

Giving IoT Services an Identity and Changeable Attributes

Marc-Oliver Pahl and Lorenzo Donini

Technical University of Munich

{pahl,lorenzo.donini}@tum.de

Abstract—The Internet of Things (IoT) pervades our surroundings. It softwarizes our physical environments. Software controls devices that interface their physical environments. The IoT is often privacy, safety, and security critical. Consequently, it requires adequate mechanisms for securing its services. For reasons such as heterogeneity, complexity, and lack of deployment there is little research on IoT service security.

Our work creates a base for IoT service security. We give IoT services secure identities and attributes. Using site-local X.509v3 certificates with short lifetimes, we show how service attributes can securely be changed at runtime. This enables enforcing security policies even on distributed, loosely coupled IoT nodes. Our central mechanisms are pinning certificates to service executables, and autonomously managing the short certificate lifetimes. We assess the resulting renewal traffic and power consumption.

Index Terms—IoT, security, certificates, x.509, microservices, autonomous service management, unattended nodes, metadata

I. INTRODUCTION

Internet of Things (IoT) devices pervade our environments. In contrast to earlier distributed systems, IoT devices not only offer compute power but typically also access to their physical environment via sensors and actuators.

Attacking IoT devices is attractive. First, the IoT is a massively distributed resource. The massive deployment of similar devices makes device-type-specific attacks attractive. The partially standardized interfaces, e.g. REST, facilitate attacks. When taking many IoT devices together, they become the perfect tool for attacks such as Distributed Denial of Service (DDoS) as the Mirai botnet showed [1]. With the expected increase in compute power [2], preventing unwanted access to IoT devices becomes even more relevant.

A second reason for attacks lays in the Cyber-Physical dimension: IoT devices often continuously collect privacy-critical data about their environments, e.g. “when am I home and how long do I stay in the bathroom”. In addition, via their actuators they enable making their environment uncomfortable, e.g. very cold, or even hostile, e.g. robot arms in a factory that hunt workers in their reach. The Stuxnet attack pointed in that direction already [3], [4].

Unwanted access to privacy-critical data, and unintended control of devices become even more interesting when the formerly isolated IoT entities get connected. There are promising proposals for such joint orchestration, often coming from the Pervasive Computing domain [5], [6]. More recently, different groups proposed realizing IoT scenarios by composing so-

called microservices that act as building block, providing small functionality such as unified access to an IoT device [7]–[10].

A central insight when analyzing the IoT is that it consists of communicating software services. Each IoT device runs a service that offers a remote-control interface. So-called controllers run other services that implement joint orchestration of the devices, e.g. for realizing a climate control. The (micro-) service-oriented approaches require even more services for implementing IoT scenarios. Consequently, *securing the IoT means securing its services*.

Securing the IoT is essential [11], [12]. However, there is almost no IoT service-security research yet. This is not surprising due to the heterogeneity and distribution of the IoT, and due to the missing of a common middleware. However, security cannot be added to systems later. It has to be an integral part of IoT middleware [13]. Therefore, we consider it the right time for investigating, **how to secure IoT services and their metadata?**

Fundamental requirements for securing IoT services are *secure identities* for authentication, and *secure metadata* as base for different mechanisms such as authorization. We show how certificates help giving services an identity and secure attributes. To reflect the dynamic, distributed nature of the IoT, we show how the securely added attributes can be changed at runtime, enforcing security policies even on distributed loosely coupled IoT nodes.

Our solution is based on pinning X.509v3 certificates to service executables, and autonomously managing *short certificate lifetimes* [14]. We extend the classic Public-Key Infrastructure (PKI) with decentralized Certification Authorities (CA) that make our solution scale. We autonomously manage the certificates to make the complexity manageable. We assess the renewal traffic and power consumption resulting from our short life-time certificates.

Our design protects services against diverse attacks by securing the runtimes, metadata, and –with the help of the used middleware– inter-service communication throughout the entire service lifecycle; protecting against: node impersonation, eavesdropping, man-in-the-middle attacks, sybil attack, unauthorized communication, unauthorized code changes including code injection, and Denial of Service (DoS).

The paper starts with an overview on the most relevant state of the art (section II). Section III introduces the IoT setting we use for illustrating our solution. We introduce our security methodology in section IV. Section V evaluates the resulting network traffic, CPU load, and energy consumption.

II. RELATED WORK

In [15] we presented a black box approach for modeling service behavior and using the resulting models for firewalling bidirectional service access. This is relevant for securing services we do not have any control of, e.g. vendor-proprietary software stacks running on IoT devices.

