proceedings

MDPI

Proceedings

DNS-Based Dynamic Authentication for Microservices in IoT †

Daniel Díaz Sánchez *, Andrés Marín López, Florina Almenares Mendoza and Patricia Arias Cabarcos ‡

Telematic Engineering Department, Carlos III University of Madrid, Avenida de la Universidad 30, 28911 Leganés, Spain; dds@it.uc3m.es (D.D.S.); amarin@it.uc3m.es (A.M.L.); florina@it.uc3m.es (F.A.M.); ariasp@it.uc3m.es (P.A.C.)

* Correspondence: dds@it.uc3m.es

† Presented at the 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018), Punta Cana, Dominican Republic, 4–7 December 2018.

‡ currently supported by an Alexander von Humboldt fellowship at Universität Mannheim.

Published: 25 October 2018

check for updates

Abstract: IoT devices provide with real-time data to a rich ecosystems of services and applications that will be of uttermost importance for ubiquitous computing. The volume of data and the involved subscribe/notify signaling will likely become a challenge also for access and core netkworks. Designers may opt for microservice architectures and fog computing to address this challenge while offering the required flexibility for the main players of ubiquitous computing: nomadic users. Microservices require strong security support for Fog computing, to rely on nodes in the boundary of the network for secure data collection and processing. IoT low cost devices face outdated certificates and security support, due to the elapsed time from manufacture to deployment. In this paper we propose a solution based on microservice architectures and DNSSEC, DANE and chameleon signatures to overcome these difficulties. We will show how trap doors included in the certificates allow a secure and flexible delegation for off-loading data collection and processing to the fog. The main result is showing this requires minimal manufacture device configuration, thanks to DNSSEC support.

Keywords: IoT; microservices; DNSSEC; DANE; chameleon signatures

1. Introduction

Internet of Things (IoT) devices are equipped with multiple sensors that gather data in the closest environment to the user. From health devices to sports wrists and domestic appliances, these devices provide with real-time data to a rich ecosystems of services and applications which can personalize the user environment as expected in the ubiquitous computing scenarios. The volume of data and the involved subscribe/notify or polling signaling will likely become a challenge also for access and core netkworks. Many designers have identified Cloud computing, Edge computing or Fog computing[1] as a candidate to alleviate the core of the network from this background traffic. IoT devices have a second characteristic: the embedded software and firmware are challenging for users to update. According to the January 2015 Federal Trade Commission report [1] on IoT device security, authentication APIs

[1] Despite there are subtle differences, both concepts address moving data closer to the user and perform some processing near the client requiring the instantiation of new services. We address the protection of those services, so due to that, now on we will refer only to Fog.

and system updates are the biggest problems of IoT market. The reduced cost of many of IoT devices make them prone to present security design flaws, due to the popularization and reuse of mature but outdated hardware, development platforms and software development kits.

This is where microservices modular alternative offer an immediate benefit: update of components is easier than updating the whole system (firmware). Microservices structures an application in a set of loosely coupled services that implement different functionalities. An application can be divided in a set of little dependent modular components which can be separately instantiated and allows continuous delivery and deployment of complex distributed applications. Microservice architecture favors the operation and management such as the easy replacement or update of a microservice. Microservices require strong security support for Fog computing, to rely on nodes in the boundary of the network for secure data collection and processing.

Using containers for microservice virtualization favours the easy update of applications. The nomadicity of those services, when they are part of an application involving a personal and mobile IoT device, is needed to use opportunistically the Fog. IoT applications can declare their required services so that the Fog can reply with a best-effort offer to meet such requirements. The Fog must specify in its offer the hosting and execution of the declared services under a given quality, the address and name assignment, and the cooperation to guarantee the secure access to them. Thus among the properties of a service in the Fog we can identify the following: identification (IP or domain name resolvable within the Fog); reachability (Fog or even Internet); and the transparent migration, replication or load balance of the service instance.

The problem this paper addresses is how to provide Fog deployed service properties in a secure way. Today PKI is still the most popular and common standard to use, but there is not an easy solution to outdated certificates of low-end IoT devices or how can IoT applications dynamically deploy the involved microservices securely in the Fog. In Section 2 we review certificate pinning as the current solution to enhance PKI security. Section 3 reviews the Domain Name System Security Extensions (DNSSEC) and DNS-based Authentication of Named Entities (DANE) for integrating PKI support in DNS. In Section 4 we introduce the reader to Chameleon Signatures which we use in our proposal detailed in Section 5. The conclusions are presented in Section 7.

2. PKI and Certificate Pinning

Public Key Infrastructure has not a single trusted root which allows the verification of every existing certificate. Let alone the self-signed certificates, there is as set of authorities which satisfy legal and societal requirements for being considered as trusted roots. Those authorities' root certificates are incorpored to client software or operating systems in their trusted authorities repository as Root CAs. There has been an accelerated growth [2] of the CAs number, and Certificate pinning has been introduced as a tool to help users with the lack of trust in the growing number of CAs, and the threats of fake certs issued by attackers.

