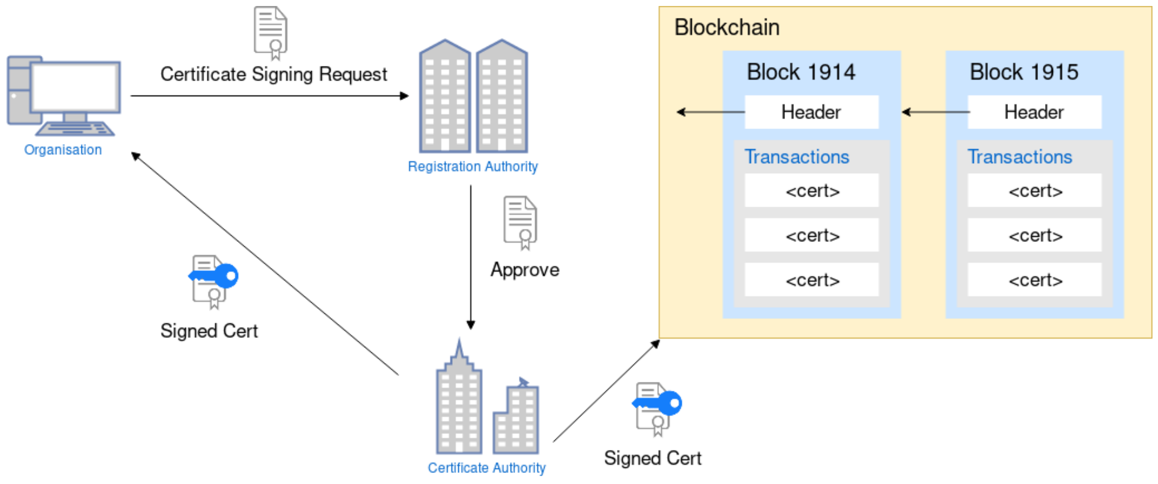


Fig. 4.4: Adding a certificate to the blockchain

Using MultiChain, a node or set of nodes will be allowed to mine blocks at a given time. The transaction will be bundled into a block and broadcasted to the entire network by one of these nodes. As this is a private blockchain, proof of work is not needed. Other nodes will go through the transactions ensuring that the certificates are valid.

The block is accepted if the other nodes begin working on the next block. If instead the block is seen as incorrect then it would be ignored and replaced by the next block instead.

This configuration is similar to the system described in section 3.7. The protocol in section 4.3.3 is designed so that the PKI being used is agnostic to the resolution as a whole. If a better solution came about which did not use a blockchain or certificates, this could be used instead. Currently, as discussed in section 4.1.4, a private blockchain has a significant number of advantages over both public blockchains and traditional centralised databases which suits the needs of our system. The system described in section 3.7 is also the most elegant PKI which uses a blockchain.

Implementation 5

This chapter describes how each component from section 4 was set up and developed.

5.1 Authoritative Server

BIND is a DNS software suite that implements a domain name resolver, domain name authoritative server, and administration tools [52]. BIND is considered the Unix de facto standard, servicing over 70% of domains [53].

For this project BIND was set up on a local machine and configured with a fake DNS hierarchy set up. DNSSEC was enabled for each zone. This hierarchy consists of a root and a TLD called `scarlett` which contains two subdomains. These are `example.scarlett` and `google.scarlett`, each with their own respective name servers. This set up is depicted in Fig 5.1.

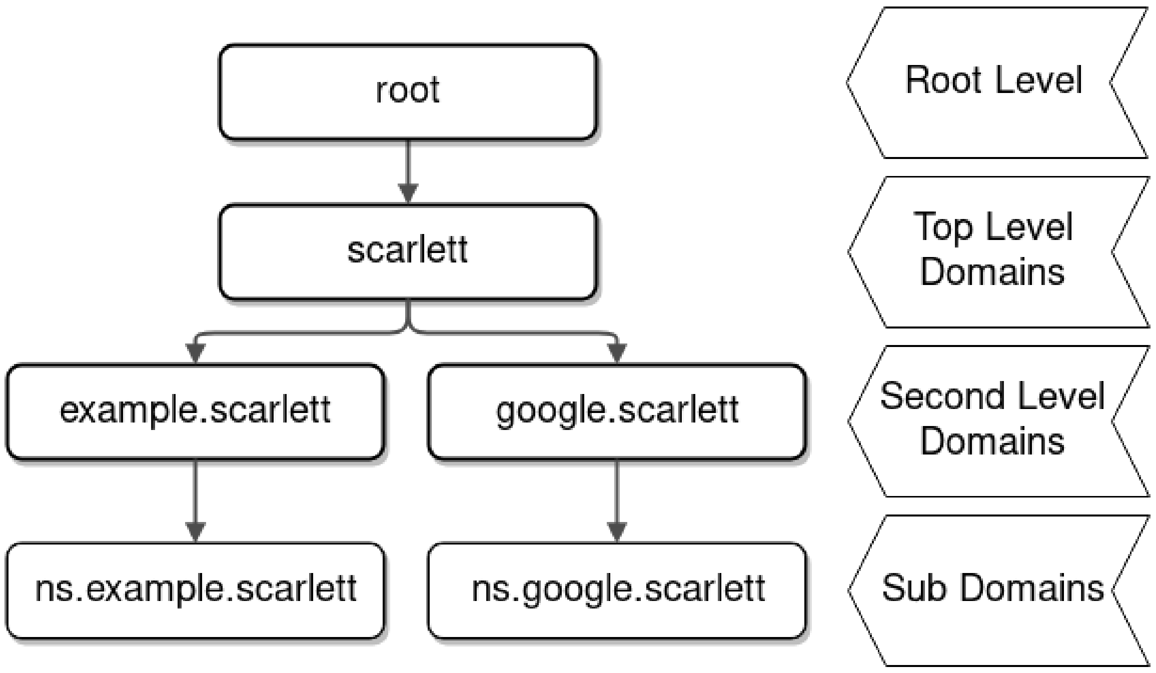


Fig. 5.1: Local DNS Hierarchy

The root zone had a KSK stored on the machine to act as a “trust anchor” when performing traditional DNSSEC requests. All zones had both a KSK and a ZSK, except for `example.scarlett` which used a single CSK. This is to demonstrate validation with our designed protocol.

5.2 Certificate Database

MultiChain uses LebelDB for its local database, which stores the state of each block and its location on disk [54]. Instead of bootstrapping an entire blockchain and setting up a series of IP addresses to be CAs and root CAs, a simple mySQL database was used instead. This was intended to mimic the principle of MultiChain’s local database.

The database stored an X.509 certificate in PEM format, with a key indicating which domain it belonged to. It was also set up to distinguish ZSKs and KSKs, although in practice only a CSK would be used so this distinction would not be necessary.