This paper is complementary by providing security for services that are built to interact with an IoT middleware [7]. Our setting (section III) therefore resembles *classic software distribution frameworks*. Common *software distribution frameworks* for PCs or smartphones are Ubuntu Core [16] for deploying services to computers, and Google Android Play Store [17] and Apple App Store [18], [19] for smartphones. All three examples deploy software to single host systems with user interfaces. In contrast, we target distributed systems of unattended nodes. Consequently, in addition to their challenges we have to enable trust between the distributed loosely connected compute nodes forming our IoT systems.

Table I compares relevant features of the existing service management solutions with our proposed solution. All use asymmetric cryptography for signing data, providing a secure identity and integrity.




	 Core			proposed
App signature	Signed by the store (GPG)	Self-signed APK	Apple-issued certificate	Signed by dev, store and site
Integrity check	Before installation	Before installation	At launch time	Periodically
Service distr.	Store -> Device	Store -> Device	Store -> Device	Store -> SLSM -> NLSM(s)
Cert. renewal	No	No	No	Yes
Target platform	Any device running Ubuntu Core	Android smartphone	iOS smartphone	Any device capable of running Java

Table I
Feature comparison to widely deployed service distribution solutions.

Different to PCs or smartphones, Things do not form a trusted system. Also, IoT services typically run for a long time. Integrity checks of the executable at installation time are not sufficient. Our periodic checks provide better security.

Like us, some solutions use *certificates to secure services in distributed systems*. Some use local Certificate Authorities (CAs) like us but for different purposes. Others use Certificate Revocation Lists (CRLs) for change-propagation, making them less disruption tolerant than us. The following solutions address authentication between distributed IoT nodes but not service integrity and metadata protection. Like us, Panwar and Kumar [20] introduce *local CAs* to eliminate pre-shared keys on IoT devices. They target low-power IoT devices communicating using CoAP and DTLS. They show that certificates can well be used in constraint environments.

Similarly, Kim et al. [21] present a local authorization mechanism, leveraging a dedicated “Auth node” (similar to a CA) to distribute session keys to other nodes. They target protecting communication channels between heterogeneous IoT devices. The scalability problem in this case is solved by allowing multiple distributed synchronized CAs.

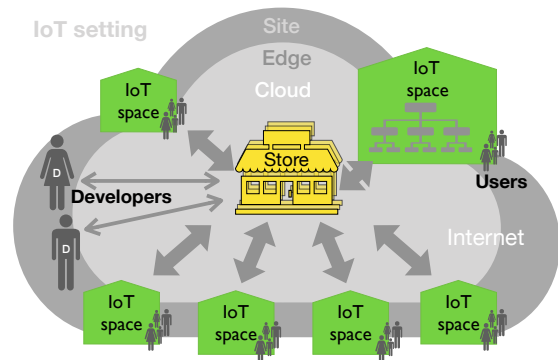


Fig. 1. Distributed developers and locally-distributed compute nodes.

Finally, some works propose *short lifetime certificates for other purposes* than securing services. Here, concerns exist towards real world deployments including scalability and performance. Micali [22] computes the effort of renewing all Internet certificates frequently. His conclusion is that the traffic and computational effort are both significant. We do not face this problem as we issue our short-lifetime certificates only locally, resulting in low latency and better load distribution. Micali proposes optimized Certificate Revocation Lists (CRLs). So do Naor and Nissim [23]. Contrary to us, such centralized approaches are not tolerant to network disruptions.

Rivest proposed short renewal cycles already in 1998 [14] to make CRLs obsolete. Back then main problems were that there was no suitable way to renew certificates frequently, and that such an architecture does not scale throughout the Internet.

Topalovic et al. [24] focus on browser-based certificate revocation. They reaffirm the concept of short-lived certificates to mitigate damages caused by certificate compromise as well as the complexity introduced by certificate revocation.

To overcome CRLs and solve the problem of scalability, we introduce *autonomous* certificate renewal and use it to renew certificates only *locally* within an IoT site. Our first ideas were well received as a poster at NOMS 2018 [25].

III. SETTING

Our security architecture fits for services running on distributed networked nodes under a common management authority. A service’s life-cycle is (a) development → (b) distribution → (c) configuration → (d) deployment → (u) update. Similar to the mobile computing App economy, we assume distributed developers and central stores for service distribution [26]. See figs. 1 and 2.

Figure 1 gives an overview on the described setting. On the left, distributed developers upload their creations to a central store. It distributes the services to the distributed IoT spaces. In the space on the top right, the distributed IoT devices running the services are shown. The assumed compute devices are regular IoT things, e.g. smart light switches, that will soon have enough compute power to run services in addition to their own functionality [2]. In the figure, the IoT sites are locally managed. However, our architecture does not make assumptions on the location of the entities.

