future internet

Article

A Security Framework for the Internet of Things in the Future Internet Architecture

Xiruo Liu [1], Meiyuan Zhao [2], Sugang Li [3], Feixiong Zhang [3] and Wade Trappe [3,*]

1 Intel Labs, Hillsboro, OR 97124, USA; xiruo.liu@intel.com
2 Google, Mountain View, CA 94043, USA; meiyuanzhao@gmail.com
3 Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854, USA; sugangli@winlab.rutgers.edu (S.L.); feixiong@winlab.rutgers.edu (F.Z.)
* Correspondence: trappe@winlab.rutgers.edu; Tel.: +1-848-932-0909

Academic Editors: Georgios Kambourakis and Constantinos Kolias
Received: 5 June 2017; Accepted: 25 June 2017; Published: 28 June 2017

Abstract: The Internet of Things (IoT) is a recent trend that extends the boundary of the Internet to include a wide variety of computing devices. Connecting many stand-alone IoT systems through the Internet introduces many challenges, with security being front-and-center since much of the collected information will be exposed to a wide and often unknown audience. Unfortunately, due to the intrinsic capability limits of low-end IoT devices, which account for a majority of the IoT end hosts, many traditional security methods cannot be applied to secure IoT systems, which open a door for attacks and exploits directed both against IoT services and the broader Internet. This paper addresses this issue by introducing a unified IoT framework based on the MobilityFirst future Internet architecture that explicitly focuses on supporting security for the IoT. Our design integrates local IoT systems into the global Internet without losing usability, interoperability and security protection. Specifically, we introduced an IoT middleware layer that connects heterogeneous hardware in local IoT systems to the global MobilityFirst network. We propose an IoT name resolution service (IoT-NRS) as a core component of the middleware layer, and develop a lightweight keying protocol that establishes trust between an IoT device and the IoT-NRS.

Keywords: Internet of Things; security; future Internet; name resolution; key management

1. Introduction

The initial concept and implementation of the Internet of Things (IoT) appeared as early as the 1980s and became popular in late 1990s [1]. Recent developments in many relevant areas, including automation, wireless sensor networks, embedded systems and micro-electromechanical systems (MEMS), has accelerated the evolution of the Internet of Things (IoT) [2,3]. Currently, IoT applications exist in nearly every field and are playing an increasingly important role in our daily life [4] (e.g., healthcare systems, building and home automation, environmental monitoring, infrastructure management, energy management and transportation systems), which has led to the recent proliferation of IoT systems. According to the Federal Trade Commission (FTC), the number of IoT devices has already outnumbered the number of people in the workplace [5], and the number of wireless devices connected to the Internet of Things will be about 26 billion by 2020 and will greatly outnumber hub devices (smartphones, tablets and PCs) [6].

As a result of the recent trend of extending the boundary of the Internet to include a wide variety of non-traditional computing devices, the Internet of Things makes the connection between the real world and the virtual world tighter than ever before. However, connecting various stand-alone IoT systems through the Internet brings many challenges, such as scalability, naming, resource constraints, mobility, inter-operability, security and privacy. To address these challenges, new IoT solutions have been proposed that utilize clean slate future Internet architectures, such as MobilityFirst [7], XIA [8], NDN [9], and Nebula [10]. These approaches rely on the new features provided by future Internet architectures and solve important problems such as mobility, robustness and evolvability. However, one major hurdle facing IoT is security and privacy. As defined in [5], “IoT" refers to the *connectivity between everyday objects and the Internet* and *the ability to exchange data between them*. As a result, potential security and privacy risks exist in a broad scope, ranging from the physical world to the Internet, and can be exploited to harm people. For example, a compromised IoT device may facilitate attacks on other systems. Unauthorized access may result in the leakage and misuse of personal information. A breach in the Internet may feed back to the physical world and create risks and threats to people’s physical safety.

Although conventional security will address many problems, there are unique aspects associated with IoT security that necessitate a broad and holistic approach to secure the IoT. IoT devices are typically deployed in an unattended manner, which facilitates the possibility of physical attacks. Low-end IoT devices are incapable of performing heavier, conventional cryptographic algorithms due to their constrained resources. A large quantity of IoT devices adopt wireless as a means to communicate, which is open to eavesdropping and jamming. To improve IoT security, this paper investigates how to securely integrate the IoT into the Internet, and provides several major contributions: (1) we present a unified IoT architecture based on a future Internet infrastructure with the focus on a newly introduced component, *IoT middleware*, which smoothly integrates local IoT systems to the global Internet; (2) we investigate the vulnerabilities of the existing IoT systems and provide security measures to protect against attacks and exploits through our unified IoT solution. Notably, we introduce a secure name resolution framework at the IoT middleware to enable seamless name translation and services as well as facilitate security; (3) lastly, we propose a delegation-based three-party key management protocol to establish symmetric keys within local IoT systems for secure operations, which completes the last piece of key management for the whole IoT architecture and hence enables security coverage from local IoT systems to the global future Internet.

The rest of the paper is organized as follows: we first investigate current IoT solutions in Section 2 and examine their security and privacy threats in Section 3. Then, in Section 4, we present a design for an IoT platform based on a representative future Internet architecture, MobilityFirst. Specifically, we introduce an IoT middleware to manage IoT devices and data in Section 5. Based upon our security and privacy analysis, security countermeasures are proposed to protect against those challenges. In particular, to cope with the capability limitations of low end IoT devices, we develop a key provisioning protocol to enable secure communication in local IoT systems in Section 6.

2. Survey of the Evolution of IoT Architectures

Many current IoT designs do not support applications seamlessly. Historically, stand-alone IoT systems were proprietary implementations, such as DF-1 [11], MelsecNet [12], Smart Distributed System (SDS) [13], and BACnet [14]. These fragmented solutions were typically integrated vertically and characterized as “silo” solutions, as shown in Figure 1a. A large number of independent legacy IoT systems co-exist across the Internet. However, their “silo” nature conflicts with the open spirit of the Internet and hence introduces problems of inter-operability and service-level interaction, which limits the benefits of IoT systems and could impede large-scale IoT deployment.

As the traditional approach has many fundamental issues, overlay-based IoT architectures were proposed in an attempt to unify IoT solutions. Figure 1b shows this approach, where an overlay IoT forms a network on top of the current Internet. Standard control and data APIs were adopted to connect IoT devices and the Internet. Data from various devices in different IoT deployments is pushed to central IoT servers through IoT gateways across the Internet. IoT applications then directly subscribe to the IoT server. Low Throughput Networks (LTN) [15], Global Sensor Networks (GSN) [16,17] and Constrained RESTful Environments (CORE) [18,19] are representatives of the standardization efforts involving the overlay-based IoT approach. This overlay approach, however, has several weaknesses that makes it not ideal for wide deployment. First, the merging of control and data forwarding at the central server is a bottleneck and causes scalability concerns. Second, the central IoT server is a single point failure and represents an ideal point of attack. Further, this solution does not seamlessly support mobility, which is a major requirement of the future Internet and a desirable characteristic for IoT applications.

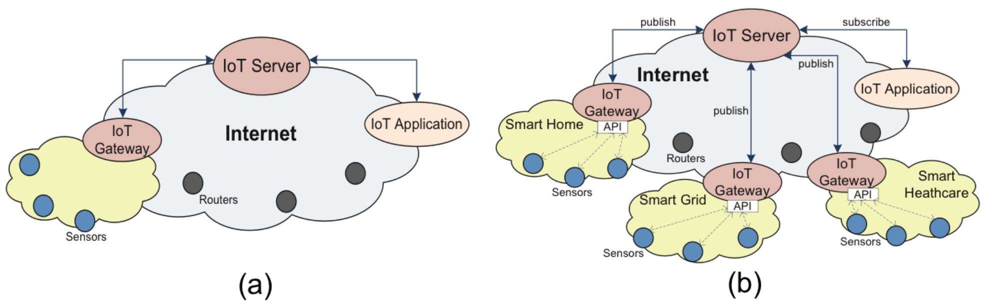


Figure 1. Legacy Internet of Things (IoT) architectures: (**a**) Silo architecture of standalone IoT systems. (**b**) The overlay-based IoT architecture connects standalone IoT systems with the IoT server through standardized APIs on the IoT gateways.

Recently, several works [20–25] focus on establishing an authorization framework for IoT, and enable access control in the IoT. However, these piece-wise solutions do not provide a complete framework to address many practical issues facing IoT systems, such as identification, addressing, mobility, interoperability and scalability. Another trend to enhance IoT systems is the introduction of an abstract middleware layer to enable object virtualization, such as FI-WARE [26] and ETSI oneM2M [27]. The ETSI oneM2M introduces a Service Capability Layer as the virtualization layer, through which heterogeneous IoT devices and applications may exchange information. FI-WARE is based on ETSI oneM2M and integrates the Open Mobile Alliance Next Generation Services Interface to manage and exchange context information.

Because of the IoT's increasing importance, future Internet designs need to design for the IoT. IoT systems can also benefit from many of the new features and protection mechanisms being proposed for the future Internet infrastructure. In particular, *NDN-based IoT* and *MobilityFirst-based IoT* are two representatives of future Internet based IoT systems as they support the key features of IoT as well as follow the current trends of evolving the Internet. As *data* plays a pivotal role in the IoT, the NDN-based IoT was proposed to support content retrieval. On the other side, MobilityFirst-based IoT builds upon the MobilityFirst network [28,29], whose major goal is to support *large scale mobility*, and therefore is strategically designed to handle scenarios where IoT devices are moving. One typical mobile IoT application scenario is vehicular networks, where sensors installed in the moving vehicles collect data and provide the data to the relevant applications through the underlying IoT infrastructure. In this paper, we use the MobilityFirst future Internet architecture as the basis for our IoT design in Section 4. We note, though, that some of our security mechanisms are general and can be applied to other future Internet based IoT systems.

3. General Security Analysis of IoT Systems

The IoT extends the Internet to the physical world and thus poses many new security and privacy challenges. Some of the problems are due to the intrinsic characteristics of the IoT and its differences compared to traditional networks, while others arise as a result of the integration of the IoT and the Internet. As shown in Figure 2, various adversaries may come in at different points to attack IoT systems. To protect against those attacks, it is important to examine the security problems according to the information flows and potential adversarial points of control. Below, we outline four security and privacy problems:

- **Authentication and physical threats**: highly distributed deployments of a large number of IoT devices, such as RFID tags and wireless sensors, will generally be deployed in public areas without any protection, which makes the devices difficult to manage and vulnerable to physical attacks. For example, an illegitimate sensor may register itself claiming that it is at one location while it is actually at a different location. Or a sensor installed in a room monitoring the room temperature is moved to another room by a malicious person. This introduces the challenge of authenticating IoT devices, which involves recognizing the device and verifying its association with a correct topological address.
- **Integrity**: the unattended environment for IoT devices also makes data integrity a concern. Once deployed, most of these devices will operate in a self-supported manner. As with very limited maintenance or even no maintenance, tampering data is a much easier task than in a supervised wired network. Further, as a result of a natural loss of calibration or a deliberate perturbation of the measurement environment by an attacker, the data collected by IoT devices is quite likely to have low quality and might be corrupted at the environmental level. In short, IoT data may be noisy and easy to spoof and forge.
- **Confidentiality**: the communication method between devices and the gateway is primarily wireless, which results in confidentiality risks. For example, eavesdropping is a major concern in wireless networks. Unfortunately, unlike many other wireless environments, such as cellular and Wi-Fi networks, it is difficult for IoT networks to provide confidentiality for data transmission due to the resource-constrained nature of low-end devices, which are a large fraction of IoT devices [30]. Different from typical devices in traditional wired and wireless networks, such as smartphones, tablets, PCs and routers, most of the devices in future IoT networks are active sensors or passive RFID tags, which have very limited resources and capabilities. Constraints on power, computational capability, storage and other aspects of an IoT device introduce a high barrier for it to perform the necessary operations to achieve data confidentiality, such as through encryption and key management.
- **Privacy**: as an existing public concern for monitoring and interacting with the real world, the consequence of information leakage in local IoT networks becomes exacerbated when integrated into the global Internet. By connecting real world objects and information through the Internet, data may become accessible to various organizations and domains across the Internet, instead of only being revealed to a small group, which makes it more likely to be exposed to sophisticated malicious parties and therefore increases the probability of being exploited and attacked.

