TLS/PKI Challenges and Certificate Pinning Techniques for IoT and M2M Secure Communications

Daniel Díaz-Sánchez, *Senior Member, IEEE*, Andrés Marín-Lopez, *Member, IEEE*, Florina Almenárez Mendoza, *Member, IEEE*, Patricia Arias Cabarcos, *Member, IEEE*, and R. Simon Sherratt, *Fellow, IEEE*

***Abstract*—Transport layer security (TLS) is becoming the *de facto* standard to provide end-to-end security in the current Internet. IoT and M2M scenarios are not an exception since TLS is also being adopted there. The ability of TLS for negotiating any security parameter, its flexibility and extensibility are responsible for its wide adoption but also for several attacks. Moreover, as it relies on public key infrastructure (PKI) for authentication, it is also affected by PKI problems. Considering the advent of IoT/M2M scenarios and their particularities, it is necessary to have a closer look at TLS history to evaluate the potential challenges of using TLS and PKI in these scenarios. According to this, this paper provides a deep revision of several security aspects of TLS and PKI, with a particular focus on current certificate pinning solutions in order to illustrate the potential problems that should be addressed.**

***Index Terms*—Transport layer security, DTLS, public key infrastructure, trusted third party, certificate pinning, Internet of Things, machine to machine.**

I. Introduction

IN MANY senses IoT/M2M technology is mature, but there is a lack of technical [1] and regulatory [2] consensus concerning security. Concerned by this lack of security and the increasing population of devices, the U.S. Federal Trade Commission (FTC) organized a workshop in November 2013 and reported in January 2015 [3] on the major concerns on device security: APIs, authentication, and update processes. As discussed in the report, IoT devices have different sizes, shapes and purposes, but they share a set of differentiating attributes from other technologies that demand special attention from a security perspective. The majority of IoT devices are furnished with more or less limited processing power that, considering economies of scale, is managed by a similarly constrained operating system. The operating system, typically Linux, can be reprogrammed to overstep the original device purpose. Moreover, this can happen without user knowledge or consent since rarely these devices have a monitoring system that helps to realize changes in software or network configuration and unusual connections.

The re-use of hardware platforms, drivers and development environments allows a vulnerability found in a device, that can be easily tampered with, i.e., a smart wristband, to be exploited in a big population of devices using similar hardware or software as, for instance, a car with some kind of infotainment. Insufficient security analysis for IoT devices and apps may lead to security risks for unexpected use cases.[1] Additionally, the decrease in hardware, software, development and production costs may motivate companies with no previous security experience, to introduce potentially vulnerable devices in the market. It should be also considered that re-using hardware and software should not be a problem, but a benefit, on its own, since vulnerabilities in components can be detected and fixed in less time. However, there is an enormous disparity regarding the support and update of IoT devices. Thus, support and update are critical in IoT/M2M as outdated devices can be the way in to million of homes, companies, critical facilities, and other devices.

The micro-services architecture [4] has been proposed over time to alleviate updates. It is an architectural style that structures any application as a collection of loosely coupled services implementing the application functionality. Beyond its ability to split a complex application into small pieces and put all together when needed, the goal of micro-services is that every component can be independently instantiated and updated. This favours continuous delivery [5] and continuous deployment of complex distributed applications. Basically, it eases maintenance and development operations [6] as well as it improves agile development [7].

Manuscript received April 23, 2018; revised September 28, 2018 and March 8, 2019; accepted April 29, 2019. Date of publication May 2, 2019; date of current version November 25, 2019. This work was supported in part by Project MAGOS under Grant TEC2017-84197-C4-1-R, in part by Project INRISCO under Grant TEC2014-54335-C4-2-R, and in part by the Comunidad de Madrid (Spain) through the Project CYNAMON co-financed by European Structural Funds (ESF and FEDER) and the Alexander von Humboldt Post-Doctoral Program under Grant P2018/TCS-4566. *(Corresponding author: Daniel Díaz-Sánchez.)*

D. Díaz-Sánchez, A. Marín-Lopez, F. A. Mendoza, and P. A. Cabarcos are with the Department of Telematic Engineering, Universidad Carlos III de Madrid, 28911 Madrid, Spain (e-mail: dds@it.uc3m.es; amarin@it.uc3m.es; florina@it.uc3m.es; ariasp@it.uc3m.es@it.uc3m.es).

R. S. Sherratt is with the Department of Biomedical Engineering, University of Reading, Reading RG6 6AH, U.K. (e-mail: r.s.sherratt@reading.ac.uk).

Digital Object Identifier 10.1109/COMST.2019.2914453

[1] *Fitness tracking app Strava gives away location of secret U.S. army bases*, The Guardian, World edition, 28th jan 2018.

Fog computing or Fog, is an evolution of the cloud computing model [8], in which resources are moved to the edge of the network or beyond [9]. The advent of Fog technology, and Mobile Edge Computing, allows extending the concepts of Cloud Computing to the network edge. Also Network Function Virtualization (NFV) and other embodiments [10] share similar objectives. Nevertheless, the geographic distribution, proximity to consumers and support for high mobility rates, are fundamental features of Fog/Edge computing needed for a consistent IoT/M2M development. Conservative estimations calculate that IoT/M2M ecosystem will contribute with more than 50 billion devices [11] considering personal devices, sensors and actors to give support to concepts as Smart Cities, Smart Metering [12], Wereable Computing [13] and Crowd Sensing, among others.

IoT/M2M applications can benefit from the use of micro-services. Declarative and asymptotic strategies [14] would allow these devices to have a minimum operating system and perform a declaration to request near computing resources, as Fog, to instantiate certain micro-services. Since micro-services are software components in constant development and revision from manufacturers and published in repositories, devices could get the most recent images of the micro-services ready at run time. In this way, update problems would be minimized.

As discussed, IoT/M2M presents several security problems that should be addressed and that can be sometimes reduced by means of micro-services. However, IoT devices will have constant communication with Cloud Computing, Edge Computing, or Fog Computing infrastructures depending on the purpose of the communication. Among these purposes, it is possible to find communications between components of distributed applications or just communications serving different data-to-cloud strategies. In most of the cases, the biggest amount of traffic is expected to be concentrated in the vicinity of the devices, fruit of the cooperation among application components (micro-services) and/or devices and between devices and Fog/Edge/Cloud Computing backends. However, despite less numerous, interactions between devices and backends for data consolidation will be frequent.

The traffic generated among devices and services that will be transported by these protocols, may contain personal or critical information and should be adequately protected. In fact, these protocols are required to support, at least, service authentication and confidentiality. Moreover, it may also be critical to provide support for micro-service dynamic authentication, since many of these services will be instantiated on application request.

Considering the scale of the problem, it is necessary to provide adequate protocols that let devices fulfil their purpose securely, requiring no centralized management.

The vast majority of proposed IoT/M2M protocols focus on solving concrete problems aside from security. IoT and M2M have inherited the use of Web services or APIs according to the Representational State Transfer (REST) [15] architecture. However, HTTP and TCP are not suitable for resource constrained (limited) devices as they require keeping state in both endpoints and HTTP presents a significant overhead. For that reason, the activity of the Constrained RESTful Environments (CoRE) IETF team concentrates on providing an adequate RESTful architecture proposing 6LoWPAN [16] and Constrained Application Protocol (CoAP) [17], [18]. Thus, the major goal of 6LoWPAN is to allow constrained devices to use IPv6 by simplifying the device requirements, whereas allows them using an immense address space for a better adoption of IoT/M2M.

CoAP allows both unicast and multicast restful communications for IoT/M2M. CoAP relies on UDP as transport protocol permitting asynchronous message oriented interactions with a very low overhead and supporting proxies and caches. It defines a messaging model over UDP with a very small header providing TCP-like reliability with optional message confirmation. For the supported communication patterns, which can be one to one or one to many, CoAP allows applications to enable Automatic Repeat-reQuest according to the conditions but provides transaction identifiers independent from message identifiers for an improved flexibility.

When it comes to security, these protocols usually rely on Transport Layer Security (TLS) [19], or its datagram version (DTLS) [20]. In fact, HTTP, CoAP, Quick UDP Internet Connections (QUIC) [21], among other applicable protocols in the context of IoT/M2M [22] use TLS or DTLS for confidentiality and authentication [23].

Since TLS relies on Public Key Infrastructure (PKI) [24], [25] for authentication, it is necessary to have a closer look at the history of TLS/DTLS to evaluate the challenges of using TLS and PKI in IoT/M2M environments. Therefore, this article revises several aspects of TLS/DTLS and PKI, with a particular focus on Certificate Pinning to illustrate the potential problems that should be addressed for a secure inclusion of IoT and M2M for our daily lives.

A. Article Organization

Considering the need of evaluating the challenges and also considering the dependency of IoT/M2M end-to-end protocols on TLS, this article performs such a evaluation as follows.

This article describes the evolution of the TLS protocol in detail in Section II, addressing: TLS handshake and its latency in Sections II-A and II-B; problems TLS has faced due to protocol, cypher suite or compression mechanism design attacks in Section II-C. Since PKI is, nowadays, one of the cornerstones of TLS, this article makes a deep revision of the current trust problems of PKI that can affect TLS in Section III and the evolution of the TLS security over the time in Section III-B considering both PKI and vulnerability related problems.

The need for certificate pinning is reasoned in Section IV. Despite some research has already coped with security in IoT and M2M [26]–[28] and performed Certificate Pinning techniques comparisons [29], [30], this article not only explores current solutions to the problems of trust, impersonation attacks [31], and lack of auditing, but also focuses on their use in IoT/M2M scenarios. Certificate Pinning techniques are described in Sections IV-A–IV-F and compared in Section V discussing their viability for IoT/M2M scenarios. Finally, open challenges and research directions are discussed in Section VI and conclusions in Section VII.

II. Transport Layer Security

This section discusses the interest of TLS [19] for end-to-end protection in IoT.

TLS was designed to provide confidentiality and integrity to end-to-end communication. IPSEC [32] is also a remarkable end-to-end security protocol due to its penetration in security solutions. IPSEC can provide confidentiality among network nodes relying on key exchange [33], or using pre-shared keys. It can also be used as authentication protocol by means of the "Authentication Header" providing end-to-end security equivalent to TLS, in fact, this is the default mechanism in IPv6.

IPSEC, despite relegated in practice to the establishment of tunnels and Virtual Private Networks (VPN), is an excellent protection mechanism that can provide the same services than TLS provides, or complement other protocols bringing better security. It should be considered, that despite end-to-end security is a necessary requirement for secure communications, it is not the only one. Behringer [34] argued that the end-to-end security provided by TLS is perceived as enough in general, but network security is also necessary and thus IPSEC, among others. Controlling malicious activities from endpoints, monitor and cryptographically isolate certain links, and protect against IP spoofing are several tasks that should be considered beyond end-to-end security.

Other protocols, e.g., Kerberos, do not provide authentication on their own, but communicate authorization decisions generated by other services as SAML [35]. This also requires applications to be designed considering the protocol. There are many other protocols that allow mutual authentication [36]–[38], including TLS, that require at least one of the parties involved in the transaction, typically the server, to disclose its identity, so they do not provide full privacy to participants.

Thus, being conscious that there are many other protocols that enable authentication in distributed environments, even providing full privacy, this article deals with the current state of TLS and the involved PKI usage for accessing online services by IoT devices. The reason for analysing TLS is that this is the only end-to-end protocol that can be considered globally accepted and typically requires the authentication of one of the endpoints, thus the lack of server privacy is not considered.

TLS is an excellent tool for establishing secure connections in IoT environments considering that many of the connections will be opportunistic such as those concerning service discovery or name resolution, needed for accessing local computing resources [23]. Despite TLS used in combination with PKI requires IoT devices to handle PKI certificates, which can be resource consuming, it provides a good versatility. TLS provides confidentiality, authentication and allows the negotiation of almost every security parameter. Both the client application and the service can actively participate in the negotiation using user space libraries. Due to these reasons, TLS has been adopted by several transport protocols as mentioned before.

TLS was originally designed to work on top of a TCP/IP stack and thus, it is connection oriented. However, TLS has been complemented with versions that, using the same security negotiation mechanism, work over UDP [20], and even SCTP [39].

Moreover, the specification of TLS describes an extension mechanism [40]–[42] to support new functionalities. TLS extensions are the preferred mechanisms to add new functionality that were not initially considered by the protocol. Extensions add additional information to the handshake messages augmenting the negotiation capabilities of the protocol whereas keeping compatibility with older versions. To achieve that goal, TLS endpoints ignore extensions they do not understand.

The following sections will discuss the recent versions of TLS [19] and DTLS [20]. DTLS re-uses TLS sub-protocols and handshake messages. It just adds the necessary resiliency to UDP (loss and duplicate datagram management) to serve as a secure UDP transport for other protocols like TLS does over TCP. For that reason, the rest of the article will make no distinction among them unless necessary.

A. Handshake in TLS

TLS provides a secure connection over transport protocols with optional server-only or mutual authentication. To create the secure channel, TLS performs a key exchange during the handshake to derive a secret key to protect the channel. RSA static is the oldest and simpler mechanism for key exchange. It has been available since the earliest versions of SSL. In this key exchange mechanism, shown in Fig. 1, the client generates a "pre-master" key, encrypts it with the public key of the server (whose certificate has been previously delivered to the client using the "Certificate" message), and then sends the encrypted "pre-master" key to the server using the "ClientKeyExchange" message. In this way, the protocol manages to exchange a key with the server in a secure way. The server decrypts the "pre-master" using its private key. This proof of possession of the private key provides the server authentication.

The exchange of the "Finished" message triggers the verification of the integrity of previously exchanged messages that would fail if the server could not decrypt the "ClientKeyExchange" message.

The major concern with RSA Static is that it cannot guarantee the concept of "forward secrecy" [43], thus it cannot guarantee past communications will be confidential in the future [44]. This happens with any key exchange mechanism in which the long-term secret used to protect the communication is a shared key that, despite encrypted, is delivered through the network. Basically, a passive attacker can store the encrypted key exchange together with the encrypted traffic waiting to break or steal the server private key. If the attacker manages to get the key, he can decrypt every previously recorded and every forthcoming session protected with that private key.

Diffie Hellman [44] (DH) provides forward secrecy and can be applied to TLS as shown in Fig. 2. In DH, the server sends a "ServerKeyExchange" message after the server certificate, that contains the DH parameters or an elliptic curve calculated by

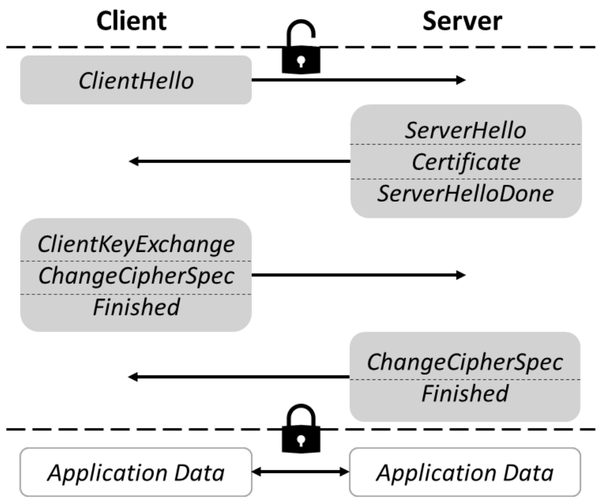

Fig. 1. TLS handshake with RSA Static. The client sends a "ClientHello" message containing a list of the supported key exchange, cipher suites and compression mechanisms so the server can enforce its selection with the "ServerHello" message. The server delivers the server certificate and finishes the negotiation using "Certificate" and "ServerHelloDone" messages respectively. The "ClientKeyExchange" message contains the "pre-master" key encrypted with the server public key so only the server can decrypt the "pre-master" key and derive the master key. The client also delivers "ChangeCipherSpec" and "Finished" messages indicating it has derived the master key from the "pre-master" key and forthcoming traffic must be protected with the master key, and so acknowledges the server by delivering "ChangeCipherSpec" and "Finished" messages.

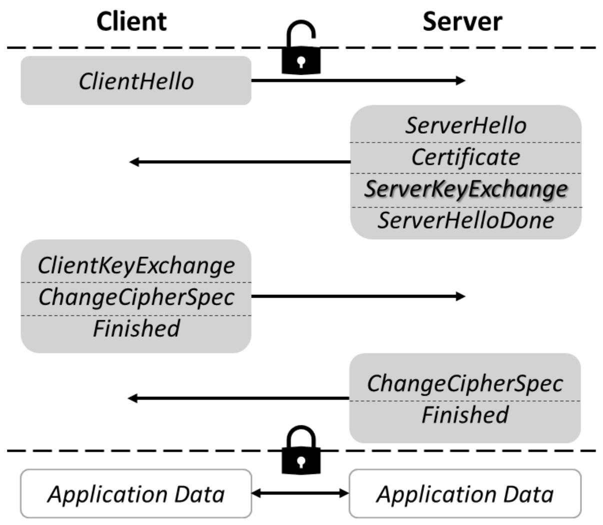

Fig. 2. TLS handshake with Elliptic Curve Diffie Hellman (ECDH). After the negotiation with the hello messages, the server delivers its certificate. The key exchange is then initiated by the server with the "ServerKeyExchange" message that contains the ECDH key material, so the client can derive an ephemeral key. The "ServerKeyExchange" contains also a signature with the server key that verifies with the certificate delivered in the "Certificate" message. The client can derive an ephemeral key and deliver it to the server using the "ClientKeyExchange" so both endpoints can derive a master key according to DH. Alike the handshake presented in Fig. 1, no encrypted key is delivered during the handshake enabling forward secrecy.

the server. This information is paired with an ephemeral public key generated by the server. The client generates also an ephemeral key compatible with the server key and delivers it to the server. In this way, both endpoints can derive a long-term shared secret avoiding this shared secret to be encrypted and sent over the network. Beyond key exchange, the authentication is achieved in this case with the server signature over the parameters in the "ServerKeyExchange" message, thus the client can verify the signature against the server certificate.

B. Improvements in TLS Handshake Latency

The TLS handshake requires two Round Trip Time (RTT) delays to finish. From Fig. 1 and Fig. 2, the reader should note the number of TLS messages to be exchanged among endpoints does not depend on the key exchange or authentication mechanism. This handshake time, together with the TCP handshake (do not apply to DTLS), can be a considerable long time for devices that demand a fast interaction, as those delivering bursts of data while moving. This may happen frequently in several scenarios in IoT and vehicular networks. To improve the protocol agility, some specifications define abbreviated "session resumption", so devices can resume previously established sessions with TLS. The originally abbreviated handshake specification [19] used session identifiers managed by the server. Other specifications allow the client to store "session tickets" that can be redeemed later [45] preventing the server from storing client state.

Nevertheless, negotiation in TLS can have an important impact on protocol efficiency depending on the transport protocol and the selected cipher suites. For instance, IoT/M2M devices, especially those using constrained radio interfaces, should observe that packets larger than the Maximum Transfer Unit (MTU) are fragmented increasing latency and energy expenditure. There are several attacks that can force devices to fragment data under certain circumstances [46]. Moreover, besides cipher suite selection, the negotiation of compression can be also critical to avoid attacks, as will be discussed in Section II-C, but also to improve efficiency [47].

The aforementioned abbreviated handshake alleviates the problem of resuming a secure channel between entities in IoT. For instance, several applications require devices to update a given resource or to get information periodically. However, several other applications in IoT/M2M may require to perform requests to resources that will not repeat again during a reasonable period of time. Thus, the ability of resuming previous sessions adds no benefit to these applications. For that reason, there is a need for optimizing TLS handshake by reducing its latency in any case.

