Computers & Security 97 (2020) 102000

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Understanding the security of app-in-the-middle IoT

Hui Liu[*], Juanru Li, Dawu Gu

Shanghai Jiao Tong University, Shanghai, China

A R T I C L E I N F O

Article history:
Received 3 April 2020
Revised 8 July 2020
Accepted 5 August 2020
Available online 12 August 2020

Keywords:
Internet of Things (IoT)
IoT security
Consumer IoT
Smart home
Security analysis

A B S T R A C T

In recent years IoT platforms and smart-home systems have rapidly grown. Meanwhile, mobile apps have been widely accepted as user interfaces in these consumer IoTs, allowing users to retrieve processed data and issue specific commands. We notice that these companion apps are also used as gateways, providing Internet connectivity for resource-constrained devices, and its mobility advantage over static gateways further promotes applications of this kind. In this paper, we extracted this pattern into a new architecture called *app-in-the-middle IoT*. We provided a holistic view of what app-in-the-middle IoT is and introduced its attack surface by comparing it with two well-studied IoT architectures, which we refer to as cloud-in-the-middle IoT and trigger-action platform IoT. We detailed the similarities and differences between the three architectures, derived security goals of app-in-the-middle IoT, and drew the key to analyzing it from authentication, access control, and availability aspects. We adopted a method of building an abstract model and extracting the concept of *token* from the working process. To achieve security goals, the token needs to own these properties: mutual authentication, unforgeability, and resistance to replay attacks. We argue that the role the app plays is critical to the working process, which affects how the properties of the token are satisfied. During analysis, we find that the application scenarios significantly influence the role of the app. Therefore, we discussed the security of different situations separately. For each scenario, we indicated how the token should be generated and distributed to meet the security goals, and summarized several security rules. We analyzed several practical cases, which demonstrate that violating these rules can lead to severe consequences, such as unauthorized access, information leakage, irrevocable authorization, and device hijack.

1. Introduction

IoT greatly facilitates the lives of people on-the-go and at home by enabling the connectivity of surrounding physical devices. These consumer-oriented devices can be health monitoring devices (such as glucose monitors and fitness trackers), smart door locks when we use short-term house rental services (e.g., Airbnb), and smart home devices, providing meaningful data to the user and performing specific actions. Devices communicate with clouds to enable data integration, device remote access, and device sharing. They are usually controlled through companion apps installed on users' smartphones. Smartphones have been in people's daily life for many years, users have become accustomed to using them to interact with surrounding IoT devices, regarding the device companion app as an interface to retrieve processed data, issue specific commands, and configure automatic rules.

Unlike a desktop computer or a smartphone that connects to the Internet with high-speed networks, lots of IoT devices are battery-powered, have limited computation capability, and may be used in the mobility, unable to support the power-consuming and resource-demanding WiFi or Cellular networks connections thus lacking the capability of direct Internet connection. Therefore, these devices may use protocols such as Bluetooth Low Energy (BLE) to communicate with the more powerful smart mobile device nearby, establishing personal area wireless connectivity. That is, smartphones act as a gateway/hub, providing Internet connectivity for resource-constrained devices.

The companion app does the protocol wrapping, enabling the device to communicate with the cloud. Due to the relative position of the app, we name this pattern *app-in-the-middle IoT*. It is different from the existing well-discussed IoTs, especially the one with Hub-connected devices. We will introduce the differences and similarities in detail in Section 3. Here we take several points as examples to depict.

Previous works that discuss consumer IoT security fall into two categories. One is the security of the cloud-based IoT (which we name as *cloud-in-the-middle*), for example, Zhou et al. studied the

[*] Corresponding author.
E-mail addresses: ice_wisdom@sjtu.edu.cn (H. Liu), jarod@sjtu.edu.com (J. Li), dwgu@sjtu.edu.com (D. Gu).

https://doi.org/10.1016/j.cose.2020.102000
0167-4048/© 2020 Elsevier Ltd. All rights reserved.

interaction between the three components (i.e., devices, apps, and clouds) (Zhou et al., 2019). The other is security analysis on the systems that the cloud can run trigger-action rules (e.g., SmartApps in SmartThings), which we name as *Trigger-action platform IoT*. For example, Fernandes et al. introduced vulnerabilitis caused due to over-privileged smart apps (Fernandes et al., 2016a). These works ignore the impacts of the app that plays different roles in the app-in-the-middle IoT. There also exists works focusing on the interaction between the app and Bluetooth devices. Sivakumaran et al. showed that malicious apps co-located with the device companion app could access the device through the BLE channel, and proposed the way to defend this kind of attack application-layer security (Sivakumaran and Blasco, 2019). Naveed et al. showed that unauthorized apps could access the external device (e.g., Bluetooth device) due to the coarse-grained permission provided by the Android system (Naveed et al., 2014b). These works studied the threats that can affect the app-device interaction but failed to put the app-device interaction in the larger picture. We notice that secure app-device interactions need to seek the help of the cloud to, for example, authenticate each other and further provide support for functions like device sharing, while cloud-device communication is proxied by the app in the middle. The fundamental problem is that app-device authentication needs to be done through the cloud, while cloud-device communication is proxied by the app. This interdependence substantially expands the attack surface of the app-in-the-middle IoT.

In this work, we are the first to propose the new genre of IoT architecture called app-in-the-middle IoT, in which devices depend on the smartphone's Internet connectivity to communicate with their clouds. We abstract the features of this particular IoT architecture and thoroughly discuss its security. We take the approach of forming an abstract model and extracting the *token* concept. By analyzing the token property, we conclude potential threats and propose security rules. The practical cases we analyze demonstrate that violating these rules can lead to severe consequences. This work advances the understanding of this particular type of consumer IoT, thereby improving the security status of smart devices of this type.

The rest of this paper is structured as follows: we provide an overview of app-in-the-middle IoT in Section 2. This section also introduces attacks that can be conducted on the device-app Bluetooth channel. Section 3 introduces the differences and similarities between app-in-the-middle IoT and two existing architectures that have been widely discussed in research papers, particularly with regard to the function of components and security threats. Section 4 details security analysis of app-in-the-middle IoT. Section 5 reviews the security rules and provides several cases that show the consequences of violating security rules. Related work is described in Section 6, and we conclude the paper in Section 7.

2. Background

2.1. Overview of app-in-the-middle IoT

Previous work has revealed the use of mobile devices as gateways. Seol et al. (2015) and Zachariah et al. (2015) proposed the very same smartphone-centric approach to connect resource-constrained IoT devices to the Internet. Pereira and Aguiar, 2014 point out that with varied local and wide-area connectivity capabilities, smartphones offer a unique opportunity to serve as mobile gateways for other more constrained devices with local connectivity. Santos et al. (2016) proposes an IoT-based mobile gateway solution for mobile health (m-Health) scenarios where gateway autonomously collects information and acts as a communication channel. Meanwhile, smartphones are already a common choice of gateways for a myriad of devices in practice,

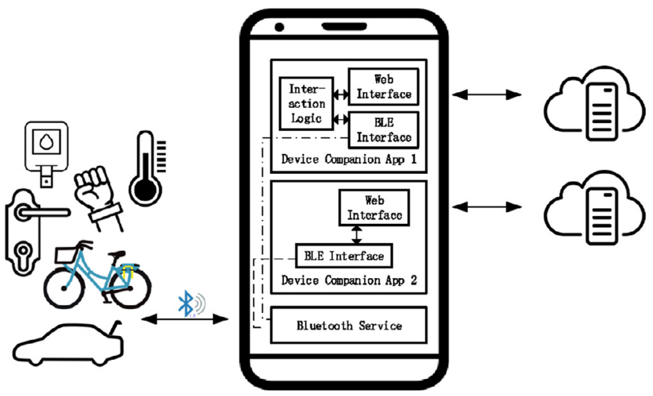


Fig. 1. The app-in-the-middle IoT.

such as wearable devices, smart appliances and health devices, smart locks that used on the shared bicycles and electric motorcycles, and smart car trunk locks that enable car trunk sharing and trunk delivery. We classify this particular kind of IoT as *app-in-the-middle IoT*, as shown in Fig. 1.

We clarify this concept by describing several preliminary definitions: (i) The IoT devices that adopt this architecture are featured by the attributes that they are power-constrained, have only wireless low power communication capability, and need to communicate with remote clouds to report data or accept instructions. (ii) Devices use smartphones as gateways to connect to the Internet, benefiting from the convenience of mobility and flexibility. The widely adopted wireless protocol is Bluetooth, as almost all smartphones are equipped with Bluetooth technology whose communication range and power consumption are suitable for this application scenario. (iii) Device companion apps are installed on the smartphones to send application-level data and instructions back and forth between devices and their corresponding clouds. (iv) The role of the companion app can vary greatly.

Previous work considers gateway/hub as an intermediate node, holding the view that hub-connected devices follow a similar model to cloud-connected devices and the existence of the gateway/hub will not affect the interactions between the device, the companion app, and the cloud (Zhou et al., 2019). However, we argue that app-in-the-middle does have an impact on the interactions. Its security needs to be thoroughly discussed.

AITM IoT takes the smartphone as the hardware gateway and the device companion app as the software gateway. The app and the device need to authenticate each other at the application-layer because the lower link can not provide identity assurance (Sivakumaran and Blasco, 2019). The authentication, however, requires the involvement of the cloud because these two do not share any pre-set secrets. On the other hand, only the smart mobile device is able to access the cloud. Ideally, it transmits messages between the IoT device and the cloud, without eavesdropping or modifying the data. But the app is under threat; the data may be obtained or even reformatted by attackers.

In addition, the application scenario also affects the role that the app needs to play. For devices whose functions are collecting and reporting data to clouds, like fitness wristbands, smart thermometers, and blood glucose meters, their companion apps mainly wrap and forward the data. However, for those devices that are controlled by users to perform specific actions, such as smart door locks and car trunks, their companion apps may handle more complex interaction logic to satisfy the advanced requirements of users, such as privacy protection, partial control, and temporary user authorization and revocation.