We developed the presented solution as part of our Distributed Smart Space Orchestration System (DS2OS) [27]. For coupling IoT services, DS2OS uses a data-centric peer-to-peer overlay, the Virtual State Layer (VSL) [7], [9], [10]. The VSL implements data-centric coupling. It manages data for its connected services and VSL services only communicate over this structured data [28].

The VSL implements a service-centric data-oriented IoT management [7]. The realization of complex IoT scenarios happens via mashing-up several microservices. An example is connecting a *Gateway* service to a thermometer service with an *Orchestration* logic services, and another *Gateway* service towards a valve in order to implement a room climate controller. A successful deployment of a (micro-) service-oriented architecture requires autonomous management to cope with the complexity and the often-missing expertise.

The VSL enables publish-subscribe and push/ pull communication [7], [9]. It also provides stream communication, and data-centric remote function calls [29]. Key features are semantically rich service discovery [28], [30], and the dynamic binding of services at run time [7]. The VSL uses Transport Layer Security (TLS) [31]. Therefore, we use the introduced X.509 certificates [32] for securing these channels. In addition, the VSL implements role-based access control [33].

Figure 2 gives an overview on the described service coupling within an IoT site. The peer-to-peer overlay is spanned by so-called Knowledge Agents (KA). Each service connects to its nearest KA to get access to the overlay. VSL services are labeled *srv*, *NLSM*, *SLCA*, and *SLSM* in the figure.

While the VSL can encrypt service communication, it currently lacks an automated service authentication and metadata security mechanism. For its operation it requires the availability of different service metadata such as interface descriptions, or access role identifiers. Service executable and metadata are packed to so-called service packages. The IoT service authentication and metadata protection mechanisms introduced in this paper fill this gap.

Within a site we assume a hierarchical *service management* [27]. Each node runs a Node-Local Service Manager (NLSM) that manages all node-local services. Each site has a Site-Local Service Manager (SLSM) that interfaces to the Store. It makes site local optimizations regarding service placement, and it offers the management user interface.

As shown in fig. 2, we assume distributed IoT compute nodes. Each KA runs on a different node. The IoT has heterogeneous and possibly unreliable network links. Therefore, a requirement to our security solution is that it runs fully distributed. Security properties must be verified node-locally.

IV. SERVICE SECURITY ARCHITECTURE

Through a service’s life (section III) our solution enables

- securing service integrity and metadata integrity,
- authenticating developers,
- authenticating the store and validating its operation,
- providing services with a cryptographic identifier and proving their belonging to an IoT site,

- dynamic changes to the secured metadata, and
- fully distributed authorization that tolerates communication delays and network connectivity disruptions.

Figure 2 shows the lifecycle of a service from the development on the right over the distribution via a global store to the deployment and update on the left (section III). On the bottom right in the legend a service package is shown. It gets distributed into IoT sites and deployed to one or multiple site-local computing nodes. In the following, the identifiers in parentheses “(a)” refer to fig. 2.

A. Introducing Certificates

The bottom right shows the container for DS2OS services, the service package (section III). It consists of (1) the service executable and its (2) metadata. For securing it we introduce multiple certificates (3). One at each authority in the service life-cycle: at the developer, the store, and the local site.

Our security bases on standard X.509 v3 certificates [32] with two custom fields:

- 1) a cryptographic hash over the executable, and
- 2) a cryptographic hash over the metadata.

Both hashes protect the integrity of the executable and the metadata. Integrating the hash of the executable *pins* the certificate and the metadata to the specific executable, effectively resulting in the intended secure service identity. It enables *authentication* and *integrity* validation.

Information that has to be exchanged frequently between communication partners, or that changes during the life-cycle of a service should be included directly in the certificate via extensions for the reasons given next. In our pilot we add a third x.509 extension field that carries the access-role identifiers. Directly having additional data in the certificate brings the advantage that this data is always transmitted with the certificate, e.g. at the TLS handshake. For the VSL access identifiers this helps as they are needed for all data accesses.

The main reason for having the access IDs in the certificate and not in the metadata is that these IDs change within the local site. We want to verify the operations happening between the developers and the compute node, namely the store, and the SLSM. Only *not* changing the metadata keeps its hash unchanged and verifiable through all certificates. This implements the desired property to *validate the well-behavior of the participating intermediate processing entities* since changes become obvious via changed or invalid hashes.