Recent TLS versions incorporate several improvements [48] as removing the use of RSA Static to improve forward secrecy, and reducing the handshake to 1RTT or even 0RTT, depending on the case. TLS has several layered sub-protocols that manage TLS functionality internally. The "ChangeCipherSpec" sub-protocol was in charge of signaling the other part the forthcoming messages should be delivered encrypted with the session key. This sub-protocol triggered the verification of the handshake messages to verify integrity. This sub-protocol can close the connection if handshake messages were manipulated, thus there is a chance to perform a Denial of Service attack. Basically, it is necessary to wait until the

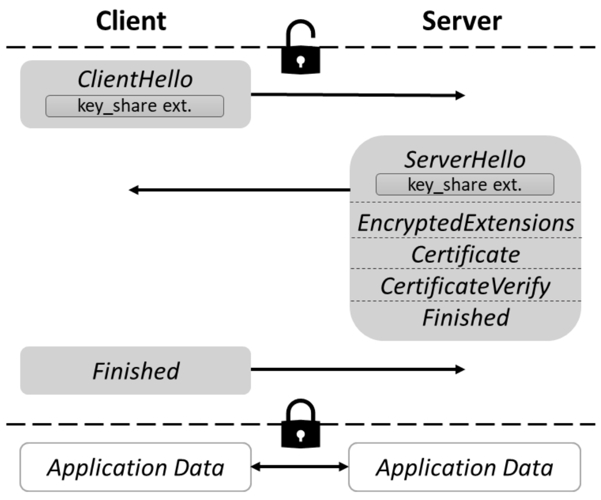

Fig. 3. TLS 1.3 handshake 1RTT. In this simplified TLS handshake, "ServerKeyExchange" and "ClientKeyExchange" messages have been removed, so DH parameters for key exchange, public keys and pre-shared key labels are delivered in a extension called "key_share". More precisely, if (EC)DHE key establishment is in use, the client sends a list of named DH groups within the extension and the server should be in one of the groups of the client share. If so, after the "ServerHello" message, the server delivers also a "key_share" containing the server's ephemeral key. "EncryptedExtensions" and "CertificateRequest" contains responses to client extensions (if any) and a request for client authentication (if mutual authentication is enabled) respectively. Also, a new message called "CertificateVerify" is used for server authentication that contains a signature over the handshake messages exchanged so far.

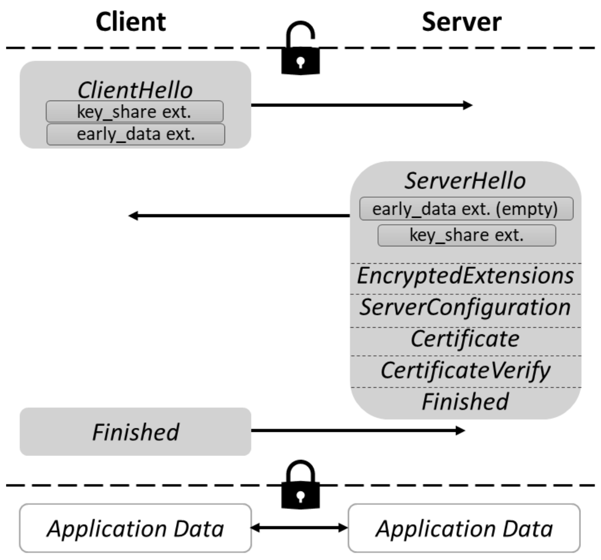

Fig. 4. TLS 1.3 handshake 0RTT. If client and server share a PSK (from a previous handshake or by other means), clients can deliver encrypted data in the first message using the "early_data" extension.

"ChangeCipherSpec", several RTTs after the start of the handshake, to realize the attack. Due to that, this sub-protocol has been removed so any message after "ServerHello" should be encrypted. In this way, active adversaries, manipulating handshake messages, can be blocked sooner.

Moreover, "ServerKeyExchange" and "ClientKeyExchange" have been substituted by the extension "KeyShare" for key exchange in recent TLS versions, as shown in Fig. 3. The rest of the messages are kept in the same order as the original protocol, and the server authentication is performed by signing the previous handshake messages. The signature is placed in the message "CertificateVerify" to keep TLS backwardly compatible.

Recent TLS versions also propose the 0RTTs handshake, presented in Fig. 4, that is equivalent to the standard handshake, shown in Fig. 3, with the exception it delivers client data in a extension called "early_data". In such a way, the application protocol over TLS, can use that extension to push the request to the server during the handshake, so once the TLS secure channel is created the response can be sent. Older versions needed to wait until the handshake has finished to send the request to the server, increasing the latency. This reduced handshake enforces every message after "ClientHello" to be encrypted with a secret derived from the client secret in a "keyShareEntry" (an entry of the "key_share" extension), and requires the server to advertise DH semi-static parameters.

It should be noted that despite the upper protocol requests can be delivered directly as part of the handshake, the security properties of TLS are reduced with the 0RTT handshake [49]. Reply attacks are possible since the server should incorporate random data to avoid these kind of attacks in the first message, but in this case, they are delivered after the "ServerHello" message. Moreover, the use of semi-static DH parameters by the server dare the principle of "forward secrecy" since, at least, 1RTT is necessary to establish the ephemeral secret. Thus, the first client message may not meet that principle. Server DH semi-static parameters should be known to the client before the handshake, due to previous interactions or by any other means, as discovery protocols [50]. In any case, it is recommended to limit their validity to a week, so any related attack window of opportunity is reduced.

C. Security Considerations

TLS and its predecessors (SSL) have suffered two different kind of attacks. The first is based on the protocol conceptualization and its structure. The second, based on the Public Key Infrastructure, is not directly attributable to the protocol but affects it since PKI is an important part of TLS. The problems PKI brings into TLS are explained in Section III and the current solutions are also explained in later sections.

TLS not only provides confidentiality and optional authentication, but also protects against downgrade attacks willing to enforce previous (non secure) TLS versions or the use of a weak protocol. Moreover, TLS provides message authentication and integrity. This section describes the most important attacks related to TLS conceptualization, implementation and structure. Some of them are documented by the IETF [51] and others are individually described in the literature. A sample of the most relevant attacks will now be presented.

The re-negotiation attack was discovered in 2009. It allowed to perform a plain text injection in SSL 3.0 using the protocol re-negotiation. Basically, the attacker was not able to decrypt messages but to inject its own requests at the beginning [52].

It was solved including a handshake message verification during re-negotiation [53].

Browser Exploit Against SSL/TLS (BEAST) [54], discovered in 2011, allowed an attacker to circumvent the same origin policy (prevents a page script to contact different pages except both are from the same domain) in TLS 1.0. It was solved in the following TLS version. During the meantime, its was proposed to use RC4 as stream cypher since was immune to BEAST. Unfortunately, in 2013 a vulnerability was discovered advising implementers against using RC4.

RC4 was not free of attacks before. It was immune to BEAST since TLS allowed using RC4 only under certain circumstances that limited its use. The RC4 vulnerabilities of 2013, found statistical deviations in the algorithm that made it inadvisable for preventing BEAST [55]. Later on, it was discovered the possibility to recover plain text after observing big TLS traffic and due to that, RC4 was permanently forbidden in TLS [56].

Compression Ratio Info-leak Made Easy (CRIME) [57], discovered in 2012, allowed an attacker to find plain text messages exploiting padding and compression, thus it was possible to steal authentication cookies. This attack did not affect TLS exclusively, but affected also SPDY, HTTP and others. Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH) [58], that was based on CRIME, was presented in 2013. BREACH permitted attackers to extract sensitive information, including authentication information, after observing certain data. That allowed attackers to redirect the victim to malicious sites or even inject content in the Web pages being accessed through the encrypted channel. Whereas the protection against CRIME was possible eliminating TLS compression and SPDY headers, current TLS implementations are still vulnerable to BREACH, since it is not feasible to eliminate compression from application protocols.

In 2013, an attack was presented that enabled attackers to block logout messages by injecting a TCP termination message (TCP FIN) without the knowledge of the victim [59]. To achieve the result it was not necessary to infect the user machine but to compromise a hotspot or any other network element in the path.

The first versions of SSL were vulnerable to the "padding oracle attack" discovered in 2002, that allowed using a server (oracle) to find out if padding was correct or not, allowing to decrypt messages with the server key without its knowledge. The attack was feasible using CBC. In 2013, a variant of the padding attack, called Lucky Thirteen [60], allowed breaking the message authentication in TLS analysing the time spent in encryption (timing side-channel attack). It was solved with an extension to the TLS specification [61]. Padding Oracle On Downgraded Legacy Encryption (Poodle) was presented in 2014, showing how Cipher Block Chaining (CBC) in SSL3.0 is vulnerable to padding attacks. Despite the majority of the servers use, at least, TLS 1.0, the attack required forcing the use of SSL3.0 as a fall-back, frequently supported until 2015 [62].

In 2014 and 2015 two critical vulnerabilities, known as HeartBleed and BERSerk, that affected OpenSSL and other widely adopted implementations, were presented. Heartbleed allowed attackers to exploit a bug for extracting data from servers; BERSerk exploited a bug in ASN.1 that permitted man-in-the-middle attacks in several implementations.

Factoring RSA Export Keys (FREAK), identified in 2015, exploits an old restriction to SSL/TLS exportation introduced by the government of the United States. This restriction limited the size of RSA keys to 512 bits. In 2010 it was demonstrated that breaking short RSA keys was simple enough to become a problem. FREAK relied on a downgrade attack. It forced the victim to use an old abandoned version of SSL supporting the restriction. The attack consisted on influencing the cipher suite negotiation to enforce the use of weak algorithms [63]. Logjam, also from 2015, was similar to FREAK but enforcing an old restriction related to Diffie Hellman.

Decrypting RSA with Obsolete and Weakened eNcryption (DROWN) [64], announced in 2016, could be used to attack servers using a combination of versions instead of a concrete one. It used an adaptive-chosen-cipher text attack combined with a downgrade to SSLv2 (that was still supported by many servers). The attack was helpful in reducing the effort needed for a man-in-the-middle attack. It could be estimated that the 33% of the servers in 2016 were affected.

III. TLS/PKI Problems and Security Evolution

TLS is becoming the preferred security protocol in modern IoT/M2M protocols to authenticate services and protect the communication with them. TLS supports several authentication mechanisms beyond PKI, as pre-shared keys, but in general services rely on PKI. Thus, services are bound to domain names and those domain names are tied to an X.509 certificate to authenticate the service. In few words, we expect an X.509 certificate that bounds a domain name to a public key pair, to be issued by a trusted certificate authority. In practice, PKI certificates contain a extension ("SubjectAltNames") [25] with the domain name(s) in which this certificate can be used.

The major concern with x509 certificates is that they were not designed for the concrete purpose of authenticating domain names. Due to that, the *subject* field (the entity to whom it is issued) is an X.500 directory name [65], [66] and not a domain name.

X.500 [67], [68] and X.509 [69] are related so services, people and other entities are described in directories and certificates are bound to directory entries. Thus, directories can be used to find entities that will be eventually authenticated using an X.509 certificate with the appropriate *subject* name. Moreover, certificates could be fetched from directories. So within an organization, directory entries and certificates have a univocal relation.

This distinctive feature in X.509 certificates names creates two fundamental problems [70] that affect certificate validation during a TLS handshake. First, PKI defines a hierarchy but there is no single root under every certificate can be validated. In contrast, there are several independent roots available with their own hierarchy. In such a way, well-known root Certification Authorities (CAs) are incorporated to the client application by the software manufacturer or are kept as part of the operating system. That list should be updated regularly in order to include new CAs or remove those compromised or ceased. Nowadays, CA lists are quite similar regardless the software manufacturer but have subtle differences. Since there is no PKI root authority, the composition and distribution of the lists is the responsibility of the manufacturer of a particular software. However, users are allowed to add certificates that were not originally in the list despite they can be hard to remove afterwards [71].

Second, every CA included in the list, thus trusted in advance, can issue certificates for any domain name, being the owner of that domain name unaware of that. This is an antagonist behaviour compared to DNS, that does not allow those changes. In DNS, every DNS zone is sovereign of its sub-domains and only the root, that is unique, is free to behave with impunity, but fortunately managed by trustworthy entities. X.509 certificates, when used for authenticating domain names in TLS, use a certificate extension called *subjectAltName* [25] that allows a certificate to reference alternative names irrespectively of the certificate *subject* field. Among these alternative names, domain names are considered. In this way, PKI certificates define a 1 to N relation between the certificate and the N domain names the certificate authenticates. This relation is unidirectional since the domain owner cannot point out which certificate should be used for a given domain name, except by using DANE that will be discussed in Section IV-F.

CAs are usually managed by competent serious organizations, sometimes subject to auditing. Notwithstanding, the lack of a bidirectional and verifiable relation between the entity that vouches for a certificate protecting a resource, and the owner of that resource, allows any CA to masquerade any domain name without the knowledge of the domain owner. As a simple example, an attacker can obtain a certificate for a popular domain name from a compromised or cooperating CA, so any man-in-the-middle masking that domain, will be fully trusted by the victim, since the certificate can be verified and contains the domain name. The victim will trust the certificate and authenticate the server having no way to verify if the owner of the domain allows that certificate to be used for authenticating its domain.

Compromising a CA can be considered difficult and quite infrequent, but the risk exists and there are several well documented attacks to popular services as Google and Facebook that will be discussed in Section IV. In order to illustrate the dimension of the problem, the following section discusses the current state of PKI certificates and the Certificate Authorities.

A. Current State of PKI Certificates

The Electronic Frontier Foundation (EFF) keeps a database of certificates [72] in the project "SSL observatory", used in SSL/TLS handshakes. That database is no longer updated at the time of writing this article but the EFF offers a dump of around 16GB corresponding to 2011. Other organizations, as Qualys SSL labs, maintain a database of certificates and verification services that allows to find out if it is secure to connect to a server using TLS by observing the server configuration [73]. The data used in this section to illustrate the problems of PKI certificates are based on EEF and Qualys SSL labs data.

Currently there are more than 200 independent certificate authorities in lists provided by operating system and applications. Currently, the biggest trusted root CAs repositories (root stores) are those handled by software companies as Apple, Microsoft and Mozilla [74]. Those root CAs expanded to 1482 trusted CAs (not only root but intermediate CAs) controlled by 651 organizations in 2010.

Several CAs issue subordinate CA certificates allowing the later to issue certificates as if they were issued by the first. For instance, the CA named "C = DE, CN = Deutsche Telekom Root CA 2" had 252 sub CAs in 2011 and "C = US, CN = GTE CyberTrust Global Root" had 93 sub CAs [75]. The disproportionate proliferation of sub CAs has lead to extreme situations. "TrustWave" admitted some clients were issued subordinate CA certificates allowing them to issue certificates in the name of TrustWave. That amount of trusted third parties is becoming a serious management problem leading to attacks that are discussed in Section IV.

When it comes to server certificates, in 2010 there were more than 16.2 million servers listening at port 443 (HTTP over TLS default port) but just 11.3 million (38%) were able to respond to a SSL/TLS handshake and only 4.3 million had a valid certificate. The rest, over the 60%, used either malformed or unverifiable certificates. According to the EFF, some server certificates had a valid signature but were signed with keys from CAs known to be compromised time ago [75] whereas others were issued to subject names as "localhost" or even IP addresses.

The major concern about using malformed or untrusted certificates when users are involved is that browsers allow users to continue the interaction even if the certificate cannot be validated. This behaviour is known as "click(ing) through" security [76]. The literature concludes that the majority of average users do not understand the warnings shown by browsers upon a handshake with a defective certificate and decide to access the service [77] since PKI is complex and hard to understand. So PKI on its own does not provide protection as defective certificate warnings can be circumvented.

Eckersley [78] discussed in 2011 that besides the general use of secure protocols versus their unprotected counterparts is much more secure, there are still several attacks that basically rely on how certificates are issued and verified. These structural problems on their own, thus not considering cryptographic weaknesses and protocol design flaws, allow to perpetrate sophisticated attacks.

These attacks can be carried out in the following situations:

- if an attacker is able to compromise a CA or its Web frontend (also known as Registration Authority or RA) that conveys certificate requests to the CA.
- if a router close to the CA is compromised, since this allows to read and manipulate outgoing CA email (since STARTTLS is subject to downgrade attacks).
- if a recursive DNS used by a CA is compromised with DNS Cache Poisoning Issue ("Kaminsky bug" - CVE-2008-1447) that helps the attacker preventing a CA from verifying a domain name.

TABLE I
TOP-15 COUNTRIES WITH ROOT CAs FROM EFF OVER A SAMPLE OF 1355551 PKI CERTIFICATES

Country	Code	Valid Certs	Valid CAs	% Certs	% CAs
United States	US	1031408	130	76.09	17.86
South Africa	ZA	116437	3	8.59	0.41
United Kingdom	GB	78896	15	5.82	2.06
Belgium	BE	34650	8	2.56	1.10
Japan	JP	20749	30	1.53	4.12
Netherlands	NL	16610	15	1.23	2.06
Germany	DE	15389	294	1.14	40.38
Israel	IL	14166	6	1.05	0.82
Spain	ES	4098	27	0.30	3.71
France	FR	3814	26	0.28	3.57
Bermuda	BM	1956	4	0.14	0.55
Poland	PL	1883	8	0.14	1.10
Republic of Korea	KR	1524	14	0.11	1.92
Switzerland	CH	1415	14	0.10	1.92
Italy	IT	1277	11	0.09	1.51
Total		1355551	728		

- attacking other protocols as TCP or BGP to gain access to the emails sent to the victim's domain.
- governments or corporations with access to a cooperating or owned CA that request the issuance of a malicious certificate for a target domain [79].

In general, every aforementioned mechanism for compromising a CA is worrying, but the participation of governments, corporations, special-interest groups or lobbys is specially alarming. The main reason is that trusted certificate lists are global and observe no jurisdiction whereas CAs belongs to companies or institutions that are present in different countries with different legal regulations. Those institutions could be also misused to attack other countries. Table I shows the list of countries with root CAs in 2011 [72].

Beyond cross-regulation issues among countries and the interests CA operators may have, it is important to note that the consequences of PKI certificates issued by malicious or inadequately managed CAs, can transcend the good practises of domain owners, who have no way to defend against, or even discover, a PKI certificate that has been issued to their domain without their consent. There are documented evidences of this kind of problems with PKI certificates, either issued by compromised CAs or stolen. Despite those issued by compromised CAs are more dangerous (can target any domain), stolen ones can bring severe consequences if domain owners do not realize the problem during a long time.

An evidence of the frequency of these attacks can be extracted from Eckerley and Burns analysis [80]. They analysed Certificate Revocation Lists (CRLs) from the CAs monitored by the SSL observatory at the EFF [72]. CRLs were analysed considering the reason why every certificate was included in the CRL. Table II shows the data of those experiments between June and October 2011. It can be appreciated a significant increment on the number of revocations due to a compromised CA. Moreover, since there is no obligation to indicate the reason, it may happen the "Unspecified" category conceals many other compromised certificates.

CRLs have increased the number of items in the last few years giving a correct idea not only of the explosion of certificate revocations, but also of the significant increase of PKI adoption [78]. Due to that, it is undeniable these problems will increase if PKI is generally adopted by IoT/M2M solutions since the vast majority of IoT/M2M protocols are relying on TLS/DTLS for security.

TABLE II
REVOCATION REASONS COLLECTED BY EFF BETWEEN JUNE AND OCTOBER 2011

Reason June 2011	Occurences	Reason Oct 2011	Occurrences
`null` field	876049	`null` field	921683
Affiliation Changed	27089	Affiliation Changed	41438
CA Compromise	55	CA Compromise	248
Certificate Hold	52786	Certificate Hold	80371
Cessation Of Operation	700770	Cessation Of Operation	690905
Key Compromise	59527	Key Compromise	73345
Privilege Withdrawn	4589	Privilege Withdrawn	4622
Superseded	66415	Superseded	81021
Unspecified	174444	Unspecified	168993

The results previously shown are useful for illustrating the problem of the excess of Trusted Third Parties (TTPs) (CAs in PKI), and despite the data was collected between 2011 and 2013, the problem is still worrying. In fact, the number of CAs and certificates is still growing and there are several documented cases of problems related to misused or malicious certificates that happened after 2011 as discussed in Section IV.