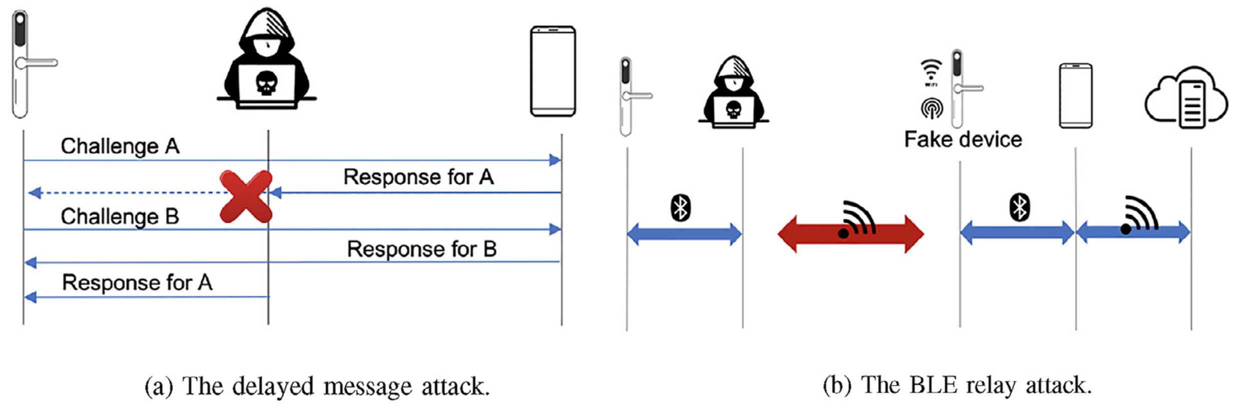


Fig. 2. Two attacks that endanger BLE.

2.2. Bluetooth Low Energy

BLE (Bluetooth Low Energy) technology is an ideal tool for implementing IoT (Internet of Things) applications, as it consumes little power and provides wireless communication. Many devices and services using this technology already available on the market.

Attacks against BLE. Two classes of adversaries can threaten the BLE security at a relatively low cost: passive attackers that sniff the wireless traffic by simply using low-cost hardware such as Ubertooth One and Adafruit Bluefruit LE Sniffer; active attackers that manipulate the traffic with lots of mature skills available (Jasek, 2017). Possible attacks include: eavesdropping attack that sniffs the unencrypted traffic and extracts sensitive data; device impersonation attack that clones the device using MAC address and other information; replay attack that reuses sensitive commands such as unlock the device; delayed message attack that jams specific message and replays it later, as shown in Fig. 2(a). These attacks combining with system flaws may result in severe consequences; for example, an impersonated device may harm the system by injecting malicious data and receiving sensitive commands that leak user privacy.

The BLE standard provides security mechanisms such as pairing and bonding to protect the connection from these attacks. Ideally, a pairing scheme named LE Secure Connections provides link encryption, which can effectively prevent adversaries. However, as the devices are power constrained, they can barely have display or input capabilities. The Bluetooth pairing process can only use the mode that adopts default keys, which renders the attackers effortlessly decrypt the established connections (Ryan, 2013). In addition, the Bluetooth service on the phone is shared between all installed apps that have corresponding permissions. The coarse-grained permission issue (Naveed et al., 2014a) and co-located app attacks (Sivakumaran and Blasco, 2019) may cause unauthorized access.

Besides the attacks mentioned above, it is worth noting that a specific MITM attack called relay attack poses a new threat to the BLE channel. As shown in Fig. 2(b), the attacker clones the MAC address of the lock, which is used by the mobile to identify the lock and pretends to be the real lock talking to the mobile. At the same time, the attacker pretends to be the mobile and talks to the real lock. The attacker relays the messages, and none of the entities are aware of the attack. The relay attack on the BLE channel rises from the ranks due to its low attack cost and high prevention difficulty. It only requires the attacker to have two Bluetooth devices and the capability of transmitting data between these two entities over long distances. But detecting and preventing the attack is non-trivial. Existing approaches includes measuring the strength of the received signal, detecting the time delay introduced by the attack, and testing for co-presence through ambient sensing. The relay attack jeopardizes users because the assumption of the device being physically present is broken by arbitrarily extending the communication distance.

3. Demysfying app-in-the-middle IoT

This section introduces the function of the components of two existing architectures that have been widely discussed in research papers, and summarizes the security threats they face. By comparing the similarities and differences between the app-in-the-middle (AITM) IoT and the two architectures, we try to answer the question: what makes the AITM IoT fundamentally different.

Many pieces of IoT security research focus on two architectures, which we refer to as the cloud-in-the-middle architecture and the trigger-action platform architecture. We introduce them separately in Sections 3.1 and 3.2. A summary of the differences between the three types of IoT architecture regarding the functions of the components is shown in Table 1.

3.1. Cloud-in-the-middle architecture

As shown in Fig. 3(a), a cloud-in-the-middle (CITM) IoT has three components, namely: device, vendor cloud, and device companion app. The vendor cloud is the central point that manages interactions with the device. The device communicates directly with the cloud if it is IP-enabled; otherwise, it uses energy-efficient protocols such as Z-Wave and ZigBee to connect to a hub/gateway, which itself is an Internet-connected device that adapts protocols and relays messages between the device and the cloud. To clarify the function of each component, we elaborate on a typical working procedure of the IoT system of this architecture. This procedure is representative since the other two architectures also work similarly or based on it. We use WiFi-enabled devices as an example to describe the *working procedure*. The differences brought by WiFi-enabled devices and hub-connected devices only affect the device setup step; the effect will be reflected when inspecting the attack surface.

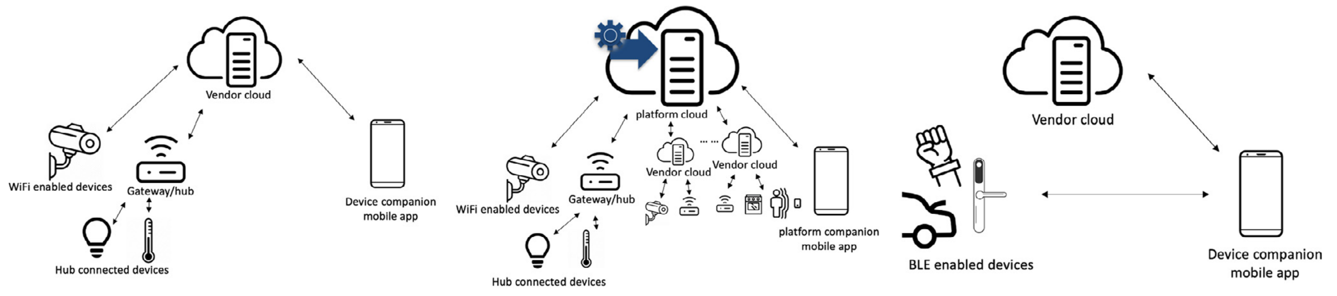
a) Device setup and user binding: To be able to access the Internet, the WiFi-enabled device needs to acquire the WiFi credential under the help of the companion app. Several schemes, such as SmartConfig and Airkiss, can achieve this provisioning. After that, the device is able to establish a connection with the cloud. The device and the user associate with each other by respectively authenticating to the cloud, and the binding relationship is created and maintained by the cloud.

b) Device remote control and monitoring: Authenticated users can remotely view the status of their bound devices and issue

Table 1
Three types of IoT architecture and their differences in terms of the functions of their components.

Architecture	Component	Feature and function
CITM IoT*	Device	IP-enabled or hub-connected; communicate with the cloud directly or via a hub; stable connection with the cloud.
	Vendor Cloud	Central component; vendor-specific; authenticate users and devices; manage binding relationships and temporary authorization; interact with devices; run simple rules that provide limited control automation; enforce access policy; maintain audit logs and device logs.
	Device Companion App	Device-specific; interface for users to set up, manage, remotely monitor and control devices; interface for users to make temporary authorization and revocation.
Trigger-Action Platform IoT	Device	Same as CITM IoT device.
	Platform Cloud	Same as CITM IoT vendor cloud (optional); integrate other vendor clouds; support complicated automation; run trigger-action rules; apply access control policy of the rules.
	Platform Companion App	Same as CITM IoT device companion app (optional); interface for users to manage (i.e., install, configure, and uninstall) trigger-action rules; interface for users to associate with other platforms.
AITM IoT**	Device	Resource-constrained; battery-powered; smartphone connected, mostly BLE enabled; intermittent connection with the cloud.
	Vendor Cloud	Same as CITM IoT vendor cloud except for the automation capability.
	Device Companion App	Same as CITM IoT device companion app; interact with devices through the smartphone-provided Bluetooth channel; proxy traffic between devices and clouds; involved in the entire working process.

* CITM IoT refers to Cloud-in-the Middle IoT architecture as elaborated in Section 3.1. ** AITM refers to App-in-the Middle IoT architecture as elaborated in Section 3.3.



(a) Cloud-in-the-Middle IoT (b) Trigger-Action Platform IoT (c) App-in-the-Middle IoT

Fig. 3. Three types of IoT architecture.

commands through the device companion app. Besides, the user can create some simple rules that instruct the cloud to interact with the device automatically. For example, the user can set a rule that turns on the conditioner at 7 pm on weekdays. When the time is up, the cloud will automatically send the corresponding command. In this way, the system achieves limited automation.

c) Temporary authorization and revocation: Device owners may want to share their devices with some temporary users. The owner authorizes a user to access a device within a specified period and expects the access automatically revoked after the authorization expired. The user can also actively revoke the authorization in the companion app.

d) Device unbinding: The user can make the unbinding request in the companion app or reset the device to unbind the device.

From the above description, we extract the features and functions of the three components of the CITM IoT architecture. The *device* is IP-enabled or hub-connected; they collect data and perform commands using sensors and actuators, and communicate with the cloud directly or via a hub. The *cloud* is central to the system, responsible for authenticating users and devices, managing binding relationships and temporary authorization, enforcing access policies, interacting with devices, and maintaining audit logs (to answer the question "Who did what, where, and when?") and device logs (information about device lifecycle events, e.g., connections, errors); it also supports limited automation The *app* is device-specific, supporting at most a range of devices from the same vendor; it provides the interface for users to set up and manage devices, remotely monitor and control devices, and make temporary authorization and revocation.

Attack surface. Existing researches have explored the vulnerabilities of CITM IoT from many aspects. We only consider design flaws here since we want to draw lessons for the security analysis of other IoT architectures; hence the implementation issues (e.g., the UAF vulnerabilities in firmware and apps) are out of the scope.

