Secure Service Virtualization in IoT by Dynamic Service Dependency Verification

Hajoon Ko
Harvard University
hrko@g.harvard.edu

Jiong Jin
Swinburne University of Technology
jiongjin@swin.edu.au

Sye Loong Keoh
University of Glasgow
Syeloong.keoh@glasgow.ac.uk

***Abstract*—Virtualizing Internet of Things (IoT) services is a concept of dynamically building customized high-level IoT services that rely on the real time data streams flowing from low-level standalone IoT devices. IoT service virtualization is essential when a myriads of IoT devices can get on-line, interact with each other, exchange data and based on them create one's own service. Especially, when virtualization occurs across multiple externals domains, it's crucial for clients to verify the source of virtual services, i.e. whether they are built based on authentic original service sources. Also, original services sources must be constantly aware of the identity of entities who (recursively) virtualize their services. To address these issues, this paper proposes IoT Service Dependency Tree (SDT) validation scheme. SDT uses service dependency trees and dependency signature trees, which enable clients to validate the original sources of a virtual IoT service, verify its service dependency relationships, and have original service sources to be constantly notified of the list of entities (recursively) virtualizing their services. This paper explains SDT scheme and presents use cases for IoT service virtualization where SDT can be applied. Our experimental analysis shows SDT is scalable for practical use.**

Keywords*—*IoT security, IoT service virtualization, service dependency

I. Introduction

Internet of Things (IoT) research is of special interest to academia and industry. In an academic context, IoT opens up new research grounds to explore, including fields such as network systems, cloud computing, data mining, machine learning, social networking and security & privacy. In the business context, IoT promises a highly productive business ecosystem which attracts a great number of application developers, businessmen and IT service providers worldwide, to automate complex cyber-physical systems in order to be cost-effective. Proliferating IoT networks and digital devices will provide smarter services and greater convenience to humans than in the past, thus creating a new era of technology.

As the number of IoT devices increases, multiple IoT devices are likely to collaborate to address high-level issues such as comprehensive data aggregation, analysis and processing. As a consequence, IoT devices providing services can be dynamically composed or *virtualized* [1]–[3], hence facilitating re-use of IoT services. IoT virtualization is a concept of dynamically building customized high-level IoT services which rely on the real-time I/O data streams from low-level IoT sensors/actuators. An IoT device having access to the Internet can potentially interact with any other IoT devices in the world, exchange data with them, and thereby creating a customized virtual service for its own clients. Virtualized IoT services provide scalability to build large-scale pervasive systems, and they can be dynamically composed and disbanded according to the requirements and context.

Meanwhile, security becomes complex because a virtualized IoT service is a composition of many IoT devices. A client who wishes to use a virtualized IoT service needs to: (1) authenticate the identity of the virtualized IoT service (2) verify if it has the capability of providing the claimed service. In particular, virtualized IoT services may have complicated service dependency, and virtualization may occur not only within a closed local network but through external open networks governed by mutually untrusted administrators or organizations [4]. Furthermore, virtualized IoT services may be re-virtualized in a recursive manner. To this end, it is important that a client who wishes to use a virtualized IoT service has a comprehensive view of the the virtual IoT service's underlying service composition, identifies external IoT services it depends on and verifies their real-time service dependency association, so that a virtual IoT service cannot falsely overstate its an association with other IoT services as its underlying service.

To address these security problems of virtual IoT services, this paper proposes IoT Service Dependency Tree (SDT) validation mechanism. In SDT, each virtual IoT service node creates its own *service dependency tree* that represents which external IoT service nodes its virtual service depends on. A client of a virtual IoT service can validate the virtual service by sending a random challenge number (*nonce*) to the virtual IoT service node and get it dynamically signed by all its depending lower-level IoT service nodes. Finally, the virtual IoT service node returns to the client a dynamically generated light-weighted *dependency signature tree*, as a proof of authorization. Then the client verifies the signatures in the tree and compares the tree's structural consistency with the virtual IoT service node's previously declared service dependency tree. SDT ensures that a virtual IoT service node cannot overstate its service composition to its client or higher-level virtual IoT service node by falsely declaring a service dependency tree branches it does not have. Thus, each client can verify the promised service composition of a virtual IoT service node. Furthermore, SDT enforces each virtual IoT service node to non-repudiably notify its identity to its all depending (physical/virtual) IoT service nodes in a dynamic manner, therefore every recursive step of virtualization is transparently visible to all its depending IoT service nodes.

This paper's contributions are as follows:

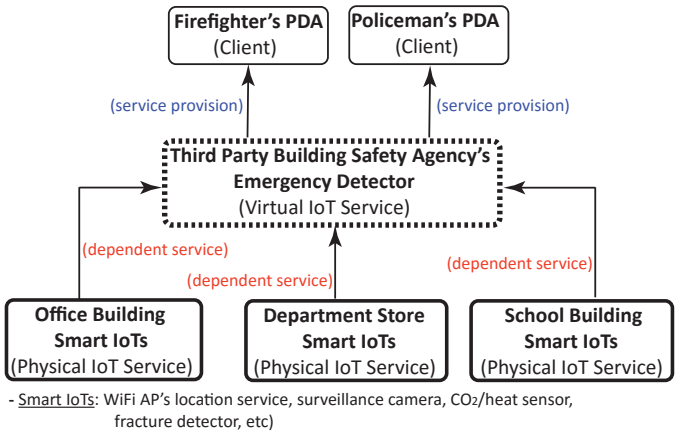

Fig. 1: A single-level IoT virtualization for smart building safety management

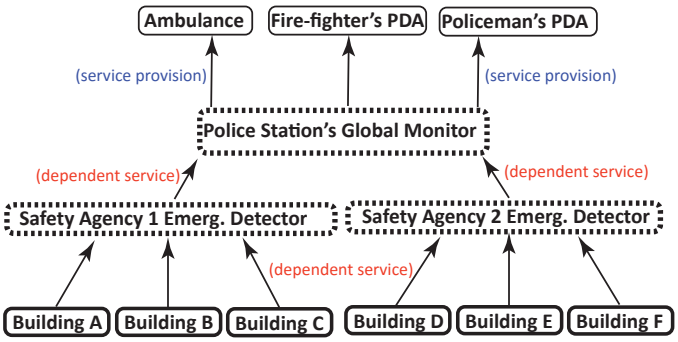

Fig. 2: Multi-level IoT virtualization

- Proposing SDT as a mechanism for clients to verify a virtual IoT service by comparing its service dependency tree with its dynamically created dependency signature tree.
- Allowing each physical IoT service node to be dynamically aware of the identity of all recursive virtual IoT service nodes who virtualize its service, which promotes transparency in service virtualization.