5.3 Resolver

A resolver was written using LDNS, which is a C library designed to make programming DNS applications simpler [55].

The resolver is able to perform look-ups for a variety of records and validate their RRSIGs by extracting public keys from the certificates stored in the database. It is also possible to perform traditional DNSSEC validation, although this was only used for testing.

5.4 Samples

A number of examples using the resolver and other DNS administration tools demonstrate various aspects of the protocol and its comparison to traditional DNSSEC.

5.4.1 Using a Certificate for Verification

It is firstly important to show that it is possible to use a certificate to validate these RRSIG RRs. For the `google.scarlett` domain, a ZSK and KSK was set up, and an RRSIG was generated.

The following is the response to a query for the `google.scarlett` DNSKEY RRs using the Unix DNS debugging tool `dig`:

```
$ dig @localhost google.scarlett. DNSKEY +dnssec

; <<>> DiG 9.13.0-dev <<>> @localhost google.scarlett. DNSKEY +dnssec
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1422
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: dbaca0f6b919e8fa20b763ec5adc74df67e71185f26617d9 (good)
;; QUESTION SECTION:
;google.scarlett.              IN      DNSKEY

;; ANSWER SECTION:
google.scarlett.        7200    IN      DNSKEY  256 3 13 Hwh+
    iU3jsicGF2rOyEIrjSm8B2IBl0ogn8d/mNa+bJ4pZmAjl8mluni5 QjsB5yyAKv/aj9vIalABGv/
    V2uCQvQ==
google.scarlett.        7200    IN      DNSKEY  257 3 13 2mUuN22CH+
    XezUTkgudoHWxkXO3rCKOzNT5TraLwLrhfGFxCP/SDr0wg ny0ZkHpjvSQZSFdLoHe/i3qxJUQ+
    QQ==
google.scarlett.        7200    IN      RRSIG   DNSKEY 13 2 7200 20180515144101
     20180415144101 50590 google.scarlett.
    Ih8tkQJDyEewpGll2uR5tDpOm3EgXQMn4M53tV090Ox9K+0dzGafjhge
    qbvUYZS7otWwc9kYBA8xeLRRW1L0Ug==

;; Query time: 0 msec
;; SERVER: ::1#53(::1)
;; WHEN: Sun Apr 22 12:41:19 IST 2018
;; MSG SIZE  rcvd: 343
```

The following is the output of the resolver, which is validating the DNS RRs using the designed protocol:

```
$ ./bin/main @localhost -v 2 -t DNSKEY -val-RR google.scarlett.

Querying for: google.scarlett.

------------------------------
Verifying Resource Record
------------------------------
RRSIGs of type DNSKEY to validate: 1

DNSKEY string: google.scarlett. IN DNSKEY 256 3 13 Hwh+
    iU3jsicGF2rOyEIrjSm8B2IBl0ogn8d/mNa+bJ4pZmAjl8mluni5QjsB5yyAKv/aj9vIalABGv/
    V2uCQvQ==


13 DNSKEY string: google.scarlett. IN DNSKEY 257 3 13 2mUuN22CH+
     XezUTkgudoHWxkXO3rCKOzNT5TraLwLrhfGFxCP/SDr0wgny0ZkHpjvSQZSFdLoHe/i3qxJUQ+QQ
     ==
14
15
16 Trying to verify with zsk...failure
17 Trying to verify with ksk...success
18 Verification result of DNSKEY RRSIG for google.scarlett.: All OK
```

The resolver begins with a query for the DNSKEY RRs for `google.scarlett.`[6] Line 8 shows there is only a single RRSIG to validate, which is the signature of the DNSKEY RRset.

The validation process begins next.[7] The certificate containing the public key for the ZSK[8] (Line 10) and KSK[9] (Line 13) are extracted and formed into a DNSKEY string. These DNSKEY strings can be compared with the output from `dig`, which shows that they are identical.

The RRSIG is first tested with the ZSK, which fails at line 16. This is expected, as DNSKEY RRsets are signed with the KSK. The KSK attempts to verify the RRSIG next, which succeeds at line 17.

Since these certificates are coming from a trusted state of the blockchain placed there by the certificate authorities, these keys are trusted. Hence the validation process ends at line 18.

DNSKEY RRs are not necessary for this protocol, so DNSKEY validation in particular is not relevant. This is simply to show a certificate being transformed into a DNSKEY and this being used for basic validation.

5.4.2 Name Server Records and Glue

NS and “glue” A RRs are vital to the DNS. When performing a DNS look-up, the resolver is directed to a number of name servers where zones are delegated. It is important that these NS and “glue” RRs can be validated. The following is the validation process of the NS and “glue” RRs for `google.scarlett`:

```
$ ./bin/main @localhost -v 2 -t NS -val-RR -t A google.scarlett.

Querying for: google.scarlett.

```

[6]Indicated by `-t DNSKEY` flag
[7]Indicated by the `-val-RR` flag
[8]Indicated by the `256` [19]
[9]Indicated by the `257` [19]

```
5  ------------------------------------
6  Verifying Resource Record
7  ------------------------------------
8  RRSIGs of type NS to validate: 1
9
10 DNSKEY string: google.scarlett. IN DNSKEY 256 3 13 Hwh+iU3jsicGF2rOyEIrjSm8B2IBl0ogn8d/mNa+bJ4pZmAjl8mluni5QjsB5yyAKv/aj9vIalABGv/V2uCQvQ==
11
12
13 DNSKEY string: google.scarlett. IN DNSKEY 257 3 13 2mUuN22CH+XezUTkgudoHWxkXO3rCKOzNT5TraLwLrhfGFxCP/SDr0wgny0ZkHpjvSQZSFdLoHe/i3qxJUQ+QQ==
14
15
16 Trying to verify with zsk...success
17 Verification result of NS RRSIG for google.scarlett.: All OK
18
19
20 ------------------------------------
21 Verifying Resource Record
22 ------------------------------------
23 RRSIGs of type A to validate: 1
24
25 DNSKEY string: google.scarlett. IN DNSKEY 256 3 13 Hwh+iU3jsicGF2rOyEIrjSm8B2IBl0ogn8d/mNa+bJ4pZmAjl8mluni5QjsB5yyAKv/aj9vIalABGv/V2uCQvQ==
26
27
28 DNSKEY string: google.scarlett. IN DNSKEY 257 3 13 2mUuN22CH+XezUTkgudoHWxkXO3rCKOzNT5TraLwLrhfGFxCP/SDr0wgny0ZkHpjvSQZSFdLoHe/i3qxJUQ+QQ==
29
30
31 Trying to verify with zsk...success
32 Verification result of A RRSIG for google.scarlett.: All OK
```

First, a query for `google.scarlett` is made, requesting a NS RR.[10] In the response, two RRSIGs will be present - one for the NS RRset (Line 8), and another for an A RRset (Line 23). The resolver is set to validate both of these RRSIGs.[11]

For the NS validation, the two certificates for `google.scarlett` are converted to DNSKEY strings at lines 10-13. The ZSK certificate succeeds in verifying the NS RR at line 16. The resolver moves on to the A validation, and once again the ZSK certificate succeeds between lines 23-31. This is as expected, and the next request can be securely made.

