Fog-based Secure Communications for Low-power IoT Devices

LUCA FERRETTI, MIRCO MARCHETTI, and MICHELE COLAJANNI, University of Modena and Reggio Emilia, Italy

Designing secure, scalable, and resilient IoT networks is a challenging task because of resource-constrained devices and no guarantees of reliable network connectivity. Fog computing improves the resiliency of IoT, but its security model assumes that fog nodes are fully trusted. We relax this latter constraint by proposing a solution that guarantees confidentiality of messages exchanged through semi-honest fog nodes thanks to a lightweight proxy re-encryption scheme. We demonstrate the feasibility of the solution by applying it to IoT networks of low-power devices through experiments on microcontrollers and ARM-based architectures.

CCS Concepts: • **Security and privacy → Security protocols**; *Public key encryption*; • **Networks → Security protocols**;

Additional Key Words and Phrases: Security, fog computing, Internet-of-things, publish/subscribe, encryption, proxy re-encryption, secure communications, confidentiality, authorization

ACM Reference format:
Luca Ferretti, Mirco Marchetti, and Michele Colajanni. 2019. Fog-based Secure Communications for Low-power IoT Devices. *ACM Trans. Internet Technol.* 19, 2, Article 27 (March 2019), 21 pages.
https://doi.org/10.1145/3284554

1 INTRODUCTION

The number and heterogeneity of interconnected devices including wireless sensors, wearable devices, household appliances, and smart objects are continuously increasing. However, these key enablers of the modern society open new vulnerabilities and insecure scenarios. Indeed, the inherent limitations of storage, computational capacity, and energy consumption characterizing most devices prevent the adoption of secure protocols over large networks of low-power objects. The deployments based on platforms where objects establish a secure communication channel with a cloud service, such as Amazon Web Services IoT [1] and Google Cloud IoT [25], require an always-on Internet connectivity, increase network latencies and augment mobile network expenses. If the cloud service is temporarily unavailable, then even communications among local objects become impossible. In many scenarios, such as health-care, home automation, and control of industrial processes, this drawback is not acceptable. The fog computing paradigm aims to address issues related to cloud-assisted architectures by moving some operational capabilities from the cloud toward the edge of the network. All applications requiring low latency, local storage and filtering

Authors' addresses: L. Ferretti, M. Marchetti, and M. Colajanni, University of Modena and Reggio Emilia, Via Vivarelli 10, Modena 41125, Italy; emails: {luca.ferretti, mirco.marchetti, michele.colajanni}@unimore.it.

1533-5399/2019/03-ART27 $15.00
https://doi.org/10.1145/3284554

of sensitive data, and high resiliency in terms of ability to operate while objects are disconnected from the cloud can benefit of the fog paradigm.

This article proposes a novel fog-based architecture that enables secure local communications among IoT devices. The proposed solution combines the flexibility and security of cloud-based IoT deployments with the scalability and resiliency of fog computing. Existing proposals assume that the fog network is responsible for establishing secure communications among things and that each fog node is trusted and secure against external attackers [19]. This assumption is unsuitable for devices that are often deployed in adversarial environments and exposed to a wider range of attacks. The naïve solution of allowing direct and secure communications among things is inapplicable to low-power devices, because it requires key management and public key cryptographic algorithms that are computationally too expensive for most objects. In the proposed protocol, things leverage the computational capabilities of semi-honest fog nodes to exchange encrypted messages. While fog nodes guarantee scalability and resiliency of the system, the protocol guarantees end-to-end confidentiality of the information exchanged among IoT devices. The protocol is based on an original symmetric proxy re-encryption scheme that is suitable to the limited computational capacity of many connected low-power things. This article has four main contributions to the state of the art that we summarize below.

We design a novel fog architecture that moves a portion of the responsibility of implementing message delivery from cloud services to local fog nodes. Our proposal extends current fog architectures to support thing-to-thing secure communications without requiring neither always-on connectivity with a cloud service nor strong security assumptions on all fog nodes. We assume fog nodes to be semi-honest participants that must not access to the contents of the messages exchanged among things. This design choice is an improvement with respect to previous work in which local communications among things are enabled by trusted components that act as key distribution centers [40] or execute expensive cryptographic operations on behalf of low-power devices [27].

We design novel communication protocols for the management of cryptographic materials required to uphold thing-to-thing confidentiality of exchanged messages. The main idea is to leverage proxy re-encryption to design a scalable publish/subscribe communication scheme in which a device stores one encryption key and executes affordable encryption operations independently of the number of things with which it has to communicate. Related work usually focus on attribute-based encryption that is applicable just to powerful devices, such as smartphones, and cannot be executed on low-power things [2]. Other solutions based on proxy re-encryption [3, 24, 55, 61] are inapplicable to the proposed scenario due to computational or architectural constraints.

We design and implement an original symmetric proxy re-encryption scheme that is applicable to low-power microcontrollers characterizing many connected devices. Experimental evaluations show that even an Arduino Uno with an 8-bit microcontroller has enough computational capacity to encrypt and decrypt messages according to the proposed proxy re-encryption scheme. This result is achieved by leveraging careful design and optimization of the scheme based on the Edward25519 elliptic curve. Implementations have been proposed in the context of key exchange protocols for low-power devices [21, 38, 48]. To the best of our knowledge, this is the first implementation of a symmetric proxy re-encryption protocol for microcontrollers and energy-efficient ARM architectures.

Finally, this is the first proposal supporting secure thing-to-thing communications among devices that depend on different cloud services, enabling scenarios where federated cloud providers allow the exchange of information among their associated devices.

The article is organized as follows. Section 2 compares our proposal with related work. Section 3 discusses system and threat models of the reference fog computing scenario. Section 4 describes the

proposed fog-based architecture and protocols. Section 5 discusses the novel proxy re-encryption algorithm based on elliptic curve cryptography and applicable to low-end microcontrollers. Section 6 describes how the proposed protocol supports secure thing-to-thing communications among objects associated with different cloud services. Section 7 presents the experimental results demonstrating the applicability of the proposed solution to real low-power devices. Finally, Section 8 discusses conclusions and future work.

2 RELATED WORK

Guaranteeing communications security in IoT networks is a multifaceted research area involving many aspects and scenarios. In this article, we refer to highly constrained IoT devices that have very limited computational power (they are powered by microcontrollers or energy efficient ARM architectures) and very small storage and memory (from few to hundreds of kilobytes).

Challenges include the design of scalable and secure solutions enabling administration and management of a large number of devices. Cloud-assisted IoT networks are emerging as a de-facto standard for very popular scenarios [4, 5, 62, 63]. In this paradigm, all communications are managed by the central cloud service, which acts as an intermediate to enable secure communications and to implement powerful business logic and data analytics. Despite being widely adopted, this paradigm presents some drawbacks already highlighted in the literature [62]: Internet connectivity is required also for communications among things connected to the same local network, communications are delayed due to the latency of connections to geographically distributed cloud services, and IoT devices leveraging mobile connections may incur in unnecessary networking costs. These issues should be solved by adopting distributed approaches that allow direct communications among connected things and the execution of intelligent applications near the edge of the network. The literature refers to such approaches as *distributed IoT* [49, 59] and fog computing [13, 56]. This article proposes a protocol to implement secure communications based on the fog paradigm while maintaining the benefits of cloud-assisted solutions.