- Device backdoor: Some devices include a tiny web cloud in their firmware that allows users to access the device directly. The device implements only a simple username/password authentication scheme, which entirely relies on the limited computing and storage resources of the device, completely bypassing the rich resources and security functions that the cloud can provide. Even worse, the device may include a deliberately hidden authentication interface, and anyone possessing a super account can fully access the device (yearinfo).
- Device discovery and network access provisioning: For WiFi-enabled devices, WiFi credentials can be sniffed by a local wireless attacker during WiFi provisioning (Li et al., 2018); a user may wrongly choose the malicious device that attacker put nearby with the same identifier as the user's device, which also results in WiFi credential leakage (Valente and Cardenas, 2017). For hub-connected devices, the process of a device establishing a connection with a hub can be vulnerable; for example, an attacker can launch a downgrade attack when a Z-Wave device pairing with a hub, which grants the attacker full access to the device (Fouladi and Ghanoun, 2013).
- Device authentication: During device binding, what credentials the cloud use to authenticate the device influence the security severely. Devices leveraging DeviceID and other hard-coded or easily-inferred device identifying information are vulnerable to

secret leakage, thus resulting in a bunch of attacks. For example, an attacker possessing leaked DeviceIDs can impersonate the original device, thus launching the data injection, privacy breach, and denial-of-service attacks; the attacker can preemptively bind a real device with his account, making the user holding the genuine device not able to bind; the attacker can also send unbinding requests, causing the user to lose control of the device (Chen et al., 2019b; Zhou et al., 2019).
- Cloud: The cloud may not enforce a strict check on the message received from the device; for example, the cloud accepts binding requests even if the device is already in the bound state, allowing the attacker who sends the binding request to take control of the device; the cloud may not associate the device-cloud connection with the user account, making an attacker with an attacker account capable of manipulating the connection (Zhou et al., 2019).
- Device-Cloud app-level protocol: The app-level communication protocol used between the device and the cloud have a significant impact on security. For example, if MQTT is deployed, a temporary user can plant a backdoor while authorized to use the device legally. When the specific trigger event occurs, the pre-set command will be executed even if the user who issued the command has been revoked from accessing the device (Jia et al., 2020). The distinguishability of status report messages and keep-alive messages allows an attacker to intentionally block status report messages, which blinds the cloud from acquiring information from the device and endangers the log integrity (OConnor et al., 2019).

We consider the user is properly authenticated by the cloud, and the communication channel between the device and the cloud is well protected (e.g., SSL), as they have been fully discussed in past research.

3.2. Trigger-action platform architecture

As shown in Fig. 3(b), the trigger-action platform architecture also contains three components: device, platform cloud, and platform companion app. Its most significant difference from the cloud-in-the-middle architecture is the function of the platform cloud. Fundamentally, the platform cloud works the same as the vendor cloud in the way that it interacts with the platform-produced device, while the difference lies in that the platform cloud can integrate other vendor clouds hence controlling a myriad of devices. Furthermore, the integration brings the possibility of complicated automation, which is implemented in the form of trigger-action rules (SmartThings name them SmartApps). All in all, the trigger-action platform architecture can be seen as built on top of the cloud-in-the-middle architecture while emphasizing the trigger-action rules.

From the view of component functions, the platform cloud can be seen as two parts: (1) functioning as vendor clouds, but supporting complex automation rules; (2) functioning as third-party integration clouds that run trigger-action rules while device interaction achieved by invoking vendor cloud interfaces. Therefore, apart from playing the same role as they do in the cloud-in-the-middle architecture, the platform cloud and the platform companion app also support complicated automation: The cloud runs trigger-action rules that automatically monitor and control associated devices, and enforce access control policies that specify which rules can access which devices and perform which actions; The platform companion app provides interfaces for the user to manage (i.e., install, configure, and uninstall) trigger-action rules and associate with other platforms.

Attack surface. The attack surface of the cloud-in-the-middle IoT is applicable to trigger-action platform IoT. Here we concentrate on the threats posed by the introduction of trigger-action rules.

- Cloud access control: the cloud control which actions a rule can perform on devices by granting permissions. The permission may be designed too coarse-grained such that a rule can abuse the permission to cause unexpected results; for example, a rule that only requires the *lock* permission is granted with the permission that can perform the *unlock* action (Fernandes et al., 2016a).
- Trigger-action rules: a rule may request more permission that it needs, and include implicit code paths that execute behaviors beyond the user's expectation. Multiple rules may conflict on controlling the same device or produce implicit inter-rule trigger-action chains.
- Protocol: many platforms use the OAuth protocol to integrate vendor clouds. The OAuth token is critical since it can be used to control bound devices. The platform manages token in a centralized fashion, which can cause massive damage once the platform is compromised and the token is leaked (Fernandes et al., 2018).
- Companion app: The app may embed the OAuth client ID and secret that used to integrate vendor clouds, which would result in OAuth token leakage and further endanger bounded devices (Fernandes et al., 2016a).

3.3. App-in-the-middle architecture

As shown in Fig. 3(c), what makes app-in-the-middle IoT different is the relative position of the app in this architecture: it locates between the device and the cloud. The device depends on the app to proxy traffic to and from the cloud. This app-reliant feature prevents the cloud from automatically executing commands to control the device; consequently, the app-in-the-middle IoT cannot integrate into the trigger-action platform IoTs.

Devices in this architecture are resource-constrained, battery-powered, often demand long standby time, and have limited wireless low power communication capabilities. To leverage the smartphone as a hardware gateway, they often choose the Bluetooth Low Energy (BLE) as the wireless communication protocol. The device companion app communicates with the device through the Bluetooth channel the smartphone provides.

Attack surface. We enumerate possible attacks that may endanger the app-in-the-middle IoT, based on the attack surfaces of the above two architectures. The entire workflow of the AITM IoT faces similar threats due to the similarity in the objectives of the all IoT system and the function of the cloud. But the role the app plays undoubtedly complicates the security situation.

- Device direct access: the device may accept connection from any BLE-enabled apps with none or simple authentication mechanisms.
- Device discovery and binding: the app may wrongly choose the device with the same identifier as the user's device; the process of device pairing with a smartphone may leak the encryption key, causing a device impersonation attack.
- App-device communication channel: the Bluetooth channel is managed in a way that all the authorized apps share the same channel by the Android sytem, which leads to the co-located app attacks that a malicious app who co-locates with the device companion app and is permitted to use the Bluetooth channel can access the device (Sivakumaran and Blasco, 2019). The root cause of this attack is that the smartphone provides coarse-grained permission when control which app can access the channel that app communicates with the device

Table 2
The attack surface of three types of IoT architecture.

	CITM IoT	Trigger-Action Platform IoT	AITM IoT
Device	• Device backdoor: interface for direct access with weak authentication; hidden authentication interface. • Device discovery and network access provisioning: device identification; WiFi provisioning; hub-connected device pairing. • Device identification: identifying information presented to the cloud may be hard-coded or easily-inferred.		
	-	-	• Device direct access through Bluetooth channel • Bluetooth connection, pairing, and binding
Cloud	• Device authentication: information used to authenticate the device may be falsifiable. • Device state management: may allow illegal device state manipulating messages. • Device-user binding management: may allow binding status change by non-owner users. • Temporary authorization management: authorization may be irrevocable. • Log Integrity: any event that deviates from the preset can compromise log integrity.		
	-	• Cloud access control (coarse-grained permission) • Trigger-action rules (over-privilege; inter-rule conflict and collusion)	• Device-cloud connection
App	-		
	-	• Leak hard-coded sensitive information (e.g. OAuth secret)	• App-device communication channel (e.g. co-located app attack) • Sensitive information leakage (e.g. device access token)
Protocol	• Device-Cloud app-level protocol: flawed mechanism; distinguishable messages.		
	-	• OAuth token leakage	• Weak pairing • BLE vulnerabilities • App-device application-level protocol

(Naveed et al., 2014a), similar to the cloud managing the trigger-action rules.

- Device-cloud connection: the communication between the device and the cloud may not well protected since the device is the resource-constrained. The app may manipulate the communication between the device and the cloud, making the device and the cloud out of synchronization.
- Device authentication: the information used by the cloud to authenticate the device can be manipulated by the app, which can cause device impersonation attack, privacy breach, and data injection attack, etc.
- Temporary authorization and revocation: the app may leak sensitive information that enables the temporary user to keep accessing the device after the authorization revoked. The app may not successfully inform the cloud of the authorization revocation, rendering the temporary user access the device stealthily (Ho et al., 2016).
- Cloud log integrity: the app may blind the cloud if it deliberately blocks certain messages.

By far, we have introduced all the three architectures. A comprehensive summary of possible threats of each architecture is shown in Table 2. Despite similarities, the fact that the app may manipulate the communication between the device and the cloud makes the app-in-the-middle different from other architectures, and its security needs to be scrutinized, especially around the role the app plays.

4. Exploring AITM security

In this section, we describe our threat model and the security goals of app-in-the-middle IoT. We start the analysis by limiting the companion app to a software gateway and abstract the *token* concept to ease the analysis. After that, we complicate the role of the app and delve into the app-in-the-middle IoT security.

4.1. Threat model and security goals

Threat model. We assume three types of adversaries.

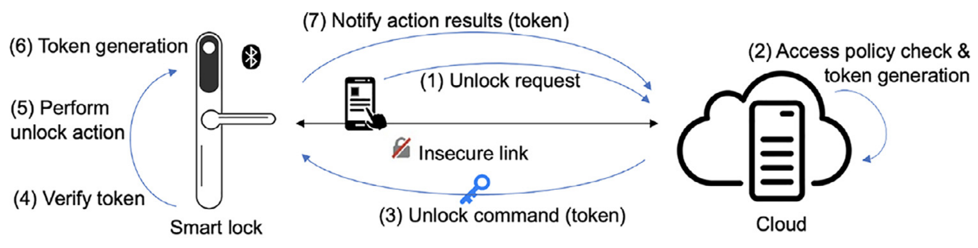
- *Co-located attackers* that co-locates with the device companion app on the Android phone, either controls a malicious app that has been granted with Bluetooth permissions or manipulates the traffic of the companion app.
- *BLE attackers* that locate near the smartphone, with wireless passive and active attack capabilities, including sniffing, jamming, and packet injection.
- *Malicious temporary users* that attempt to maintain access after the authorization is revoked. They have physical access to the device and complete control of the device companion app.
- We consider these adversaries can obtain the companion app and the device of the same type as the victim, and can extract any information from the app and the device they possess, using methods including reverse engineering, firmware extraction, and even runtime debugging; we refer to them *acknowledged adversaries* when emphasizing these abilities.

Considering the component features, functions, and attack surface of AITM IoT summarized in Section 3.3, we expect the system to achieve the following security goals.