[10]Indicated by `-t NS` flag

[11]Indicated by the `-val-RR -t A` flag

The `example.scarlett` domain only has a single certificate. The same request is made on this domain below:

```
1  $ ./bin/main @localhost -v 2 -t NS -val-RR -t A example.scarlett.
2
3  Querying for: example.scarlett.
4
5  ----------------------------------------
6  Verifying Resource Record
7  ----------------------------------------
8  RRSIGs of type NS to validate: 1
9  Failed to get certificate example.scarlett. ZSK from the database: 11
10
11 DNSKEY string: example.scarlett. IN DNSKEY 257 3 13
       tP2dIWHtAQXTDDSrUNM8EjBQDzLC3DJkKBqWAtd34+
       WpgxtN7KKSFimHCYsdHaNPuLjWEGR0pgWhG9yFL8unCQ==
12
13
14 Trying to verify with zsk...no zsk
15 Trying to verify with ksk...success
16 Verification result of RRSIG 0 for example.scarlett.: All OK
17
18
19 ----------------------------------------
20 Verifying Resource Record
21 ----------------------------------------
22 RRSIGs of type A to validate: 1
23 Failed to get certificate example.scarlett. ZSK from the database: 11
24
25 DNSKEY string: example.scarlett. IN DNSKEY 257 3 13
       tP2dIWHtAQXTDDSrUNM8EjBQDzLC3DJkKBqWAtd34+
       WpgxtN7KKSFimHCYsdHaNPuLjWEGR0pgWhG9yFL8unCQ==
26
27
28 Trying to verify with zsk...no zsk
29 Trying to verify with ksk...success
30 Verification result of RRSIG 0 for example.scarlett.: All OK
```

Once again a NS (Line 8) and “glue” (Line 22) RR is returned in the response.

To demonstrate that only a single key is in use, the output for the resolver shows a failure at line 9 in retrieving the ZSK for the `example.scarlett` zone. This is expected. A single certificate is used and creates a KSK (or CSK) key at line 11. The validation fails at line 14 for a ZSK as one does not exist, and subsequently passes with the KSK (or CSK) at line 15.

The same process to validate the A RRSIG is repeated at lines 22-29. Once again the ZSK does not exist (Line 25) and the validation process passes with a single key (Line 28-30).

This process can be repeated throughout the DNS resolution, quickly validating each zone without any more requests necessary than traditional DNS. Once again, in the case a zone was cached the keys for zones higher in the hierarchy would not be needed to verify a RR as the blockchain allows the certificates to be trusted at any level, i.e. a trust anchor is no longer necessary.

5.4.3 Traditional DNSSEC

To demonstrate the convoluted verification in traditional DNSSEC, the A RR validation process is shown below using a DNS administration tool called `delv`. We can see the series of responses necessary to get DNSKEY and DS RRs, which will also include all the RRSIGs for those particular RRsets. The verification continues back down to the `google.scarlett` zone.

```
$ delv @localhost google.scarlett. +vtrace
;; fetch: localhost.home/A
;; fetch: localhost.home/AAAA
;; fetch: localhost/A
;; fetch: localhost/AAAA
;; fetch: google.scarlett/A
;; validating google.scarlett/A: starting
;; validating google.scarlett/A: attempting positive response validation
;; fetch: google.scarlett/DNSKEY
;; validating google.scarlett/DNSKEY: starting
;; validating google.scarlett/DNSKEY: attempting positive response validation
;; fetch: google.scarlett/DS
;; validating google.scarlett/DS: starting
;; validating google.scarlett/DS: attempting positive response validation
;; fetch: scarlett/DNSKEY
;; validating scarlett/DNSKEY: starting
;; validating scarlett/DNSKEY: attempting positive response validation
;; fetch: scarlett/DS
;; validating scarlett/DS: starting
;; validating scarlett/DS: attempting positive response validation
;; fetch: ./DNSKEY
;; validating ./DNSKEY: starting
;; validating ./DNSKEY: attempting positive response validation
;; validating ./DNSKEY: verify rdataset (keyid=14200): success
;; validating ./DNSKEY: signed by trusted key; marking as secure
;; validating scarlett/DS: in fetch_callback_validator
;; validating scarlett/DS: keyset with trust secure
;; validating scarlett/DS: resuming validate
;; validating scarlett/DS: verify rdataset (keyid=14579): success
;; validating scarlett/DS: marking as secure, noqname proof not needed
;; validating scarlett/DNSKEY: in dsfetched
;; validating scarlett/DNSKEY: dsset with trust secure
;; validating scarlett/DNSKEY: verify rdataset (keyid=61128): success
;; validating scarlett/DNSKEY: marking as secure (DS)
;; validating google.scarlett/DS: in fetch_callback_validator
;; validating google.scarlett/DS: keyset with trust secure
;; validating google.scarlett/DS: resuming validate
;; validating google.scarlett/DS: verify rdataset (keyid=22479): success
;; validating google.scarlett/DS: marking as secure, noqname proof not needed
;; validating google.scarlett/DNSKEY: in dsfetched
;; validating google.scarlett/DNSKEY: dsset with trust secure
;; validating google.scarlett/DNSKEY: verify rdataset (keyid=50590): success
;; validating google.scarlett/DNSKEY: marking as secure (DS)
;; validating google.scarlett/A: in fetch_callback_validator
;; validating google.scarlett/A: keyset with trust secure
;; validating google.scarlett/A: resuming validate
;; validating google.scarlett/A: verify rdataset (keyid=20114): success
;; validating google.scarlett/A: marking as secure, noqname proof not needed
; fully validated
google.scarlett.        7200    IN      A       192.168.2.0
google.scarlett.        7200    IN      A       192.168.2.20
google.scarlett.        7200    IN      RRSIG   A 13 2 7200 20180515144101
    20180415144101 20114 google.scarlett.
    NwtRpH7DHJyNTw8rswkLXpa5pdeKNF0JwFZG7bhUlc5aCo1JNqYoin/n PCjYSpOAb+4Tg7A/
    HyUJkexpUmSeLg==
```

Evaluation

6

Returning back to section 4.3.3, the question of if this protocol still suffers from IP fragmentation is analysed in significant detail in this chapter. DNSSEC deployment statistics and certificate sizes are also examined.

