The DNS to Reinforce the PKIX for IoT Backend Servers: Implementation and Evaluation

Ibrahim Ayoub
Afnic
Université Paris-Saclay
Yvelines, France
ibrahim.ayoub@afnic.fr

Gaël Berthaud-Müller
Afnic
Yvelines, France
gael.berthaud-muller@afnic.fr

Sandoche Balakrichenan
Afnic
Yvelines, France
sandoche.balakrichenan@afnic.fr

Kinda Khawam
Laboratoire DAVID
Université de Versailles Saint-Quentin-en-Yvelines
Versailles, France
kinda.khawam@uvsq.fr

Benoît Ampeau
Afnic
Yvelines, France
benoit.ampeau@afnic.fr

***Abstract*—The current Public Key Infrastructure on the Internet depends on binding names to public keys via the digital X.509 certificates. These certificates are issued by certificate authorities (CAs), and only certificates verified by one of these CAs are accepted. This model requires TLS (Transport Layer Security) clients to store dozens of trusted CA certificates and has proved to lack immunity against security breaches. The Domain Name System (DNS) could reinforce and complement the functioning of CAs. The DNS-Based Authentication of Named Entities (DANE) protocol is designed to use DNS to bind certificates or keys to domain names by adding TLSA resource records (RRs) to zones. Verification is done by fetching the certificate's TLSA RR and matching it against it. DANE utilizes DNSSEC, which guarantees the integrity and authenticity of DNS responses. Besides TLS servers, TLS clients could also have TLSA RR and be verified via DANE, allowing mutual authentication between clients and servers. In this paper, we implement DANE and perform mutual authentication for IoT backend servers. Our use case is a mutual authentication process between LoRaWAN's Join and Network servers upon receiving a *join request* from a LoRaWAN end-device. We study the latency introduced by mutual authentication via DANE and compare it to traditional CA.**

***Index Terms*—DNS, DANE, DANCE, IoT, LoRaWAN, Authentication**

I. Introduction

Almost every connection to a server on the Internet involves two basic steps: a Domain Name System (DNS) resolution and public key certificate verification. DNS maps between domain names and IP addresses, allowing clients to use names to initiate connections. The Public Key Infrastructure using X.509 digital certificates (PKIX), on the other hand, helps the clients verify the server's identity and eventually set up a secure connection. Clients in this connection model, the commonly used model for web communications, are not required to prove their identity, and this burden falls solely on the servers. Having mutual authentication between clients and servers requires that clients also possess the equivalent of a public key certificate (e.g., X.509 certificate) that they should provide to servers. Mutual authentication is desirable to reinforce session security, and it is more so in IoT environments. IoT networks could benefit from mutual authentication between their different network elements. Such networks usually involve end-devices that communicate with gateways via radio frequencies and IP-enabled backend servers that receive and send data to the end-devices via the gateways. The prevalent security practices in IoT today include having keys pre-shared and saved on end-devices and backend servers to allow secure sessions between the two and using the PKIX for authenticating the communication between the IP-enabled backend servers in the network. Sharing keys is done by different methods. Hard coding keys on the device and servers, printing them on the device, or even sending them by email, for example, are all key-sharing methods used today that clearly pose a security risk. For communication between backend servers, setting up secure sessions requires using the PKIX, which, besides its cost, has some drawbacks and is not necessarily optimal. It relies on multiple root certificate authorities (CAs) for issuing digital certificates. The root CAs have self-signed certificates and can sign and provide certificates to intermediary CAs. Intermediary CAs are responsible for issuing certificates to domains and servers. Being authenticated is based chiefly on owning a certificate issued by an intermediary CA and entities that trust that CA will trust the certificate and the information it contains. These CAs are distributed in nature and do not have a central trust anchor, and each major vendor, Google and Apple, for example, has its list of CAs (root store) that it trusts [1]. Moreover, these CAs could be compromised and could issue rogue certificates [2]. On the other hand, abandoning public CAs and resorting to self-signed certificates to cut down on expenses does not work smoothly, especially when multiple stakeholders are involved. For example, in our work

This work was supported by the French National Research Agency (ANR) - PIVOT (ANR-20-CYAL-0002)

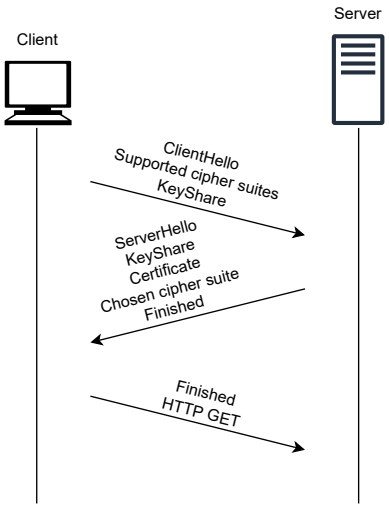

Fig. 1. TLS-handshake

on IoT Roaming [3], the stakeholders were universities, and they voiced their discontent about having our company acting as a single root CA for the different servers in the network.