A more recent study, that analysed the HTTPS certificate ecosystem over a bigger sample [81], insists also on the problem of the growing number of TTPs. Table III shows a comparison of the size of the sample used in different studies over the time. The study demonstrated several problems [81] already detected by EFF [72], as the fact that in August 2013 only the 67% of the servers listening to the port 443 were able to finish a TLS handshake. Moreover, it detected that from a sample of 8.1 million certificates, only 3.2 million were trusted. The rest were self-signed certificates (48%), certificates issued by unknown CAs (33%) and certificates issued by known but untrusted CAs (19%) [81].

Regarding CAs, the study found 1832 certificates from CAs belonging to 683 organizations spread among 57 countries but with the 99% of the CAs concentrated in 10 countries. From all those organizations with access to a trusted CA and able to issue certificates without restriction, just the 20% belongs to commercial CAs. The rest of the organizations are religious institutions, museums, libraries, and more than 130 financial institutions. In other words, organizations that are not commercial CAs, control 1350 out of 1382 (74%) of the CA certificates trusted either directly or not by browsers, suggesting a big trust problem.

B. TLS Security Evolution

After the discussion of PKI problems regarding trust and the global PKI ecosystem, this subsection presents and discusses data regarding the evolution of TLS perceived security over the time. The data has been fetched from the database of SSLLabs [73] that, since 2012, publishes [82] a monthly security analysis of the most visited servers, as part of the project SSLPulse [83] that presents a radiography of the TLS security. SSLPulse performs verification of a certificate and its chain but concentrates on tests concerning the supported SSL/TLS versions, their key exchange mechanisms, and the cipher suite

TABLE III
COMPARISON AMONG PKI ECOSYSTEM STUDIES [81]

Scan	EFF	First	Representative	Latest	Total
Date Completed	2010-8	2012-6-10	2013-3-22	2013-8-4	Unique
Hosts with port 443 Open	16,200,000	31,847,635	33,078,971	36,033,088	(unknown)
Hosts with HTTPS	7,704,837	18,978,040	21,427,059	24,442,824	108,801,503
Unique Certificates	4,021,766	7,770,385	8,387,200	9,031,798	42,382,241
Unique Trusted Certificates	1,455,391	2,948,397	3,230,359	3,341,637	6,931,223
Alexa Top 1 Mil. Certs	(unknown)	116,061	141,231	143,149	261,250
Extended Validation Certs	33,916	89,190	103,170	104,167	186,159

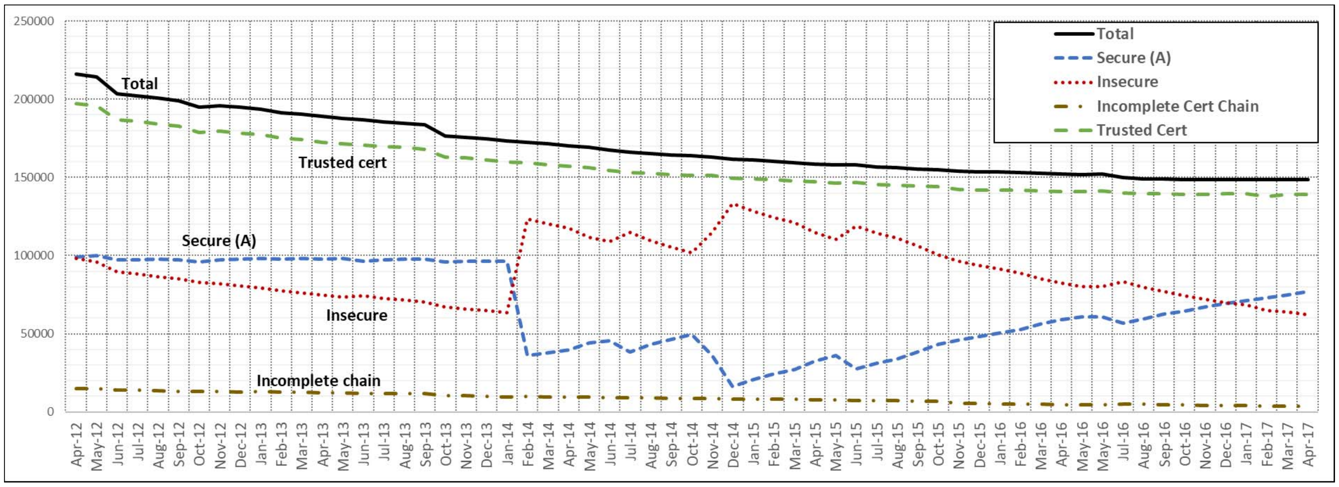

Fig. 5. SSL/TLS security evolution considering the certificate chain (only for servers with valid certificates).

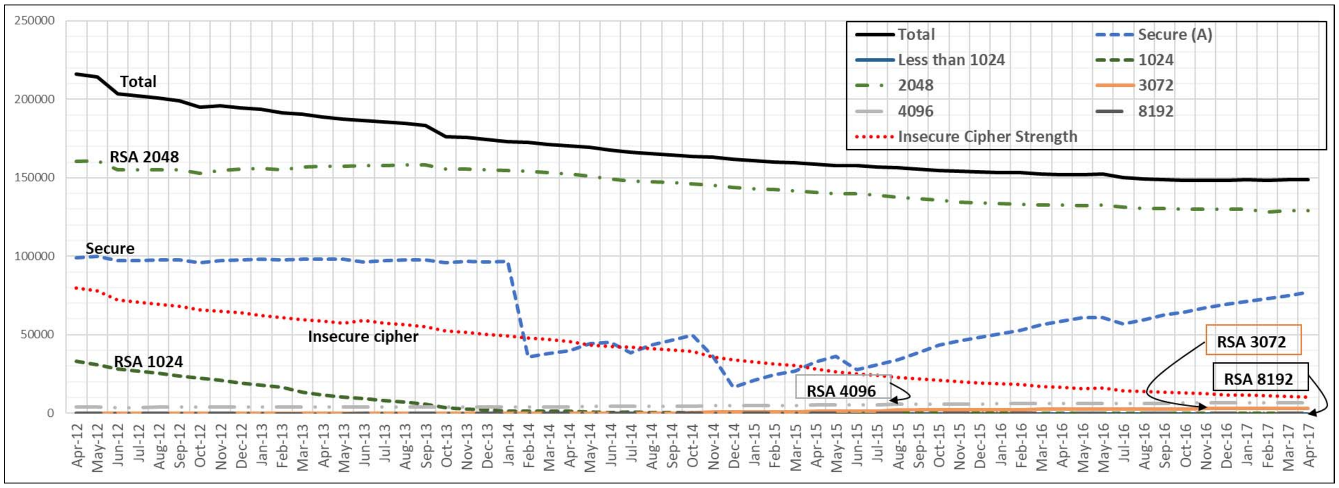

Fig. 6. SSL/TLS security evolution considering key length and cipher suite.

support. The result of the individual tests are combined into a global score ranging from 0 to 100 (it should be noted a 0 in some tests results in a 0 in the global score). According to the score, they classify servers with a mark ranging from A to F being A the best and F the worst. SSLPulse data has been downloaded and processed to compare the number of secure (A) and insecure (rest of marks) servers over the time with respect to the three different families of SSL/TLS problems. These families are certificate chain problems, key size, and protocol version or vulnerability problems as described in Section II-C.

Fig. 5 shows the proportion of secure and insecure servers against valid and complete and valid but incomplete certificate chain occurrences, showing it has no significant impact in the results. It should be noted the study concentrates on the most visited sites with valid certificates, in contrast to other experiments that consider the entire certificate ecosystem [72], [81].

Fig. 6 shows the security evolution considering the length of the certificate public key used for authentication (and sometimes for key exchange), and discovery of insecure symmetric cypher suites. The figure shows that the number

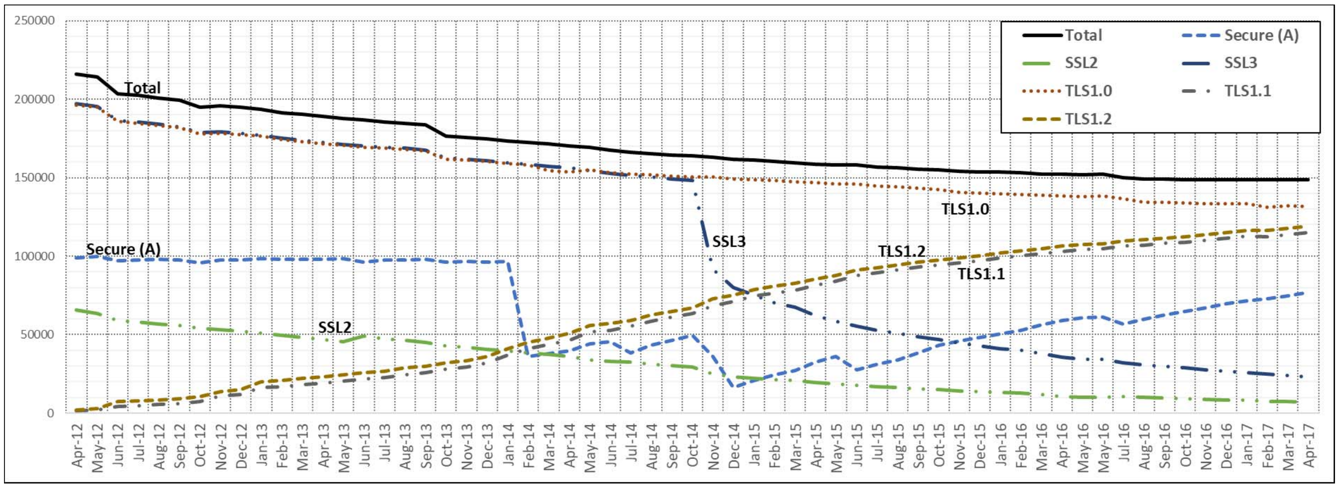

Fig. 7. SSL/TLS security evolution considering protocol versions.

of certificates with public key smaller than 1024 bits has been gradually reduced and vanished at the end of 2013 whereas the most common key length is 2048 bits. The number of insecure cipher suites has been descending, becoming incidental in 2017. Alike the case considered in Fig. 5, there is no relevant correlation between the key length and cipher suites and the evolution of secure/insecure servers.

Fig. 7 shows a significant correlation between the SSL/TLS versions and the number of secure servers. The most significant event can be attributed to the Poodle vulnerability of SSL 3.0 (CVE-2014-3566) at the beginning of 2014. This vulnerability caused a drastic decrease on the number of secure servers until SSL3 was abandoned at the end of 2014. Currently there are still a non negligible number of occurrences.

IV. Certificate Pinning Solutions

As it has been discussed, PKI has no single root authority able to verify certificates, otherwise it has a big set of independent root authorities able to verify only sibling certificates. The list of root CAs is compiled by software or operating systems manufacturers.

Section III-A reasons the disproportionate growth of CAs an its associated problems. In this section, the concept of Certificate Pinning is introduced as a tool to avoid or diminish the major current Web problem that will, undoubtedly, affect IoT and M2M restful services as well: the lack of trust.

To better illustrate the problem, consider TLS authentication. In the process of authenticating a server, the client starts a TLS handshake and obtains the server certificate during this handshake as stated in Section II. There are two distinct operations during a handshake: derive a key to protect the traffic in a secure way, and authenticate either the server or both the client and the server (mutual). Despite there are several authentication algorithms, that can be negotiated during the handshake, the party to be authenticated should provide a proof of possession of the private key associated with the certificate. The process is equivalent for server authentication and mutual, so we will concentrate on server authentication.

The client, once in possession of the server certificate, verifies the signature and builds a PKI certificate chain from the server certificate up to the first intermediate or root CA it trusts by checking the trusted CA list under use. If this verification is successful, the client can also verify the certificate revocation list published by the CA. If the verification is successful and the certificate is not revoked, it checks whether the certificate alternative name matches the server's domain name. If it matches, the connection is considered trusted.

Thus, the creation of the PKI certificate chain is the weakest part of the verification. It is known that public and private CAs, that are in the trusted list and thus trusted by the clients, have introduced intermediate sibling CAs that are therefore trusted by the clients. The purpose of this intermediate CAs can range from subordinate CAs borrowed to companies, can issue certificates without the parent CA intervention, to SSL/TLS accelerators that can access the traffic in clear text. In the first case, the intermediate CAs increase the length of the certificate chain but are transparent to the user. However, as mentioned in Section III-A, cases as the CA "C = DE, CN = Deutsche Telekom Root CA 2" that had 252 sub CAs in 2011 and "C = US, CN = GTE CyberTrust Global Root" had 93 sub CAs [75], raise concerns about the control the parent CA has over their sibling CAs.

In the second case, several network operators introduce intermediate CAs that let companies to accelerate SSL/TLS traffic [84]. By means of those intermediate CAs, providers can issue intermediate server certificates that intercepts and accelerate SSL/TLS traffic on behalf of customers (for instance Web servers with a huge traffic). In this way, servers can offload encrypted streams management to a third party. However, despite a client accessing a server can perceive an improvement in the response time, it is generally unaware of the fact its traffic is not protected end-to-end, but decrypted at an intermediate point in the network and delivered in clear text from that point to the server. Moreover, there are other worrying cases in which these solutions are incorporated to consumer electronics devices that can violate user privacy [85].

Any of these numerous root or intermediate CAs, that can be compromised or misused, can issue certificates for any domain name without the knowledge of the domain owner. There are several well-known cases that occurred without being known until later. In 2011, the Malaysian Agricultural Research and Development Institute CA was compromised and used to build a malicious tool out of the Acrobat update tool. That tool installed updates that seemed to be legitimate but that turned the client into a spy machine under the control of the hackers. Until the problem was detected and the CA certificate was revoked, they could have impersonated any Internet domain.

In 2011, a CA called Diginotar was used to issue certificates for Gmail and Facebook among others [71], [86]. Google advertised the problem through a comment from a customer in Google Groups. Other cases as Comodo in 2011 and TurkTrust in 2013, have been also very popular examples of compromised CAs. Recently, TrustWave admitted to have issued subordinate root CA certificates to clients that were able to issue PKI certificates for any domain in the planet without the control of TrustWave [86]. Those practices increase the risk of finding certificates issued by third parties without the knowledge of the owners of the domains that are forged. These attacks do not break but rather modify the trust chain, and work transparently to the user and the domain owner, so they are quite dangerous.

Despite several protocols have been proposed to manage trust, or better, to manage "trust-anchors" [87], [88] for building the certificate chain, the problems arising from compromising a CA are still present since the malicious certificate is issued by a CA that is directly or indirectly in the trusted CA list and, as it has been reasoned, there are too many.

"Certificate Pinning" is a concept that allows clients to obtain a better certainty that a certificate used by a server is not compromised. In the following sections several "Certificate Pinning" proposals will be evaluated. Certificate Transparency, described in Section IV-A and SK, in Section IV-B, propose complementary infrastructures for controlling the certificates globally together with a client cross verification; Trust Assertion for Certificate Keys (TACK), analysed in Section IV-C, proposes a cross verification controlled by the domain owners; DNS Certification Authority Authorization (CAA), described in Section IV-D, lets the domain owner limit which CAs can issue certificates to its domain; HTTP Strict Transport Security and HTTP Public Key Pinning Protocol describe new HTTP headers that enforce policies for TLS and conveys the certificate chain to be used (Section IV-E); finally, DANE together with DNSSEC are described in Section IV-F2. The solutions will be compared in Section V and their viability for IoT/M2M scenarios will be discussed.

A. Certificate Transparency

Certificate Transparency (CT) [89] was proposed as a countermeasure to the impersonation of sites. CT provides a "Certificate Pinning" or alternative verification for users and a surveillance system for CAs. Basically, it allows a verifiable structure containing traces of existing server certificates to be audited by several actors, every of them with their own interests, to detect malicious or compromised CAs.

The objectives of this proposal are: to harden malicious CAs certificate issuance for a given domain without domain owner knowledge; provide an auditing and monitoring system to allow domain owners to detect unauthorized certificate issuances; and, as a consequence of the previous, protect users from being scammed.

The verifiable structure consists of a Merkle Tree (MT) [90]. Such a tree contains a hash of an object subject to verification in every leaf. The existence of an object in the tree and the order in which it was added to the tree can be verified by means of the MT. To accomplish that, parent nodes in a MT contain a hash that combines the hashes of their children and continues until the root, that contains a hash combining the hash of every descendant. In this way, any change in either the content or the order of the leafs, alters the value of the root.

Verifying a leaf in a MT requires processing a number of nodes proportional to the logarithm of the number of nodes [91] thus, to verify an single object within a tree of a million leafs requires processing 20 nodes [92]. The tree used in CT is based on the proposed method by Crosby and Wallach [93], that uses a SHA-256 hash. Every node is calculated over a data list and the hash is 32 bytes long. Thus, for an ordered sequence of n entries $D[n] = \{d(0), d(1), \ldots, d(n-1)\}$ the Merkle Tree Hash (MTH) (MTH()) is defined in the following way [89] for an empty sequence, a single element and n elements.

An empty sequence MTH() = SHA-256(). For a single entry in the list corresponding to a tree leaf, MTH(d(0)) = SHA-256 (0x00 || d(0)). The reason to concatenate 0x00 and d(0) permits to differentiate the hash operation over the leafs from the rest of nodes, that are concatenated with 0x01. Otherwise, it would be possible or easier to generate collisions or second pre-images of the hash [94].

If $n > 1$, consider k the biggest power of two less than n so $k<n\leq 2k$. The MTH of a list of n elements, $D[n]$, defined in a recursive way is MTH($D[n]$) = SHA-256(0x01 || *MTH*(D[0 : k]) || *MTH*(D[k : n])) where || means concatenation and $D[k_1 : k_2] = d(k_1), d(k_1+1), \ldots, d(k_2-1)$ is a list of $k_2 - k_1$ elements. In this way, a signature over the resulting MTH can be used to verify the entire tree.

In CT, each leaf stores a certificate issued by a CA upon CAs request. There are two interesting verifications in the tree. The first, verifies if a given certificate belongs to the tree, i.e., it was communicated by the corresponding CA and added. The second pursues to verify that this append-only tree has not been tampered with, so the order in which certificates were added to the tree is consistent with their time stamps. In order to prove this, CT defines "Merkle Audit Path" and "Merkle Consistency Proof".

The Merkle Audit Path (MAP) for a given leaf is defined as the shortest list of nodes from the leaf to the root that allows to derive the MTH for that tree. A verifier in possession of the signed MTH, uses MAP to verify if a leaf belongs to a tree. Thus, if the MTH derived from MAP matches the MTH in

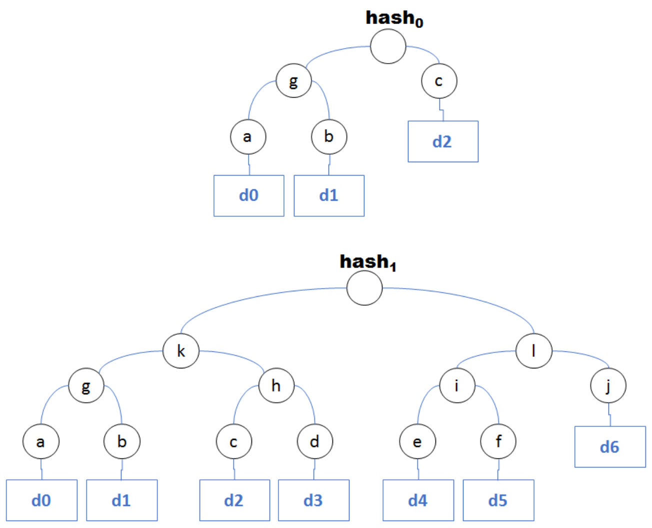

Fig. 8. Example Merkle Tree in two different instants of time to illustrate the examples of Merkle Audit Path (MAP) and Merkle Consistency Proof (MCP) calculation.

possession of the verifier, then MAP proves the leaf belongs to the tree.

The Merkle Consistency Proof (MCP) verifies the tree is append-only so existing leafs are not modified or deleted and kept ordered. Suppose a verifier has the current root hash, $MTH(D[n])$, and a previous root hash corresponding to the first m leafs, $MTH(D[0 : m])$, with $m \leq n$. MCP is the minimum list of tree nodes that allows to verify that the m first leafs are the same in both trees.