Statistics on DNSSEC were gathered by studying the zonefile of the `net` zone, which consists of approximately 35 million entries and 15 million different second level domains. Responses from these second level domains that were DNSSEC enabled were inspected.

6.1 DNSSEC Enabled Domains

Around 1.63% of second level domains in the `net` zone are signed. Fig 6.1 shows the distribution of DNSSEC algorithms that are used by these. The algorithms are decsribed in section 2.4.2.

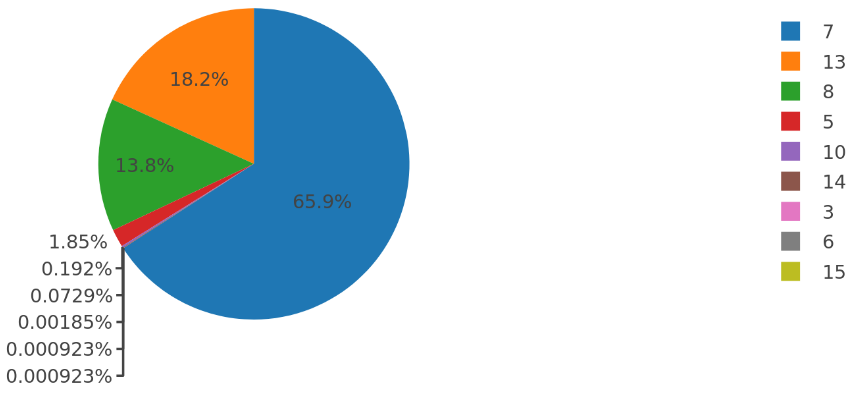


Fig. 6.1: Distribution of DNSSEC Algorithms

6.2 DNSSEC Response Sizes

Various requests for RRs and their respective signatures were made to the second level domains in the `net` zone. Response sizes and algorithm types were recorded.

A maximum packet size of 512 bytes is what was used in traditional DNS and would be an ideal response size [15].

6.2.1 DNSKEY

DNSKEY requests were expected to yield the largest response. Fig 6.2 shows a histogram of a variety of RSA algorithms.

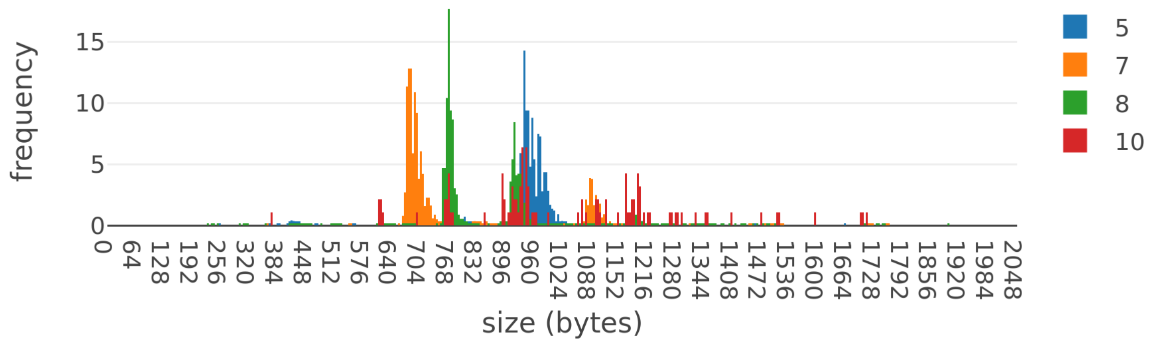


Fig. 6.2: DNSKEY RR Response Sizes from RSA Algorithms

Algorithm 7 yielded the smallest response size. This explains its prevalent use shown in Fig 6.1, as many zones will attempt to reduce the size of their DNSSEC responses in order to avoid the accessibility issues outlined in section 2.4.3. Algorithm 7 returns the smallest response as the signature uses a SHA-1 hash function which produces the smallest message digest. However, SHA-1 has been proven to cause colliding message digests, which can limit the security of a signature using this hash function [56].

Algorithm 8 produces the next smallest response size. This algorithm uses a SHA-256 hash, which is a trade-off between the resulting size of the message digest (and hence signature), and the security of the hash function.

Algorithm 10 uses a SHA-512 hash function which produces the largest signature. This would explain why many of the responses using this algorithm are very large.

It can be observed that a significant majority of these responses from all algorithms are above 512 bytes, which would not fit inside of a traditional DNS packet.

Fig 6.3 compares the most common RSA algorithm against the most common ECDSA algorithm, as determined from Fig 6.1. Each of these peaks show the various forms of DNSKEY configurations.

The first four peaks are all from zones which use algorithm 13. The first peak corresponds to domains which have a single key and RRSIG. The second peak is from domains with two keys (a ZSK and KSK) and a single RRSIG, signed by the KSK. The third peak consists of domains with two keys (a ZSK and KSK) and two RRSIGs - one signed by the KSK, and another signed by the ZSK. The fourth peak shows domains with multiple keys and RRSIGs, possibly in the process of a key rollover, or using a configuration with two ZSKs.

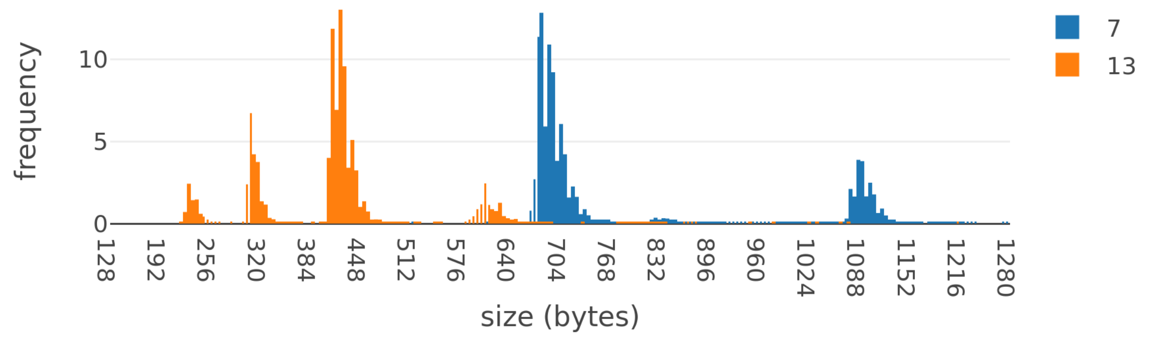


Fig. 6.3: DNSKEY RR Response Sizes from Algorithms 7 and 13

DNSSEC setups using RSA typically do not use a single key, as to achieve adequate security the key sizes are very large. The first peak with algorithm 7 corresponds to zones with two keys and two RRSIGs. The second peak shows domains with numerous keys and signatures.