Ameliorating the PKIX is not only feasible but, given the preceding, is also encouraged. The approach we deal with in this paper builds on DNS and its security extensions (DNSSEC) to reinforce the PKIX. Using DNS gives some control back to entities that require being authenticated by allowing them to publish information about their certificate in their DNS zones which, alongside data integrity and authentication provided by the DNS security extensions, mitigates the drawbacks of the PKIX in its current form.

The remainder of the paper is structured as follows. Section II discusses how DNS could be used to implement mutual authentication. Our set-up and how we used mutual authentication between two backend servers in a LoRaWAN network is presented in Section III. An evaluation of our implementation is done in section IV, and, finally, we conclude the paper in section V.

II. DNS Complementing the PKIX in IoT Environments

A. The Current PKIX

The Transport Layer Security (TLS) protocol [4], [5] is one of the main elements of security in today's Internet that relies on the Public Key Infrastructure (PKI). For the most part, secure communications over the Internet start with a TLS handshake, as depicted in Fig. 1. A TLS client connecting to a TLS server will receive that server's X.509 certificate. The elements of the certificate include information about the certificate issuer, information about the recipient of the certificate and the certificate's digital signature. The client ensures that the server is whom it claims to be by verifying that the certificate the server sent is legitimate. A chain of trust exists to help TLS clients verify the authenticity of certificates. This chain of trust starts at the top with root CAs. Root CAs are trusted by default and have self-signed certificates that allow them to issue other certificates creating intermediary CAs. Intermediary CAs, which the root CAs trust, can issue certificates for servers and websites. An X.509 certificate contains several fields describing the issuing CA and its owner. The issuing CA signs the certificate by encrypting the hash (digest) of the certificate fields with its private key. The problem with the current model is that certificates are not required to be verified by the CA that issued them. Moreover, a CA that wants its certificates to be trusted must be added to the root store containing the list of trusted root CA certificates. Different vendors and browsers have different root stores. Moreover, the root and intermediary CAs are not immune to security breaches [6], [7], and could, in theory, issue certificates for any domain like the root authority Diginotar did when it issued a fraudulent Google certificate in 2011 [2].

B. DNS and its Security Extensions

C. DNS Concepts

The history of the Internet goes back to the 1960's where it started out as a military project funded by the United States Department of Defense. The goal was to connect several pentagon institutions across the country using the existing telephone network. The project undertaken by the Advanced Research Projects Agency (ARPA) lead to the creation of ARPANET, the rudimentary ancestor of the Internet [**?**] which later evolved in hardware, software and protocols to become the modern-day Internet. Given its small-scale initial deployment, one could justly say that the Internet was not expected to become the huge global network we know today. Therefore, the network functions designed back then were in line with the existing small-scale network. One function that stands out when talking about poor design is naming.

Previously, the mapping between host names and addresses was kept in a text file called HOSTS.TXT which was retrieved by all hosts via FTP. The bandwidth required to download this file was proportional to N^2 for a network of N hosts [**?**]. This solution worked properly when N was small, but the growth of the Internet mandated that a more scalable solution is found. This solution was the Domain Name System (DNS).

The DNS was presented in RFC 1034 [**?**] and RFC 1035 [**?**]. The goal was a scalable and distributed system that performs the mapping from hostnames to addresses. In today's Internet, a crucial role of DNS is to map human-friendly domain names like example.com to machine-friendly IPv4 addresses like 192.0.2.1 or the less friendly alpha-numeric IPv6 addresses like 2001:db8:ff00:3f:d898:469f:2b95:162e. Moreover, DNS moved from a simple tool that had the sole purpose of referring to resources' location to a crucial part of the Internet. Almost any interaction with the Internet nowadays requires DNS.

