Distributed Configuration, Authorization and Management in the Cloud-based Internet of Things

Martin Henze, Benedikt Wolters, Roman Matzutt, Torsten Zimmermann, Klaus Wehrle
Communication and Distributed Systems, RWTH Aachen University, Germany
{henze, wolters, matzutt, zimmermann, wehrle}@comsys.rwth-aachen.de

***Abstract*—Network-based deployments within the Internet of Things increasingly rely on the cloud-controlled federation of individual networks to configure, authorize, and manage devices across network borders. While this approach allows the convenient and reliable interconnection of networks, it raises severe security and safety concerns. These concerns range from a curious cloud provider accessing confidential data to a malicious cloud provider being able to physically control safety-critical devices. To overcome these concerns, we present D-CAM, which enables secure and distributed configuration, authorization, and management across network borders in the cloud-based Internet of Things. With D-CAM, we constrain the cloud to act as highly available and scalable storage for control messages. Consequently, we achieve reliable network control across network borders and strong security guarantees. Our evaluation confirms that D-CAM adds only a modest overhead and can scale to large networks.**

I. Introduction

The Internet of Things (IoT) enables the world-wide interconnection of "smart things" [1] with the goal of enhancing important aspects of everyday life, e.g., in pervasive health care, assisted living, and smart cities [2]. As IoT devices allow to directly influence the physical world (e.g., Internet-connected implanted medical devices [3] or robotic arms in factories [2]), securing access control for these devices is of utmost importance to prevent severe damage [3]. Traditionally, configuration, authorization, and management are realized *within* individual networks, e.g., via cryptographically enforced access control lists [4]–[6]. This allows a user to efficiently manage and secure a *single* network within the IoT.

However, we observe an increasing trend of interconnecting previously isolated IoT networks [1]. This trend ranges from users who want to interconnect their body area network and home network to companies bridging complete factories via the Internet [1], [7]. The predominant approaches to realize these interconnections utilize the high availability and elastic resources of the cloud [1], [5], [7]. In this setting, the cloud is used to facilitate management of networks and devices as well as to configure and authorize access to devices *across* network borders. This enables device owners to configure, authorize, and manage access to their devices across different networks without having to care about network borders. More specifically, a user can manage and configure devices in different networks from a single location without having to take care of the availability and reachability of individual devices that, e.g., reside behind a firewall.

Besides these enormous benefits, outsourcing configuration, authorization, and management of (potentially safety-critical) devices to the cloud poses huge security threats [8], [9]. These threats range from a curious cloud provider accessing confidential data to a malicious provider gaining physical control over safety-critical devices. This includes rogue employees of the cloud provider and possible security breaches, jeopardizing the security of all cloud-controlled devices.

Hence, in this paper, we tackle the challenge of securely realizing configuration, authorization, and management in the cloud-based IoT. Due to the potential severity of attacks enabled by physical control, our prevalent focus lies on preventing a malicious cloud provider from controlling IoT devices. To this end, we present D-CAM, our solution for achieving **d**istributed **c**onfiguration, **a**uthorization, and **m**anagement *across* borders between IoT networks. D-CAM runs on the user-controlled gateways of individual networks and enables users to configure their *complete* federation of IoT networks from a single location. In contrast to entirely configuring IoT networks in a central manner, D-CAM reduces the cloud to act as a highly available and scalable proxy for storing and forwarding tamper-resistant control messages. This way, we achieve a reasonable trade-off between the advantages of the cloud-based IoT and strong security guarantees. More precisely, we make the following contributions:

1) We analyze the scenario of cloud-interconnected IoT networks and the resulting security challenges, especially in the presence of a malicious cloud provider.
2) To account for these security challenges, our distributed architecture, D-CAM, allows to configure, authorize, and manage IoT devices across network borders via the cloud. D-CAM ensures that only authorized parties can configure a user's IoT devices. Even a malicious cloud provider cannot tamper with the configuration of IoT devices.
3) To illustrate the feasibility of D-CAM, we fully implemented a working prototype and extensively quantify the incurred processing and storage overheads. Our results show that D-CAM can easily scale to large networks.
4) To further increase the security of D-CAM, we additionally provide a mechanism for confidentiality of configuration, authorization, and management messages.

II. Controlling IoT Networks

In this section, we provide a brief overview of our envisioned network scenario. From this we derive the security challenges for achieving secure configuration, authorization, and management for cloud-interconnected IoT networks.

© IEEE, 2017. This is the author's version of the work. It is posted here by permission of IEEE. Not for redistribution. Cite as: M. Henze, B. Wolters, R. Matzutt, T. Zimmermann, K. Wehrle: 'Distributed Configuration, Authorization and Management in the Cloud-based Internet of Things', IEEE TrustCom 2017.

A. Network Scenario and Problem Analysis

In traditional IoT deployments, a network of IoT devices is connected to the Internet (and possibly the cloud) via a gateway controlled by the user (in rare cases, an IoT device acts as the gateway directly). We assume that the communication *within* the IoT network is properly secured [4], i.e., the internal IoT network communication provides authentication and integrity protection. To allow for interaction with an IoT network over the Internet, it needs to be properly configured. This involves (i) *configuration* of individual IoT devices, (ii) *authorization* of access to these devices (e.g., for sensing and actuating), and (iii) *management* of the overall IoT network and its structure. In the following, we refer to these operations as *control operations*. Handling control operations is well-studied for traditional single-network deployments. Such networks are typically configured on the single user-controlled *gateway* that connects to the Internet and hence is predestined to enforce all control-related tasks [4]. For example, as the gateway manages connections to the Internet, it will only forward legitimate requests to its IoT devices.