This paper is organized as follows: Section II presents use cases of virtual IoT services and their security requirements. Section III explains SDT validation mechanism. Section IV shows and discusses experimental results. Section VI presents related work. Section VII concludes the paper.

II. Use Cases and Security Requirements

A. IoT Virtualization Use Case

Figure 1 describes a scenario of a city's integrated smart building safety management system, where virtual IoT services can be deployed. Each building (office, department store, school) has its own smart building system with smart sensors managed by their local administrator. While some smart building functionalities such as energy consumption monitoring, air conditioning or light failure detection can be regarded as the building's exclusive local services, other safety functionalities such as detecting earthly vibration [5], structural defects, diffusion of toxic chemicals, potential explosives [6] or malicious human activities [7] regard people's life and in case of emergency these services must be made available to external fire-fighters or policemen. Figure 1 describes such a case where each building's particular services are exported to the external third party, a building safety agency. The agency's emergency detector collects real-time data for each building's condition, analyses them and sends the results to the fire-fighters or policemen's PDA, which can be utilized for their more efficient rescue activity. In this case, each building's smart IoT sensors are *physical IoT service nodes*, while the third party agency's emergency detector is a *virtual IoT service node* whose service depends on other lower-level physical IoT service nodes. In particular, the emergency detector's service depends on three lower-level IoT services from the office, department store and school buildings [1]. We denote this service dependency structure as the emergency detector's *service dependency tree*, whose height is 1. The root node is the emergency detector, and three leaf nodes are the office, department store and school's physical IoT sensor. In reality, the emergency detector may have a direct communication with each building's physical IoT sensor(s), or may have an indirect connection via each building's representative central server which directly communicates and collects data from physical IoT sensors in its building.

While Figure 1 is a case of single-level IoT virtualization, Figure 2 describes an example of two-level IoT virtualization. This example supposes there exist many independent third-party safety agencies, each of which host a virtual IoT service based on its assigned set of buildings. The data of emergency detectors from all independent safety agencies finally converge to the police station's global emergency monitor as the second-level virtual IoT service node, which provides service to its clients such as policemen, fire-fighters and ambulance crews. In this paper, a physical IoT service node denotes a service device at the leaf node in a service dependency tree, whereas a virtual IoT service node refers to any none-leaf nodes in the dependency tree.

B. Threat Model

In our threat model, the adversary is a dishonest virtual IoT service which overstates its underlying service composition by claiming itself to have a mutually agreed association with some external IoT service(s) which it doesn't have. An adversary could also capture and replay SDT, claiming that it has the capability of providing a particular virtual IoT service. In this way, a victim client will be tricked into believing the adversary as a verified virtual IoT service.

C. Security Requirements

As in Figure 1, in order for the emergency detector to host its virtual IoT service, it is necessary that each building's (office, department store, school) IoT sensors, or their owners, agree to export their serviced data stream to the emergency detector. In this scenario, the fire-fighter or policeman, as

[1] Although each building will have multiple physical IoT sensors, for simplicity in this explanation we regard building as a single physical IoT service node.

a client of the emergency detector, has to perform three security verifications on their virtual IoT service node: (1) to authenticate the server's (emergency detector) identity, (2) to dynamically verify if the emergency detector, as a virtual IoT service node, has a mutually agreed service dependency association with its depending IoT service nodes: office, department store and school's IoT services. Furthermore, in case of multi-level IoT virtualization, (3) each physical IoT service node must be aware of the identity of all higher-level virtual IoT service nodes whose services depend on it (i.e. identity of non-leaf nodes in service dependency trees).

Issue (1) can be addressed by issuing digital certificates to each (physical/virtual) IoT service node based on Public Key Infrastructure (PKI), supporting revocation techniques such as Certificate Revocation List (CRL) [8] or On-line Certificate Status Protocol (OCSP) [2]. We address issue (2) and (3) by using SDT mechanism (see Section III). Especially, SDT ensures that a virtual IoT service node cannot lie to its client or higher-level virtual IoT service node by falsely declaring a service dependency tree it does not have, thus each client can verify the service integrity of each virtual IoT service node. Furthermore, SDT enforces each virtual IoT service node to non-repudiably notify its identity to its all depending (physical/virtual) IoT service nodes, therefore each step of IoT service virtualization is transparently visible to all its lower-level IoT service nodes.

We generalize the aforementioned three security issues into the following security requirements for IoT virtualization:

- A client must be able to authenticate the identity of the (physical/virtual) IoT service node it is communicating with.
- A client must be able to validate his communicating virtual IoT service node's declared service dependency tree in real-time, in order to verify its service integrity.
- (Physical/virtual) IoT service nodes comprising a service dependency tree must be able to authenticate each other's identity.
- In a service dependency tree, a higher-level virtual IoT service node must be able to validate its lower-level virtual IoT service node's declared service dependency trees.
- In a service dependency tree, a physical IoT service node or an intermediate virtual IoT service node must be aware of the identity of all higher-level virtual IoT service nodes whose services depend on it (i.e. all parent nodes).

III. Service Dependency Tree (SDT) Validation Mechanism

A. SDT Overview

This section describes SDT validation mechanism as a solution for the security requirements illustrated in Section II. SDT comprises 5 protocols: (1) certificate issuance, (2) service dependency tree creation, (3) top-level validation of a service dependency tree, (4) recursive validation of a service dependency tree, and (5) revocation of a service dependency.