Standard solutions for secure communications require either to distribute unique symmetric encryption keys to each device or to adopt a PKI approach [39]. The former solution does not scale, because it requires each device to store all the other devices keys and becomes unfeasible even for small networks, because IoT devices have little storage. The latter solution is inapplicable, because this class of devices cannot support the standard Public Key Infrastructure (PKI) due to their limited computational capabilities. Previous works on IoT security address these issues either by proposing novel architectures or novel cryptographic protocols and primitives. Architectural proposals include the introduction of trusted components within the local network, such as key distribution centers that maintain things keys and establish secure sessions among them [40] or servers that execute the most expensive computations of key exchange protocols (e.g., DTLS handshake) on behalf of things [27]. These approaches introduce a single point of trust within the local network that jeopardize communications security. State-of-the-art literature in the context of cryptographic protocols apply Attribute-Based-Encryption (ABE) [26]. Intuitively, ABE is a cryptographic primitive where parties can only decrypt contents for which they have been authorized by a policy. Theoretically, ABE fits well the IoT networks security requirements; however, it is not practical in many scenarios. Literature claims its feasibility on mid-power IoT devices (smartphones and RaspberryPi) with only few access control policies [2] but is still impractical on low-end devices, such as embedded platforms based on microcontrollers. Our proposal introduces novelties on both the architecture and cryptographic protocols. Compared to previous approaches, we do not introduce any trusted component. Moreover the proposed cryptographic protocol is based on primitives that are practical even for low-end microcontrollers (such as an Arduino Uno).

Our proposal leverages the *topic publish/subscribe* communication paradigm, that is, the de-facto standard for cloud-based IoT networks (e.g., MQTT [44]). To the best of our knowledge, the proposed protocol is the first that efficiently encrypts contents in publish/subscribe networks assuming semi-trusted message brokers. Other proposals in the context of publish/subscribe communication security focus on guaranteeing confidentiality of topics [45]. In this article, we do not apply these solutions, because they require expensive encryption schemes that would affect things performance and reduce applicability to low-end devices. We leave the design of a practical solution with topic confidentiality guarantees in IoT networks as an open challenge.

The proposed cryptographic protocol is based on (atomic) Proxy Re-Encryption (PRE) [11], that is, a family of encryption protocols where an intermediate party (the proxy) is able to transform ciphertexts to allow specific recipients to decrypt them and cannot access any plaintext information about the transformed ciphertexts. The literature proposes many applications and variants of PRE schemes. Applications of PRE include multi-device encrypted emails [11], multi-user disk encryption [3], key rotation for outsourced data [22, 61], and mixnets [24]. These proposals cannot be applied to the publish/subscribe scenario of this article, because recipients are not known in advance. We solve this issue by designing an efficient key distribution strategy that takes advantage of existing cloud services to distribute the due cryptographic material to fog nodes, even in presence of federated cloud architectures. To the best of our knowledge, this is the first article proposing these design choices. Variants of PRE schemes include PRE based on symmetric ciphers [41, 57], pairing cryptography [16, 17, 20], signatures of knowledge [55], and Key-Homomorphic Pseudo-Random Functions (KHPRF) [12, 37]. We exclude lightweight PRE schemes based on symmetric ciphers [41, 57], because they require a trusted key distribution center, and hence they are inapplicable to the considered scenario [43]. Pairing-based and signatures of knowledge proposals [16, 17, 20, 55] seem also inapplicable due to their expensive cryptographic primitives. Our proposal adopts symmetric key re-encryption based on KHPRF. In particular, the chosen scheme is based on the Decisional Diffie Hellman (DDH) hardness problem and on assuming hash functions as Random Oracles [15], which are common conjectures adopted by standard Internet protocols. We adopt the symmetric PRE schemes first proposed by Naor et al. [37] and further studied by Boneh et al. [12], based on KHPRF. This scheme uses asymmetric cryptographic primitives to build a stream cipher with the following advantages: pseudo-keys can be pre-computed to improve performance; pseudo-keys can be created through a distributed secure procedure (distributed pseudo-random functions [37]). To the best of our knowledge, this is the first article that leverages this protocol family to allow distributed and secure computation of re-encryption keys. Moreover, we describe the first implementation of symmetric PRE based on KHPRF for IoT devices. Our implementation is based on the Edward25519 elliptic curve [9, 50], that is, the current state-of-the-art for digital signatures (Ed25519) and key-exchange for mobile messaging [58]. To the best of our knowledge, this is the first article proposing the implementation of the considered PRE protocol based on this curve and demonstrating its applicability to low-end devices such as Arduino Uno and Arduino Due (in addition to RaspberryPi and x86 platforms).

Finally, other work on IoT security relates to physical security of things deployed in hostile environments, where secret information must be protected against gray- and white-box analyses [18, 34]. These issues are out of scope for this article; however, existing solutions (such as TPM) can be easily integrated with the proposed protocol.

3 SYSTEM AND THREAT MODELS

We consider the reference fog computing architecture defined by the literature and by relevant standards [13, 19, 28] and shown in Figure 1. The architecture includes three main entities: *things*, *cloud services*, and *fog nodes*.

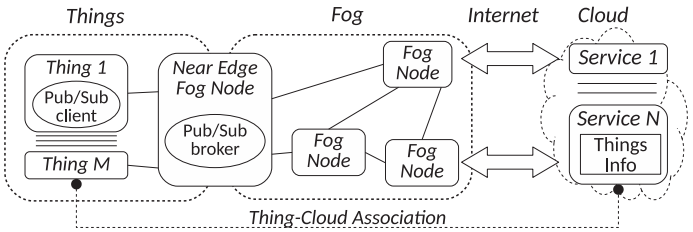

Fig. 1. Reference fog computing architectures.

- *Things* represent IoT devices [1] that are connected devices characterized by low computational capabilities and little persistent and central memory. They range from microcontrollers equipped with a few KB of memory (*class-0* devices) to low-power ARM CPUs equipped with a few hundreds of KB (*class-1* devices) [14].
- *Cloud services* (CS) are Web services accessible from the Internet by authorized users and devices. They support configuration, management, and monitoring of IoT devices and maintain the related metadata (*things information* in Figure 1). Popular cloud services for IoT include AWS IoT [1] and Google Cloud IoT [25].
- *Fog nodes* (FN) are components that can be accessed by things without requiring Internet connectivity. They provide things with additional services and resources that in traditional cloud-assisted architectures are implemented by cloud services. Fog nodes can be implemented as physical or virtual infrastructures over heterogeneous computing devices, ranging from low-power computers to clusters of powerful server machines. Examples include advanced network gateways based on medium-power ARM CPUs [6] and mini data-centers for highly available storage and data analytics [60].

We consider a configuration where things connect to fog nodes either via wireless (e.g., WiFi, Bluetooth, ZigBee) or cabled (e.g., Ethernet) protocols. Each thing is *associated* with a cloud service that maintains its identity and authentication information. We assume that things are properly configured to establish secure communications with their cloud service. However, they cannot communicate to other things or to fog nodes in a secure and scalable way due to resource constraints (see Section 2). Among all fog nodes, the *Near edge Fog nodes* (NFN) enable scalable communications among things and between things and other fog nodes. Communications are based on a *Topic-based Publish-Subscribe* (TPS) paradigm [44], where the NFN runs the broker and each thing runs a client, operating as a publisher (e.g., sensors), a subscriber (e.g., actuators), or as both.

Current proposals for fog architectures guarantee that all communications are secured end to end via cryptographic protocols between things and fog nodes, fog nodes and fog nodes, and fog nodes and cloud services. The security assumption is that all these components are trusted and protected against attacks [19]. As a result, as long as fog nodes behave honestly and are not breached by an external attacker, thing-to-thing communications can also be considered secure.

In this article, we address a more challenging threat model by weakening this security assumption and considering semi-honest fog nodes that might be interested in accessing confidential information exchanged by things. We assume that they always execute all protocols correctly and never manipulate data maliciously, but they might try to obtain confidential information from accessed data, such as the content of messages or cryptographic material. We propose a solution to enforce secure and scalable communications among things without relying on intermediate trusted components. Our proposal is related to TPS communications and guarantees confidentiality of the payloads exchanged by things against a semi-honest broker. The proposal takes into account the

nature of fog computing architectures that require scalable protocols for efficient authentication and authorization.

