This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Collaborative Validation of Public-Key Certificates for IoT by Distributed Caching

Minmei Wang, *Graduate Student Member, IEEE*, Chen Qian, *Senior Member, IEEE, Member, ACM*, Xin Li, *Member, IEEE*, Shouqian Shi, *Graduate Student Member, IEEE*, and Shigang Chen, *Fellow, IEEE*

***Abstract*— Public-key certificate validation is an important building block for various security protocols for IoT devices, such as secure channel establishment, handshaking, and verifying sensing data authenticity from cloud storage. However, certification validation incurs non-trivial overhead on resource-constrained IoT devices, because it either brings long latency or large cache space. This work proposes to utilize the power of distributed caching and explores the feasibility of using the cache spaces on all IoT devices as a large pool to store validated certificates. We design a Collaborative Certificate Validation (CCV) protocol including a memory-efficient and fast locator for certificate holders, a trust model to evaluate the trustworthiness of devices, and a protocol suite for dynamic update and certificate revocation. Evaluation results show that CCV only uses less than 25% validation time and reduces >90% decryption operations on each device, compared to a recent method. Malicious devices that conduct dishonest validations can be detected by the network using the proposed trust model.**

***Index Terms*— IoT security, certificate validation, collaboration.**

I. Introduction

IN RECENT years, Internet of Things (IoT) has attracted significant attention due to the emergence applications of industrial automation, smart devices, vehicular communication, smart cities, and smart homes [3], [29], [43]. A widely accepted definition of IoT for smart environment is that IoT is an interconnection of sensing and actuating devices that is capable of sharing information across platforms through a unified framework such as cloud [13]. Thus, a great amount of data, including both public and private information, will be generated, processed and transmitted by IoT devices.

Data authenticity for IoT devices is a critical issue. Many current IoT devices rely on a central platform to verify data authenticity [34], [36]. However, emerging and future IoT devices, such as personal health monitors, unmanned aerial vehicles, robots, and self-driving cars, become multi-functional, self-organized, and interactive. Hence due to scalability and autonomy problems, there may not be a central platform to interconnect all these devices. IoT devices may directly communicate with each other to share data. Public key cryptography (PKC) enables fundamental security protocols for IoT data communication, based on a well-functioning public key infrastructure (PKI). In fact, PKC can be widely used in IoT environment. We list the following (incomplete) important use cases of PKC for IoT. **1)** The authentication process in protocols for establishing a secure channel between two end devices or one device and a server. For example, in an IoT-based healthcare system, wearable sensors that collect human-related data need to securely communicate with other sensors, caregivers and doctors [26]. Existing approaches modify traditional end-to-end IP security protocols to adapt to IoT environments, such as DTLS [33] and HIP DEX [27], which rely on PKC for handshaking. **2)** When an IoT device retrieves sensing data that were collected by other sensors and stored in the cloud, it needs to verify the data integrity and authenticity to guarantee that data have not been tampered with or partially dropped [22]. Digital signatures of sensing data are applied for this situation. In order to verify the correctness of a data signed by the private key of the data generator, a device first needs to validate the public key via its certificate. **3)** Recently studied Blockchain-based IoT systems [8] heavily rely on PKC. **For all situations, certificate validation is an essential step.** Although certificate validation can be completed relatively easily on an ordinary computer, it incurs non-trivial overhead on resource-constraint IoT devices. For example, using an optimized method that requires only one signature verification, certificate validation still costs 1.9 seconds and certificate-based public key operations demand 95% of the overall processing time of handshaking on the WisMote platform [23], as reported in [14].

This work focuses on a specific yet important problem: how to perform **fast certificate validation** in a large IoT network. We do **not** intend to improve handshaking protocols, PKI, or PKC schemes in IoT. Instead we study the certificate validation method that is compatible to most existing PKIs and PKC algorithms. There has been existing work on reducing certificate validation cost in classic network environments. One method is to delegate certificate validation to a third party [14], [25], [26], which causes high overhead to the third party, making it be the computation bottleneck and a single point of failure. Some recent work aims at reducing the cost of checking certificate revocation status [24], [41], but provides no solution to reducing the signature verification in

Manuscript received October 14, 2019; revised August 2, 2020; accepted September 10, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor W. Lou. The work of Minmei Wang, Chen Qian, Xin Li, and Shouqian Shi was supported in part by the National Science Foundation under Grant 1750704 and Grant 1932447. The preliminary version of this article appeared at the IEEE International Conference on Computer Communications (INFOCOM), 2019. *(Corresponding author: Minmei Wang.)*

Minmei Wang, Chen Qian, Xin Li, and Shouqian Shi are with the Department of Computer Science and Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064 USA (e-mail: mwang107@ucsc.edu; cqian12@ucsc.edu; xinli@ucsc.edu; sshi27@ucsc.edu).

Shigang Chen is with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: sgchen@cise.ufl.edu).

Digital Object Identifier 10.1109/TNET.2020.3029135

1063-6692 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

certificate validation. Prefetching and prevalidation are also used for efficient certificate validation [15], [37], but they bring heavy storage cost. However, IoT devices are often deployed with limited memory.

Fast certificate validation on IoT devices seems to be a dilemma: the most effective approach is to cache as many validated certificates as possible, but it is not allowed on IoT devices with limited memory. We call the process of validating a public-key certificate via verification of CA signature as *individual validation*. Individual validation of every certificate is time-consuming. Hence none of the above methods is desired for IoT.

In this work we propose to utilize the power of distributed caching and explore the feasibility of using the cache spaces on all IoT devices as a large pool to store validated certificates, which can be accessed by any internal device. We design a **C**ollaborative **C**ertificate **V**alidation protocol (CCV), which adopts the cooperation strategy in a large IoT network and utilizes the overall computation power and storage resources. When one device d needs to validate a certificate that has been validated and cached by another device h in the network, d can request a collaborative certificate validation from h to confirm that the requested certificate matches the cached one. The design of CCV includes **three main challenges**. First, how each device can efficiently locate the holder of a certificate without storing a long index that maps every certificate to its holder. Second, how to avoid false validation results shared by the IoT devices controlled by the attacker (called malicious devices). Third, how to dynamically maintain a consistent collaborative validation when new certificates are validated and cached certificates are removed or revoked.

Our contributions of this work include the following. 1) We design a memory-efficient and fast locator for certificate holders, called OLoc, based on a recent data structure Othello Hashing [42]. 2) We introduce a trust model for CCV to evaluate the trustworthiness of each device to avoid dishonest collaborative validation from malicious devices. 3) We design a complete protocol suite for efficient OLoc update, cache replacement, and revocation status checking mechanisms in a dynamic network. Evaluation results show that CCV only uses less than 25% time compared to the certificate validation in a recent method [14]. The majority time cost of CCV is on network latency rather than local public key decryptions (reducing $> 90\%$ decryptions), hence it significantly saves computation resource.

The paper is structured as follows. We give the problem statement, network model and security model in Section II. Section III presents the design consideration of the certificate locator. We present the detailed protocol design in Section IV. We show the evaluation results and security analysis in Section V. Section VI presents the related work. We discuss the false validation problems in Section VII and Section VIII concludes this work.

II. Problem Statement and Models

A. Problem Specification and Network Model

The IoT system discussed in the paper, as depicted in Fig. 1. We consider a large IoT system including a large number (100 or more) of devices. The use cases of such system can be

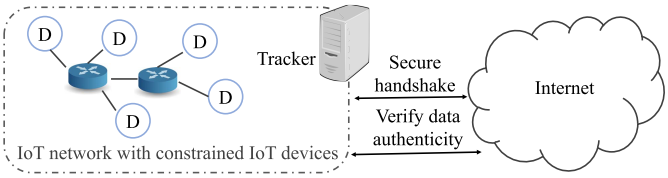


Fig. 1. Network model.

an industrial IoT control network [19], a community network with home IoT devices, or an organization/building/campus network with various IoT devices (cameras, sensors, smart office products, etc.). The system consists of the following units.

1) IoT Devices: An IoT device (or "device" in short) is a sensor or actuator with constrained computing, memory, and power resources. Each device can communicate to the Internet through the routers in the IoT system. A device sends and receives packets to/from a router using its wireless chip via either direct connection to a router ("infrastructure mode", such as those in a home WiFi network) or multi-hop forwarding ("ad-hoc mode", such as those in a low-power sensor network). Devices can communicate with each other.

2) Routers: Routers are the forwarding units to support communication among IoT devices, or between a device and the Internet.

3) Tracker: A tracker is a function running on a remote or edge server to help managing the IoT network. All IoT devices can communicate with the tracker. The tracker does **not** actually perform validation or caching for devices. **A tracker in CCV is not a single point of failure.** Our experiments will show that a tracker requires minimal computation, storage, and communication cost. Hence it can be easily replicated. In CCV, we adopt two trackers: the primary tracker and the secondary tracker. The primary tracker periodically sends the latest stored data required in CCV to the secondary tracker for backup. When the primary tracker stops functioning, CCV will use the secondary tracker. Even if the two trackers both stop working for a duration of time, IoT devices can still perform certificate validation – though not optimal.

This work focuses on the public key certificate validation problem of an IoT system. Each certificate is *uniquely identified* by its public key. We assume secure communication channels have already been established among the IoT devices and the tracker in the same local network using standard IoT security solutions such as that in WirelessHART [19]. Hence a device does *not* need to validate public keys of other devices and the tracker in the same network. A device needs to validate a public key certificate of an external node from the Internet in the following situations:

(1) Authenticate an external node during the handshaking to establish a secure session, such as that in DTLS [14], [33].

(2) Verify the authenticity of the data retrieved from a cloud, which carry the digital signatures of external nodes [22].