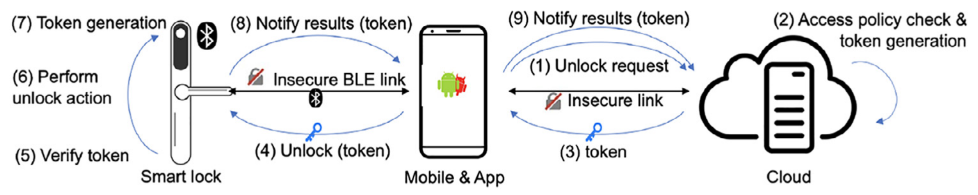
Authentication. The system should provide the cloud with authentication schemes that properly authenticate the device to defeat device impersonation, data stealing, and data injection attacks. The authentication information can be free from being tampered by the co-located and BLE attackers, and can not be utilized by malicious temporary users.

Access control. As mentioned in Section 3.3, co-located attackers can access the device due to the coarse-grained permission the Android system enforces. With adapting the operating system not an option, we consider the authorization scheme should be provided by the app-level design. This authorization scheme should not only defeat co-located attackers but also support secure device sharing, protecting systems from malicious temporary users.

Availability. Temporary authorizations should be guaranteed to be successfully revoked, and the cloud can obtain complete and untampered audit logs and device logs.


(7) Notify action results (token)
(6) Token generation
(2) Access policy check & token generation
(1) Unlock request
(5) Perform unlock action
Insecure link
(4) Verify token
(3) Unlock command (token)
Smart lock
Cloud

(a) Abstract model of simplified AITM IoT, taking smart locks as an example.



(b) Abstract model of AITM IoT, taking smart locks as an example.

Fig. 4. The token-centric abstract model.

4.2. Security analysis of a simplified AITM IoT

To ease the security inspection, we first simplify the discussion by restricting the function of the app. Besides acting as a CITM companion app, the simplified AITM companion app only plays the role of a network-layer gateway that forwards whatever it receives. We conduct security analysis by introducing the token concept. The token reflects properties that can make the scheme meet the security goals.

From the perspectives of a device already in use, what it does is performing actions as the received valid commands say (e.g., a smart lock unlocks the door), reporting information (e.g., a thermometer reports the current temperature), and notifying results (e.g., an oven has been successfully started) to the cloud. Taking a smart lock as an example, it receives an unlock command from an insecure link, verifies it, performs the action, and notifies the operation results, as shown in Fig. 4(a). We argue that every sent and received message must come together with a *token* that bears certain properties sufficient to prove its validity. The reasons are: (a) devices are power limited, messages are infrequent short-burst data rather than stream data for an extended period of time; hence persistent connections are not applicable. (b) Transport layer security is also not an option, because one-way SSL can not provide device authentication and two-way SSL is resource-intensive. As Fig. 4(a) shows, the Bluetooth smart lock communicates with the cloud through the lock companion app, which we consider as a software gateway providing an insecure link. The user issues the unlock command via the companion. The cloud receives the request, checks the access policy to make sure the user is authorized to access the device, generates the token, and responds with a valid command. The lock verifies the command, performs the unlock action, generates the token for notification, and notifies the action results. That is how an already-working simplified AITM IoT works.

The abstracted workflow is depicted around the token, which should hold several properties to defend against attackers and achieve the device-cloud secure communication.

Mutual authentication. The token must reflect the identities of both communication parties that are mutually authenticated.

Unforgeability. The token can not be forged by attackers, as the cloud and the device should be the ones capable of generating tokens.

Resistance to replay attack. The token must contain enough information for the entity to verify its freshness. Those that have been used should not be accepted again.

According to the practical cases we analyzed, tokens that meet the requirements fall into two types:

(a) The response of the challenge-response scheme: as shown in Fig. 5(a), the lock generates a signed challenge, and the cloud responds by signing the challenge together with other information (e.g., the unlock command, DeviceID). The key used for signature generation and verification can be symmetric or asymmetric. They are pre-stored in the device and the cloud as trust anchors (which is necessary for a cryptographically secure scheme). The signature can be used to authenticate the other party, and the token is unforgeable since an attacker has no knowledge of the trust anchor. By guaranteeing the randomness of the challenge and that the device deprecates previous challenges once a new challenge is sent, the replay attack and delayed message attack are prevented.

(b) Commands encrypted by key_{d-c}: with what is shown in Fig. 5(b) as an example, key_{d-c} is the symmetric key shared between the device and the cloud. It can be the symmetric trust anchor or negotiated with the cloud on the basis of the asymmetric trust anchor at the time of the device initialization. The token is generated with the key_{d-c}encrypting necessary information (e.g., the unlock command, counter, timestamp). The token inherits the mutual authentication and unforgeability property from the key_{d-c}. The key_{d-c} can generate multiple tokens. A counter or a timestamp is contained in the token to prevent replay attacks. The random is not applicable here, because it is not previously known to the device, and the device storing all the previously received token is not realistic. The counter on the cloud-side increments its value each time a token is generated. The device updates its counter by setting the value to what contained in the recently received valid token and rejects the token whose counter value is less than the device maintains. If the timestamp is used, a valid duration is preset, and the clock of the two entities are synchronized, with attackers unable to tamper with. The attack window depends on the duration value, which can be carefully set to achieve the balance between security and availability.

A summary of how these two types of token meet the requirements is listed in Table 3.

Discussion. Zhou et al. have revealed that an attacker can fake a device by using the victim device's device ID, which is solely

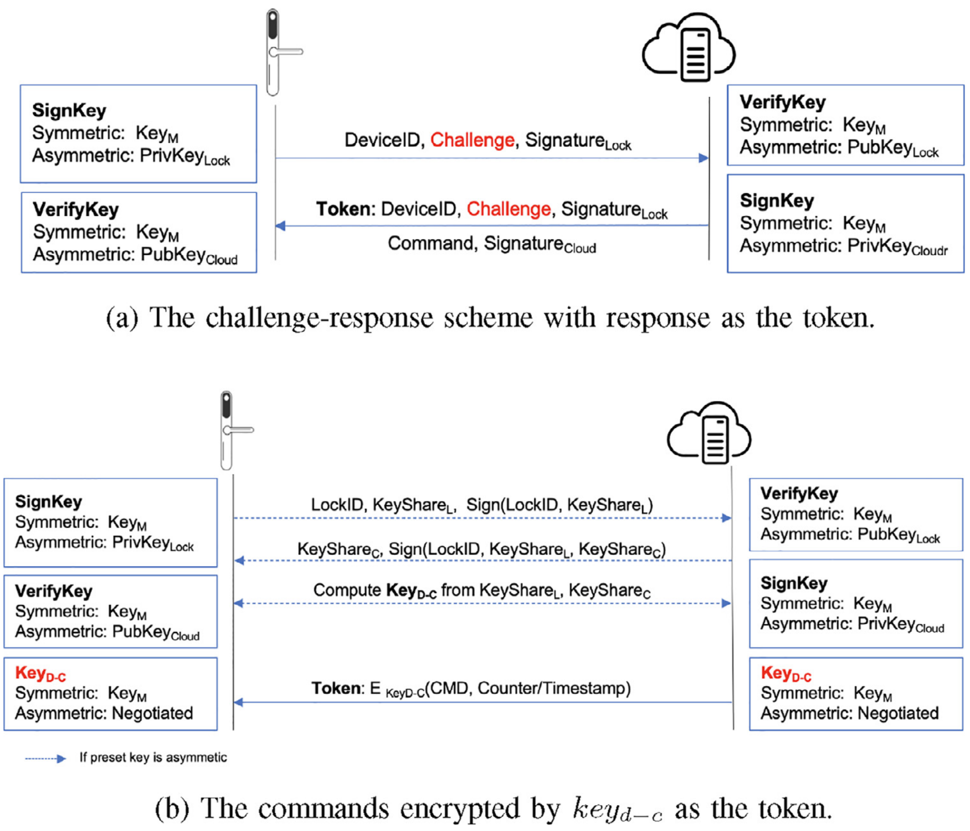

SignKey
Symmetric: Key_M
Asymmetric: PrivKey_Lock
VerifyKey
Symmetric: Key_M
Asymmetric: PubKey_Cloud
DeviceID, Challenge, Signature_Lock
Token: DeviceID, Challenge, Signature_Lock
Command, Signature_Cloud
VerifyKey
Symmetric: Key_M
Asymmetric: PubKey_Lock
SignKey
Symmetric: Key_M
Asymmetric: PrivKey_Cloudr
(a) The challenge-response scheme with response as the token.
LockID, KeyShare_L, Sign(LockID, KeyShare_L)
KeyShare_C, Sign(LockID, KeyShare_L, KeyShare_C)
Compute Key_D-C from KeyShare_L, KeyShare_C
Token: E_KeyD-C(CMD, Counter/Timestamp)
Key_D-C
Symmetric: Key_M
Asymmetric: Negotiated
If preset key is asymmetric
(b) The commands encrypted by key_d−c as the token.

Fig. 5. Two types of token.

Table 3
A brief summary of how the two types of token meet the token requirements.

Scheme	Token properties	Approaches and prerequisites
Challenge-response scheme with response as the token	Mutually authenticated	Both challenge and response are signed with the trust anchor; the trust anchor is unique for each lock and well protected.
	Unforgeable	The challenge is cryptographically random, which means unique and unpredictable; the trust anchor is well protected.
	Resistant to replay attack	The challenge is unique, and the unanswered challenge becomes invalid once a new challenge is issued.
Command encrypted by key_{d-c} as the token	Mutually authenticated	The key_{d-c} is the trust anchor and negotiated from the trust anchor.
	Unforgeable	The used parameters and algorithms are cryptographically secure; the key_{d-c} is not leaked.
	Resistant to replay attack	The counter is maintained properly; the timestamp and valid duration is securely chosen and validated, and the clock is synchronized.

determined by the information that can be collected with little effort. Combined with other flaws, a phantom device can cause attacks such as remote device hijacking, remote device substitution, remote device DoS, and illegal device occupation (Zhou et al., 2019). In this paper, we also deal with these issues as we try to figure out the necessary conditions of a secure device-cloud connection But we abstract the device ID and associated information as the concept of token, and defeat possible attacks by constraining the properties of the token. Through utilizing the token concept, it will be much simpler to analyze the more complicated app-in-the-middle scheme.

4.3. Security of the app-in-the-middle IoT

What brings the complication. In practice, the app can not be regarded merely as a transparent network-layer forwarder. From the perspective of the application-layer device-cloud connection, it is cut into two parts by the app, as shown in Fig. 4(a). The mutual authentication between the app and the cloud, and between the app and the lock is required. The app can establish a secure channel with the cloud and authenticate with each other by adopting the mature scheme in which username/password authentication information is transmitted in a certificate pinning enabled SSL connection. However, the low-level BLE channel can not prevent BLE and co-located attacks; *the mutual authentication between the app and the lock at the app-level is needed*.