To establish secure communications, all things, fog nodes, and cloud services must mutually authenticate with each other by using strong credentials, such as symmetric and asymmetric cryptographic keys. We assume that cloud services act as identity providers and roots of trust to build a scalable authenticated system and that different cloud services can use identity federation systems to enable cross-authentication among their respective things and services (e.g., OpenID [46]). However, fog nodes and things must use different protocols due to their characteristics. Fog nodes do not have strict hardware constraints and can mutually authenticate against cloud services by using TLS or DTLS protocols. Things are often not able to operate [D]TLS protocols due to low computational resources, and hence they authenticate messages at the application layer (e.g., Oscore [51]). In this case, secure things communications could be forwarded as-is to the cloud service or converted to secure [D]TLS connections by cross-protocols proxies on the NFN (e.g., Oscore-DTLS proxies [53]). In this article, we assume that the information required to exchange secure messages between things and cloud services is already deployed on the devices. As an example, this information could be hard-coded at production time (e.g., a unique secret UID defined by a consortium [33]), manually configured by the system administrator (e.g., a user accessing a configuration interface on the thing), or established through semi-supervised key-exchange protocols (e.g., a PIN-based key exchange). This design choice is orthogonal to our proposal and out of the scope of the article.

Finally, we assume that things and NFNs do not know each other and must be authorized before establishing secure communications. Authorization of communications defines how things and fog nodes can communicate to mitigate risks in case of compromised components. In the considered TPS paradigm, an authorization policy is expressed as the possibility to publish and/or subscribe to certain topics. As an example, a thing is only able to subscribe to the topics allowed by authorization policies, and even compromised things cannot obtain information about denied topics. The fog architecture represents a distributed authorization scenario, where cloud services maintain all authorization policies and fog nodes must enforce them.

In this article, we consider solutions based on the OAuth2 delegated authorization framework, for which additional variants are being defined to comply with the limited resources and unstable connectivity [52]. We propose a novel delegated authorization protocol for fog architectures that is applicable to the considered scenario of TPS operated by semi-honest fog nodes. We candidate our proposal as an extension of state-of-the-art protocols for Authentication and Authorization in Constrained Environments (ACE) that rely on trusted brokers [52].

4 FOG-BASED PROXY RE-ENCRYPTION ARCHITECTURE

We describe the novel fog-based architecture for secure and scalable communications among IoT devices. The proposal is based on a proxy re-encryption scheme to enable semi-trusted fog nodes to forward messages among things and includes novel delegated authorization protocols to maintain the ease of management of cloud services while taking advantage of the distributed nature of fog. In this section, we consider a scenario composed by a fog architecture associated to one cloud service, while extensions for multi-cloud scenarios are discussed in Section 6.

We describe our proposal as follows: Section 4.1 outlines symmetric proxy re-encryption; Section 4.2 describes the main components of the proposed architecture and provides the overview of the main operations; Section 4.3 describes the operations required to initialize all the devices and services; Section 4.4 describes the protocol required to authorize the fog node to act as the broker of the network; and Section 4.5 describes the topic-based publish/subscribe protocol extended with proxy re-encryption.

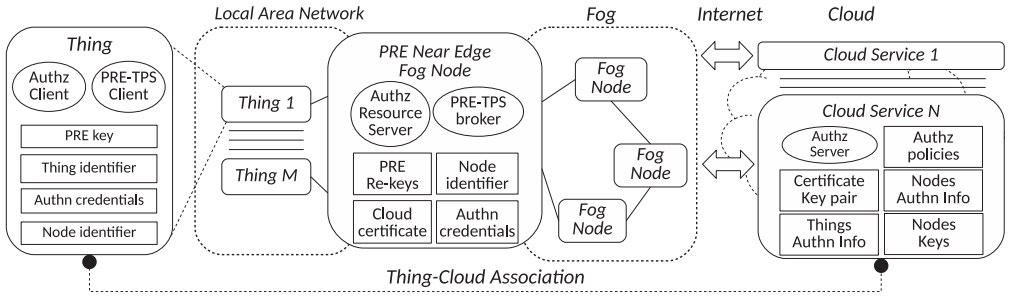

Fig. 2. Proposed fog-based IoT architecture based on proxy re-encryption topic-based publish/subscribe communications.

4.1 Base Knowledge on Proxy Re-encryption

The proposed solution is based on a proxy re-encryption (cryptographic) scheme that allows a semi-trusted proxy to transform a ciphertext computed under a secret key to another ciphertext that represents the same plaintext encrypted under a different key. The proxy does not obtain any information about the plaintext data. In this article, we consider proxy re-encryption schemes for the symmetric setting, where the same key is used for encryption and decryption operations.

The symmetric proxy re-encryption scheme includes five operations:

- $s \leftarrow pre.keygen()$: generates a new random secret key s;
- $c \leftarrow pre.encrypt(r, s, p)$: encrypts a plaintext p to the ciphertext c by using the secret key s and the nonce r;
- $p \leftarrow pre.decrypt(r, s, c)$ decrypts a ciphertext c to the plaintext p by using the secret key s and the nonce r;
- $\Delta \leftarrow pre.rekey(s_1, s_2)$: computes a re-encryption key Δ that allows to transform ciphertexts encrypted under the secret key s_1 to ciphertexts encrypted under the secret key s_2;
- $c_2 \leftarrow pre.reencrypt(r, \Delta, c_1)$: re-encrypts the ciphertext c_1 (under the secret key s_1) to the ciphertext c_2 (under the secret key s_2) by using the re-encryption key Δ and the nonce r.

Note that the scheme is bi-directional: To compute the reverse re-encryption operation one must compute the opposite of Δ and use it as the re-encryption key, as $c_1 \leftarrow pre.reencrypt(r, -\Delta, c_2)$. Moreover, note that the scheme is secure against chosen-plaintext attacks (CPA) and thus secure against passive (also, semi-honest) attackers. Although other PRE schemes exist that are secure against chosen-ciphertext attacks (CCA), they are based on cryptographic primitives that are inapplicable to constrained devices. Alternative PRE schemes are discussed in Section 2, while additional details about the proposed PRE implementation based on the Edward25519 curve are provided in Section 5.

4.2 Architecture and Information Flow

We describe the novel proposal based on PRE by referring to Figure 2, that is, an extension of current fog computing architectures discussed in Section 3. We identify three classes of devices and services:

- *Things* run the *authorization (authz) client*, which allows them to participate in authorization protocols with cloud services and fog nodes, and the *Proxy Re-Encryption Topic-based Public/Subscribe (PRE-TPS) client*, which is an extended TPS client that encrypts the payload of the messages by using a symmetric proxy re-encryption scheme. Each thing maintains an *identifier* that uniquely represents its identity and *authentication (authn) credentials*: secret

information that allows it to authenticate at the cloud service. Moreover, it maintains one *PRE key*, which is the encryption key used by the PRE-TPS client, and the *node identifier*, which identifies the fog node authorized to act as broker for the thing;

- The *near edge fog node* (NFN) runs the *authorization (authz) resource server*, which is the monitor validating things requests, and the *PRE-TPS broker*, an extended TPS broker that re-encrypts forwarded messages. The NFN maintains a list of *PRE re-encryption keys* (*re-keys*) that are required to execute re-encryption operations, a *node identifier*, and *node authentication (authn) credentials* that allow it to authenticate at the cloud service. Moreover, the *Cloud certificate* allows it to establish secure connections to the cloud service. The NFN can store information associated with multiple cloud services;
- *Cloud services* run an *authentication (authn) server* to authenticate things and fog nodes and an *authorization (authz) server* to authorize things and NFNs to communicate in compliance to *authorization policies*. They maintain *node keys*, which are cryptographic keys associated with fog nodes to enable secure communications, *certificate key pairs* to authenticate against other services, and *node* and *things authentication information*, to authenticate nodes and things, respectively.

When a thing is deployed it can only communicate by establishing a secure communication with its associated cloud service by using its *thing identifier* and *authn credentials*. To communicate with other things without using the cloud service, a thing must explicitly grant an authorization to a NFN by executing the *fog delegation flow*. This protocol allows the thing to obtain access tokens that the *authz client* can use to issue resource requests to the *authz resource server* of the NFN. Moreover, it allows the NFN to obtain the *PRE re-encryption key* to execute the re-encryption operations and forward messages. This is an interactive protocol and it is the only one requiring that the thing, the NFN, and the CS are online at the same time. After executing the fog delegation flow, things associated with the same CS can communicate with each other by using the PRE-TPS broker deployed on the NFN node in compliance to the *authorization policies* maintained by the cloud service. Things subscribe to topics, as in the common TPS paradigm. A thing that publishes content associated with a topic encrypts it by using the PRE encrypt operation and sends it to the PRE-TPS broker. The NFN receives the message and re-encrypts it for all the things that subscribed to the same topic if and only if the things are authorized to subscribe to the topic and the NFN is delegated by the things.