The majority of responses using algorithm 13 fit in a 512 byte packet. The only exception to this is key rollovers or excessively complicated DNSSEC set ups. On the contrary, a minimal amount of algorithm 7 responses fit inside a packet of this size.

The proposed protocol described in section 4.3 would not suffer from these large DNSKEY responses as these RRs are extracted from the DNS and instead placed in the blockchain.

6.2.2 A & AAAA

If IPv4 and IPv6 address responses do not fit within the packet restriction of 512 bytes in order to prevent IP fragmentation then neither will NS responses which contain a "glue".

Response sizes were observed between the most popular RSA and ECDSA algorithms. Fig 6.4 shows 99.9% of DNSSEC A RR responses for both algorithms comes within the 512 byte packet limit.

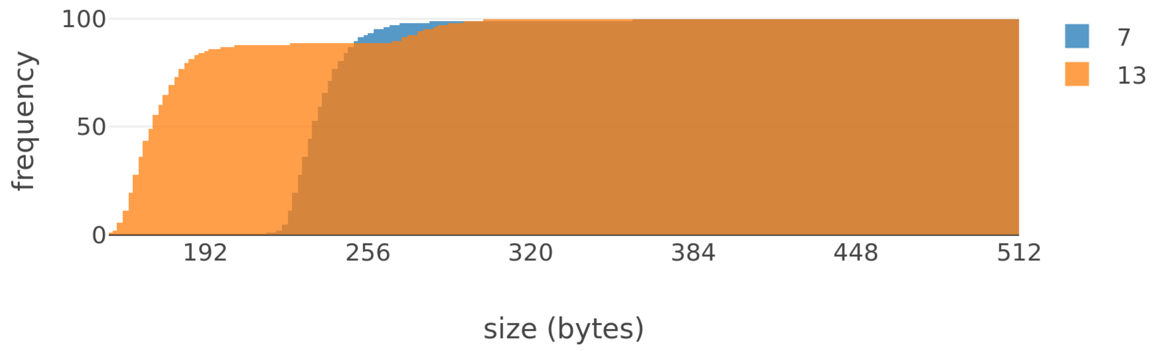


Fig. 6.4: A RR Response Sizes from Algorithms 7 and 13

Fig 6.5 also demonstrates that 99.9% of DNSSEC AAAA RR responses using either algorithm fits in a 512 byte UDP packet.

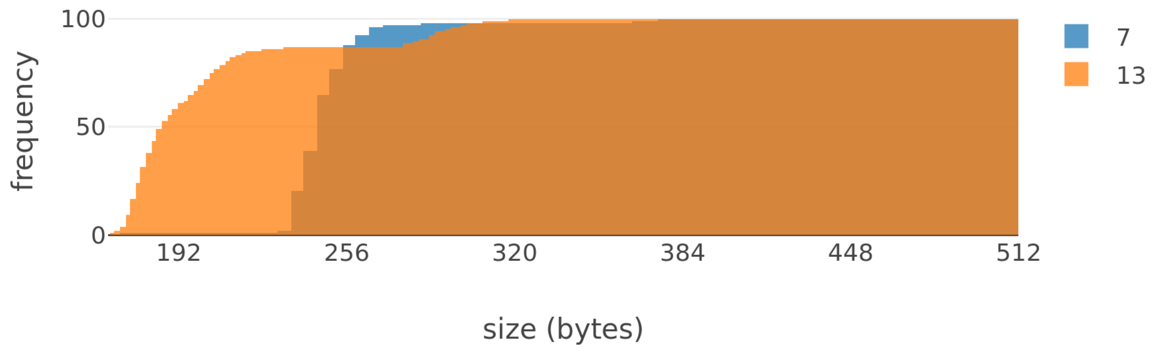


Fig. 6.5: AAAA RR Response Sizes from Algorithms 7 and 13

6.2.3 NS

It is important that a NS and “glue” RR is not subject to IP fragmentation, as this accounts for a significant portion of DNS traffic.

First, the response sizes for the most popular RSA and ECDSA algorithms were analysed. Fig 6.6 shows that 99.9% of responses using both algorithms fit inside a 512 byte UDP packet. This is an exceptionally important observation, as this could mean that key algorithms do not need to be restricted to ECDSA algorithms.

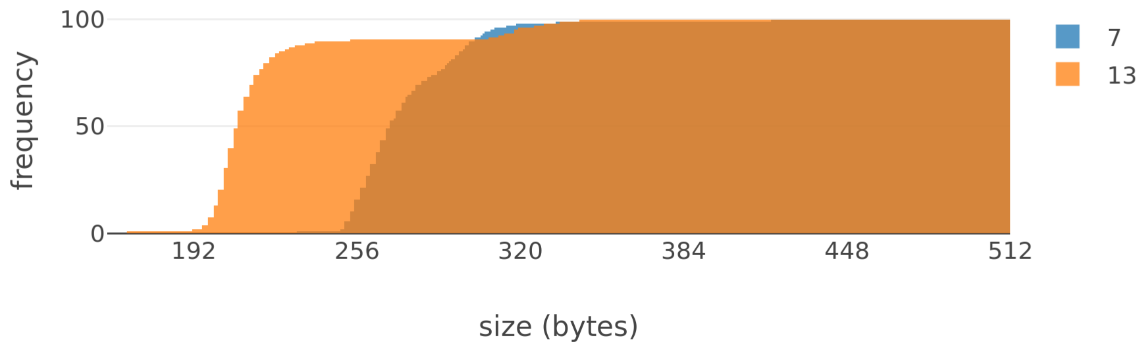


Fig. 6.6: NS RR Response Sizes from Algorithms 7 and 13

However, algorithm 7 produces the smallest responses, as shown in Fig 6.2. Algorithm 8 is the next most commonly used algorithm, which also results in larger responses due to the use of a SHA-256 hash function. Fig 6.7 shows that 99.5% of these responses also fit inside a 512 byte packet.

This means that there is a significant amount of freedom in what choice of algorithm can be used when DNSKEY RRs have been extracted from the hierarchy, as most responses will not suffer from IP fragmentation.