In our proposed SDT validation scheme, a virtual IoT service node informs its clients of its underlying service composition by creating and sending its own *service dependency tree*, which represents the complete view of its service dependency with its underlying external services. An example of a service dependency tree is Fig. 1 or Fig. 1 with the topmost client nodes (Firefighter's PDA, Policeman's PDA, Ambulance) removed. The client validates the service dependency tree by sending a random challenge number (*nonce*) to the virtual IoT service node, which has to be dynamically signed by all its depending lower-level IoT service nodes previously declared in its service dependency tree. The final output is a *dependency signature tree*. The client verifies it against the virtual IoT service's previously declared service dependency tree. If the verification is successful, the virtual IoT service is verified.

B. Certificate Issuance

Certificate Issuance

CA : Certificate authority
S : IoT (physical/virtual) service node

1) **CA** : $CERT_s = \text{Sign}(Priv_{ca}, \langle Pub_s, Info_s \rangle)$
   CA $\rightarrow$ **S** : $\langle CERT_s \rangle$

Fig. 3: Issuing an IDC and AC(s) to devices

Each (physical/virtual) IoT service device is issued a PKI certificate by its owner, administrator or a trusted authority, as described in Figure 3. Each service node uses its certificate to authenticate its identity either to its clients during service provision or to another (physical/virtual) service node during a service dependency tree creation (see Section III-G). For certificate issuance, Standard X509 certificates can be used with ECDSA [9] signatures, whose key size and the required CPU cycles are much less than RSA signatures and thus suitable for resource-constrained IoT devices that will participate in generating dependency signature trees (see Section III-F).

C. Creating a Service Dependency Tree

Figure 4 describes how two IoT service nodes create a service dependency. S denotes a higher-level IoT service node who will import service stream from S_l, a lower-level service node. For example, S_l's sensed environmental data will flow to S, who uses it to create its own higher-level virtual services. Note that in Figure 4, S is always a virtual IoT service node, while S_l can be either a virtual or physical IoT service node. In Step 1, S and S_l create an encrypted communication channel and perform standard PKI-based (e.g. DTLS) mutual authentication by using their certificates. In Step 2, S requests S_l for creating a service dependency chain between them. In Step 3, S_l declares its own service dependency tree for its service stream to be exported to S, who in turn validate S_l's declared service dependency tree in real time, by running **Top-level Validation** protocol in Fig. 5 (see Section III-D). In Step 4, S_l sends S's certificate to its directly lower-level IoT service nodes, who in turn recursively send it to their own lower-level

[2]http://www.rfc-editor.org/rfc/rfc6960.txt

Service dependency tree - Creation protocol

S : A higher-level IoT service node in a dependency tree
S$_l$: A lower-level IoT service node in a dependency tree
S$^*_{ll}$: S$_l$'s all recursively lower-level IoT service nodes

1) **S** $\leftrightarrow$ **S$_l$** : PKI-based mutual authentication by using their $CERT_s$ and $CERT_{s_l}$
2) **S** $\rightarrow$ **S$_l$** : Request for creating a service dependency tree $[S_l \rightarrow S]$
3) **S** $\leftrightarrow$ **S$_l$** : S verifies the service dependency tree(s) of S_l by running Fig. 5, where **S** := S_l, **C** := S
4) **S$_l$** $\rightarrow$ **S$^*_{ll}$** : $CERT_s$ (recursively notify S's identity to S_l's all lower-level IoT service nodes)
5) **S** $\leftrightarrow$ **S$_l$** : Sustain this channel for future's dynamic service dependency tree validation for $[S_l \rightarrow S]$

Fig. 4: Creation of an IoT service dependency tree $[S_l \rightarrow S]$ between service node S_l and S (i.e. S depends on S_l)

IoT service nodes. This step is to notify the identity of S to all its recursively depending IoT service nodes, to meet security requirement 5 in Section II-C. In Step 5, S and S_l have successfully created a service dependency chain between them and S defines its service dependency tree, accordingly. S and S_l maintain their current session to serve S's future's service dependency tree validation request made by S's future clients or higher-level IoT virtual service nodes.

Note that S_l can allow virtualizing its service to other multiple virtual IoT service nodes simultaneously, which are not part of S's service dependency tree because S's virtual service provision does not depend on them.

D. Top-level Validation of a Service Dependency Tree

Figure 5 describes how a client (C) dynamically validates a virtual IoT service node's (S) declared service dependency tree in real time. As a high-level description, the virtual IoT service node declares its service dependency tree to the client via an established encrypted channel. Then the client challenges the virtual service node by sending a nonce, which must be recursively signed by all non-root service nodes in the declared dependency tree. In the end, the virtual service node returns the generated dependency signature tree, which is to be verified by the client. In Step 1, C and S create an encrypted communication channel, and C authenticates S's identity by using standard PKI-based authentication. In Step 2, C requests S for validating its service dependency tree. In Step 3, S declares its service dependency tree and sends to C the certificates of all (physical/virtual) IoT service node nodes in its tree. In Step 4, C sends a random number, $nonce_c$, which must be signed by all service nodes in S's tree in a top-down manner. In Step 5, S appends a magic number MAG_{SDT} to the client's $nonce_c$, signs it and broadcasts a copy of it to its every immediate child node (S_j) in its service dependency tree, by using the channel created from Fig. 4, step 5.