A client when establishing a TLS connection receives the server certificate and the server returns a Proof of Possesion of the corresponding private key, by signing the pre-master key that will be used to secure the session. The client builds the certification chain, that allows verifying the path from the server certificate to the corresponding Root CA certificate. Often in the path appear intermmediate CAs and intermmediate certificates for accelerating and greater scalability of TLS connections. In some cases some solutions may violate privacy [3]. In other cases compromised or improperly managed CAs may issue certificates to DNS domains without agreement with the owners of such domains. Other cases happened in 2011, like the compromise of the Malaysian Agricultural Research and Development Institute and their CA used to generate a fake Adobe Acrobat updater. Also in 2011, the Diginotar CA was used to issue certificates for gmail and Facebook. Comodo in 2011, TurkTrust in 2013, or more recently Trustwave also known cases of security breaches.

Certificate Pinning techniques provide clients with an alternative way of increasing their trust in a server's certificate. Some techniques define global control infrastructures together with the client' cross verification. That is the case of Certificate Transparency (CT) [4] or Sovereign Keys (SK) [5]. Others pursue to limit the issuance of certificates to domains to the authority of domain owners, like DNS Certification Authority Authorization [6]. Trust Assertion for Certificate Keys (TACK) [7] proposes a cross verification under the control of domain owners. HTTP Strict Transport Security and HTTP Public Key Pinning Protocol (HPKP) [8] define HTTP headers that set strict TLS policies and allow domain owners to alert clients of security compromised certificate chains. DANE together with DNSSEC [6] offers a complementary verification of PKI certificates together with DNSSEC authentication providing different use cases.

Domain name support was introduced in PKIX [9] in the SubjectAltNames extension [10]. CAs do not need to check with the domain owner to issue a certificate for a web site in the domain. Precisely that is the main reason for certificate pinning techniques. Nevertheless, we argue that actual certificate pinning techniques do not offer the dynamicity required by IoT microservices.

CT, SK and TACK require the collaboration among devices to improve detecting malicious certificates. They were designed for browsers or high-end devices since timestamps and other information have to be stored and require the device permanent connection for receiving and processing other devices similar information.

IoT applications requiring services instantiation in the Fog may receive identifiers as IP addresses and domain names. Being domain names preferred since they are easier to reuse, reinstantiate, move, replicate and load balance across the nodes of the Fog. This is a consequence of the DNS Fog controlling the IP assignment to a domain name in the DNS Fog zone, and thus easier control of balancing or service migration. DNS provides the required dynamicity of these cases.

3. DNSSEC and DANE

DNSSEC [11,12] introduces a linkage of PKI credentials to domain names. DNSSEC offers DNS resolvers source origin authentication of DNS data, authentication of the non-existence of a domain name and the integrity of DNS data. RRs in DNSSEC are signed and the signature included in an additional registry called the RRSIG. The zone signing key (ZSK) used to sign RR is on its turn signed with the Key Signing Key (KSK) which is available at the parent zone. DNSSEC defines the delegation signer (DS) resource records which contains the KSK of the child zone to allow to delegate the security of delegated zones, as required to maintain DNS scalability, and also to build a certificate chain alternative to PKI up to the DNS Root (the empty domain) which must be signed by some recognized authority, i.e., some of the Root CAs (see Figure 1).

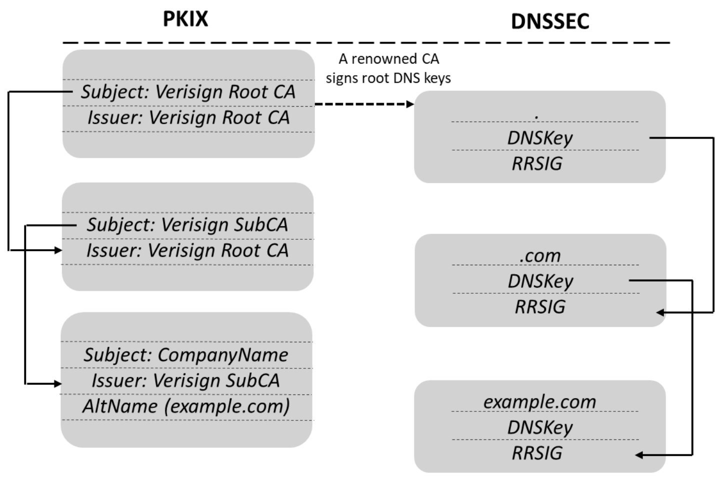


Figure 1. Comparison of verification chains in DNSSEC and PKIX. Image from [13].