Using an established web standard directly enables *node authentication* and *transport encryption* via TLS [31] (e,f). The resulting *confidentiality* protects from eavesdropping and man-in-the-middle attacks. It also ensures *data integrity*.

B. Three Trust Anchors

We do not use a classic Public-Key Infrastructure (PKI) but install different trust anchors. An entity’s public key enables verifying the certificate data [34]. Each of our three certificates is signed with a different private key. The trust anchors reflect the different trust levels of the entity relationships.

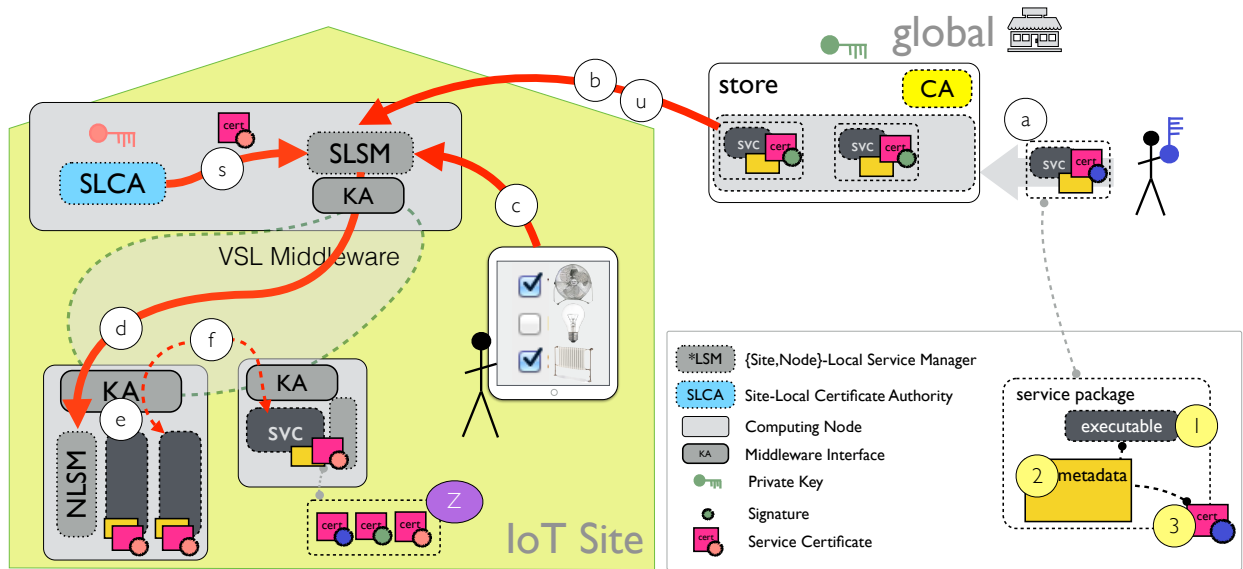


Fig. 2. Proposed Distributed Self-Managing Security Architecture.

We want to protect the entire service lifecycle. See figs. 1 and 2 (a). The *first trust anchor is the service origin: the developers*. Each developer has its own private key and registers the public counterpart at the store. The developer creates the service executable, the metadata file including his own public key and that of the store, and the certificate (a).

The next lifecycle stage is the distribution (b). After validating it with a developer’s public key, the store creates a copy of the certificate and signs it with its private key. Copying the certificate is necessary as X.509 version 3 certificates are signed exactly once [32]. The signatures validate the integrity of the certificate data, and prove its origin (authentication). Now the service package contains the executable, the metadata file, and the two certificates from the developer and the store.

The store Certificate Authority (CA) becomes our global trust anchor. Only the store’s public key has to be shared to local sites, making our solution practically usable and scale very well. The store’s public key is well-known to connected developers and IoT sites via side-channel exchange, e.g. local configuration. Knowing each-others public keys enables mutual authentication and secure data exchange. This is similar to existing software distribution frameworks (section II).

Via the SLSM, a user can install a service to the IoT site (b) (section III). Site-locally the SLSM verifies the executable, and the metadata via the cryptographic hashes in the store certificate. The distributed IoT sites only need the store’s public key as trust anchor.

However, we want the sites to securely identify developers as well. Therefore, the *developer ID* is part of the metadata file that is protected via the hash in all certificates. Though only having the store public key, the identity of the developer can therefore be validated. Analogue a unique *store ID* is saved in the metadata already at the developers. It enables tracking the store later. In case of problems, the store and the developer cannot deny their origin to the site (non-repudiation).