In specific, S runs Fig. 6 with its immediately lower-level IoT service nodes in order to deliver SDT validation message

Service Dependency Tree - Top-level Validation protocol

S : Top-level IoT virtual service node
C : Client
Sign($Priv_s$, $\langle data \rangle$) : The signature for $data$ signed by S's private key $Priv_s$
MAG$_{SDT}$: A publicly known high-entropy magic number for SDT signature message

1) **C** $\leftrightarrow$ **S** : PKI-based server authentication by using $CERT_s$
2) **C** $\rightarrow$ **S** : The client (**C**) requests the service node (**S**) for service dependency tree validation
3) **S** : $L = \{S_j \mid$ **S** depends on a lower-level service node $S_j\}$
   : $M = \{CERT_{S_j} \mid S_j \in L\}$
   : $D = \{$Declare its service dependency tree(s)$\}$
   S $\rightarrow$ **C** : $\langle M, D \rangle$
4) **C** $\rightarrow$ **S** : $\langle nonce_c \rangle$
5) **S** : $\pi_s = \text{Sign}(Priv_s, \langle nonce_c || MAG_{SDT} \rangle)$
   : $Q_s = \langle nonce_c, \pi_s \rangle$
   S $\rightarrow$ **S$_i$** : Send $\langle Q_s \rangle$ to every $S_j \in L$ (i.e. go to Fig. 6, Step 1, where $S_l := S, S_{ll} := S_j$)
6) **S** $\leftarrow$ **S$_i$** : Receive $\langle T_{s_j} \rangle$ from every $S_j \in L$ (i.e. return from Fig. 6, Step 4)
7) **S** : $T_s = \langle \pi_s, \{T_{s_j} \mid S_j \in L\} \rangle$
   S $\rightarrow$ **C** : $\langle T_s \rangle$

Fig. 5: A client's IoT service dependency tree validation of service node S

to its all (recursively) lower-level IoT service nodes.

The purpose of appending MAG_{SDT} before signing is to make it explicit for the signer that he signs it for SDT verification purpose. Using MAG_{SDT} prevents a dishonest virtual IoT service from performing a *relay* attack. Suppose a client requires a dishonest virtual service to get $nonce_c$ signed by its falsely overstated underlying IoT service nodes. And suppose the dishonest virtual service is a client of the falsely overstated IoT service node(s) but doesn't have a service dependency association with them. Then, if MAG_{SDT} is not used, the virtual service can contact those underlying services as their client, send them the original client's same $nonce_c$, get it signed, and return it to the original client, who will then believe the virtual service to have a genuine service dependency association with those falsely claimed underlying service nodes.

Recursive validation is to be discussed in detail in Section III-E. In Step 6, S receives recursively created *dependency signature trees* (see Section III-F) for $nonce_c$ from its every immediate child node (S_j). In Step 7, S merges signature trees received from its children nodes into a single tree and sends it to C, who validates it. If the signatures of the tree's every path verifies successfully and the tree is consistent with what S declared in D, C concludes S's service dependency tree is valid.

Note that in Step 5, if S does not properly sign $nonce_c$ with its private key in an attempt to hide its identity from its

further lower-level IoT service nodes, its Q_s will be rejected by its immediate child nodes (i.e. S_j, immediately lower-level IoT service nodes), because they check the existence of S's signature on Q_s whenever they receive it.

E. Recursive Validation of a Service Dependency Tree

Service Dependency Tree - Recursive Validation protocol

$\mathbf{S_{ll}}$: A lower-level IoT service node in the service dependency tree
$\mathbf{S_l}$: A higher-level intermediate IoT service node in the service dependency tree

1) $\mathbf{S_l} \rightarrow \mathbf{S_{ll}} : \langle Q_{s_l} \rangle$

2) $\mathbf{S_{ll}}$: $L = \{S_j \,|\, \mathbf{S_{ll}}$ depends on a lower-level service node $S_j\}$
$\pi_{s_{ll}} = \text{Sign}(Priv_{s_{ll}}, \langle \pi_{s_l} || MAG_{SDT} \rangle)$
$Q_{s_{ll}} = \langle Q_{s_l}, \pi_{s_{ll}} \rangle$
$\mathbf{S_{ll}} \rightarrow \mathbf{S_l}$: Send $\langle Q_{s_{ll}} \rangle$ to every $S_j \in L$
(i.e. recursively go to step 1, where $S_l := S_{ll}, S_{ll} := S_j$)

3) $\mathbf{S_{ll}} \leftarrow \mathbf{S_l}$: Receive $\langle T_{s_j} \rangle$ from every $S_j \in L$
(i.e. return from step 4 of the previous recursion)

4) $\mathbf{S_{ll}}$: $T_{s_{ll}} = \langle \pi_{s_{ll}}, \{ T_{s_j} \,|\, S_j \in L \} \rangle$
$\mathbf{S_{ll}} \rightarrow \mathbf{S_l} : \langle T_{s_{ll}} \rangle$

Fig. 6: **Recursive Validation** protocol of service dependency tree $[S_{ll} \rightarrow S_l \rightarrow S]$ for an intermediate virtual IoT service node S_l

Figure 6 describes the **Recursive Validation** protocol of a service dependency tree for an intermediate node (i.e. middle-level virtual IoT service node). This protocol is almost identical to Step 4∼7 of **Top-level Validation** protocol in Figure 5, with the exception that the role of client (C) is replaced by a higher-level intermediate IoT service node (S_l). In Step 1, a parent service node (S_l: higher-level IoT service node) sends to its child service node(S_{ll}: lower-level IoT service node) its top-down dependency signature chain (Q_{s_l}) built upon the client's initial $nonce_c$. S_{ll} verifies Q_{s_l}'s signature chain and if is consistent with the latest service dependency chain update received from Fig. 4 step 4 (**Service Dependency Creation** protocol), and Fig. 8 step 1 (**Service Dependency Revocation** protocol). In Step 2, S_{ll} extracts Q_{s_l}'s signature π_{s_l}, concatenates the SDT magic number MAG_{SDT} to it, signs it, appends this new signature $\pi_{s_{ll}}$ to Q_{s_l} to generate its new message $Q_{s_l l}$, then sends it to its every child service node (S_j: immediately lower-level IoT service node). Each child service node performs the same recursive process all the way until reaching the leaf node (lowest physical IoT service node). In Step 3, S_{ll} receives the dependency signature tree (T_{S_j}) back from each S_j. In Step 4, S_{ll} merges dependency signature trees T_{S_j} received from all its children nodes and finally sends back to S_l. Note that if S_{ll} is a leaf node, $T_{s_{ll}}$ will contain only $\langle \pi_{s_{ll}} \rangle$.