Besides, the usage scenarios also affect the role of the app. For those applications like bike-sharing, rental-sharing, the user-device binding relationship is weak. The user authenticates to the cloud to request a token for the temporarily chosen device, and the app forwards the received token to the lock and then finishes the interaction. For applications like smart door locks, the binding relationship is relatively strong. The device belongs to its owner, who may not want the cloud to have the ability of token generation. Hence some critical information is retained in the app instead of the cloud out of privacy concerns. Meanwhile, the owner may

Table 4
The summary of the two application scenarios.

		Weak	Strong
Typical application scenario		Bicycle sharing, hotel keyless check-in	Smart door locks
Features		• One-time access • No sharing • No app-device binding • Users demands high reliability because they are very likely to be charged for using the provided service	• Persistent access • Secure sharing to temporary users • App-device binding • Offline availability • Owners demand partial control out of privacy concern; demand secure sharing and revocable authorization to temporary users
Security goals	Authentication	• App-device authentication (achieved by indirect authentication through token bearing) • Device-cloud authentication (met by mutual authentication property of token)	• App-device authentication (device-app key negotiation via cloud during secure binding) • Device-cloud authentication (token mutual authentication)
	Access control	• Whether user can access device (cloud enforces, authorization by issuing token)	• Whether user can access device (cloud enforces, by referring to device-user binding relationship, and issuing token share) • Whether temporary user can access device (cloud enforces, by referring to user-device-sharedUsers list, and issuing token share; app involves, by issuing token share)
	Availability	• High reliability (vulnerable to relay attack)	• Offline availability (combining app token share and cloud token share)
Attacks		• BLE attacker • Co-located attacker • Acknowledged attacker • Relay attack	• Exfiltration of information stored in app • Malicious temporary user attacker

demand the function of securely sharing the lock to a temporary user with the help of the cloud.

Different scenarios involve different working processes and different security requirements. For example, the criteria for judging whether a smart TV is secure or not depends on the environment in which it is located. If the TV is in a home environment, it can be accessed by users in the same local area network to realize functions like screencasting. But if it is in a public environment, such as being used by businesses to play promotions and advertisements, then it should be accessible to no one but the administrator. Similarly, for devices in the app-in-the-middle architecture, their security requirements should also change according to different application scenarios. Therefore, we summarize two typical types of application scenarios of the app-in-the-middle architecture, and discuss the security separately (Table 4).

4.3.1. *Weak user-device binding relationship*

This kind of relationship indicates a typical application scenario where the cloud authorizes its authenticated users the one-time access to its resources, such as bicycle sharing, car sharing, and hotel keyless check-in, etc. Therefore, no app-device binding process is involved, and no device sharing since all users are temporarily authorized. Besides the goal of establishing a secure device-cloud connection, high reliability is demanded by the user who is very likely to be charged for using the provided resource. It needs to be guaranteed that the chosen resource is successfully authorized to the authenticated user, and the authorization result is correctly reported to the cloud.

Each time a user uses the service, the user finds an available resource offline, authenticates himself to the service provider (the cloud), and requests authorization for the resource on the app. Generally, the working process, with that shown in Fig. 6 as an example, includes (a) the app authenticating itself to the cloud, (b) the app acquiring the lock ID from the out-of-band channel and requesting the corresponding MAC address from the cloud, (c) the app-building BLE channel with the lock, (d) the app requesting a resource (the lock) of the cloud, (e) the cloud authorizing the usage by issuing the token to the user, the app issuing unlock commands together with the token, and (f) the app reporting the authorization result to the cloud. The procedure of token generation and verification (step (d) and (e)) inherits the schemes proposed in the simplified version. Therefore, secure device-cloud connection at the app level is achieved. The app-device authentication is achieved indirectly under the help of the cloud since they share no secret beforehand. The device trusts whoever sends the valid token since it counts on the cloud to have correctly done the user authentication and authorization the cloud would issue the token only when the user passes the access policy check.

The three properties of the tokennamely mutual authentication, unforgeability, and resistance to replay attackpartly meet the security goals proposed. The device authenticates the app through token ownership. Therefore, it can defeat co-located attackers, since the co-located app not possessing the token can not control the device. The reliability is not achieved since the relay attack can not be prevented. The app identifies the device through the ID and MAC address mapping relationship. The device ID information is acquired from the out-of-band channel, for example, a QR code or a string of numbers attached to the device. The corresponding MAC address is obtained from the cloud. The app establishes Bluetooth connection with the device that is identified by the MAC address. The relay attack breaks the user's assumption of the device being physically present, which can not be handled at the app-level. A practical case is demonstrated in Section 5.3.2. As for availability, the results of the action are not guaranteed to be correctly reported to the cloud. As the synchronization of the device and the cloud relies on the mobile gateway, the cloud may not receive the status report after a token is issued if the user deliberately shut down the network or the network simply fails. Therefore, the cloud has to adopt the policy that considers the authorization is successful by default unless a fresh status report message indicating the device status is received.

4.3.2. *Strong user-device binding relationship*

This kind of relationship indicates a typical application scenario where the device belongs to the user. The user demands privacy from the cloud and wants to be in a dominant position when controlling the device. Therefore, the cloud transfers part of its token generation capability to the mobile app instead of possessing the full capability of managing the device as in the weak binding scenario. The user leverages the capability of the cloud, such as

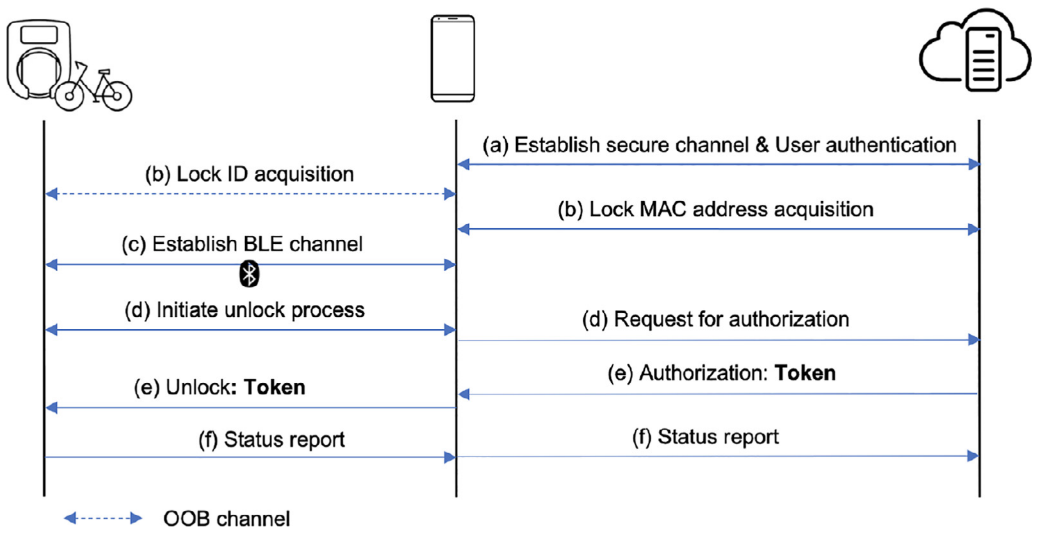

(a) Establish secure channel & User authentication
(b) Lock ID acquisition
(b) Lock MAC address acquisition
(c) Establish BLE channel
(d) Initiate unlock process
(d) Request for authorization
(e) Authorization: **Token**
(e) Unlock: **Token**
(f) Status report
(f) Status report
OOB channel

Fig. 6. The working process of an app-in-the-middle unlocking scheme that is used when the weak binding relationship between the user and the device is required.

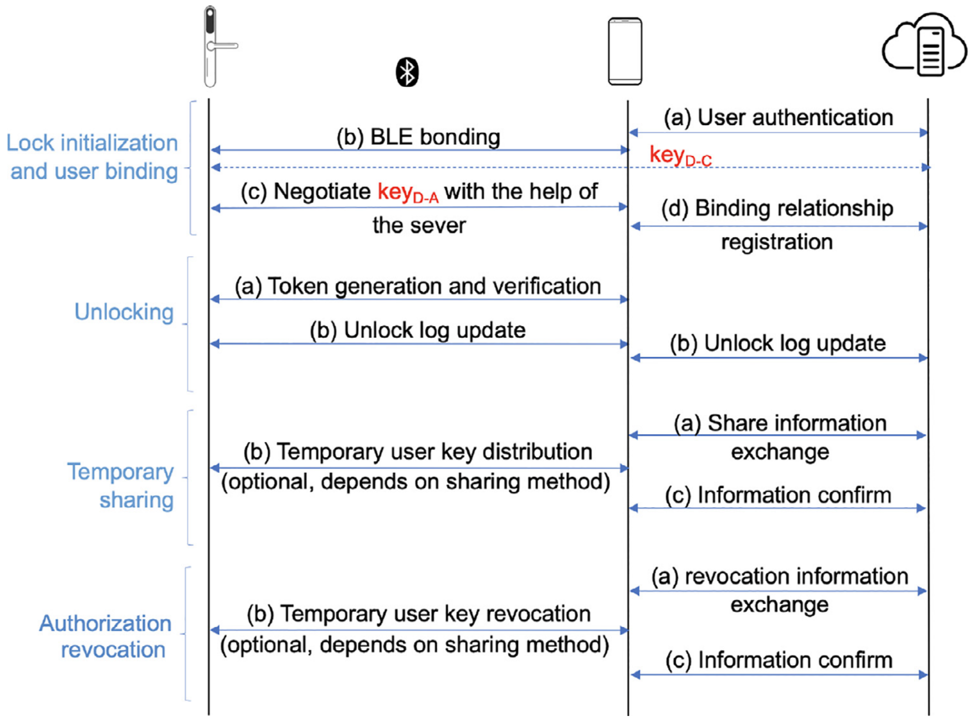

Lock initialization and user binding
(a) User authentication
(b) BLE bonding
$key_{D\text{-}C}$
(c) Negotiate $key_{D\text{-}A}$ with the help of the sever
(d) Binding relationship registration
Unlocking
(a) Token generation and verification
(b) Unlock log update
(b) Unlock log update
Temporary sharing
(a) Share information exchange
(b) Temporary user key distribution (optional, depends on sharing method)
(c) Information confirm
Authorization revocation
(a) revocation information exchange
(b) Temporary user key revocation (optional, depends on sharing method)
(c) Information confirm

Fig. 7. The working process composition of the scheme that is used in the user-device strong binding relationship.

maintaining access logs, authorizing, and revoking the access of temporary users, making it easier and more convenient to manage the device. Besides, the user demands high usability of the device. The device should function normally when the user encounters temporary network failure. For example, the user should still be able to unlock the door if the app has no Internet connection for a short period of time. We name this requirement as *offline availability*.

