Blockchain Backed DNSSEC*

Scarlett Gourley and Hitesh Tewari

Trinity College Dublin, Ireland
gourleys@tcd.ie, htewari@tcd.ie

Abstract. The traditional Domain Name System (DNS) does 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 paper proposes a new system using blockchain technology. Our system aims to provide the same security benefits as DNSSEC whilst addressing the concerns that led to its slow adoption.

Keywords: Blockchain · DNS · DNSSEC · Fragmentation · Amplification Attack · X509

1 Introduction

The DNS is a hierarchical and decentralised naming service, which translates human readable, more readily memorable names into IP addresses. In its original form is insecure and suffers from a number of protocol attacks. These include DNS spoofing/cache poisoning [18], DNS hijacking [5], and DNS rebinding [13].

A solution to this was the introduction of DNSSEC. However, DNSSEC also suffers from various problems, namely IP fragmentation [21], potential denial of service attacks [16], and complicated key management. This has caused a slow adoption of DNSSEC, with only 4% of second level domains signed [11].

In this paper we propose an alternative security extension set for DNS. This system aims to provide a client all the security benefits that DNSSEC grants, i.e. origin authentication and message integrity. We also wish to reduce responses to fit into a 512-byte UDP packet as per the original DNS standard [14], and to simplify the key management. This is in order to mitigate possible availability issues associated with DNSSEC. Finally, we intend to produce a protocol which minimises the number of requests necessary for signature validation.

* This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie)

2 Background

In this section, we present the current DNSSEC system and how it operates. We also provide a brief introduction to the X.509 Public Key Infrastructure (PKI) and blockchain technology.

2.1 DNSSEC

DNSSEC is a set of security extensions which provides origin authentication and message integrity to DNS. It does this by using public key based digital signatures. To facilitate this, some new resource records (RRs) were added [2]:

- **RRSIG** records contain a digital signature
- **DNSKEY** records contain a signing key
- **DS** records contain the hash of a DNSKEY
- **NSEC** and **NSEC3** records map 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 [2].

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 [2].

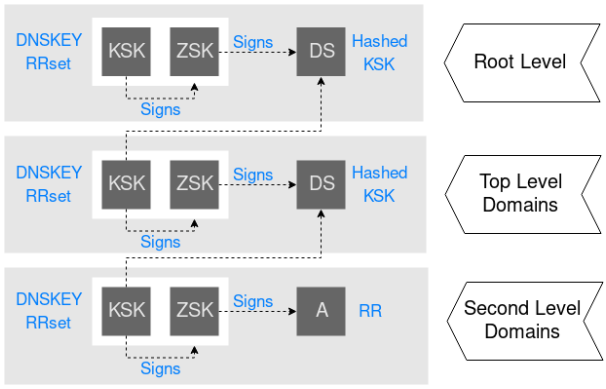

Fig. 1. Chain of Trust

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 [2]. This carries up to the root server, where the root KSK is stored on the client's machine. Fig 1 contains a visualisation of this trust chain.

2.2 PKI

A PKI is a system for managing, distributing and revoking digital certificates [22]. 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) [22].

X.509 is a digital certificate standard that is used in many internet protocols, such as HTTPS [9]. In order for an entity to receive an X.509 certificate, they must create a Certificate Signing Request (CSR) which includes their identity and their public key. This CSR gets sent to a Registration Authority [8]. A Registration Authority is approved by a CA to accept, reject and revoke certificates [8, 12]. If the CSR is accepted, an X.509 certificate signed by the CA is returned to the entity [8].

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 [22, 8].

3 Related Work

The issue with DNSSEC lies with the need to have a chain of trust up to the root server. This leads to an excessive number of requests, and hence signed responses. If a domain name was bound to a public key in a trusted manner, we could remove this chain of trust.

Blockchain based PKIs have been proposed numerous times. Blockstack operates a complete naming system where namespaces can be registered [1]. However, any user can buy any name as there is no dispute authority. This means organisations will have difficulty registering their trademarked and copyrighted names.