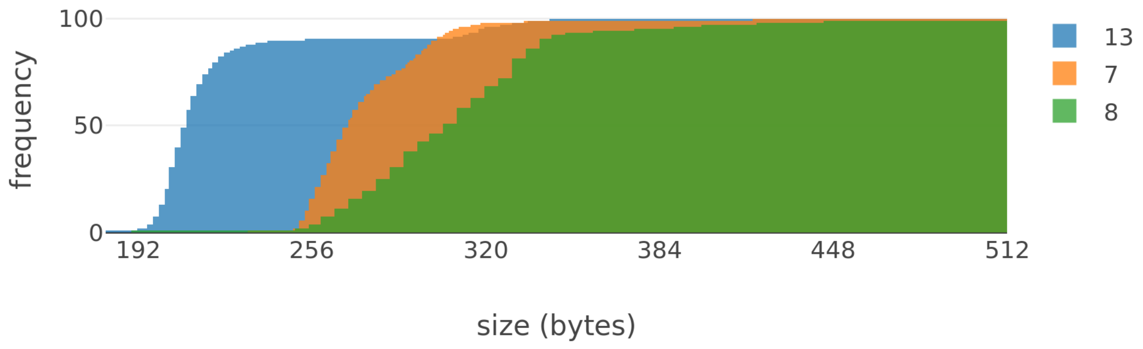


Fig. 6.7: NS RR Response Sizes from Algorithms 7, 8 and 13

6.2.4 NSEC

DNS traffic can potentially consist of a lot of NSEC responses, and so it is important to consider the implication it would have on IP fragmentation. Fig 6.8 compares algorithm 13 with algorithm 8. Algorithm 7 has been omitted as it uses NSEC3 responses instead.

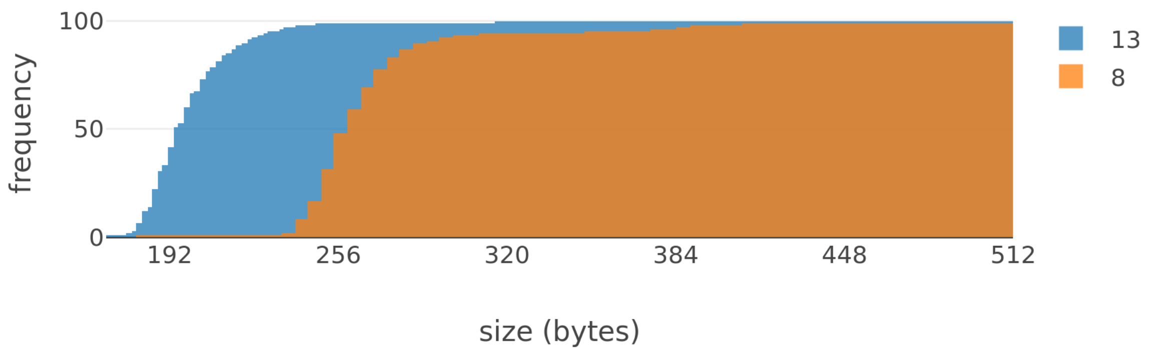


Fig. 6.8: NSEC RR Response Sizes from Algorithms 8 and 13

99.9% of responses using these two algorithms fit inside a 512 byte UDP packet. Once again, this is an important observation and allows for a flexible algorithm choice when using the protocol outlined in section 4.3.3

6.3 X.509 Certificate Sizes

Self-signed RSA certificates containing minimal information were created with various key sizes using `openssl`. Table 6.1 gives approximate minimum certificate sizes.

Algorithm 13 uses a 256-bit ECDSA key, which gives 128-bit security [8]. For this same security to be achieved using an RSA algorithm, a 3072-bit key would be necessary [8].

Key Size (bits)	Cert Size
512	696 B
1024	879 B
2048	1.3 kB
3072	1.6 kB
4096	1.9 kB

Tab. 6.1: Approximate RSA Certificate Sizes

A 256-bit ECDSA certificate was generated with the same parameters as the RSA certificates discussed. The certificate size came to be the same size as an equivalent 512-bit RSA certificate, i.e. 696 bytes.

Smaller certificates may become vital in keeping the blockchain at a manageable disk size, for both storage and look-up speeds. However, ECDSA signatures take longer to validate, and this could have a significant impact on the speed to verify DNSSEC responses [57].

7 Conclusion & Future Work

In this chapter we reflect on the system described in section 4, the demonstration shown in section 5, and the results analysed in section 6.

7.1 Goals Revisited

The goals that were listed in section 1 were:

- Provide the client with origin authentication
- Provide the client with message integrity
- Reduce the size of DNSSEC responses to the traditional DNS packet size
- Reduce number of requests necessary for signature validation
- Simplify the key management

The implementation demonstrated in section 5.4 showed that both client origin authentication and message integrity requirements were met. This is evident as signatures were validated using keys which were linked to a zone's identity using certificates. This identity binding is vital, as shown in the first version of the protocol in section 4.3.1. The use of a private blockchain provides significant advantages, as discussed in section 4.1.4. It was shown that this would also work with an existing DNSSEC setup using both a ZSK and a KSK.

The evaluation results in section 6.2 showed that only DNSKEY responses suffered from IP fragmentation. Since DNSKEY RRs would not exist anymore, and public keys are instead stored in the blockchain, this does not impact the system described in section 4. It appears the system would be capable of returning over 99% of responses in a single UDP packet, regardless of the choice of algorithm.

Traditional DNSSEC validation is shown in section 5.4.3. It is obvious that this produces far more requests, and is significantly more convoluted a process than that described in section 4.3.3 and shown in section 5.4.

Obtaining an X.509 certificate is a process many domains go through already, and the CA is the one to place these certificates on the blockchain. All that needs to be done is to sign each RRset, which is already performed in traditional DNSSEC. The need to place DS RRs in the zone above is also eliminated. It can be concluded that our proposed system's key management is simpler.

Not only is the key management simpler, but it is also extraordinarily flexible. If a more fitting PKI were discovered tomorrow, this could easily be integrated with the system with no impact on the DNSSEC response sizes from section 6.2.

7.2 Security Vulnerabilities

Due to the fact that not all zones are DNSSEC enabled, a resolver is unsure if it should expect a signature in the response. In traditional DNSSEC, the DS record in the parent zone tells a resolver that the child is DNSSEC enabled. However, the protocol described in section 4.3.3 does not use DS records.

This leaves a resolver open to a man in the middle attack, where an attacker can pretend a zone is not DNSSEC enabled by omitting signatures. The following are several potential solutions that could be implemented to mitigate this:

- A resolver could look at the blockchain and check for a valid certificate. If a certificate for a particular zone exists, then the resolver will expect a signature to be returned.
- An OPT pseudo RR[12] is a pseudo RR that was introduced as a way of extending DNS [23]. A flag is passed to a name server in this record to request RRSIG RRs in the response [58]. A flag could also be returned from the name server to the resolver specifying if the child domain is DNSSEC enabled.
- A flag could be added to the NS RR that that indicates if that name server is DNSSEC enabled. Unlike the pseudo-RR solution, this record could be signed. This might limit the backwards compatibility.