DANE [6] supports clients who need to TLS connect to a server end entity (EE) in the domain zone. DANE allows to associate DNS Resource Registers (RR) of type TLSA to domain names indicating either a credential for the server: the credential that the server must use in a TLS connection (PKIX-EE); a domain name trust anchors (TA) which links the CA authorized to issue certificates for the domain (PKIX-TA); a local CA entitled to issue trusted certificates for the domain (DANE-TA); and even to indicate an ad-hoc key for TLS (DANE- EE). DANE-TA and DANE-EE establish trust without PKI intervention, but from the verification of DNSSEC pointing to TLSA records.

A DNSSEC can modify its zone to support security to the dynamic microservices deployment of IoT scenarios. This introduces two main problems.

First, every change, even if it is a single TLSA record, requires the update of zone signatures. This is required for other microservices and IoT devices to rebuild the certificate chain and confirm the trust in a changing microservice identification. Computing the new signatures with every minor change is a high load for the DNSSEC server, even worse the signatures have to be computed while the TLSA record is changed, to ensure the proper verification of the whole scenario.

Second, DNS servers are a critical resource of a domain. Attending resources frequent updates may compromise DNS service availability, especially if updates require expensive processing like cryptographic computations. If TLSA records are to be updated frequently, this may kill the performance of the DNS server, and we do not expect Fog dynamic scenario management to compromise the availability of the core DNS service.

Even more, using the core DNS like this makes it impossible to offer a distributed and best-effort management. The Fog should do a best effort management of the microservices offloaded from the core central system management. We propose to introduce a special type of certificates which make use of chameleon signatures that are explained in Section 4 and allow the required dynamicity without requiring so frequent updates of the DNSSEC zone.

4. Chameleon Signatures

Chameleon Signatures [14] as other more popular signatures (ECDSA, etc.) provide the undeniable commitment of the signer with respect to a signed document that is sent to a receiver. The main difference is that chameleon signatures cannot be transferred by the receiver to a third party without the signer involvement. The signer is thus required to participate to check the validity of the signature.

Chameleon signatures are proved secure under the standard model. Chameleon signatures use a special hash algorithm, the chameleon hash, which has the property of being collision resistant for the signer, but the receiver can compute collisions at will. Chameleon signatures introduce a trapdoor which can be used to find collisions in such a way that the signature of signer S allows the receiver R to generate additional S signatures at will. Sending a chameleon signatures requires R to be trusted by S. Nevertheless, S is the only entity who can prove that a fake signature is indeed a fake. But S cannot show that a legit signature is fake.

Chameleon signatures are not transferable in general, i.e., they can only be checked by a single receiver. Nevertheless, the receiver can transform them into a trasferrable version [15] by publishing partially or totally the original message, since the collisions trapdoor is generated using R's public key.

Let us illustrate how the chameleon signatures work using discrete logarithms as taken from [16].

Let R be a receiver, let q be a large prime and $p = kq + 1$, let g be a generator of $\mathbb{Z}_p^*$. Choose R private key $SK_R = x \in (1, q-1)$, and the corresponding public key $PK_R = y = g^x \text{mod } q$. Let ($PK_S$,$SK_S$) be the pair of public and private keys of the signer S.

The chameleon hash of a message M takes two random numbers r and s, and it is computed over the hash of a message and r, for instance $e = \text{SHA}_{256}(M, r)$. The chameleon hash is defined in Equation (1).

$$\text{CHAM-HASH}_R(M, r, s) = \{r - (y^e g^s \text{mod } p) \text{mod } q\} \tag{1}$$

Given the message M and the chameleon hash $Hash$, R asks S to sign the chameleon hash using SK_S. The signature $sig = SIG_S(Hash)$ together with (m, r, s) can be verified using $PK_R = y$ and

PK_S, the public keys of R and S. First we compute $e = \text{SHA}_{256}(M, r)$, then $H2 = r - (y^e g^s)\text{mod } q$. Finally we check $VRFY_{PK_S}(sig) = H2$, if this holds the signature is valid.

When R wants to find a collision over a new message M', R finds a new random $k \in (1, q-1)$. Next R computes the collision for the chameleon hash as

$$r' = \text{CHAM-HASH}_R(M, r, s) + (g^k \text{mod } p)\text{mod } q. \tag{2}$$

Now R computes the new hash of the message as $e' = \text{SHA}_{256}(M', r')$, and $s' = k - e'x\text{mod } q$. The new message can be sent together with the original signature for verification (sig, M', r', s'), since r' and s' are being computed to preserve the chameleon hash.

We will follow this approach and publish the message and random numbers together with the signature to allow other parties to verify the signature.

5. DNS Secure and Dynamic Supporting IoT Microservices

The present article proposes an architecture with a DNS intercepting requests coming from the interior of the Fog to allow the required dynamicity and alleviating the core DNS (at the core of the network) which availability is critical for the dynamic management of resources.