X509Cloud is a framework which allows X.509 certificates to be added to a public blockchain network [20]. When a CSR is accepted the certificate is not only sent back to the individual, but it is also broadcasted on the X509Cloud blockchain network [20]. 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.

This framework 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.

4 Protocol Design

The aim of this section is to create a DNS security extension, which does not suffer from common issues which arise in DNSSEC deployment. If the keys in the DNS hierarchy are replaced with some other PKI, we can link an identity with these keys [22]. We can therefore remove DNSKEY and DS RRs from the DNS hierarchy and simply check the repository of whatever PKI we are using.

4.1 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 [7].

4.2 PKI

The X509Cloud blockchain network will be used as the store for X.509 certificates. A single certificate which associated a domain name with a CSK is created. An X.509 certificate is often created for a domain so that other internet protocols can be used, and this same certificate could simply be used to sign DNS RRs.

4.3 Protocol

It is important to note that at any given stage in the resolution, the X509Cloud blockchain network 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 name server was cached the resolution could securely start from there.

A look-up where we would like the IPv4 address of `google.com` and know the IP address of the name server which contains this would operate as follows:

1. A client initiates a normal lookup for the domain `google.com`, which gets forwarded to a recursive resolver
2. The recursive resolver will query the `google.com` name server for the `google.com` A RR
3. The `google.com` name server will respond with the A RR for `google.com` and a signature
4. The recursive resolver will search for the most recent entry of the `google.com` certificate in the X509Cloud blockchain network, check the validity of the certificate, and validate the signature
5. The verified A RR will be forwarded back to the client

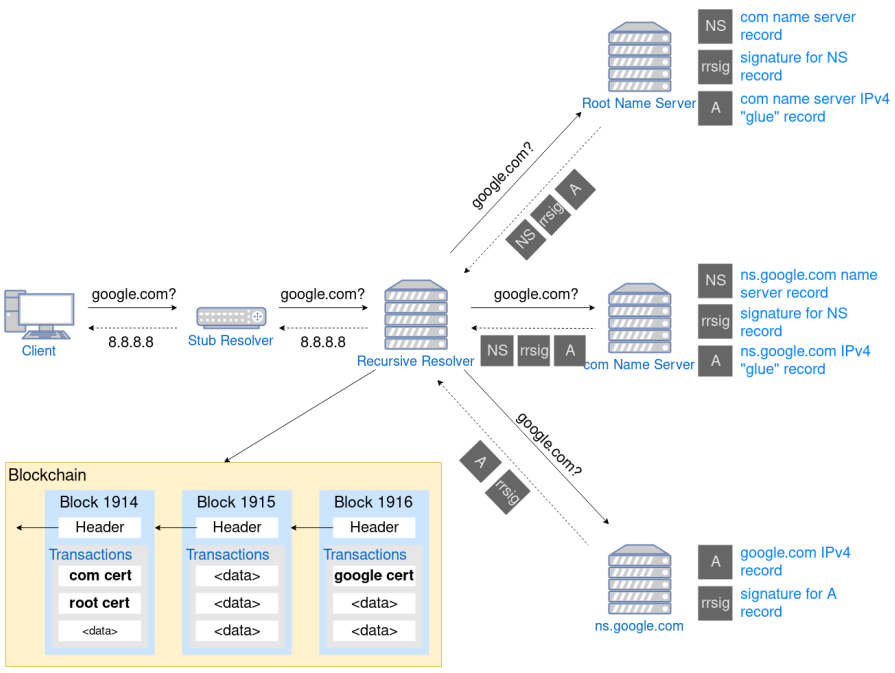

Fig. 2. Proposed Protocol

Once again, this could simply begin at the `com` name server if the IP address for the `google.com` name server was not known. The ability to begin a resolution from anywhere speeds up the verification process considerably as each level does not need to be validated.

The following is an example verification on the `example.scarlett` domain which has a single certificate containing its CSK:

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

Querying for: example.scarlett.