The main components of the DNS are:

- The Domain Name Space: The domain name space is structured as an inverted tree and it evolves from the root at the top from which the resolving begins. Just

below the root are the top-level domains (TLD). The Internet Assigned Numbers Authority (IANA) specifies four types of TLDS. TLDs could be generic TLDs (gTLD) e.g., .com, .net, country-code TLDs (ccTLD) e.g., .uk, .fr, sponsored TLDs (sTLD) e.g., .edu, .gov, and the infrastructre TLD (iTLD) .arpa. below top-level domains come the DNS domains. Figure

- Name server: Name servers are programs that have the structure of a subset of the domain name. Name servers are said to be authoritative over a domain if they can give an answer about that domain. For example, the name server example is authoritative over the domain name example.com and could return a definitive answer for it. Name servers which are authoritative over a certain domain could have subdomains over which they are authoritative. In the figure, subdomain.example.com is a subdomain of example.com and the example name server is authoritative over it. Moreover, a name server could delegate authority to lower name servers. In the figure, example.com delegates authority to delegate.example.com which means that the example name server does not answer for domains ending in delegate.example.com. Name servers have text files called $MasterFiles$ that hold information about the zone or domain over which each name server is authoritative.
- Resource Records: A unit of information in a DNS $MasterFile$ is called a $ResourceRecord$ (RR). Each RR is a piece of information about a certain domain (or node in the domain name space tree) which a DNS client may ask for. Each RR has 6 fields as specified in RFC 1035 [**?**]:
  - NAME: specifies the owner of the RR which is the domain this RR belongs to or the node where it resides.
  - TYPE: specifies the type of information the RR holds. For example, RR TYPE A will hold the 32-bit IPv4 address of the domain, RR TYPE $AAAA$ will hold the 128-bit IPv6 address of the domain [**?**], RR TYPE MX returns the mail transfer agents of the domains and so on.
  - CLASS: defines the protocol family for the RR. For example, CLASS IN stands for Internet.
  - TTL: specifies the duration the RR could be cached for before the node holding the RR is consulted again.
  - RDLENGTH: specifies the length in bytes of the RDATA field.
  - RDATA: a string that describes the resource record.
- Resolvers: act as the interface between user programs and the DNS servers. Whenever a DNS query is to be executed, a user consults a predefined resolver. This resolver could be a local one on the user's own machine or in the user's home/office, that of the user's ISP, or one of the Internet's public resolvers like google's 8.8.8.8 or cloudflare's 1.1.1.1 public DNS resolvers.

1) The DNS Lookup Process: Each DNS lookup process starts when a certain resource on the Internet is requested. A DNS query is constructed and sent to a resolver. DNS Queries ask for a 1) QNAME or query name which is the domain name to be resolved. QNAME will contain a Fully Qualified Domain Name (FQDN) which is the complete name of the host/resource on the Internet. 2) QTYPE which specifies the type of information requested about the QNAME. 3) QCLASS which specifies the class of the query. For example, QCLASS IN is for the Internet. Queries could be handled either recursively or iteratively. Iterative servers will either return a definitive answer to a query if they are authoritative over the requested domain or simply inform the client that an answer was not found and if possible give referral to other servers. Recursive servers will return a definitive answer if they are authoritative over the requested domain or carry on with the query and ask other servers to finally return the answer to the client. Figure 3 explains the process. In most cases the query is sent from the user's or DNS client's stub resolver, usually part of the browser or OS, to the recursive resolver. The recursive resolver will send the query to the root server which will give its referral to one of the TLDs. The resolver then sends the query to the TLD name server referred to it by the root server. The process repeats as the TLD refers to an authoritative server below it. This process goes on until an authoritative server of the QNAME contained in the query is reached after which this authoritative server returns a definitive answer, which is usually an address, to the recursive resolver. Finally, the recursive resolver delivers the answer to the DNS client. The process described assumes a cold cache in the resolver. This could be the case if the server's cache has been cleared or the server has just been started. In almost all other cases, and to expedite the resolving process and reduce DNS traffic on the network, the resolvers, and depending on the TTL field of every RR, temporarily cache the queries they resolve and answer related queries without consulting any authoritative servers.

D. Using DNS for Certificate Verification

The typical DNS resolution process starts at the root zone. The recursive resolver sends the query to one of the root name servers, and the root name server will send back to the resolver a referral to the appropriate Top Level Domain (TLD) such as .com, .net, for example. The original design of DNS did not account for security and is therefore prone to security and privacy risks. Consequently, in addition to the referral, a DNSSEC [8]–[10] chain of trust comprised of digital signatures is also sent to authenticate the referred TLD. The process is repeated down the DNS tree until the authoritative name server of the requested domain is reached as demonstrated in Fig. 4. DNSSEC ensures that the received data's origin is authentic and that it has not been tampered