F. Dynamically Generating Dependency Signature Trees

Figure 7 is an illustrative example of a dependency signature tree being generated during validation of a service dependency tree in protocols of Figure 5 and 6. In Figure 7, S is a 2-level virtual IoT service node depending on three directly lower-level IoT service nodes: S_{L1}, S_{L2} and S_{L3}. S_{L1}

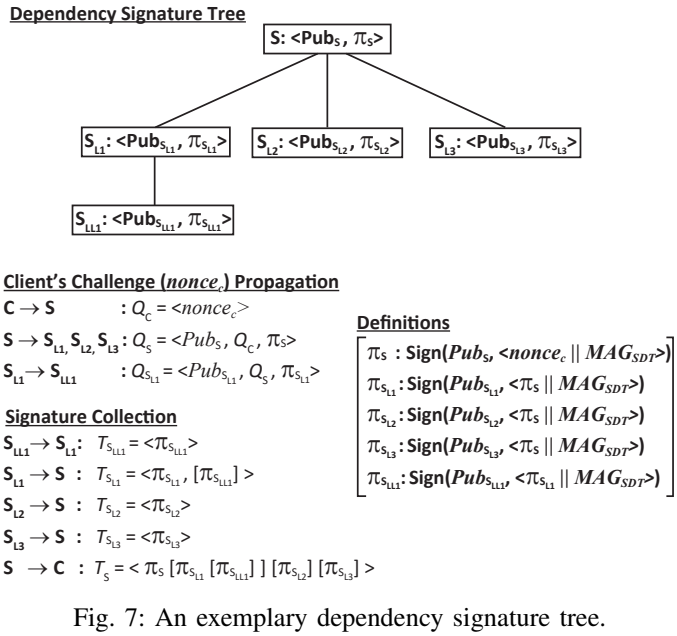

Fig. 7: An exemplary dependency signature tree.

is a single-level virtual IoT service node which has S_{LL1} as its depending lower-level IoT service node. S_{LL1}, S_{L2} and S_{L3} are physical IoT service nodes because they don't depend on other IoT service nodes.

During client challenge ($nonce_c$) propagation, each IoT service node sending down its message (Q) is supposed to additionally signs it so that its all recursively lower-level IoT service nodes are transparently aware of its identity. Before signing data, each IoT service node must additionally concatenate MAC_{SDT} to it in order to make it explicit that the signature was generated for SDT verification purpose. Each lower-level IoT service node receiving the message (Q) verifies the entire signature chain in the message starting from ($nonce_c$), and then also verifies that the last signature is created by its immediately upper-level virtual IoT service node, the same entity from service dependency chain creation in Figure 4.

During signature collection, the lower-level service nodes only need to send their signature (π), because the upper-level service nodes have already learned about the public keys of their lower-level service nodes from their service dependency chain creation in Figure 4, and they can derive the target data that is supposed to be signed at each level in the tree (the parent node's π concatenated by MAG_{SDT}. Sending and receiving only signatures reduce data overhead. Each intermediate service node collects signatures from its immediate child nodes, generates a signature tree by using its own signature as a root node and the signatures from child nodes as sub-trees of the root, and returns the generated signature tree to its parent node. At the end of this recursive signature tree generation, S gets a finalized signature tree whose each path forms a chain of signatures, where each node contains a signature signed on its parent node's signature. S sends the finalized signature tree to the client.

Finally, when the client validates the signature tree, it initially verifies the root node's signature (π_s) by using $nonce_c$

and Pub_s, and then recursively verifies each of $\pi_{S_{L1}}$, $\pi_{S_{L1}}$ and $\pi_{S_{L1}}$ by using π_s and each of their corresponding public key. Such recursive signature verification continues on all the way until reaching every leaf node in the signature tree. If the client finds that every signature in the tree is verified, and if the signature tree's structure is consistent with the service dependency tree previously declared by the S in Figure 5 step 3, he concludes that S's virtual IoT service is validated.

G. Service Dependency Revocation protocol

Revoking a Service Dependency

S	: A higher-level IoT service node in a dependency tree
$\mathbf{S_l}$	: A lower-level IoT service node in a dependency tree
$\mathbf{S^*_{ll}}$	: S_l's all recursively lower-level IoT service nodes
ID(*CERT*$_s$)	: The certificate ID of *CERT*$_s$
1) $\mathbf{S_l} \rightarrow \mathbf{S^*_{ll}}$	: $ID(CERT_s)$ (recursively notify the revoked S's identity to S_l's all lower-level IoT service nodes)
2) $\mathbf{S_l} \rightarrow \mathbf{S}$	: Notify that service dependency between **S** and $\mathbf{S_l}$ has been revoked

Fig. 8: Revoking a IoT service dependency between between service node S_l and S (i.e. S depends on S_l)

Figure 8 describes how a service dependency is revoked between two IoT service nodes (S_l and S). Strictly speaking, it is the lower-level IoT service node (S_l) who is in charge of this revocation. In Step 1, S_l sends the certificate ID of its virtual service node to be revoked to its immediately lower-level IoT service nodes, who recursively send it to their own lower-level IoT service nodes. In Step 2, S_l notifies S that their service dependency has been successfully revoked. As of this point, S cannot prove its service dependency association with S_l to its clients who newly requests for SDT validation. By default, each client runs SDT protocol once upon its connection or reconnection with the virtual IoT service and the SDT validation result holds until their connection ends. Optionally, the client can set her own refreshment period for SDT validation (e.g. 5 minutes, 10 minutes, 1 hour) while remaining in the same connection.

IV. Evaluation

We implemented SDT software as an OpenSSL C wrapper library, whose source code is approximately 1100 lines long. We evaluated SDT scheme's overhead by simulating virtual IoT service nodes with changing the number of depending external services and changing their virtualization level. In our simulation, we used multiple desktop PCs hosting a large group of virtual IoT service nodes ($\sim$ 100), which remotely communicate with other PCs' virtual IoT service nodes. Each PC, connected to LAN, is physically located in different buildings, which simulates the communication of IoT service nodes residing in different building networks. In order to simulate resource-constrained IoT devices, each PC's CPU frequency was limited at 900MHz and each virtual IoT service node instance's RAM at 500MB. The type of LAN used in our experiment was 10BASE-TX (100 MBit/s). For proper simulation we made sure that every step of service virtualization took place between virtual IoT service node instances from two different PCs residing in different buildings- not ones from the same PC. This ensures that each IoT service node communicates with remote IoT service nodes in the external domain- not local ones within the same PC. IoT service nodes used X509 certificates signed by ECDSA. Each certificate's size was 491 bytes and each signature comprising the dependency signature tree was 71 bytes.