However, as the IoT evolves, we see an increasing trend for bridging several IoT networks over the Internet. Conveniently and consistently managing a *federated* IoT network is challenging. In an naïve approach, SSH or VPNs could be used to remotely control small groups of IoT networks. However, this requires gateways to be addressable (not behind a firewall or NAT) and available (not offline, e.g., due to an unreliable wireless uplink) at configuration time, which is unrealistic for dynamic environments such as the IoT. Current state-of-the-art approaches [10]–[12] thus propose to steer control operations from the cloud. Using the cloud as a central hub to manage IoT devices of one owner across network borders eliminates the need for managing each network separately and for setting up remote management solutions. In this setting, the network owner sends control messages to the cloud, which will relay them to all gateways in the owner's federated IoT network (if a gateway is offline, it will be updated as soon as it comes back online). Such control messages can be sent in a variety of formats and protocols, e.g., using CoAP, SNMP, or NETCONF [13]. Hence, it is important to develop a system that is agnostic to the specific format and protocol for control operations.

B. Security Analysis

While the cloud enables the owner of a federated IoT network to perform control operations efficiently, this comes at the price of severe security challenges. In cloud-based systems, the prevalent security assumption is that the cloud provider can be partially trusted. Specifically, the cloud provider is typically considered to be semi-honest or honest-but-curious [14]–[17]. That is, it will not disrupt the execution of the protocol and is hence limited to merely passively gathering information. Most importantly, a cloud provider, under these assumptions, will not tamper with messages it is supposed to relay to other nodes in the network. This is a widespread and reasonable assumption if the primary goal is to protect the confidentiality of data. However, as the IoT connects the physical world to the Internet, security in the cloud-based IoT is not only about the privacy of information but also required to guarantee (physical) safety. As a severe example, an adversary could remotely gain control over a pacemaker to modify a patient's heart rate [18] after gaining access to the cloud. Hence, only assuming an honest-but-curious cloud provider when considering control operations in the cloud-based IoT does not offer adequate protection for safety-critical tasks.

We derive severe attacks a dishonest cloud provider (or rogue employees and entities attacking the cloud) can launch:
Modification Attack: Changing messages before forwarding them, e.g., to change parameters in a configuration message.
Insertion Attack: Creating new messages and sending them to devices in the network, e.g., to gain access to a specific device. This class of attacks also includes duplication of legitimate messages (replaying) to cause an inconsistent system state.
Reorder Attack: Changing the order of messages before distributing them in the network, e.g., to change the semantics of the request contained in the messages.
Withhold Attack: Deciding to (temporarily) not pass on certain messages to the network, e.g., to block the de-authorization of access to devices.

All these attacks have in common that they can lead to severe consequences, e.g., if the cloud provider (or a rogue employee or someone attacking the cloud provider) uses them to gain control over an actuator in the physical world. To account for these types of attacks, we assume a *malicious-but-cautious cloud provider* [16] and protect our system accordingly. In this attacker model, the cloud provider can launch any attack as long as this leaves no evidence. Notably, this does not imply that the cloud provider indeed behaves maliciously. Rather, it acknowledges that the cloud provider (or an employee) *can* potentially behave maliciously or be subject to attacks. Neglecting the resulting attack vectors would enable attackers, e.g., to gain control over devices in the user's IoT network without leaving any evidence. This attacker model is especially well-suited for our scenario, as cloud providers face serious consequence if misconduct is detected.

In this work, we do not aim to protect against insider attacks, e.g., by hacked gateways within the IoT network. Still, we show that D-CAM provides accountability, i.e., misbehavior of gateways (through errors or attacks) can be identified.

III. D-CAM Design

The goal of this paper is to overcome the identified severe security challenges by realizing distributed configuration, authorization, and management (control operations) in the cloud-based IoT in the presence of a malicious-but-cautious cloud provider. We present D-CAM, our solution that bases on the principle of hash chains [19] to create a distributed administrated log of *control messages* performing control operations. This allows us to create a secure timeline [20] of these messages, which can be verified by any gateway in the federated IoT network. In the following, we focus on achieving integrity and availability of control messages. We describe how to additionally achieve message confidentiality in Section VI.

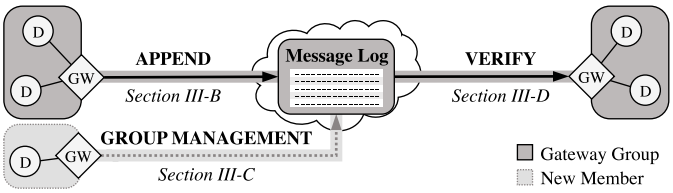

Fig. 1: D-CAM's design centers around a message log which allows gateway group members to append new messages and verify the contained messages.

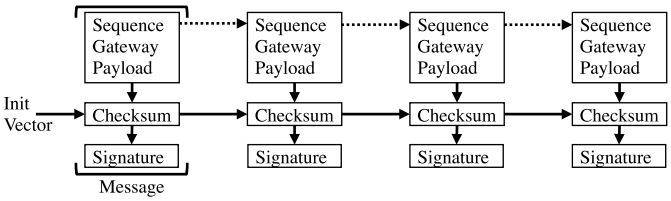

Fig. 2: Each message in the message log is digitally signed by the originating gateway. All messages in the message log are interlinked via a hash chain.