Fig. 8 shows the same tree in two different times. The first has a hash labelled as hash_0 and the second hash_1. For the tree with hash_1, the MAP for d_0 will be the list $[b, h, l]$ since d_0 can derive a, but needs b to calculate g, h to calculate k and finally l to calculate hash_1. d_0 belongs to the tree if hash_1 matches the expected MTH. Equivalently, MAP for $d4$ will be $[f, j, k]$.

If both trees are considered, a consistency proof between the tree with hash_0 and hash_1 will be $MCP = [c, d, g, l]$. c and g will be used to verify hash_0, d to verify k and l to verify hash_1. In this way, it can be verified that hash_1 has been generated from hash_0 and thus, it is consistent.

CT defines three components that are *log server*, *monitor* and *auditor*. *Log servers* guard CT MTs. Despite the number of *log servers* needed to handle the current Internet is not specified (nor who is in charge of them) [89], some research indicates that around a thousand servers are needed in all the world, [95] that may be managed by CAs, Internet Service Providers and other parties.

When a valid certificate is sent to a *log server*, it issues a "Signed Certificate Stamp" (SCT). The SCT is a *log server* promise to incorporate the certificate into the tree in a time less than the "Maximum Merge Delay" (MMD). The certificate will be included in a leaf of the tree in a structure that includes the certificate an the SCT. Then a hash is calculated over that structure and added to the tree.

Every time a certificate is added to the tree, the hash of every node affected by the change is recalculated down to the root. The resulting hash at the root is then signed leading to the "Signed Tree Header" (STH). Thus, only STH (one per tree) and SCTs (one per certificate) are signed with the *log server* key pair to allow secure tree verification.

In order to be accepted by a *log server*, candidate certificates should pass a PKI verification. It requires building a certification path from the certificate to a trusted CA, according to the list of trusted CAs accepted by the *log server*. Accordingly, every *log server* shall publish its trusted CA list [89]. The procedure for CAs is different and requires pre-certificates with poisoned extensions as explained later in this section.

In this way, there is no room for self-signed certificates or those issued by local CAs (security islands) as well as other use cases that modify locally the trust-anchor to cope with a high dynamicity, as those described later in Section IV-F2.

Monitors are entities that inspect and verify the operation of a *log server*. *Monitors* have particular interests, looking for certificates for a given domain or set of domains, but may overlook others. Every *monitor* should inspect every new entry in every monitored *log server* and may keep a copy of the entire log. Hence, *monitors* should periodically obtain the *log server* entries and the STH, verify the signature and perform consistency verifications.

Finally, *auditors* take partial information from a *log server* as input and verify that the information is consistent with previously collected evidences. According to the specification, an *auditor* can be a TLS client or an independent entity that provides services to TLS clients. Basically, an *auditor* verifies the consistency of two SCTs of the same tree, at the same *log server*, with a consistency check (MCP).

In general, *auditors* are aimed to be part of the TLS clients [95], so the verification of SCTs is devolved upon them. This is important as TLS clients should reject certificates without a valid SCT. However, it is suggested that *monitors* could not only verify the log integrity and look for an interest, but also provide free or paid services to CAs and domain owners. Moreover, *monitors* can be operated by domain owners, and even act as *auditors* on behalf of the TLS clients. CT operation is presented in Fig. 9.

CT requires *monitors* with particular interests to warn the owners of the monitored domain names so that owners can take appropriate actions when a log is misused. Either failing to insert a certificate in the MMD time or violating the order in the tree (consistency) is considered misuse. *Auditors* can detect failed insertions requesting MAPs for every observed SCT. *Auditors* cooperation to detect consistency violations can be achieved by a "gossip" protocol, as suggested in the specification.

Supporting CT entails servers to deliver SCT together with the certificate, so clients can cross-verify SCTs against certificates. Since *log servers* are monitored by *monitors* and invalidated upon misuse, verifying SCTs may suffice to harden impersonation using certificates from a compromised CA. CT proposes several alternatives to deliver SCTs with certificates [89], [95].

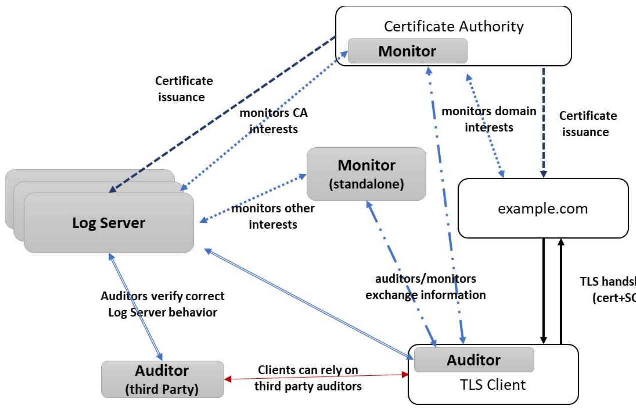

Fig. 9. Interaction among participants in Certificate Transparency. Certificate issuance is a one time operation. TLS handshake is synchronous whereas the rest of the interactions (auditor-monitor, auditor-log, monitor-log) are asynchronous.

- Embedded in the certificate: the SCT can be embedded in the certificate using an X.509v3 extension [96]. The problem is that the SCT is obtained after the certificate is accepted by the *log server*, and embedding the SCT inside the certificate afterwards will invalidate the signature. For that reason, CAs willing to embed the SCT in the certificate, should send a pre-certificate identical to the certificate that will be eventually issued. A poisoned extension will be added to the certificate using a critical X.509 extension and signed (TBSCertificate [24]). In this way, the pre-certificate serves as a log entry, so can be used to verify, but cannot be delivered in a TLS handshake. This mechanism does not require changes in the clients or servers.
- TLS extension: in this case, the time stamp (SCT) is delivered during the TLS handshake separately from the certificate by means of a TLS extension [19] with an specific type. This mechanism requires TLS extensions to be supported by the endpoints thus, requires a change in both entities despite nowadays TLS extensions are well supported.
- OCSP stapling: the time stamp is delivered using the well known TLS extension “Certificate Status Request” [97]. To indicate this extension actually contains an SCT rather than OSCP information, a special Object Identifier is used.

Security considerations for the client: The specification does not indicate how *log servers* publish or distribute the necessary keys to verify SCT signatures. In the best case, this *log server* public key list should be distributed with the software or fetched using another out of band protocol. Thus, it requires an additional trust list beyond the PKI one.

The detection of failure in the addition of a new certificate requires clients to request one MAP per observed SCT. This raises privacy concerns as the involved parties can trace the client by observing requested SCTs. To overcome the problem, clients can use trusted third party auditors, for instance from the ISP. Also SCTs can be verified in batches asynchronously so the time in which the SCT was fetched cannot be learn by other parties. However it does not prevent parties from learning the SCTs unless they are altered with noise, what requires an extra effort from client side.

Any given certificate and its associated SCT can be verified using a Signed Tree Header, from the same *log server*, that was signed Maximum Merge Delay (MMD) after SCT time stamp. In order to verify the SCT, it is enough to request a MAP to the log. However, there is a window of opportunity for an attack that depends on MMD, that is the time a *log server* waits for accumulating insertion requests, so they can be added in batch rather than individually saving costly cryptographic operations. Due to that, CT is a system able to detect problems with certificate in hours [95] but its effectiveness in dynamic environments is inversely proportional to MMD. Moreover, in order to detect misbehaving *log servers*, the system demands a global adoption.

Additionally, there is no clear indication of the expected behaviour of the CAs regarding the *log servers* to be used. On the one hand, it is stated CAs may use some logs at its convenience [95]; on the other hand, it is said CAs will request the addition of a certificate to every available *log server* [92]. In any case, it is agreed there will be no synchronization among the different *log servers* leading a different tree per *log server*. That complicates monitors to find a log able to verify a given certificate.

Regarding the client effort, it should have access to a list of *log server* public keys to let the *auditor* verify the SCT signature, or have a strong trust relation with an external auditor, which in practise is the same problem. Moreover, a certificate verified with Certificate Transparency should also pass a PKI certificate chain and certificate status validation [98].

B. *Sovereign Keys*

Just like Certificate Transparency, “Sovereign Key Cryptography for Internet Domains” proposes a public verifiable and auditable append-only structure [92] that associates every certificate with a Sovereign Key (SK). A SK can be associated with one or more certificates and used to cross-sign the final certificate.

One of the objectives of the SK is to protect clients against Man In The Middle (MITM) attacks or impersonation. To achieve this goal, clients supporting SK should verify that the public key pertaining to a certificate, used by a server, has been cross-signed with the SK registered for the domain of the server.

Unlike CT, SK enables the definition of an alternative route to the server in case of impersonation, MITM attack or connection blocking (every of them detected by SK). In this way, beyond alerting the user, that may not be effective [77], [99] as alerts are frequently ignored if there is no alternative way, SK provides other routes to reach the service.

The verifiable structure used by SK consists on an append-only “TimeLine” whose entries are relationships between domain names and SKs (keys), being SKs different from those used by server certificates.

The purpose of the TimeLine is to store and preserve the history of SK-domain relations. The latest entry for a

TABLE IV
RECORD ENTRY IN A SOVEREIGN KEYS TIMELINE

Purpose	Type	Field
Monotonicity	`uint 64 bit`	serial number
	`uint 64 bit`	time stamp
Sovereign Key	`char[256]`	name
	`bool`	wildcard
	`char[]`	key type
	`char[]`	sovereign public key
	`char[]`	protocols
	`uint 64 bit`	expires on
In case of revocation	`char[][]`	inheriting name(s)
Evidences for claim	`char[]`	cacertchain
	`char[]`	DNSSEC_evidence
	`char[]`	claim_signature
TimeLine Signature	`char[]`	signature

domain, together with the set of updates, changes on the service domains, renovations, revocations, and other adjustments, is valid. A request for a new SK, that will be stored in the structure contains the fields described in Table IV.

A request for addition to the TimeLine should contain data related to the SK and the service (or services) to whom it will be related. The field "Sovereign Key" contains the domain name ("name") that the SK applies to. If the "Wildcard" is unset [100] a different SK can be used per sub domains; otherwise only the SK will be valid for every sub domain.

Regarding the type of key, expressed with "key type", the specification [100] proposes the use of ECC for an optimum storage and compression [101]. The SK key par should be generated by the requester. The public key together with a proof of possession of the private key should be added to the request. The field "Sovereign public key" contains the public key from SK.

The structure allows limiting the services to which the SK is applicable by means of the "protocols" field. The field contains a text chain with the alternative routes to the services delimited with semicolon. For instance, HTTP at port 8080 or an onion routing address [100]. Finally, that table entry also contains the expiration date in the field "expires on".

The field "inheriting names" allows SKs to be re-issued for the domain if current SKs are compromised or revoked. The value is a list of domains allowed to request the addition of a new entry under these circumstances. In fact, it is a delegation to re-register the domain if, and only if, the SK is revoked. In this way, SK avoids the domain owners listed in the field to alter others SKs if their domains are compromised but not the original SK.

Requesting the inclusion of an SK in a domain, requires to provide evidence that the domain is under the control of the requester. This evidence can be a certificate signed by a trusted CA containing the domain name in a `subjectAltNames` extension, or a DANE DNSSEC response (see Section IV-F2). Evidence obtained during the addition of a SK that clients can verify later.

The server holding the TimeLine has to perform an OCSP verification [102] before adding the SK. The OCSP verification is not added to the record due to space constraints.

In order to guarantee the request is consistent and to avoid manipulations from intermediate entities during the request process or from the TimeLine server after the request, the request is signed with the private key of the SK and the result added to the field "claim_signature". This signature serves as a proof of possession as well.

Despite it is not directly stated in the specification [100], it is understood the fields contained in "Sovereign key", "In case of revocation", and "evidences for claim" are protected by the signature in "claim_signature". Any further change performed by the requester later on is added as a new record to the history preserved by the TimeLine.

The signature of the TimeLine server over all the fields of the Table IV is performed with a private key belonging to the TimeLine server and can be performed offline. Thus, to guarantee the append-only feature of the TimeLine an increasing serial number and a time stamp is added to every entry.

The TimeLine structure can also incorporate other entries:

- References to other TimeLines. It is possible entities managing a TimeLine get the TimeLine key compromised thus, requests for SK addition can be sent to several TimeLines. The record field "Incorporate-by-reference" allows linking a registry in a TimeLine with other registries in different TimeLines corresponding to the same operation.
- Revocations provides a mechanism for the owner of the SK to revoke the key. An effective revocation requires the revocation date, the name of the SK (domain name), and a signature over the parameters. The revocation information will be added to the TimeLine with a unique serial number and a time stamp.
- Re-issuing of a revoked SK allows a revoked SK to be re-issued upon an evidence provided from a domain listed in the field "In case of revocation". The structure will be added to the TimeLine with a serial number and a time stamp.
- Protocol changes tracks changes in the field "Protocols".
- Root CA list changes. Every entity managing a TimeLine should keep and maintain a list of trusted CAs. The list can be modified and every change in the list is published in the TimeLine with a serial number and a time stamp. Alike Certificate Transparency, described in Section IV-A, self-signed certificates, domain CAs and security islands cannot be used with SK despite they can be trusted by means of DANE-EE and DANE-TA described in Section IV-F2.

SKs define three entities that are *TimeLine Servers*, *Mirrors* and *clients*. *TimeLine Servers* manage and custody TimeLines. Unlike CT, SK specifies the set of *TimeLine Servers* for the current and foreseeable Internet should have *N* entries with *N* between 10 and 30. It is also stated *TimeLine Servers* should be chosen to guarantee diversity in jurisdiction, operational philosophy or security policies, hence the service will be available even if several *TimeLine Servers* are compromised or disabled.

Every *TimeLine Server* should have a key pair for the signature of the TimeLime as it has been previously explained. According to the SK specification the list of *TimeLine Servers* and their corresponding public keys should be distributed together with the software as happens with PKI trusted root CA lists.

"Mirrors" increase the overall system performance and availability by keeping updated copies of the *TimeLine Servers*. *"Mirrors"* should be identified by an IP address, port and public private key, introducing an additional list of keys.

SK do not define consistency checks based on the structure as CT does, but at protocol level. *TimeLine Servers* should answer *mirrors* and *clients* with all the entries since a given serial number *S*. Every record returned by a *TimeLine Server* should be accompanied by a freshness message called "Timeline Freshness Message" (TFM) with the fields: "TimeStamp" of the request; "Highest Serial Number to date" with the highest serial number of SK records available in the *TimeLine Server* at the time of the request; "Highest CA update Serial Number to Date" the highest serial number of CA update record, and the signature of the TimeLine.

The specification describes the assessment of the freshness considering responses younger than 24 hours as fresh, responses older than 24 hours and younger than 48 hours as acceptable, those older than 48 hours but younger than 2 weeks as unacceptable and older than 2 weeks as fatal.

In order to check the operation of a *TimeLine Server*, the verifier needs to check the TFM of the responses, determining a *TimeLine Server* fails in its duty if, observing the TimeLine:

- two different entries in the TimeLine with the same serial number
- two different entries with discrepancies in the timestamp
- a TFM with a preceding timestamp and a serial number higher than a previously observed TFM
- and entry for an SK with an invalid signature

If the verifier, either a *Mirror* or a *Client* detects a failure in a *TimeLine Server*, it should keep a copy of the entries, distrust the failing *TimeLine Server* and add the *TimeLine Server* to a bad *TimeLine Server* record that can be learnt by other *Mirrors* and *Clients*.

In order to synchronize the list of bad *TimeLine Servers* among parties, the SK protocol uses a field called "renegation_traking" of 32 bits, that contains the less significant bytes of the hash of the bad server list. If a *Client* receives a message whose "renegation_traking" field is inconsistent with the one it keeps, it starts a synchronization process that requires exchanging their lists.

Security considerations for the client: the SK specification [100] does not clearly state how the target domain of a certificate added to the TimeLine is verified, whereas other proposals do, as DANE.

If the signature of the record is performed offline, there is no indication of the time it takes unlike CT that promises to incorporate the record before the MMD.

As it has been discussed before, every *TimeLine Server* should have a key pair for the TimeLine signature. That key pair should be distributed to *Mirrors* and *Clients* with the software. Moreover, *Mirrors* are identified by IP, port and public key, hence an additional set of keys is introduced in the system.

The specification does not indicate whether the TimeStamp of the TFM corresponds to the time of the request or the time of the latest record. In the first case, the server should sign immediately upon *Mirror* request, leading to an amplification attack that can be dangerous if the number of requests grow. In the second case, the server should not need to sign immediately but, unless the channel between the server and the client is protected, an entity in the middle could use previous *TimeLine Server* responses to masquerade recent updates.

In regard to privacy, *Mirrors* can learn the IP address of the *Clients* verifying certificates. The specification proposes two alternatives. The first is that *Clients* should use *Mirrors* managed by their ISPs. In this way, despite the ISP can learn the domains a given *Client* visits, they can already do that by means of the DNS servers. The second consist on a forwarding mechanism initiated by the client. It is proposed every *Mirror* should have two ports, the main one (443) to send the request to the *Mirror* and a second, called "SK mirror port" or MP, that is used to receive and forward responses. In such a way, *Clients* can use a *Mirror* as a proxy improving privacy. However, the second proposal does not protect users from cooperating *Mirrors*.

Finally, the specification states a *Client* requests information to *Mirrors* every 24 hours. Despite this reduces the load of the *Mirrors*, it opens an attack window of 24 hours.

C. Trust Assertion for Certificate Keys

Trust Assertion for Certificate Keys (TACK) [103] allows users to bind a domain with a certificate using a structure called TACK, signed with a "TACK signing key" or TSK, that is chosen by the domain owner. TSKs are trusted by clients and should not be changed frequently whereas changes to server certificates are not limited in frequency. TACK constitutes, in practise, a change in PKI trust model as it moves the trust from the root CA list to the TSKs. Moreover, it proposes a revocation mechanism for compromised TLS certificates whereas an overlapping mechanism for updating TSKs in which the old and new TSK coexist during a time until the old is finally disabled.

Unlike aforementioned proposals, TACK does not introduce a global verifiable structure. TSKs and TACKs are generated and delivered by servers so the client can process and keep them to establish a long-term trust relation with a server.

The system defines two different life-cycles, one for TACKs and other for TSKs. The TACK-TSK life-cycle is the following:

- TSK generation: the server generates an ECDSA key [104] that would be used to sign one or more domain name TACKs.
- TACK creation: the TACK contains meta data to associate a server certificate with a TSK. Once generated, the TACK is signed with the TSK.
- TACK deployment: a TACK binding the TLS certificate and the TSK is given to every server under the domain to be protected. Those servers advertise the TACK setting the "activation flag".
- TACK re-generation: when the TACK expires or the server changes the TLS certificate, a new TACK is generated.
- TACKs revocation: if a TLS certificate is compromised, a new TACK can be created incrementing the field "min generation".

- TACKs deactivation: the server owner can deactivate a TACK unsetting the "activation flag", so servers can remove it after a period of 30 days.
- TACKs overlapping: when a TSK binding a TLS certificate with a given server by means of a TACK, needs to be changed, the server publishes a new TACK signed with the new TSK. The new TACK is distributed together with the old TACK during a period of time, so clients can activate the new association whereas the service is not disrupted.

The fields in a TACK delivered by the server to the client during a TLS handshake are the following:

- `public_key`: contains the integers corresponding to a point in the elliptic curve p-256 [105] that represents the public key of the TSK that signed the TACK.
- `min_generation`: contains the value corresponding to the field `min_generation` of the TSK associated with the TACK.
- `generation`: signals the generation of the TACK so every other TACK whose `generation` is less than the maximum `min_generation` of the signing TSK is considered revoked.
- `expiration`: date after the TACK is considered expired.
- `target_hash`: a SHA256 hash [106] of the public key [24] from the TLS certificate used by the TLS server.
- `signature`: an ECDSA signature using the TSK over all the previous fields.