To achieve this, we propose a soft delegation of zones to be managed by others DNS servers, but without the effective delegation of the zone. Those DNS servers will play the role of DNS forwarders, intercepting DNS requests to be resolved within the Fog. The logical consequence is that the management will be decentralized from the core of the network, and concentrating a considerable amount of traffic in the Fog.

By soft delegation we refer to a schema where the delegatee (the one to whom the zone is soft delegated) can dynamically incorporate new DNS records to the zone on behalf of the core DNS, at request of the microservice dynamic deployment, reinstantiation, etc. The incorporation of new records must allow the verification of credentials, that requires that records must be signed and the corresponding signature incorporated in RRSIG records. Moreover, the whole zone has to be signed when a single new record is incorporated. We use TLSA DANE records protected with DNSSEC by the Fog DNS, and that requires the Fog DNS being able to generate signatures as the core DNS, but without having access to the keys of the core DNS, neither KSK nor ZSK.

The DNS Fog faces thus a considerable number of threats and that requires replacement and revocation strategies that can minimize the risks when the possibility of the DNS Fog being compromised increases.

The Fog DNS forwarder is a service that can be instantiated in a opportunistic way in a computation node for microservices, in the virtualization infrastructure or in the Fog as illustrated in Figure 2. In that figure, services that require visibility from the Internet (out of the Fog) have to be use the gateway. This is something that has to be specified when they are declared for instantiation.

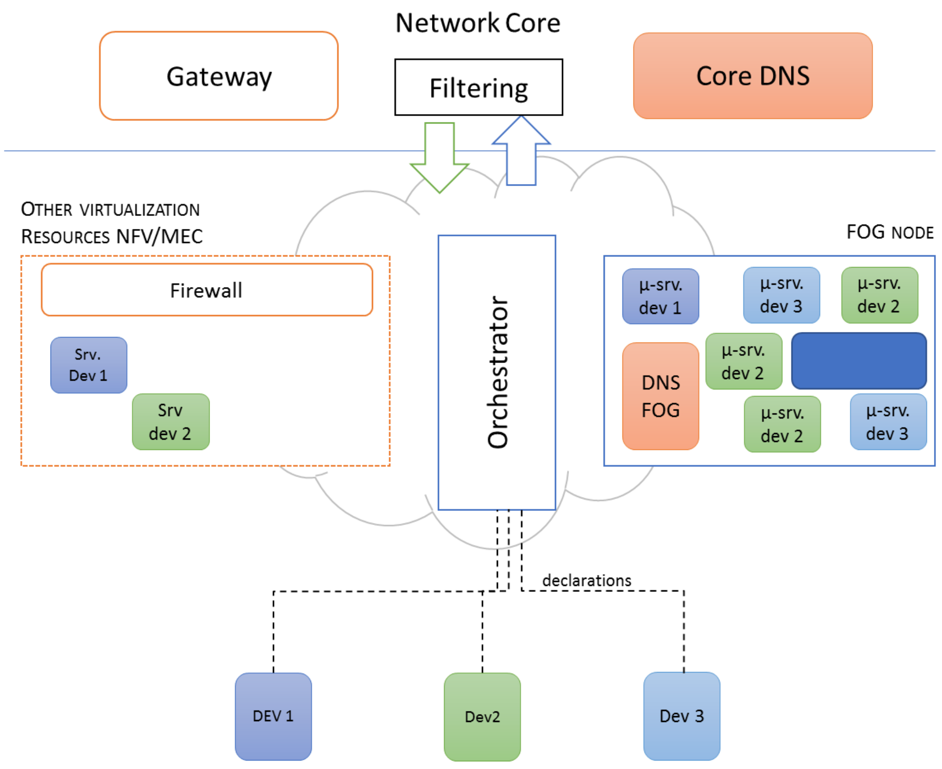


Figure 2. DNSFog as a microservice in a node filtered and firewalled from the core of the network.

5.1. Soft Delegation Proposal

In DNSSEC responses conveying one RR include the RRSIG with the signature of the RR. The resolver must use the DNSKEY RR to authenticate the RRsets. The proposed soft delegation proposes to use a pattern RRSIG pre-signed with the DNSKEY of the core DNS and delegated to the Fog DNS. The signature will involve a chameleon hash so that the Fog DNS can compute collisions for the dynamic modification of records.

The core DNS will be the signer S as explained in Section 4 and the Fog DNS. The core DNS generates a pair of public and private keys called PK_{core} and SK_{core}. It will include a DNSKEY record with the hash of PK_S that will be identified as the zone signing key in ZSK, that will also include a key tag that we will call KT_R in reference to participant (receiver or delegatee) R. The KT_R tag has to be signed by the KSK of the core DNS, to extend the trust in the core DNS key to this new key.