The *collaborative certificate validation scheme* investigated in the paper can be modeled as follows. When a device receives a public key certificate that has already been verified and cached by another device in the network (called the holder), the device needs to locate the cached certificate and ask the holder to confirm it. Otherwise it needs to run individual validation. Each device caches a (limited) number

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

of certificates validated by itself. When a device receives a request from another device in the network to validate a certificate, it will response based on the result from its cache.

B. Security Model

We assume the internal communication among IoT devices or between a device and the tracker is secure. The secure communication channels have been established using standard solutions such as the security protocol of WirelessHART [19]. Group keys and session keys have been successfully distributed. We do not consider attacks on the communications between two IoT devices or a device and the tracker. All devices, except those controlled by an attacker, are willing to collaborate. The goal of a device is to maximize the functioning of the entire network rather than maximizing the functioning or lifetime of itself. We assume the tracker is trusted.

This work focuses on the research problem of efficient validation of public key certificates, assuming there exists a well-functioning PKI. This research is not about building a better PKI. Hence we do not consider attacks during the PKI validation process.

An attacker can control a number of devices in the network to conduct malicious behaviors, which are referred as *malicious devices*. Malicious devices are "malicious-but-cautious" and may collude. The tracker stores a *trust value* for every device to indicate the likelihood that the device is legitimate. Malicious devices may conduct the following attacks.

(1) **False validation attack:** A malicious device provides false certificate validation results to other devices.

(2) **Self-promoting attack:** A malicious device promotes its trust value by claiming that it helped other devices validate certificates. However, it did not.

(3) **Defamation attack:** A malicious device claims that a legitimate device provides wrong certificate validation results.

(4) **Traitor attack:** When a diplomatic attacker senses their reputation is dropping because of providing malicious devices, it can provide good services for a period of time to gain a high reputation. Then it provides malicious services after it gains high reputation.

(5) **Whitewashing attack:** Attackers can discard their current identities and re-enter the systems when they have very low trust levels and cannot be selected as collaborators.

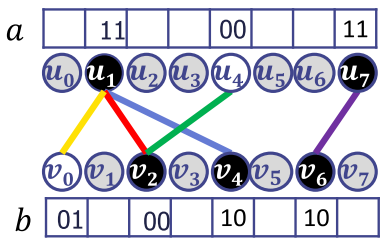
(6) **Collusion attack:** Two or more malicious devices improve their trust values by claiming that they helped each other, However, they provide false validation results when helping other devices to validate certificates.

(7) **Sybil attack:** A malicious party creates many fake identities to enter the system. The malicious party can destroy the CCV protocol because fake identities cannot cache certificates. Or the malicious party can conduct further attacks such as the false validation attack based on fake identities.

In addition, a device never individually validates a certificate that is not required by its own need, in order to avoid DoS or resource exhaustion attacks. It only caches a certificate validated by itself in prior communications and provides validation confirmation of this certificate to another device.

III. Design Consideration of Certificate Locator

One major challenge of collaborative certificate validation is to allow each device efficiently locate another device that has validated and cached the certificate of the public key to use. As the tracker knows the cached certificates of each device, the problem is how to design a structure to store all certificate-to-device information for each device. A simple solution is to let each device maintain a complete index of all certificate-to-device mappings. This method is not scalable because every mapping requires more than 1000 bits of memory, assuming a public key is 1024-bit long. A more advanced method is that each device maintains $m-1$ counting Bloom filters [10] (m is the number of devices in the network). Each Bloom filter represents the set of certificates of a device. The drawback of this method is that locating the holder of a certificate requires up to $m-1$ Bloom filter lookup operations, which is time-consuming especially on IoT devices. Hence they are both impractical for IoT.



k	*value*	$h_a(k)$	$h_b(k)$
k_0	01	7	6
k_1	10	1	0
k_2	11	1	2
k_3	01	1	4
k_4	00	4	2

Fig. 2. Example of othello hashing.

In this work we utilize and improve a recent innovation called Othello Hashing [42] to design a memory-efficient and fast locator for certificate holders. In addition, existing design of Othello Hashing does not fully satisfy the requirement of the locator hence we propose an improvement design called Othello-based Locator (OLoc). Every device stores an OLoc.

Othello Hashing is used to represent a set of key-value pairs. Given a set of keys K and each key k is mapped to a value $v \in V$. Let $n = |K|$. An Othello Hashing structure is a seven-tuple $< m_a, m_b, h_a, h_b, \mathbf{a}, \mathbf{b}, G >$ defined as follows.

- Integers m_a and m_b is the size of Othello. $m_a \approx 1.33n$, $m_b \approx n$.
- A pair of uniform random hash functions $< h_a, h_b >$ maps keys to integer values $0, 1, \ldots, m_a - 1$ and $0, 1, \ldots, m_b - 1$ respectively.
- $\mathbf{a}$ and $\mathbf{b}$ are two arrays including m_a and m_b elements respectively.
- G is a bipartite graph which is used to determine the values in $\mathbf{a}$ and $\mathbf{b}$.

Fig. 2 shows an example of Othello Hashing of 5 keys k_0 to k_4. Each key has a corresponding 2-bit value. The bipartite graph G consists of two groups of vertices us and vs. We use a pair of uniform hash functions h_a and h_b. For each key k, we will place an edge in G based on $h_a(k)$ and $h_b(k)$. For example, $h_a(k_0) = 7$ and $h_b(k_0) = 6$. Then an edge is placed to connect u_7 and v_6. The constructed bipartite graph G needs to be acyclic. If there is a cycle, Othello needs to choose another pair of uniform hash functions to re-build G. The possibility to re-build G is small ($\Theta(1/n)$) and it has proved that the cost to build Othello for n keys is $O(n)$ [42]. Once such bipartite graph G is built, the elements of $\mathbf{a}$ and $\mathbf{b}$ can be trivially assigned proper values such that $\mathbf{a}(h_a(k)) \oplus \mathbf{b}(h_b(k))$ is the value of k.

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

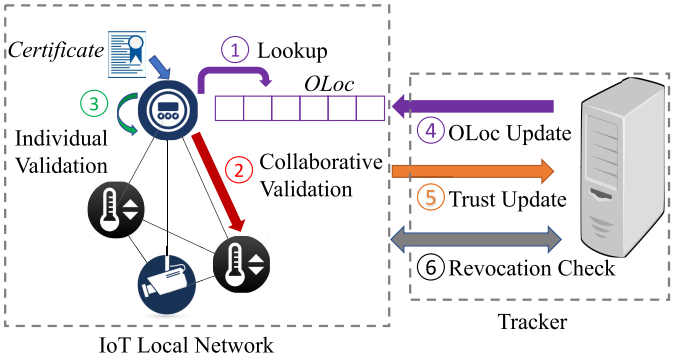


Fig. 3. Protocol overview of CCV.

After the arrays **a** and **b** being built, finding the value of a given key k is extremely fast. Othello can simply retrieve $\mathbf{a}(h_a(k))$ and $\mathbf{b}(h_b(k))$ and compute their XOR, requiring only two memory access operations.

Complexity: The space cost of the two arrays in Othello is small, around $2.33nl$ bits, where l is the length of each value. Each lookup only requires two memory access and one XOR operations –very small constant. It has also proved that the expected time to add, delete and update a key-value pair is $O(1)$ [42].

Opportunities and Challenges of Using Othello: We find that Othello Hashing is a good fit for the application of memory-efficient certificate locator. Let each key be a public key and the corresponding value be the holder of the certificate. We realize that, to perform locator lookups, only the two hash functions $< h_a, h_b >$ and arrays **a** and **b** need to be stored in a device. The construction information, such as the key-value pairs and the bipartite graph G are shared by the entire network and not needed for lookups. Hence these information can be stored at the tracker. Note the Othello construction and update operations are relatively more complex than lookups. Hence the IoT devices can avoid these operations and only be responsible for efficient lookups. The tracker has plenty of resources for construction and is responsible for updating and sending the updated Othello arrays to the devices.

However one limitation of Othello Hashing is that, if we search a key k' that is not in K, Othello will return an arbitrary value. It is because $\mathbf{a}(h_a(k'))$ and $\mathbf{b}(h_b(k'))$ will be two arbitrary elements. Hence if a certificate C is not cached by any device in the network, the locator will point to an arbitrary device. Falsely locating a holder will waste both communication bandwidth and latency. In the next section we will present an improved design of an *Othello-based Locator* (OLoc) to reduce the rate of false holder locating.

IV. Protocol Design

A. Protocol Overview

Fig. 3 illustrates the overview of the proposed protocol CCV. The whole system contains many IoT devices and a working tracker. The CCV protocol runs on both the IoT devices and the tracker.

Protocol on an IoT Device d: When a device d needs to validate a certificate C, CCV works as follows.

Part 1: d searches the Othello-based Locator (OLoc) to look for a holder of C.

Part 2: If the OLoc indicates that there is a holder h of C, devices d and h conduct collaborative validation based on the cached certificate on h.

Part 3: If the OLoc indicates that there is no holder of C, d runs individual validation.

Part 5: If a holder h confirms the validation of C, d will forward this event to the tracker with a probability to allow the tracker to monitor the trustworthiness of h.

Protocol on the Tracker: The tracker is responsible for updating the OLoc of different devices, monitoring the trust values, and removing revoked certificates. The protocol on the tracker operates as follows:

Part 4: Since the certificates cached in the network change gradually, the tracker needs to update the OLoc for each device to keep track of the update-to-date holder information. It then distributes the updated OLoc to each device.

Part 5: When the tracker receives a forwarded validation confirmation showing that h just helped d, it will verify the correctness of this confirmation and update h's trust value accordingly.

Part 6: When the tracker receives new revocation lists from CAs, it notifies the holders to remove these certificates from their caches.