The associations or "pins" between TSKs and domain names, represented by TACKs, are organized in repositories or "stores". Every store keeps a map that relates Fully Qualified Domain Names (FQDNs) with one or more attribute sets. Among those attributes, it can be found the issuance date, the expiration date, the TSK public key (or its hash), and the field `min_generation`, that should be equal for every TACK signed by the same TSK. A client may have one or more stores that can be local (optional) or provided by a remote party, and can share pins with other clients. The protocol allows clients to download TACKs from others, and publish discovered TACKs using a trusted third party.

TACKs are delivered to the client using a TLS extension, so clients can verify they are connecting to the appropriate server corroborating the received TACK with the information contained in the stores. When a compatible server receives a TLS handshake message, it negotiates the use of the TACK TLS extension and delivers a TACK to the client. If the client has learnt the same TACK several times, it can create a "pin" between the domain name and the TSK within one of its stores. The validity of the "pin" is equal to the period of time the relation has been observed, limiting the impact of erroneous or malicious "pins". The exchange and verification processes come next.

The client verifies the TLS handshake with TACK extension as valid if the handshake results in an encrypted channel, the TACK extension is present and the TACK delivered is valid. A TACK is considered valid if the "generation" field is greater than the "min_generation" field (from the TSK), "expiration" is dated in the future, "target_hash" is correct and the signature verifies. Once the TACK is verified, the client looks for a "pin" in its stores for the server domain name. If the stored TACK is equal to the one received, the connection is flagged as "Confirmed" otherwise as "Contradicted". If there is no association registered for the server domain name the connection is flagged as "unpinned". In the case of "unpinned", if the server TACK has been observed before, the client can activate a "pin" and create an association with an end date: end =current + MIN(30 days, current − initial), being "initial" the date of the initial observation.

Security considerations for the client: As it has been mentioned, a client can use one or more stores, one local and others being remote. The remote ones can be provided by third parties but neither the organizations managing the repositories nor the requirements for a third party to become trusted, are specified. Moreover, if repositories containing TACKs related to a collection of servers from a given geographical area, belong to an operator or provide TACKs by topic, client queries can reveal interest or habits raising privacy concerns.

The specification also states clients can share "pins" with other clients, and even publish those they have discovered using a trusted third party or sharing service. This raises privacy concerns as other entities can learn user habits and interests. Moreover, only valid TACKs can be shared as they can be easily verified, thus clients cannot add perturbations to the observed TACKs to avoid profiling. Finally, this requires the client to support a sharing protocol that may require additional storage and processing power.

D. DNS Certification Authority Authorization

DNS Certification Authority Authorization (CAA) lets domain owners specify which CAs can issue certificates for their domain names. Unlike CT or SK, the proposal does not define a verifiable structure to store evidences of certificate issuance, nor a mechanism for cross verification. It just provides a mechanism for CAs to check if they are allowed to issue a certificate for a given domain upon reception of a Certificate Signed Request (CSR). Thus CAA intervenes only before issuing the certificate.

CAA defines a DNS record called "Certification Authority Authorization (CAA) DNS Resource Record" that allows the domain owner, or the entity managing the primary DNS for that domain, to specify a list of authorized CAs for the purpose of issuing a certificate. In this way, any compliant CA should query the domain owners' DNS server for a CAA record in order to verify if it is authorized for that domain. In this way, the inadvertent erroneous issuing risk is reduced.

In a similar way as TLSA DANE records [107], discussed in Section IV-F, CAA is under the control of the domain owner and no third party is involved. However, the fundamental difference is that CAA helps the CA to determine if it is authorized before issuing the certificate, whereas TLSA allows clients to verify if a server certificate, used in TLS, is authorized by the domain owner for the purpose of authenticating an encrypted connection.

According to the specification, compliance with the CAA DNS record is necessary, but not sufficient condition, for

TABLE V
CAA RR EXAMPLE

Domain name	TTL	Class	Type	Value
example.com.	3600	IN	CAA	0 issue "ca.example.net"
example.com.	3600	IN	CAA	0 iodef "mailto:sec@example.com"
example.com.	3600	IN	CAA	0 iodef "http://rep.example.com/"

certificate issuance since certificate requests should also comply with the CA "Certificate Policy". As part of this criteria, it is required CAs publish their "Certificate Practices Statement" (CPS) and count with an external auditing process. CAA does not pursue creating security islands with local CAs, but enforces any certificate should be issued by a CA in the CAA record. The CAA in the record should be a trusted CA for the client hence, the CA should be in the client's trusted CA list. In this way, if a CA listed in a CAA record for a domain is not trusted by a given client but issued a TLS server certificate, that client cannot trust the TLS server certificate irrespectively of the CAA record.

CAA records cannot help clients in cross verification since a TLS server certificate can have a long life (around years) while the domain owner can change its providing CA and thus, the CAA record, several times during the life of TLS certificates.

A CAA RR consists on a set of flags and label-value pairs known as properties. Several different properties can be associated with a domain by publishing different CAA RR under the same domain DNS. A property can be flagged as "issuer critical" indicating it should be correctly interpreted by the issuer (CA) before issuing or otherwise desist from issuing. The most representative CAA properties are:

- `issue <Issuer Domain Name>`: authorize the owner of the domain "Issuer Domain Name" to issue certificates for domain names managed by the consulted DNS.
- `issuewild <Issuer Domain Name>`: same as previous but allows wildcard domain names.
- `iodef <URL>`: defines the url where inconsistent certificate request attempts should be reported to. Uses the IODEF format [108]

Table V shows an example in which the domain name `example.com` asserts the only authorized CA for that domain is `ca.example.net`, inconsistent requests should be reported through email and a URL.

Despite the specification recommends CAA records to be authenticated with DNSSEC, it is not mandatory. Thus, it would be possible an attacker drops, alters or inserts fraudulent CAA records if DNSSEC is not used.

CAA does not prevent impersonation or fraudulent use of certificates so it contributes to "certificate pinning" but is not a mechanism on its own. Security considerations for the client cannot be discussed as no client is involved in CAA.

E. HTTP Strict Transport Security and HTTP Public Key Pinning Protocol

HTTP Strict Transport Security (HSTS) [109] describes a mechanism that allows Web sites using HTTP to declare they are accessible only by means of an encrypted connection, as HTTPS. Thus, the specification is limited to HTTP that can be used over TLS [110] with the URI schema "https".

HSTS is based on previous research [111], [112] facing threats from passive and active attackers. In the first group of attacks, an attacker listens to the network for session cookies that, despite delivered in first place through a protected channel (HTTPS), are delivered in clear text (HTTP) when the client loads other resources.

In the second group of attacks, an attacker can use poisoned DNS servers or modify unprotected frames to obtain that session information. Then, the traffic can be redirected to unprotected Web servers or to servers using self signed certificates, since, as discussed in Section III, clients will "click through" upon a warning. Other threats as phishing or malware are not addressed by HSTS.

HSTS defines an HTTP header that should be delivered using HTTP over TLS to the client (User Agent or UA) with the format that follows [113].

```
Strict-Transport-Security
   = ''Strict-Transport-Security'' '':''
             [ directive ] *( '';'' [ directive ] )
   directive = directive-name [''='' directive-value ]
   directive-name = token
   directive-value = token | quoted-string
```

Among the directives included in the header, the UA should remember the server policy ("max-age"), if sub-domains should be treated in the same way ("includeSubDomains") or if it should be added to the list of permanent HTTPs servers ("preload"). Clients should store and keep HSTS policies in the UA. Some browsers include some preloaded servers avoiding users to access them without TLS. Nevertheless, HSTS does not prevent problems derived from malicious certificates issued by compromised CAs, but prevents sessions to be redirected to insecure protocols and hence, prevents session information from being stolen.

HTTP Public Key Pinning Protocol (HPKP) [114] lets client detect when a trust or certificate chain has changed unexpectedly. HPKP defines an HTTP header that lets the UA to learn which `SubjectPublickKey` structures should be present in the certificate chain in future TLS connections with the same server. The objective is to avoid MITM attacks based on compromised certificates and it should be used in conjunction with HSTS.

Thus, HPKP defines a relation among a domain name and a certificate chain. The proposed mechanism follows a "trust-on-first-use" (TOFU). The first time the client accesses the server has no knowledge about the server. So, it would not be able to detect a MITM attack. However, in this first connection, the client (UA) learns the valid certificate chain for that server avoiding future attacks.

The HPKP HTTP header should be delivered over TLS to a client (UA) with the following format (in which fields "token" and "quoted-string" are formatted according to [113]).

```
Public-Key-Directives
    = directive *( OWS '';'' OWS directive )
    directive = directive-name
    directive-name = token
    directive-value = token/ quoted-string
```

The "Pin Directive" allows the Web server to state the certificates that should be associated with the host. To do so, it provides a sequence of SHA256 hashes of the `SubjectPublickKey` structures of the expected certificate chain. The directive "max-age" specifies the amount of time (in seconds) a client should remember the chain, considering the server as a "Known Pinned Host". Alike HSTS, it also includes a directive called "includeSubDomains" and a "report-uri" to report unsatisfactory verification.

Security considerations for the client: HPKP defines a mechanism known as TOFU since the first time the client connects to the server it lacks the necessary information to validate the association (pin) so it would not be able to detect a MITM attack. Moreover, since average users disregard browser warnings, it is also feasible a MITM attack even for "Known Pinned Host".

HSTS and HPKP are designed for HTTP only. Moreover, clients should store HPKP associations and remember HSTS policies.

F. DNS-Based Authentication of Named Entities

As discussed in Section III, PKI has not a single root authority able to verify every certificate. DNS has a single root domain (.) under which every domain tree grows forming a tree (.com., .net. and others). Thus, unlike PKI in which every trusted CA can act with impunity issuing certificates, in DNS only the root can act with impunity altering the domain name database.

Branches or sub-domains in DNS are delegated to their respective domain owners. Due to that, DNS has an infrastructure ready to provide an adequate certificate pinning mechanism re-using the DNS infrastructure with several clear benefits: it does not require the distribution of new credentials since DNS is already deployed and can provide authenticated records using security extensions (DNSSEC); it does not require the creation of new services and lets domain owners to manage their trust relations on their own, without the intervention of third parties.

Section IV-F1 briefly describes DNS security extensions for a better understanding of DNS-based Authentication of Named Entities [107] (DANE) that will be described in Section IV-F2.

1) Introduction to DNSSEC: Domain Name System Security Extensions (DNSSEC-bis) [115], [116] belong to a set of specifications from the IETF that allows DNS clients (resolvers) to authenticate the source of DNS responses, authenticate the non existent domain responses (avoiding certain attacks), and to verify the integrity of DNS responses. It does not provide confidentiality unless used together with TLS or DTLS [117], [118].

DNSSEC [119] was first proposed in 1997 but the first version had major scalability problems since parent zones should sign records upon changes in every delegated branch or child zone. The current proposal, known as DNSSEC-bis, proposes an indirection in the signature using records called "Delegation Signer (DS) Resource Records" that improve scalability.

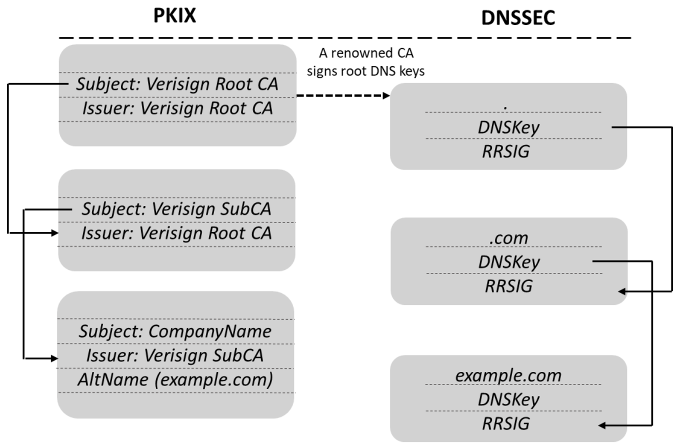

Fig. 10. Trust chain comparison between DNSSEC and PKIX [70]

TABLE VI
DNSSEC RESPONSE EXAMPLE - A RR AND ITS RESPECTIVE RRSIG

Domain name	TTL	Class	Type	Value
abc.mydomain.com.	3600	IN	A	163.117.141.197
abc.mydomain.com.	3600	IN	RRSIG	**A** 5 3 3600 20040509183619 (20040409183619 38519 mydomain.com. OMK 8rAZlepfzLWW75D xd63jy2wswESzxDK G2f9AMN1CytCd10 cYISAxfAdvXSZ7xu jKAtPbctvOQ2ofO7 AZJ+d01EeeQTVBP q4/6KCWhqe2XTjnk VLNvvhnc0u28aoSs G0+4InvkkOHknKx w4kX18MMR34i8lC 36SR5xBni8vHI=)

Every DNS record in DNSSEC is delivered together with and additional record known as RRSIG, that contains a signature over the original record using a Zone Signing Key (ZSK). RRSIG records allows authenticating the information from the DNS. The ZSK is certified by the Key Signing Key (KSK), a longer term key. KSKs and ZSKs are local to their respective DNS zones so, in order to create a trust chain to the parent, a parent DNSSEC uses DS records to indicate the KSKs for the child zones. This signature delegation process, coherent with DNS domain name delegation, is followed down to the root.

The DNS root authority itself, has its own practice statement and a complex ceremony to roll a new RootZone KSK.[2] This ceremony not being altered is the anchor of trust in DNSSEC. The sequence of DS records from the root to any leaf in the DNS tree leads to a trust chain alternative to PKI trust chain. Fig. 10 shows a comparison between both certificate chains (the reader should note the DNS root is unique whereas PKI is not and depends on a given certificate).

Table VI shows an example of DNSSEC record that is discussed next.

[2]See details at http://data.iana.org/ksk-ceremony/iana.org/ksk-ceremony.

The value of a RRSIG is composed by several fields. The first indicates the type of record being signed (`A` in the example). The next field indicates the algorithm used to create the signature (`5` RSA/SHA1 in the example). Then, the RR contains the registers or **labels**, that are used to validate the records generated from a wild card. In the example, it has the value of 3 thus, the original is composed by 3 labels. `abc.mydomain.com` is the original since `abc.mydomain.com` is 3 labels long (`abc`,`mydomain`, **com**) hence it has not been generated using a wild card.

If 2 was use instead of 3, the field would show the domain name was generated from `*.mydomain.com` and that should be used for the verification according to the specification [116]. Then, the original RR TTL is included (`3600` in the example) avoiding old compromised signing keys to be used after the TTL. After the TTL, the record includes the expiration date (`20040509183619`) and the start date (`20040409183619`), the *key tag* used to identify the signing key (`38519`) as well as the signing key name (`mydomain`). Several RRs can use the same RRSIGs, the set of RRs sharing the RRSIG are known as RRSet. Finally, the signature of the RRSet, is generated as follows [115]:

```
signature = sign(RRSIG_RDATA | RR(1) | RR(2)... )
```

where | means concatenation; `RRDATA` are the fields of the `RRSIG` record including the canonical zone name and excluding the signature field; as mentioned, every RR pertaining to the RRSIG contributes with `RR(i)`, that is generated in the following way:

```
RR(i) = owner | type | class | TTL |
                 RDATA length | RDATA
```

Every `RR` contributing to the signature should belong to the same signing zone, keep the original TTL, and should have the same class/type.

The combination of owner, class, and type can be used to determine that the RRSIG in the example authenticates A RRs from `abc.mydomain.com` and no wildcard has been used. Thus, the signature can be verified using a DNSKEY of the zone `mydomain.com` with the *key tag* `38519` using the algorithm `5`.

2) DNS-Based Authentication of Named Entities (DANE): DNS-based Authentication of Named Entities [107] (DANE) relies on DNSSEC to authenticate DANE DNS records with the purpose of associating domain names with credentials (PKI certificates). As it was discussed in Section III, the fraudulent use of certificates is among the most worrying problems of PKI when used with TLS. The problem is that any compromised CA can issue certificates for well-known sites that will pass PKI validation and serve to the purpose of the attack, without the knowledge of the domain owner. As the aforementioned proposals, DANE also pursues creating a kind of association among domain names and certificates ("Certificate Pinning") resistant to those attacks.

DANE allows domain owners to include information about authentication credentials of their permanent services in their DNS. Considering DNS is typically queried for name resolution by the client immediately before connecting to the server, the client can receive information pertaining to the credential that should be received from the server during a TLS handshake before actually connecting to the server. Thus, avoids malicious certificates to be used to impersonate servers managed by the domain owner (see Figure 11).

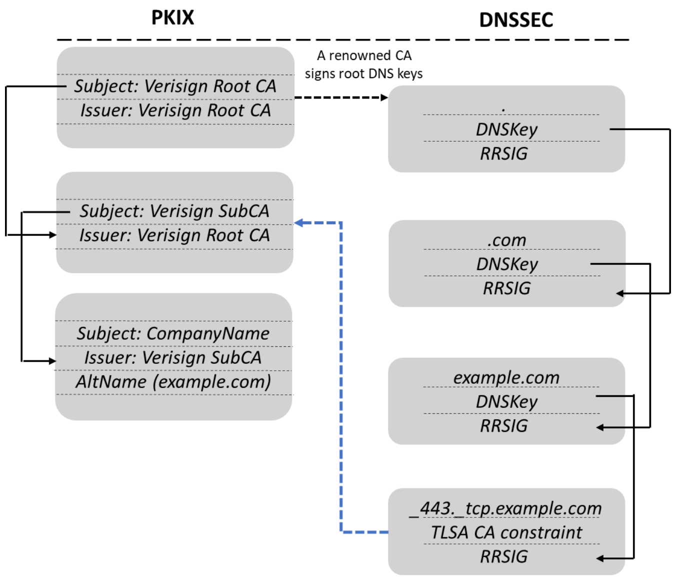

Fig. 11. DANE use case: specifying the CA that should be used by a service (PKIX-TA). The figure shows the DANE record can state the CA that should have issued the server certificate is "Verisign SubCA" so the TLSA record constraints the CA. The solid lines represent the hierarchy in both DNSEC/DANE and PKI. The dashed lines indicate the relation among entities.

DANE defines a new DNS record called TLSA that allows a DNS zone to assert how clients, resolving domain names of that zone, should process certificates received through a TLS connection. DANE specifications [107], [120] define the following use cases depending on the parameters of the register:

- CA constraint: the record specifies the certificate, or the public key of the certificate of the CA, that should have issued the TLS certificate. It does not affect the way the client handles trust since the TLS certificate should pass PKI validation. It just specifies exactly which CA should have issued the server certificate, and for that reason is called "CA constraint" (PKIX-TA [121]). See Fig. 12.
- Server certificate limitation: the record contains the certificate (or the public key) of the TLS certificate used by the server. It is called "service certificate constraint" (PKIX-EE [121]) since it defines the certificate the server should use. Despite the record specifies a concrete certificate, the certificate should pass PKI validation. See Fig. 13.
- CA specification: the record contains the certificate (or the public key) of the CA that should have issued the certificate used by the server. Unlike the first use case, it is not necessary the certificate passes a PKI validation. In this case, DANE modifies the way the client manages trusts and due to that is called "trust anchor assertion" (DANE-TA [121]). It alters client trust since specifies a concrete CA that should be trusted even if

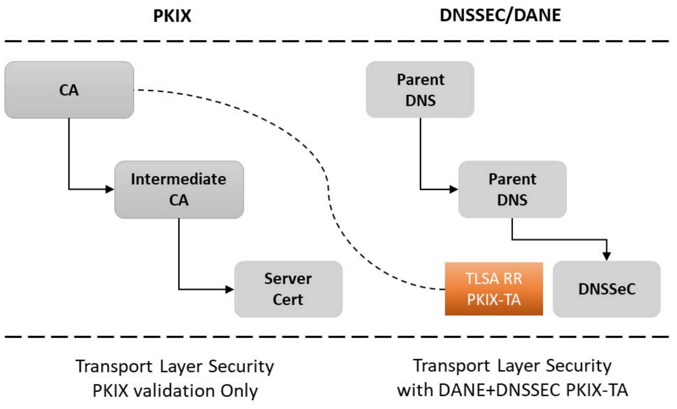
PKIX
DNSSEC/DANE
CA
Intermediate CA
Server Cert
Parent DNS
Parent DNS
TLSA RR PKIX-TA
DNSSeC
Transport Layer Security PKIX validation Only
Transport Layer Security with DANE+DNSSEC PKIX-TA

Fig. 12. DANE use case for specifying the CA (PKIX-TA). Server Cert should pass PKIX validation. PKIX-TA avoids compromised CA attacks.