In the considered threat model the semi-honest NFNs might be interested in accessing information for which they are not authorized. As in typical fog computing architectures, we assume that all network connections between fog nodes and the cloud service are protected by using standard secure protocols (e.g., TLS and DTLS). Finally, following the same approach as ongoing standardization efforts for authentication and authorization protocols in access constrained environments [54], we do not make assumptions on the protocols used for local network communications and consider using cryptographic protocols at the application layer.

In the following, we describe the main operations of the proposed protocols. Please note that we do not discuss many protocol implementation technicalities that are out of the scope of the article. As an example, we assume that all the applications and services use the same cryptographic parameters, security-level, cryptographic primitives, and service interfaces.

4.3 System Setup Operations

The system setup operations include the initialization of the CS, the association of a thing to a CS, and the registration of the NFN at a CS.

Cloud service initialization: $\langle \langle sk, pk \rangle_{cs}, \mathcal{T}_{cs} \rangle \leftarrow cs.initialize(uid_{cs})$. A CS generates the parameters that are required to comply to the fog-based architecture and to offer the related services. The input of the routine is its unique identifier, which we denote as uid_{cs} (e.g., the FQDN of the service). The output of the routine includes the cloud service key pair $\langle sk, pk \rangle_{cs}$ and the set of supported topics, which we denote as $\mathcal{T}_{cs}$. We note that the topics are defined by the cloud service, and this information is usually public for existing cloud IoT services (e.g., AWS IoT topics [1]). The cloud service *key pair* $\langle sk, pk \rangle_{cs}$ is used to execute asymmetric encryption and signatures. For ease of presentation, we use a single key pair for the different protocols. As an example, we use pk_{cs} to denote the public key used to encrypt data through asymmetric encryption protocols and to verify a signature (see Section 6).

Thing association: $auth_d \leftarrow cs.associate\_thing(uid_d)$: A thing associates with a CS by providing its unique identifier, which we denote as uid_d, to obtain (authentication) credentials $auth_d$. This is an operation that already exists in cloud IoT architectures and can be executed by the device manufacturer to sell ready-to-use things (e.g., such as existing AWS-compliant IoT devices [1]) or by a user or system administrator for configurable devices. The CS can use its preferred protocol to authenticate things, and the credentials generation process is out of the scope of our proposal.

Fog node registration: $<auth_n, s_{(n,CS)}> \leftarrow cs.register\_node\,(uid_n)$: A NFN registers at a CS with its unique identifier uid_n and obtains its (authentication) credentials $auth_n$. During the registration, the cloud service generates the node key $s_{(n,CS)}$, which is the re-encryption key that the cloud service associates to the node, by using the key generation routine of the re-encryption scheme (see Section 4.1). The cloud service maintains this key locally and does not send it to the node or to any third party. Moreover, it also maintains the authentication information to authenticate the node in future communications. As for things credentials, the node credentials generation process is out of the scope of the proposed protocol.

4.4 Fog Delegation Flow

A thing executes the *fog delegation flow* to leverage a NFN to forward its messages within the network. In the following, we first describe the endpoints exposed by the NFN and by the CS, and then we describe how they are used in the fog delegation flow.

The *Node delegation endpoint*: $\pi \leftarrow n.delegate\_cs\,(uid_{CS}, DRA)$. The NFN accepts requests by a thing that include the target cloud service identifier uid_{CS} and the *delegation request attestation DRA*. The routine returns to the thing a proof of the correct execution of the delegation process, which we denote as π. The thing verifies the correctness of the proof with regard to its request by using its authentication credentials.

The *cloud service delegation endpoint*: $\langle \Delta, \pi \rangle \leftarrow cs.delegate\_n\,(uid_n, DRA)$. The cloud service accepts requests by the NFN that include the delegation request attestation *DRA* and the node identifier uid_n. The routine returns to the node the re-encryption key Δ and the delegation proof π. The endpoint must be only available on a protected channel to registered fog nodes, and the participants must mutually authenticate with each other.

We describe the fog delegation flow by referring to Figure 3. For ease of presentation, we use letters from *(a)* to *(e)* to denote the different phases.

(a) The thing executes a discovery protocol over the local network to identify the candidate node. Discovery protocols are strictly dependent on the local network protocol, and their details are out of scope and orthogonal to our proposal. We only assume that the output of the discovery protocol includes a response by the NFN with its unique identifier uid_n. This identifier must be the same used to register the node at the cloud service.

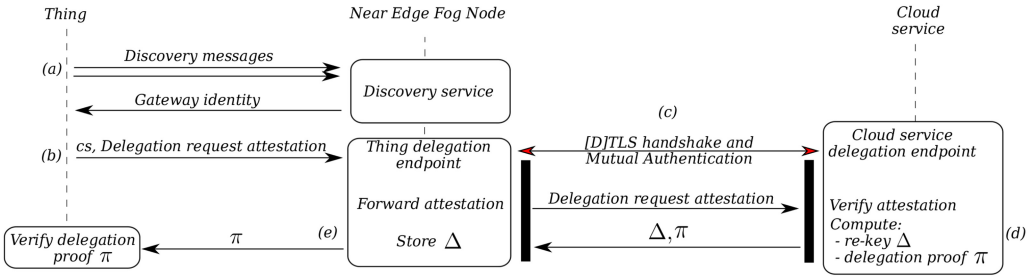

Fig. 3. Fog delegation flow protocol.

(b) The thing generates a *PRE key* defined as *thing network key*, which we denote as s_d, by using the key generation routine of the PRE scheme (see Section (4.1)). It builds a metadata packet that includes a random nonce r, the unique identifier of the thing uid_d, and the unique identifier of the node uid_n. Then, it derives an encryption key from its authentication credentials $auth_d$ with a key derivation function, which we denote as f. Depending on the type of credentials—cryptographic or password-based—key derivation should be implemented through a scoped derivation function, such as HKDF [32], or through password-based key-derivation function such as PBKDF2 [30]. The thing encrypts the PRE key and authenticates it together with the metadata packet by using a standard Authenticated Encryption with Associated Data (*AEAD*) scheme [29]. The thing builds the *delegation request attestation DRA* by sending r, uid_d, and the encrypted packet:

$$DRA = \left\langle r, uid_d, AEAD_{f(auth_d)}\left(r, s_d, \langle uid_{cs}, uid_n, uid_d\rangle\right)\right\rangle. \tag{1}$$

The output of the AEAD scheme is the encryption of key s_d and a tag that guarantees its authenticity and that binds it to the nonce r and the metadata packet $\langle uid_{cs}, uid_n, uid_d\rangle$. Finally, the thing invokes the fog delegation endpoint as $n.delegate\_cs\,(uid_{cs}, DRA)$.

(c) The node and the CS establish a secure channel and mutually authenticate. Then, the node forwards the information received by the thing to the CS by invoking the delegation endpoint as $cs.delegate\_n\,(uid_n, DRA)$.

(d) The cloud service verifies the integrity of *DRA* against the things credentials $auth_d$ by using the nonce r and the thing identifier uid_d and by adding its unique identifier uid_{cs} and the identifier of the node that authenticated on the secure channel uid_n. If verification succeeds, then the cloud service decrypts the packet, obtains the thing network key s_d, and computes the re-encryption key by using the re-encryption key computation routine $\Delta = pre.rekey\,(s_d, s_{cs})$. Moreover, the cloud service computes the proof as $\pi = MAC_{f(auth_d)}(r \parallel uid_d)$, where $\parallel$ denotes a secure operator to provide multiple inputs to the MAC function [36]. The cloud service returns Δ and π to the node, which stores Δ and forwards π to the thing.