The complete working flow includes device initialization and user binding process, normal usage process, temporary sharing, and authorization revocation process, as shown in Fig. 7, with a smart door lock as an example. The device is initialized and bond with the user on first use, and controlled by the app during normal use. It can be temporarily shared with another user, whose authorization can be revoked at any time. During the device initialization and owner registration process, the app establishes a BLE bonding connection and negotiates a secret key (we denote it as key_{d-a}) with the device with the help of the cloud. Eventually, the user is registered as the owner of the lock and ready to control the lock with the key_{d-a} being the cornerstone of security.

We consider secrets that are stored in the app can be extracted using a combination of smartphone vulnerability exploits, malware, malicious third-party apps, etc. Meanwhile, the strong user-device binding relationship scenario requires the cloud not to possess complete device controlling capability out of privacy concerns. Therefore, the secure design would be that the token generation involves both the locally stored secrets in the app and ones received from the cloud. This design prevents the attacker from retrieving the critical information (for example, the key_{d-a}) from

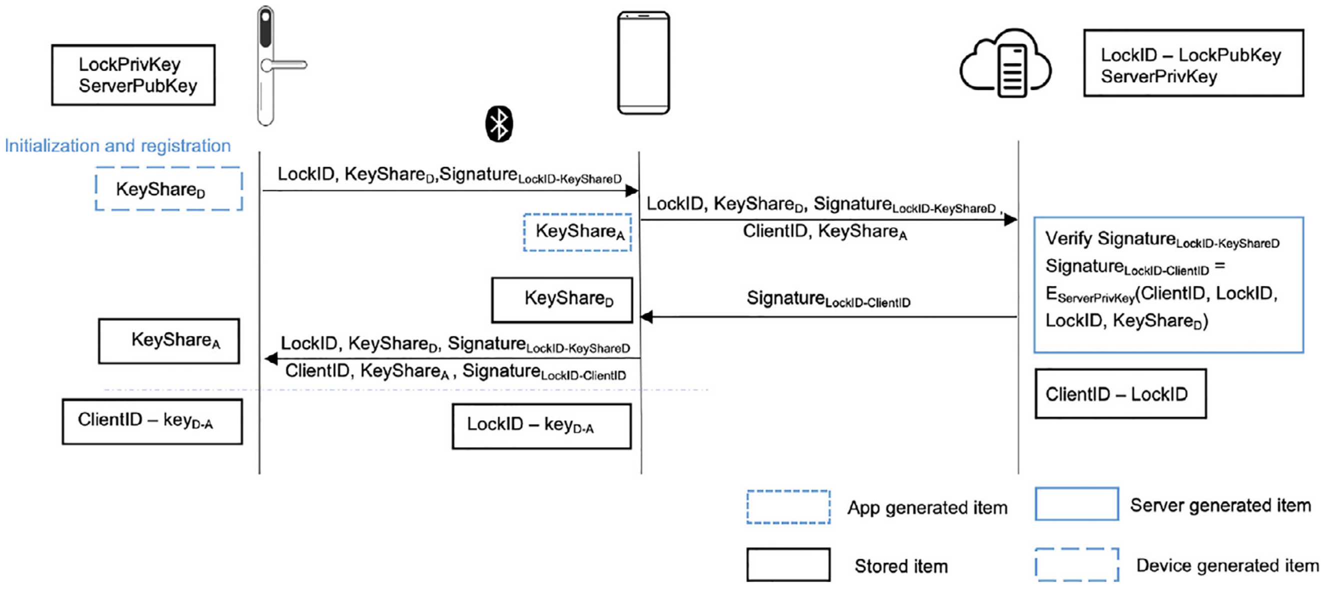

LockPrivKey
ServerPubKey
LockID – LockPubKey
ServerPrivKey
Initialization and registration
KeyShare_D
LockID, KeyShare_D, Signature_LockID-KeyShareD
LockID, KeyShare_D, Signature_LockID-KeyShareD, ClientID, KeyShare_A
KeyShare_A
Verify Signature_LockID-KeyShareD
Signature_LockID-ClientID = E_ServerPrivKey(ClientID, LockID, LockID, KeyShare_D)
KeyShare_D
Signature_LockID-ClientID
KeyShare_A
LockID, KeyShare_D, Signature_LockID-KeyShareD
ClientID, KeyShare_A, Signature_LockID-ClientID
ClientID – LockID
ClientID – key_D-A
LockID – key_D-A
App generated item
Server generated item
Stored item
Device generated item

Fig. 8. A secure scheme to negotiate key_{d-a}.

the app hence endangering the whole system whatever secure the other part of the design is. At the same time, the cloud does not have full control of the device since the critical key_{d-a} is retained in the app.

A secure design. We describe the design in three stages according to the working process shown in Fig. 7. Same as the simplified design, the pre-shared root key (symmetric or asymmetric) between the device and the cloud is the security cornerstone of the whole design.

- In the device initialization and user binding process, the device and the mobile launch a BLE bonding process and establishes a secure BLE channel, in which the device and the app further negotiate a shared secret key key_{d-a} with the help of the cloud, as shown in Fig. 8. The app informs the cloud the device-app binding relationship. A secret key key_{d-c} is either previously shared between the cloud and the device (the root key) or negotiated via the app during this process.
- In the unlocking process, the token generation is the main concern. The token can be time-based and counter-based, which aims to prevent token replay attack. If the time-based scheme is adopted, the cloud receives the unlock request from the app and returns a token share $Share_c = TOTP(key_{d-c}, C_T)$, $C_T = \lfloor \frac{T-T_0}{T_x} \rfloor$ where T is the current time, T_0 is the initial time and T_x is the valid time span that both the lock and the cloud has agreed on. Otherwise, if the counter-based scheme is adopted, the token share $Share_c$ is determined by the counter value, that is, $Share_c = TOTP(key_{d-c}, C_{Value})$. The app generates the token by taking the $Share_c$, its token share $Share_a$ (key_{d-a}) and the challenge received from the lock or the predefined unlock command as input.

  $$Token = F(Share_a, Share_c, CMD/Challenge)$$
- In the temporary sharing and authorization revocation process, the app notifies the cloud which user will be temporarily authorized and generates $Share_{a'}$ for the temporary user. The temporary user can receive the $Share_c$ from the cloud if the authorization is still valid. The owner can revoke the authorization at any time by simply notifying the cloud to stop issuing token shares to the temporary user.

The security of the key_{d-a} is essential. The device and the app negotiate key_{d-a} in the lock initialization and user binding process. The device and the app must be mutually authenticated with the help of the cloud. That is, the app confirms the identity of the device by trusting the cloud to verify the signature of the cloud, which is $Signature_{LockID-KeyShare_D}$ in the example, and the lock confirms the identity of the app by trusting the cloud to authenticate the app, with the app having access to the signature signed by the cloud (that is, $Signature_{LockID-ClientID}$)as a piece of evidence. Therefore, the key_{d-a}, which derives from the $KeyShare_D$ and $KeyShare_A$, bears the property of the app-device mutual authentication. The negotiation of key_{d-c} is based on the existed trust between the lock and the cloud, and its security is not affected by the insecure communication channel.

The offline availability is achieved. The token is mainly generated from the two token shares $Share_c$ and $Share_a$. The $Share_a$ assures that the app participates in the token generation, and the cloud can not control the lock without the knowledge of the user. The time-based $Share_c$ introducing an attribute of token validity period makes the token acceptable only in a predefined period of time. This attribute is determined by the cloud such that temporary user authorization and revocation can be enforced securely by the cloud. Meanwhile, the token validity period makes the app can work offline as long as the $Share_c$ is not overdue. The counter-based $Share_c$ facilitates the app to cache a few token shares ahead of time and hence making the offline availability realizable.

However, when it comes to the temporary user authorization and revocation, the temporary user may act in a malicious way by caching a tile of token shares. Even if the user has notified the cloud to revoke the authorization, the cloud has no reliable approach to inform the lock that the pre-issued token shares (counter value) are invalid, in the circumstances that the temporary user is the only one that may get approach to the lock for a period of time. A scheme combining the time-based $Share_c$ and counter-based $Share_c$ can effectively control the attack window.

4.4. Discussion

We use the token concept to ease the analysis of the security of AITM IoT. The properties of the token are closely related to the security goals that AITM IoT must achieve. Notably, Aman et al. (2018) also adopted a token-based scheme, featuring its dynamic energy-security level, i.e., flexible key size, when authenticating IoT devices. In their work, the authentication is based on the physically unclonable functions (PUFs), which makes the scheme free from storing secrets, hence resistant to physical attacks. The protocol leverages the OAuth 2.0 token together with different key sizes to achieve energy-security scaling. And formal verification of the correctness and security analysis of the protocol is presented. In contrast, the token in our paper is more like an abstract concept. We state properties a secure scheme should meet, without specifying a concrete implementation. We exclude physical attackers from our threat model and take the pre-installed key as trust anchors, considering most existing consumer IoT devices are incapable of defending this kind of attack. However, despite these differences, the token-based protocol is of the same purpose as the secure design we proposed in Section 4.3.2. The formal verification of the correctness and security analysis presented can be seen as proof of the realizability of our proposed secure design. If we treat the app as another device that needs to establish a secure connection to the device, we can seamlessly implant the token-based protocol into our work to authenticate the entities and negotiate key_{d-a}. The token-based scheme proposed in Aman et al. (2018) can be seen as one of the secure implementations that apply for the scenario we talk about.

For practical AITM IoT systems to be secure, the cryptographic operations can not be circumvented. The device must correctly verify the received token. For devices, lightweight ciphers such as elliptic curve cryptography (ECC), Simon, and Speck can be adopted to meet the resource constraints. The time required for ECC scalar multiplication, one-way hash function, and symmetric encryption are 17.1 ms, 0.32 ms, and 5.6 ms, respectively (Sadhukhan et al., 2020). We can compute the computational overhead by identifying the primitive operations used in the secure scheme. Moreover, from the cases we studied in Section 5.3, some devices have already applied cryptographic algorithms (but in the wrong way). For these devices, the overhead of deploying a secure solution is negligible. All changes in the scheme are easy for the app to apply since the update can be easily delivered. The firmware can be updated through the app.

5. Practical analysis and issues

In this section, we introduce rules that must be met, and the analysis approach we use to analyze the real world AITM IoT. Then we present case studies, showing that violating these rules can cause unauthorized access, information leakage, etc.

5.1. Security rules

We depict rules that for weak binding systems and strong binding systems.