B. Bootstrap Phase and Devices Management

Each IoT device or the tracker is associate with a digital certificate that is issued by a certificate authority (CA). Two steps are required in the bootstrap phase. 1) Registration. Each device is registered on the tracker using its certificate. The tracker then audits the identity of the device to filter fake identities. One way is to put physical device information in its certificate [39]. Or CA charges money for the certificate to prevent attackers from getting enough identities for the Sybil attack [5]. Legitimate devices will be added into the device list. 2) Keys generation. Communication channels are built and session keys are established between each two IoT devices and between the tracker and a device. All two entities can create a session key to build a secure channel through the certificate-based key agreement protocol or other efficient key-generation protocols, such as [30].

The system is dynamic with IoT devices join and leave at any time. If a device joins the system, the device follows the two steps in the bootstrap phase. If a device leaves the system, the tracker will delete it by deleting the device in the device list and not choosing it as collaborators in the OLoc. Then the tracker sends messages to inform all the other devices to delete the session key with this device.

C. Definition of Messages

There are many types of messages exchanged among devices and the tracker in CCV. The CCV protocol is driven by messages and events. We define the following types of messages before we describe details of CCV. Each message is defined by a tuple where the first element is the message type. All messages are exchanged above the secure communication channels inside the network.

- $<$ *REQUEST_VALI*$, C, u, v$ $>$: Device u sends this message to request device v to validate the certificate C.

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

- $<$ *REPLY_VALI*, $C, u, v, r >$: Device u sends this message to reply device v the certificate validation result r.
- $<$ *REPLY_NO_CERT*, $C, u, v >$: Device u sends this message to reply device v that it does not cache the certificate C.
- $<$ *UPDATE_TRUST*, $u, t, E >$: Device u sends this message to tell the tracker t to update the trust value according to the collaborative validation events E. E includes the information of the prior *REPLY_VALI* messages.
- $<$ *FALSE_VALI_RESULTS*, $u, t, F >$: The tracker t sends the message to tell the device u that the prior validation confirmation F is false.
- $<$ *NEW_CACHE*, $C, u, t >$: Device u sends this message to tell the tracker that it caches the certificate C.
- $<$ *DELETE*, $C, u, t >$: Device u sends this message to tell the tracker that it no longer caches the certificate C.
- $<$ *UPDATE_OLOC*, $O, u, t >$: The tracker sends this message to device u to request u to update its OLoc to O.
- $<$ *REVO_CERT*, $C, u, t >$: The tracker sends this message to device u to remove revoked certificate C.
- $<$ *CHECK_REVO*, $C, u, t >$: Device u sends this message to the tracker to check whether the certificate C is revoked or not.
- $<$ *REPLY_REVO*, $C, u, t, r >$: The tracker replies device d the result r of the revocation status of certificate C.

D. Othello-Based Locator and Update

This subsection describes Parts 1 and 4 presented in Sec. IV-A and Fig. 3.

Upon receiving a validation requirement of a certificate C of a public key k^+ from the upper layer, a device d first checks its local cache to see whether it has cached C. If C is cached and the two versions are identical, then d confirms the validity of C. Otherwise, d needs to determine whether there is another device being the holder of C and which device it is.

Every device stores an Othello-based Locator (OLoc). The lookup key of OLoc is a public key (identifier of a certificate) and the lookup result should indicate the holder of the certificate. As discussed in Sec. III, one limitation of Othello Hashing is that, if a certificate C is not cached by any device in the network, the locator will point to an arbitrary device. Assume the network has n devices and each device can be referred by a l-bit index: $l = \lceil \log_2 n \rceil$. Our innovation is to extend the lookup value τ of an Othello to $l + l'$ bits for the certificate C of the public key k^+. The l least significant bits (LSBs) of τ is the index of the holder i and the l' most significant bits (MSBs) is the check code c of this certificate. c is determined by the hash value $H(k^+)$ using a CRC hash function H. Since $H(k^+)$ is longer than l' bits,c can simply be the l' LSBs of $H(k^+)$.

Fig. 4 illustrates an example of the OLoc lookup. When a device searches its OLoc for the holder of the certificate C of k^+, it compares whether the l' LSBs of $H(k^+)$ matches the check code c return by OLoc. If they match, it is highly likely that the certificate is actually cached by the holder. By "highly likely", we mean that there is still a probability that C is not cached but matches the check code, called a *false matching*. Such probability is around $1/2^{l'}$ depending

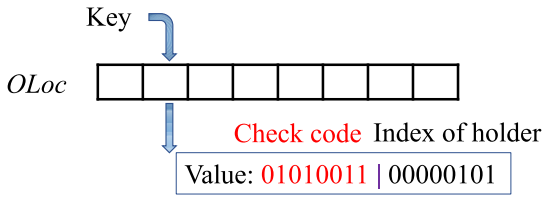


Fig. 4. OLoc lookup in CCV.

on the length of the check code l'. Longer check code causes lower false matching rate. The existence of false matchings does not hurt the correctness and security of CCV, but will slightly increase the communication cost. In the example of Fig. 4, both l and l' are set to 8, which can be adjusted based on the system requirements.

When the device d gets index i and the check code matches $H(k^+)$, d sends message $<$ *REQUEST_VALI*, $C, d, h >$ to another device h whose index is i to perform collaborative validation. If the check code does not match $H(k^+)$, the device terminates CCV and conducts individual validation.

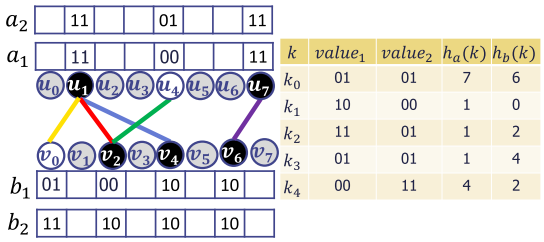
On the tracker side, the OLoc at every device should be dynamically updated to reflect the update-to-date certificate to holder mapping. At the very beginning, Othello is empty. Then the tracker updates the OLoc of all devices at a fixed interval and distributes newly updated OLoc to the devices. Although the tracker is responsible for updating all devices in the network, *these updates are efficient and scalable because all devices may share a same OLoc*. When a device caches a certificate, it sends a *NEW_CACHE* message to notify the tracker. Hence the tracker keeps track of all cached certificates in the network and updates the OLoc. The updated OLoc is then sent to the devices using the *UPDATE_OLOC* message.

We apply another optimization based on the IoT network features. It is possible that one certificate C is cached by multiple devices in the network. Hence C may have multiple holders, any of which can be a valid result of a holder locator. In the construction stage of Othello, we choose the index of one holder to be the lookup result of OLoc for the public key k^+ in C. However, it is reasonable to choose the most suitable holder of C for different devices when there are multiple feasible options. To construct the OLoc of a device d, CCV may choose the holder with the shortest network distance (e.g., smallest hop count) to d. One may note that using this optimization, the OLoc on different devices may be different. Two devices on different locations in the network may be close to different holders. However, constructing these different versions of OLoc is still efficient. They share the same set of keys and the same bipartite graph G, because G depends only on the set of keys, rather than their values. Note that computing G is the most time-complex step during the construction of an Othello. Once G is obtained, determining the arrays **a** and **b** is trivial. Hence all devices can still share a same G and the arrays **a** and **b** can be computed in a short time. An example of two different OLoc sharing a same G in shown in Fig. 5. In addition, many devices in network proximity are still able to use a same OLoc.

In addition, the trust values of IoT devices maintained by the tracker are used to filter malicious devices. Hence the tracker will only select the holders whose values are above a

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



k	$value_1$	$value_2$	$h_a(k)$	$h_b(k)$
k_0	01	01	7	6
k_1	10	00	1	0
k_2	11	01	1	2
k_3	01	01	1	4
k_4	00	11	4	2

Fig. 5. Multiple OLoc construction.

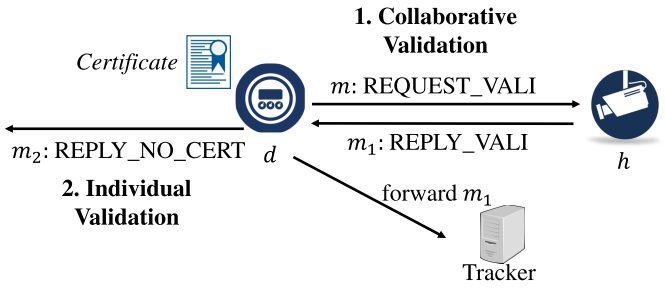


Fig. 6. Collaborative validation process.

pre-determined threshold θ ($\theta \geqslant 0.5$). The tracker updates trust values periodically.

E. Collaborative Validation

To request a collaborative validation of certificate C, device d sends a message $< \textit{REQUEST\_VALI}, C, d, h >$ to the holder h. Upon receiving this message, the holder h searches its cache to find the certificate of the public key k^+ on C. If such certificate exists and is identical to C, it replies d with a message $< \textit{REPLY\_VALI}, C, h, d, `Correct' >$. If the certificate exists but is different from $C$, it replies $d$ with a message $< \textit{REPLY\_VALI}, C, h, d, `Wrong' >$. If no certificate of k^+ is cached, it replies d with a message $< \textit{REPLY\_NO\_CERT}, C, h, d >$. The main reason of a missing certificate is that previously cached certificates may be replaced by others due to the lack of cache space, while this information has not been updated to OLoc. The other reason is the false matchings. Note all these messages should be signed by h's private key for non-repudiation purposes. To avoid malicious devices send lots of *REQUEST_VALI* messages to exhaust the resource, the holder can record the number of request messages from other devices during a time interval. Once the holder detects an unusually larger number of request messages from a device, it reports this abnormal behavior to the tracker. The tracker can then remove this malicious device from the system.