(e) The thing receives π and re-computes it locally to verify that the cloud service authorized the node. Since the MAC is computed over secret information known only by the thing and by the cloud service, external adversaries that have access to the local network cannot forge a valid proof.

4.5 TPS Extended with Proxy Re-encryption

We define the operations to implement the topic-based publish/subscribe communication system where contents are encrypted by using the proxy re-encryption (PRE-TPS) scheme. The node exposes the *publish* and *subscribe* endpoints that expose the same interface as the standard TPS

framework and implements the forwarding module that extends TPS with re-encryption capabilities. In the following, we describe the PRE-TPS operations by considering two things, d_1 and d_2, that publish and subscribe to a topic on a node previously authorized by both things through the *fog delegation flow* (see Section 4.4). The node knows the re-encryption keys associated with the two things, which we denote as Δ_1 and Δ_2, respectively.

To publish data m associated with a topic t, a thing d_1 generates a nonce r and computes the encrypted message as $c_1 = pre.encrypt(t \parallel r, s_{d_1}, m)$. The message submitted to the PRE-TPS broker is the tuple $\langle uid_{d_1}, t, r, c_1 \rangle$. The node receives the tuple and computes an intermediate ciphertext as $c_t = pre.reencrypt(t \parallel r, \Delta_1, c_1)$. Then, the node re-encrypts the message for the thing d_2 that subscribed to the given topic as $c_2 = pre.reencrypt\,(t \parallel r, -\Delta_2, c_t)$.

The node must execute a number of re-encryption operations equal to the number of things subscribed to the topic t plus 1. Although this may seem an expensive operation that could affect the scalability of the system, by looking at the internals of the PRE implementation we note that batch of operations can be optimized through algorithms for batch computation of exponentiation operations that share a common base [47].

5 IMPLEMENTATION OF THE SYMMETRIC PROXY RE-ENCRYPTION SCHEME

We describe the implementation of the symmetric proxy re-encryption (PRE) scheme based on (twisted) Edward curves focusing on the details of the implementation for the standard Edward25519 curve that provides 128-bit security. Although our approach can be used with many elliptic curves, we chose this one as the first candidate due its very efficient and secure point addition operation. Interesting curves for future evaluations include the popular NIST curves [42] and experimental curves specifically designed for microcontrollers and proposed in the literature [48] but still not thoroughly tested. In this section, we assume that the reader is familiar with finite fields theory and related cryptographic hardness conjectures. Section 5.1 describes the implementation of the symmetric PRE scheme for elliptic curves, while Section 5.2 describes the detailed implementation based on twisted Edward curves.

5.1 Symmetric PRE Based on Key-homomorphic Pseudo-random Functions

We describe the symmetric PRE scheme proposed in Reference [12] and the hybrid variant, where messages are encrypted by also using a symmetric cipher. The security of the scheme relies on the Decisional Diffie Hellman (DDH) hardness conjecture in the random oracle model [12, 15]. These are both commonly accepted assumptions used in many standard security protocols.

We consider the additive finite field $\mathbb{G}$ of prime order q where the DDH problem is hard. The group $\mathbb{G}$ is a subgroup of a properly defined elliptic curve (see Section 5.2). Moreover, we consider a hash function that maps variable-length binary strings to the group as $\mathcal{H}_G(\cdot) : \{0,1\}^* \rightarrow \mathbb{G}$. In the following, we describe the implementation of the symmetric PRE scheme operations defined in Section 4.1. All routines operate over $\mathbb{G}$ or over $\mathbb{Z}_q$, that is, the multiplicative cyclic group built over the integers modulo the order of $\mathbb{G}$,

$$s \leftarrow pre.keygen() : \qquad s \xleftarrow{\$} \mathbb{Z}_q, \tag{2}$$

$$c \leftarrow pre.encrypt(r, s, p) : \qquad c = p + s \cdot \mathcal{H}_G(r), \tag{3}$$

$$p \leftarrow pre.decrypt(r, s, c) : \qquad p = c - s \cdot \mathcal{H}_G(r), \tag{4}$$

$$\Delta \leftarrow pre.rekey(s_1, s_2) : \qquad \Delta = s_2 - s_1, \tag{5}$$

$$c_2 \leftarrow pre.reencrypt(r, \Delta, c_1) : \qquad c_2 = c_1 + \Delta \cdot \mathcal{H}_G(r), \tag{6}$$

where the operator $\xleftarrow{\$}$ denotes uniform sampling in a field ($\mathbb{Z}_q$, in this case); $p \in \mathbb{G}$ denotes the plaintext; $c, c_1, c_2 \in \mathbb{G}$ denote ciphertexts; $r \in \{0,1\}^*$ denotes the nonce; $s, s_1, s_2 \in \mathbb{Z}_q$ denote secret

keys; and $\Delta \in \mathbb{Z}_q$ denotes the re-encryption key. A concise (though informal) description of how PRE works is as follows: The scheme builds a pseudo-random function (PRF) from the hash function $\mathcal{H}_G(\cdot)$ and the secret key s and uses it as a stream cipher. That is, the output of the PRF is used as the stream key to encrypt message p by using the point addition operation.

The scheme is usually adopted as an hybrid encryption scheme where a symmetric cipher encrypts the data and the PRE scheme encrypts the symmetric cipher key. To this aim, we add two routines *pre.hencrypt* and *pre.hdecrypt* that implement hybrid encryption and decryption.

In hybrid encryption, a thing must first generate a random group element used as a session key for the symmetric cipher. The common method is to generate a random integer in $\mathbb{Z}_q$ and then to compute scalar multiplication with a base element of the group as in the Diffie-Hellman key exchange protocol. Since scalar multiplication is the most expensive operation, we propose to compute it by providing a random binary string to the hash function $\mathcal{H}_G(\cdot)$. This design choice relies on collision resistance of the hash function, and thus it does not affect the security assumption of the original PRE schemes that relies on the stronger random oracle security model. In the following, we define the hybrid encryption and decryption routines by using a generic CPA-secure encryption scheme *sym* with encryption and decryption routines $c \leftarrow sym.encrypt(k, m)$ and $m \leftarrow sym.decrypt(k, c)$, where k is the secret key, m denotes the plaintext data, and c denotes the ciphertext [31]. The hybrid encryption routine is implemented as follows:

$$
\begin{aligned}
c \leftarrow & pre.hencrypt(r, s, m): \quad && k_b \xleftarrow{\$} \{0,1\}^{\lambda}, \;\; k_g = \mathcal{H}_G(k_b), \;\; k_c = \mathcal{H}_b(k_g), \\
& && ek = pre.encrypt(r, s, k_g), \\
& && em = sym.encrypt(k_c, m), \\
& && c = \langle ek, em \rangle . \qquad (7)
\end{aligned}
$$

The thing generates a random bit string of size λ (the security level) and maps it to an element of the group by using the hash function $\mathcal{H}_G(\cdot)$. This value is encrypted by using the PRE encryption routine. Moreover, it is used to compute the symmetric key for the symmetric scheme *sym* by using a key derivation function, which we denote as $\mathcal{H}_b(\cdot)$. Since a group element has the same entropy required by the security level of the symmetric encryption, is is possible to use a single group element as the input of a standard cryptographic hash function as a secure key derivation function, such as SHA256.

The hybrid decryption routine computes decryption operations of the PRE scheme and of the symmetric ciphers as follows:

$$
\begin{aligned}
m \leftarrow & pre.hdecrypt(r, s, c): \quad && k_g = pre.decrypt(r, s, c.ek), \\
& && k_c = \mathcal{H}_b(k_g), \\
& && m = sym.decrypt(k_c, c.em). \qquad (8)
\end{aligned}
$$

5.2 Implementation Based on Curve25519 and Edward25519

Elliptic curves represent the most efficient mathematical foundation to build cryptographic primitives based on finite fields hardness conjectures. We consider curves expressed by using Montgomery and (twisted) Edward representations, which represent the state-of-the-art for efficient and secure cryptographic protocols and are already deployed in many Internet protocols.