-----------------------
Verifying Resource Record
-----------------------
RRSIGs of type NS to validate: 1
Failed to get certificate example.scarlett. ZSK from the
    database: 11

DNSKEY string: example.scarlett. IN DNSKEY 257 3 13
    tP2dIWHtAQXTDDSrUNM8EjBQDzLC3DJkKBqWAtd34+
    WpgxtN7KKSFimHCYsdHaNPuLjWEGR0pgWhG9yFL8unCQ==

```

```
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
```

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 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 if RRSIGs suffer from IP fragmentation still exists.

5 DNSSEC Response Sizes

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.

Around 1.63% of second level domains in the `net` zone are signed. Fig 3 shows the distribution of DNSSEC algorithms that are used by these.

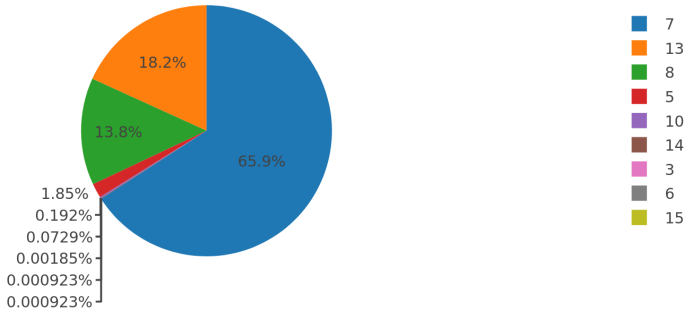

Fig. 3. Distribution of DNSSEC Algorithms

5.1 DNSKEY

DNSKEY requests were expected to yield the largest response. Fig 4 shows a histogram of a variety of RSA algorithms.

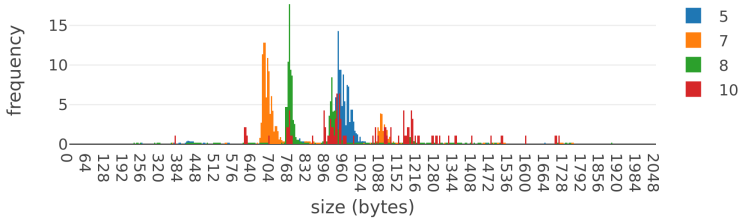

Fig. 4. DNSKEY RR Response Sizes from RSA Algorithms

Algorithm 7 yielded the smallest response size. This explains its prevalent use shown in Fig 3, as many zones will attempt to reduce the size of their DNSSEC responses in order to avoid accessibility issues. 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 [19].

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 [14].

Fig 5 compares the most common RSA algorithm against the most common ECDSA algorithm, as determined from Fig 3. Each of these peaks show the various forms of DNSKEY configurations.

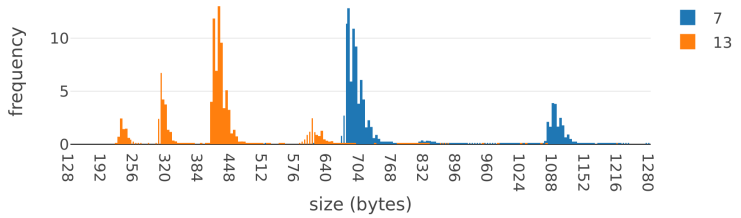

Fig. 5. DNSKEY RR Response Sizes from Algorithms 7 and 13

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.

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

5.2 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 presents a cumulative graph showing that 99.9% of DNSSEC A RR responses for both algorithms comes within the 512 byte packet limit.

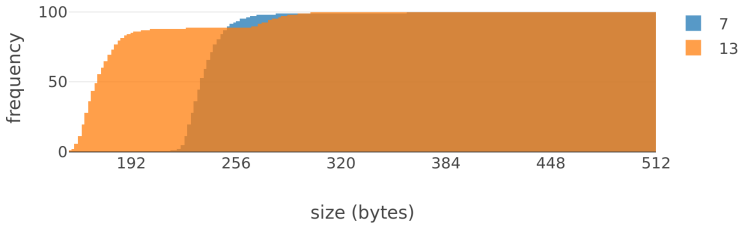

Fig. 6. A RR Response Sizes from Algorithms 7 and 13

Fig 7 also demonstrates that 99.9% of DNSSEC AAAA RR responses using either algorithm fits in a 512 byte UDP packet.

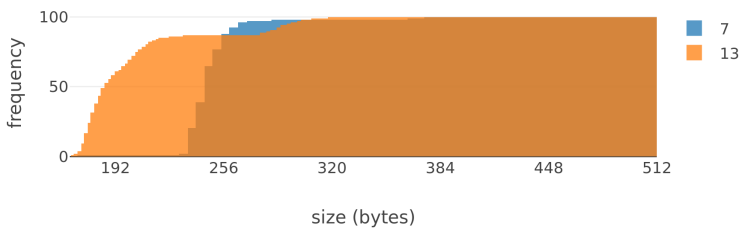

Fig. 7. AAAA RR Response Sizes from Algorithms 7 and 13

5.3 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 8 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.

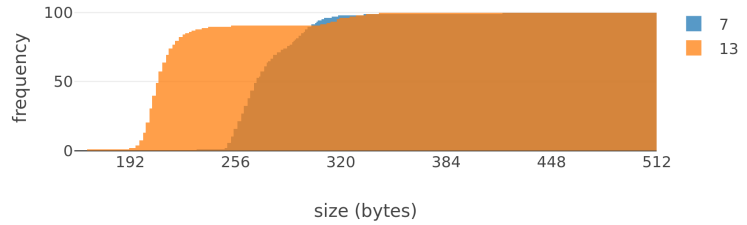

Fig. 8. NS RR Response Sizes from Algorithms 7 and 13

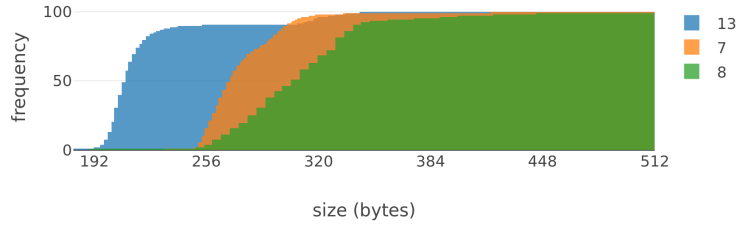

Fig. 9. NS RR Response Sizes from Algorithms 7, 8 and 13