Once the device d receives the *REPLY_VALI* message with a 'Correct' value from a holder h, it knows the validation of certificate C is confirmed and it can use C for incoming communications. If d receives the *REPLY_VALI* message with a 'Wrong' value. It will discard the certificate C and still forward this event to the tracker. If d receives the *REPLY_NO_CERT* message, it runs individual validation. Fig. 6 shows the collaborative validation process.

One additional step is that d may forward the confirmation events to the tracker, in order to improve the trust value of the holders that provide validation. It sends a message $< \textit{UPDATE\_TRUST}, d, t, E >$, where E includes one or more *REPLY_VALI* messages received during the past period of time as well as their digital signatures. In order to save message cost and reduce tracker overhead, d does *not* forward every collaborative validation event but on a sampling basis. The intuition of using sampling is that when a malicious device keeps providing false validation results, then statistically it will be detected. The sampling rate can be dynamically adjusted. For example, the sampling rate can be higher at the beginning to quickly filter malicious devices. When the system is stable, the sampling rate is set to be lower to reduce communication cost and the computation overhead of the tracker.

F. Individual Validation and Caching

This subsection describes Part 3 of CCV.

If the device d chooses to run individual validation of certificate C, this process consumes computation resource and relatively long latency on d. After C being validated, d will cache C in its local memory. One of the following three cache replacement strategies will be used to replace old certificate: random, FIFO (first in, first out), and LRU (Least recently used).

If an old certificate C_o is replaced by a new one, d sends a message $< \textit{DELETE}, C_o, u, t >$ to the tracker. The updates of cached certificates will be sent to the tracker immediately to make the tracker knows the up-to-date certificate-to-device information.

Besides validating the certificate, d also needs to check the revocation status of the certificate. It sends $< \textit{CHECK\_REVO}, C, d, t >$ to the tracker to query the revocation status. If the certificate is included in the revocation list stored in the tracker. The tracker will send a $< \textit{REVO\_CERT}, C, d, t >$ message to call back C and let d stop using C and remove it from the cache.

G. Trust Model and Updates

As discussed in the security model, the malicious devices may conduct seven types of attacks to the whole system. In order to facilitate the detection of malicious devices and make them have lower probability to be the selected holder, it is necessary and essential for CCV to introduce a trust model in the protocol. In this section, we first introduce the trust model and then describe the trust update protocol of CCV (Part 5).

1) Trust Model: In CCV, we adopt the following definition of trust from [6].

Definition 1: In the IoT network, a device d's trust to another device d' is the subjective expectation of d of receiving positive outcomes through the communications with d'.

Specifically, the trust value in CCV quantifies the expectation to receive correct validation results of certificates. The range of trust value between two devices is $[0, 1]$. At the beginning, the trust value between each two devices is set to be 0.5.

In general, trust can be categorized into two classes: direct trust and indirect trust. **Direct trust** is the trust that is calculated by direct communications between two devices. **Indirect trust** is the trust that is calculated by indirect recommendations, which will be explained later.

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

We consider direct and indirect trust and use past collaboration behaviors between every two devices to measure the trust value. Direct trust is based on the certificate validation results between a validation requester and the holder. The tracker maintains two arrays to record the number of honest validation events s and the number of dishonest validation events f between a requester d and a holder d' during the trust update interval, respectively. Note that every device forwards collaborative validation events to the tracker at a sampling rate. Then the tracker will audit whether the holder honestly validated the certificates or not. Once a validation result is audited, the tracker will update the corresponding number in the arrays. With the information of s and f for any two cooperative devices, we adopt a subjective logic framework [17], [18] in the binary domain to compute the corresponding direct trust value.

Let $\mathbb{X} = \{x, \bar{x}\}$ be a binary domain with binomial random variable $X \in \mathbb{X}$. In our trust model, x represents that device d trusts the device d'. In the subjective logic framework, a binomial opinion about a state value x is the ordered quadruplet $w_x = (b, d, u, a)$, where $b + d + u = 1$ and $b, d, u \in [0, 1]$. b represents the belief mass in support of x being true, d represents the disbelief mass in support of x being false, and u is the uncertainty mass representing the vacuity of evidence. a is the base rate which represents the prior probability of x without any evidence. a is set to be 0.5. The probability of a binomial opinion about value x is

$$p = b + au \tag{1}$$

It can be shown that posteriori probabilities of binary events can be represented by the beta distribution $Beta(\alpha, \beta)$ [17]. α and β parameters can be expressed as a function of the observations (s, f) and the base rate a.

$$\alpha = s + 2a, \quad where\ 0 < a < 1, s \geq 0$$
$$\beta = f + 2(1 - a), \quad where\ 0 < a < 1, f \geq 0$$

Then the beta distribution can be expressed by

$$f(p \mid \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha - 1}(1 - p)^{\beta - 1}$$

The expectation probability of value x is

$$E(x) = \frac{\alpha}{\alpha + \beta} = \frac{s}{s + f + 2} + \frac{2}{s + f + 2} a \tag{2}$$

Combining formula (1) and (2), b, d, u are calculated by the following formula.

$$b = \frac{s}{s + f + 2}, d = \frac{f}{s + f + 2}, u = \frac{2}{s + f + 2}$$

One consideration is that more punishment need to be given when there exist dishonest validation events. Thus the trust value of malicious device can be sharply decreased when the tracker detects its dishonest validations. We add γ parameter to adjust the influence of dishonest validation on the trust value. Therefore, b, d, u is are finally calculate by the following formula.

$$b = \frac{s}{s + \gamma f + 2}, e = \frac{\gamma f}{s + \gamma f + 2}, u = \frac{2}{s + \gamma f + 2}$$

Direct trust value is $b + au = \frac{s+1}{s+\gamma f+2}$. Larger γ causes lower trust value with dishonest validation results.

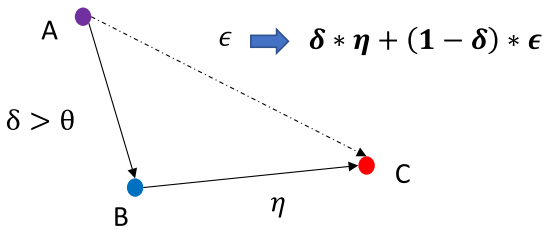


Fig. 7. Calculation of indirect trust.

Indirect trust is based on the recommendation. A device can aggregate trust recommendations from its trusted devices. The trust value between two devices is used as the measurement for choosing recommenders. Then threshold θ is introduced to select recommenders. Fig. 7 shows how to calculate indirect trust. Device A trusts B with a direct trust value δ and device A trusts C with a direct trust value ϵ. When there is an update between device B and C, and the new trust value is η. If the trust value from A to B exceeds the threshold θ, B can recommend C to A with trust value η. The updated value is $\delta\eta + (1 - \delta)\epsilon$. Hence, A trusts C with the updated trust value with this recommendation scheme. We set θ to 0.5 in our experiments.

2) Trust Updates: When the tracker receives an *UPDATE_TRUST* message, it verifies the collaborative validation events in E. The tracker maintains two arrays A and B to store the numbers of honest and dishonest collaborative validation events between two devices during the trust update interval respectively. $A[i][j]$ denotes the number of honest validations that j provides to i, and $B[i][j]$ denotes the number of dishonest validations that j provides to i. The tracker also stores the trust value $T[i][j]$ at current time which denotes the degree that i trusts j. The update algorithm is shown as Algorithm 1.

Algorithm 1 Trust Value Update Algorithm

Input: A, B, θ
1: **for** Every event of holder v helping device u **do**
2: Compute direct trust t_d;
3: Update $T[u][v] = pT[u][v] + qt_d$;
4: **for** Every device i which has $T[i][u] > \theta$ **do**
5: Compute indirect trust $t_i = T[i][u] * T[u][v] + (1 - T[i][u]) * T[i][v]$;
6: Update $T[i][v] = pT[i][v] + qt_i$;
7: **end for**
8: **end for**

For every computed direct or indirect trust value of device u for another holder device v, the tracker calculates the moving average combining the historical trust value $T[u][v]$ and the newly computed trust value t_{uv}: $T[u][v] = pT[u][v] + qt_{uv}$, where $p, q \in (0, 1)$ and $p + q = 1$. p and q denotes the weight of historical trust values and newly updated trust values respectively. We set p to be 0.4, which gives more weight to the updated trust value for the device.

In our model, assume the number of honest validation events of device u and the holder device v during the update interval is s. The number of dishonest validation events is f. When $f \geqslant 1$, for $\gamma > \frac{s}{f}$, the newly computed trust value

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

$\frac{s+1}{s+\gamma f+2} < \frac{s+1}{2s+2} < 0.5$. Larger γ causes a higher decrease. Therefore, our model can make the trust value dramatically decrease below 0.5 to detect the malicious device by setting a larger γ when the false validation behaviors are detected by the tracker.

If the tracker finds that a validation is dishonest and the certificate is not valid, it sends a *FALSE_VALI_RESULTS* message to tell the device.

H. Revocation Check

Revocation status check is important in certificate validation but time-consuming on devices. In CCV, the tracker actively downloads the Certificate Revocation List (CRL) [21]. A device can send a *CHECK_REVO* message to request the revocation status of a certificate. This design makes the device start using the certificate simultaneously while waiting for the reply from the tracker. The tracker replies about the status using a *REPLY_REVO* message. If the certificate is revoked according to the tracker, the device stops using it, removes it from the cache, and rolls back to the prior state. In addition, when the tracker updates its local revocation list and finds existing certificates cached in the network are expired, it will send *REVO_CERT* messages to the holders of these certificates for removing them. If all the trackers stop functioning, the device uses Online Certificate Status protocol (OCSP) [28] to query the CA's OCSP server about the revocation status of the certificate.