We first give an overview of elliptic curves and we describe the adopted symbols notation. For thorough details, please refer to existing literature and standards (e.g., References [7, 8, 50]). Then we describe the proposed design choices for the symmetric proxy re-encryption scheme and the hybrid variant outlined in Section 5.1.

Overview and Notation. Elliptic curves can be described in Montgomery and twisted Edward forms as:

$$v^2 = u^3 + a \cdot u^2 + u, \qquad u, v \in \mathbb{Z}_p, \tag{9}$$

$$-x^2 + y^2 = 1 + d \cdot x^2 \cdot y^2, \qquad x, y \in \mathbb{Z}_p. \tag{10}$$

It is possible to convert Montgomery coordinates (u, v) to twisted Edward coordinates (x, y) and vice versa by using closed-form equations. Moreover, one can compute more efficiently many types of operations by using alternative coordinates representations. Coordinates in their native form (u, v) and (x, y) are called *affine coordinates*. *Projective coordinates* include three coordinates (U, V, W) and (X, Y, Z) such that $(u = \frac{U}{W}, v = \frac{V}{W})$ and $(x = \frac{X}{Z}, y = \frac{Y}{Z})$. The proposed implementation leverage projective coordinates to avoid expensive field inversion operations.

Standard curves supporting the Edward representation have been defined at two security levels. Curve25519 and its twisted Edward representation Edward25519 ensure 128-bit security, that is, the recommendation until 2030. Curve448 and Edward448 ensure 224-bit security [50] and is suggested for very long term use and critical data. The design choices of this article can apply to both security levels. We only propose performance evaluation for Curve25519 and Edward25519 as they are more appropriate for the considered scenario.

Implementation of $\mathcal{H}_G(\cdot)$. The hash function is defined as $\mathcal{H}_G(\cdot) : \{0, 1\}^* \rightarrow \mathbb{G}$, where the input is a variable-length binary string and the output is an element of the group $\mathbb{G}$, that is, an element of the subgroup within the elliptic curve generated by the generator g with cofactor o. The implementation of $\mathcal{H}_G(\cdot)$ includes three main operations:

- the computation of the hash function $\mathcal{H}_u(\cdot) : \{0, 1\}^* \rightarrow (\mathbb{Z}_p, \mathbb{Z}_p)$ that maps the input to a point of the Montgomery curve (u, v). This function can be implemented by using two different approaches: an iterative algorithm or a special-purpose constant-time function tailored for the considered curve. We detail both approaches below;
- the computation of the point in (twisted) Edward coordinates through closed form equations. This operation is based on known conversion formulas and uses projective coordinates to obtain the point (X, Y, Z);
- the multiplication of the point by the cofactor generated by the considered base point to map it to the valid subgroup. Since the cofactor is a small value, the point addition operation is used instead of a general purpose scalar multiplication. Since the cofactor of Edward25519 is $o = 8$, this takes three point addition operations.

The first approach to compute the hash function $\mathcal{H}_u(\cdot)$ is an iterative algorithm. In certain scenarios, where encryption or decryption endpoints are exposed to adversaries, non-constant time algorithms might enable side-channel timings attacks. Since adversaries considered in this article are semi-honest and thus are not able to operate this kind of attack, we do not exclude this solution. Moreover, in the considered scenario the function can be used in offline pre-computation operations, thus not introducing vulnerabilities even against malicious attackers. The function is as follows. A value $u_1 \in \mathbb{Z}_p$ is generated from a standard hash function $\mathcal{H}_b(\cdot)$ (such as SHA256) and is used to compute the value $v2_1 \in \mathbb{Z}_p$ by using the Montgomery curve equation, that is, $v2_1 = u_1^3 + a \cdot u_1^2 + u_1$. If $v2_1$ is a square in $\mathbb{Z}_p$, then the value $v_1 = \sqrt{v2_1}$ can be computed, and the coordinate (u_1, v_1) is a valid element of the group $\mathbb{G}$. Otherwise, another coordinate u_2 is generated by using the same hash function on the invalid value u_1. The procedure is repeated i times until a couple of values (u_i, v_i) that satisfies the curve equation is found.

The second approach uses a deterministic equation that maps binary string to the coordinate (u, v). For Curve25519, the state of the art is the *elligator squared* algorithm [10], which is also

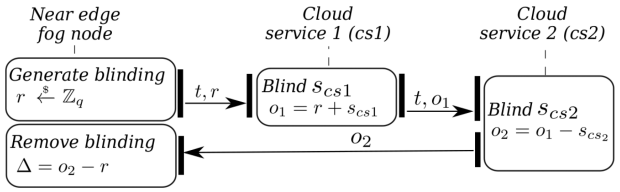

Fig. 4. Secure distributed re-encryption key computation with direct interactions between the cloud services.

currently used for secure key exchange protocols in popular mobile messaging applications [58]. With respect to existing implementations, we implement a variant that outputs the coordinate of the Montgomery curve in projective form, that is, it outputs two values U and W such that $u = \frac{U}{W}$ and the sign for determining coordinate V. Since in the second phase we must transform the Montgomery point to the twisted Edward point, we do not compute V.

6 MULTI-CLOUD ARCHITECTURE

We discuss how the proposed protocol supports communications among things associated with different cloud services. The involved cloud services have to agree explicitly, and they have to use the same identifiers to represent the federated topics.

We define the *inter-service agreement* operation that allows the NFN to obtain a re-encryption key to forward messages between things managed by different cloud providers. A mandatory security requirement is that cloud services must not be able to access each other's secret information. The standard re-encryption key generation routine of re-encryption protocols does not satisfy this requirement, because it is executed by a single trusted party. To solve this issue we propose two protocol variants for semi-honest participants: The first is the simplest and fastest but requires direct communications between the two cloud services; the second causes higher computation overhead but all communications are mediated by the fog node. The aim of the proposed protocols is to compute the re-encryption key generation, that is, to compute a sum in $\mathbb{Z}_q$ (see Section 5.1) in a secure distributed way. Each cloud service provides its addend and does not acquire any information about the other addend nor about the result. The result is only disclosed to the fog node, which, however, does not get any information about any of the inputs. In both protocols, we assume that the two cloud providers have a previous agreement. The protocols allow them to identify the other cloud service involved in the re-encryption key generation. Moreover, we assume that all communications in the protocol are secured by using [D]TLS channels and the endpoints are mutually authenticated.

We describe the first protocol variant by referring to Figure 4, where we identify the involved cloud services as cloud service 1 (CS1) and cloud service 2 (CS2). The fog node generates a blinding value $r \xleftarrow{\$} \mathbb{Z}_q$ and sends it to CS1. CS1 uses it to blind its secret key s_{cs1} by computing the blinded key $o_1 = r + s_{cs1}$. CS2 receives the blinded key o_1 and uses it to compute $o_2 = o_1 - s_{cs2}$. CS2 sends o_2 to the fog node, which can compute the re-encryption key by removing the initial blinding value, as $\Delta = o_2 - r = s_{cs1} - s_{cs2}$.