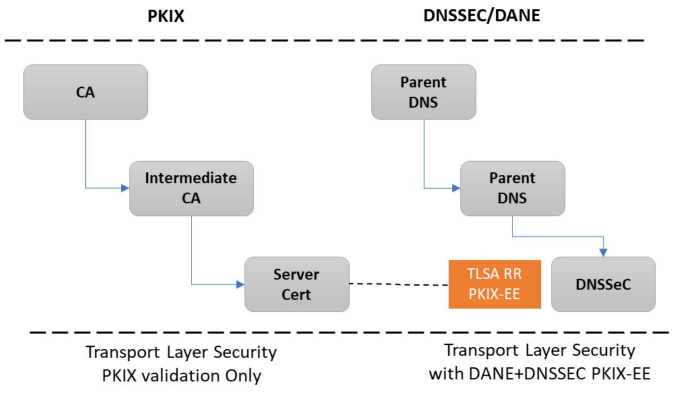
PKIX
DNSSEC/DANE
CA
Intermediate CA
Server Cert
Parent DNS
Parent DNS
TLSA RR PKIX-EE
DNSSeC
Transport Layer Security PKIX validation Only
Transport Layer Security with DANE+DNSSEC PKIX-EE

Fig. 13. DANE use case for specifying server certificate (PKIX-EE). Server Cert should pass PKIX validation. PKIX-EE avoids compromised CA attacks.

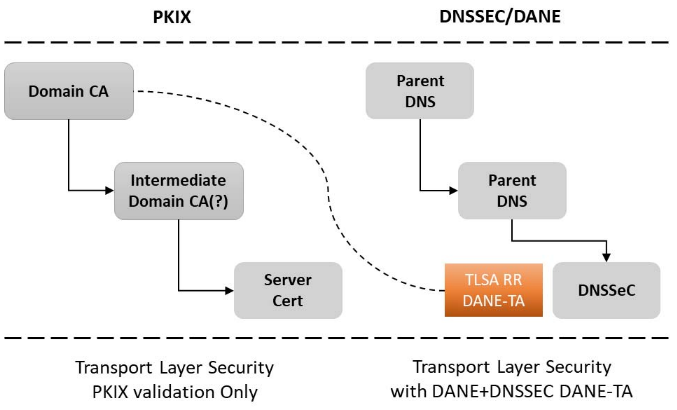
PKIX
DNSSEC/DANE
Domain CA
Intermediate Domain CA(?)
Server Cert
Parent DNS
Parent DNS
TLSA RR DANE-TA
DNSSeC
Transport Layer Security PKIX validation Only
Transport Layer Security with DANE+DNSSEC DANE-TA

Fig. 14. DANE use case for specifying a domain CA (DANE-TA). Server Cert does not need to pass PKIX validation and will not pass it unless the domain CA is added manually to the trusted CA list. Allows Domain CAs (and security islands).

the CA is not trusted by the client. This use case gives support to "security islands" within domain so domains can issue their own certificates using their own CA. See Fig. 14.

- Server certificate specification: the record contains the certificate (or the public key) of the TLS certificate used by the server. This use case, called "domain-issued certificate" (DANE-EE [121]), allows a domain to issue its own certificates without a third party or CA. Alike "trust anchor assertion", it does not require PKI validation altering client's trust. In fact, this use case enables self-signed and ephemeral certificates to be issued by a domain and trusted by the client. See Fig. 15.

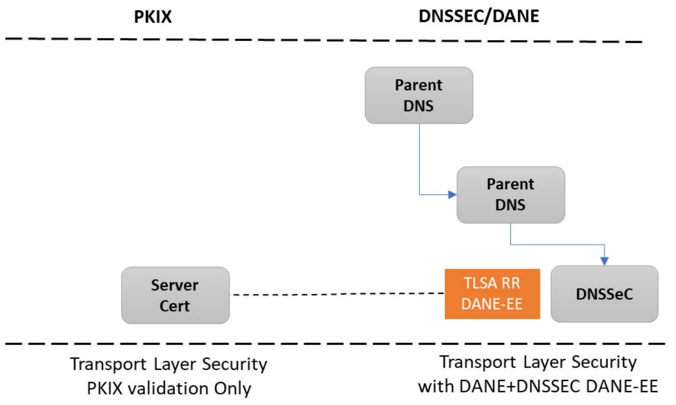
PKIX
DNSSEC/DANE
Parent DNS
Parent DNS
Server Cert
TLSA RR DANE-EE
DNSSeC
Transport Layer Security PKIX validation Only
Transport Layer Security with DANE+DNSSEC DANE-EE

Fig. 15. DANE use case for specifying a server certificate (DANE-EE). Server Cert does not need to pass PKIX validation and will not pass it anyway. Allows Server Certs of any kind (including self-signed certificates) that can be trusted by means of DANE.

The TLSA record defines a "selector" that signals whether the content of the records refers to a "Full Certificate" [24] (Cert) or to its public key, "SubjectPublicKeyInfo" [24] (SPKI). Moreover, it is possible to indicate if the registry refers to the exact content or its hash by means of the field "matching". Finally, the "certificate association data" field, defined by the previous two fields, defines the object target of the association as raw (Full), hashed with SHA-256 or SHA-512 for both certificate or public key.

Table VII shows a TLSA example record returned by a DNS asserting the server `abc.xyz.com` with a REST interface [15] running on port 443, should use a certificate issued by a concrete CA.

The example in Table VII contains a TLSA DANE record and its related RRSIG DNSSEC record. The TLSA indicates server `abc.xyz.com` should use a certificate (`3` - "domain-issued certificate") whose full binary structure (`0` - "Full Certificate") should have a SHA-256 hash (`1` - SHA-256) with a value equal to the one provided.

In order to avoid manipulation and to authenticate the TLSA records, the TLSA record should be delivered together with its corresponding RRSIG record. The RRSIG record in the example protects a `TLSA` RR, using the algorithm `5` (RSA/SHA1) [122]. Stronger protection is achieved with RSA/SHA-256 or RSA/SHA-512 (IETF RFC 5702), and ECDSA/SHA-256 (IETF RFC 6605). The RRSIG record refers to a domain name with `3` labels with a TTL of `900`. It also contains the valid from/to dates, the *key tag* with the value `14703` and the zone to which the record belongs and from where the signing public key should be retrieved to verify the record.

TABLE VII
TLSA EXAMPLE RECORD FOR ABC.XYZ.COM

Domain name	TTL	Class	Type	Value
_443._tcp.abc.xyz.com.	893	IN	TLSA	3 0 1 (8CB0FC6 C527506A053F4F 14C8464BEBBD6 DEDE2738D1146 8DD953D7D6A3 021F1)
_443._tcp.abc.xyz.com.	893	IN	RRSIG	**TLSA** 5 5 900 (20171117202722 20171018192722 14703 xyz.com. O J4b/O7J+Fh3KV DGG8nH9X5dRS Xgl3j7/S/PbB1Rc zcYfYxdu5C9xZS ALCz0MgVFW6 Kcne74ou5R/Wd +CDKm7mYYGa 28MrtE1boNLuO Ta6kOpFcNjW+U YnMFQuc6S8zh U1G7LH0n3TbM 3rJS0wH0u6J4Ng tiowZS2VlkwPez 3D8=)

Security considerations for the client: DANE specifies the compliant TLS client should support the three fields of TLSA records: certificate usage, selector and matching type. Clients have to support the four certificate usage values (use cases) possible in TLSA records, and also to use selector types 0 (full cert) and 1 (public key) to compare a certificate association with a certificate from the TLS handshake using hashes (SHA-256 or 512) or directly as specified by the matching type field of the TLSA record.

V. Comparison of Certificate Pinning Solutions for IoT/M2M

This section contains a comparison of several aspects of the discussed Certificate Pinning solutions in more detail than other discussions [29], [30], [95], comparing more solutions and with a particular focus on its application for IoT/M2M scenarios. Table VIII, at the end of the section, contains a summary of the most important aspects of the comparison.

A. Use of Side Channels

Side channels allow to perform some of the necessary validations but do not use the original protocol. For example, the use of TLS can trigger the verification but it will be performed using a different protocol or a different instance of the same protocol.

The impact of the use of side channels depends on when the side channel will be used. The first case consists on the use of the side channel during the TLS handshake. The second corresponds to the use of the side channel in a different moment, generating data that can be used during a future TLS handshake.

CT does not require the use of side channels during the TLS handshake but proposes auditors, at client side, to contrast information with monitors asynchronously and optionally to share observed SCTs with other clients by means of a gossip protocol. Despite the use of side channels during the handshake is not mandatory, it is possible to verify SCTs synchronously. This synchronous verification could be necessary in IoT and M2M since devices may suspend their activity during long periods of time difficulting asynchronous verification and the use of gossip protocol. Regarding the difficulties with the gossip protocol, it does not mean an IoT device cannot cooperate, but the effectiveness of the cooperation should be contextualized. An IoT/M2M device can be limited in storage capacity affecting observed SCTs storage. Moreover, it should be taken into account the energy balance between the energy expenditure in transmissions needed to fulfil the device's purpose, and those motivated by the CT protocol.

SK requires the use of side channels to verify the pin. The server delivers the pin (data structure signed with the SK) through TLS together with the certificate, and the pin should be verified. The verification is not necessarily performed online, it can be performed later on, or by a daemon running in the client that downloads verification information from a mirror periodically (to keep SKs fresh for later verification). The same considerations of CT can be applied to SK since devices may not be always online for keeping the information fresh.

TACK does not require side channels since the server provides all the necessary information during the handshake. The server distributes the domain signing key known as TSK with an activation flag. Clients just need to store and remember TSKs for future verification of TACKs signed with stored TSKs. However, the specification describes that TACKs can be optionally downloaded from a trusted third party. Despite there is no description concerning the way clients trust these third parties, downloading and keeping a list of trusted third parties, can be considered as the use of a side channel. Moreover, TACK also proposes clients to share their observed TACKs, requiring additional side channels.

CAA is a mechanism for domain owners to advertise what CA is currently authorized to issue certificates for their domain. CAA does not involve clients so no side channel can be considered.

HSTS and HPKP are limited to HTTP. The first defines policies to enforce the use of TLS and the other makes a promise about the certificate chain a server will use in a future. Since HSTS and HPKP information is delivered simultaneously using HTTP over TLS and needs no further verification, this protocol does not rely on side channels.

Finally, DANE uses DNS with security extensions for verifying the TLS certificate. For that reason it uses a side channel (DNS) immediately before the TLS handshake.

B. Instant Recovery From Loss of Key - IR

One of the most worrying problems of PKI is to have a private key compromised. This problem in PKI have associated several processes as: certificate revocation, which involves the CA to have the certificate added to the revocation list; request of a new certificate; and service re-establishment with the new certificate.

TABLE VIII
SUMMARY OF THE CERTIFICATE PINNING TECHNIQUES COMPARISON. READ ○/●/– AS "NO"/"YES"/"DO NOT APPLY" RESPECTIVELY

	CT	SK	TACK	CAA	HSTS-HPKP	DANE
Use of Side channels	●	●	○	–[1]	○	●
During TLS handshake	○	○	○	–	○	○
Other time	●[2]	●[2]	○	–	○	●[3]
Instant recovery (from loss of key)	●	●	○	●	○	●
Uncertainty time	variable[4]	24 hours				RR TTL
Global/Targeted attack detection	●[5]	●[5]	●[5]	–	●[6]	●[5]
Impersonation only	●[5]	●[5]	●[5]		●[6]	●[5]
Impersonation and poisoning	●[5]	●[5]	–			●[7]
Domain owner detection	●[5a]	●[5b]	●[5c]	●[5d]	●[5c]	●[5c]
Additional TTP	●	●[8]	○[9]	○	○	○
Need TTP list distribution	●	●				
Instant start up	●	●	○	●	●	●
Maximum time until trustworthy verification	variable[4]	24 hours		–		
Unmodified Servers	●[10]	●	●[10]	●	●[11]	●
TLS extension	●[12]	○	●		○	○
Certificate modification	●	●	○		○	○
Necessary Storage	●	●	●	○	●	○
Evidences or relations	●	●	●		●	
Public key lists	●	●	○		○	
Low storage requirements	●	●	●		●	
PKIX only	●	●	●	●	●	○
Security island support						●[13]
Self-signed certificate support						●[13]
Management						
Third party	●	●	●[9]	○	○	○
Domain owner	○	○	●	●	●	●

[1] No TLS client involved in CAA
[2] Entities exchange data asynchronously not necessarily during handshake
[3] Client may obtain DANE records when performing DNS address resolution
[4] $t = \max(\text{MMD} + \text{MVD}, \text{MMD} + \text{MSD})$
[5] Requires global adoption
[5a] A monitor looking for changes in log servers related to the concrete domain should exist
[5b] An entity looking for changes in TimeLine servers related to the concrete domain should exist
[5c] Domain owners should perform self-auditing to notice
[5d] CAA can report inconsistent certificate requests to domain owners
[6] Not for clients visiting the server for the first time
[7] Requires compromising DNSSEC keys
[8] Requires a list of TTPs for TimeLine servers and also mirrors
[9] TACK can optionally rely on third party repositories but describes no TTP enrolment
[10] TLS extensions are pervasively supported thus, it is not considered a modification
[11] HPKP requires modifying HTTP but not TLS servers
[12] Optional, the SCT can be embedded in the certificate
[13] DANE-TA supports domain CA and DANE-EE supports any certificate

In case of loss, CT requires the issuance of a new certificate to be notified to the appropriate logs. The previous certificate is not explicitly revoked since the addition of a new certificate for the domain name implicitly revokes the previous certificate. Despite this process can be triggered immediately after the new certificate is issued, it should be considered the addition of the new certificate can be delayed up to MMD.

After MMD, the new certificate can be asynchronously verified by monitors. Let "Maximum Verification Delay" (MVD) be the time for verification. The total recovery time is bounded to a maximum $t = \text{MMD}+\text{MVD}$. Moreover, the verification of an SCT can be triggered by an auditor, requesting information to a monitor, that eventually will reach a log. The time until the first client connects to a server that provides the new SCT and thus, an auditor verifies, or a monitor requests the verification of a SCT can be called "Maximum SCT Delivery" time (MSD) and corresponds to $t = \text{MMD}+\text{MSD}$. For that reason, the total time is $t = \max(\text{MMD}+\text{MVD},\text{MMD}+\text{MSD})$. That time should be considered since monitors do not need to verify every log server nor look for all the domains, so it can happen an SCT is not verified in a long time.

SK otherwise defines an explicit revocation mechanism that distinguishes between the TLS certificate and the domain SK key. It provides a mechanism to recover the SK using the "incorporate by reference". In general, it is possible to use a new key immediately after the loss of the previous one. Alike CT, there are certain time considerations that should be taken into account.

In SK, there is no definition for the maximum time it can take the addition of a new record. It can be added immediately or in batches, however SK assesses the freshness of the data received by the client being the period of the 24 first hours, in which any information is considered fresh, a significant window of opportunity for an attack. Despite new keys can be used immediately, previous keys can be considered valid during at least 24 hours. So, it would be enough to prevent clients under attack from updating SK information during 24 hours. SK proposes a cooperation protocol to keep a list of fresh TimeLine servers that could reduce the time a compromised TimeLine server is considered fresh. However, IoT devices cannot guarantee they can commit resources to cooperation.

TACK allows to recover from a server key loss but not immediately. A TACK can be substituted according to a plan with a key roll-over mechanism since the proposal supports key overlapping during a time. If a TACK is compromised (which is equivalent to a server key compromised), it can be revoked increasing the field "min_generation". After that, a new key can be announced, but it can take some time until the change to the new key is effective.

HTTP Public Key Pinning protocol (HPKP) makes a promise about the certificate chain that will be used by the server during a time established in the field "max_age" of the pining directive. So new keys cannot be incorporated immediately except for clients connecting for the first time.

In regards to DANE, since clients query DANE information from the DNS before connecting to the server, the minimum time for incorporating a new key after a server key loss is the DANE RR Time To Live (TTL).

C. Global Attack Detection - GA

A global attack in this context is an attack where the server certificate has been substituted by a malicious one, issued by a compromised CA, and every client accessing the victim server can observe the malicious certificate. A malicious certificate, in this case, passes the PKI verification. Due to that, this section evaluates not only the ability of the different proposals to defend clients and domain owners against this kind of attacks, but also the effect of the proposals over domains not adopting each solution.

Two different cases should be considered. In the first, the attacker compromises the CA and tries to impersonate a TLS server using a TLS server with the malicious certificate, but does not add the malicious certificate to the verifiable structure (if any) described by the proposal. This attack will be called "impersonation only attack". In the second, the attacker not only compromises the CA and tries to impersonate a TLS server using a malicious certificate, but also adds the malicious certificate to the verifiable structure. This attack will be called "impersonation with poisoning attack".

In the event of an impersonation only attack, CT clients can defend against the attack if and only if CT is globally adopted by domain owners, since clients will reject any TLS handshake not providing a SCT as a proof. Domain owners will not notice the attack since auditors do not notify monitors about servers not complying with CT, or that complied in the past and have stopped doing so. Clients not supporting CT will not notice the attack and may be exposed to the attack for a long time since the compliant domain owner cannot notice the attack as well, so cannot react to it.

If CT is globally adopted by clients but optionally by servers, a legitimate TLS server using a legitimate certificate, but unwilling to support CT will be indistinguishable from a malicious one. Clients will not finish the handshake since there is no possibility to verify the certificate by means of CT thus, domain owners not supporting CT will be excluded from secure traffic. CT compliant domain owners will also not notice the attack.

If CT is globally adopted by clients and servers, clients will detect the attack, but not domain owners since there is no trace of malicious certificate in log serves.

In the event of a impersonation with poisoning attack, the previous considerations can be applied except that domain owners have a chance to detect the attack if they monitor actively every log sever.

The CT gossip protocol and the auditor-monitor interaction, is orchestrated around the verification of SCTs, but there is no mechanism to signal the lack of an SCT or to detect servers that delivered SCTs in the past and stopped doing so. If auditors were able to notify monitors about servers that stop delivering SCTs, monitors would be able to investigate the issue and notify domain owners contributing to an early detection. Unfortunately, that functionality is not considered in CT.

IoT/M2M global adoption of CT is not plausible due to the heterogeneity of devices and solutions, so global attack detection may not be feasible.

In the event of an impersonation only attack, entities supporting SK can verify if the associated SK key is in force, by requesting every available record for that domain since a given time or sequence number. Hence, if SK is globally adopted by domain owners but optionally by clients, SK aware clients will detect the attack since the TLS certificate provided by the server lacks the signature of the SK.

If SK is globally adopted by clients but not servers, SK would have the same problem as CT, it would be enough for an attacker to impersonate the TLS server with the malicious certificate without offering the SK, so malicious servers will be indistinguishable from legitimate servers not using SK. If SK is globally adopted by both domain owners and clients, the attack will be detected. In any case, domain owners will not notice the attack since no new record will be added to the *TimeLine server*.

An impersonation with poisoning attack in SK is more complex to achieve, since SK is expected to be better protected and less exposed than TLS server keys. Nevertheless, if SK is compromised, and the domain owner does not realize, SK provides no protection since the domain owner cannot revoke the SK.

Cooperation among devices in SK protects from TimeLine server tampering, but not from an attack to a concrete *TimeLine server*.

If TACK is globally adopted by clients, in the event of an impersonation only attack, clients are able to detect TLS servers changing from a legitimate certificate to a malicious one since the later will have no TSK signature. As happens in CT and SK, legitimate servers not supporting TACK will be treated in the same ways as malicious ones. If TACK becomes mandatory for servers and optional for users, only TACK aware clients will be protected from this kind of attacks. Even if TACK is adopted by both domain owners and clients, there is no way domain owners can notice their TLS servers are being impersonated unless clients share not only observed TACKs but report the absence of TACKs for a give domain name.