A. Design Overview

D-CAM operates in a scenario where multiple IoT networks are interconnected via the cloud to form one larger, virtual IoT network. As each individual IoT network is connected to the cloud via a dedicated user-controlled gateway, their interconnection requires the federation of said gateways, which we refer to as a user's *gateway group*. The task of D-CAM is the reliable distribution of control operations to all gateways in a gateway group in the presence of a malicious-but-cautious cloud provider. We assume that each gateway has a cryptographic identity, i.e., a public/private key pair, and is controlled by the user owning the IoT network.

D-CAM relies on distributed managed, cloud-hosted *message logs* for each gateway group, to which all members of the gateway group can securely append messages as illustrated in Figure 1. Furthermore, each gateway can verify integrity and authenticity of the message log. Messages in the message log immediately reflect control operations. As our focus lies on realizing the secure distribution of *arbitrary* control messages in federated IoT networks, we deliberately abstract from specific approaches for configuring individual IoT devices (e.g., CoAP, SNMP, or NETCONF). Furthermore, control messages in D-CAM also include the management of the gateway group itself, i.e., adding and removing gateways (dashed line in Figure 1). In summary, D-CAM's message log is maintained in a distributed manner within gateway groups and the cloud is reduced to a highly available message store and relay.

B. Appending to the Message Log

The goal of D-CAM is to ensure that only authorized gateways can append control messages to the message log. Furthermore, no one should be able to modify, reorder, or remove messages. To achieve this goal, we protect control messages with a combination of sequence numbers, a hash chain, and digital signatures as shown in Figure 2.

We describe the process of appending one message to the message log and from now on refer to the gateway appending the message as its *initiator*. To avoid message collisions, the initiator reads the sequence number of the most recent message, increases it by one, and adds it to the new message (dashed lines in Figure 2). If two gateways simultaneously append a message, they will use the same sequence number and hence D-CAM is able to detect and resolve the collision.

Furthermore, the initiator creates a checksum over the message itself and the checksum of the directly preceding message using a cryptographic hash function (solid lines in Figure 2). Thereby, we create a hash chain [19] that cryptographically links all messages in the message log. Due to the pre-image resistance of cryptographic hash functions, messages can neither be altered nor reordered without invalidating the hash chain. The first message in the message log contains a random *initialization vector* instead of a previous checksum.

To allow other gateways in the gateway group to verify the integrity and authenticity of a message, the initiator digitally signs each message. This signature covers the checksum and thus also ensures integrity and authenticity of all previous messages. Subsequently, the initiator sends the message to the cloud, where it is stored and distributed to all gateways in the gateway group. Gateways that are offline or temporarily unavailable will update when they come back online.

Optimization. Creating reasonably secure digital signatures leads to a non-negligible performance overhead. Hence, with D-CAM we aim to reduce the amount of required digital signatures *without* diminishing the security level. We observe that in IoT deployments control messages often arrive in batches, e.g., if the network owner configures new devices or changes authorization of device access. If a gateway appends a batch of messages to the message log, it will add a digital signature only to the last message and send the complete batch to the cloud. The integrity and authenticity of the other messages in the batch remains enforced by the hash chain.

C. Management of Gateway Groups

D-CAM uses the message log to secure the management of gateway groups, i.e., ensure that only authorized gateways participate in a gateway group. Thus, D-CAM provides the same security level for gateway group management as for control operations. We consider three group management operations: (i) *creation* of a gateway group, (ii) *adding* gateways to a group, and (iii) *removing* gateways from a group.

When creating a federated IoT network, the network owner also *creates* a new gateway group. To do so, she connects to one of her gateways and creates the gateway group, as well as a corresponding message log with a random initialization vector. To announce the creation of this new group and implicitly adding itself as the first group member, the gateway creates an initial message using the initialization vector as identifier.

To *add another gateway* to her gateway group, the network owner connects to the new gateway and creates a join request that is stored in the cloud (outside the message log). Now, she can connect to any gateway in her gateway group to review and accept the pending join request, thereby validating the public key of the joining gateway. To complete adding the gateway to the gateway group, a group member appends a message to

the message log that grants the public key of the new group member the right to append messages to the message log. Now, the new gateway is a full member of the gateway group.

Removing gateways from a gateway group in D-CAM works similarly to adding gateways. Any member of the gateway group can append a message to the message log that removes another gateway from the gateway group by revoking its public key. Upon receiving this message, the remaining members will not accept any further messages signed by the removed entity.

Optimization. In certain scenarios, it might not be desirable to allow each gateway group member to perform control operations, e.g., if a gateway is deployed in an untrustworthy environment or physically exposed. Hence, D-CAM also supports *passive gateways*, i.e., gateways that can only be configured using D-CAM but cannot initiate control operations. Gateways suspected to be especially vulnerable thus do not jeopardize the security of the whole network if they are compromised.

IoT devices themselves can also be managed with D-CAM. Here, D-CAM additionally stores routing information in the message log, i.e., to which gateway a device is connected.

D. Verifying the Message Log

Whenever a gateway receives a message batch from the cloud, D-CAM must verify its integrity and authenticity. The gateway verifies each message sequentially: First, the gateway verifies the message's checksum by computing the hash value over the message and the previous message's checksum. Then, the gateway reads the public key of the message's initiator from a local cache. The cache is updated whenever a non-passive gateway is added or removed (cf. Section III-C). This ensures that only messages by authorized gateway group members are accepted by D-CAM. Finally, the gateway verifies the message's signature and continues with the next message.