We describe the second protocol variant by referring to Figure 5. The fog node sends a request to the CS1 endpoint including its public key pk_{gw} and the identifier of CS2. CS1 retrieves the public key of CS2, that is, pk_2, generates a random blinding value $r \xleftarrow{\$} \mathbb{Z}_q$ and computes $er = encrypt_{pk_2}(r)$, where *encrypt* is a standard public-key encryption scheme. CS1 blinds its key as $o_1 = r + s_{cs1}$ and produces a signature of the produced data as $\sigma = sign_{sk_{cs1}}(er, o_1)$. CS1 returns er, o_1, and σ to the fog node. The node cannot decrypt er but forwards it to CS2, which decrypts it,

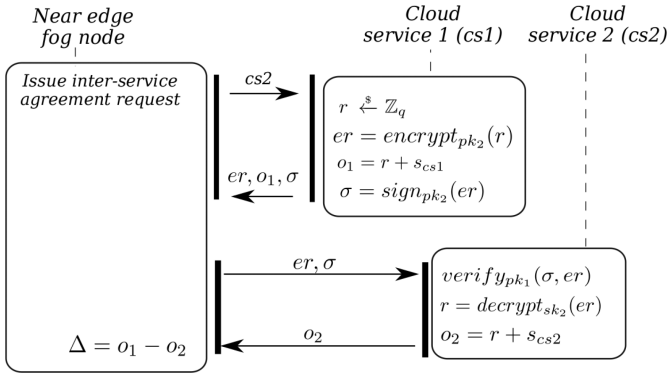

Fig. 5. Secure distributed re-encryption key computation without interactions between the cloud services.

obtains the blinding value r, uses it to blind its own key as $o_2 = r + s_{cs2}$, and returns it to the fog node. The node can now compute the re-encryption key $\Delta = o_1 - o_2 = s_{cs1} - s_{cs2}$.

After the execution of any of the two protocols the broker has a Δ that allows it to re-encrypt messages from the intermediate encrypted representation associated with CS1, which we denote as $c_{t,1}$, to the intermediate encryption associated with CS2, which we denote as $c_{t,2}$, by using the re-encryption routine $c_{t,2} = pre.reencrypt(t \parallel r, \Delta, c_{t,1})$, and vice versa by computing $c_{t,1} = pre.reencrypt(t \parallel r, \Delta, c_{t,2})$ (note that here r denotes the random nonce that is specific to the message, as described in Section 4.5).

7 PERFORMANCE EVALUATION

This section proposes an experimental evaluation that demonstrates the applicability of our proposal to realistic IoT scenarios comprising heterogeneous devices, ranging from low-power MCUs to more powerful ARM platforms. We first describe results of micro-benchmarks that evaluate the time required to execute the cryptographic primitives leveraged by our protocol on different architectures. These results are then used to provide an analytic evaluation of the feasibility of the main operations required to execute our protocols: encryption, re-encryption, and decryption

Performance evaluation is based on our implementation of the proposed cryptographic scheme for proxy re-encryption, that is, a novel library based on the C programming language that reuses some code of the Nettle cryptographic library [35] (v3.3) for integer fields arithmetic and operations over the Edward25519 elliptic curve. Our library is portable and supports both the GNU/Linux Application Binary Interface (ABI), as well as the Embedded ABI of the MCUs and CPUs used for the experimental evaluation. The generation of binaries for different platforms is automatic and only requires to select the proper configuration parameters at compile time. The main differences among platforms relate to the sizes of primitive data types and to platform-specific software dependencies. In particular, on the GNU/Linux ABI we compile the library against the GMP library [23] for big integers arithmetic and leverage the SHA2 implementation from the Nettle library. The library for the Embedded ABI is compiled against *mini-gmp* [23], which implements a subset of the GMP library in C and a stand-alone C implementation of the SHA2 algorithm. This choice minimizes the memory footprint to cope with the low amount of storage and memory available on embedded platforms. We acknowledge that both implementations could be further optimized for faster execution and smaller footprints by introducing architecture-specific assembly code, as commonly proposed by related work on cryptographic protocol implementations tailored for embedded platforms [21, 38]. However we opted for designing a portable library completely written in C that can be easily applied to many different architectures.

Table 1. Timings of $\mathbb{Z}_p$ and Edward25519 Operations

	$\mathbb{Z}_p$ field operations			Edward25519 operations			
	Inversion	Add.	Mul.	Add.	Scalar Mul.	$\mathcal{H}_G(\cdot)$	
						It. (avg.)	Elligator
x86_64	20.2μs	0.025μs	0.013μs	1.08μs	280μs	45.6μs	49.5μs
RPi3 mini-gmp	440μs	0.26μs	2.1μs	22.9μs	6.4ms	1.2ms	1.4ms
RPi3 gmp	213μs	0.013μs	1.01μs	11.85μs	3.45ms	690μs	694μs
RPi1 mini-gmp	1.1ms	4.7μs	8.9μs	64.7μs	16.2ms	3.88ms	3.75ms
RPi1 gmp	610μs	5.7μs	8.1μs	39.6μs	9.53ms	2.01ms	2.07ms
Arduino Due	12.6ms	15μs	51μs	700μs	169ms	44.3ms	39.1ms
Arduino Uno	668ms	606μs	2.5ms	34ms	8,242ms	2146ms	2,202ms

Table 2. Computational and Storage Costs for Protocol Operations

	Thing (Encryption/Decryption)		Fog Node (Re-Encryption)	
	Offline	*Online*	*Offline (stateful)*	*Online*
Asymptotic costs	$O(1)$	$O(1)$	$O(\|d\|)$	$O(\|d\|)$
Cryptographic costs	$2 \cdot \mathcal{H}_G(\cdot) + mul$	add	$\mathcal{H}_G(\cdot) + \|d\| \cdot mul$	$(1 + \|d\|) \cdot add$
Storage [ecc point]	1	0	|d|	0

We measure the execution times of cryptographic primitives on the following five platforms:

- *x86_64 architecture (Core i5-2520M @ 2.50GHz)*: This platform represents a typical personal computer and should provide a reference for the proposed results. It runs code compiled with GCC 7.2.1 on the 64-bit Fedora 26 Workstation OS;
- *RaspberryPi 3 (Model B)*: This platform includes the Broadcom BCM2837 SoC based on a 64-bit ARM Cortex-A53 quad-core CPU operating at 1.2GHz. It runs code compiled with GCC 6.3.0 on the 32-bit Linux Raspbian Stretch OS;
- *RaspberryPi 1 (Model B)*: This platform includes the Broadcom BCM2835 SoC based on an ARMv6 single-core CPU operating at 700MHz. It runs code compiled with GCC 6.3.0 on the 32-bit Linux Raspbian Stretch OS;
- *Arduino Due*: This platform is based on the Atmel SAM3X8E, a 32-bit ARM Cortex-M3 CPU operating at 84MHz, with 512KB of persistent memory and 96KB of RAM memory. It runs code compiled with the GCC 4.8.3 cross-compiler included in the Arduino IDE 1.8.5 for the ARM Embedded ABI;
- *Arduino Uno (Revision 3)*: This platform is based on the 8-bit Atmega328p AVR MCU operating at 16MHz, with 32KB of persistent memory and 2KB of RAM memory. It runs code compiled with the AVR-GCC 4.9.2 cross-compiler included in the Arduino IDE 1.8.5.

We run single-threaded code that does not take advantage of multi-core architectures. We do not consider this as a limitation due to the characteristics of the scenario: low-power devices are always single-threaded; more powerful devices that support multi-threading will likely act as fog nodes and can easily parallelize operations at the data level.

We show the micro-benchmark results in Table 1, the theoretical analysis of computational and storage costs in Table 2, and analytic results of the execution times required for complete protocol operations in Table 3. In all tables, columns represent different types of operations.

Table 3. Estimated Timings for Protocol Operations

	Thing (Encryption/Decryption)		Fog Node (Re-Encryption)	
	Offline	*Online*	*Offline (stateful)*	*Online*
x86_64*	370μs	1μs	45μs + 280μs	2.2μs
RPi3 gmp *	4.8ms	11.9μs	694μs + 3.45ms	23.7μs
RPi1 gmp	11.7ms	39.6μs	2ms + 9.7ms	79.2μs
Arduino Due	247ms	700μs	39ms + 169ms	1.4ms
Arduino Uno	12s	34ms	2s + 8s	68ms

Results of Table 1 refer to the execution times of the primitive operations on the integer field $\mathbb{Z}_p$ and on the Edward25519 elliptic curve, respectively. All Edward25519 operations have been computed on projective coordinates, and no conversion from or to affine coordinates have been included. For completeness, we provide multiple results for RaspberryPi platforms that are produced by compiling the code with and without the optimized GMP library. This can be useful to evaluate the performance of these platforms when used without a Linux OS. Performance of software using the GMP library is generally significantly faster except for field addition operations on the RaspberryPi 1 platform. Moreover, we show results for the two different variants of hash function $\mathcal{H}_G(\cdot)$, based on the iterative approach (denoted as *It.* in the table) and the elligator function (see Section 5.2). The two approaches are comparable, making the elligator function the preferred one, thanks to its resistance to timings attacks.