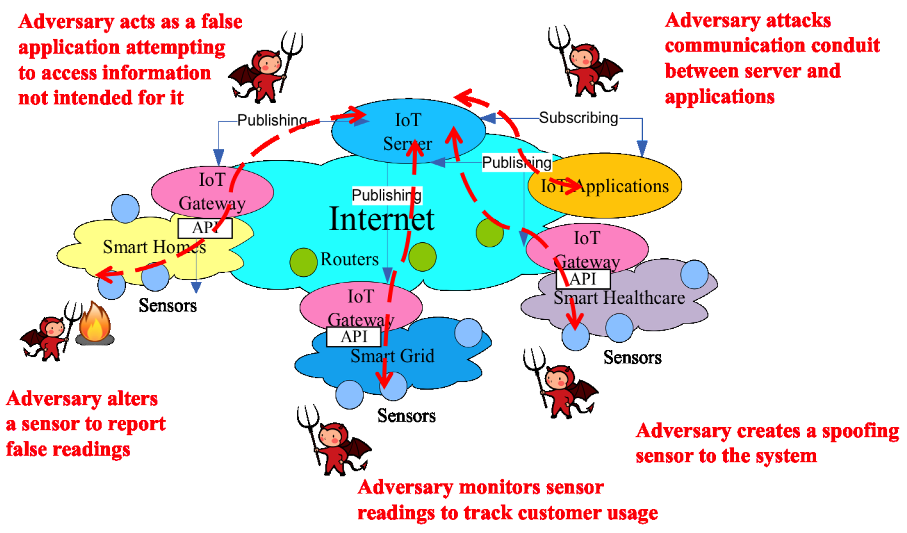

Adversary acts as a false application attempting to access information not intended for it
Adversary attacks communication conduit between server and applications
IoT Server
Publishing
Subscribing
Publishing
IoT Applications
IoT Gateway
API
Smart Homes
Publishing
Internet
Routers
IoT Gateway
API
Smart Healthcare
Sensors
IoT Gateway
API
Smart Grid
Sensors
Sensors
Adversary alters a sensor to report false readings
Adversary creates a spoofing sensor to the system
Adversary monitors sensor readings to track customer usage

Figure 2. Potential threats for the IoT systems.

Conventional security and privacy techniques are not necessarily appropriate for the IoT due to the special characteristics of the IoT. The attractive prospect of IoT applications, as well as the strong needs of increasing public confidence about security and privacy issues, requires new and comprehensive solutions to not only protect local IoT devices but also the broader Internet aspect of the IoT. In the following sections, we will examine the problems aforementioned and explore security and privacy techniques to support the IoT infrastructure based on the MobilityFirst network, one of the representatives of future Internet architectures.

4. MobilityFirst-Based IoT Architecture

4.1. Overview of the MobilityFirst Infrastructure

MobilityFirst is a future Internet architecture that aims at providing fundamental solutions to the challenges facing the current Internet. The primary design goals that drive MobilityFirst are *mobility* and *trustworthiness*, two major trends of the Internet. In order to achieve these two design goals, MobilityFirst breaks targeted specific design goals, including seamless host and mobility, no single root of trust, intentional data receipt, proportional robustness, content addressability and evolvability. These design goals come together to solve the challenges faced by the current Internet.

Figure 3 shows that MobilityFirst introduces a new protocol stack and replaces the original "narrow waist" of the current Internet (i.e., TCP/IP) with a new name-based service layer, which consists of the *Name Certificate & Resolution Service (NCRS)* and the *Global Name Resolution Service (GNRS)*. This new service layer is centered on the concept of "flat" security-based GUIDs for network objects, a single abstraction that serves as both the network object's identity and the public key. Various types of network entities, e.g., a smartphone, a person, a group of IoT devices, a piece of content or context, can obtain their globally unique identifiers from the NCRS, which provides translation services between human readable names and public-key based globally unique identifiers. The GNRS provides a clean separation of the identifiers and the dynamic network address locators and supports on-the-fly binding of names to network addresses as needed for dynamic mobility. Therefore, the use of the GUID at the name-based service layer enables mobility-centric services at scale and provides the foundation of a trustworthy network.

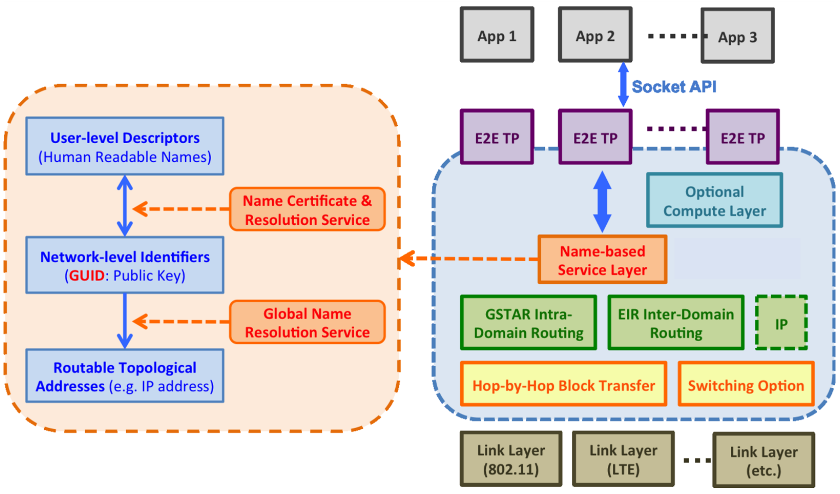

App 1
App 2
App 3
Socket API
E2E TP
E2E TP
E2E TP
Optional Compute Layer
Name-based Service Layer
GSTAR Intra-Domain Routing
EIR Inter-Domain Routing
IP
Hop-by-Hop Block Transfer
Switching Option
Link Layer (802.11)
Link Layer (LTE)
Link Layer (etc.)
User-level Descriptors (Human Readable Names)
Name Certificate & Resolution Service
Network-level Identifiers (GUID: Public Key)
Global Name Resolution Service
Routable Topological Addresses (e.g. IP address)

Figure 3. The protocol stack of the MobilityFirst architecture.

With the name-based service layer, the MobilityFirst architecture is able to relieve or at least mitigate many pains in today's Internet. In the MobilityFirst network, due to the separation of the name and the network locator, users may request content by the content name directly, without bothering with the current network address. The routers handle the content request by querying the GNRS for a list of the up-to-date content storage locations in a timely fashion and then fetching the content from the nearest storage location [31]. This approach also handles content mobility well with support from the on-the-fly/late binding of names to addresses and the hop-by-hop storage-aware transport protocol MFTP [32]. The GUID, which is a public key, allows MobilityFirst to integrate trustworthiness as one of the basic properties of the architecture. Deriving the identifiers in a cryptographic manner makes many security measures, e.g., authentication, accountability and encryption, intrinsic to the MobilityFirst network thus facilitating further protection against various attacks.

4.2. MobilityFirst-Based IoT Architecture Design

As a robust and trustworthy mobility-centric architecture with abundant in-network services, MobilityFirst can address many challenges that today's IoT is facing, such as scalability, mobility, content retrieval, inter-operability, and security/privacy. Therefore, we propose a unified solution to support IoT based on the MobilityFirst network architecture [33]. Our IoT platform consists of four basic components as shown in Figure 4. Though this IoT architecture is designed in the context of the MobilityFirst network, it is general and can be applied to other similar network infrastructures, which adopts the separation of identifiers from network locators, such as [8,34–36].

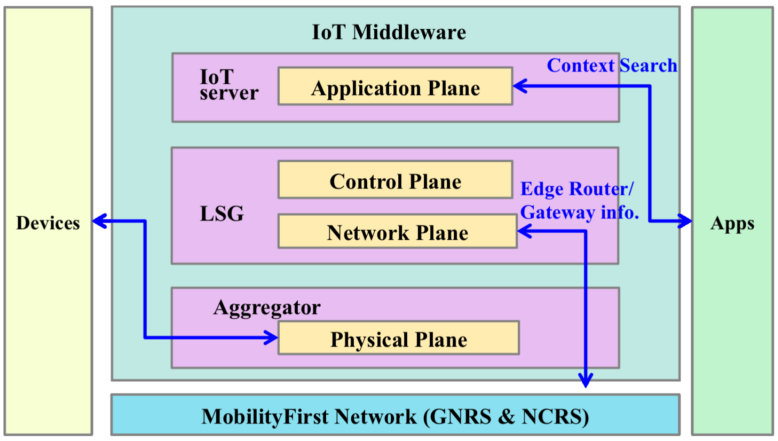


Figure 4. The four basic components of the IoT MobilityFirst architecture: devices, IoT middleware, MobilityFirst network and applications.

- **Devices**: a wide variety of devices, e.g., sensors, actuators and tags, that use embedded techniques to sense, communicate and/or interact with the external environments.
- **MobilityFirst network**: MobilityFirst provides connectivity for different distributed IoT devices and applications. Due to the seamless mobility support of MobilityFirst, besides well-established IoT system, more dynamic IoT applications/systems can be developed on top of it.
- **IoT middleware**: MobilityFirst-based IoT middleware integrates three functional layers—*Aggregator*, *Local Service Gateway (LSG)*, and the *IoT server*. The aggregator supports sensor abstraction to hide the hardware specifics for sensors and presents a single interface to query and subscribe to the sensor data. LSG connects the local IoT system to the global Internet and provides necessary management, including naming resolution, key management and context processing services. The IoT server is a logically centralized server in the control plane that manages subscription memberships as well as provides a lookup service so that subscribers may query for the IoT data/services.
- **Applications**: end users who consume the IoT data and may feed back to the external environment through actuators.

Figure 5 explains how the IoT architecture works with the support of the MobilityFirst infrastructure. The aggregator collects data and then the LSG sends the aggregated data to the storage location. The raw data might be processed by the LSG for context refining/aggregating purposes. Meanwhile, the LSG publishes the data information, which contains a data GUID, the access control policy and the storage location information (e.g., human readable names or network address), to the IoT server so that end users may query the IoT server about where to fetch the data. The IoT server may decide how to enforce the access control policy: either at itself or push it to the NCRS/GNRS. The data consumer, typically an application, first makes a query at the IoT server for data information through its edge router and then fetches the data from the storage location or the aggregator directly.

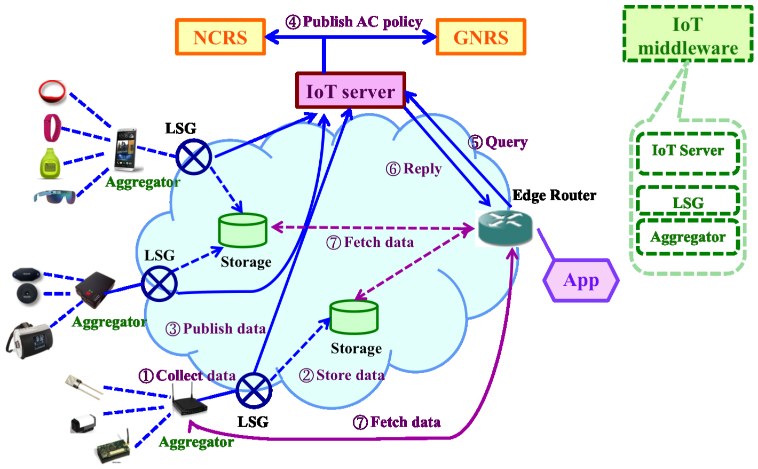


Figure 5. MobilityFirst-based unified IoT architecture.

4.3. Protect IoT through Existing MobilityFirst Network Services

As shown in Figure 4, the MobilityFirst-based IoT architecture consists of four basic building blocks: *Devices*, *IoT middleware*, *MobilityFirst network* and *Applications*. Our goal is to provide clean and secure data to various applications/services in the upper layer and make their development/management easy. Thus, as illustrated in Figure 6, we integrate the security/privacy mechanisms into the network, the IoT middleware and the lower layer "devices", but not the application layer.

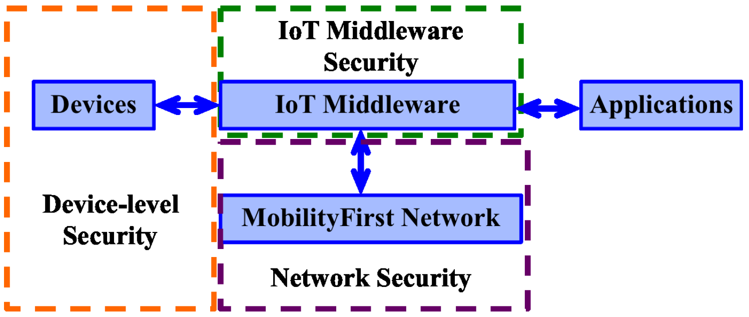


Figure 6. Security/privacy mechanisms are introduced into devices, IoT middleware and MobilityFirst network.