V. Simulation Results and Security Analysis

We implement a complete version of CCV in a packet-level discrete-event simulator running on a desktop with 3.6GHz Intel(R) Core(TM) i7-7700 CPU. To better simulate the performance of CCV for IoT devices, the actual processes of cryptographic operations are implemented and the latencies, including cryptographic latency and network latency, are simulated. The reason is that the cryptographic latency on a desktop does not reflect the actual cryptographic overhead on an IoT device. Hence in the simulator, we use the latency data of cryptographic operations gathered from a WisMote platform featuring a 16MHz MSP430 micro-controller [14], [23]. We use SHA256 for hash operations, elliptic curve NIST P-256 for PKC, and AES-128 for symmetric-key operations in secure communications among in-network IoT devices. We compare CCV to individual certificate validation [14], in which validating the certificate chain only requires one single decryption operation. The average time of such individual certificate validation on WisMote is around 1.9sec with 13.9ms standard deviation as reported in [14]. Note most existing certificate validation methods typically require multiple intermediate certificates in a chain, and the validation overhead grows linearly with the number of intermediate certificates [14]. For the simulated network latency, every two IoT devices are connected by one or more hops of routers. We define one hop latency in our simulation to be 20 ms [31], and the maximum hop count in the network is set to be 5.

In our experiments, we simulate a number of IoT devices running the CCV protocol to collaboratively validate the certificates. We use 'I-Valid' to refer to individual validation. **For fair comparison, in CCV we assume at the system start time, no certificate is validated and cached in the network. OLoc will use the cache space.** Each device in CCV also stores session keys between each other devices and the tracker. Every certificate in CCV must be individually validated once and cached for further use. The tracker updates the OLoc every five seconds.

We evaluate and compare the following six metrics of CCV.

1) **Latency** is the average time from receiving a certificate to finishing validation on a device.

2) **Number of local decryptions** is the number of times of running public key decryption to validate certificates. It characterizes the computation overhead on devices.

3) **Average number of messages per certificate** evaluates the communication cost. The number of messages for the certificate validated through caching by its received device is 0. When the received device individually validate the certificate after not finding the helpers, the number of message is 1, because *NEW_CACHE* message will be generated and sent to the tracker. When a certificate is successfully validated by the helper, the number of messages is 2. But when the helper has not validated the certificate and then the certificate is validated by its received device, the number of message is 3.

4) **Throughput** is the maximum number of certificate validations on the simulator. Although the computation resource on the simulator is different from that on a device, this metric still reflects whether CCV reduces resource overhead.

5) **Computational cost for OLoc update** measures the overhead of the tracker.

6) **Trust value changes** are used to detect malicious devices.

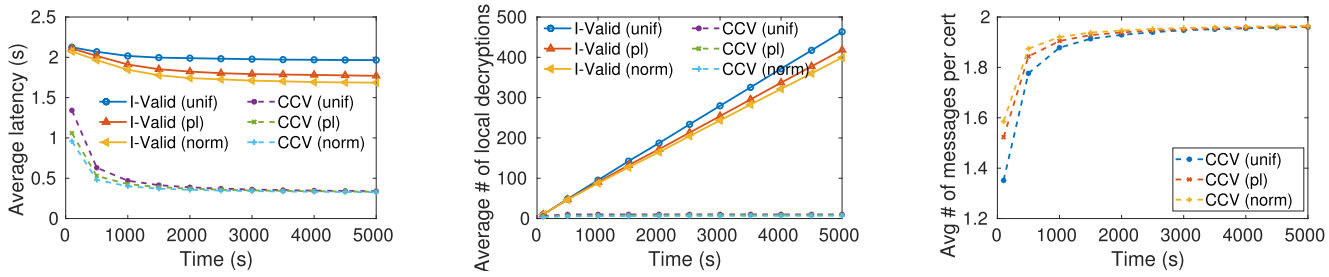
The number of events that require certificate validations happening on devices follows the Poisson distribution. The parameter λ denotes the average number of events happening in one second, which is used to adjust the frequency of events. Note the number of events that requires a particular certificate may not follow a uniform distribution. For example, some popular data or Internet server may be accessed by many devices. Hence we vary this distribution in three types: namely uniform, normal, and power law distributions.

A. Evaluation Results

Performance Varies With Time: Assuming at the system start time, no certificate is validated and cached in the network. Then the devices validate and cache certificates gradually. We may expect that the validation latency of CCV will decrease when time increases. Fig. 8 shows the performance comparison of CCV and I-Valid, by varying the time. In this set of experiments, the number of IoT devices is 100, the total number of certificates is 1000, λ is set to be 10, and the memory size (OLoc, symmetric keys and cached certificates) is set to be 32 KB. I-Valid also uses cache space to store certificates. We show the results for uniform, normal, and power law distributions. From Fig. 8(a), we find that the average latency to validate a certificate for CCV decreases during time 0s to 2000s and then remains stable after 2000s due to collaborative validations. CCV only uses 25% time compared to I-Valid. More importantly, Fig. 8(b) shows that CCV always requires around 10 decryptions per device, while this number can be > 400 for I-Valid at time 5000s. CCV reduces more than 99% local decryption operations and significantly saves the computation cost. The latency of CCV

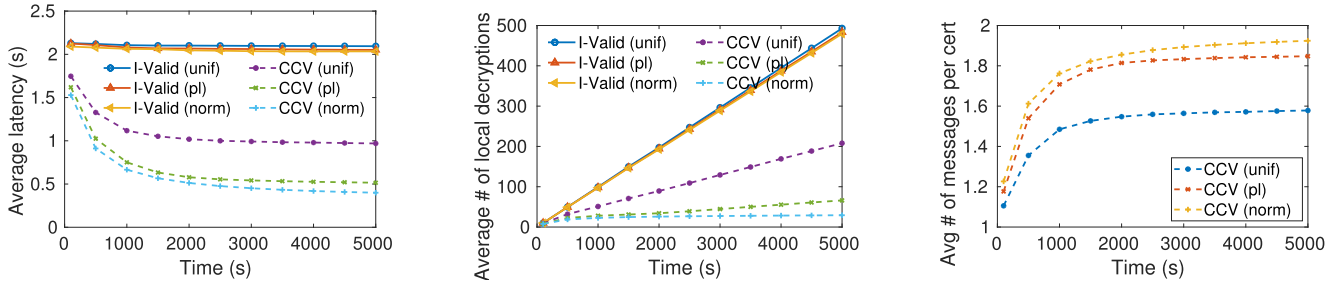
Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Validation latency (b) Number of local decryptions (c) Average # of messages per cert

Fig. 8. Performance varies with time.

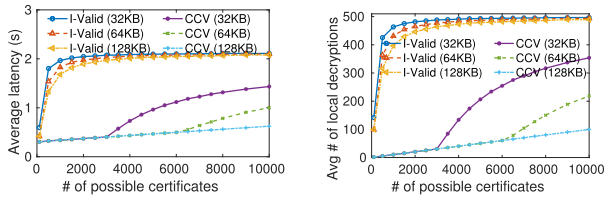


(a) Validation latency (b) Number of local decryptions (c) Average # of messages per cert

Fig. 9. Performance varies with different distributions.

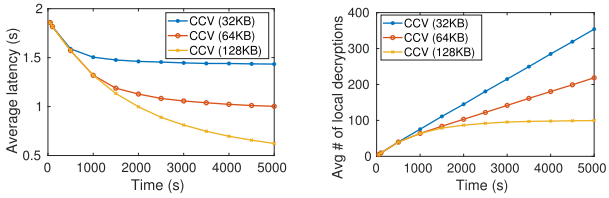
is mainly the network latency. From Fig. 8(c), we can see that the average number of messages for each certificate in CCV increases during time 0s to 2000s, but will be stable after 2000s. It is because more collaborative validation will be used than individual validation. CCV is very communication-efficient: the number of messages per certificate is close to two. The two messages are *REQUEST_VALI* and *REPLY_VALI* message respectively. We also find that different distributions have relatively small influence on the performance of CCV in this experiment, because 100 devices with 32kb cache size can cache almost all the 1000 certificates. In fact, power law and normal distributions are more desired for I-Valid protocol because it prefers a small part of certificates which be used for many times, causing high cache hit rates. We then conduct an experiment by changing the number of possible certificates to be 5000, other settings are the same. Fig. 9 shows the results. We can find that power law and normal distributions achieve better performance than uniform distribution when it is hard for cache pool to cache all the certificates. In the following experiments, we use uniform distribution, which reflects the real number of possible certificates.

Varying the Number of Certificates and Cache Size: We conduct experiments by varying both the number of total possible certificates and the cache size to evaluate their influence. In this set of experiments, the number of devices is 100, λ is set to be 10, and the time is a 5000s duration, we set the cache size to be 32kb, 64kb and 128kb respectively. The total number of possible certificates varies from 100 to 10000. The results are shown in Fig. 10. We find that CCV outperforms I-Valid protocol with lower latency and lower number of local decryptions. The total number of possible certificates influences the performance of CCV and I-Valid, because the whole system needs to cache more certificates with larger number of possbible certificates, otherwise certificates



(a) Validation latency (b) Number of local decryptions

Fig. 10. Performance varies with the number of possible certificates and cache size.