- Rule W#1: Relay attack is essentially unsolvable by app-level schemes. Additional out-of-band information should be considered to confirm the presence of the device.
- Rule W#2: Tokens should be counter&timestamp-based or challenge-response based. Otherwise, they can be replayed. If challenge-response based, the device must only accept the recently issued challenge to defeat the delayed message attack. For counter-based schemes, the device must keep the freshest counter value in the record and reject tokens with previous counter values
- Rule W#3: Tokens should be generated at the cloud. Otherwise, the acknowledged attacker can use the app code logic and arbitrarily access the device.
- Rule S#1: The device and the app must be mutually authenticated with the help of the cloud during the device initialization and user binding process. That is, the key_{d-a} bears mutual authentication.
- Rule S#2: The key_{d-a} must not be exposed on the BLE channel.
- Rule S#3: The key_{d-a} must not be exposed to the temporary users.
- Rule S#4: There is no secure way to simultaneously achieve the goal of offline availability and strictly revocable authorization.

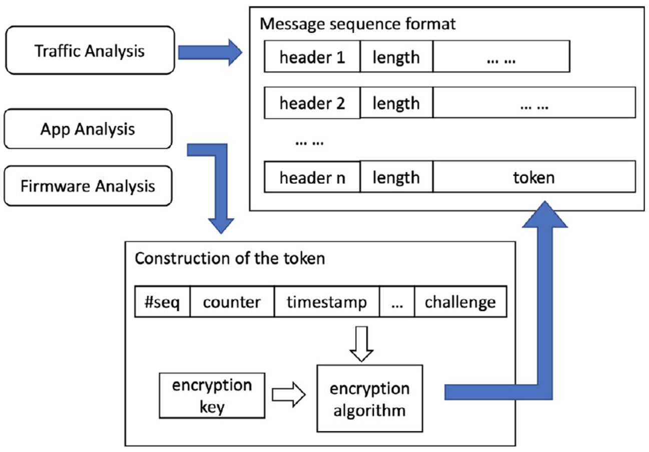


Fig. 9. Analyzing the token construction.

5.2. Analysis approach

We conduct the analysis from two aspects: the construction of the token and the scheme used to implement temporary authorization if the functionality exists. We start with the BLE traffic and the app and turn to the device firmware for extracting the semantics of the token generation if necessary, as shown in Fig. 9.

5.2.1. Traffic analysis

Tokens are doomed to appear in the BLE traffic. We start by capturing and analyzing the BLE traffic. The traffic may or may not be encrypted, which depends on whether the BLE pairing process is initiated by the app. The class of encryption is determined by the input/output capability of the app and the device. Considering the power constraint and the convenience of use (pairing needs to press the button or compare the number displayed on the lock and the app), usually pairing of class 0 or no pairing is used in practice. The pairing of class 0 uses a fixed key that we can use to decrypt the traffic, which means no encryption protection at the link level. We use differential traffic analysis to help distinguish the fixed part from the random part of the command.

5.2.2. App analysis

To recover the generation process of the token, we need to extract relevant information from the app. For Android apps, we use Appspear (Yang et al., 2015) to unpack the app in case that it is packed or obfuscated. We dynamically instrument some specific classes and monitor the parameter of the *writeCharacteristic* API. Once the parameter contains the fixed part of the command, we can locate the random part and forwardly slice the trace. The challenge is how to recover the semantics of the operation of the token. The generation of token involves encryption. The encryption key can be observed in the trace.

5.2.3. Firmware analysis

For those systems in which the app forwards messages back and forth between the cloud and the lock, the key information can only be obtained from the firmware. Besides the obtainment of the firmware and decompiling the code, the challenge of the firmware analysis lies in three aspects: (1) locate the code that handles the token; (2) identify the cryptographic algorithms that are used to generate or verify the token; (3) recover the full logic of token generation and verification.

5.2.4. Vulnerability evaluation

At the traffic level, we replay the commands at different intervals to test if the device accepts the reused token and obtain the token timeout information. We concentrate on app analysis since the information obtained from the traffic is limited and the difficulty in firmware acquisition. We inspect the traces to determine if cryptographic misuses exist, for example, to check whether a hard-coded key is used to encrypt the token share. If the server share is found to be part of the token, we reuse it multiple times to find out if the server has a mechanism to limit the lifespan of the share. For the strong binding scenario, we focus on the device initialization stage that negotiates key_{d-a}. If no authentication check (e.g., signature verification) instructions are found in the trace before the point where the key is used for encryption, we consider it vulnerable to man-in-the-middle attacks. If the temporary authorization function exists, we dynamically instrument both of the app running with the owner account and a temporary user account to obtain the log of the entire process to assess if the token is revocable. By comparing two traces, we check whether the owner's key_{d-a}, which is found in one trace, is also used in the other trace.

The analysis is semi-automatic, relying on the analysts' involvement to set up the device, interact with the app, and configure instrumentation rules. Automatic testing tools like The Monkey (yearinfo) are available to trigger interactions. However, it will increase the size of the trace, which makes the analysis of the trace difficult. Likewise, we could use preset instrumentation rules to improve automation, but it will reduce the accuracy of the trace. The automation is also limited when identifying vulnerabilities from the trace. Inferring the semantic of each component that impacts the token (for example, identifying the ECDH key agreement protocol and the negotiated key from the trace) is nontrivial. Nonetheless, well-studied cryptographic misuses (such as hard-coded key, non-random IV, and short key sizes) can be automatically identified from the trace. The instrumented app runs about two times slower to generate the trace, which is better than Android's debugging infrastructure (Li et al., 2014).

5.3. Case studies

5.3.1. Weak binding #1

The application scenario is motor-bike sharing. The unlock command (token) is a combination of fixed strings hard-coded in the app. It violates rule W#2 by not implementing any scheme to defeat replay attacks. Therefore, BLE attackers and co-located attackers can sniff the token, and acknowledged attackers can reuse the app logic to infer the token. It violates rule W#3 by completely generating the token at the app-side, hence acknowledged attackers could reuse the app logic to generate tokens at will.

5.3.2. Weak binding #2

This is a bicycle sharing application. The user scans the QR code on the bike, authenticates to the cloud, and requests the corresponding token. The token is the challenge-response type that meets all the security requirements. But it is vulnerable to the relay attack, which is the fundamental flaw that can not be tacked by app-level security schemes. The attacker can put the QR code of Bike A on Bike B and set a cloned device with the MAC address of A near B. The user scans the fake QR code, obtains the token for A, connects the cloned device, and sends the token. Once the cloned device received the token, it transmits the token to the attacker, and the attacker can use the token to unlock Bike A. The user is charged for obtaining the token, but the device he intends to use is still locked. This case violates the rule W#1 since no extra information is provided for the user to confirm the presence of the communicating device.

5.3.3. Weak binding #3

The system adopts a counter-based token. Because of the violation of rule #2, BLE attackers, who jam the BLE connection and sniff the token, can keep the token for later use. The token is valid until the device receives a valid token since the counter is not used together with the timestamp that defines a valid duration.

5.3.4. Strong binding #1

The device is a smart door lock. In the lock initialization and account binding process, the lock and the app perform ECDH key exchange via the BLE channel, which generates a shared key between them. The shared key is stored by the lock and sent to the cloud by the app. The cloud binds the lock id, user id, and the shared key together.

A complete unlocking process begins with the app sending a wakeup signal to the lock. The lock sends back a challenge, and the app acquires the shared key from the cloud and generates the token by encrypting the challenge with the shared key. The lock received the token and decrypted it with the stored shared key. If the results match the challenge the lock sent before, the unlock action is performed. The lock is a door lock that is widely used in short-term rental housing. The landlord needs to temporarily authorize a tenant to unlock. The temporary authorization relies on the cloud to do access control. The cloud checks the tenant account. If it is authorized, the cloud will accept the challenge forwarded by the tenant app and generates the token. The tenant app receives the token and uses it to unlock the device. The tenant app acts as the forwarding point for the whole time.

In this case, the secret key plays the role of key_{d-a}. However, its generation process involves no device-app authentication, which violates rule S#1, hence is vulnerable to the MITM attack. A BLE attacker can launch the MITM attack when the lock is initialized and completely controls the device.

5.3.5. Strong binding #2

In the lock initialization and account binding process, the lock sends a *channelpwd* to the app via the BLE channel. The *channelpwd* is a secret value that is hard-coded in the lock. The cloud receives the lock id, user id, and *channelpwd* from the app and stores the binding relationship. The challenge-response mode is used in this locking system. The lock generates a random string and sends it to the app. The app uses *channelpwd*, which is acquired from the cloud, to encrypt the random string. The encryption result is used as a token. In the temporary authorization situation, the cloud sends the *channelpwd* directly to the authorized temporary user. The temporary user conducts the unlocking process as the owner does.

The secret key *channelpwd* is hard-coded in the lock and transmitted to the app via the BLE channel without any protection, which violates rule S#2, resulting in the attacker capable of unlocking the lock at any time. It also violates rule S#3, causing the irrevocable authorization.

6. Related work

Cloud-in-the-middle IoT security. Zhou et al. (2019) studied the interactions between clouds, devices, and applications, and discovered several new vulnerabilities by inspecting state transitions. OConnor et al. (2019) revealed that the always-responsive and on-demand messages sent between the device and the cloud are independent and distinguishable, which endangers the integrity and availability of the information the device provides. Chen et al. (2019b) conducted a security analysis of remote binding using a state-machine model. Chen et al. (2019a) revealed that attackers in the vicinity could compromise the local binding. Li et al. (2018) showed the flawed WiFi provisioning scheme leaks WiFi credentials to local wireless attackers.

Trigger-action platform IoT security. Tian et al. (2017) guarantees that the capabilities and called APIs requested by an app actually align with the app description. Fernandes et al. (2016a) reveals vulnerabilities due to over-privileged smart home applications. Fernandes et al. (2016b) introduces a solution that forces apps to make their data use patterns explicit and then enforce the declared information flows while preventing all other flows. Rahmati et al. (2018) proposes a risk-based permission model in smart homes. Ding and Hu (2018) reveals that smart home devices may interact with each other through shared physical environments. Celik et al. (2018b) leverages static analysis of smart applications to detect possible physical channel interactions between multiple home automation applications (IoT apps). Celik et al. (2019) detects when an individual app and interactions among apps lead to unsafe and insecure states. Valente and Cardenas (2017) provides contextual integrity to the permission granting process of IoT apps. Celik et al. (2018a) proposes a static taint analysis tool called SAINT for tracing sensitive data flows in IoT applications.