The rich in-network services of MobilityFirst enable many powerful functionalities to protect against a wide variety of attacks. For example, the NCRS serves the role of a certificate registration center that associates human-readable names to certificates. This enables various security methods, such as encryption and authentication, to secure the data transmission above the IoT device layer. Together with the security schemes at the device level, which provide secure communication channels between IoT devices and the Internet, the confidentiality of the data flow from the bottom (IoT devices) to the top (applications) can be achieved.

The security mechanisms enabled by network services, i.e., the GNRS and the NCRS, build a shield against many attacks, such as false registration attacks and device misplacement attacks. *False registration attack* refers to an adversary registering a device with a fake location. *Misplacement attack* is another form of attack involving manipulating device location by moving a device to another location without updating the new location. The common characteristic of these two attacks is *location forgery* and is similar to *GNRS false announcement attack*, which is defined as a network object with identifier $GUID_1$ in network $NA1$ inserting or updating a GNRS mapping $< GUID_1, NA2 >$ with wrong network address to the GNRS. Therefore, we can apply the same philosophy of preventing GNRS false announcement attacks to location forgery attacks in the context of the IoT. These protection mechanisms are presented in [37] and hence we will not discuss them in this paper due to the space limitation.

Another significant contribution provided by network services is privacy protection, such as GNRS access control discussed in [38]. The access control in the GNRS protects the data privacy, and also increases the difficulty of launching attacks by restricting adversarial access to information that is essential for launching an attack, whatever that attack might be. Access control enforced at the GNRS query is a powerful tool as it can provide the GNRS mapping owner, who is typically the data owner or a surrogate in the context of IoT, the ability of choosing who it is willing to communicate with. With the support of access control in the GNRS, IoT devices or data owners can protect the IoT data's location information contained in the GNRS mappings against unauthorized disclosure, while at the same time ensure the mapping's accessibility to legitimate subscribers or applications. In addition, GNRS access control can support advanced services, such as allowing the mapping owner to decide when and where it is reachable. These fine-grained functionalities provided by GNRS access control make it possible to specify detailed policies/regulations while distributing the data collected by the IoT devices.

Our MobilityFirst-based IoT architecture provides a unified solution to solve issues of inter-operability as well as support mobility. Our design also can benefit from many built-in security features of the MobilityFirst network infrastructure. However, this architecture still has some unique security concerns that arise and are related to general security concerns mentioned in Section 3. Specifically, most of the security properties and advanced services are based on the concept of a GUID, which is essentially a public key. Unfortunately, low-end IoT devices, which account for a majority of IoT devices, cannot afford to perform expensive asymmetric crypto operations. This leads to key management challenges for our IoT solution. Therefore, the following sections will address these unique security concerns with the focus on key management.

5. IoT Middleware Security

The IoT platform in MobilityFirst introduces a new essential component, *IoT middleware*, to handle data processing and distribution in the data plane as well as system management in the control plane. The IoT middleware consists of three layers, *aggregator*, *local service gateway (LSG)* and *IoT server*, so that it can support a variety of functionalities, including managing IoT devices, processing the raw data collected by the devices and then distributing the processed data through the MobilityFirst network. This approach allows a clean separation of IoT systems from the underlying network so that the network is only responsible for transmitting the IoT data, just as it handles other network traffic. Therefore, integrating our IoT solution into MobilityFirst architecture has a lightweight touch that is easy to deploy. Further, decoupling the IoT middleware from the underlying network infrastructure makes our IoT solution more general and is easily ported to other network designs that share similar features with MobilityFirst. For example, the middleware layer is independent of the network infrastructure and supports object virtualization that enables interoperability between heterogeneous devices regardless of the underlying networking medium. In particular, we note that the separation of our middleware from the network allows for our IoT security solutions to be easily deployed on the existing Internet infrastructure, as well as other clean slate architectures, such as NDN [9] and XIA [8]. Further, the security mechanisms adopted in our middleware provide secure communication and key management functions that are generally both necessary and applicable to other styles of IoT deployments.

In the IoT middleware architecture described in Figures 4 and 5, the *aggregator* sitting at the bottom level is usually located near or at the gateway of the local network. This is the first stage of data processing as the aggregator collects raw data directly from a variety of heterogeneous IoT devices and filters out the redundant data. The introduction of the aggregator hides the complex heterogeneous hardware from upper layer applications and facilitates application development. The *LSG* level contains one *IoT Name Resolution Service (IoT-NRS)* and several other functional components depending on the system and application requirements. The top level is the *IoT server*, which is essentially a logically centralized but physical distributed database that provides lookup service for data consumers. The aggregator, the LSG and the IoT server form the middleware of MobilityFirst-based IoT and connect the various IoT devices with the upper layer applications.

Our security measures are primarily integrated in the LSG layer, while the access control policy is enforced at the IoT server, who might push it further to the GNRS. From the perspective of the data consumer, which is typically an application, to either fetch the processed data from the data storage location or get the raw data from the aggregator directly, it needs to know where to fetch the data. Therefore, the data consumer looks up the data storage location at the IoT server with its identity. On the other side, the IoT server evaluates the request with the access control policy defined by the IoT data owner. If the decision is "approve", it returns the data location to the data consumer (possibly with a token depending on the specific access control policy). Then the consumer may be able to fetch the data or setup a subscription. In the case where the IoT server pushes the enforcement of the access control policy to the GNRS, the GNRS will grant a token to the data consumer if the access decision is "approve". Then the data consumer can present the token to the IoT server to request data information.

At the data producer side, efficient and secure IoT device and data management is not only necessary but crucial for providing accurate and fresh IoT data. Particularly, key and identifier management plays an essential role in identifying/verifying IoT devices as well as establishing/maintaining confidential communication channels in the local IoT systems. However, due to the intrinsic characteristics of IoT devices, especially low-end devices, there are key management challenges unique to IoT systems. To cope with those challenges, we developed an IoT Name Resolution Service at the LSG to enhance the security, efficiency and inter-operability.

5.1. Overview of the Name Resolution Framework in MobilityFirst-Based IoT

As discussed in Section 4.1, the GNRS and the NCRS center on the concept of the GUID, which serves as the identifier and the public key. Within the scope of the IoT, however, the identifier application/assignment operation and relevant cryptographic computations may be beyond the capability of most IoT devices. For the remainder of the paper, we are not worried about devices with strong computational capability and sufficient resources, such as smartphones, laptops and tablets. Instead, we focus on low-end IoT devices, such as small sensors and actuators, which will account for the majority of IoT devices. These low-end devices, unfortunately, have very limited computational capabilities and strict resource constraints in terms of storage, power, and bandwidth. Hence, they are unable to perform many heavy duty tasks, including the computation operations of public/private cryptography. To enable secure communication, we propose a lightweight approach by using symmetric keys to replace the GUID in the scope of local IoT systems. In other words, an IoT device's identification associated with membership within a local IoT system is associated with a symmetric membership key shared between the device and the local group authority, i.e., the LSG. Symmetric cryptography is chosen to make the membership key because it is lightweight and has much less computational requirements.

On the other side, in order to smoothly integrate the local IoT system into the global MobilityFirst network and take advantage of the rich network services provided by MobilityFirst, it is necessary to maintain use of the GUID in the scope of the broader global network so that data consumers can contact IoT devices or retrieve/subscribe IoT data with the corresponding GUIDs. In order to reconcile the conflicts of the two different cryptographic schemes (used in two different scopes), integrating an IoT name resolution service (shown in Figure 7) in the middleware to handle naming-related issues is the best solution as it keeps a clean separation between the underlying network infrastructure and the IoT system. The IoT name resolution service (IoT-NRS) bridges the gap between different cryptographic schemes used in the local IoT network and the global Internet. This design makes IoT systems flexible, extensible and easy to manage.

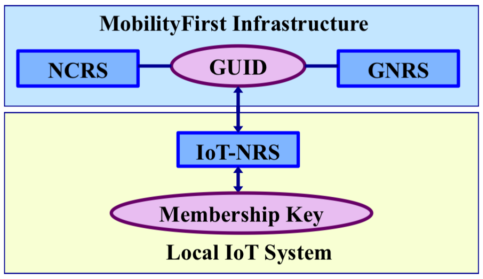


Figure 7. Three-tier name resolution framework of the MobilityFirst-based IoT architecture: Name Certificate & Resolution Service (NCRS), Global Name Resolution Service (GNRS) and IoT Name Resolution Service (IoT-NRS). The IoT-NRS connects the GUID used in global MobilityFirst network and the symmetric membership key used in the local IoT system.

5.2. Major Functionalities of IoT Name Resolution Service

Our IoT Name Resolution Service (IoT-NRS) can be considered as a simplified embodiment of the MobilityFirst name-based service layer in the context of the local IoT systems. It assigns identifiers, establishes long-term keys (i.e., membership keys) and provides key/identifier related management. In the following sections, we envision the major functionalities the IoT-NRS should have.

- **GUID Assignment**: IoT-NRS serves as a surrogate and works together with the NCRS to assign GUIDs to network objects, such as IoT data, certain devices or a group of IoT devices, to assist advanced network services in the global Internet. The relation between the GUID and the device/data might be *one-to-one*, *one-to-many* and *many-to-one*. From the application point of view, generally the data is of interest and one piece of data has one GUID, which is associated with a list of attributes describing the data object. With respect to devices, it is more often than not that a group of devices map to a single GUID. As an example, it is reasonable to assign one GUID to all of the temperature sensors in a particular room. However, sometimes one device may obtain a GUID if necessary. For example, if the device is the only device in its category or it is important for the application. Also, sometimes one sensor may associate with two or more GUIDs. For example, a multi-function sensor attached to Tom's mug may have $GUID1$ with attributes of location information and $GUID2$ associated with the mug's temperature.
- **Membership Credential Establishment**: IoT-NRS at the LSG may act as a group authority to establish membership credentials associated with the IoT device in the scope of the local IoT system. This long-term membership credential identifies the device in the local group. A proper symmetric cryptographic algorithm is chosen to generate such a credential (i.e., a symmetric key) so that low-end devices can afford to perform security operations. After establishing the membership credential, short-term keys (for example, session keys) and functional keys (for example, attestation key) may be derived from the membership key.
- **Name Translation between GUID and Membership Key**: Two directional name mapping between the GUID and the membership key connects the local IoT system to the global network so that IoT-related operations can be performed seamlessly.
- **Membership Management**: IoT-NRS should also provide membership management, where the membership is represented by the GUID and/or the membership key. Such management includes

renewal and revocation operations. Both the GUID and the membership key have expiration times to guarantee key freshness and security strength. Therefore, renewing in a timely and efficient fashion is essential to keep the device/data “alive" in the system. On the other hand, revocation is needed to remove compromised/dysfunctional devices so that potential security/privacy threats can be reduced.

Figure 7 illustrates the flow of IoT-NRS's major functions. The most important function of the IoT-NRS is to serve as a registrar. During device discovery, when a new IoT device joins a local group, the IoT-NRS acts as the group authority and registers this new device. The registration establishes a long-term membership key with the new device using a proper key provisioning protocol, which depends on the device's capability, as well as associates the membership key with the device's attributes/identity, such as manufacturer, model, serial number, function, etc. In the global context, the IoT-NRS serves as a surrogate to apply and manage GUIDs for the low end IoT devices within the local group and the data they generated because these devices and data do not have the ability to run asymmetric cryptographic algorithms. After initial device discovery and registration, the IoT-NRS manages the identifiers and keys, and performs tasks that include renewal, verification and revocation.

6. Delegation-Based Key Provisioning Protocol

6.1. Overview

In the scope of local IoT systems, it is critical to discover edge devices and establish long-term membership credentials between those devices and a group management authority (i.e., the IoT-NRS in our IoT architecture) in a lightweight fashion. In the device discovery process, enrolling a new device and establishing a secure membership credential between edge devices such as sensors and actuators can be difficult, as such devices can have heterogeneous and potentially limited computational capabilities. This property can limit interactions for certain protocols and further limit management of membership keys/credentials.

Therefore, we propose a delegation-based key provisioning protocol in this section to facilitate the device discovery process and establish the membership key for the newly added edge device. Specifically, in the context of our MobilityFirst-based IoT architecture, this protocol can serve as: (1) a grouping tool in the initial stage of setting up a local IoT network; (2) registration tool in the device discovery process when a new IoT device wants to join an existing IoT system. Our key provisioning protocol, though, is general and can have broad application. Other use cases will be explored in Section 6.9.

Problem Description

To generalize our goal, we would like to establish *lightweight* symmetric key(s) between one or more edge IoT devices (referred to as a *Child* device), such as sensors, RFID tags and actuators, and a computing device (referred to as a *Guardian* device) within an IoT network. Here, we emphasize the lightweight nature of the output key and involved cryptographic operations associated with the Child device because of the intrinsic characteristics of the IoT systems as discussed in previous sections.