7.3 Key Types

An ECDSA key is much smaller than an equivalently secure RSA key [8]. However, ECDSA key validation is slower than RSA validation, and the affects of this on a resolver is undetermined [57].

Whilst the evaluation in section 6.2 shows that there is quite a bit of flexibility in terms of using RSA algorithms, more research could be performed into this, especially relating NSEC3 records.

[12]Records used to provide other types of information without being stored in a zone file

There is also the question of the blockchain size, and section 6.3 shows that ECDSA certificates consume notably less disk space. This gives a trade off of storage and flexibility.

7.4 Blockchain

As the implementation was never tested on a deployed blockchain, it is hard to know how fast it would be to retrieve certificates. There is a lack of studies on the topic of transaction look-ups, which presents us with two issues.

The first is the problem of bloating the blockchain, and how many certificates are stored until it becomes too slow to search. Eventually, certificates expire and must be renewed, keeping all the fresh certificates at the top of the blockchain. It becomes a question of how many zones must be using this solution before it becomes too slow.

One proposed solution would be to have a separate blockchain per TLD, although this could make it confusing for CAs to upload certificates. It might be necessary that domain owners become involved in the uploading of certificates in order to simplify the process. This could make the blockchain susceptible to denial of service attacks.

Another problem is the use of MultiChain streams, and the speed in using one per zone. There is a lot of scope to research this and various other methods of locating a transaction which contains the necessary certificate. A “BC” RR, as discussed in 4.3.2 could be used, which could contain a number of things:

- The hash of the certificate
- The transaction address containing the certificate
- A unique name which identifies the stream item

However, introducing another RR would require also create another RRSIG. This might push packets above 512 bytes for RSA digital signature schemes and require the use of ECDSA, reducing the protocol’s flexibility.

7.5 Closing Remarks

This project aimed to investigate alternative solutions to implement a security extension for DNS. It focused on new technologies, such as blockchain based naming services and encryption within the DNS protocol.

Although a feasible solution has been designed, there is still much more to be done to form a full implementation. There are also still many unanswered questions. A lot of research on ECDSA and the performance of using a blockchain is yet to be done. However, these things fell outside the scope of the project.

The protocol described in this report allows the PKI used to be completely agnostic to the overall function. If a better solution came about that did not use X.509 certificates, such as another blockchain PKI which tackles issues with trademark and copyright disputes, it would be usable with this design. This was a major focus point, taking inspiration from Blockstack remaining agnostic from the underlying blockchain.

To conclude, security is not easy. There are many aspects to consider from the perspective of a system administrator and a user. This report aimed to keep these groups in mind whilst attempting to fulfil a flexible and simple protocol, without compromising on security.

Bibliography

[1] Sooel Son and Vitaly Shmatikov. “The hitchhikers guide to DNS cache poisoning”. In: *International Conference on Security and Privacy in Communication Systems*. Springer. 2010, pp. 466–483 (cit. on pp. 1, 9).

[3] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. “Protecting browsers from DNS rebinding attacks”. In: *ACM Transactions on the Web (TWEB)* 3.1 (2009), p. 2 (cit. on pp. 1, 10).

[4] Gijs Van Den Broek, Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. “DNSSEC meets real world: dealing with unreachability caused by fragmentation”. In: *IEEE communications magazine* 52.4 (2014), pp. 154–160 (cit. on pp. 1, 13).

[5] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. “DNSSEC and its potential for DDoS attacks: a comprehensive measurement study”. In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM. 2014, pp. 449–460 (cit. on pp. 1, 13).

[7] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014 (cit. on pp. 3, 4).

[8] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. “Recommendation for key management part 1: General (revision 3)”. In: *NIST special publication* 800.57 (2012), pp. 1–147 (cit. on pp. 4, 49, 52).

[9] Ralph C Merkle. “Protocols for public key cryptosystems”. In: *Security and Privacy, 1980 IEEE Symposium on*. IEEE. 1980, pp. 122–122 (cit. on p. 4).

[10] Roger W Younglove. “Public key infrastructure. How it works”. In: *Computing & Control Engineering Journal* 12.2 (2001), pp. 99–102 (cit. on pp. 5, 6, 33).

[11] Matt Cooper, Yuriy Dzambasow, Peter Hesse, Susan Joseph, and Richard Nicholas. “RFC 4158-Internet X. 509 Public Key Infrastructure: Certification Path Building”. In: *Online at ftp://www. ietf. org/rfc/rfc4158. txt* (2005) (cit. on p. 5).

[12] D Cooper, S Santesson, S Farrell, et al. “Internet X. 509 Public Key Infrastructure Certificate and CRL Profile”. In: *RFC5280* (2008) (cit. on pp. 5, 6, 25).

[13] Warwick Ford and Michael S Baum. *Secure electronic commerce: building the infrastructure for digital signatures and encryption*. Prentice Hall PTR, 2000 (cit. on p. 5).

[14] Paul Mockapetris. “Rfc 1034: Domain names-concepts and facilities, 1987”. In: *URL: ftp://ftp. isi. edu/in-notes/rfc1034. txt* () (cit. on pp. 7, 9).

[15] Paul Mockapetris. “RFC 1035Domain namesimplementation and specification, November 1987”. In: *URL http://www. ietf. org/rfc/rfc1035. txt* (2004) (cit. on pp. 7, 8, 13, 45).

[16]S Thomson, C Huitema, V Ksinant, and M Souissi. “Rfc 3596: Dns extensions to support ip version 6”. In: *Disponibles en: https://www. ietf. org/rfc/rfc3596. txt, Consultada* 2 (2003) (cit. on p. 7).

[17]IP RFC0791. “Internet Protocol”. In: *J. Postel. September* (1981).

[18]S Deering RFC2460 and R Hinden. “Internet Protocol, Version 6 (IPv6) Specification”. In: *S. Deering* (1998).

[19]Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. “RFC 4034-Resource Records for the DNS Security Extensions (2005)”. In: *URL http://www. ietf. org/rfc/rfc4034. txt* (2008) (cit. on pp. 10–12, 40).

[20]B Laurie, G Sisson, R Arends, D Blacka, et al. “RFC 5155: DNS security (DNSSEC) hashed authenticated denial of existence”. In: *Request for Comments* 5155 (2008) (cit. on p. 11).

[23]Joao Damas, Michael Graff, and Paul Vixie. “Extension mechanisms for DNS (EDNS (0))”. In: (2013) (cit. on pp. 13, 52).

[24]Robert Braden. “RFC-1122: Requirements for internet hosts”. In: *Request for Comments* (1989), pp. 356–363 (cit. on p. 13).

[25]Roger M Needham. “Denial of service”. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. ACM. 1993, pp. 151–153 (cit. on pp. 13, 15).