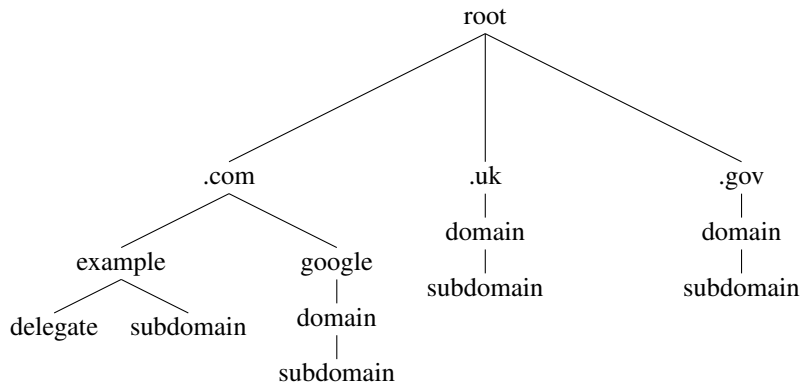

Fig. 2. Caption

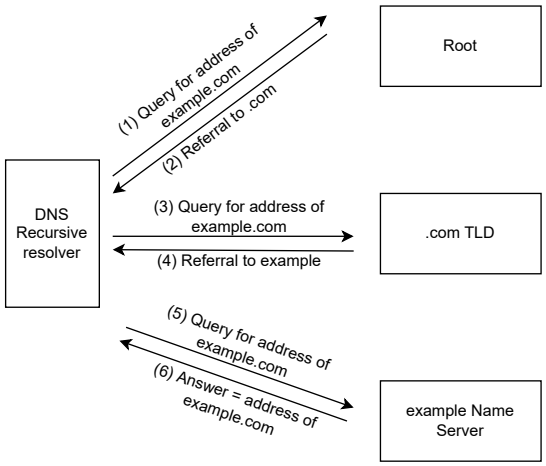

Fig. 3. DNS Lookup Process

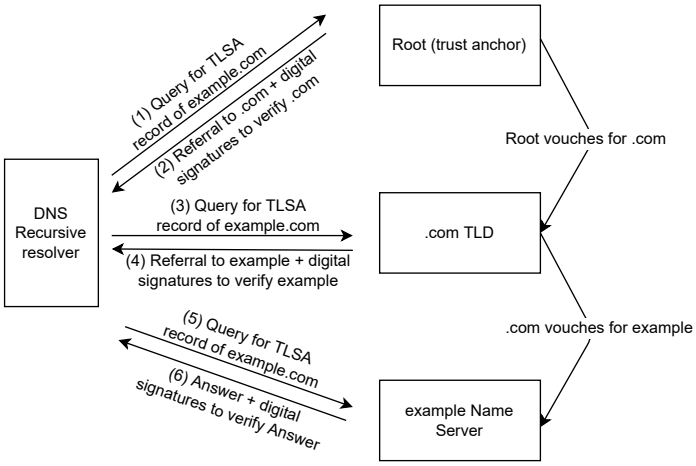

Fig. 4. DNSSEC chain of trust

with on the way. It is evident, however, that DNSSEC does not ensure the privacy of DNS traffic.

The DNS-Based Authentication of Named Entities (DANE) [11]–[13] gives control back to zone owners regarding verifying their certificates. DANE's main feature is the TLSA Resource Record (RR) that owners can add to their zone. TLSA RRs form an association between certificates and the domains to which those certificates were given. A typical TLSA RR for a TLS server having the hostname www.example.com, using TCP, and listening on port 443 looks something like Fig. 5.

The fields of the TLSA RR are [11]:

- Certificate Usage Field (1 byte): could be '0' to specify a CA certificate or the public key of such a certificate limiting the CAs that can issue certificates for domains owning the TLSA RR, '1' to specify an end entity certificate or its public key, and '2' to specify a certificate or its public key.
- Selector Field (1 byte): this specifies which part of the certificate received from the server will be matched against the association data. It could be '0' for 'Full Certificate' or '1' for SubjectPublicKeyInfo.
- Matching Type Field (1 byte): this specifies how the certificate association is presented. It could be '0' for 'Exact match', '1' for SHA-256 hash, or '2' for SHA-512 hash.
- Certificate Association Data Field: this specifies the 'certificate association data' to be matched depending on the Selector value.

Clients intending to connect to a TLS server will receive a certificate, and instead of validating that certificate using a traditional CA, they will perform a DNS query to fetch the TLSA RR from the zone the server to which they are connecting belongs. If the information in the TLSA RR matches the information received from the TLS server, validation is complete, and a secure connection can be initiated. The question may arise: why would zone owners trust DANE since DNS's infrastructure is not designed with security capabilities? Furthermore, why would zone owners move from dealing with CAs to dealing with DNS? In fact, DANE uses DNS Security Extensions (DNSSEC) [8]–[10] to guarantee the authenticity and integrity of the TLSA RRs.