Optimization. In duality to appending messages, processing time for verifying a message is dominated by checking the digital signature. Again, our scheme based on hash chains allows us to selectively employ an optimization. D-CAM can verify message batches by iteratively checking the checksum of each message but verifying only the signature of the last message. With this optimization, we can guarantee the correctness of all messages in the batch only after verifying the last message. Hence, the batch size constitutes a trade-off between improved verification time and required buffer space as well as more complicated failure recovery. Notably, this does not constitute a trade-off between security and performance.

E. Trimming the Message Log

The cumulated amount of control messages generated by a gateway group will steadily increase over time. This becomes problematic as gateways joining a gateway group after a while need to process an excessive amount of messages to catch up with the current network state. At the same time, we observe that older control messages might be obsoleted by new messages, e.g., when overwriting a configuration or revoking an authorization. To leverage this potential for space reduction, D-CAM allows to *trim* the message log by eventually starting a new message log based on the network state at the time of trimming, thereby pruning all obsoleted messages. This allows for significantly shorter bootstrapping times for new gateways.

A dedicated gateway group member (e.g., the oldest) constantly monitors the message log's amount of obsolete messages. If this amount exceeds a specific threshold (group or device dependent), the dedicated gateway trims the message log. To this end, the gateway uploads a complete snapshot of the current network state to the cloud and adds a snapshot message to the message log. The snapshot message contains the snapshot's storage location and the hash value of the snapshot. When a new gateway joins a gateway group, it is provided with the hash over the latest snapshot and thus only has to verify the message log starting from the latest snapshot.

IV. Security Discussion

We briefly discuss how D-CAM protects against the attacks we identified (cf. Section II-B) and hence guarantees integrity and authenticity of control operations in the cloud-based IoT.

Modification Attack. Digital signatures ensure that no unauthorized entity, e.g., a malicious cloud provider, can modify a message. Any modification will invalidate the message's signature and is easily detectable by any member of a gateway group, causing a malicious-but-cautious cloud provider to refrain from launching this attack. Even with our optimization to not sign each message, we can easily detect mismatches in the hash chain if non-signed messages are modified.

Insertion and Reorder Attacks. No unauthorized entity can append new messages to the message log as they are unable to create valid digital signatures. Replaying, i.e., duplicating legitimate, signed messages, is prevented as this would imply recurring sequence numbers and checksum mismatches in the hash chain. The same detection strategy can be used for preventing reordering attacks, which would result in a mismatch in sequence numbers and a broken hash chain.

Withhold Attack. In contrast, detecting withholding of messages requires additional effort. We briefly outline two approaches: First, the members of a gateway group can use a side channel (e.g., by directly contacting each other) to periodically exchange status information, i.e., the sequence number and checksum of the latest message. Second, and without a side channel, each gateway can periodically append a heartbeat message to the message log, indicating that currently no updates are to be expected. As gateway group members need to be updated to the latest version to append to the message log, this will detect missing heartbeat messages, which indicates either a gateway failure or a withhold attack. This approach's overhead can be parameterized by adjusting the heartbeat frequency. Furthermore, its storage overhead can be limited by trimming older heartbeats (cf. Section III-E).

Further Security Considerations. When adding gateways to a gateway group, the cloud provider might withhold or modify join requests. The network owner will notice such attacks when reviewing join requests (cf. Section III-C). When trimming the message log, the snapshot stored in the cloud cannot be modified as the hash value cryptographically binds

the snapshot to the message log (cf. Section III-E). Although not specifically designed to protect against insider attacks, D-CAM provides a tamper-resistant, verifiable log of all control operations. Hence, we can detect misbehavior (e.g., device defects or attacks) and blame the originating gateway.

To conclude, D-CAM's approach of a cryptographically protected message log offers protection against the identified attacks, even in the presence of a powerful attacker. Attack attempts are detected by D-CAM which prevents, e.g., physical harm. Furthermore, network owners can launch appropriate countermeasures and collect evidence of attacks.

V. Evaluation

To prove the feasibility of D-CAM and quantify its performance, we evaluate its processing, storage, and communication overheads. Based on these results, we compare D-CAM to other remote management approaches such as VPNs or SSH. As a basis for our evaluation, we implemented a prototype for the gateway component in the C programming language. We rely on OpenSSL 1.0.1k for the cryptographic operations, libjansson 2.7 for serializing messages using JSON, and MySQL 5.5 for persistently storing state at the gateways, e.g., the list of gateways in the gateway group. As an exemplary embedded device for the gateway, we chose the Raspberry Pi Model B+ with a 700 MHz ARM11 processor, 512 MB of RAM, and Raspbian Jessie Lite Linux as operating system. To properly select the employed cryptographic primitives, we followed the recommendations of NIST [21]. More precisely, we use SHA-256 as hash function as well as two different digital signature schemes (to enable their comparison): RSA with 2048 Bit keys and ECDSA with NIST curve P-256.

A. Processing Overhead

First, we evaluate the processing overhead for appending messages to and verifying messages in the message log. We refer to a *signing interval* of k if a gateway signs on average each k-th message (cf. Section III-B). Analogously, a *verification interval* of k means that a gateway on average checks the digital signature of each k-th message (cf. Section III-D). For each result, we performed 30 runs, each consisting of the processing, i.e., appending or verifying, of 10 000 messages. We show the mean processing time for one message with 99% confidence intervals. We distinguish between the time required for creating respectively verifying the hash chain and the digital signature, including parsing and serializing messages, the lookup of public keys, and all other calculations.

Appending to the Message Log. The processing time for a gateway to append one message to the message log is influenced by the signing interval and the message length.