The core DNS instantiates a DNS Fog that will play the receiver role R as explained in 4 with a fresh pair of public and private keys (PK_{FOG}, SK_{FOG}).

The core DNS generates a pattern record that limits the delegation of the Fog DNS. The pattern will reflect the subdomains that can be altered by the Fog DNS. Let `xyz.com` be the domain of the core DNS, in the Tables 1 and 2 we show some examples of the pattern in textual format. The patterns allow the soft delegation and limit the capability of the Fog DNS to alter records not delegated by the core DNS.

Table 1. Example of pattern record with placeholders.

Domain Name	TTL	Class	Type	Value
`<T>[.L].xyz.com.`	MAX_TTL	IN	`<R>`	rnd

The core DNS sets and signs the pattern record so that the signature can be transferred to a resolver. The resolver can thus verify the core DNS signature and determine that the new data of the records filled by the Fog DNS matches the pattern set by the core DNS.

The record pattern format is a proposal to improve readability, but it may be replaced by other mechanisms like a bloom filter [17], including the disallowed registers or a hash and URL to a security policy.

Table 1 shows the example fields of the pattern record limiting the Fog DNS. `<T>` represents a positional marker of a final label and `[.L]` allows optional delegation of whole subdomains without a zone delegation. `MAX_TTL` sets the maximum TTL to records associated to the pattern. `<R>` is a positional marker to indicate the type of record this pattern allows to verify. Finally rnd is a random number selected by the core DNS. Table 2 shows particular examples of the pattern record to illustrate its usage.

Table 2. Example of pattern record with placeholders and usage

Domain Name	TTL	Class	Type	Usage
`<T>[.L].xyz.com.`	MAX_TTL	IN	`<R>`	Pattern scheme
`<T>.xyz.com.`	3600	IN	A	Allows to verify RRs of type Address (A) for domain xyz.com (like example.xyz.com). The TTL is limited to 3600
`<T>.fog1.xyz.com.`	1800	IN	A	Allows to verify RRs of type Address (A) for domain fog1.xyz.com (vending_z34_backup_service.fog1.xyz.com) it would not match A RR of the domain xyz.com
`<T>.xyz.com.`	720	IN	TLSA	Allows to verify TLSA records for the domain xyz.com

The core DNS (S) generates the signature for the pattern record:

1. serializes the pattern record: `RR(i) = owner | type | class | TTL | RDATA length | RDATA`, where RDATA =<T>.xyz.com | rnd.

   Resulting in msg = owner | type | class | TTL | RDATA length | <T>.xyz.com | rnd = $RR(n)$
2. S chooses random r and s.
3. computes the pattern record hash $\text{hash}_{\text{SHA256}}(msg) = m$, and $e = \text{hash}_{\text{SHA256}}(m, r)$
4. S computes the chameleon hash with the public key of the Fog DNS (R): $\text{CHAM-HASH}_R(m, r, s) = \text{chash}(m, r, s) = r - (y^e g^s \text{mod } p) \text{mod } q = \text{RR}_{\text{PATTERN}}$
5. finally the RRSIG is computed as mandated by DNSSEC `signature = sign(RRSIG_DATA | RR(1) | RR(2)... )` including r and s. The signature (in the example RSA-SHA256), computes the hash $\text{shash} = \text{hash}_{\text{SHA256}}(\text{RRSIG\_DATA} \mid r \mid s \mid \text{RR}_{\text{PATTERN}})$.

   Then it is cyphered with SK_S to obtain the signature $\text{sig} = \text{sig}(\text{RRSIG\_DATA}|r|s|\text{RR}_{\text{PATTERN}})$
6. The signature, message and random numbers are sent to Fog DNS for including in the domain zone data: $(\text{sig}, r, s, \text{msg})$.
7. That record is formatted according to DNSSEC. The algorithm id is 254 (PRIVATEOID), which allows a different verification mechanism. An example is given in Table 3.

The Fog DNS configured by the core DNScan now intercept Fog incoming requests and resolve them with authenticated records.

Table 3. Example of signed pattern record.

Domain Name	TTL	Class	Type	Value
<T>.xyz.com	3600	IN	A	32_bit_random_number (rnd)
<T>.xyz.com	3600	IN	RRSIG	**A 254** 3 3600 20170509183619 (20170409183619 32478 xyz.com. b64enc($\text{sig}(\text{RRSIG\_DATA} \vert r \vert s \vert \text{RR}_{\text{PATTERN}}), r, s$))

Verification by a resolver requires the pattern record, and also (PK_{FOG}, g, p, q, r, s) that will be included as DS type records using private OIDs, not requiring any additional protocol.

PK_{FOG} does not require to be certified by a trusted CA, using DANE-TA or DANE-EE (as explained in Section 3. Often the public key of the Fog DNS will be ephemeral in highly dynamic environments where the Fog DNS requires frequent reinstantiation operations.