In principle, the impersonation with poisoning attack has no sense in TACK since there is no verifiable structure, but in practise, it could be possible to achieve the same result by poisoning the third parties that clients can use to obtain trusted TACKs. There is no clear definition of the requirements an entity should fulfil in order to become a trusted party nor definition of the security of the protocol used to fetch TACKs from them. In any case, if the TSK is also compromised, there is no way to detect both attacks.

HPKP allows compatible clients to detect a global attack if the attack happens after they have visited the TLS server for the first time, when they get the trust chain that will be used in the future. Non HPKP aware clients and those aware but visiting the site for the first time after the attack starts would not notice.

In the event of an impersonation only attack, if the domain owner supports DANE, only clients supporting DANE will notice the attack. Clients do not need to rely on previously stored information to detect it. However, if the domain owner does not support DANE, even DANE clients will not notice the attack. The attack cannot be, in principle, detected by the domain owner unless it uses DANE and another party supporting DANE finds an inconsistency between the authenticated authoritative information provided by the DNS and the information obtained during the TLS handshake.

If clients globally adopt DANE but not domain owners, a DANE client cannot be aware of an attack if the domain owner does not support DANE. However, the attack is only effective for a DANE aware domain owner if the attacker compromises not only the CA but also the domain owner's DNS (in order to remove or alter DANE information from the DNS server). The latest can be considered an impersonation with poisoning attack, that will be useful if and only if, the attacker not only compromise the DNS server but also gains access to the DNSSEC ZSKs.

D. Targeted Attack - TA

A targeted attack is similar to a global attack. In this case, the malicious certificate cannot be observed by every client accessing the victim server but by a group of devices that constitute the target group.

The effectiveness of CT, SK, TACK, and HPKP, in this case, is equivalent to that of the global attack discussed in Section V-C. Again, as discussed before, many proposals as CT, SK, and TACK, that do not report TLS servers that stop being compliant, will have an equivalent protection. Others, that expect a different behaviour, as HPKP if first visited before the attack, would detect these kind of attacks promptly. Any expected benefit or potential benefit from cooperation among entities will be reduced in this case since the group able to observe the attack is reduced. Considering IoT/M2M devices, as reasoned before, may not be able to commit resources for cooperation, the degree of protection would be the same.

In the case of DANE, since there is no expected cooperation and devices would not store any previous information, the protection is formally equal to the global attack.

E. Trusted Third Party Dependency - TTP

This section evaluates the impact of additional trusted third parties on the participants. As it has been already discussed in Section III, the increasing number of directly or indirectly trusted CAs (TTPs), distributed with current software, constitutes a problem of trust, and is one of the weakest points in PKI applied to TLS. Certificate Pinning solutions have been proposed to overcome this PKI weakness, preventing some attacks in a more or less efficient way. However, some of the analysed solutions rely on additional TTPs, that may eventually grow and become a problem. Moreover, managing a whole new TTP structure may require extra effort from both client and server sides.

Log servers in CT use a public key pair to sign the SCT they issue upon the addition of new TLS certificates. CT does not clarify how these new key pairs are obtained, distributed or revoked, but it can be supposed the list of CT current authorized key pairs will be distributed together with the software, in the same way as PKI. In this way, as described in the specification, CT requires auditors embedded in clients to have knowledge of CT TTPs or to delegate to external components, but does not clarify how trust will be established between clients and external auditors. Managing lists of CAs is a problem on its own, but worse in IoT/M2M since many devices may get updates later that expected or never be updated. For that reason, requiring IoT/M2M devices to be updated with an additional TTP list constitutes an additional problem.

In the case of SK, every *TimeLine server* has its own key pair. SK clearly states the *TimeLine server* list and their corresponding keys should be distributed with software in the same way as trusted certificates in PKI. Moreover, mirrors in SK should identify by means of their IP/port and public key, so SK introduces two additional key sets (*TimeLine server* and mirrors). Management and distribution of the two lists may be a problem for IoT/M2M devices that can be frequently out of date.

TACK does not need additional TTPs. Every domain has its own TSK that is distributed during the activation. The domain is responsible of the distribution requiring no cooperation from software manufacturers. Alike TACK, HPKP does not introduce new TTPs, it just conveys what CAs should be found in the certificate chain provided by the TLS server during a handshake. Nevertheless, TACK proposes third parties to provide sets of trusted TACKs. Despite these third parties act as mere online stores for TACKs, the list of authorized TACKs may need some management.

DANE relies on DNSSEC for authenticating DANE DNS responses. These responses, which can be used to verify TLS certificates, are signed with a ZSK, which is authorized by the parent with a signature, then successively down to the root that is signed by a reputed CA. The CA signing the unique DNS root is well known and verifiable with the current PKI certificate list. For that reason, DANE does not require the use of additional trusted third parties.

F. Instant Start Up - IS

This section discusses if proposed solutions allows a TLS server to start using a certificate immediately after issuance without generating trust problems in clients.

CT requires the CA to register the certificate in a log server in order to get the SCT, that will be delivered through a extension in TLS or directly embedded in the certificate. CT log servers add new certificates asynchronously in a time shorter than MMD. A certificate provided by a TLS server is trusted if the SCT is signed by a log server public key that the client trusts. If the log server has been compromised, participants may need up to a time $t = \max(\mathrm{MMD}+\mathrm{MVD},\mathrm{MMD}+\mathrm{MSD})$, as discussed in Section V-B, to verify a SCT trustworthily. In IoT/M2M, CT does not suppose any improvement over PKI, since in both cases a TLS certificate can be used immediately and CT provides the same degree of security than PKI during t.

SK does not hamper instant use of TLS certificates since the only requirement is the domain owner to register a domain key (SK) beforehand. After a domain owner registers the SK for its domain, any new certificate can be used instantly. The only requirement for a cross verification is to have the new certificate signed with the domain SK. Alike CT, if a *TimeLine server* has been compromised, it would be necessary to wait more than 24 hours, according to the freshness defined by the protocol, to have a trustworthy cross verification. So the degree of protection, at least during a 24 hours window, would be the same as with plain PKI.

In the case of TACK, there is a time in which the TACKs to remove are distributed together with their replacements thus, a change should be planned ahead. Hence, it is possible to start using a new certificate with its corresponding TACK immediately, but the distribution process can affect the convergence time. It should be also considered that TACK allows third parties to provide trusted TACK sets, so they should be updated accordingly upon changes.

HPKP allows the immediate use of new certificates by TLS servers. The major concern is that clients whose first interaction with the server happens after the change, cannot determine whether the certificate is legitimate or not.

Finally, DANE is considered an alternative independent channel. Since DANE clients will not store any previous record of a given server beyond the TTL time of the record, they will take as valid the new certificate.

G. Unmodified Servers - US

One of the mayor concerns of Certificate Pinning is the addition or alteration of protocols to support the proposals. Significant changes will add complexity to clients complicating the adoption. This section analyses changes required by the previously discussed Certificate Pinning solutions.

CT proposes alternative mechanisms to deliver SCTs to the client during a TLS handshake. The first consists on delivering SCTs using TLS extensions with two possibilities, using a CT TLS extension or an OSCP one. The second describes how the SCT can be directly incorporated to the certificate by registering a pre-certificate before issuing the final certificate. In the first case, CT requires clients to support the TLS extensions described in the proposal. Despite supporting an exotic TLS extension may be challenging, CT considers the use of the widely adopted OSCP TLS extension. Thus, the use of TLS extensions is not a big concern since are widely supported. In the second case, there is no need to change the protocol since the SCT is embedded in the certificate so, the complexity is moved to the PKI software.

Nevertheless, it should be considered CT proposes a gossip protocol for clients to exchange SCTs. That new protocol should be supported by clients. In regard to IoT/M2M, it is coherent to assume TLS extension support is not a problem since they are supported by reference TLS implementations as OpenSSL. Concerning the gossip protocol, and any other protocol supporting the interaction among parties (i.e., auditor-monitor), their support can be more challenging since additional software requires to be updated, but more precisely due to the reasons discussed in Section V-A.

In SK specification, it is suggested the SK signature over the certificate chain is transmitted during the handshake as part of the server certificate. In this case, there is no need for modifying the protocol.

Alike CT, TACK requires the use of TLS extensions that are well supported in IoT. However, TACK may need support for trusted third parties providing trusted TACKs to clients.

HPKP and HSTS require changes in HTTP headers in both clients and servers thus, servers can deliver the chain and clients process it. Clients unable to understand those headers should ignore them, so it would not be a problem in IoT/M2M.

Finally, DANE does not require any modification to the server or clients. Due to that, it can be used whenever the client DNS resolver supports DNSSEC and is able to process DANE RRs.

H. Necessary Storage - NS

Certificate Pinning solutions pursue to provide some of the following characteristics: verifiable structures that can be audited; cross verification for clients; or grounds for trust based on collections of evidences. Hence, some information should be stored in different entities considered by the solution. It is necessary to evaluate the impact of these proposals on the storage.

CT requires the storage of SCTs at client side (auditor) to be verified by monitors or shared by means of the gossip protocol. Additionally, CT requires storing a list of TTPs or log servers that should be trusted. According to the expected size of SCTs and pubic keys, it would not be a problem event for IoT/M2M unless the number of SCTs or keys grow exaggeratedly. Moreover, CT requires auditors to store evidences (pieces of the MT or node sequences) that will be used to detect errors and malicious behaviours.

SK requires the storage of TimeStamps, SKs and lists of mirrors with their corresponding keys to feed the synchronization protocol and allow the verification. The specification makes a study of the estimated demanded size according to the data entropy, showing the requirements can be afforded by IoT/M2M devices.

The TACK specification defines "stores" as the place within the client where to store observed and downloaded TACKs. According to the TACK structure, it should not be a problem to store a reasonable number of TACKs in IoT/M2M.

Alike the aforementioned solutions, HPKP requires storing both HPKP associations and HSTS policies in the client, but it should not be a significant information volume for IoT/M2M.

DANE does not require the client to store any information.

VI. Open Challenges and Research Directions

This section describes open challenges and research directions to improve TLS/certificate pinning in the context of IoT and M2M. **Adoption** is among the most critical problems in Certificate Pinning. On the one hand, the lack of a global adoption can harden attack detection as discussed in Sections V-C and V-D. According to the discussion, clients may be unable to differentiate malicious servers from those not adopting the solution. On the other hand, requiring global adoption may hamper innovation in server authentication and security protocols. There is an interesting research area on improving proposed certificate pinning protocols, or other information systems, to convey extended information to clients so they can differentiate among services not supporting a given solution voluntarily from those that may be malicious. Beyond the current possible outcome of these solutions: of secure (connect) or not secure (do not connect); it would be interesting to increase the outcome space by defining a third option that represents **uncertainty**.

This is important for IoT/M2M devices. Commercially available off-the-shelf software or hardware, which is going to be integrated into an IoT device ecosystem long time after its manufacturing process, it may become not only a security threat [123] due to certain vulnerabilities detected after production, but also unusable if it cannot reach backends. Certificate Pinning solutions, if adopted globally, can lead to a scenario in which devices cannot reach backends to receive the appropriate updates and can be easily misused.

Future reseach considering uncertainty, should explore solutions where devices engage in extended authentication by other means, for instance using application protocol layers, that eventually let them reach the appropriate backends. Also solutions based on advertising alternative routes to the service as part of the certificate pinning solution, should be explored. This is presently incorporated of SK (see Section IV-B) to prevent denial of service.

Every of the certificate pinning solutions analyzed in Sections IV-A to IV-F focus on making it difficult for malicious endpoints to forge others' identity by means of compromised or cooperating CAs. As stated in the analysis, despite every solution has its own mechanism, the combination of several solutions or even their hybridization may contribute to a better one.

The results obtained from various solutions can be combined for judging the security of a given service using PKI by means of a risk assessment engine. Several works [124]–[126] argue that an evidence collection process and its ulterior analysis can be of paramount importance to overcome several attacks. These solutions can be also incorporated into IoT/M2M environments and carried out by constrained devices.

When it comes to the hybridization, there are several interesting topics to be addressed. For instance, CT (see Section IV-A) and SK (see Section IV-B) keep a central structure that according to several cryptographic properties or using timestamps, are useful to detect several attacks, if inspected regularly for misuse, and to find out when the attack happened. Despite these solutions have demonstrated their effectiveness under certain circumstances, the control of the structure may fall under a single private entity diminishing Internet neutrality. Other solutions, as TACK (see Section IV-C) or DANE (see Section IV-F2) do not keep a historical record, so they cannot be audited for misuse, but allow clients to get an immediate response regarding the trustworthiness of a given service. A very interesting research can be carried out proposing the use of verifiable structures locally combined with solutions providing an immediate response. In such a way, solutions as DANE or TACK, could not only provide an immediate verification but also keep an auditable historic record of the credentials they vouch for.

Finally, the support of constrained devices opens an interesting research. Considering the limitations present on nowadays IoT/M2M devices, certificate pinning solutions should address the verification of the certificates requiring a minimum storage (as reasoned in Section V-H) and energy expenditure. Moreover, as part of the offloading strategies, it is becoming a trend to move certain communication services to the cloud and to the fog. Since the access to these services is performed using TLS or DTLS, certificate pinning solutions play an important role. It is worth mentioning the work of DNS PRIVate Exchange (DPRIVE) Working Group of the IETF on DNS privacy, that focuses on providing confidentiality to DNS transactions relying on TLS/DTLS for this purpose.

As part of the offloading strategy in IoT/M2M, DNS resolvers will probably be replaced by a stub resolver that forwards any DNS request to a resolver located near the device relying on DNS over (D)TLS to guarantee privacy. Due to that, the correct authentication of the resolver's certificate is especially worrying since it enables a huge number of attacks.

VII. Conclusion

TLS has been widely used on Internet with a high degree of success. Thus, it is reasonable it has been adopted by IoT and M2M protocols. However, TLS and PKI tackle with static services whereas services in IoT/M2M can be dynamic and several orders of magnitude more numerous. Moreover, IoT/M2M require adequate security support to address limitations in processor, memory and battery and multiple vendors.

The global picture of PKI presented in the article, identifies problems (Section III-A) in regards to the number of authorities that combined with the increasing number of endpoints to protect, reveals that alternative certificate validation and attack detection mechanisms are desirable to guarantee an adequate degree of security.

Certificate Pinning technique (Section IV) is being envisaged as a way to strengthen the trust in the system. This article has reviewed and compared the different proposals for certificate pinning in the context of IoT/M2M. Section V has analyzed the proposals in different aspects. Let us review the main points we have found.

Only CT, SK and DANE use side channels. CT for exchanging SCTs data, SK to verify the pin, and DANE for DNS initial queries. In case of key loss, CT, SK, CAA and DANE provide instant recovery, with different but bounded uncertainty times.

In a global attack scenario, IoT/M2M global adoption of CT is required. This is not plausible due to the heterogeneity of devices and solutions, so global attack detection may not be feasible. Similar problems appear for SK in which malicious servers will be indistinguishable from legitimate servers not using SK. This is also the case of TACK if adopted by both domain owners and clients since there is no way domain owners can notice their TLS servers are being impersonated unless clients share not only observed TACKs but report the absence of TACKs for a give domain name. HPKP only protects clients from a global attack to already visited and trusted TLS servers. This may strongly limit the dynamicity of interactions with the IoT increasing huge number of apps, services and devices. DANE also requires global adoption in servers and clients to protect from an impersonation only attack, if the domain owner supports DANE, only clients supporting DANE will notice the attack. There are no significant differences regarding instant start up, and server modification.

CT and SK are the only approaches that require additional TTPs. The need for extra storage is common to all proposals but CAA and DANE, since they get the information from the DNS. DANE is the only that supports security islands and scenarios of domain owner self-signed certificates. CT, SK and TACK require third party management, while the others can be managed by the domain owner.

From all that, we conclude that DANE offers a set of characteristics desirable for IoT and its dinamic scenarios. DANE uses an existing side channel (DNS) but does not require using a new side channel during TLS handshake. A query for the address record is most often performed before the TLS handshake. Thus services can frequently update certificates and instantly recover from loss of keys, without imposing convergence times or involving third parties. DANE does not require collaboration between IoT limited devices in global attack scenarios, nor additional TTPs, lowering operation and management costs.

References

[1] C. M. Medaglia and A. Serbanati, "An overview of privacy and security issues in the Internet of Things," in *The Internet of Things*, D. Giusto, A. Iera, G. Morabito, and L. Atzori, Eds. New York, NY, USA: Springer, 2010, pp. 389–395. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-1-4419-1674-7_38

[2] R. H. Weber, "Internet of Things—New security and privacy challenges," *Comput. Law Security Rev.*, vol. 26, no. 1, pp. 23–30, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0267364909001939

[3] *Internet of Things: Privacy & Security in a Connected World*, Federal Trade Commission, Washington, DC, USA, 2015.

[4] C. Richardson. (2016). *Microservice Architecture Patterns and Best Practices*. Accessed: Feb. 12, 2016. [Online]. Available: http://microservices.io/index.html

[5] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Softw.*, vol. 32, no. 2, pp. 50–54, Mar./Apr. 2015.

[6] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May/Jun. 2016.

[7] C. Yang, P. Liang, and P. Avgeriou, "A systematic mapping study on the combination of software architecture and agile development," *J. Syst. Softw.*, vol. 111, pp. 157–184, Jan. 2016.

[8] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput. (MCC)*, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[9] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23511–23528, 2018.

[10] "OpenFog architecture overview," Fremont, CA, USA, OpenFog Consortium, White Paper, Feb. 2016.

[11] D. Evans, "The Internet of Things: How the next evolution of the Internet is changing everything," vol. 1, San Jose, CA, USA, CISCO, White Paper, pp. 1–11, 2011.

[12] B. Plale *et al.*, "CASA and LEAD: Adaptive cyberinfrastructure for real-time multiscale weather forecasting," *Computer*, vol. 39, no. 11, pp. 56–64, Nov. 2006.

[13] T. Martin and J. Healey, "2006's wearable computing advances and fashions," *IEEE Pervasive Comput.*, vol. 6, no. 1, pp. 14–16, Jan./Mar. 2007.

[14] G. Pollock, D. Thompson, J. Sventek, and P. Goldsack, "The asymptotic configuration of application components in a distributed system," College Sci. Eng., School Comput. Sci., Univ. Glasgow, Glasgow, U.K., Rep., 1998. [Online]. Available: http://www.dcs.gla.ac.uk/~joe/auxiliary/papers/Personal/AsymptoticConfig.pdf

[15] R. T. Fielding, "REST: Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Inf. Comput. Sci., Univ. California at Irvine, Irvine, CA, USA, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[16] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over low-power wireless personal area networks (6LoWPANs): Overview, assumptions, problem statement, and goals," RFC 4919, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–12, Aug. 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4919.txt

[17] D. Karaman *et al.*, "Managing 6LoWPAN sensors with CoAP on Internet," in *Proc. 23rd Signal Process. Commun. Appl. Conf. (SIU)*, 2015, pp. 1389–1392.

[18] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," RFC 7252, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–112, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7252.txt

[19] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," RFC 5246, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–104, Aug. 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5246.txt

[20] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," RFC 6347, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–32, Jan. 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6347.txt

[21] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is QUIC? Provable security and performance analyses," in *Proc. IEEE Symp. Security Privacy*, May 2015, pp. 214–231.

[22] P. K. Verma *et al.*, "Machine-to-machine (M2M) communications: A survey," *J. Netw. Comput. Appl.*, vol. 66, pp. 83–105, May 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516000990

[23] H. Tschofenig and T. Fossati, "Transport layer security (TLS)/ datagram transport layer security (DTLS) profiles for the Internet of Things," RFC 7925, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–61, Jul. 2016. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7925.txt

[24] D. Cooper *et al.*, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," RFC 5280, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–151, May 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5280.txt

[25] P. Yee, "Updates to the Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," RFC 6818, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–8, Jan. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6818.txt

[26] M. L. Sichitiu and M. Kihl, "Inter-vehicle communication systems: A survey," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 2, pp. 88–105, 2nd Quart., 2008.