App-in-the-middle IoT security. Ho et al. (2016) analyzed smart locks that are the DGC (Device-Gateway-Cloud) architecture and indicate the state consistency attack due to this architecture, which enables irrevocable control of the lock. Xu et al. (2019) studies the problem that malicious devices use specific profiles to break the security of Android and threaten the privacy of users due to failure to authenticate the profiles provided by the device. Naveed et al. (2014b) discusses the Android platform's flaws in the design and control of the external resource's permissions, which causes unauthorized apps to access the external device, and proposes a corresponding solution, DaBinder. Sivakumaran and Blasco (2019) reveals that other applications on the phone can unauthorized access (read and write) the data of the paired protected BLE device. Demetriou et al. (2017) proposes a new technique that achieves fine-grained, situation-aware access control of IoT devices over a home area network. Cristalli et al. (2019) studies the problem of mutual authentication between two apps running on two different devices and communicating over a short-distance channel.

7. Conclusion

We proposed a new IoT architecture named app-in-the-middle IoT and adopted a methodology that builds an abstract model and extracts a concept called *token* to analyze its security. Application scenarios significantly influence the role of the app, hence affecting how the properties of the token are implemented. We respectively inspected the generation and distribution of the token in different usage scenarios, and proposed the prerequisites that must be met to be secure, based on which several security rules is given. Violations of these rules may cause consequences such as unauthorized access, information leakage, irrevocable authorization, and device hijack, as shown by the cases we have analyzed.

Declaration of Competing Interest

There are no interests to declare.

CRediT authorship contribution statement

Hui Liu: Conceptualization, Methodology, Investigation, Writing - original draft, Writing - review & editing. **Juanru Li:** Conceptualization, Resources, Writing - original draft. **Dawu Gu:** Supervision, Project administration, Funding acquisition.

Acknowledgments

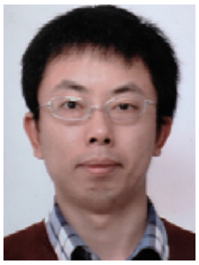
This work was partially supported by the Key Program of National Natural Science Foundation of China (Grant No. U1636217), the General Program of National Natural Science Foundation of China (Grant No. 61872237), and the Major Project of the Ministry of Industry and Information Technology of China (Grant No. 2018-36). We especially thank Ant Financial Services Group for the support of this research within the SJTU-AntFinancial Security Research Centre.

References

Aman, M.N., Taneja, S., Sikdar, B., Chua, K.C., Alioto, M., 2018. Token-based security for the internet of things with dynamic energy-quality tradeoff. IEEE Internet Things J. 6 (2), 2843–2859.
Celik, Z.B., Babun, L., Sikder, A.K., Aksu, H., Tan, G., McDaniel, P., Uluagac, A.S., 2018a. Sensitive information tracking in commodity IoT. In: Proceedings of the 27th {USENIX} Security Symposium ({USENIX} Security 18), pp. 1687–1704.
Celik, Z.B., McDaniel, P., Tan, G., 2018. Soteria: Automated IoT safety and security analysis. In: Proceedings of the 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), pp. 147–158.
Celik, Z.B., Tan, G., McDaniel, P.D., 2019. Iotguard: Dynamic enforcement of security and safety policy in commodity IoT.. In: Proceedings of the 2019 NDSS.
Chen, J., Sun, M., Zhang, K., 2019. Security analysis of device binding for ip-based IoT devices. In: Proceedings of the 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE, pp. 900–905.
Chen, J., Zuo, C., Diao, W., Dong, S., Zhao, Q., Sun, M., Lin, Z., Zhang, Y., Zhang, K., 2019b. Your IoTs are (not) mine: On the remote binding between IoT devices and users. In: Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, pp. 222–233.
Cristalli, S., Lu, L., Bruschi, D., Lanzi, A., 2019. Detecting (absent) app-to-app authentication on cross-device short-distance channels. In: Proceedings of the 35th Annual Computer Security Applications Conference. ACM, pp. 328–338.
Demetriou, S., Zhang, N., Lee, Y., Wang, X., Gunter, C.A., Zhou, X., Grace, M., 2017. Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps. In: Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 122–133.
Ding, W., Hu, H., 2018. On the safety of IoT device physical interaction control. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 832–846.
Fernandes, E., Jung, J., Prakash, A., 2016a. Security analysis of emerging smart home applications. In: Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP). IEEE, pp. 636–654.
Fernandes, E., Paupore, J., Rahmati, A., Simionato, D., Conti, M., Prakash, A., 2016b. Flowfence: Practical data protection for emerging IoT application frameworks. In: Proceedings of the 25th {USENIX} Security Symposium ({USENIX} Security 16), pp. 531–548.
Fernandes, E., Rahmati, A., Jung, J., Prakash, A., 2018. Decentralized action integrity for trigger-action IoT platforms. In: Proceedings of the 2018 Network and Distributed System Security Symposium.
Fouladi, B., Ghanoun, S., 2013. Security evaluation of the z-wave wireless protocol. Black hat USA 24, 1–2.
Ho, G., Leung, D., Mishra, P., Hosseini, A., Song, D., Wagner, D., 2016. Smart locks: Lessons for securing commodity internet of things devices. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. ACM Press, New York, New York, USA, pp. 461–472.
Jasek, S., 2017. Blue Picking – Hacking Bluetooth Smart Locks. Hack In The Box.
Jia, Y., Xing, L., Mao, Y., Zhao, D., Wang, X., Zhao, S., Zhang, Y.,. Burglars IoT paradise: Understanding and mitigating security risks of general messaging protocols on IoT clouds. In: Proceedings of the IEEE Symposium on Security and Privacy (SP), pp. 838–854.
Li, C., Cai, Q., Li, J., Liu, H., Zhang, Y., Gu, D., Yu, Y., 2018. Passwords in the air: Harvesting Wi-Fi credentials from SmartCfg provisioning. In: Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks, pp. 1–11.
Li, J., Yang, W., Shu, J., Zhang, Y., Gu, D., 2014. Indroid: An Automated Online Analysis Framework for Android Applications. Crisis Intervention Team (CIT).
Naveed, M., Zhou, X., Demetriou, S., Wang, X., Gunter, C.A., 2014a. Inside job: understanding and mitigating the threat of external device mis-bonding on android. In: Proceedings of the Network and Distributed System Security Symposium. Internet Society, Reston, VA.

Naveed, M., Zhou, X.-y., Demetriou, S., Wang, X., Gunter, C.A., 2014b. Inside job: Understanding and mitigating the threat of external device mis-binding on android. In: Proceedings of the 2014 NDSS.
OConnor, T., Enck, W., Reaves, B., 2019. Blinded and confused: uncovering systemic flaws in device telemetry for smart-home internet of things. In: Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks, pp. 140–150.
Pereira, C., Aguiar, A., 2014. Towards efficient mobile M2M communications – Survey and open challenges. Sensors 14(10), 19582–19608.
Rahmati, A., Fernandes, E., Eykholt, K., Prakash, A., 2018. Tyche: A risk-based permission model for smart homes. In: Proceedings of the 2018 IEEE Cybersecurity Development (SecDev). IEEE, pp. 29–36.
Ryan, M., 2013. Bluetooth: With low energy comes low security. In: Proceedings of the 7th {USENIX} Workshop on Offensive Technologies.
Sadhukhan, D., Ray, S., Biswas, G., Khan, M., Dasgupta, M., 2020. A lightweight remote user authentication scheme for IoT communication using elliptic curve cryptography. J. Supercomput..
Santos, J., Rodrigues, J.J.P.C., Silva, B.M.C., Casal, J., Saleem, K., Denisov, V., 2016. An IoT-based mobile gateway for intelligent personal assistants on mobile health environments. J. Netw. Comput. Appl. 71, 194–204.
Seol, S., Shin, Y., Kim, W., 2015. Design and realization of personal IoT architecture based on mobile gateway. Int. J. Smart Home 9 (11), 133–144.
Sivakumaran, P., Blasco, J., 2019. A study of the feasibility of co-located app attacks against {BLE} and a large-scale analysis of the current application-layer security landscape. In: Proceedings of the 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 1–18.
Telnet backdoor vulnerabilities impact over a million IoT radio devices, https://www.zdnet.com/article/critical-vulnerabilities-impact-over-a-million-iot-radio-devices/. 2019. Accessed: 2020-03-20.
Tian, Y., Zhang, N., Lin, Y.-H., Wang, X., Ur, B., Guo, X., Tague, P., 2017. Smartauth: User-centered authorization for the internet of things. In: Proceedings of the 26th {USENIX} Security Symposium ({USENIX} Security 17), pp. 361–378.
Ui/application exerciser monkey. https://developer.android.com/studio/test/monkey. 2020. Accessed: 2020-03-20.
Valente, J., Cardenas, A.A., 2017. Security & privacy in smart toys. In: Proceedings of the 2017 Workshop on Internet of Things Security and Privacy, pp. 19–24.
Xu, F., Diao, W., Li, Z., Chen, J., Zhang, K., 2019. Badbluetooth: Breaking android security mechanisms via malicious bluetooth peripherals.. In: Proceedings of the 2019 NDSS.
Yang, W., Zhang, Y., Li, J., Shu, J., Li, B., Hu, W., Gu, D., 2015. Appspear: Bytecode decrypting and dex reassembling for packed android malware. In: Proceedings of the International Symposium on Recent Advances in Intrusion Detection. Springer, pp. 359–381.
Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., Dutta, P., 2015. The internet of things has a gateway problem. In: Proceedings of the 16th International Workshop. ACM Press, New York, New York, USA, pp. 27–32.
Zhou, W., Jia, Y., Yao, Y., Zhu, L., Guan, L., Mao, Y., Liu, P., Zhang, Y., 2019. Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms. In: Proceedings of the 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 1133–1150.

Hui Liu is currently a Ph.D. candidate in the Department of Computer Science and Engineering at Shanghai Jiao Tong University. She received her B.S. degree from East China University of Science and Technology in 2012. Her research interests include IoT (Smart Homes) security and software security.



Juanru Li is currently a postdoc at Shanghai Jiao Tong University. He received his Ph.D. degree from Shanghai Jiao Tong University in 2019. His research interests include developing secure crypto software and systems, and hardening existing crypto software against evolving security threats.

Dawu Gu is a professor at Shanghai Jiao Tong University in Computer Science and Engineering Department. He received his Ph.D. degree from Xidian University in 1998. He is the director of the Lab of Cryptology and Computer Security (LoCCS). His research interests include cryptography, software security, hardware and embedded security, big data and cloud security with privacy, financial security techniques, etc.