First, we vary the signing interval between 1 and 25 and fix the message length to 2 500 Byte, which allows to encode even larger control messages. Our results in Figure 3 (note the logarithmic scale in this plot) show that the processing time for creating the hash chain does not depend on the signing interval while the time for creating the digital signatures significantly decreases for an increasing signing interval.

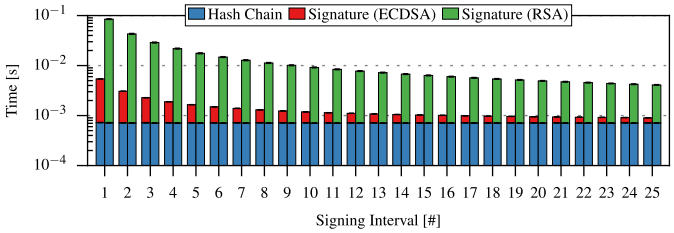

Fig. 3: Mean duration for appending a message of length 2 500 Byte to the message log, depending on varying signing intervals. Increasing the signing interval reduces the average time spend in the predominant signing operation.

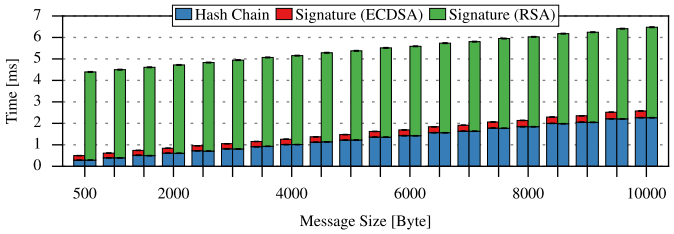

Fig. 4: Mean duration for appending a message of varying payload size to the message log with a signing interval of 20 and a group size of 1. The time for appending one message increases linearly with the payload size.

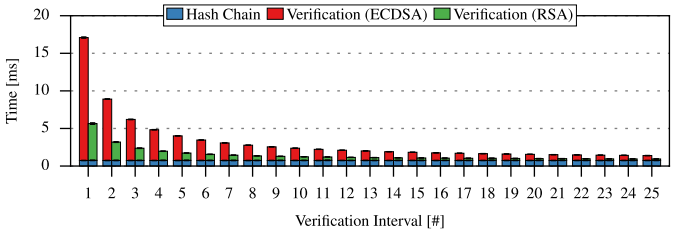

Fig. 5: Mean duration for verifying one message of size 2 500 Byte in the message log, depending on the verification interval. By increasing this interval the average time spent for the predominant verification operation is reduced.

Especially for smaller signing intervals, we see that ECDSA strongly outperforms RSA as expected due to their different performance asymmetries [4]. For a signing interval of 20, using RSA (ECDSA) allows to append 202 messages/s (1 052 messages/s). Furthermore, we observe only little additional savings for increasing the signing interval beyond 20, especially when using ECDSA as signature scheme.

Hence, we now fix the signing interval to 20 and vary the message length between 500 and 10 000 Byte in steps of 500 Byte. In Figure 4, we observe that processing time increases roughly linearly for increasing message sizes. This is mainly due to an increased time for creating the checksum for longer messages. Again, we observe a superior performance of ECDSA compared to RSA. For a message size of 500 Byte, we can process 228 messages/s (2 004 messages/s) with RSA (ECDSA). This decreases to 154 messages/s (388 messages/s) with RSA (ECDSA) for a message size of 10 000 Byte.

Verifying the Message Log. The processing time for verifying a message depends on the verification interval and the message size. Additionally, processing time might be influenced by the number of gateways that created messages.

To study the influence of the verification interval on processing time, we vary the verification interval between 1 and 25. Message size is fixed to 2 500 Byte and the gateway group size to 1. As shown in Figure 5, the time for verifying hash chains

does not depend on the verification interval while the time for verifying digital signatures decreases with an increased verification interval. Here, RSA benefits from the performance asymmetry and outperforms ECDSA. For a verification interval of 20, RSA enables us to verify 991 messages/s compared to only 611 messages/s for ECDSA. Increasing the verification interval beyond 20 offers little performance gains.

Hence, we now set the verification interval to 20 while keeping the group size at 1 and evaluate the impact of varying the message size between 500 and 10 000 Byte in steps of 500 Byte and depict the results in Figure 6. The time for verifying one message increases approximately linearly with an increasing message size. This stems from an increase in verifying the hash chain checksums and validating the digital signature. We again notice a superior performance of RSA over ECDSA. Using RSA (ECDSA) allows us to verify 1 840 messages/s (888 messages/s) for messages of size 500 Byte. For a message size of 10 000 Byte, these numbers decrease to 387 messages/s (315 messages/s) for RSA (ECDSA).

Next, we analyze the impact of the number of gateways in the gateway group. We fix the verification interval to 20, message size to 2 500 Byte, and increase gateway group size from 1 to 100. Gateways will append messages one by one, i.e., the first gateway will append its second message only after all other gateways have appended a message. Our results in Figure 7 show that the verification time does not depend on the number of gateways in the gateway group.

Remarks. Setting both signing and verification interval to 20 constitutes a reasonable trade-off between processing time and required buffer space for verification. Furthermore, if the goal is to optimize performance of appending messages in D-CAM, ECDSA is preferable over RSA. However, RSA shows a superior performance for verifying messages. For a gateway group of size n, a message has to be verified by n gateways while it is appended only once. Thus, especially for large gateway groups, selecting RSA is recommended. Notably, the size of the gateway group does not influence the time required for verifying messages. This is expected as long as we can keep the public keys of all gateway group members in memory. Even on a resource-constrained Raspberry Pi, we can cache the public keys of hundreds of gateways.