Once the resolver has verified the ZSK PK_{core} of the core DNS, it only has to follow the [12] to obtain the pattern record, r, s, g, p, q, and PK_{Fog} from the Fog DNS or using ZeroConf. The resolver is able to verify the signature of records from Fog DNS.

Let us present an example. Let the Fog DNS receive a request to add an A type RR with value 163.117.141.197 associated to domain name srv.xyz.com. The Fog DNS will create the corresponding DNS record as follows:

1. find a new random $k \in (1, q-1)$
2. compute the collision for the chameleon hash ($\text{CHAM-HASH}_R(m, r, s) = \mathrm{RR}_{\text{PATTERN}}$ as in Equation (1): $r' = \mathrm{RR}_{\text{PATTERN}} + (g^k \mathrm{mod}\ p) \mathrm{mod}\ q$.
3. serialize the record: RDATA =`163.117.141.197`, and msg′ = owner | type | class | TTL | RDATA length | RDATA. That will be the m' of Section 4.
4. compute $e' = \mathrm{SHA}_{256}(\mathrm{msg}', r')$, and $s' = k - e' x \mathrm{mod}\ q$.
5. this ensures that $\mathrm{shash} = \mathrm{hash}_{\text{SHA256}}(\text{RRSIG\_DATA} \,|\, r' \,|\, s' \,|\, \mathrm{RR}_{\text{SRV.XYZ.COM}}) = \mathrm{hash}_{\text{SHA256}}(\text{RRSIG\_DAT} \,|\, r \,|\, s \,|\, \mathrm{RR}_{\text{PATTERN}})$.

6. The new A record can be added to the zone together with r' and s'.
7. That record will be formatted according to DNSSEC using a private algorithm as previously mentioned. The following table illustrates an example of the DNSSEC records. Note the signature value is the same (the signature is not altered) but the new values of r', s' are included to make it transferable:

Domain name	TTL	Class	Type	Value
srv.xyz.com	3600	IN	A	`163.117.141.197`
srv.xyz.com	3600	IN	RRSIG	**A 254** 3 3600 20170509183619 (20170409183619 32478 xyz.com. b64enc(sig(RRSIG_DATA$\|r\|s\|$RR$_{\text{SRV.XYZ.COM}}$), r', s'))

Let Q be a resolver sending a query of type A for the domain name `srv.xyz.com`. The request is intercepted by the Fog DNS and initially it gets the information:

Q queries type A records of domain name xyz.com and within the answer it gets the pattern and the new record:

Domain name	TTL	Class	Type	Value
<T>.xyz.com	3600	IN	A	32_bit_random_number
<T>.xyz.com	3600	IN	RRSIG	**A 254** 3 3600 2017050918361 (2017040918361 32478 xyz.com. b64enc(sig, r, s))
srv.xyz.com	3600	IN	A	`163.117.141.197`
srv.xyz.com	3600	IN	RRSIG	**A 254** 3 3600 2017050918361 (2017040918361 32478 xyz.com. b64enc(sig, r', s'))

Let Q obtain from Fog DNS the parameters g, p, q, r, s and PK_{FOG}.

Q has verified that ZSK (PK_{core} of the core DNS from the previously obtained KSK and it knows its *key tag* is 32478.

The verification performed by Q consists in the following:

1. According to [12] Q knows ZSK, and verifies that:
   - RRSIG RR and other records have the same `owner`, `class` and `type`;
   - the zone name in the RRSIG matches the record to verify;
   - values of expiry, start and TTL fields are consistent;
   - verifies the signature algorithm to match with the DNSKEY and the key is a ZSK;
2. Q verifies the signature value of the pattern RRSIG record (retrieving r, s and computing msg, e, computing the chameleon hash, apply the signature hash to get *chash* and verifying the signature $VRFY_{PK_S}(sig) = chash$) and checks it is the same value as in the srv.xyz.com RRSIG record.
3. Q verifies the $\mathrm{RR}_{\text{PATTERN}}$ matches the $\mathrm{RR}_{\text{SRV.XYZ.COM}}$ record checking the domain name, maximum TTL and record type.
4. Q computes RDATA =`163.117.141.197`, and msg′ = owner | type | class | TTL | RDATA length | RDATA.
5. Q retrieves r', s' and sig from the RRSIG record.
6. Q computes $\mathrm{hash}_{\text{SHA256}}(msg', r') = e'$
7. Q computes $r' - (y^{e'} g^{s'}) \bmod p) \bmod q = \mathrm{RR}_{\text{PATTERN}}$, and $r - (y^e g^s) \bmod p) \bmod q = \mathrm{RR}_{\text{SRV.XYZ.COM}}$ and checks that $\mathrm{RR}_{\text{SRV.XYZ.COM}} = \mathrm{RR}_{\text{PATTERN}}$. If they match only the Fog DNS private key $SK_{Fog} = x$ can compute the collision.
   This verification proofs that the record has been generated by the Fog DNS authorized by the core DNS.