Configuration and deployment (c,d) happen within an IoT

site at the SLSM. As presented in section III, we assume a fully distributed hierarchical service management. The distributed IoT nodes are per-se not a trusted domain. To create the necessary trust between compute nodes and services, we introduce a *third trust anchor*, a so-called *Site-Local Certification Authority (SLCA)*.

At configuration, the SLSM makes another *copy of the certificate*. Service configuration requires changing additional certificate data. In our case, the user can change a service’s access roles (c). The *SLCA signs* the third, the local certificate with a *site-local key* (s). See left of fig. 2.

The site-local SLCA certificates enable all locally running IoT services to verify each-other’s certificates using the site’s public key. Sharing the site’s *public key* locally enables *all distributed IoT nodes to authorize* all services. As a desired consequence, with this check only services that were explicitly installed through the SLSM are allowed to run.

On deployment (d), the Node-Local Service Manager (NLSM) verifies the executable and the metadata before executing a service. The SLCA’s site-local signature creates the *trust between the distributed IoT nodes*.

Each stage of the management architecture verifies the integrity and authentication of the entire service package (see fig. 2). Since the IoT compute nodes provide many attack vectors, and since IoT services can be expected to run for a long time we propose *checking the integrity* of the metadata and the service executable *periodically* to prevent malicious tampering, e.g. via side channel attacks that alter data.

For full *verification of all intermediate service package managers* (store, SLSM) the developer certificate, the store certificate, and the site-local certificate are stored in the service package (see Z in fig. 2). If the SLSM knows the public key of the developer, e.g. via a directory, it can verify that the store did not alter the hashes in the certificate. Analogue, the NLSMs can verify the operation of the SLSM and the store when knowing the store’s and the developer’s public keys.

A drawback of the added security is that it requires additional storage and produces more traffic when transferring service packages. This might be critical in certain IoT installations (see section II). It can be omitted at the price of not being able to verify the entire processing chain.

In case of our setting, where the VSL provides service discovery and inter-service communication, a service authenticates at the VSL middleware using its certificate. Consequently, only authorized services can communicate locally [7]. Via the used TLS this communication is always secured. All parts of the security framework automatically enforce the security policy: services have to be authenticated and authorized via their local signature before they are allowed to run and communicate resulting in *security-by-design* [13].

C. Enabling Change Propagation

All three, developer, store, and SLSM add newly signed certificate (copies). Each time data can be altered on purpose, e.g. to change access rights.

As said before, those metadata that are changed on the service packages' path from the developer to the computing node must be part of the certificate to ensure full verification of the metadata "end-to-end".

Changing metadata is especially attractive at the local site (c). In our pilot, developers propose a set of access rights that are stored as metadata. Site-users can then locally restrict those rights (c). In our implementation only a subset of access rights is stored in the site-local certificate.

Metadata cannot only be changed at installation or deployment time of a service but also while it is running. In case of a single compute node this is easy as the metadata can simply be changed for all service instances node-locally.

Our distributed case is more complex. Site-locally, we introduce *short lifetime certificates, enabling service metadata changes at runtime*. In section V we evaluate the effects of the certificate lifetime on traffic and power consumption.

In case a user changes the metadata of a running service at the SLSM interface (c), e.g. the access rights, the SLSM tries to replace the metadata and the certificate at all compute nodes running the service by contacting the node-local NLSMs. Even for nodes that do not have connection to the SLSM, *metadata changes are enforced latest one certificate lifetime after the change*. Assuming synchronized node clocks, all certificates will expire after at most one certificate lifetime, automatically withdrawing all access rights.

A problem identified by Rivest [14] is the effort for renewing certificates. Therefore, we introduce a *local automated certificate renewal process*. Before a certificate expires, the NLSM contacts the SLSM via a signing request. The SLCA gets the current metadata from the SLSM, creates a new certificate, signs it and sends it back to the NLSM that exchanges it in the service package. Via a hook the new certificate *reauthorizes* the VSL communication.

As everything happens site-locally, and as the number of IoT devices can be expected to be limited, scalability is less of an issue compared to running such renewals for the entire

Internet (see section II). However, depending on the bandwidth and reliability of the links, different renewal intervals can make sense. See section V.

D. Updates

Service updates are straightforward in our solution. A new version of a service is obtained from the store. Its integrity is verified on all stages *identically to a new service deployment*. The only difference is that the local metadata changes are directly reflected from the old version in the SLSM when possible. In case the new version has new metadata the user is asked before deploying the update to the compute nodes (c).

The service package is transferred to the IoT nodes running the service and their NLSM starts the replaced executable [27].