Trust does not come from nowhere. To build a mutual trust between the Child and the Guardian, we need to extract the new trust relationship from an existing trust relationship. With that being said, we start from a pre-established trust relationship, with which we can attest the identity and integrity of the target Child. As a result of this reasoning, a third party is introduced into our protocol, known as the *Parent* device. The Parent, for example, can be the owner of the Child device or the manufacturer of the Child. Essentially, the Child and the Parent may have a pre-established trust relationship, in which the Child device stores therein a *parent key* that indicates the Parent's privilege on the Child.

Now the challenge rests with how to leverage a device's Parent key and Parent trust establishment to delegate certain privileges to a new party, referred to herein as a Guardian, to enable the IoT edge device (Child) to establish a shared key securely with the new party, which thus acts as the edge device's Guardian. In other words, taking advantage of a pre-existing trust relationship between a Parent device and a Child device, we would like to yield a lightweight symmetric key shared between the Child device and the Guardian device.

6.2. System Model

Figure 8 describes the system model for the delegation-based key provisioning protocol, which consists of three participants: *Child*, *Guardian* and *Parent*. A local IoT network may include a Child device and a Guardian device. A Parent device, can be either within or outside this IoT network. This Parent may be, but not necessarily, a (relatively) more capable computing device, such as a server computer, gateway device, personal computer, smartphone, or any other type of computing device. For example, the Parent may be a computing system of a manufacturer, such as the manufacturer of the Child device. The importance of the Parent arises from its existing trust relationship with the Child, from where we can extract and establish a new trust relationship.

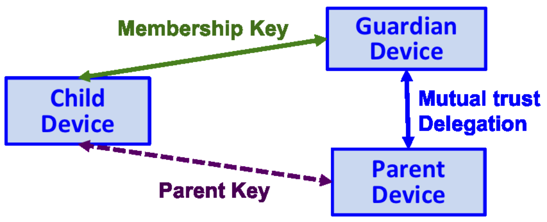


Figure 8. The block diagram illustrates the framework of the delegation-based key provisioning protocol: *Child* device is an edge IoT device who wants to join the IoT system and establish a *membership key* with the Guardian device; *Guardian* device is the group authority of the local IoT network, who contacts Parent device to verify Child device and obtains authorization; *Parent* device has a pre-existing trust relationship with Child device in terms of the *Parent key*, and provides verification/delegation to the Guardian device so that Guardian device and Child device can reach a resultant key agreement: membership key.

The Child is an edge IoT device, e.g., sensor, actuator, etc. To indicate its trust relationship and parenthood status with regard to the Child, the Parent may provision a *parent key* into a non-volatile storage of the Child device. This non-volatile storage may be a secure storage, accessible only in a trusted execution environment (TEE) [39,40] of the Child device. In some cases, this TEE may be implemented using separate hardware circuitry, such as a separate micro-controller from a main processor of the child device, e.g., a central processing unit (CPU).

A Guardian device is also present in the network. In our MobilityFirst-based IoT architecture, the Guardian is the IoT-NRS of the local IoT network. In other embodiments, the Guardian may take the form of a gateway, smartphone, or consumer computer with an installed delegation protocol package. As an example, the Guardian may be a central authority for a domain (such as an entire IoT network or a portion thereof). The symmetric *membership key* shared only between the Child and the Guardian is the local long term membership credential established by our key provisioning protocol.

Thus, in the context of Figure 8, the two parties to establish a trust relationship represented by a key are the Child and the Guardian. A third party Parent has an existing parent key provisioned to the Child, though the Child and the Parent may or may not have a direct communication channel at this point. The only available communication channels are the one between the Child and the Guardian and the one between the Guardian and the Parent through out a protocol session. With the delegation provided herein, key provisioning between the Child and the Guardian can occur, given that the Parent is a trusted third party to provide authentication and authorization of the Guardian to the Child.

6.3. Security Requirements

The delegation from the Parent to the Guardian with respect to the Child can occur to address a variety of concerns. First, IoT edge devices have highly heterogeneous capabilities. The protocol for establishing mutual trust and keys described in Section 6.6 can support capable devices as well as highly resource-constrained devices. Second, establishment of the IoT device keys may be user-friendly, which conventionally requires the protocol to either leverage a trusted public key infrastructure (PKI) that is transparent to the user, or to remove dependency on trusted servers but minimize dependency on manual interaction by users. Traditional PKI or Kerberos services [41] can be challenging to establish on a global scale, and may require substantial IT professional expertise as well as computational resources for cryptographic processing. On the other hand, solutions for key provisioning without a server are often insecure, such as pre-shared key approaches, lacking resilience by working directly with fixed manufacture/device keys, or involving cumbersome and critical manual interaction such as pairing. We try to address these issues in our delegation-based key provisioning protocol.

Particularly, when developing the key provisioning protocol, the following properties and requirements should be taken into consideration:

- **Scalability**: As many IoT systems are large scale deployments, it would be extremely difficult to establish a secure credential for each device if human intervention is required. To enhance usability and chance of practical deployment, it is important and necessary to remove manual operations by the user on an out-of-band (OOB) channel, such as secure pairing. Instead, a trusted third party is introduced to facilitate automatic trust establishment as well as reduce human error.
- **Consistency**: All the three protocol participants should have a consistent view of the protocol session. Namely, the three participants involved know who the peers are with respect to the specific session.
- **Dynamic mutual trust and delegation**: With the success of the protocol, each pair of the involved participants can reach a mutual trust at the end of the procedure. Further, the Guardian establishes the mutual trust dynamically with the Parent and obtains delegation before executing the privilege with respect to the Child.
- **Secrecy and forward secrecy**: The principle of least privilege should be followed, implying that any resultant secret key should only be shared between the involved two parties. No third party should be able to distinguish the resultant key from a random key. Also, the *forward secrecy* should be conserved so that the compromise of a long-term secret key will not affect the confidentiality of past communication sessions [42].
- **Lightweight**: As we focus on those resource limited devices, efficiency is a priority. When designing the protocol, the Child, which is typically, though not necessarily, a resource-constrained device, should be protected by shifting work load to the Guardian and the Parent as much as possible.
- **Average administration requirements**: The protocol should require as little professional knowledge as possible. The configurations are automatic and user-friendly so that people do not need to be IT experts to operate the protocol.

6.4. Adversarial Model

A key exchange protocol without a well-defined adversarial model cannot be considered secure. Hence, here we summarize the assumptions regarding potential adversaries. We assume an active network adversary, who may take full control of all the communication channels (namely, the communication channel between the Guardian and the Child as well as the one between the Guardian and the Parent). This attacker can eavesdrop on messages, modify them at will, inject its own message, delete messages, delay message delivery, duplicate any message and/or replay it later. It may even be able to initiate new protocol sessions and interleave messages from different sessions [43,44].

To facilitate the dynamic establishment of the mutual trust between the Guardian and the Parent, we assume a trusted certification authority, e.g., a Public Key Infrastructure (PKI), is available so that the more capable parties (i.e., the Guardian and the Parent in our context) can use their long-term private information, such as private keys and certificates, to authenticate their identities with the assistance of the trusted certification authority. With respect to the Guardian and the Parent, if its long-term private information that is used to authenticate itself is leaked and is accessible to attackers, then we consider it to be corrupted.

As for the Child, we assume it is a resource-constrained device with necessary hardware protections. That means the adversary is at most a weak software adversary but it is not a hardware adversary. The attacker can only access and/or overwrite regular applications or the operating system, but not non-volatile secure storage, such as a Trusted Execution Environment (TEE) [39,40]. Also, we assume the adversary cannot modify the hardware circuitry. With that being said, the "Parent" key (shown in Figure 8) representing the pre-existing trust relationship between the Parent and the Child should be protected properly, e.g., in a TEE or implemented using separate hardware circuitry. If the "Parent" key on the Child is leaked or modified, we consider the Child to be compromised.

6.5. Protocol Design Choices and Considerations

In order to develop a delegation-based key provisioning protocol that satisfies the requirements mentioned in Section 6.3 in the presence of attackers that were described in Section 6.4, we first investigate several existing key exchange protocols and authentication protocols. Based upon the protocol analysis, we incorporate and customize Choo's three-party key distribution protocol (3PKD) and the SIGMA protocol to accommodate our needs. In the rest of this section, we will discuss the design choices and considerations.

6.5.1. Investigation of Three Party Key Exchange Protocols

There are several existing three party key exchange protocols. Particularly, we examine two representatives: Choo's 3PKD [45] protocol and Needham-Schroeder protocol [46]. One of the major differences between these two protocol is that Needham-Schroeder protocol is good for establishing a one-time use shared key (e.g., session key), while Choo's 3PKD protocol can generate a long-lived shared key. In the Needham-Schroeder protocol, the Parent, which serves as the key distribution center (KDC), generates the secret key and sends it to the one of the two peer parties (the Child and the Guardian) directly. Then the party, who receives the key from the Parent, shares the key with the other peer party with proper challenge-response verification. With the Needham-Schroeder protocol, all three protocol participants know the shared key that will be used between the two peer parties. However, the Parent does not necessarily need to know this shared key. This fact increases the risk of secret leakage and requires the generated key to be used and discarded as soon as possible because once the Parent is comprised, the shared key generated with the assistance of the Parent will be subject to leakage.

In Choo's 3PKD, the Parent provides a "session key" to the Child and the Guardian directly at the same time, and thus this "session key" may be used as a seed, with which the Child and the Guardian can compute a shared key by themselves without being known by the Parent. This strategy follows the least privilege principle and is preferred for generating a long-term key. Even if the server is comprised later, the shared key derived from the seed will not be comprised.

Another problem with Needham-Schroeder protocol is its vulnerability regarding a replay attack [47], which is fixed in the Kerberos protocol [41] by the adding a timestamp. Kerberos is another widely used key distribution protocol based on the Needham-Schroeder protocol. However, Kerberos is not suitable for our purposes for the following reasons: (1) it suffers from the single point of failure; (2) the centralized KDC model is not flexible and scalable enough to support heterogeneous Parent devices on a global scale. Different edge IoT devices (Child devices) may have different Parent devices, which have pre-existing trust relationships and can provide verification and/or delegation regarding to the corresponding Child devices. Due to the vast type and quantity of the Child devices, the Parent devices are also heterogeneous and numerous. As a result, extending Kerberos to our context (e.g., a global-wised MobilityFirst-based IoT architecture) is not feasible. (3) Kerberos has a strict time requirement, which means the clocks of the involved devices must be synchronized within configured limits. However, accurate clock synchronization might be too expensive for many low end IoT devices. Hence, in this protocol, we prefer to use nonces instead of timestamps.

On the other hand, Choo's 3PKD protocol has a solid security proof and no attack against it has been identified yet. Therefore, based upon the analysis above, we chose Choo's 3PKD protocol as the blueprint for the delegation-based key provisioning protocol. However, Choo's 3PKD protocol alone cannot satisfy our requirements and achieve our goal. There is an assumption in Choo's 3PKD protocol that the Parent is trusted by both the Child and the Guardian before the protocol execution. Unfortunately, this is not necessarily always true in our setting. If the Guardian and the Parent have already established a secure communication channel, then they may leverage such an advantage to facilitate the Parent to delegate the Guardian the privilege to establish a key with the Child. Alternatively, if the Guardian does not know the Parent previously and/or there is no secure communication channel between the Guardian and the Parent, these two parties need to build mutual trust on the fly during the on-going protocol session. This requires incorporating an authentication and delegation process between the Guardian and the Parent, which will be discussed in the following section.

6.5.2. SIGMA Protocol

The Diffie-Hellman (DH) key exchange protocol [48,49] is one of the most fundamental two-party key exchange protocols. Though itself is not an authenticated key agreement protocol, it is the basis for many popular authenticated protocols. One serious vulnerability of the Diffie-Hellman key exchange is the man-in-the-middle-attack [50]. To address the weaknesses of the Diffie-Hellman protocol, a variety of variant two-party key exchange protocols have been proposed, including the STS protocol [51], Photuris protocol [52] and SIGMA [44] protocol.

Unfortunately, many of those variants have flaws that were identified later on. To the best of our knowledge, SIGMA is still a secure two-party authenticated key exchange protocol that has no effective attack against it. SIGMA takes advantage of the public key infrastructure and adopts *"SIGn-and-MAc"* approach to authenticated Diffie-Hellman with digital signatures. It serves as the cryptographic basis for the Internet Key Exchange (IKE) standard (version 1 and version 2) [44,53,54]. In our key provisioning protocol, SIGMA is chosen as a building block and is overlaid with the 3PKD protocol to enable the Guardian and the Parent to authenticate each other on demand. This choice also brings other benefits, such as forward secrecy.