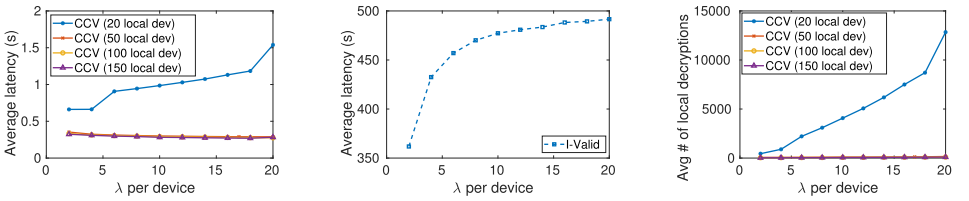
(a) Validation latency (b) Number of local decryptions

Fig. 11. Performance varies with time and cache size.

are validated by individual validation. When the number of total possible certificates increases, the latency (Fig. 10(a)) and number of decryptions (Fig. 10(b)) both increase in CCV, but still provides advantages compared to I-Valid. There is a sudden increase at 3000s certificates for CCV with 32KB cache size and a sudden increase at 6000s with 64KB cache size in both latency and the number of decryptions. This is because it is hard for cache pool to cache all the certificates, causing more individual validations. However, when the cache is 128KB, the whole cache pool enlarges and there is no such problem. We also conduct another experiments to analyze the

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Validation latency (b) Validation latency of I-Valid (c) Number of local decryptions

Fig. 12. Performance varies with λ per device and the number of devices.

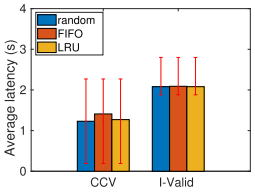


Fig. 13. Validation latency with cache replacement strategies.

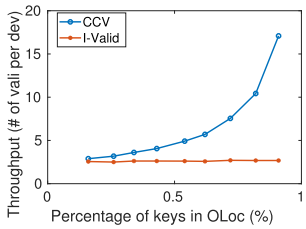


Fig. 14. Throughput comparison on simulator.

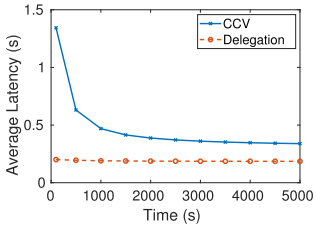


Fig. 15. Latency comparison on simulator.

influence of cache size. In this set of experiments, the number of devices is 100, λ is set to be 10, the total number of possible certificates is 10000, the cache size is set to be 32KB, 64KB and 128KB respectively. Fig. 11 shows the results. We can find that the average latency with 128KB keeps decreasing during time 0s to 5000s. CCV with 128KB cache provides the best performance among them. This is because the cache pool capacity with 32KB and 64KB per device is not big enough.

Varying λ per Device and the Number of Devices: Larger number of validation events happening per second indicates more validation requirements, which brings higher resource pressure for the devices. In this experiment, we evaluate the performance varies with λ per device. From Fig. 12(b), we can see the obvious increase of average latency for I-Valid when λ increases. In CCV more IoT devices can release the pressure, because the overall cache pool capacity increases. Fig. 12 shows the performance varying with λ per device and the number of devices. In this set of experiments, there are 3000 possible certificates, the cache size is set to be 64KB, and the time is 1000s duration. We can find that the average latency of CCV with 50, 100 and 150 devices is small and stays stable versus λ per device, which shows that CCV has a good performance with concurrent certificates validation requests with the shared distributed cache. There is a performance degradation with 20 devices because the cache pool with 20 devices cannot cache all the 3000 certificates, causing many individual validations for each device. Thus more certificates will be individually validated by its receiver, which is shown in Fig. 12(c).

Cache Replacement: This set of experiments vary cache replacement strategies including random, FIFO and LRU. Fig. 13 shows that the strategy has little influence on the latency of CCV, with random and LRU being slightly better.

Throughput: In this set of experiments, we compare the validation capacity of CCV and I-Valid by keeping devices performing validations in the simulation. The simulator simulates 100 devices simultaneously. The signing and verification algorithm is ECDSA with 160 bits keys. Fig. 14 shows that the number of validations on a device in one millisecond. From Fig. 14, we can see that CCV has a much better throughput compared to I-Valid protocol, especially when large percentage of public keys are recorded in OLoc.

B. Comparison With Delegation-Based Method

We also compare our CCV with delegation-based method. In delegation-based method, there is a trusted third party to help IoT devices to validate certificates. The third party can be a smart gateway, a delegation server and etc. IoT devices do not individually validate certificates. Instead, they send certificates validation requests to the third party, and then wait for the reply. After receiving the validation results, IoT devices can cache the validated certificates. Certificates can be validated by directly comparing with the cached ones for each IoT device. When the cache is full, cached certificates are replaced by newly cached ones according to three cache replacement strategies: random, FIFO (first in, first out) and LRU (Least recently used). In this set of experiments, the number of IoT devices is 100, the total number of possible certificates is 1000, λ is set to be 10 and the memory size of each device is 32KB. We assume there is a server which serves as the trusted third party. And the server has adequate computation power and memory. Fig. 15 shows the results. After 5000s, the average latency of each certificate for CCV and delegation-based method is 0.34s and 0.19s respectively. We can find that delegation-based method has better performance than CCV, because all the certificates are validated by the server with good computation and storage power. However, the server needs to continuously help IoT devices validate the certificates, once the server fails, the whole system

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE I

TIME FOR BUILDING OLOC

# certificates	100	1000	5000	10000	100000
Time (ms)	0.20	1.48	6.59	13.03	131.51

TABLE II

TIME FOR BUILDING OLOC WITH SAME KEYS BUT DIFFERNET VALS

# different vals	10	100	200	300	400
Time (ms)	2.05	2.98	4.33	5.98	7.52

TABLE III

TIME FOR UPDATING TRUST VALUES

# events	100	500	1000	5000	10000	100000
Time (ms)	0.87	3.9	7.754	37.422	105.434	1238.2

cannot validate certificates. CCV can still perform certificate validation when the tracker stops functioning for a duration of time.

C. Tracker Overhead

The heaviest task for the tracker is to update the OLoc. Table. I shows the time to construct an OLoc with different number of certificates. The results show that it is very time-efficient for the tracker to update the OLoc with a gigantic size of certificates. CCV also builds different OLoc to choose the most suitable holder of C for different devices when there are multiple choices. The reasons for multiple copies of C are the update delay of OLoc at the begining and the detection of malicious devices. The number of different values among different OLoc is small. Table. II shows the time to construct an OLoc based on an existing OLoc with 10000 shared keys but different values. The results show that it is very time-efficient for the tracker to build the OLoc when there is little difference in values with an existing OLoc.

We also evaluate the overhead for updating trust values. Table III shows the time to update trust values with different number of collaborative events when the number of device is 1000. The results show that it is time-efficient to update trust values.

Storage Cost: We evaluate the storage cost for the tracker in CCV. The tracker needs to store the following data. 1) The tracker stores trust values between IoT devices and two arrays to record the number of honest and dishonest validation events between devices. 2) The tracker stores all cached certificate-to-device information to build OLoc. 3) The tracker stores OLoc. 4) The tracker stores keys which include public keys of devices and symmetric keys between each device. 5) The tracker stores devices information including the IP of each device and the distance information between each two devices. Fig. 16 shows the storage cost. We can find that the tracker requires a small memory cost. The tracker needs about 1.24 MB storage with 200 devices and 10^4 possible certificates.

D. Security Results and Analysis

We first provide the evaluation results and then analyze the influence of parameters in the trust model. We also provide a detailed analysis about how CCV defends against six major attacks conducted by malicious devices mentioned in Sec. II-B.

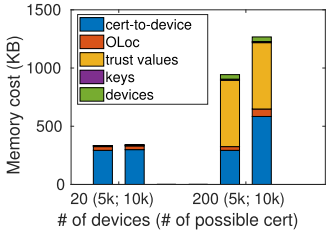


Fig. 16. Storage cost for the tracker.

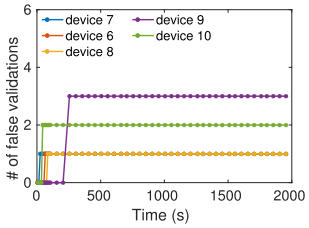
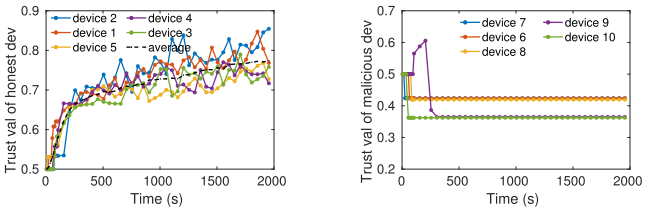


Fig. 17. Number of false validations.



(a) Trust values of honest devices (b) Trust values of malicious devices

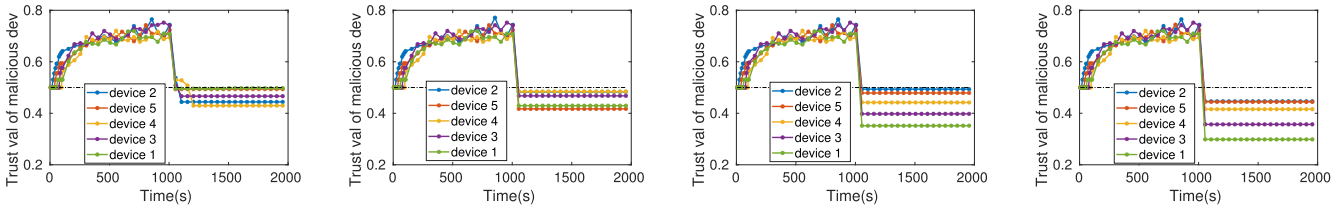
Fig. 18. Trust values.