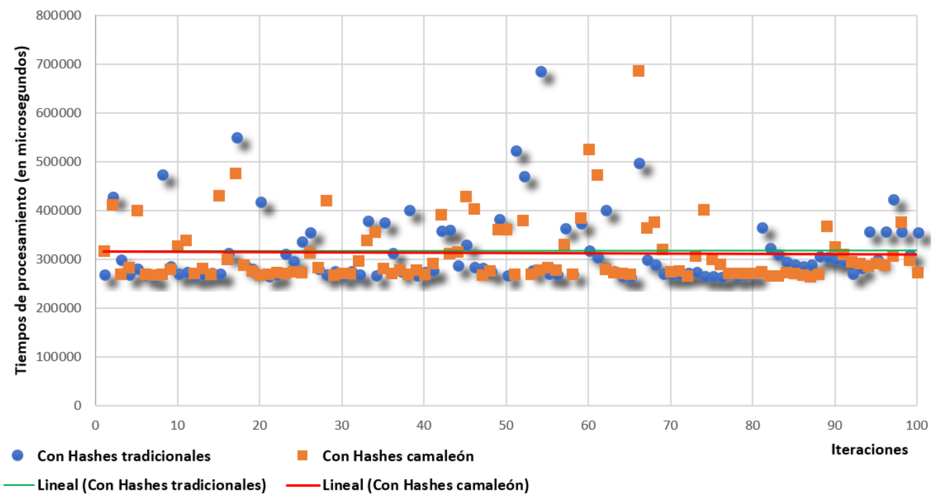
6. Results

We have performed some experiments to find the overhead of chameleon signatures over traditional signatures. The overhead is due to the computation of the chameleon hash and the computation of collisions on a new message. We have used a traditional signature scheme RSA 2048 bits with message digest SHA256. We have performed an experiment of computing the collision for a new message versus computing the SHA256 hash.

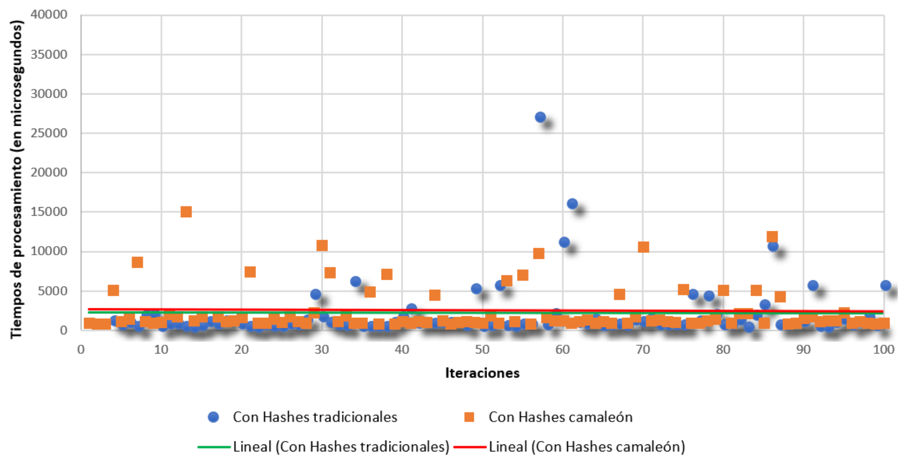
We have repeated 100 times the experiment, using a non-optimized program in OpenSSL. In Figure 3 we show the results of the computation of traditional signature vs the chameleon signature.

The y-axis is the time in ms, the x-axis is the sample number. The figure at the right shows the generation time of a new RSA signature versus generating the chameleon hash and finding a collision of the chameleon hash. The mean times are 315.13 ms versus 332.61 ms, a 5.5% increment only. The figure at the left shows the signature verification time versus the time of finding a collision and checking the message pattern. The mean times are 2.12 ms versus 2.29 ms, a 8% increment only. These values are likely to be improved, depending on the signature scheme, in particular in RSA:

Finding a collision in the chameleon hash	Signing a new message
1. $k \in (1, q-1)$ 2. $r' = \text{CHAM-HASH}_R(M, r, s) + (g^k \bmod p) \bmod q$. 3. $e = \text{SHA}_{256}(M', r')$ 4. $s' = k - ex \bmod q$.	1. $e = \text{SHA}_{256}(M')$ 2. $sig = SIG_{SK_S}(e) = e^d \bmod n$.



(**a**) Generation



(**b**) Verification

Figure 3. Processing time (in microseconds) traditional signature vs chameleon hashes (100 samples).