However, algorithm 7 produces the smallest responses, as shown in Fig 4. 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 9 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.

5.4 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 10 compares algorithm 13 with algorithm 8. Algorithm 7 has been omitted as it uses NSEC3 responses instead.

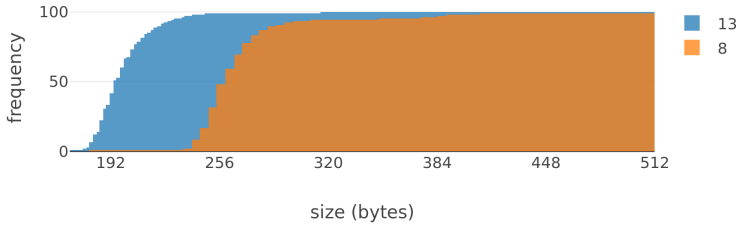

Fig. 10. NSEC RR Response Sizes from Algorithms 8 and 13

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 proposed protocol.

6 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 the number of requests necessary for signature validation
- Simplify the key management

We concluded in section 4 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 which are available on the X509Cloud network.

The evaluation results in section 5 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.

We describe in section 4 that a convoluted validation process is no longer necessary, as the blockchain allows us to trust the certificate at any level in the DNS lookup process. This produces far less requests than traditional DNSSEC. We can conclude the number of requests necessary are reduced.

As for key management, all that needs to be done is to sign each RRset which is already performed in traditional DNSSEC. Only a single key is necessary for the proposed protocol. 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 5.

7 Future Work

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 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:

- If a certificate for a particular zone exists on the blockchain, then the resolver will expect a signature to be returned.
- A flag is passed to a name server in the OPT pseudo RR could 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.

One of our aims was to try and keep the protocol flexible, allowing the PKI to be agnostic to the overall protocol, and keeping the choice of algorithm for signatures open. An ECDSA key is much smaller than an equivalently secure RSA key [4], but validation is slower than RSA validation [17]. The affects of this on a resolver is undetermined. More research could be performed into this, especially relating NSEC3 records.

References

1. Ali, M., Nelson, J.C., Shea, R., Freedman, M.J.: Blockstack: A global naming and storage system secured by blockchains. In: USENIX Annual Technical Conference. pp. 181–194 (2016)

2. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: Rfc 4034-resource records for the dns security extensions (2005). URL http://www. ietf. org/rfc/rfc4034. txt (2008)
3. Back, A., et al.: Hashcash-a denial of service counter-measure (2002)
4. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management part 1: General (revision 3). NIST special publication **800**(57), 1–147 (2012)
5. CactusVPN: All you need to know about dns hijacking (2017), https://www.cactusvpn.com/beginners-guide-online-security/dns-hijacking/
6. Conrad, D.: Rfc 3225: Indicating resolver support of dnssec. online], Dec (2001)
7. Consortium, I.S.: dnssec-signzone(8) - linux man page (2018), https://linux.die.net/man/8/dnssec-signzone
8. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet x. 509 public key infrastructure certificate and crl profile. RFC5280 (2008)
9. Cooper, M., Dzambasow, Y., Hesse, P., Joseph, S., Nicholas, R.: Rfc 4158-internet x. 509 public key infrastructure: Certification path building. Online at ftp://www. ietf. org/rfc/rfc4158. txt (2005)
10. Damas, J., Graff, M., Vixie, P.: Extension mechanisms for dns (edns (0)) (2013)
11. DC Communications, I.: Dnssec deployment report (2018), http://rick.eng.br/dnssecstat/
12. Ford, W., Baum, M.S.: Secure electronic commerce: building the infrastructure for digital signatures and encryption. Prentice Hall PTR (2000)
13. Jackson, C., Barth, A., Bortz, A., Shao, W., Boneh, D.: Protecting browsers from dns rebinding attacks. ACM Transactions on the Web (TWEB) **3**(1), 2 (2009)
14. Mockapetris, P.: Rfc 1035—domain names—implementation and specification, november 1987. URL http://www. ietf. org/rfc/rfc1035. txt (2004)
15. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
16. van Rijswijk-Deij, R., Sperotto, A., Pras, A.: Dnssec and its potential for ddos attacks: a comprehensive measurement study. In: Proceedings of the 2014 Conference on Internet Measurement Conference. pp. 449–460. ACM (2014)
17. van Rijswijk-Deij, R., Sperotto, A., Pras, A.: Making the case for elliptic curves in dnssec. ACM SIGCOMM Computer Communication Review **45**(5), 13–19 (2015)
18. Son, S., Shmatikov, V.: The hitchhiker's guide to dns cache poisoning. In: International Conference on Security and Privacy in Communication Systems. pp. 466–483. Springer (2010)
19. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full sha-1. In: Annual International Cryptology Conference. pp. 570–596. Springer (2017)
20. Tewari, H., Hughes, A., Weber, S., Barry, T.: X509cloud—framework for a ubiquitous pki. In: Military Communications Conference (MILCOM), MILCOM 2017-2017 IEEE. pp. 225–230. IEEE (2017)
21. Van Den Broek, G., van Rijswijk-Deij, R., Sperotto, A., Pras, A.: Dnssec meets real world: dealing with unreachability caused by fragmentation. IEEE communications magazine **52**(4), 154–160 (2014)
22. Younglove, R.W.: Public key infrastructure. how it works. Computing & Control Engineering Journal **12**(2), 99–102 (2001)