B. Storage and Communication Overhead

To analyze the per-message storage overhead as well as the influence of trimming the message log on communication, we rely on analytical and simulative methods.

Per-Message Overhead. The per-message overhead of D-CAM stems from header fields (e.g., sequence number and initiator identifier), the checksum required for the hash chain, and the digital signature. More precisely, the overhead consists of 36 Byte for the header, 32 Byte for the checksum, plus 258 Byte (72 Byte) for encoding the RSA (ECDSA) digital signature. As the size of the header, hash, and signature stay constant for varying payload sizes, this overhead decreases from 65.2% (28%) for messages of size 500 Byte to 3.26% (1.4%) for messages of size 10 000 Byte for RSA (ECDSA).

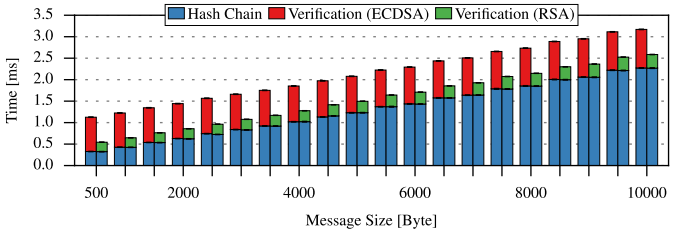

Fig. 6: Mean duration for verifying a message of varying payload size with verification interval 20 and a group size of 1. Due to checking the hash chain, the time required for message verification scales linearly with the payload size.

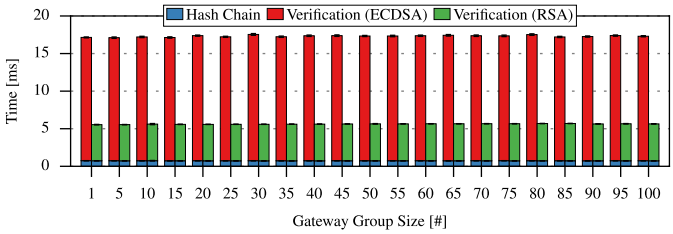

Fig. 7: Mean duration for verifying one message of length 2 500 Byte in the message log, depending on the number of gateways in the gateway group. The group size has a negligible impact on the entire verification duration.

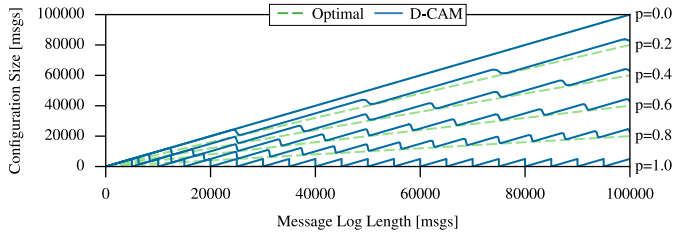

Fig. 8: The influence of trimming the message log depends on the probability of messages being obsoleted ($p = 0, 0.2, ..., 1$). D-CAM at most incurs a fixed overhead (the specific amount is a configurable parameter).

Influence of Trimming the Message Log. The behavior of D-CAM's trimming approach depends on the amount of obsolete messages in the message log. We study this behavior with a simulative approach where we consider message logs of up to 100 000 messages and let D-CAM trim the message log whenever it observes at least 5 000 obsolete messages. We iteratively append messages, where each inserted message may obsolete a previous one with a probability of $0, 0.2, ..., 1$. In Figure 8, we compare the number of messages a new gateway has to process to the optimal number, i.e., only non-obsoleted messages. Each experiment was conducted 1 000 times with random seeds and we depict the mean amount of messages to be verified. We omit confidence intervals to ease readability, as the 0.99 confidence intervals are < 204 messages for all values. Indeed, our results show that D-CAM at most incurs a fixed overhead of 5 000 messages (the specific amount is one of D-CAM's parameters). Furthermore, the number of trimming operations required (indicated by drops) scales with the probability of obsolete messages, ranging from 20 when all messages are obsoleted to 0 if none become obsolete.

Remarks. Our evaluation of D-CAM's storage and communication overhead leads to two observations. First, if reducing storage space is important, using ECDSA as signature scheme is the better choice. Notably, the required storage space can further be reduced by increasing the signing interval (cf. Sec-

tion III-B). Second, when considering the amount of messages that need to be processed by a new gateway joining a gateway group, D-CAM incurs only a constant overhead compared to an optimal solution that directly deletes any obsolete messages.

C. Comparison to Other Remote Management Approaches

Although D-CAM provides more functionality, e.g., group management and an audit log, than established remote management approaches such as VPNs or SSH, it is still interesting how D-CAM performs compared to said approaches. As our goal is to achieve a *consistent* configuration of the whole federated IoT network, VPNs and SSH require one connection from each gateway to each other gateway to communicate all control operations. In a network of N gateways, this results in sending N messages for each control operation and adding as well as maintaining N new connections for each new gateway. Considering bandwidth constraints of gateways, e.g., mobile uplinks, this becomes infeasible already for small networks. Contrarily, D-CAM only sends one message per control operation from a gateway to the cloud, irrespective of the network size. Thus, D-CAM's scalability is not bound by bandwidth. Furthermore, D-CAM's design reduces setup and management costs and is less susceptible to misconfiguration.