7. Conclusions

IoT presents several challenges: a notable increment of the background traffic in the core of the network due to signaling of subscribe/notify models, a lack of security support due to the limited nature of devices, the outdated support of PKI certificates in low-end devices, and the overall difficulty for updating the security support of IoT devices. The adequacy of microservice architectures for IoT will alleviate the background signaling if deployed in the Fog. Nevertheless, proposed certificate pinning schemas do not offer the dynamicity required by microservices.

In this paper we have proposed a DNS-based dynamic authentication for microservice architecture in IoT. We propose using a DNSSEC forwarder at the Fog, which can perform frequent updates to the DNS to cope with the demand of microservice creation, migration, and reinstantiation. We propose using a transferable, key exposure free scheme of chameleon signatures, with a low computation cost for the Fog forwarder and verifiers, which uses DANE to offer the TLSA records which can be used for establishing the TLS secured sessions for microservices to interact.

There are still some open problems, most notably TLS connections can have a longer life than established by DNS TTL. In case of instantiation of new copies of a highly demanded microservice for load balancing, we are able to fast disseminate the TLSAs records of the new instances. But existing connections will persist and make it more difficult balancing the load with the fresh copies of the microservice.

Funding: This research was partially funded by the Spanish Government under Project grants TEC2014-54335-C4-1-R and TEC2017-84197-C4-1-R.

References

1. Commission, F.T. *Internet of Things: Privacy & Security in a Connected World*; Federal Trade Commission: Washington, DC, USA, 2015.
2. Durumeric, Z.; Kasten, J.; Bailey, M.; Halderman, J.A. Analysis of the HTTPS Certificate Ecosystem. In Proceedings of the 2013 Conference on Internet Measurement Conference, Barcelona, Spain, 23–25 October 2013; pp. 291–304. doi:10.1145/2504730.2504755.
3. Pandya, G.K. Nokia's MITM on HTTPS Traffic from Their Phone. Available online: https://gaurangkp.wordpress.com/2013/01/09/nokia-https-mitm/ (accessed on 23 April 2018).
4. Laurie, B.; Langley, A.; Kasper, E. Certificate Transparency. *RFC 6962 (Experimental)* **2013**. doi:10.17487/RFC6962.
5. Laurie, B. Secure the Internet. *Nature* **2012**, *491*, 325–326.
6. Hoffman, P.; Schlyter, J. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*; RFC 6698 (Proposed Standard); Internet Engineering Task Force: Fremont, CA, USA, 2012; doi:10.17487/RFC6698.
7. M. Marlinspike, T.P. Trust Assertions for Certificate Keys (Draft-Perrin-tls-Tack-02.txt). Available online: http://tack.io/draft.html (accessed on 25 January 2018).
8. Hodges, J.; Jackson, C.; Barth, A. *HTTP Strict Transport Security (HSTS)*; RFC 6797 (Proposed Standard); Internet Engineering Task Force: Fremont, CA, USA, 2012; doi:10.17487/RFC6797.
9. Chokhani, S.; Ford, W.; Sabett, R.; Merrill, C.; Wu, S. *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*; RFC 3647 (Informational); Internet Engineering Task Force: Fremont, CA, USA, 2003; doi:10.17487/RFC3647.
10. Santesson, S. *Internet X.509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name*; RFC 4985 (Proposed Standard); Internet Engineering Task Force: Fremont, CA, USA, 2007; doi:10.17487/RFC4985.
11. Arends, R.; Austein, R.; Larson, M.; Massey, D.; Rose, S. *Resource Records for the DNS Security Extensions*; RFC 4034 (Proposed Standard); Internet Engineering Task Force: Fremont, CA, USA, 2005; doi:10.17487/RFC4034.
12. Arends, R.; Austein, R.; Larson, M.; Massey, D.; Rose, S. *Protocol Modifications for the DNS Security Extensions*; RFC 4035 (Proposed Standard); Internet Engineering Task Force: Fremont, CA, USA, 2005; doi:10.17487/RFC4035.
13. Barnes, R.L. Let the Names Speak for Themselves: Improving Domain Name Authentication with DNSSEC and DANE. *The Internet Protocol J.* **2015**, *15*, 201–213. ISSN 1944-1134.
14. Krawczyk, H.; Rabin, T. Chameleon Hashing and Signatures. *IACR Cryptol. ePrint Arch.* **1998**, *1998*, 10.
15. Boyar, J.; Chaum, D.; Damgård, I.; Pedersen, T. Convertible undeniable signatures. In *Conference on the Theory and Application of Cryptography*; Springer: Berlin, Germany, 1990; pp. 189–205.
16. Ateniese, G.; de Medeiros, B. On the Key Exposure Problem in Chameleon Hashes. In *Security in Communication Networks*; Blundo, C., Cimato, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 165–179.
17. Broder, A.; Mitzenmacher, M. Network applications of bloom filters: A survey. *Internet Math.* **2004**, *1*, 485–509.