[27] J. Kim, J. Lee, J. Kim, and J. Yun, "M2M service platforms: Survey, issues, and enabling technologies," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 61–76, 1st Quart., 2014.

[28] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the Internet of Things: A survey of existing protocols and open research issues," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1294–1312, 3rd Quart., 2015.

[29] J. Clark and P. C. van Oorschot, "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements," in *Proc. IEEE Symp. Security Privacy*, Berkeley, CA, USA, 2013, pp. 511–525.

[30] J. Amann *et al.*, "Mission accomplished? HTTPS security after diginotar," in *Proc. Internet Meas. Conf. (IMC)*, London, U.K., 2017, pp. 325–340. [Online]. Available: http://doi.acm.org/10.1145/3131365.3131401

[31] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2027–2051, 3rd Quart., 2016.

[32] S. Kent and K. Seo, "Security architecture for the Internet protocol," RFC 4301, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–101, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4301.txt

[33] C. Kaufman, "Internet key exchange (IKEv2) protocol," RFC 4306, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–99, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4306.txt

[34] M. H. Behringer, "End-to-end security," *Internet Protocol J.*, vol. 12, no. 3, pp. 20–26, Sep. 2009. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac123/ac147/archived_issues/ipj_12-3/ipj_12-3.pdf

[35] S. Cantor, J. Kemp, R. Philpott, and E. Maler, *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, document saml-core-2.0-os, Org. Adv. Struct. Inf. Stand., Mar. 2015. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[36] R. Canetti and H. Krawczyk, "Security analysis of IKE's signature-based key-exchange protocol," in *Proc. Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA, 2002, pp. 143–161.

[37] H. Krawczyk, "SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie–Hellman and its use in the IKE protocols," in *Proc. 23rd Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA, Aug. 2003, pp. 400–425. doi: 10.1007/978-3-540-45146-4_24.

[38] W. Aiello *et al.*, "Just fast keying: Key agreement in a hostile Internet," *ACM Trans. Inf. Syst. Security*, vol. 7, no. 2, pp. 242–273, 2004.

[39] A. Jungmaier, E. Rescorla, and M. Tuexen, "Transport layer security over stream control transmission protocol," RFC 3436, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–9, Dec. 2002. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3436.txt

[40] B. Aboba and D. Simon, "PPP EAP TLS authentication protocol," RFC 2716, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–24, Oct. 1999. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2716.txt

[41] T. Dierks and C. Allen, "The TLS protocol version 1.0," RFC 2246, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–80, Jan. 1999. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2246.txt

[42] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "Transport layer security (TLS) extensions," RFC 3546, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–29, Jun. 2003. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3546.txt

[43] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography* (Discrete Mathematics and Its Applications), 1st ed. Boca Raton, FL, USA: CRC Press, Oct. 1996. [Online]. Available: http://amazon.com/o/ASIN/0849385237/

[44] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs Codes Cryptography*, vol. 2, no. 2, pp. 107–125, 1992. doi: 10.1007/BF00124891.

[45] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, "Transport layer security (TLS) session resumption without server-side state," RFC 5077, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–20, Jan. 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5077.txt

[46] R. Hummen *et al.*, "6LoWPAN fragmentation attacks and mitigation mechanisms," in *Proc. 6th ACM Conf. Security Privacy Wireless Mobile Netw. (WiSec)*, Budapest, Hungary, 2013, pp. 55–66. [Online]. Available: http://doi.acm.org/10.1145/2462096.2462107

[47] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure CoAP for the Internet of Things," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3711–3720, Oct. 2013.

[48] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the TLS 1.3 handshake protocol candidates," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, Denver, CO, USA, 2015, pp. 1197–1210.

[49] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe, "Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication," in *Proc. IEEE Symp. Security Privacy (SP)*, San Jose, CA, USA, 2016, pp. 470–485.

[50] D. J. Wu, A. Taly, A. Shankar, and D. Boneh, "Privacy, discovery, and authentication for the Internet of Things," in *Computer Security—ESORICS 2016*, I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, Eds. Cham, Switzerland: Springer Int., 2016, pp. 301–319. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-45741-3_16

[51] Y. Sheffer, R. Holz, and P. Saint-Andre, "Summarizing known attacks on transport layer security (TLS) and datagram TLS (DTLS)," RFC 7457, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–13, Feb. 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7457.txt

[52] E. Rescorla. (2009). *Understanding the TLS Renegotiation Attack*. Accessed: Apr. 3, 2018. [Online]. Available: http://www.educatedguesswork.org/2009/11/understanding_the_tls_renegoti.html

[53] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov, "Transport layer security (TLS) renegotiation indication extension," RFC 5746, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–15, Feb. 2010. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5746.txt

[54] T. Duong and J. Rizzo, "Here come the ⊕ Ninjas," vol. 320, 2011. Accessed: Apr. 23, 2018. [Online]. Available: http://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf

[55] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt, "On the security of RC4 in TLS," in *Proc. SEC*, 2013, pp. 305–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=2534766.2534793

[56] A. Popov, "Prohibiting RC4 cipher suites," RFC 7465, Internet Eng. Task Force, Fremont, CA, USA, pp. 1–6, Feb. 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7465.txt

[57] J. R. T. Duong. (Sep. 2012). *The CRIME Attack*. [Online]. Available: https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit#slide=id.g1d134dff_1_222

[58] Y. Gluck, N. Harris, and A. Prado. (2013). *BREACH: Reviving the CRIME Attack*. [Online]. Available: http://breachattack.com

[59] B. Smyth and A. Pironti, "Truncating TLS connections to violate beliefs in Web applications," presented at the 7th USENIX Workshop Offensive Technol., Washington, DC, USA, 2013. [Online]. Available: https://www.usenix.org/conference/woot13/workshop-program/presentation/Smyth

[60] N. J. A. Fardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in *Proc. IEEE Symp. Security Privacy*, Berkeley, CA, USA, Mar. 2013, pp. 526–540.

[61] P. Gutmann, "Encrypt-then-MAC for transport layer security (TLS) and datagram transport layer security (DTLS)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7366, pp. 1–7, Sep. 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7366.txt

[62] R. Barnes, M. Thomson, A. Pironti, and A. Langley, "Deprecating secure sockets layer version 3.0," Internet Eng. Task Force, Fremont, CA, USA, RFC 7568, pp. 1–7, Jun. 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7568.txt

[63] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel, "A cross-protocol attack on the TLS protocol," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, Raleigh, NC, USA, 2012, pp. 62–72. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382206

[64] C. Paar *et al.*, "DROWN: Breaking TLS using SSLv2," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 689–706. [Online]. Available: https://www.usenix.org/sites/default/files/sec16_full_proceedings.pdf

[65] D. Chadwick, *Understanding X. 500: The Directory*. London, U.K.: Chapman & Hall, 1994.

[66] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet X.509 public key infrastructure certificate and CRL profile," Internet Eng. Task Force, Fremont, CA, USA, RFC 2459, pp. 1–129, Jan. 1999. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2459.txt

[67] W. Yeong, T. Howes, and S. Kille, "X.500 lightweight directory access protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 1487, pp. 1–21, Jul. 1993. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1487.txt

[68] S. Hardcastle-Kille, "X.500 and domains," Internet Eng. Task Force, Fremont, CA, USA, RFC 1279, pp. 1–15, Nov. 1991. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1279.txt

[69] K. Igoe and D. Stebila, "X.509v3 certificates for secure shell authentication," Internet Eng. Task Force, Fremont, CA, USA, RFC 6187, pp. 1–16, Mar. 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6187.txt

[70] R. L. Barnes, "Let the names speak for themselves: Improving domain name authentication with DNSSEC and DANE," *Internet Protocol J.*, vol. 15, no. 1, pp. 201–213, Mar. 2015.

[71] H. Hoogstraaten, "Black Tulip: Report of the investigation into the DigiNotar certificate authority breach," Fox-IT BV, Delft, The Netherlands, Rep. PR-110202, 2012.

[72] EF Foundation. (2010). *The EFF SSL Observatory*. Accessed: Apr. 11, 2018. [Online]. Available: https://www.eff.org/observatory

[73] Qualys Labs. (2009). *SSL Server Test*. Accessed: Apr. 19, 2018. [Online]. Available: https://www.ssllabs.com/ssltest/index.html

[74] J. A. Berkowsky and T. Hayajneh, "Security issues with certificate authorities," in *Proc. IEEE 8th Annu. Ubiquitous Comput. Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2017, pp. 449–455.

[75] J. B. P. Eckersley, "Is the SSliverse a safe place?" in *Proc. 27th Chaos Commun. Congr. (CCC)*, 2010, pp. 1–56.

[76] N. Good *et al.*, "Stopping spyware at the gate: A user study of privacy, notice and spyware," in *Proc. ACM Symp. Usable Privacy Security (SOUPS)*, 2005, pp. 43–52. [Online]. Available: http://doi.acm.org/10.1145/1073001.1073006

[77] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying wolf: An empirical study of SSL warning effectiveness," in *Proc. 18th Conf. USENIX Security Symp. (SSYM)*, Berkeley, CA, USA, 2009, pp. 399–416. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855768.1855793

[78] P. Eckersley. (2011). *How Secure Is HTTPS Today? How Often Is It Attacked?*. Accessed: Apr. 11, 2018. [Online]. Available: https://www.eff.org/deeplinks/2011/10/how-secure-https-today

[79] C. Soghoian and S. Stamm, "Certified lies: Detecting and defeating government interception attacks against SSL (short paper)," in *Proc. 15th Int. Conf. Financ. Cryptography Data Security (FC)*, 2012, pp. 250–259. doi: 10.1007/978-3-642-27576-0_20.

[80] P. Eckersley and J. Burns. (2011). *The (Decentralized) SSL Observatory*. Accessed: Apr. 11, 2017. [Online]. Available: http://static.usenix.org/events/sec11/tech/slides/eckersley.pdf

[81] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the https certificate ecosystem," in *Proc. ACM Conf. Internet Meas. Conf. (IMC)*, Barcelona, Spain, 2013, pp. 291–304. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504755

[82] SSL Labs. (2017). *SSL Pulse—Survey of the SSL Implementation of the Most Popular Web Sites*. Accessed: Apr. 21, 2017. [Online]. Available: https://www.trustworthyinternet.org/ssl-pulse/

[83] I. Ristic. (2012). *SSL Pulse—To Make SSL More Secure and Pervasive*. Accessed: Feb. 12, 2017. [Online]. Available: https://www.trustworthyinternet.org/blog/2012/4/25/ssl-pulse-to-make-ssl-more-secure-and-pervasive/

[84] W. Chou, "Inside SSL: Accelerating secure transactions," *IT Prof.*, vol. 4, no. 5, pp. 37–41, Sep./Oct. 2002.

[85] G. K. Pandya. (2013). *Nokia's MITM on HTTPS Traffic From Their Phone*. Accessed: Apr. 23, 2018. [Online]. Available: https://gaurangkp.wordpress.com/2013/01/09/nokia-https-mitm/

[86] S. Schultze and S. B. Roosa, "Trust darknet: Control and compromise in the Internet's certificate authority model," *IEEE Internet Comput.*, vol. 17, no. 3, pp. 18–25, May/Jun. 2013. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MIC.2013.27

[87] R. Housley, S. Ashmore, and C. Wallace, "Trust anchor management protocol (TAMP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 5934, pp. 1–91, Aug. 2010. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5934.txt

[88] S. Ashmore and C. Wallace, "Using trust anchor constraints during certification path processing," Internet Eng. Task Force, Fremont, CA, USA, RFC 5937, pp. 1–8, Aug. 2010. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5937.txt

[89] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," Internet Eng. Task Force, Fremont, CA, USA, RFC 6962, pp. 1–27, Jun. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6962.txt

[90] R. Merkle, "Method of providing digital signatures," U.S. Patent 4 309 569, Jan. 1982. [Online]. Available: https://www.google.com/patents/US4309569

[91] G. Becker *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis*, Ruhr-Universität Bochum, Bochum, Germany, 2008.

[92] B. Laurie and C. Doctorow, "Secure the Internet," *Nature*, vol. 491, pp. 325–326, Nov. 2012.

[93] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in *Proc. 18th Conf. USENIX Security Symp. (SSYM)*, Montreal, QC, Canada, 2009, pp. 317–334. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855768.1855788

[94] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *Proc. 11th Int. Workshop FSE*, Delhi, India, Feb. 2004, pp. 371–388. doi: 10.1007/978-3-540-25937-4_24.

[95] DigiCert Inc. (2016). *Certificate Transparency*. [Online]. Available: https://www.certificate-transparency.org/

[96] S. Farrell, "Other certificates extension," Internet Eng. Task Force, Fremont, CA, USA, RFC 5697, pp. 1–8, Nov. 2009. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5697.txt

[97] D. Eastlake, III, "Transport layer security (TLS) extensions: Extension definitions," Internet Eng. Task Force, Fremont, CA, USA, RFC 6066, pp. 1–25, Jan. 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6066.txt

[98] A. Deacon and R. Hurst, "The lightweight online certificate status protocol (OCSP) profile for high-volume environments," Internet Eng. Task Force, Fremont, CA, USA, RFC 5019, pp. 1–22, Sep. 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5019.txt

[99] C. Herley, "So long, and no thanks for the externalities: The rational rejection of security advice by users," in *Proc. ACM Workshop New Security Paradigms Workshop (NSPW)*, 2009, pp. 133–144. [Online]. Available: http://doi.acm.org/10.1145/1719030.1719050

[100] P. Eckersley. (2012). *Sovereign Key Cryptography for Internet Domains*. Accessed: Jan. 25, 2018. [Online]. Available: https://www.eff.org/sovereign-keys

[101] D. M. Gordon, "A survey of fast exponentiation methods," *J. Algorithms*, vol. 27, no. 1, pp. 129–146, 1998.

[102] S. Santesson *et al.*, "X.509 Internet public key infrastructure online certificate status protocol—OCSP," Internet Eng. Task Force, Fremont, CA, USA, RFC 6960, pp. 1–41, Jun. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6960.txt

[103] M. Marlinspike, "Trust assertions for certificate keys," Internet Eng. Task Force, Fremont, CA, USA, Working Draft, Internet-Draft draft-perrin-tls-tack-02, Jan. 2013. [Online]. Available: http://www.ietf.org/internet-drafts/draft-perrin-tls-tack-02.txt

[104] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Security*, vol. 1, no. 1, pp. 36–63, 2001. doi: 10.1007/s102070100002.

[105] "Digital signature standard (DSS)," Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, Rep. FIPS PUB 186-4, 2009.

[106] *Secure Hash Standard*, FIPS Standard 180-2, 2002.

[107] P. Hoffman and J. Schlyter, "The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA," Internet Eng. Task Force, Fremont, CA, USA, RFC 6698, pp. 1–37, Aug. 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6698.txt

[108] R. Danyliw, J. Meijer, and Y. Demchenko, "The incident object description exchange format," Internet Eng. Task Force, Fremont, CA, USA, RFC 5070, pp. 1–92, Dec. 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5070.txt

[109] J. Hodges, C. Jackson, and A. Barth, "HTTP strict transport security (HSTS)," Internet Eng. Task Force, Fremont, CA, USA, RFC 6797, pp. 1–46, Nov. 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6797.txt

[110] E. Rescorla, "HTTP over TLS," Internet Eng. Task Force, Fremont, CA, USA, RFC 2818, pp. 1–7, May 2000. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2818.txt

[111] C. Jackson and A. Barth, "ForceHTTPS: Protecting high-security Web sites from network attacks," in *Proc. 17th Int. World Wide Web Conf.*, 2008, pp. 525–534.

[112] C. Jackson and A. Barth, "Beware of finer-grained origins," in *Proc. Web Security Privacy (W2SP)*, 2008, pp. 1–7. [Online]. Available: http://seclab.stanford.edu/websec/safelock/fgo.pdf

[113] R. Fielding *et al.*, "Hypertext transfer protocol—HTTP/1.1," Internet Eng. Task Force, Fremont, CA, USA, RFC 2616, pp. 1–176, Jun. 1999. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2616.txt

[114] C. Evans, C. Palmer, and R. Sleevi, "Public key pinning extension for HTTP," Internet Eng. Task Force, Fremont, CA, USA, RFC 7469, pp. 1–28, Apr. 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7469.txt

[115] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Resource records for the DNS security extensions," Internet Eng. Task Force, Fremont, CA, USA, RFC 4034, pp. 1–29, Mar. 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4034.txt

[116] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Protocol modifications for the DNS security extensions," Internet Eng. Task Force, Fremont, CA, USA, RFC 4035, pp. 1–53, Mar. 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4035.txt

[117] Z. Hu *et al.*, "Specification for DNS over transport layer security (TLS)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7858, pp. 1–19, May 2016. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7858.txt

[118] T. Reddy, D. Wing, and P. Patil, "DNS over datagram transport layer security (DTLS)," Internet Eng. Task Force, Fremont, CA, USA, RFC 8094, pp. 1–13, Feb. 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8094.txt

[119] D. Eastlake, III, "Domain name system security extensions," Internet Eng. Task Force, Fremont, CA, USA, RFC 2535, pp. 1–47, Mar. 1999. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2535.txt

[120] R. Barnes, "Use cases and requirements for DNS-based authentication of named entities (DANE)," Internet Eng. Task Force, Fremont, CA, USA, RFC 6394, pp. 1–12, Oct. 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6394.txt

[121] O. Gudmundsson, "Adding acronyms to simplify conversations about DNS-based authentication of named entities (DANE)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7218, pp. 1–5, Apr. 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7218.txt

[122] D. Eastlake, III, "RSA/SHA-1 SIGs and RSA KEYs in the domain name system (DNS)," Internet Eng. Task Force, Fremont, CA, USA, RFC 3110, pp. 1–7, May 2001. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3110.txt

[123] R. J. Ellison and C. Woody, "Supply-chain risk management: Incorporating security into software development," in *Proc. IEEE 43rd Hawaii Int. Conf. Syst. Sci. (HICSS)*, 2010, pp. 1–10.

[124] P. Arias-Cabarcos *et al.*, "A metric-based approach to assess risk for 'on cloud' federated identity management," *J. Netw. Syst. Manag.*, vol. 20, no. 4, pp. 513–533, 2012.

[125] C. Skalka, X. S. Wang, and P. Chapin, "Risk management for distributed authorization," *J. Comput. Security*, vol. 15, no. 4, pp. 447–489, 2007.

[126] N. Li and J. Feigenbaum, "Nonmonotonicity, user interfaces, and risk assessment in certificate revocation," in *Proc. Int. Conf. Financ. Cryptography*, 2001, pp. 166–177.

Daniel Díaz-Sánchez received the Ph.D. degree in telematics engineering from the University Carlos III of Madrid, where he is an Associate Professor. His research interests include distributed authentication/authorization, content protection, distributed computing, fog computing, IoT, and smart cities.

Andrés Marín-Lopez received the Ph.D. degree in telecommunication engineering from the Universidad Politécnica of Madrid. He is an Associate Professor with the University Carlos III of Madrid. His research interests include ubiquitous computing: limited devices, trust, and security in next-generation networks.

Florina Almenárez Mendoza received the Ph.D. degree in telematics engineering from the University Carlos III of Madrid, where she is an Associate Professor. Her research interests include trust and reputation management models, identity management, and security architectures in ubiquitous computing.

Patricia Arias Cabarcos received the Ph.D. degree in telematics engineering from the University Carlos III of Madrid. She is an Alexander von Humboldt Post-Doctoral Fellow with Universität Mannheim. Her interests include authentication, identity management, and information systems security.

R. Simon Sherratt received the Ph.D. degree in electronic engineering from the University of Salford, U.K. He is a Professor with the University of Reading, U.K. His research interests include smart homes, personal area networks, wearable devices, and their security, all with a focus for mass-market healthcare.