To quantitatively compare D-CAM to VPNs and SSH, we performed measurements using our evaluation setup. We use OpenVPN 2.3.4 as well as OpenSSH 6.7 with RSA 2 048 Bit keys and AES-256 in CBC mode (providing the same security level as D-CAM). The transmission of a message of size 2 500 Byte over OpenVPN (OpenSSH) results in $2\,925\times N$ Byte ($2\,766\times N$ Byte) application layer payload (for group size N), compared to 2 826 Byte in D-CAM (irrespective of group size). Hence, already for networks of 3 gateways, D-CAM significantly reduces the communication overhead compared to utilizing VPNs or SSH. We observe similar trends for processing time (signing/verification interval of 20).

D. Concluding Observations

We specifically designed D-CAM to scale to large network sizes. Our evaluation results confirm that the processing time for appending to the message log and verifying it are not noticeably impacted by the size of the gateway group. Similarly, the storage and communication overhead of D-CAM does not depend on the group size. D-CAM scales linearly in the size of the message log, being bound only by available storage space. Our message log trimming approach further helps in reducing required storage space. Additionally, D-CAM has no trade-off between security and performance. We provide the same level of security as digital signatures and additionally protect against modification, insertion, reordering, and withholding of messages. Increasing D-CAM's signing and verification intervals allows to reduce the processing overhead. The trade-off here is that messages must be buffered at a receiving gateway before they can be verified. To conclude, D-CAM provides a high level of security against powerful adversaries such as a malicious-but-cautious cloud provider at reasonable costs with respect to processing and storage overhead.

VI. Confidentiality

Certain scenarios also require the confidentiality of control messages, e.g., configurations of IoT devices may be industry secrets. In D-CAM, we thus can encrypt all control messages to only allow authorized gateways to read their content. We efficiently encrypt messages using a symmetric *group key* (e.g., using AES-256) shared within a gateway group. However, the possibility to arbitrarily add or remove gateways renders the key distribution challenging as we demand that gateways are only able to read messages from the message log that were appended during their membership. Hence, we change and redistribute the group key whenever group membership changes. For exchanging the group key, we rely on the public keys of gateways. Each time a gateway appends a message to add or remove another gateway, it also changes the group key. The group key is then encrypted for each gateway that will be a member after the addition or removal using its public key and then appended to the message log. Thus, only current group members can decrypt the new group key and thus any following messages. An in-depth analysis of the incurred processing overheads [4] shows that overheads are reasonable and well worth the additional protection of confidentiality.

VII. Related Work

Several approaches to *control access to data in the cloud-based IoT* have been proposed. In the context of health data, Lounis et al. [5] leverage attribute-based encryption to perform access control. Similar approaches have been proposed by Thilakanathan et al. [22] based on a secure data sharing protocol and Liu et al. [23] by employing attribute-based signcryption. On a more general scale, SensorCloud [4], [24] provides a generic security architecture for outsourcing IoT data to the cloud. In all of these approaches, access control is solely performed to protect the confidentiality of data and does not consider the potentially safety-critical access to actuation capabilities. This problem is addressed by Picazo-Sanchez et al. [6], who realize fine-grained access control for commands sent to an IoT device. However, their ciphertext-policy attribute-based encryption scheme induces processing overheads in the order of seconds compared to D-CAM's overhead in the order of milliseconds. Furthermore and in contrast to our work, all previous approaches do not consider secure federation of IoT networks. They either require a central trusted entity for access control [5], [22], [23] or operate solely within isolated networks [4], [6], [24]. In contrast, D-CAM realizes full configuration, authorization, and management in the cloud-based IoT. Porambage et al. [25] realize secure multicast in the IoT. They, however, do not consider many-to-many messages and the management of gateway groups.

Secure audit logs protect integrity and authenticity of log files [26]. Schneier and Kelsey [26] present a generic secure logging scheme that allows to detect deletion or modification attempts even on compromised hosts. Waters et al. [27] propose an encrypted and searchable audit log that also protects confidentiality. Although these approaches do not consider a distributed setting, i.e., multiple entities contributing to a log,

they provide us with valuable input. Especially searchable encryption would allow to decrypt only relevant messages in the message log. Considering a distributed setting, Accorsi [28] proposes to apply trusted computing to ensure authenticity and confidentiality of log entries. In contrast, we do not require an additional entity that can become a single point of failure.

Finally, our approach is inspired by well-established *blockchain* approaches. Bitcoin [29] uses a blockchain to store monetary transactions. It has been extended to implement decentralized lookup stores [30] and access control [31]. In contrast to these approaches, D-CAM's inherently strong trust within gateway groups eliminates the need for costly consensus protocols, e.g., block mining in Bitcoin. Performance improvements proposed for Bitcoin, such as block pruning or leader election [32], are similar to the storage optimizations of D-CAM. However, block pruning still requires to verify the whole blockchain when joining the system while D-CAM requires to only verify messages since the last trimming.

VIII. Conclusion

In this paper, we present D-CAM to realize *distributed* configuration, authorization, and management in the cloud-based IoT *across* borders of IoT networks. D-CAM runs on the gateways in a federated IoT network and allows users to control their *complete* federated IoT network from each of their gateways without having to care about the reachability and availability of individual devices. D-CAM utilizes the concepts of hash chains and digital signatures to create a secure and distributed administrated log of control messages stored in the cloud, thereby restricting the cloud to act as a highly available and scalable proxy for relaying and storing secured control messages. This allows us to ensure the integrity, authenticity, and confidentiality of control messages, even in the presence of a powerful attacker such as a malicious-but-cautious cloud provider. D-CAM's tamper-resistant log of all control operations additionally allows to detect and pinpoint internal attackers. Thus, and in contrast to related work, D-CAM is especially well-suited for controlling access to actuating capabilities of safety-critical devices.