[26]Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: (2008) (cit. on pp. 13–16).

[27]James Surowiecki. “Cryptocurrency”. In: *Technology review* 114.5 (2011), pp. 106–107 (cit. on p. 14).

[28]Adam Back et al. “Hashcash-a denial of service counter-measure”. In: (2002) (cit. on p. 14).

[29]John R Douceur. “The sybil attack”. In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260 (cit. on p. 16).

[30]Matthew Dempsky. “Dnscurve: Link-level security for the domain name system”. In: *Work in Progress, draft-dempsky-dnscurve-01* (2010) (cit. on p. 17).

[32]Jakob Schlyter and Paul Hoffman. “The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA”. In: (2012) (cit. on p. 19).

[33]Tim Dierks and Eric Rescorla. “Rfc 5246: The transport layer security (tls) protocol”. In: *The Internet Engineering Task Force* (2008) (cit. on p. 19).

[34]Richard Barnes. “Use cases and requirements for DNS-based authentication of named entities (DANE)”. In: (2011) (cit. on p. 19).

[37]Andreas Loibl and J Naab. “Namecoin”. In: *namecoin. info* (2014) (cit. on p. 21).

[39]Muneeb Ali, Jude C Nelson, Ryan Shea, and Michael J Freedman. “Blockstack: A Global Naming and Storage System Secured by Blockchains.” In: *USENIX Annual Technical Conference*. 2016, pp. 181–194 (cit. on pp. 22–24).

[42]Bastian Fredriksson. “A Distributed X. 509 Public Key Infrastructure Backed by a Blockchain”. In: (2017) (cit. on pp. 24, 25).

[43]Hitesh Tewari, Arthur Hughes, Stefan Weber, and Tomas Barry. “X509CloudFramework for a ubiquitous PKI”. In: *Military Communications Conference (MILCOM), MILCOM 2017-2017 IEEE*. IEEE. 2017, pp. 225–230 (cit. on p. 25).

[44]Gideon Greenspan. "MultiChain Private Blockchain, White Paper". In: *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf* (2015) (cit. on pp. 27–29).

[51]Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. "RFC 4035: Protocol modifications for the DNS security extensions, 2005". In: *URL http://www. ietf. org/rfc/rfc4035. txt* () (cit. on pp. 30, 31).

[56]Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. "The first collision for full SHA-1". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 570–596 (cit. on p. 46).

[57]Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. "Making the case for elliptic curves in DNSSEC". In: *ACM SIGCOMM Computer Communication Review* 45.5 (2015), pp. 13–19 (cit. on pp. 50, 52).

[58]D Conrad. "RFC 3225: Indicating Resolver Support of DNSSEC". In: *online], Dec* (2001) (cit. on p. 52).

Webpages

[2]CactusVPN. *All you need to know about DNS hijacking*. 2017. URL: https://www.cactusvpn.com/beginners-guide-online-security/dns-hijacking/ (visited on May 3, 2018) (cit. on pp. 1, 9, 10).

[6]Inc. DC Communications. *DNSSEC Deployment Report*. 2018. URL: http://rick.eng.br/dnssecstat/ (visited on Apr. 23, 2018) (cit. on p. 1).

[21]Internet Assigned Numbers Authority. *Domain Name System Security (DNSSEC) Algorithm Numbers*. 2017. URL: https://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml (visited on May 3, 2018) (cit. on p. 12).

[22]Internet Assigned Numbers Authority. *Delegation Signer (DS) Resource Record (RR) Type Digest Algorithms*. 2012. URL: https://www.iana.org/assignments/ds-rr-types/ds-rr-types.xhtml (visited on May 3, 2018) (cit. on p. 12).

[31]Daniel J. Bernstein. *All you need to know about DNS hijacking*. 2009. URL: https://dnscurve.org/out-implement.html (visited on May 3, 2018) (cit. on p. 17).

[35]Nick Johnson. *Ethereum Name Service Documentation: Introduction*. 2016. URL: https://docs.ens.domains/en/latest/introduction.html (visited on May 3, 2018) (cit. on p. 20).

[36]Nick Johnson. *Hosting a DNS domain on the blockchain*. 2017. URL: https://hackernoon.com/hosting-a-dns-domain-on-the-blockchain-58a41dc50028 (visited on May 3, 2018) (cit. on p. 20).

[38]*Namecoin Wiki: How to browse .bit domains*. URL: https://wiki.namecoin.org/index.php?title=How_to_browse_.bit_domains (visited on May 3, 2018) (cit. on p. 22).

[40]bitinfocharts. *BitInfoCharts*. URL: https://bitinfocharts.com/ (visited on Apr. 4, 2018) (cit. on pp. 22, 24).

[41]Blockstack. *blockstack-browser*. URL: https://github.com/blockstack/blockstack-browser (visited on May 3, 2018) (cit. on p. 24).

[45]MultiChain. *MultiChain Permissions Management*. URL: https://www.multichain.com/developers/permissions-management/ (visited on May 3, 2018) (cit. on pp. 27, 28).

[46]MultiChain. *Customizing blockchain parameters*. URL: https://www.multichain.com/developers/blockchain-parameters/ (visited on May 3, 2018) (cit. on p. 28).

[47]MultiChain. *MultiChain Data Streams*. URL: https://www.multichain.com/developers/data-streams/ (visited on May 3, 2018) (cit. on p. 28).

[48]MultiChain. *Moving on from big blockchains*. URL: https://www.multichain.com/blog/2016/01/moving-on-from-big-blockchains/ (visited on May 3, 2018) (cit. on p. 28).

[49]MultiChain. *Blockchains VS Centralized Databases*. URL: https://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases/ (visited on May 3, 2018) (cit. on p. 29).

[50]Internet Systems Consortium. *dnssec-signzone(8) - Linux man page*. 2018. URL: https://linux.die.net/man/8/dnssec-signzone (visited on May 1, 2018) (cit. on p. 29).

[52]Internet Systems Consortium. *BIND*. 2018. URL: https://www.isc.org/downloads/bind/ (visited on May 3, 2018) (cit. on p. 37).

[53]Don Moore. *DNS server survey*. 2014. URL: http://mydns.bboy.net/survey/ (visited on Apr. 29, 2018) (cit. on p. 37).

[54]MultiChain. *table_format.txt*. 2016. URL: https://github.com/MultiChain/multichain/blob/master/src/leveldb/doc/table_format.txt (visited on May 3, 2018) (cit. on p. 38).

[55]NLnet Labs. *LDNS*. 2018. URL: https://www.nlnetlabs.nl/projects/ldns/about/ (visited on May 3, 2018) (cit. on p. 38).

List of Figures

List of Tables