V. EVALUATION

We begin the evaluation, assessing the security of our solution. Then we look at the performance and scalability since both are major concerns when using short lifetime certificates (section II). Finally we assess the energy consumption to reflect that IoT nodes are often resource constraint.

A. Security Evaluation

Our solution protects service executables and their metadata using certificates with distributed CAs. Following, we briefly discuss how the security properties are met.

Secure bootstrapping of the VSL prevents *node impersonation* and *sybil* attacks. The process equips all services including KAs, SLSM, SLCA, and NLSMs with the site-local certificates from the start. It uses the presented methods to authenticate all VSL KA nodes, ensuring full authentication of all services and overlay nodes from the start. Later started services authenticate to the VSL KAs, resulting in authentication of all services all the time.

We use standard X.509 mechanisms not breaking the properties of this solution. The VSL secures all data exchange using TLS with the presented certificates, preventing *eavesdropping* and *man-in-the-middle* attacks. Together with the service authentication, it prevents *unauthorized communication* and thereby *Denial of Service (DoS)* attacks. The periodic checks of the executables prevent *unauthorized code changes* including *code injection*.

B. Performance and Scalability Evaluation Setting

The following evaluation uses our Java VSL-based implementation [7]. The SLSM, NLSM, and SHE communicate securely over the VSL REST interface using HTTPs.

The *physical testbed* consists of five computers with more resources than required. We do this instead of using resource constraint devices as we expect IoT devices to be more powerful in the future. For the *energy and cpu load measurements* we use Raspberry Pis. Our setting is 4 nodes with 20 idle services and NLSM each, and a node hosting SLSM and SLCA.

Our used key length is 2048bit. It is directly correlated to the certificate size and therefore the measured traffic. Each certificate contains 0-3 accessIDs, randomly determined per service when spawned resulting in 0-60B extra size.

C. Certificate Lifetime vs. Traffic

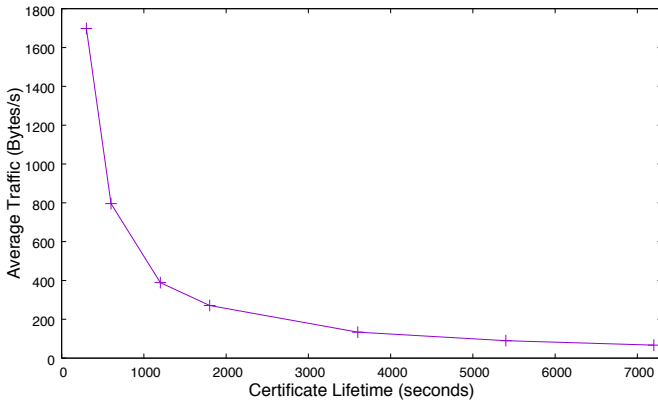


Fig. 3. Certificate renewal traffic depending on the certificate lifetime.

For identifying a suitable certificate lifetime we measured the average traffic in Bytes/s depending on the chosen certificate lifetime. As expected the traffic decreases the longer the certificate lifetime is. Fig. 3 shows the results with a random backoff enabled (see [25]). We identify 3600s=1h as good compromise between short lifetime and low traffic.

D. Computational Load and Energy Consumption per Node

CPU usage and energy are a key resource in the IoT. For the purpose of the test, very short certificate lifetimes were used (5 minutes maximum validity period). Figure 4 shows average values over 1s intervals from the NLSM and 20 DS2OS services running on a Raspberry Pi. We see spikes in the CPU load and the energy consumption where the certificates get replaced and verified. The measurement shows that the total CPU load during certificate renewals reaches about 30%, saturating one core on the Raspberry Pi 3.

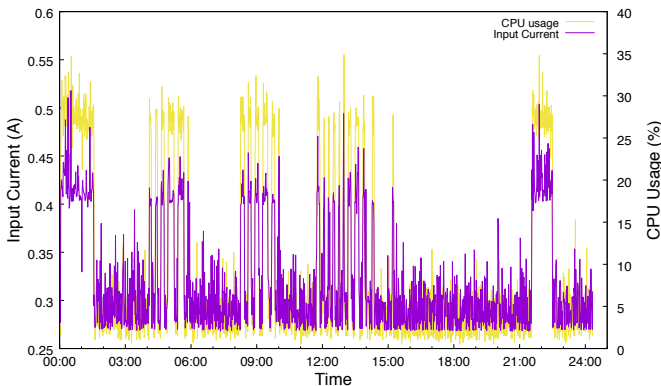


Fig. 4. 1s power and CPU averages of NLSM certificate renewals.