To our knowledge, SDT is the first security scheme that enables clients to efficiently validate recursively virtualized IoT services. There is no previous work that provides the same level of security as SDT that allows clients to recursively verify the depending nodes of a virtualized IoT service as well as their service dependencies. As such, our experiment compares the performance of SDT with a naive scheme, both of which provide the same level of security. The naive scheme forces each client to manually and recursively contact all depending nodes of the virtual IoT service, authenticate them and query about the genuineness of their dependency relationship with the virtual IoT service. In contrast, SDT simplifies the client's duty to only contacting the virtual IoT service and verifying the dependency signature tree generated and returned by it. This considerably reduces the naive scheme's overhead, while maintaining the same level of security.

Our experiment assumed a client device's communication with a virtual IoT service node. We measured the delay and exchanged data size for SDT verification scheme between the client and the service node. These overhead measurements include the comprehensive communication cost incurred during exchanging certificates, generating and sending dependency trees, and verifying them.

The experiment was conducted in 2 parts. The first part of the experiment assumed a single-level virtual IoT service node and we changed the number of its depending external IoT services from 10 to 100. The experimental results are summarized in Table I. As the number of depending service nodes increased, the delay and exchanged data size between the virtual IoT service node and client for SDT scheme increased linearly. When a virtual IoT service node depended on 10 external IoT services, its verification took 26.4 ms and its exchanged total data size with the client was 6.54 kilobytes. The delay remained to be less than 100 ms while the virtual service node's number of depending IoT services was below 40. When the virutal IoT service node depended on 100 external IoT services as the worst case scenario, its verification time and data overhead was 245.2 ms and 57.04 kilobytes. The overall overhead increased linearly with the number of depending IoT service nodes, which implies SDT is practically usable. When SDT's performance is compared to the naive scheme, SDT's delay is approximately 54% smaller and its data size overhead is 57% smaller, on average. The naive scheme incurs a larger overhead because the client has to create as many new network connection as the number of depending nodes of the virtual IoT service to verify. On the other head, SDT removes this overhead of creating extra network connections by using the technique of verifying dynamically generated dependency signature trees (see Section III-D, III-E and III-F)

In the second part of the experiment, we changed the virtual IoT service node's virtualization level from 1 to 7, where each intermediate virtual IoT service node node has exactly 2 child service nodes (i.e. two depending service nodes). Thus, the top-

Schemes & Metrics \ # of Depending IoTs		10	20	30	40	50	60	70	80	90	100
Delay (ms)	***Naive scheme***	45.3	95.1	135.2	165.3	218.8	247.8	314.6	351.0	398.3	430.4
	SDT scheme	26.4	46.6	69.7	91.5	113.0	156.9	175.1	200.2	223.8	245.2
Exchanged Data (KB)	***Naive scheme***	11.03	21.13	30.60	41.34	51.43	61.53	71.64	81.74	91.84	101.94
	SDT scheme	6.54	12.15	17.13	23,28	28.98	34.59	40.21	45.82	51.43	57.04

TABLE I: Overhead comparison for a single-level virtual IoT service node over various number of depending IoT service nodes

Schemes & Metrics \ Virtualization Level [# of total IoTs]		1 [1]	2 [3]	3 [7]	4 [15]	5 [31]	6 [63]	7 [127]
Delay (ms)	***Naive scheme***	4.3	11.9	32.8	61.2	130.4	275.8	532.2
	SDT scheme	4.3	8.1	15.3	31.9	68.2	136.3	268.9
Exchanged Data (KB)	***Naive scheme***	1.01	3.03	7.07	15.15	31.31	63.63	128.27
	SDT scheme	1.01	2.13	4.38	8.86	17.84	35.79	71.70

TABLE II: Overhead comparison for a virtual IoT service node over various virtualization levels

level virtual IoT service node's service dependency tree formed a complete binary tree. We use this binary tree structure only as a standard example for our experiment. In real scenarios, SDT does not have to be a binary tree and its dependency tree can be in any shape. The experimental result is summarized in Table II. When the virtualization level increased to 5, the delay and data overhead was 68.2 ms and 15.65 kilobytes, which is reasonably manageable. At virtualization level 7 as the worst case scenario, the delay was 268.9 ms and data size 64.02 kilobytes. We expect virtualization level 5 to be more than sufficient in most use cases.

Note that SDT's overhead is only one-time. That is, the client uses SDT only once when it makes an initial connection or re-connection with the virtual IoT service node. To this end, SDT is best applicable to services where the client continuously uses the same service within the same connection with the service node, such as a building condition monitoring service in Figure 1, 1.

When processing dependency signature trees between intermediate virtual IoT service nodes in the service dependency tree, we observed approximately 175% delay reduction when setting their network sockets' `TCP_NODELAY` option to be true, which disables Nagle's network packet buffering algorithm [3]. In SDT, this configuration change enforces the client's challenge and each IoT service node's signature to be sent out immediately once prepared, rather than waiting for buffer outgoing data packets to become a full size.

V. Discussion

SDT prevents a dishonest virtual IoT service from overstating its underlying service composition to its clients. However, SDT does not necessarily prevent it from understating its underlying service composition. For example, a dishonest virtual IoT service who has a service dependency association with an external service *A*and *B* may claim to its client that it has an association only with service *A*, omitting service *B*. A dishonest virtual IoT service may have a motivation to make such a false claim to the client if service *B* is a sensitive service that has a conflict of interest with the client. Based on SDT, it's possible for an honest virtual IoT service to prove to clients that it has no association with some sensitive service, such as service *B*. The virtual IoT service makes a service dependency relationship with both service *A* and service *B*. In this case, service *B* has to volunteer to help the virtual IoT service to prove to its clients that it has no service dependency with service *B*. When a client requests for a signature on its nonce, the virtual IoT service forwards it to both service *A* and *B*. Service *A* appends MAG_{SDT} to the nonce and signs, while service *B* appends MAG_{NSDT} to it and signs. MAG_{NSDT} is a new magic number representing the opposite of MAG_{SDT}, which implies that the signer has no service dependency association with the IoT virtual service in concern.