As our evaluation shows, D-CAM's high level of security comes at modest costs. Even on a resource-constrained gateway, D-CAM is able to process more than 640 messages per second for a reasonable choice of system parameters. Notably, D-CAM's processing overhead depends only on the number of messages to be processed and does not increase with the size of the gateway group. Furthermore, D-CAM's message log trimming scheme results in at most a fixed storage overhead compared to a system managing configuration, authorization, and management centrally in the cloud. D-CAM does not only show comparable performance to other remote management approaches (e.g., VPNs and SSH) for small networks but significantly scales better for larger networks. In conclusion, D-CAM allows to securely realize distributed configuration, authorization, and management in cloud-interconnected IoT networks even in the presence of powerful attackers at modest costs in terms of processing and storage overhead.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation program 2014-2018 under grant agreement no. 644866 and the Excellence Initiative of the German federal and state governments. This manuscript reflects only the authors' views and the funding agencies are not responsible for any use that may be made of the information it contains.

References

[1] M. Henze *et al.*, "A Comprehensive Approach to Privacy in the Cloud-based Internet of Things," *FGCS*, vol. 56, 2016.
[2] L. Atzori *et al.*, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, 2010.
[3] J. Sametinger *et al.*, "Security Challenges for Medical Devices," *Commun. ACM*, vol. 58, no. 4, 2015.
[4] R. Hummen *et al.*, "A Cloud Design for User-controlled Storage and Processing of Sensor Data," in *IEEE CloudCom*, 2012.
[5] A. Lounis *et al.*, "Secure and Scalable Cloud-Based Architecture for e-Health Wireless Sensor Networks," in *ICCCN*, 2012.
[6] P. Picazo-Sanchez *et al.*, "Secure Publish-Subscribe Protocols for Heterogeneous Medical Wireless Body Area Networks," *Sensors*, vol. 14, no. 12, 2014.
[7] W. Haerick *et al.*, "5G and the Factories of the Future," 5G-PPP White Paper, 2015.
[8] M. Henze *et al.*, "Moving Privacy-Sensitive Services from Public Clouds to Decentralized Private Clouds," in *CLaw Workshop*, 2016.
[9] M. Henze *et al.*, "Veiled in Clouds? Assessing the Prevalence of Cloud Computing in the Email Landscape," in *TMA*, 2017.
[10] F. Li *et al.*, "Efficient and Scalable IoT Service Delivery on Cloud," in *IEEE CLOUD*, 2013.
[11] M. Henze *et al.*, "User-driven Privacy Enforcement for Cloud-based Services in the Internet of Things," in *FiCloud*, 2014.
[12] A. Botta *et al.*, "Integration of Cloud computing and Internet of Things: A survey," *FGCS*, vol. 56, 2016.
[13] M. Slabicki and K. Grochla, "Performance Evaluation of CoAP, SNMP and NETCONF Protocols in Fog Computing Architecture," in *IEEE/IFIP NOMS*, 2016.
[14] O. Goldreich, *Foundations of Cryptography: Basic Applications*, 2004.
[15] M. Henze *et al.*, "CPPL: Compact Privacy Policy Language," in *ACM WPES*, 2016.
[16] M. D. Ryan, "Enhanced Certificate Transparency and End-to-End Encrypted Mail," in *NDSS*, 2014.
[17] M. Henze *et al.*, "Practical Data Compliance for Cloud Storage," in *IEEE IC2E*, 2017.
[18] A. Greenberg and K. Zetter, "How the Internet of Things Got Hacked," WIRED Security, 2015.
[19] L. Lamport, "Password Authentication with Insecure Communication," *Commun. ACM*, vol. 24, no. 11, 1981.
[20] P. Maniatis and M. Baker, "Secure History Preservation through Timeline Entanglement," in *USENIX Security*, 2002.
[21] E. Barker *et al.*, "Recommendation for Key Management – Part 1: General (Rev. 3)," NIST Special Publication 800-57, 2012.
[22] D. Thilakanathan *et al.*, "A Platform for Secure Monitoring and Sharing of Generic Health Data in the Cloud," *FGCS*, vol. 35, 2014.
[23] J. Liu *et al.*, "Secure sharing of Personal Health Records in cloud computing: Ciphertext-Policy Attribute-Based Signcryption," *FGCS*, vol. 52, 2015.
[24] M. Henze *et al.*, "SCSlib: Transparently Accessing Protected Sensor Data in the Cloud," in *AASNET*, 2014.
[25] P. Porambage *et al.*, "Group Key Establishment for Secure Multicasting in IoT-enabled Wireless Sensor Networks," in *IEEE LCN*, 2015.
[26] B. Schneier and J. Kelsey, "Secure Audit Logs to Support Computer Forensics," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 2, 1999.
[27] B. R. Waters *et al.*, "Building an Encrypted and Searchable Audit Log," in *NDSS*, 2004.
[28] R. Accorsi, "BBox: A Distributed Secure Log Architecture," in *EuroPKI*, 2010.
[29] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2009.
[30] M. Ali *et al.*, "Blockstack: Design and Implementation of a Global Naming System with Blockchains," Tech. Rep., 2016.
[31] G. Zyskind *et al.*, "Decentralizing Privacy: Using Blockchain to Protect Personal Data," in *IEEE SPW*, 2015.
[32] I. Eyal *et al.*, "Bitcoin-NG: A Scalable Blockchain Protocol," in *USENIX NSDI*, 2016.