6.6. Protocol Specifications

As illustrated in Figure 8, there are three participants in the protocol: the Child, Guardian and Parent. The Child and the Parent share a symmetric key kek representing their pre-established trust relationship, but they can not communicate directly. The Guardian has direct communication channels with both the Child and the Parent. The output of the delegation-based key provisioning protocol is a symmetric key sk shared only between Child and Guardian.

Figure 9 presents the format and the flow of nine messages in the "full fledge" delegation-based key provisioning protocol. Table 1 lists the notations used in Figure 9. For simplicity, the Child, the Guardian and the Parent are denoted by C, G and P respectively in the following discussion. Below we present the delegation-based key provisioning protocol specifications:

1. $G \rightarrow C : \{N_1, ID_G\}$

   G sends Message 1 to C to initiate a protocol session. The goal is to establish a trust relationship and a corresponding symmetric key to represent such trust. To construct Message 1, G generates a random nonce N_1 and sends it together with its identifier ID_G to C in plaintext.

2. $C \rightarrow G : \{N_1, ID_G, N_2, ID_C, ID_P\}$

   Currently, C has no knowledge regrading G, but sends back Message 2 to direct G to P for further authentication and delegation. Message 2 contains identity information for all three parties, and initial random nonces from C and G for a protocol session.

3. $G \rightarrow P : \{N_1, ID_G, N_2, ID_C, ID_P\}$

   Upon receiving Message 2 from C, G first verifies N_1 and ID_G are from Message 1 as well as no duplicate identifiers, i.e., $ID_C \neq ID_G$ and $ID_P \neq ID_G$ and $ID_C \neq ID_P$. If all the verifications succeed, G then forwards Message 2 as Message 3 to C's parent party P;

4. $P \rightarrow G : \{N_1, ID_G, N_2, ID_C, ID_P, N_3, g^x\}$

   Upon receiving Message 3 from G, P should validate the following:

   (a) All the identity information are correct.
   (b) G can be authenticated.
   (c) C is indeed its child entity.
   (d) Requested privilege on C can be authorized to G.
   (e) Authorization decision could be based on G's identity and an additional data structure for the authorization. In different embodiments, various information may be included in the authorization data structure. Such data structure may be sent and verified by P and G in protocol exchange with Messages 4–6.

   Then P generates its own nonce N_3, the Diffie-Hellman private exponent x and public value g^x to construct Message 4, which corresponds to the first message of SIGMA protocol. Message 4 includes all three nonces so as to enhance the three-party context. This works as a challenge to G to reduce the chance of a denial of service (DoS) attack on P.

5. $G \rightarrow P : \{N_1, N_2, N_3, g^y, sig_G(g^x || g^y), [ID_C, cert_G]_{ak}\}$

   Now G and P engage in a mutual authentication from Message 4 to Message 6. G first verifies that N_1, ID_G, ID_P, N_2 and ID_C are from Message 3 as well as g^x is an element of the Diffie-Hellman group. Then G sends Message 5, which corresponds to the second message of the SIGMA protocol, with the enhanced protocol context (e.g., matching identifiers and all three nonces) and its own Diffie-Hellman public value g^y. G signs Diffie-Hellman public values with its public key certificate $cert_G$, and then computes a Message Authentication Code (MAC) on G's own identity, further bound with C's identity using the derived key from both parties Diffie-Hellman values. Note that the key ak used to compute the MAC is derived by Equations (1) and (2).

$$dk = prf(hash(N_1 || N_2 || N_3), g^{xy}) \tag{1}$$

$$ak = prf(dk, 1) \tag{2}$$

6. $P \rightarrow G : \{\alpha, \beta, sig_P(g^y || g^x), [cert_P]_{ak}\}$

   P parses and validates the following in Message 5:

   (a) Verify N_1, N_2, N_3, $cert_G$ and ID_C ($ID_C \neq ID_G$) are from Message 4.
   (b) Verify $cert_G$ and G is authorized to receive a key for C.
   (c) User the public key contained in $cert_G$ to verify the signature $sig_G(g^x || g^y)$.
   (d) Verify g^y is an element of the Diffie-Hellman group.
   (e) Use ak derived as in Equations (1) and (2) to verify the MAC tag on $[ID_C, cert_G]_{ak}$.

Then P sends back Message 6, which corresponds to the third message of the SIGMA protocol, with matching protocol context, signing Diffie-Hellman public values in reverse order (to protect against reflection attacks) with its own public key certificate $cert_G$, and computes the MAC on P's own identity, using the same derived key ak. Note that Message 6 introduces additional data structures as part of the 3PKD protocol, namely two key provisioning tokens, upon making delegation decision to G:

(a) P as the key provisioning server, generates a random key material k, which will be used later by G and C to compute shared secret key sk, to be delivered to both G and C.
(b) P generates token α for G using Equation (4), which includes authenticated data for this protocol instance (all identifiers and nonces) and encrypts the key k using ek, which is derived by Equation (3).
(c) P generates an associated token β for C using Equation (5), which includes the same authenticated data for this protocol instance (all identifiers and nonces) and encrypts the key k using kek, which is the existing parent key that P has established with C, prior to this protocol execution.
(d) As the key material k is securely encapsulated into α and β, P sends both tokens to G in Message 6, in addition to SIGMA authentication data structures.

$$ek = prf(dk, 2) \tag{3}$$

$$\alpha = [N_1||ID_G||N_2||ID_P||ID_C||N_3||\{k\}]_{ek} \tag{4}$$

$$\beta = [N_1||ID_G||N_2||ID_P||ID_C||N_3||\{k\}]_{kek} \tag{5}$$

7. $G \rightarrow C : \{\beta\}$

Upon receiving Message 6, G first validates as the following:

(a) Verify N_1, ID_G, N_2, ID_P, ID_C and N_3 are from Message 5.
(b) Parse $[cert_P]_{ak}$ as $cert_P$ and tag'.
(c) User ak to verify tag'.
(d) Verify $cert_P$.
(e) Verify signature $sig_P(g^y||g^x)$.

If all the validations succeed, at this point, G and P have completed mutual authentication and established a fresh secret symmetric key ek derived from the Diffie-Hellman handshake and the current session information as in Equation (3). Then G uses ek to decrypt token α and extract the secret k, which implies G receives an authorization from P. Then G directly extracts β from Message 6, and forwards it to C as Message 7 to further confirm with C whether they can reach the agreement that P has granted authorization to this delegation bounding to the same session.

8. $C \rightarrow G : \{[sid, ID_C]_{ck}\}$

C validates the token β with matching session information (namely, N_1, ID_G, N_2, ID_P and ID_C are from Message 2), then recovers the same key material k with kek. To construct Message 8, C computes session ID sid as in Equation (6) and uses key ck derived by Equation (7) to authenticate the message. As an option, G and C could further engage in an additional exchange locally to establish a pair secret ps (ps could be $null$). This pair secret may be used as a contribution to the session information and further derivation, so that P has no knowledge of the established key between G and C;

$$sid = hash(N_1||ID_G||N_2||ID_P||ID_C||N_3||ps) \tag{6}$$

$$ck = prf(k, 1||sid) \tag{7}$$

9. $G \rightarrow C : \{[sid, ID_G]_{ck}\}$

   G parses Message 8 as sid, ID_C and MAC tag. G first verifies ID_C is from Message 2 and then verifies sid derived by Equation (6). G computes ck with Equation (7) and uses it to verify the tag. At last, G constructs Message 9 and sends it to C as an acknowledgement. Locally, G computes key sk with Equation (8) as the final outcome of the protocol.

$$sk = prf(k, 2||sid) \tag{8}$$

10. C: validate Message 9 and compute sk

    C parses Message 9 as sid, ID_G and MAC tag. C verifies ID_G is from Message 1, validates sid and then use ck to verify tag. C also computes the shared secret key sk with Equation (8).

    Message 8 and Message 9 are created by C and G, respectively and validated by G and C, respectively, to complete the 3PKD protocol to confirm the matching session information, and consistency in the provided key material from P. Therefore, all three parties reach an agreement that the delegation has been properly granted and confirmed by C, P and G, and derive sk as the final outcome of the protocol.

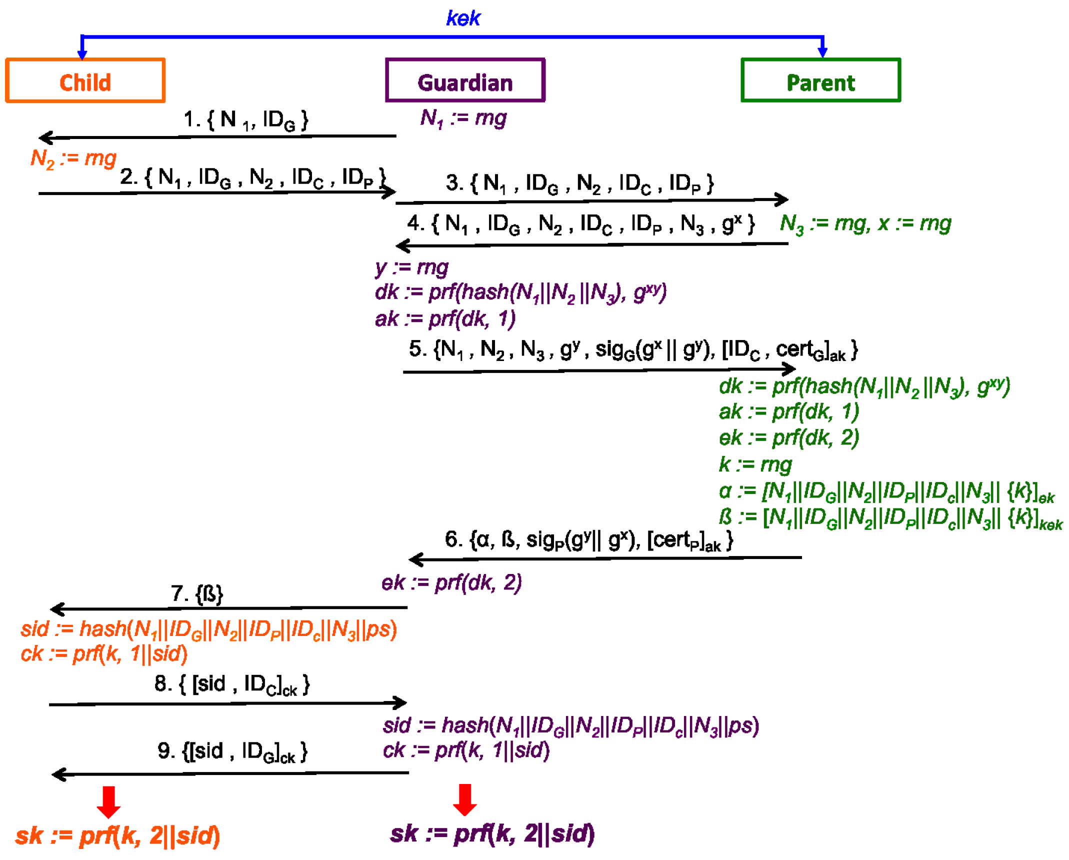

kek
Child
Guardian
Parent
1. { N 1, IDG }
N1 := rng
N2 := rng
2. { N1 , IDG , N2 , IDC , IDP }
3. { N1 , IDG , N2 , IDC , IDP }
4. { N1 , IDG , N2 , IDC , IDP , N3 , gx }
N3 := rng, x := rng
y := rng
dk := prf(hash(N1||N2||N3), gxy)
ak := prf(dk, 1)
5. {N1 , N2 , N3 , gy , sigG(gx || gy), [IDC , certG]ak }
dk := prf(hash(N1||N2||N3), gxy)
ak := prf(dk, 1)
ek := prf(dk, 2)
k := rng
α := [N1||IDG||N2||IDP||IDc||N3|| {k}]ek
ß := [N1||IDG||N2||IDP||IDc||N3|| {k}]kek
6. {α, ß, sigP(gy|| gx), [certP]ak }
ek := prf(dk, 2)
7. {ß}
sid := hash(N1||IDG||N2||IDP||IDc||N3||ps)
ck := prf(k, 1||sid)
8. { [sid , IDC]ck }
sid := hash(N1||IDG||N2||IDP||IDc||N3||ps)
ck := prf(k, 1||sid)
9. {[sid , IDG]ck }
sk := prf(k, 2||sid)
sk := prf(k, 2||sid)

Figure 9. Message flow of key provisioning protocol.

Table 1. Protocol Notations.