When a virtual IoT service's some of existing service dependencies get revoked, the client who has already established a connection with the virtual IoT service has to re-run SDT protocol with it to update this information. By default, SDT protocol runs only at the initial connection or re-connection between a client an a virtual IoT service. However, each client can additionally set its reasonable period for re-running SDT protocol depending on the nature of service. In our experiment, each virtual IoT service can host up to 10 underlying IoT services with any virtualization level less than 4, if the maximum allowed communication cost for SDT validation is 30 ms. A virtual IoT service can host up to 50 underlying IoT services with any virtualization level less than 6, if the maximum allowed communication cost for SDT validation is 100 ms.

VI. Related Work

IoT service virtualization was introduced by Zhang *et al.* [1]. They developed a virtual sensor editor tool which creates a virtual representation of sensors (on a PC), based on real time data streams flowing from remote physical sensors. This

[3]https://en.wikipedia.org/wiki/Nagle%27s_algorithm

approach assumed a closed local network scenario excluding the possibilities of external attacks, and thus security was not taken into account. We improved IoT virtualization to be security-aware by using SDT mechanism, so that it operates securely through insecure open networks.

OAuth [10], [11] allows a service provider and its client to verify a third party's eligibility for directly accessing the client's resource stored in the service provider. Upon the client's approval, the third party gets an access token, with which it can retrieve the client's resource stored in the server. In OAuth, granting a third-party access to a client's resource is analogous to granting service virtualization in SDT. However, SDT makes difference in that it allows recursive service virtualization and allows the physical IoT service node (i.e. original resource owner) and end-users to dynamically verify the service dependency tree (i.e. resource dependency tree).

Macaroon [12] is an improved version of Internet cookies that can be delegated and whose caveats can be added by the delegator (e.g. a third party server) over each delegation. Each macaroon stands for its holder's capabilities. Macaroon is used for a primary and third party server to authorize their clients. On the other hand, SDT is used for a physical IoT service node (primary server) and clients to verify a virtual IoT service node (third party server).

Oh *et al.* [13] proposes an access control framework for *Web of Things* (WoT). In this work, each IoT resource is mapped to a unique HTTP URI and a set of access control policies. Whenever a client accesses an URI, its mapped access control policies are checked on the client. The proposed WoT access control is designed for client authorization purpose, and unlike SDT, it does not consider recursive resource virtualization.

Our current CamFlow research aims at applying Information Flow Control (IFC) [14] to IoT devices by using a kernel module [15] and middleware [16] as trusted IFC enforcement software, whose integrity is dynamically verified by remote attestation [17] and TPM [18] technology. When IFC is applied to IoT devices, each device or data owner can have control on where his data may continuously flow across multiple IoT nodes. However, this scheme is unsuitable for verifying the detailed service dependency of a virtual service. SDT achieves this by using a service dependency tree and dependency signature tree.

Numerous previous research works propose cloud-based IoT services [19]–[21], where the cloud server collects data from IoT devices, processes them and provides service to its clients. In this framework, all IoT devices resides behind the wall of the cloud server and clients cannot directly access them. SDT proposes a more decentralized security approach such that each client is allowed to directly interact with individual IoT devices without the intervention of central cloud servers. Since each IoT device's data are not stored or processed by the third party server, data privacy for IoT device owners is better preserved.

SDT is compatible other existing security mechanisms. For example, Moreno *et al.* [22] proposes decentralized IoT service access control based on client certificates, which is issued by the service domain manager who grants clients access to each IoT resource. SDT can adopt client certificates for mutual authentication between virtual servers and clients. Keoh *et al.* [23] proposes improved DTLS protocol standards optimized for resource-constrained IoT devices. SDT can run over such optimized DTLS protocols for efficient communication among clients and virtual/physical servers.

Some research in wireless sensor networks (WSN) have proposed data aggregation schemes based on privacy homomorphism [24], which allows direct computation on encrypted data. For example, concealed data aggregation (CDA) scheme [25] calculates SUM and AVERAGE of data collected from multiple WSN nodes, while homomorphic stream cipher (HSC) scheme [26] supports efficient data aggregation for mean, variance and standard deviation. [27] proposes a cloud service framework for storing and managing client data as an encrypted form. However, ciphertext-based data processing has limitation for tasks such as high-level data analysis, interpretation and pattern recognition. To the end, SDT allows virtual IoT service nodes to have full access to their depending services, while ensuring their service dependencies to be transparently visible to their clients and lower-level servers.

There has been much research on efficiently revoking compromised nodes in an ad-hoc network. For example, [28] and [29] propose revocation algorithms for bad nodes based on votes from neighbouring nodes. The core idea is to predefine different keys for each future timeslot, and split each future key into multiple shares by using a multivariate secret-sharing algorithm [30], keeping each share in a different node. A particular node can retrieve its next timeslot's key only if all external nodes holding its future secret shares approves him. However, according to this scheme a node cannot be revoked until its current timeslot expires. SDT supports immediate revocation of compromised nodes by dynamically revoking dependency branches within a service dependency tree.

VII. Conclusion

In this paper we addressed security challenges for IoT service virtualization and proposed SDT scheme as the solution. In SDT, each virtual IoT service node declares its own service dependency tree and allows its clients to validate the declared tree in real-time by sending a challenge number to the virtual IoT service node, which must be signed recursively by its all depending IoT service nodes comprising its declared tree. The virtual IoT service node finally returns the client a dependency signature tree whose structure should be exactly the same as the service node's originally declared service dependency tree. The client verifies the signature tree by verifying each node's signature with its corresponding IoT service node's public key. The size of the dependency signature tree is optimized to contain only the minimal information necessary to verify the entire tree. Our experimental results indicate SDT incurs only a small overhead and thus efficiently applicable in real scenarios.