DNSSEC provides data integrity and data origin authentication. In addition, DNSSEC can explicitly answer in case of record non-existence. In each DNSSEC-enabled zone, there

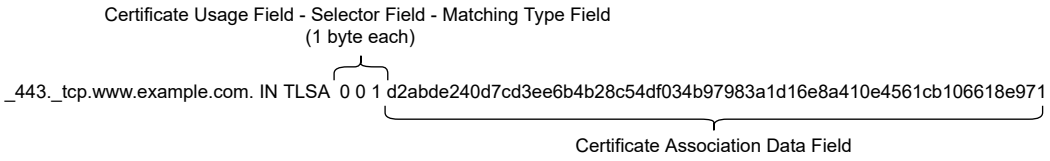

Fig. 5. Example of a TLSA resource record

should be two pairs of keys: a private/public Key Signing Key (KSK) pair and a private/public Zone Signing Key (ZSK) pair. KSKs are used to sign keys, while ZSKs are used to sign non-key data. The anchor of trust that DNSSEC depends on is the root zone's public KSK. This key should be known to all resolvers using the DNSSEC chain of trust.

E. Mutual Authentication via DANE

In its original form [11], [13], DANE is concerned with authenticating TLS servers. TLS clients, on the other hand, were not thought to have certificates that needed verification. This was the main driver behind creating the DANE Authentication for Network Clients Everywhere (DANCE) IETF working group [14]. The Internet Engineering Task Force (IETF) is a standardization body responsible for issuing Internet Standards as Request for Comments (RFCs). IETF has several working groups, each with a primary theme that it addresses. The DANCE working group aims to extend DANE to enable TLS client authentication using certificates or raw public keys. Like TLS servers, TLS clients will have their raw public keys or certificates and corresponding TLSA RRs in the DNS zone they belong to. This allows mutual authentication via DANE between the TLS clients and servers where a client can check and authenticate the public key or certificate received from the server, which in turn could verify the public key of the certificate received from the client. The DANCE working group have issued two internet drafts so far, *TLS Client Authentication via DANE TLSA RRs* [15] and *TLS Extension for DANE Client Identity* [16]. Both drafts are active at the time of writing. The first draft [15] describes how to publish TLSA RRs for TLS clients. Client identities here are assumed to be represented by a DNS domain name. Just as it is when verifying servers, TLS clients that plan on being authenticated via DANE must have a raw public key or a certificate binding them to a public key. The keys or certificates have corresponding TLSA RRs allowing verifying them via DNS. The clients should always signal their intent to be verified with DANE to spare the server from sending unnecessary DNS queries. The second draft [16] addresses that as it specifies a TLS extension that allows clients to express their support for DANE and their intent to be verified and allows them to share their DANE identity with the server.

III. System Model

This paper aims to implement the DANCE WG's drafts [15], [16] in an IoT environment where a reliable mutual authentication is vital. We modified the Golang TLS library and added support for the DANE client ID extension in TLS 1.3 (client and server sides). Moreover, we adapted the implementation to add support for TLS 1.2, where the extension was added in the client Hello message allowing the client id to be sent in it. We chose to implement in a LoRaWAN network by allowing mutual authentication between two backend servers. LoRaWAN stands for Long Range Wide Area Network, one of several Low Power Wide Area Network (LPWAN) protocols. We have chosen LoRaWAN since it fits the criteria we seek: constrained end-devices in the radio space and several IP-enabled backend servers in the IP-space. These backend servers communicate with each other and could benefit from mutual authentication. Another reason for choosing LoRaWAN is its low cost and ease of deployment compared to other IoT LPWAN technologies. The source code is available at *https://gitlab.rd.nic.fr/dance*.

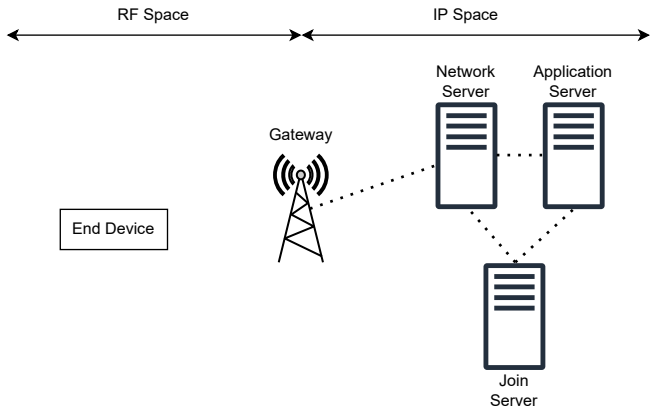

Fig. 6. LoRaWAN-architecture

The basic architecture of a LoRaWAN network is depicted in Fig. 6. LoRaWAN end-devices are constrained radio transceivers that send data to backend servers or receive data from backend servers. The backend servers are IP-enabled servers and are separated from the radio frequency space via a gateway which sits between the radio and IP spaces. The set-up is demonstrated in Fig. 7. Usually in a LoRaWAN network, if a device wants to join the network, it will send a *join request* that the gateway will receive. The gateway forwards the *join request* to the Network Server, which then connects to the Join Server to verify it. The *join accept* is then sent back to the gateway, which delivers it to the end-device.