When idle, the IoT device draws 270mA. VSL pings and other low-level periodic mechanisms [7] lead to an increase in power consumption between 50-100mA. Energy consumption spikes occur when the NLSM generates a new Certificate Signing Request and sends it to the SLSM. Cryptographic operations are currently not hardware-accelerated and fully executed in software, resulting in up to 500mA.

When multiple certificates need to be renewed at the same time, an NLSM requests them sequentially, therefore not exceeding a certain power consumption threshold. I/O

operations to store certificates and verify metadata are slow on the Raspberry Pi. A full certificate renewal takes 2.5s.

After roughly 18 minutes, we simulated disconnecting a compute node. As soon as the device is reconnected to the network, all certificates have expired and need to be renewed. The sudden increase in traffic and power consumption can be seen around minute 22 in the plot.

We also measured the SLSM/ SLCA node. It has to serve all 4 other NLSMs. In our corresponding measurement it is more often at the shown peak energy consumption of 550mA and about 35% CPU load.

E. Scalability SLSM

The presented solution targets IoT nodes like single-board computers and SoCs capable of running Linux, such as Raspberry Pi, Beaglebone, or Intel Galileo. To prove that our solution runs without excessively increasing the load on existing devices, we analyzed how many certificate renewals per second an SLSM could handle.

Our measurement with a very high continuous amount of certificate renewal requests revealed that a maximum of 3.5 requests could be handled by the SLSM per second on the Raspberry Pi. The average processing time was 270 milliseconds per request. As the requests become more distributed over time, the necessity to have a higher request throughput on the SLSM diminishes in real deployments.

VI. CONCLUSION

We presented a fully distributed self-managing solution for equipping IoT services with a secure identity and secure metadata. Our approach covers the entire life cycle of a service from development over distribution to deployment, configuration, and update. It enforces its security policies that only authorized services can run locally, and that metadata changes are reflected even to disconnected nodes after a defined time, resulting in security-by design.

The presented solution is applicable to all distributed systems that are at least part-time connected. Our certificate-based solution provides integrity verification for service executables and their metadata. It identifies developers, stores, and sites at all stages of the hierarchical service management.

The proposed security architecture enables continuous changes of a service's metadata. It guarantees that the metadata of each running service is updated within maximum one certificate lifetime. If nodes are disconnected their services cannot run anymore after that time ensuring the highest security level. The autonomy of our fully local solution fits unattended nodes. It has a high scalability due to the decentralized operation.

Providing adequate security is a fundamental requirement for establishing the IoT. Integrity, non-repudiation, and confidentiality are key security properties that our solution provides. Higher-level security mechanisms such as authorization can only be deployed when these properties are provided.

Therefore, our work lays the base for implementing advanced security mechanisms such as access control that can then implement security and privacy of IoT data. Ideally this happens by-design.

GLOSSARY

CA	Certificate Authority
DS2OS	Distributed Smart Space Orchestration System
ID	Identifier
KA	Knowledge Agent
NLSM	Node-Local Service Manager
PKI	Public-Key Infrastructure
SLCA	Site-Local Certificate Authority
SLSM	Site-Local Service Manager
TLS	Transport Layer Security
VSL	Virtual State Layer

ACKNOWLEDGMENT

This research has been supported by the German Federal Ministry of Economic Affairs and Energy (BMWi) project DE-CENT (0350024A), the German Federal Ministry of Education and Research (BMBF) in the project DecADe (16KIS0538), and the German-French Academy for the Industry of the Future project SCHEIF.