Symbol	Notation
rng	Random Number Generator
$\|\|$	Concatenation
N_i	i^{th} nonce generated in the protocol session
ID_X	Identifier of the protocol participant X
g	Generator of Diffie-Hellman group
x	Parent's Diffie-Hellman private component
y	Guardian's Diffie-Hellman private component
$prf(k, s)$	Pseudo random function with k as the key and s as the seed
$hash(M)$	One way hash function on material M
$sig_X(M)$	Signature on material M using the private key of X
$[M]_k$	Message Authentication Code of M using key k
$[A\|\|\{B\}]_k$	Authenticated encryption: B is encrypted and $A\|\|B$ is authenticated using key k
$cert_X$	Certificate of the protocol participant X
α	Token generated by Parent and granted to Guardian
β	Token generated by Parent and granted to Child
kek	Parent key shared between Parent and Child
k	Random key generated by Parent for Guardian and Child
dk	Intermediate component derived from Diffie-Hellman
ak	Symmetric key derived from dk and used for computing message authentication code
ek	Symmetric key derived from dk and used for protecting token α
sid	Session identifier
ck	Symmetric key derived from sid and k and shared between Guardian and Child
sk	Membership key shared between Guardian and Child
ps	Local secret shared only between Guardian and Child (could be $null$)

At this point, the protocol concludes with the successful establishment of a secret key between G and C to represent their trust relationship delegated by P. Stated another way, at completion of the protocol, all three parties confirm they reached a three-party agreement on the delegation. This is typically referred to as a matching conversation. For the agreement on the delegation, the result of the protocol is that:

(a) C agrees: P and G are corresponding parties in the delegation as P and G; and P and G know that C agrees;

(b) P agrees: G can be authorized to receive the delegation on C; and both C and G know that P agrees;

(c) G agrees: P has successfully authorized the delegation to C, and C is indeed P's child; C and P know that G agrees.

6.7. Discussions and Security Analysis

The delegation-based key provisioning protocol specified in Section 6.6 involves three parties and contains nine messages. However, in the context of the MobilityFirst-based IoT architecture, there is another round of message exchange before the execution of this key provisioning protocol. When grouping a set of IoT devices to form an IoT network or when a device is joining an existing IoT network, the IoT-NRS initiates the device discovery process by sending out registration beacons periodically. An IoT device responds to the beacon with its intention to join the network. After the beacon and the response message exchange, to register the new device and establish a long-term membership credential, the IoT-NRS initiates a new session of our key provisioning protocol by sending Message 1 of the protocol with its identifier and a fresh nonce. We do not explicitly include the first round of message exchange into the key provisioning protocol specification because: (1) the beacon and the response message exchange is sent over an open channel and is irrelevant to the security operations; and (2) the context of MobilityFirst IoT architecture does not limit the applicability of our key provisioning protocol. This protocol is general and can be applied to other scenarios, which will be discussed in Section 6.9.

Both Choo's 3PKD protocol and SIGMA protocol are proven to be secure *individually*. However, combining them together and customizing them to be applicable for resource-limited devices does not necessarily imply the result is *secure*. One of our major contributions includes providing security guarantees for overlaying Choo's 3PKD protocol and SIGMA protocol as well as adopting lightweight measures to reduce overhead. Below, we will discuss our contributions to the key provisioning protocol in detail and show that all the security requirements listed in Section 6.3 are achieved.

6.7.1. Protocol Security Analysis

The key provisioning protocol provides strong security on provisioned keys by leveraging two different protocols, namely a three-party key distribution protocol (3PKD) and a two-party SIGMA protocol, for parent-guardian authentication. These two protocols are overlaid with *enhanced protocol session context (all identifiers and nonces)* so that all protocol participants have consistent and a correct view of the protocol session. Upon the successful completion of the key provisioning protocol, each party needs to confirm its commitment to the fact that they agree on, and contribute (e.g., contribute a nonce) to the shared secret(e.g., a shared key or a seed used to generate a shared key).

Introducing enhanced protocol session context into the protocol design excludes the possibility of an identity misbinding attack, where an attacker successfully convinces the peers that the key exchange succeeded, yet the key agreement is bound by each of the participants to a wrong "peer" [44]. One of the measures adopted in the key provisioning protocol to prevent identity-related attacks, e.g., a misbinding attack, is adding identifiers in the corresponding messages when necessary. We assume the identifiers of Child C, Guardian G and Parent P are ID_C, ID_G and ID_P respectively. In the context of the MobilityFirst-IoT, there are two types of identifiers : the globally-used GUID and the locally-used membership credential. Thus, ID_G and ID_P should be the GUIDs of G and P respectively. However, as C is typically a low-end IoT edge device, it does not necessarily have a GUID. Further, before the completion of the key provisioning protocol, as C has not successfully finished the device registration yet, C does not hold a local membership credential of G's network. In fact, ID_C should be a "local" identifier recognizable by P and used to facilitate the establishment of a long-term membership key shared with G. To summarize, ID_C is neither a GUID nor a local membership key. Instead, ID_C is only a local identifier that uniquely identifies C at P. For example, assuming P is a device manufacturer and C is a product device from P, then ID_C could be the model and serial number that is stored in P's database. In other application scenarios of the key provisioning protocol, ID_G and ID_P are not necessarily GUIDs, which is specific to the design of the MobilityFirst network. Alternatively, ID_G and ID_P could be any other forms of identifier that allow G and P to identify and further authenticate each other.

The first stage of our protocol contains the first three messages and corresponds to the first stage of Choo's 3PKD protocol, in which each of the two peer parties contributes a nonce to the new protocol session. Also, as the "server" P is unknown to G at this point, C explicitly specifies its Parent P by providing ID_P. This design provides flexibility, which many other existing server-based key distribution protocols, such as Kerberos, lack. In Message 2, the identity of the Parent is specified by the Child, and hence the Guardian knows who to contact for delegation from the Child. In practice, as a Child may have several pre-existing trust relationships that can be leveraged, the Child has the freedom to direct the Guardian to any of the Parents sharing an established trust. In case one Parent is unreachable by the Guardian or the mutual trust establishment between the Guardian and the Parent fails, the Guardian may inform the Child of the delegation failure. Then the Child can in turn provide another Parent as the new referee.

In the second stage (Message 4–6), which corresponds to the SIGMA protocol, a Parent-Guardian authentication is completed using the Diffie-Hellman exchange. This authentication process, facilitated by a PKI, is essential to exclude the possible collusion of the Child and the Parent. As this authentication request is initiated by G in Message 3, P sends its Diffie-Hellman value, which can be pre-generated, as a challenge to G to reduce the chance of a denial of service (DoS) attack on P as the Diffie-Hellman algorithm requires a lot of computing resource and G has to pay the price first.

The last stage of our protocol (Message 7–9) corresponds to the second stage of the 3PKD protocol. In this stage, the key material k provided by P is delivered to C, and then G and C reach a key agreement based upon k. The protocol is concluded by a handshake between G and C so that both of them confirm that the other party has computed the correct shared secret with respect to the correct protocol session. In our embodiment, we introduced the session context in terms of *session* identifier sid, which is derived from the complete session information (all identifiers, nonces and optional local secret between G and C), and use it as a glue to combine SIGMA protocol between the first stage and the second stage of 3PKD protocol. With the approach of repeating the session information through the protocol, each of the parties has a fresh and consistent view of the current protocol session, which significantly increases the difficulty of launching attacks targeting the context of a protocol instance, such as replay attacks and misbinding attacks. At the end of the third stage, dynamic mutual trust between the Child and the Guardian and the Parent-to-Guardian delegation are achieved. Secrecy and forward secrecy are also satisfied through handling key materials properly.

The key provisioning protocol may further realize stronger security on provisioned keys as compared with existing pairing protocols. In contrast to pairing protocols that do not have a security proof (which causes them to be troublesome due to possible man-in-the-middle attacks and frequent human errors in any manual operations), the protocol as described may be performed with no manual intervention from the user for key provisioning. Compared with traditional key provisioning protocols that do not rely on a server or key infrastructure, such as secure pairing, there is no manual operation by a user on an out-of-band (OOB) channel. Instead, the device Parent party is leveraged as the trusted third party to facilitate trust establishment. Such design reduces human error, hence satisfying average administration requirements and increasing usability. Further, it removes the requirement of close proximity for key provisioning, as required by most OOB channels involving manual operations by human, and hence increases the chance of practical deployment and satisfies the scalability requirement.

6.7.2. Lightweight Approach

This key provisioning protocol is carefully designed for IoT networks as a light key solution. As one of the most important features, the protocol design favours the resource-limited Child device C in all possible means. The strategy is to minimize the number of messages C needs to process and shift the work load to those more capable devices, namely G and P, as much as possible. Consequently, the key provisioning complexity is transferred to the edge device's Parent party and the Guardian device, making it possible for minimally complex IoT edge devices to operate with secure interactions.

Compared with public key certificates, our protocol implements only lightweight symmetric key operations and secure storage for at least a single symmetric key and protected key operations with the stored key. Specifically, all of the security operations on the Child are symmetric cryptographic computations, which makes the protocol feasible for low-end IoT edge devices.

Timestamps and *nonces* are two typical security measures to guarantee the freshness of a protocol instance and to prevent replay attacks. Timestamps require strict time synchronization, and consequently increases the hardware cost as an oscillator with high accuracy is not necessarily cheap. On the other hand, nonces require a (pseudo) random generator and some form of database to retrieve if a nonce appeared before. As a comparison, nonces are relatively cheaper, and hence we chose to use nonces to protect against replay attack. Unlike a Kerberos system, the Child device is not required to have secure clock synchronization.

Furthermore, the protocol can be implemented with a minimal number of protocol messages handled by the child device. The protocol can thus minimize the computation and power costs to a Child device, such that lightweight devices like sensors, actuators, and microcontroller units (MCUs) can be provisioned with secret keys securely. In addition, the first four messages are sent as plaintext in open channels to reduce overhead. Modifying and/or replaying the messages in the first stage will result in protocol abortion later as each piece of session information will be verified in the following stage.

6.8. Proof of Concept

In order to validate the proposed protocol, we implemented a prototype of the delegation-based key provisioning protocol. The discussion includes the framework design, implementations choices as well example results from the implementation demonstration.

6.8.1. Modularized Prototype Framework

Figure 10 shows the modularized prototype framework design. Each party's protocol package includes three major components: *Instance Manager*, *Protocol Manager* and *Utility Functions*. Each party should be able to work independently to perform tasks, such as receive messages, parse messages, perform designated computations, construct messages and send messages. In the protocol system model, there are two client/server based communication channels. For the channel between the Guardian and the Child, the Guardian serves as the "server", while the Child is the "client". On the other channel between the Parent and the Guardian, the Parent serves as the "sever" while the Guardian is the "client".

Modularized design allows easy adjustment if the protocol design changes or the parameters change. The instance manager creates an instance of the corresponding party at the device, and then performs configurations, e.g., setting up a port number.

The protocol manager manages all of the sessions of the protocol instance on the device. The protocol manager consists of four modules: *Session Manager*, *Finite State Machine*, *Networking* and *Message*. As one party may open more than one protocol session related to different peer parties, the *Session Manager* is introduced to coordinate different sessions and handle related operations, including initiating a new protocol session, terminating an active protocol session, update and query local session database. For example, the Guardian as a local group authority may receive several registration requests from different edge IoT devices so that it needs to open multiple protocol sessions (one for each requesting IoT device), which requires a session manager to coordinate. This module is also necessary even for edge IoT devices as one edge device as a Child might join two or more different IoT networks at the same time. The *Networking* module is responsible for receiving and transmitting encoded messages through the network interface. The *Message* module parses and constructs messages as well as performs related cryptographic computations. The *Finite State Machine* (FSM) is the core module that handles protocol state transitions. We incorporate an event-driven FSM to take actions upon input events, including a message being constructed and sent successfully, successfully receiving a message, time out, result of parsing and verifying a message, etc.

Utility Functions component provides two categories of utility functions: *Message Encoding/Decoding*, which provide API for the *Message* module to parse and construct messages, and *Crypto Library Wrapper*, which wraps the underlying cryptography library and provides cryptographic computation APIs for the *Message* module.