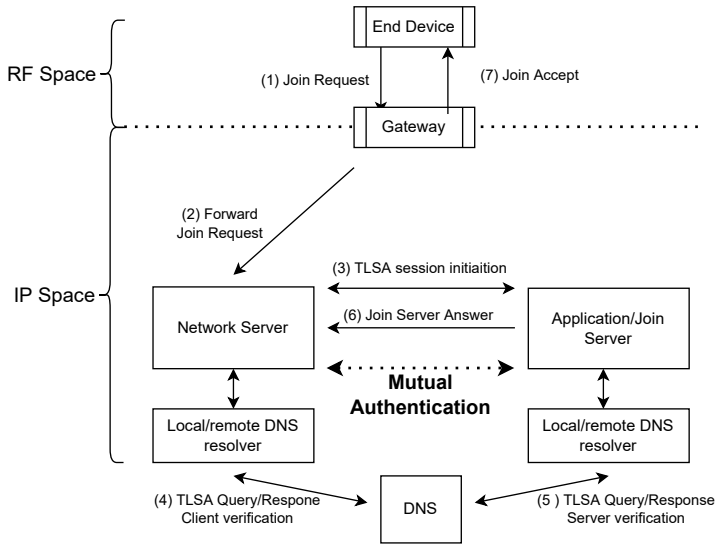

Fig. 7. System Model

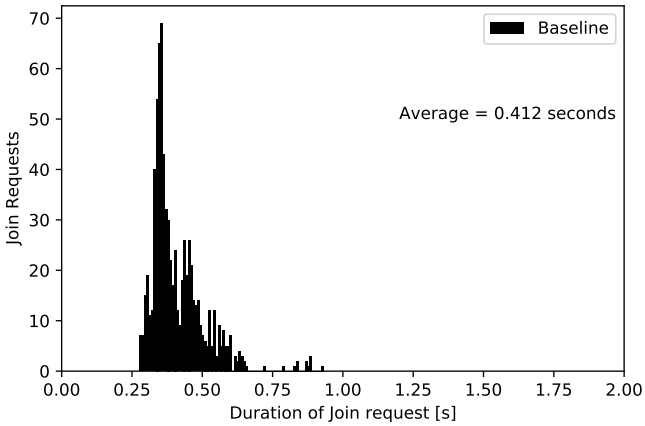

Fig. 8. Duration of Join Requests (Baseline)

Before the Network Server contacts the Join Server, the two servers perform mutual authentication via a TLS handshake that includes a certificate exchange. In our implementation, each server verifies the other's certificate using DNS. Each server would have already added a TLSA RR of its certificate in its DNS zone. The TLSA RR of one server is retrieved by the other to verify the certificate it received. Several scenarios are tested to study the effect of using DNS.

For our experiments, the end-device, Gateway, Network Server and Application/Join Server are located in France, and we vary the location of the DNS resolver to study the effect it has on performance. We start with a local resolver on the Network Server and the Application/Join Server and study the difference between enabling caching and not enabling it. We also study the case of remote resolvers using Google's public DNS resolver (8.8.8.8) and a resolver we set up in Singapore. Singapore was chosen to maximize the distance between the DNS clients and the resolver. These scenarios are compared to the baseline case of not using DANE for certificate verification and instead using a single CA. The single CA, in our case, is our company that owns both servers. It has a self-signed certificate which allows it to issue certificates for both servers and act as a trusted CA for both. Hence, both servers will trust it and the certificates it issues, allowing mutual authentication to happen.

IV. Evaluation

In this section, we provide the results of the measurements we did in various scenarios: A baseline case to which we compare the results of our implementation, and then several scenarios of mutual authentication via DANE but with a varying configuration and location of the recursive resolver. We start with a local resolver on each server and study the effect of caching. We then move to use a remote resolver. The remote resolvers we used are Google's public DNS resolver (8.8.8.8) and a resolver in Singapore. The metric we are considering is the time elapsed starting when the gateway receives the *join request* from the end-device and until it receives back the *join accept* from the Network Server. We study how using DANE for mutual authentication affects the duration of the *join request*. In each scenario, the end-device sent a *join request* every 10 seconds for 7 hours. However, due to duty cycle restrictions (1% in our region), not all join requests were successful. The baseline case to which we compare our results is the case where DANE is not used, but instead, authentication is done by using a CA. The CA in the baseline case is our company which has a self-signed certificate which allows it to issue certificates to the servers. While mutually authenticating, the servers exchange and verify each other's certificates because both servers trust the CA.