REFERENCES

- [1] C. Koliadis, G. Kambourakis, A. Stavrou, and J. M. Voas, "DDoS in the IoT - Mirai and Other Botnets." *IEEE Computer*, 2017.
- [2] R. Want, "When Cell Phones Become Computers," *Pervasive Computing*, *IEEE*, vol. 8, no. 2, pp. 2–5, 2009.
- [3] S. Karnouskos, "Stuxnet worm impact on industrial cyber-physical system security," *IECON Proceedings (Industrial Electronics Conference)*, pp. 4490–4494, 2011.
- [4] D. Kushner, "The real story of stuxnet," *IEEE Spectrum*, vol. 50, no. 3, pp. 48–53, 2013.
- [5] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Cla, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016. [Online]. Available: <http://ieeexplore.ieee.org/document/7322178/>
- [6] V. Raychoudhury, J. Cao, M. Kumar, and D. Zhang, "Middleware for pervasive computing: A survey," *Pervasive and Mobile Computing*, Sep. 2012.
- [7] M.-O. Pahl and S. Liebald, "Designing a Data-Centric internet of things: Vsl," in *2019 International Conference on Networked Systems (NetSys) (NetSys'19)*, Garching b. München, Germany, Mar. 2019.
- [8] D. Lu, D. Huang, A. Walenstein, and D. Medhi, "A Secure Microservice Framework for IoT," in *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. *IEEE*, 2017, pp. 9–18.
- [9] M.-O. Pahl, G. Carle, and G. Klinker, "Distributed Smart Space Orchestration," in *Network Operations and Management Symposium 2016 (NOMS 2016) - Dissertation Digest*, 2016.
- [10] M.-O. Pahl and G. Carle, "The Missing Layer - Virtualizing Smart Spaces," in *10th IEEE International Workshop on Managing Ubiquitous Communications and Services 2013 (MUCS 2013, PerCom 2013 adjunct)*, San Diego, USA, 2013, pp. 139–144.
- [11] S. Misra, M. Maheswaran, and S. Hashmi, "Security Challenges and Approaches in Internet of Things," *Security Challenges and Approaches in Internet of Things*, 2017.
- [12] G. Strazdins and H. Wang, "Open security and privacy challenges for the Internet of Things," in *2015 10th International Conference on Information, Communications and Signal Processing (ICICS)*. *IEEE*, 2015, pp. 1–4.
- [13] A. Cavoukian, "Privacy by Design: Leadership, Methods, and Results." *European Data Protection*, pp. 175–202, 2013.
- [14] R. L. Rivest, "Can We Eliminate Certificate Revocations Lists?" *Financial Cryptography*, 1998.
- [15] M.-O. Pahl and F.-X. Aubet, "All eyes on you: Distributed Multi-Dimensional IoT microservice anomaly detection," in *2018 14th International Conference on Network and Service Management (CNSM 2018)*, Rome, Italy, Nov. 2018.
- [16] *Ubuntu Core 16 - Security*, Canonical, 8 2017, version 2.0.0 rc9.
- [17] *Android security white paper*, Google, 5 2015.
- [18] F. Cuadrado and J. C. Duenas, "Mobile Application Stores: Success Factors, Existing Approaches, and Future Developments," *IEEE Communications Magazine*, vol. 50, no. 11, pp. 160–167, Nov. 2012.
- [19] Apple Incorporated, "iOS Security Guide," Tech. Rep., May 2016.
- [20] M. Panwar and A. Kumar, "Security for IoT: An effective DTLS with public certificates," in *2015 International Conference on Advances in Computer Engineering and Applications (ICACEA)*. *IEEE*, 2015, pp. 163–166.
- [21] H. Kim, E. Kang, E. A. Lee, and D. Broman, "A Toolkit for Construction of Authorization Service Infrastructure for the Internet of Things." *IoTDI*, pp. 147–158, 2017.
- [22] S. Micali, "Efficient Certificate Revocation," Cambridge, MA, USA, Tech. Rep., 1996.
- [23] M. Naor and K. Nissim, "Certificate Revocation and Certificate Update." *USENIX Security Symposium*, 1998.
- [24] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, D. Boneh, and S. , "Towards short-lived certificates," 2012.
- [25] M.-O. Pahl and L. Donini, "Securing IoT Microservices with Certificates," in *Network Operations and Management Symposium (NOMS)*, Apr. 2018.
- [26] M.-O. Pahl and G. Carle, "Taking Smart Space Users into the Development Loop," in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. New York, NY, USA: ACM, 2013, pp. 793–800.
- [27] M.-O. Pahl, "Multi-tenant iot service management towards an iot app economy," in *HotNSM workshop at the International Symposium on Integrated Network Management (IM)*, Washington DC, Apr. 2019.
- [28] M.-O. Pahl and G. Carle, "Crowdsourced Context-Modeling as Key to Future Smart Spaces," in *Network Operations and Management Symposium 2014 (NOMS 2014)*, May 2014, pp. 1–8.
- [29] M.-O. Pahl, "Data-Centric Service-Oriented Management of Things," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, Ottawa, Canada, May 2015, pp. 484–490.
- [30] M.-O. Pahl and S. Liebald, "A modular distributed iot service discovery," in *International Symposium on Integrated Network Management (IM)*, Washington DC, USA, Apr. 2019.
- [31] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: <https://rfc-editor.org/rfc/rfc8446.txt>
- [32] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008. [Online]. Available: <https://rfc-editor.org/rfc/rfc5280.txt>
- [33] R. S. Sandhu, "Role-Based Access Control." *Advances in Computers*, 1998.
- [34] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, Feb. 1978.