Trust Value Evaluation: We monitor the trust values changes for both honest and malicious devices. In the set of experiments, the number of devices is 100, the number of certificates is 1000, time is set to a 2000s duration, and the cache size is 32 KB for each device. The initial trust value between any two devices is set to be 0.5. γ for the trust model is set to be 10. The trust value will be dynamically updated due to collaborative validation. The percentage of malicious devices is 5%. For malicious devices, they randomly choose to provide honest or dishonest validation when receiving collaborative validations. Fig. 18 shows the trust value changes. We show the results of five randomly chosen honest (Fig. 18(a)) and the five malicious devices (Fig. 18(b)). The other honest devices show similar results. We also show average trust values changes of all the honest devices, which is denoted by average in Fig. 18(a). We find that the trust values of honest devices increase to > 0.5 and are maintained in a high level. On the other hand, the trust values of malicious devices are all < 0.5 once they provide false validations. They will be filtered by the tracker during OLoc construction. We also show the number of total false validations by malicious devices in Fig. 17. When malicious devices are selected as helpers, they may conduct false validations. However, their trust values will dramatically decrease below 0.5 once the dishonest behaviors are detected by the tracker. Thus, they cannot be chosen as helpers anymore and the number of false validations will remain same.

Varying Parameter γ: Parameter γ is used to adjust the punishment degree of dishonest validations. Larger γ will cause lower trust value of the device when the tracker detects it has done dishonest validations. In this set of experiment,

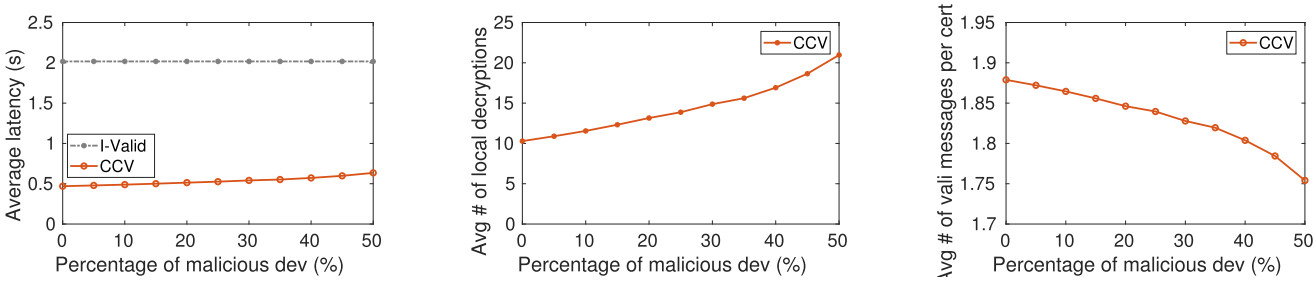
Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) γ = 1 (b) γ = 2 (c) γ = 5 (d) γ = 10

Fig. 19. Trust values of malicious devices with different γ.



(a) Validation latency (b) Number of local decryptions (c) Average # of messages per cert

Fig. 20. Performance varies with percentage of malicious devices after 2000s.

the setting is the same as that in the trust value evaluation experiment, except that γ is set to be 1, 2, 5, 10 respectively. The malicious devices begin to provide false validations after 1000s. Fig. 19 shows the trust values changes of malicious devices with different γ. There is a sharp decrease of the trust values near 1000s because of false validations provided by the malicious devices. The sharp decrease shows that the tracker detects the malicious behaviors and updates trust values of them timely. Different γ causes different degrees of decrease. Fig. 19(d) shows the largest decrease. The trust values dramatically decreases below 0.5. Thus we set γ to be 10 in our experiments, because it is sufficient to make malicious devices not chosen to be the collaborators once the dishonest validations are detected.

Varying the Percentage of Malicious Devices: Fig. 20 shows the performance of CCV varying with the percentage of malicious devices, ranging from 0% to 50%, after 1000s. At that time, most malicious devices will be detected and not used but the capacity of the whole cache pool decreases. Hence we find that the average number of decryptions (Fig. 20(b)) increases for CCV protocol with more malicious devices. The average number of decryptions for I-Valid keeps increasing with time, which is shown in Fig. 8(b). It reaches 95.15 at 1000s, which is much larger than 20.97 for CCV. The latency (Fig. 20(a)) has a slightly increase from 0.469 to 0.635. The reason is that more malicious devices indeed decreases the capacity of cache, causing the increase of average latency. But the cache pool of the remaining good devices can still support such number of total certificates, thus the increase of the latency is small. The average message per certificate decreases from 1.88 to 1.75 in Fig. 20(c) because more individual validation is conducted. However, CCV protocol still achieves a much better performance on average latency and number of local decryptions, compared to I-Valid.

False Validation Attack: The evaluation of trust value changes has been analyzed and the results, such as those in Fig. 17, indicate that after a short period of time, the malicious devices are not able to conduct false validation attacks.

Self-Promoting Attack: Trust value of each device is maintained and updated by the tracker. It is hard for a malicious device to promote itself to be a collaborator.

Defamation Attack: Each collaborative validation event must carry a digital signature of the holder for authenticity and non-repudiation purposes. The digital signature will be verified by the tracker. Hence a malicious device cannot forge a false validation event from a honest device unless it owns the private key of the honest device.

Traitor Attack: As shown in Fig. 18(b) and Fig. 19, once the malicious device provides dishonest validations, the trust value will drastically drop below the threshold, thus it can be selected as a helper in CCV anymore. Even the attacker intends to provide good validation when it senses its lower trust value. However, it does not have the chance anymore.

Whitewashing Attack: The tracker can audit the identity of the device. Thus, the high cost will effectively prevent the whitewashing attack.

Collusion Attack: Two or more malicious devices may improve their trust values by claiming that they helped each other. However, a malicious device will eventually provide a number of dishonest validations, which will be detected by the tracker statistically. In our model, the trust value reduction from one dishonest validation will be much larger than the trust value improvement from one honest validation. Hence it is only possible that the colluding malicious devices claim collaborative validations much more frequently than providing dishonest validations. Extremely high frequency of collaborative validations will also be detected by the tracker.

Sybil Attack: The tracker can audit the identity of the device, which is shown in the bootstrap phase. Thus, the high cost will effectively prevent the Sybil attack.

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

VI. Related Work

Certificate Validation: Certificate-based PKIs are responsible for creating, managing, distributing, storing and revoking public key certificates, such as X.509. They are widely used in Web browsing (TLS), email (S/MIME) and document authentication. Efficient certificate validation has attracted a broad attention of the research community in recent years [1], [2], [14], [15], [20], [26], [37], [41]. Some approaches delegate validation task to a trusted third party, such as a smart gateway [26], [38], a delegation server [14] and local ISPs [2]. Some approaches use prefetching and prevalidation techniques to reduce certificate validation cost [15], [37], which can remove the time pressure from the certificate validation. However, this approach brings huge cost for memory. Some approaches aim to reduce cost for checking certificates revocation status [12], [24], [41]. For example, Gañán *et al.* [12] proposes a collaborative certificate status checking mechanism to distribute certificate revocation information in Vehicular Ad Hoc Networks, which can provide a quick response to check revocation status.

Trust Model: A trust model is used to build trust relations between two nodes in a network. Trust is mainly divided into direct trust and indirect trust. Direct trust is computed by direct communications, while indirect trust is calculated by recommendation. Some algorithms can be used to calculate direct trust, such as subjective logic [17], [18], fuzzy method [6], [35], Bayesian [4] and game theory [9]. For indirect trust, devices in the trust model can get trust information from their trusted devices (recommender) [7], [32], then the trust value can be updated. In the trust model, direct trust and indirect are always combined to calculate the trust value [7], [11], [16], [32], [44]. For example, Feng *et al.* [11] proposes the NBBTE algorithm to establish the direct trust and indirect values between two nodes by comprehensively considering and combining various factors. Another efficient distributed trust model (EDTM) has been proposed in [16], which considers both direct and indirect trust and takes more trust metrics such as the energy level information into consideration besides communication behaviors.

VII. Discussion

One critical concern is the false validation results of the certificates provided by malicious IoT devices. In CCV, the IoT device will continue to communicate with another devices or servers when their certificates are validated, regardless of the wrong validation results from malicious devices. However, collaborative validation results will be also forwarded to the server on a sampling basis at the same time. The tracker will audit the validation results. If the tracker detects false validation results, it will send *FALSE_VALI_RESULTS* message to the corresponding device. The device will then stop the connection immediately when it receives the message. The latency to discover false certificates for the device is not too long. The main cost lies in the communication overhead between the device and the tracker. On the other hand, the tracker will punish the malicious device by drastically decreasing its trust value, which can be shown in Fig. 18(b). Thus, the malicious device cannot be chosen as collaborators anymore. To balance the overhead of the tracker and the security of whole system, all the certificates validation results are forwarded to the tracker at the beginning in order to quickly filter malicious devices. When the system is stable after a period of time, we can then set a proper sampling rate.

VIII. Conclusion and Future Work

In this article we design and evaluate the CCV protocol for fast public-key certificate validation. Our contributions include a memory-efficient and fast locator for certificate holders, called OLoc; a trust model for CCV to evaluate the trustworthiness of each device to avoid dishonest collaborative validation from malicious devices; and a complete protocol suite for efficient OLoc update, cache replacement, and revocation status checking mechanisms in a dynamic network. Evaluation results show that CCV significantly saves computation resource and validation latency on IoT devices. In future, we will apply distributed caching for storing revocation list and checking revocation status of the certificates.

References

[1] K. Akkaya, N. Saputro, S. Tonyali, M. Cebe, and M. Mahmoud, "Efficient certificate verification for vehicle-to-grid communications," Florida Int. Univ., Miami, FL, USA, Tech. Rep., 2017.

[2] A. Alrawais, A. Alhothaily, X. Cheng, C. Hu, and J. Yu, "SecureGuard: A certificate validation system in public key infrastructure," *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 5399–5408, Jun. 2018.

[3] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.