Figure 8 is the histogram of the duration of successful join requests in the baseline case where DANE is not used. The average *join request duration* is around 0.412 seconds.

A. Using a Local DNS Resolver

The first scenario we tested included using a local resolver on each server. A resolver is installed separately on the Network Server and on the Application/Join Server. Each server is configured to use its local resolver when retrieving the TLSA RR of the server it is authenticating. We ran the test and did our measurements once without enabling caching at the resolver and another with caching enabled with 20-minute TTL values on both servers. In the context of DNS, caching allows the resolvers to save the query responses and to reply directly to such requests (i.e. without repeating the whole DNS query process). The responses stay in the cache according to their TTL values. Resolvers will initiate a recursive resolving process to fetch a RR if this RR is not in the cache or if it is but has been there for more than the TTL value specified in it. Figure 9 is the histogram of the duration of successful join requests in the case where a local DNS resolver

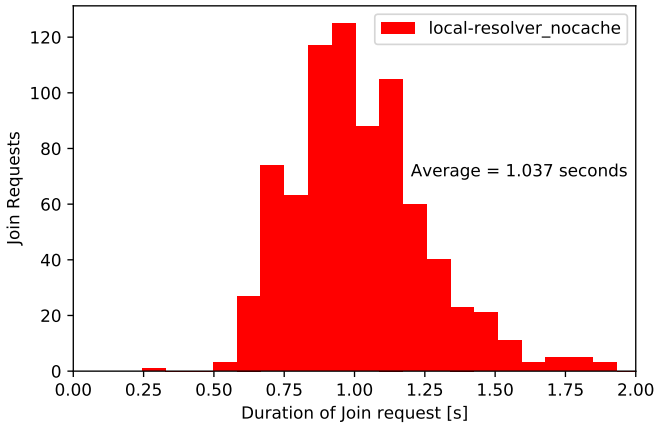

Fig. 9. Duration of Join Requests (Local resolver without caching)

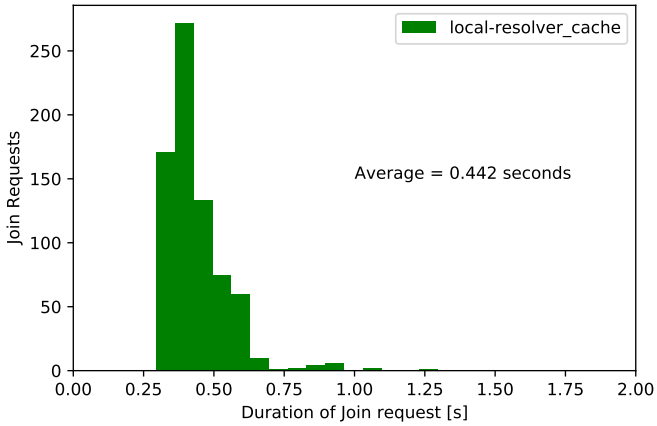

Fig. 10. Duration of Join Requests (Local resolver with caching)

is used. Caching is not enabled at the resolvers; hence, the resolvers will have to recursively go through the DNS query process to fetch the TLSA RR every time. This resulted in an increased delay, and the *average join request duration* went up to around 1.037 seconds. This delay went down again to around 0.442 seconds after caching was enabled on both servers with a TLSA RR Time To Live (TTL) of 20 minutes. This is demonstrated in Figure 10.

Figures 9 and 10 show that using DANE for mutual authentication does introduce a significant delay to the joining process as the *average join request duration* more than doubled and went from 0.412 seconds for the baseline case to 1.037 seconds when using a local resolver without caching. However, the caching at the recursive resolver reduced the latency, and the *average join request duration* went down to 0.442 seconds.

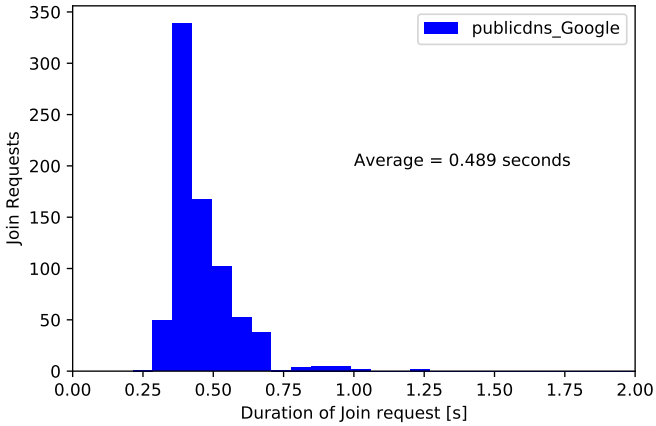

Fig. 11. Duration of Join Requests (Remote DNS Resolver - Google)

hist-singapore.pdf

Fig. 12. Duration of Join Requests (Remote DNS Resolver - Singapore)