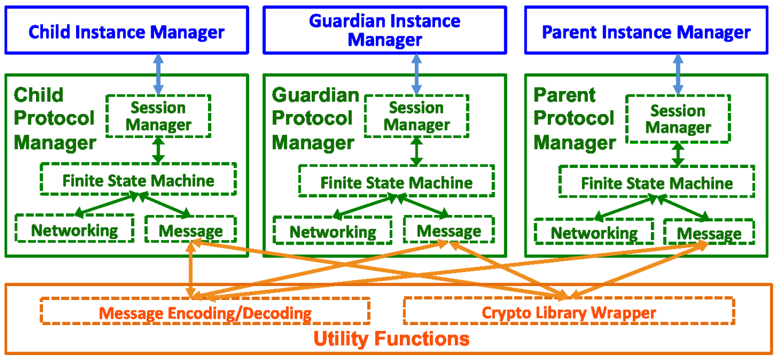


Figure 10. Modularized prototype framework design: the framework includes three major modules (instance manager, protocol manager and utility functions) for three protocol participants.

6.8.2. Implementation Choices and Considerations

As for the underlying crypto library, we mainly investigated two lightweight crypto libraries: wolfSSL (previously known as CyaSSL) [55] and mbed SSL (formerly known as PolarSSL) [56]. Both of them are lightweight embedded SSL libraries, providing cryptographic and SSL/TLS/DTLS capabilities for IoT devices with minimal coding overhead. We implemented two suites of crypto library wrappers, one for wolfSSL and one for mbed SSL. At the point of implementing the protocol prototype, to the best of our knowledge, mbed SSL is more stable and easier to use with respect to the cryptographic algorithms used in our protocol. In the final protocol prototype, we adopted mbed SSL as the underlying crypto library and its wrapper as crypto/networking APIs to facilitate the crypto and networking operations.

In the protocol specifications, authenticated encryption is used to protect two tokens carrying the secret key material k as well as the session information. The prototype adopts AES Galois/Counter Mode (GCM) [57] as the authenticated encryption algorithm, which is a block cipher with both encryption and authentication functionalities. With AES-GCM, the key material k is first encrypted and then authenticated together with the session information to generate an authentication tag. As a result, the token α and β consist of three parts: a plaintext of the session information, a ciphertext of the key material and an authentication tag.

The session manager module has a responsibility of monitoring and checking the session information. For example, in a new session, when a party receives the fresh session information, namely relevant identifiers and nonces, from an incoming message, it should first validate the freshness and the correctness of the session information. Specifically, it is important to verify whether the nonce has appeared before. Therefore, creating and maintaining a local database to store and retrieve all the past session information is necessary. Currently, we simply use a database file to store all the previous session data. The session manager looks up and updates the database file upon receiving the session data from a fresh protocol session.

Major implementation choices for the cryptography algorithms is listed in Table 2 and parameter settings are listed in Table 3.

Table 2. Cryptograhpy Algorithm Choices.

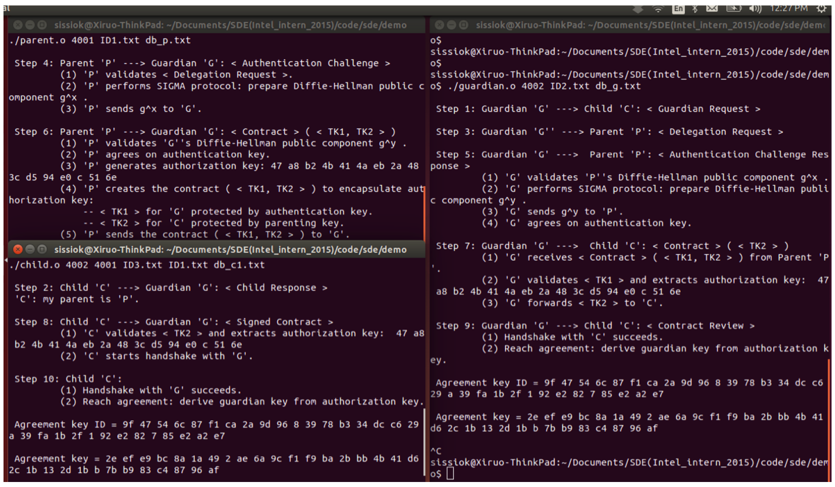
Cryptography Operation	**Algorithm Choice**
hash	SHA-256
pseudo random function (prf)	HMAC-SHA256
Diffie-Hellman Groups	2048-bit MODP Group [58]
asymmetric crypto	2048 bits RSA
Message Authentication Code (MAC)	HMAC-SHA256
Certificate standard	X.509

Table 3. Cryptograhpic Parameter Settings.

Cryptographic Parameter	Settings
symmetric key size	32 bytes
key identifier size	32 bytes
identifier size	32 bytes
MAC tag size	32 bytes
AES-GCM tag size	16 bytes
SHA-256 digest size	32 bytes
RSA key size	2048 bits
Diffie-Hellman key size	2048 bits

6.8.3. Demo of the Prototype

In this section, we present the demo output (shown in Figure 11) at Child, Guardian and Parent respectively. The demo was run on a single Linux machine so that we used different port number to differentiate the different communication channels. In practice, IP addresses and port numbers are usually used instead of purely port numbers.



```
sissiok@Xiruo-ThinkPad: ~/Documents/SDE(Intel_intern_2015)/code/sde/demo
./parent.o 4001 ID1.txt db_p.txt

 Step 4: Parent 'P' ---> Guardian 'G': < Authentication Challenge >
         (1) 'P' validates < Delegation Request >.
         (2) 'P' performs SIGMA protocol: prepare Diffie-Hellman public c
omponent g^x .
         (3) 'P' sends g^x to 'G'.

 Step 6: Parent 'P' ---> Guardian 'G': < Contract > ( < TK1, TK2 > )
         (1) 'P' validates 'G''s Diffie-Hellman public component g^y .
         (2) 'P' agrees on authentication key.
         (3) 'P' generates authorization key: 47 a8 b2 4b 41 4a eb 2a 48
3c d5 94 e0 c 51 6e
         (4) 'P' creates the contract ( < TK1, TK2 > ) to encapsulate aut
horization key:
             -- < TK1 > for 'G' protected by authentication key.
             -- < TK2 > for 'C' protected by parenting key.
         (5) 'P' sends the contract ( < TK1, TK2 > ) to 'G'.
```

```
sissiok@Xiruo-ThinkPad: ~/Documents/SDE(Intel_intern_2015)/code/sde/demo
./child.o 4002 4001 ID3.txt ID1.txt db_c1.txt

 Step 2: Child 'C' ---> Guardian 'G': < Child Response >
 'C': my parent is 'P'.

 Step 8: Child 'C' ---> Guardian 'G': < Signed Contract >
         (1) 'C' validates < TK2 > and extracts authorization key:  47 a8
 b2 4b 41 4a eb 2a 48 3c d5 94 e0 c 51 6e
         (2) 'C' starts handshake with 'G'.

 Step 10: Child 'C':
         (1) Handshake with 'G' succeeds.
         (2) Reach agreement: derive guardian key from authorization key.

 Agreement key ID = 9f 47 54 6c 87 f1 ca 2a 9d 96 8 39 78 b3 34 dc c6 29
a 39 fa 1b 2f 1 92 e2 82 7 85 e2 a2 e7

 Agreement key = 2e ef e9 bc 8a 1a 49 2 ae 6a 9c f1 f9 ba 2b bb 4b 41 d6
2c 1b 13 2d 1b b 7b b9 83 c4 87 96 af
```

```
sissiok@Xiruo-ThinkPad: ~/Documents/SDE(Intel_intern_2015)/code/sde/dem
o$
sissiok@Xiruo-ThinkPad:~/Documents/SDE(Intel_intern_2015)/code/sde/dem
o$
sissiok@Xiruo-ThinkPad:~/Documents/SDE(Intel_intern_2015)/code/sde/dem
o$ ./guardian.o 4002 ID2.txt db_g.txt

 Step 1: Guardian 'G' ---> Child 'C': < Guardian Request >

 Step 3: Guardian 'G'' ---> Parent 'P': < Delegation Request >

 Step 5: Guardian 'G' --->  Parent 'P': < Authentication Challenge Res
ponse >
         (1) 'G' validates 'P''s Diffie-Hellman public component g^x .
         (2) 'G' performs SIGMA protocol: prepare Diffie-Hellman publi
c component g^y .
         (3) 'G' sends g^y to 'P'.
         (4) 'G' agrees on authentication key.

 Step 7: Guardian 'G' --->  Child 'C': < Contract > ( < TK2 > )
         (1) 'G' receives < Contract > ( < TK1, TK2 > ) from Parent 'P
'.
         (2) 'G' validates < TK1 > and extracts authorization key:  47
 a8 b2 4b 41 4a eb 2a 48 3c d5 94 e0 c 51 6e
         (3) 'G' forwards < TK2 > to 'C'.

 Step 9: Guardian 'G' ---> Child 'C': < Contract Review >
         (1) Handshake with 'C' succeeds.
         (2) Reach agreement: derive guardian key from authorization k
ey.

 Agreement key ID = 9f 47 54 6c 87 f1 ca 2a 9d 96 8 39 78 b3 34 dc c6
29 a 39 fa 1b 2f 1 92 e2 82 7 85 e2 a2 e7

 Agreement key = 2e ef e9 bc 8a 1a 49 2 ae 6a 9c f1 f9 ba 2b bb 4b 41
d6 2c 1b 13 2d 1b b 7b b9 83 c4 87 96 af

^C
sissiok@Xiruo-ThinkPad:~/Documents/SDE(Intel_intern_2015)/code/sde/dem
o$
```

Figure 11. A demonstration example illustrating the message flow and the session information printout at the Guardian, the Child and the Parent.

At the beginning, the Parent P configured itself as a server and opened a listening socket waiting for the Guardian G. G played a dual role and thus configured itself as a server for the Child C and as a client of P. C was configured as a client of G. In the demo, we used the TCP socket provided by the mbed SSL library as the networking socket. In a practical IoT network deployment, this can be replaced by other networking sockets customized for IoT environments, e.g., DTLS [59]. Also, there is an additional Message 0 before the first message of the protocol to initiate the connection from the client (Child) to the server (Guardian).

Below we briefly illustrates the message exchange flows shown in Figure 11.

- **Step 1**: G initiates the protocol and sends the first message $<$ *Guardian Request* $>$ to C.
- **Step 2**: C replies to G with $<$ *Child Response* $>$ indicating the identity of P.
- **Step 3**: G sends $<$ *Delegation Request* $>$ to P.
- **Step 4**: P performs SIGMA protocol and sends $<$ *Authentication Challenge* $>$ to G.
- **Step 5**: G replies P's challenge with $<$ *Authentication Challenge Response* $>$.
- **Step 6**: P grants two tokens to G in $<$ *Contract* $(TK1, TK2)$ $>$. P close the socket.
- **Step 7**: G forwards $<$ *Contract* $(TK2)$ $>$ to C.

- **Step 8**: C starts the final handshake to confirm the key agreement in $<$ *Signed Contract* $>$ with G.
- **Step 9**: G finishes the handshake with $<$ *Contract Review* $>$ to confirm the key agreement with C.
- **Step 10**: C confirms the success of the handshake and key agreement.

At the end, we can see the key ID, which is derived from the session data, and the key material are consistent at both the Child and the Guardian.

6.9. Use Cases

The delegation-based key provisioning protocol is not only a lightweight membership enrolment protocol specified for the MobilityFirst-based IoT architecture, but can also serve as the cryptographic core of general-purpose key provisioning, where established keys may represent various trust relationships with different scope, semantics and criticality/impact.

Delegation of key provisioning in accordance with our protocol assumptions can be applied to a wide variety of use cases. As an example, the delegation techniques can also be used for a special function delegation. For instance, a symmetric attestation scheme may involve the Parent as an established Verifier to delegate the capability for the Guardian (a new Verifier) to challenge and verify the Prover (Child) using a symmetric attestation key. Embodiments may further support provisioning an additional Verifier (Guardian) for the same Prover. In such cases, a fresh and unique key may be established between the Prover and the new Verifier, with the original Verifier's authorization and delegation. A device provisioning function that establishes a new root of trust key for a device could also use the above protocol.

Another example can be applied to an ownership transfer protocol, which allows the original owner (Parent) to transfer all its privileges on the target device (Child) to the new owner (Guardian). In this case, a new ownership key is established on the Child, and replaces the old ownership key as the root of trust.

To summarize, with respect to key management, different keys have different usage purposes and thus key semantics are essential to clarifying any ambiguity. Our delegation-based key provisioning protocol can work with various key semantics and provide a lightweight approach to establish key material for IoT systems.