[4] F. Bao and I.-R. Chen, "Dynamic trust management for Internet of Things applications," in *Proc. Int. Workshop Self-Aware Internet Things*, 2012, pp. 1–6.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 299–314, Dec. 2002.

[6] D. Chen, G. Chang, D. Sun, J. Li, J. Jia, and X. Wang, "TRM-IoT: A trust management model based on fuzzy reputation for Internet of Things," *Comput. Sci. Inf. Syst.*, vol. 8, no. 4, pp. 1207–1228, 2011.

[7] I.-R. Chen, J. Guo, and F. Bao, "Trust management for SOA-based IoT and its application to service composition," *IEEE Trans. Services Comput.*, vol. 9, no. 3, pp. 482–495, Jun. 2016.

[8] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[9] J. Duan, D. Gao, D. Yang, C. H. Foh, and H.-H. Chen, "An energy-aware trust derivation scheme with game theoretic approach in wireless sensor networks for IoT applications," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 58–69, Feb. 2014.

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area Web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[11] R. Feng, X. Xu, X. Zhou, and J. Wan, "A trust evaluation algorithm for wireless sensor networks based on node behaviors and D-S evidence theory," *Sensors*, vol. 11, no. 2, pp. 1345–1360, Jan. 2011.

[12] C. Gañán, J. L. Muñoz, O. Esparza, J. Mata-Díaz, J. Hernández-Serrano, and J. Alins, "COACH: Collaborative certificate status checking mechanism for VANETs," *J. Netw. Comput. Appl.*, vol. 36, no. 5, pp. 1337–1351, Sep. 2013.

[13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[14] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle, "Delegation-based authentication and authorization for the IP-based Internet of Things," in *Proc. IEEE SECON*, Jun./Jul. 2014, pp. 284–292.

[15] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards viable certificate-based authentication for the Internet of Things," in *Proc. ACM Workshop HotWiSec*, 2013, pp. 37–42.

[16] J. Jiang, G. Han, F. Wang, L. Shu, and M. Guizani, "An efficient distributed trust model for wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1228–1237, May 2015.

[17] A. Jøsang, "A logic for uncertain probabilities," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 9, no. 3, pp. 279–311, 2001.

[18] A. Jøsang and T. Bhuiyan, "Optimal trust network analysis with subjective logic," in *Proc. IEEE Securware*, Aug. 2008, pp. 179–184.

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[19] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle, "When HART goes wireless: Understanding and implementing the WirelessHART standard," in *Proc. IEEE ETFA*, Sep. 2008, pp. 899–907.

[20] S. Kitajima and M. Mambo, "Verifying the validity of public key certificates using edge computing," in *Proc. Springer SICBS*, 2017, pp. 330–336.

[21] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "CRLite: A scalable system for pushing all TLS revocations to all browsers," in *Proc. IEEE SP*, May 2017, pp. 539–556.

[22] X. Li, H. Wang, Y. Yu, and C. Qian, "An IoT data communication framework for authenticity and integrity," in *Proc. IEEE/ACM IoTDI*, Apr. 2017, pp. 159–170.

[23] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Proc. ACM/IEEE IPSN*, Apr. 2013, pp. 153–166.

[24] M. Mahmoud, "Efficient certificate verification for vehicle-to-grid communications," in *Proc. Springer FNSS*, 2017, pp. 3–18.

[25] S. R. Moosavi *et al.*, "Session resumption-based end-to-end security for healthcare Internet-of-Things," in *Proc. IEEE CIT/IUCC/DASC/PICOM*, Oct. 2015, pp. 581–588.

[26] S. R. Moosavi *et al.*, "SEA: A secure and efficient authentication and authorization architecture for IoT-based healthcare using smart gateways," *Procedia Comput. Sci.*, vol. 52, pp. 452–459, Jun. 2015.

[27] R. Moskowitz and R. Hummen, *Hip Diet Exchange (DEX)*, document draft-moskowitz-hip-dex-00 (WiP), IETF, 2012.

[28] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet public key infrastructure online certificate status protocol—OCSP," RFC 2560, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2560.txt

[29] A. Oracevic, S. Dilek, and S. Ozdemir, "Security in Internet of Things: A survey," in *Proc. IEEE ISNCC*, May 2017, pp. 1–6.

[30] N. Park and N. Kang, "Mutual authentication scheme in secure Internet of Things technology for comfortable lifestyle," *Sensors*, vol. 16, no. 1, p. 20, Dec. 2015.

[31] C. Pei *et al.*, "WiFi can be the weakest link of round trip network latency in the wild," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[32] S. E. A. Rafey, A. Abdel-Hamid, and M. A. El-Nasr, "CBSTM-IoT: Context-based social trust model for the Internet of Things," in *Proc. IEEE MoWNeT*, Apr. 2016, pp. 1–8.

[33] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," RFC 6347, Jan. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6347.txt

[34] D. F. Santos, A. Perkusich, and H. O. Almeida, "Standard-based and distributed health information sharing for mHealth IoT systems," in *Proc. IEEE HEALTHCOM*, Oct. 2014, pp. 94–98.

[35] K. Singh and A. K. Verma, "A fuzzy-based trust model for flying ad hoc networks (FANETs)," *Int. J. Commun. Syst.*, vol. 31, no. 6, p. e3517, Apr. 2018.

[36] T. Song, R. Li, B. Mei, J. Yu, X. Xing, and X. Cheng, "A privacy preserving communication protocol for IoT applications in smart homes," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1844–1852, Dec. 2017.

[37] E. Stark, L.-S. Huang, D. Israni, C. Jackson, and D. Boneh, "The case for prefetching and prevalidating TLS server certificates," in *Proc. NDSS*, 2012, pp. 1–15.

[38] F. Van den Abeele, T. Vandewinckele, J. Hoebeke, I. Moerman, and P. Demeester, "Secure communication in IP-based wireless sensor networks via a trusted gateway," in *Proc. IEEE ISSNIP*, Apr. 2015, pp. 1–6.

[39] L. Wang and J. Kangasharju, "Real-world sybil attacks in BitTorrent mainline DHT," in *Proc. IEEE GLOBECOM*, Dec. 2012, pp. 826–832.

[40] M. Wang, C. Qian, X. Li, and S. Shi, "Collaborative validation of public-key certificates for IoT by distributed caching," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 847–855.

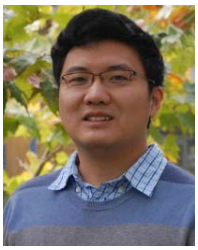
[41] A. Wasef and X. Shen, "EMAP: Expedite message authentication protocol for vehicular ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 1, pp. 78–89, Jan. 2013.

[42] Y. Yu, D. Belazzougui, C. Qian, and Q. Zhang, "A concise forwarding information base for scalable and fast name lookups," in *Proc. IEEE ICNP*, Oct. 2017, pp. 1–10.

[43] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for smart cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.

[44] T. Zhang, L. Yan, and Y. Yang, "Trust evaluation method for clustered wireless sensor networks based on cloud model," *Wireless Netw.*, vol. 24, no. 3, pp. 777–797, Apr. 2018.

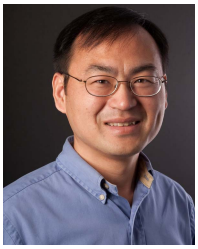
Minmei Wang (Graduate Student Member, IEEE) received the B.E. degree from the Nanjing University of Posts and Telecommunications in 2014 and the M.Sc. degree from Nanjing University in 2017. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of California at Santa Cruz. Her research interests include the Internet of Things and network security.



Chen Qian (Senior Member, IEEE) received the B.Sc. degree from Nanjing University in 2006, the M.Phil. degree from The Hong Kong University of Science and Technology in 2008, and the Ph.D. degree from The University of Texas at Austin in 2013, all in computer science. He is currently an Associate Professor with the Department of Computer Science and Engineering, UC Santa Cruz. His research interests include computer networking, data center networks and cloud computing, the Internet of Things, and software defined networks. He has published more than 60 research articles in a number of top conferences and journals, including ACM SIGMETRICS, IEEE ICNP, IEEE ICDCS, IEEE INFOCOM, IEEE PerCom, ACM UBICOMP, ACM CCS, IEEE/ACM TRANSACTIONS ON NETWORKING, and IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He is a member of the ACM.

Xin Li (Member, IEEE) received the B.Eng. degree in communication engineering from the University of Electronic Science and Technology of China, the M.S. degree in electrical engineering from the University of California at Riverside, and the Ph.D. degree from the Department of Computer Science and Engineering, UC Santa Cruz. His research interests include network security, software defined networking, network functions virtualization, and the Internet of Things.

Shouqian Shi (Graduate Student Member, IEEE) received the B.Sc. degree from the University of Science and Technology of China. He is currently pursuing the Ph.D. degree with the Department of Computer Engineering, UC Santa Cruz. His research interests include cloud computing, forwarding data structures, and routing algorithms.



Shigang Chen (Fellow, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China in 1993 and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign in 1996 and 1999, respectively. After graduation, he had worked at Cisco Systems for three years before joining the University of Florida in 2002. He is currently a Professor with the Department of Computer and Information Science and Engineering, University of Florida. His research interests include the Internet of things, big data, cybersecurity, RFID technologies, intelligent cyber-transportation systems, and so on. He has published over 200 peer-reviewed journal articles/conference papers. He holds 13 U.S. patents. He served in various chair positions or as committee members for numerous conferences. He holds the University of Florida Research Foundation Professorship and the University of Florida Term Professorship. He has served as an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE/ACM TRANSACTIONS ON NETWORKING, and a number of other journals. He is an ACM Distinguished Scientist.

Authorized licensed use limited to: GOOGLE. Downloaded on January 02,2021 at 00:15:21 UTC from IEEE Xplore. Restrictions apply.