B. Using Remote DNS Resolvers

The caching proved useful in decreasing the latency introduced from the DNS resolving process even if the resolvers were remote. Figure 11 is the histogram of the duration of successful join requests in the case where the resolver is Google's public DNS resolver (8.8.8.8). The *average join request duration* is around 0.489 seconds. Figure 12 is the histogram of the duration of successful join requests in the case where the resolver is in Singapore. The *average join request duration* is around 0.887 seconds.

Figures 11 and 12 demonstrate the importance of caching to reduce the latency introduced by using DANE. The *average join request duration* in both cases (0.489 seconds for Google public DNS and 0.887 seconds for Singapore DNS resolver) is less than the *average join request duration* when using a local resolver but without caching.

V. Conclusion

Mutual authentication in IoT environments is crucial to enhance the overall security of networks. The current PKIX is prone to security risks and could benefit from using DNS to reinforce it. DNSSEC guarantees the integrity and authenticity of answers received via DNS. In this paper, we used DANE to perform mutual authentication between LoRaWAN Network and Join Servers. We evaluated the performance by measuring the time elapsed between *join requests* and *join accepts* at the level of the gateway. The implementation showed that DNS could be used for authenticating certificates by adding appropriate TLSA RRs in the DNS zone of the domain

whose certificate is to be verified. The performance evaluation showed that DNS seems to add latency to the authentication process, but that could be overcome by caching, which can achieve latency values close to the baseline case. The next step is to extend the mutual authentication process to the end-device and have it perform mutual authentication with the Application Server, for example. This is not straightforward for constrained IoT devices as the exchange required to perform this authentication is much larger than the MTU (Maximum Transfer Unit) values in the radio space. Compression and fragmentation techniques could be beneficial. Moreover, this paper did not take into account the transport protocol used during DNS resolution. Some DNS-encrypting protocols like DNS-over-TLS and DNS-over-HTTPS preserve the privacy of DNS requests and responses, but use TCP, which introduces additional latency.

REFERENCES

[1] Chris Kemmerer, "What is a root store?" 2019. [Online]. Available: https://www.ssl.com/faqs/what-is-a-root-store/

[2] Google Security Blog, "An update on attempted man-in-the-middle attacks," 2011. [Online]. Available: https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html

[3] S. Balakrichenan, A. Bernard, M. Marot, and B. Ampeau, "IoTRoam: design and implementation of an open LoRaWAN roaming architecture," in *IEEE Global Communications Conference (GLOBECOM)*, Madrid, Spain, Dec. 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03100628

[4] E. Rescorla and T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5246

[5] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[6] European Union Agency for Cybersecurity (ENISA), "Operation black tulip: Certificate authorities lose authority," 2011. [Online]. Available: https://www.enisa.europa.eu/media/news-items/operation-black-tulip/

[7] B. Corbitt, "Do you "trust" me?" 2013. [Online]. Available: https://www.intersecworldwide.com/blog/do-you-trust-me

[8] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, "DNS Security Introduction and Requirements," RFC 4033, Mar. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4033

[9] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, "Resource Records for the DNS Security Extensions," RFC 4034, Mar. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4034

[10] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, "Protocol Modifications for the DNS Security Extensions," RFC 4035, Mar. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4035

[11] P. E. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," RFC 6698, Aug. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6698

[12] Ólafur Gumundsson, "Adding Acronyms to Simplify Conversations about DNS-Based Authentication of Named Entities (DANE)," RFC 7218, Apr. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7218

[13] V. Dukhovni and W. Hardaker, "The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance," RFC 7671, Oct. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7671

[14] "Dane authentication for network clients everywhere (dance)." [Online]. Available: https://datatracker.ietf.org/wg/dance/about/

[15] S. Huque, V. Dukhovni, and A. Wilson, "TLS Client Authentication via DANE TLSA records," Internet Engineering Task Force, Internet-Draft draft-ietf-dance-client-auth-00, Mar. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-dance-client-auth-00

[16] S. Huque, V. Dukhovni, and A. Wilson, "TLS Extension for DANE Client Identity," Internet Engineering Task Force, Internet-Draft draft-ietf-dance-tls-clientid-00, Mar. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-dance-tls-clientid-00