7. Conclusions

In this paper, we first investigated existing IoT solutions and analyzed their disadvantages. We focused in depth on security aspects of the various IoT architectures. Starting with a general security analysis, we identified security and privacy concerns unique to IoT systems. Then we proposed a unified IoT architecture design based on the MobilityFirst network that addresses security concerns and increases confidence about the assurable operation of the Internet of Things. We introduced a new layer in the architecture, which we refer to as the IoT middleware, that connects heterogeneous hardware in local IoT systems to the global MobilityFirst network. One of the core components of the IoT middleware is the IoT-NRS (IoT name resolution service), which is a device registration service that provides naming and key management. Further, we developed a delegation-based key provisioning protocol that establishes a long-term membership key between an IoT device and the IoT-NRS, with the assistance of a trusted third party. The protocol is specifically designed to be trustworthy while also being lightweight. Moreover, this protocol can support many functions that might be expected in IoT systems, such as ownership transfer, or establishment of an attestation key. Our efforts extend the Internet to include the real world objects and therefore facilitates the formation of a large and new network inclusive of embedded devices.

Acknowledgments: Support from the United States National Science Foundation Future Internet Architecture project (grants CNS-134529 and CNS-1040735) is gratefully acknowledged.

Author Contributions: Xiruo Liu, Sugang Li, Feixiong Zhang and Wade Trappe designed the MobilityFirst-based Internet of Things architecture; Xiruo Liu and Meiyuan Zhao designed and evaluated the delegation-based key

provisioning protocol and its prototype; Xiruo Liu implemented the prototype; Xiruo Liu and Wade Trappe wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. Ashton, K. That 'Internet of Things' Thing. *RFID J.* **2009**, *22*, 97–114.
2. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A Survey. *Comput. Netw.* **2010**, *54*, 2787–2805.
3. Giusto, D.; Lera, A.; Morabito, G.; Atzori, L. *The Internet of Things*; Springer: New York City, NY, USA, 2010.
4. Strategic Business Insights (Firm). *Six Technologies with Potential Impacts on US Interests out to 2025*; Technical Report; The National Intelligence Council: Washington, DC, USA, 2008.
5. Federal Trade Commission. *Internet of Things—Privacy and Security in a Connected World*; FTC: Seattle, WA, USA, 2013.
6. *Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020*; Gartner Inc.: Stamford, CT, USA, 2013.
7. MobilityFirst Future Internet Architecture. Available online: http://mobilityfirst.winlab.rutgers.edu/ (accessed on 19 June 2017).
8. eXpressive Internet Architecture. Available online: https://www.cs.cmu.edu/~xia/ (accessed on 19 June 2017).
9. Named Data Networking. Available online: http://named-data.net/ (accessed on 19 June 2017).
10. Nebula Future Internet Architecture Project. Available online: http://nebula-fia.org (accessed on 19 June 2017).
11. CoRE Working Group *DF-1 Protocol and Command Set Reference Manual*; IETF Standards; Fremont, CA, USA, 1996.
12. MelsecNet. Available online: https://eu3a.mitsubishielectric.com/fa/en/ (accessed on 19 June 2017).
13. Distributed System (SDS). Available online: http://holjeron.com/products/sds-products/ (accessed on 19 June 2017).
14. Newman, M. *BACnet: the Global Standard for Building Automation and Control Networks*; Momentum Press: New York City, NY, USA, 2013.
15. European Telecommunications Standards Institute (ETSI). *Low Throughput Networks (LTN): Functional Architecture*; ETSI GS LTN 002 v1.1.1; European Telecommunications Standards Institute (ETSI): Valbonne, France, 2014.
16. Global Sensor Networks. Available online: https://github.com/LSIR/gsn/wiki (accessed on 19 June 2017).
17. Aberer, K.; Hauswirth, M.; Salehi, A. Infrastructure for Data Processing in Large-scale Interconnected Sensor Networks. In Proceedings of the IEEE International Conference on Mobile Data Management, Mannheim, Germany, 7–11 May 2007; pp. 198–205.
18. Shelby, Z. *RFC6690: Constrained RESTful Environments (CoRE) Link Format*; IETF Standards; CoRE Working Group: Fremont, CA, USA, 2012.
19. Shelby, Z.; Hartke, K.; Bormann, C.; Frank, B. *RFC7252: The Constrained Application Protocol (CoAP)*; IETF Standards; CoRE Working Group: Fremont, CA, USA, 2014.
20. Lee, C.T.; Yang, C.H.; Chang, C.M.; Kao, C.Y.; Tseng, H.M.; Hsu, H.; Chou, P.H. A Smart Energy System with Distributed Access Control. In Proceedings of the IEEE International Conference on Internet of Things, Cambridge, MA, USA, 6–8 October 2014.
21. Cirani, S.; Picone, M.; Gonizzi, P.; Veltri, L.; Ferrari, G. IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sens. J.* **2015**, *15*, 1224–1234.
22. Blazquez, A.; Tsiatsis, V.; Vandikas, K. Performance Evaluation of OpenID Connect for an IoT Information Marketplace. In Proceedings of the 81st IEEE Vehicular Technology Conference (VTC Spring), Glasgow, UK, 11–14 May 2015; pp. 1–6.
23. Seitz, L.; Selander, G.; Gehrmann, C. Authorization Framework for the Internet-of-Things. In Proceedings of the 14th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Madrid, Spain, 4–7 June 2013; pp. 1–6.
24. Fotiou, N.; Kotsonis, T.; Marias, G.F.; Polyzos, G.C. Access Control for the Internet of Things. In Proceedings of the International Workshop on Secure Internet of Things (SIoT 2016), Crete, Greece, 27 September 2016; pp. 29–38.

25. Gerdes, S.; Bergmann, O.; Bormann, C. *Delegated CoAP Authentication and Authorization Framework (DCAF)*; IETF Internet Draft: Fremont, CA, USA, 2015.
26. Nitti, M.; Pilloni, V.; Colistra, G.; Atzori, L. The Virtual Object as a Major Element of the Internet of Things: A Survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1228–1240.
27. oneM2M. Available online: https://www.onem2m.org/ (accessed on 20 June 2017).
28. Li, J.; Zhang, Y.; Nagaraja, K.; Raychaudhuri, D. Supporting Efficient Machine-to-machine Communications in the Future Mobile Internet. In Proceedings of the IEEE Wireless Communications and Networking Conference Workshops (WCNCW), Paris, France, 1–4 April 2012; pp. 181–185.
29. Li, S.; Zhang, Y.; Raychaudhuri, D.; Ravindran, R. A Comparative Study of MobilityFirst and NDN based ICN-IoT Architectures. In Proceedings of the 10th IEEE International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine), Rhodes Island, Greece, 18–19 August 2014; pp. 158–163.
30. Trappe, W.; Howard, R.; Moore, R.S. Low-Energy Security: Limits and Opportunities in the Internet of Things. *IEEE Secur. Priv.* **2015**, *13*, 14–21.
31. Zhang, F.; Nagaraja, K.; Zhang, Y.; Raychaudhuri, D. Content Delivery in the MobilityFirst future Internet Architecture. In Proceedings of the 35th IEEE Sarnoff Symposium (SARNOFF), Newark, NJ, USA, 21–22 May 2012; pp. 1–5.
32. Su, K.; Bronzino, F.; Ramakrishnan, K.; Raychaudhuri, D. MFTP: A Clean-Slate Transport Protocol for the Information Centric MobilityFirst Network. In Proceedings of the 2nd ACM International Conference on Information-Centric Networking, San Francisco, CA, USA, 30 September–2 October 2015; pp. 127–136.
33. Li, S.; Zhang, Y.; Raychaudhuri, D.; Ravindran, R.; Zheng, Q.; Dong, L.; Wang, G. IoT Middleware Architecture over Information-Centric Network. In Proceedings of the 2015 IEEE Globecom Workshops, San Diego, CA, USA, 6–10 December 2015; pp. 1–7.
34. Moskowitz, R.; Nikander, P. *Host Identity Protocol (HIP) Architecture*; IETF Internet Standard, RFC 4423; The Internet Society: Fremont, CA, USA, 2006.
35. Andersen, D.G.; Balakrishnan, H.; Feamster, N.; Koponen, T.; Moon, D.; Shenker, S. Accountable Internet Protocol (AIP). In Proceedings of the ACM SIGCOMM Conference on Data Communication, Seattle, WA, USA, 17–22 August 2008.
36. Pan, J.; Jain, R.; Paul, S.; So-In, C. MILSA: A New Evolutionary Architecture for Scalability, Mobility, and Multihoming in the Future Internet. *IEEE J. Sel. Areas Commun.* **2010**, *28*, 1344–1362.
37. Liu, X.; Trappe, W.; Zhang, Y. Secure Name Resolution for Identifier-to-Locator Mappings in the Global Internet. In Proceedings of the 22nd IEEE International Conference on Computer Communications and Networks (ICCCN), Nassau, Bahamas, 30 July–2 August 2013; pp. 1–7.
38. Liu, X.; Trappe, W.; Lindqvist, J. A Policy-driven Approach to Access Control in Future Internet Name Resolution Services. In Proceedings of the 9th ACM workshop on Mobility in the Evolving Internet Architecture, Maui, HI, USA, 7–11 September 2014; pp. 7–12.
39. GlobalPlatform. *The Trusted Execution Environment: Delivering Enhanced Security at a Lower Cost to the Mobile Market*; GlobalPlatform Inc.: Redwood City, CA, USA, 2011.
40. GlobalPlatform. *TEE Systems Architecture v1.0*; GlobalPlatform Inc.: Redwood City, CA, USA, 2011.
41. Steiner, J.G.; Neuman, B.C.; Schiller, J.I. Kerberos: An Authentication Service for Open Network Systems. In Proceedings of the USENIX Winter, Dallas, TX, USA, 12 February 1988; pp. 191–202.
42. Kaliski, B.; Arnold, T.S.; Schlafly, R.; Markowitz, M.; Yin, Y.L.; Arazi, B.; Blake, I.; Chen, L.; Finkelstein, L.; Fumy, W. et al. *IEEE Standard Specifications for Public-Key Cryptography*; IEEE Computer Society: Los Alamitos, CA, USA, 2000.
43. Canetti, R.; Krawczyk, H. Security Analysis of IKE's Signature-based Key-Exchange Protocol. In *Advances in Cryptology—CRYPTO 2002*; Springer: New York City, NY, USA, 2002; pp. 143–161.
44. Krawczyk, H. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols. In *Advances in Cryptology-CRYPTO 2003*; Springer: New York City, NY, USA, 2003; pp. 400–425.
45. Choo, K.K.R.; Boyd, C.; Hitchcock, Y.; Maitland, G. On Session Identifiers in Provably Secure Protocols. In *Security in Communication Networks*; Springer: New York City, NY, USA, 2004; pp. 351–366.
46. Needham, R.M.; Schroeder, M.D. Using encryption for authentication in large networks of computers. *ACM Commun.* **1978**, *21*, 993–999.
47. Denning, D.E.; Sacco, G.M. Timestamps in Key Distribution Protocols. *ACM Commun.* **1981**, *24*, 533–536.

48. Merkle, R.C. Secure Communications over Insecure Channels. *ACM Commun.* **1978**, *21*, 294–299.
49. Diffie, W.; Hellman, M.E. New directions in cryptography. *IEEE Trans. Inf. Theory* **1976**, *22*, 644–654.
50. Adrian, D.; Bhargavan, K.; Durumeric, Z.; Gaudry, P.; Green, M.; Halderman, J.A.; Heninger, N.; Springall, D.; Thomé, E.; Valenta, L.; et al. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; pp. 5–17.
51. Diffie, W.; Van Oorschot, P.C.; Wiener, M.J. Authentication and Authenticated Key Exchanges. *Des. Codes Cryptogr.* **1992**, *2*, 107–125.
52. Karn, P.; Simpson, W. *Photuris: Session-Key Management Protocol*; IETF Network Working Group: Fremont, CA, USA, 1999.
53. Harkins, D.; Carrel, D. *The Internet Key Exchange (IKE)*; Technical Report, RFC 2409; IETF Network Working Group: Fremont, CA, USA, 1998.
54. Kaufman, C. *Internet Key Exchange (IKEv2) Protocol*; IETF Network Working Group: Fremont, CA, USA, 2005.
55. wolfSSL. Available online: https://www.wolfssl.com/wolfSSL/Home.html (accessed on 19 June 2017).
56. ARM. mbedTLS. Available online: https://tls.mbed.org/ (accessed on 19 June 2017).
57. Dworkin, M. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. In *NIST Special Publication 800-38D*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2007.
58. Kivinen, T. *More Modular Exponential (MODP) Diffie-Hellman Groups for Internet Key Exchange (IKE)*; RFC3526; IETF Network Working Group: Fremont, CA, USA, 2003.
59. Rescorla, E.; Modadugu, N. *Datagram Transport Layer Security Version 1.2*; RFC6347; IETF Network Working Group: Fremont, CA, USA, 2012.