References

[1] J. Zhang, Z. Li, O. Sandoval, N. Xin, Y. Ren, R. Martin, B. Iannucci, M. Griss, S. Rosenberg, and A. Cao, J. an Rowe, "Supporting Personizable Virtual Internet of Things," in *Proc 10th International Conference on Ubiquitous Intelligence and Computing, and Autonomic and Trusted Computing (UIC/ATC)*, pp. 329–336, IEEE, 2013.

[2] S. Alam, M. Chowdhury, and J. Noll, "Senaas: An event-driven sensor virtualization approach for internet of things cloud," in *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*, pp. 1–6, Nov 2010.

[3] A. Mahmud, R. Rahmani, and T. Kanter, "Deployment of flow-sensors in internet of things' virtualization via openflow," in *Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on*, pp. 195–200, June 2012.

[4] H. Ning, H. Liu, and L. Yang, "Cyberentity security in the internet of things," *Computer*, no. 4, pp. 46–53, 2013.

[5] S. N. Pakzad and G. L. Fenves, "Statistical analysis of vibration modes of a suspension bridge using spatially dense wireless sensor network," *Journal of Structural Engineering*, vol. 135, no. 7, pp. 863–872, 2009.

[6] S. Simi and M. V. Ramesh, "Real-time monitoring of explosives using wireless sensor networks," in *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*, A2CWiC '10, pp. 44:1–44:7, ACM, 2010.

[7] *Intelligent Building Hazard Detection Using Wireless Sensor Network and Machine Learning Techniques*, pp. 485–492.

[8] D. W. Chadwick, A. Otenko, and E. Ball, "Role-based Access Control with X. 509 Attribute Certificates," *Internet Computing, IEEE*, vol. 7, no. 2, pp. 62–69, 2003.

[9] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International Journal of Information Security*, pp. 36–63, 2001.

[10] B. Leiba, "Oauth web authorization protocol," *Internet Computing, IEEE*, vol. 16, no. 1, pp. 74–77, 2012.

[11] E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague, "Oauth demystified for mobile application developers," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, (New York, NY, USA), pp. 892–903, ACM, 2014.

[12] A. Birgisson, J. G. Politz, lfar Erlingsson, A. Taly, M. Vrable, and M. Lentczner, "Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud," in *Network and Distributed System Security Symposium*, 2014.

[13] S. W. Oh and H. S. Kim, "Study on access permission control for the web of things," in *Advanced Communication Technology (ICACT), 2015 17th International Conference on*, pp. 574–580, 2015.

[14] A. C. Myers and B. Liskov, "A Decentralized Model for Information Flow Control," in *17th Symposium on Operating Systems Principles (SOSP)*, pp. 129–142, ACM, 1997.

[15] T. Pasquier, J. Singh, D. Eyers, and J. Bacon, "CamFlow: Managed Data-Sharing for Cloud Services," *IEEE Transactions on Cloud Computing*, 2015.

[16] J. Singh, T. F. J.-M. Pasquier, and J. Bacon, "Securing Tags to Control Information Flows within the Internet of Things," in *International Conference on Recent Advances in Internet of Things (RIoT'15)*, IEEE, 2015.

[17] M. Nauman, S. Khan, X. Zhang, and J.-P. Seifert, "Beyond Kernel-level Integrity Measurement: Enabling Remote Attestation for the Android Platform," in *Trust and Trustworthy Computing*, pp. 1–15, Springer, 2010.

[18] S. Bajikar, "Trusted Platform Module (TPM) Based Security on Notebook PCS-White Paper," *Mobile Platforms Group Intel Corporation*, pp. 1–20, 2002.

[19] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through internet of things," *Internet of Things Journal, IEEE*, vol. 1, pp. 112–121, April 2014.

[20] S. Abdelwahab, B. Hamdaoui, M. Guizani, and A. Rayes, "Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler," *Internet of Things Journal*, vol. 1, no. 3, pp. 276–288, 2014.

[21] T. Zhang, "Defending Connected Vehicles Against Malware: Challenges and a Solution Framework," *Internet of Things Journal*, vol. 1, no. 1, pp. 15–20, 2014.

[22] A. Skarmeta, J. Herna?ndez-Ramos, and M. Moreno, "A decentralized approach for security and privacy challenges in the Internet of Things," in *Internet of Things (WF-IoT)*, pp. 67–72, IEEE, 2014.

[23] S. L. Keoh, S. Kumar, and H. Tschofenig, "Securing the internet of things: A standardization perspective," *Internet of Things Journal, IEEE*, pp. 265–275, 2014.

[24] J. Domingo-Ferrer, "A provably secure additive and multiplicative privacy homomorphism," in *Proceedings of the 5th International Conference on Information Security*, pp. 471–483, 2002.

[25] J. Girao, D. Westhoff, and M. Schneider, "Cda: concealed data aggregation for reverse multicast traffic in wireless sensor networks," in *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, pp. 3044–3049 Vol. 5, 2005.

[26] C. Castelluccia, E. Mykletun, and G. Tsudik, "Efficient aggregation of encrypted data in wireless sensor networks," in *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*, pp. 109–117, 2005.

[27] M. Henze, S. Bereda, R. Hummen, and K. Wehrle, "SCSlib: Transparently Accessing Protected Sensor Data in the Cloud," in *The 5th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2014)*, p. 370??375, ScienceDirect, 2014.

[28] T. Moore, J. Clulow, S. Nagaraja, and R. Anderson, "New strategies for revocation in ad-hoc networks," in *Proc. 4th European Conference on Security and Privacy in Ad-hoc and Sensor Networks*, ESAS, pp. 232–246, Springer, 2007.

[29] H. Chan, V. D. Gligor, A. Perrig, and G. Muralidharan, "On the distribution and revocation of cryptographic keys in sensor networks," *IEEE Trans. Dependable Secur. Comput.*, vol. 2, no. 3, pp. 233–247, 2005.

[30] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, pp. 612–613, Nov. 1979.