Experimental results show that the most expensive operation for the RaspberryPi platform, that is, scalar multiplication, takes 9.53ms. We note that generating a new PRE key, including session key generation, requires the computation of two $\mathcal{H}_G(\cdot)$ functions, a scalar multiplication and one addition. The resulting time is still lower than the latency of most wireless communication and comparable to the latency of many communications over cabled networks. Thus, this kind of device should be able to leverage many existing cryptographic protocols without incurring in significant delays. This result is in line with previous research demonstrating that this type of device can support expensive asymmetric cryptographic protocols, such as attribute-based encryption [2], although limited to certain scenarios (see Section 2).

Table 2 shows the theoretical analysis of the proposed protocol. The table distinguishes online and offline operations. Offline operations can be computed only for stateful architectures where a fog node knows the next nonce values used by things. As an example, the things and the near edge fog node use a counter as the nonce of the encryption scheme (see Section 5.1) and share its state. Moreover, both x86_64 and RaspberryPi 3 are multi-core architectures and support execution of four parallel threads, and thus their throughput would be even greater than those already suggested by the table. The first row includes asymptotic computational costs with regard to an increasing number of connected things, which we denote as $|d|$. The second row includes the same costs in terms of cryptographic operations. This information is useful to distinguish the type of computations required by each protocol operation and obtain a realistic estimation of the performance. As an example, although the fog node must compute a linear number of operations both offline and online, most of the actual effort is spent only during the offline phase that requires the most expensive cryptographic primitives, which are $\mathcal{H}_G(\cdot)$ and scalar multiplication. Finally, the third row shows the amount of storage required to maintain the due cryptographic material for offline pre-computation. This cost is expressed in terms of number of elliptic curve points. We note that for the Edward25519 curve, a single point represented in projective coordinates requires 96 bytes.

Results of Table 3 show the estimated timings of protocol operations based on experimental results of Table 1 and the theoretical analysis of Table 2. The proposed results are based on the elligator hash function and refer to a scenario in which the fog node does not introduce additional delays due to saturation. We highlight that a multi-core RaspberryPi can achieve a higher throughput by generating multiple re-encryption keys in parallel, one for each core. We can analytically estimate that a RaspberryPi 3 can leverage its four cores to achieve a throughput of about 825 re-encryption keys per second. These results lead us to the conclusion that any device with computational capabilities similar to those of a RaspberryPi 3 is suitable to implement the role of PRE-enabled publish/subscribe broker in a fog architecture.

Results regarding Arduino platforms are more interesting for the IoT scenarios considered in this article. Scalar multiplication and the hash function $\mathcal{H}_G(\cdot)$ are the two most expensive operations for the Arduino Due platform. Based on these results, we can estimate that the time required for executing an encryption routine of the PRE scheme is about 247ms. Depending on the application scenario, this amount of time might introduce non-negligible latencies and may limit the applicability of the proposed solution. Computations over the Arduino Uno take more than 8s for scalar multiplication and more than 2s for $\mathcal{H}_G(\cdot)$. As a result, executing the encryption routine requires about 12s. To make the scheme usable on this kind of platform, the devices should pre-compute the pseudo-keys and using 96 bytes of memory for each of them. This design choice makes it possible to offload all the expensive operations and let the Arduino Uno only compute point addition operations in real time. By adopting this optimization, an encryption routine can be completed in just 68ms even on the Arduino Uno.

Finally, an interesting result regards field inversion, required to convert projective coordinates to affine coordinates, thus reducing the bandwidth of all protocols that transmit elliptic curve points. Results about field inversion are shown in Table 1. This optimization is especially appealing when IoT devices must directly send data through metered network connections that adopt pay-per-use billing policies. Results show that any platform having computational capabilities similar or higher than an Arduino Due can apply this operation and benefit from bandwidth reduction. However, low-end computational capabilities of the Arduino Uno lead to an execution time of field inversion operations of about 668ms, making it more convenient to send points in projective coordinates in the majority of workloads and networks. In this scenario it is possible to let the fog node perform the conversion operation.

8 CONCLUSIONS

This article proposes architecture and cryptographic protocols to improve security guarantees of fog computing. Our solutions leverage semi-honest fog nodes to guarantee the scalability, resiliency, and end-to-end confidentiality of IoT network communications. The proposal is based on a symmetric proxy re-encryption cryptographic scheme designed to fit the highly constrained devices that communicate in a topic-based publish/subscribe paradigm. This design choice allows a thing to exchange messages with any number of other things in a secure way by only using a single cryptographic key. Experimental results based on a prototype implementation show that our proposal fits even low-end devices based on 8-bit microcontrollers, paving the way for secure and scalable IoT networks at acceptable costs. As a final contribution, we propose an extension of our protocol that allows local and secure message exchange even among things associated to different Cloud services. We achieve this goal by designing two different protocols for secure distributed computation of the proxy re-encryption key. Future work within the same research field includes the design of novel security protocols able to withstand active attacks according to more challenging threat models. Other interesting developments relate to dynamic network scenarios in which things and fog nodes can join and leave without the intervention of a network administrator.

REFERENCES

[1] Amazon Web Services. 2018. AWS IoT. Retrieved from https://aws.amazon.com/iot/.

[2] Moreno Ambrosin, Arman Anzanpour, Mauro Conti, Tooska Dargahi, Sanaz Rahimi Moosavi, Amir M. Rahmani, and Pasi Liljeberg. 2016. On the feasibility of attribute-based encryption on Internet of Things devices. *IEEE Micro* 36, 6 (2016), 25–35.

[3] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. 2006. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.* 9, 1 (2006), 1–30.

[4] Luciano Barreto, Antonio Celesti, Massimo Villari, Maria Fazio, and Antonio Puliafito. 2015. An authentication model for IoT clouds. In *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*.

[5] Luciano Barreto, Antonio Celesti, Massimo Villari, Maria Fazio, and Antonio Puliafito. 2015. Security and IoT cloud federation: Design of authentication schemes. In *Proceedings of the International Internet of Things Summit*. Springer.

[6] Paolo Bellavista and Alessandro Zanni. 2017. Feasibility of fog computing deployment based on docker containerization over RaspberryPi. In *Proceedings of the ACM 18th International Conference on Distributed Computing and Networking*.

[7] Daniel J. Bernstein. 2006. Curve25519: New Diffie-Hellman speed records. In *Proceedings of the IACR International Workshop on Public Key Cryptography*. Springer.

[8] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. 2008. Twisted Edwards curves. In *Proceedings of the IACR International Conference on Advances in Cryptology (Africacrypt'08)*. Springer.

[9] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *J. Cryptogr. Eng.* 2, 2 (2012), 1–13.

[10] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.

[11] Matt Blaze, Gerrit Bleumer, and Martin Strauss. 1998. Divertible protocols and atomic proxy cryptography. *Proceedings of the IACR International Conference on Theory and Applications of Cryptographic Techniques (Eurocrypt98)*.

[12] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. 2013. Key homomorphic PRFs and their applications. In *Proceedings of the IACR International Conference on Advances in Cryptology (CRYPTO'13)*.

[13] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing*.

[14] A. Keranen C. Bormann, M. Ersue. 2014. *RFC7228: Terminology for Constrained-node Networks*. RFC.

[15] Ran Canetti, Oded Goldreich, and Shai Halevi. 2004. The random oracle methodology, revisited. *J. ACM* 51, 4 (2004), 557–594.

[16] Ran Canetti, Shai Halevi, and Jonathan Katz. 2004. Chosen-ciphertext security from identity-based encryption. In *Proceedings of the IACR International Conference on Theory and Applications of Cryptographic Techniques (Eurocrypt'04)*. Springer.

[17] Ran Canetti and Susan Hohenberger. 2007. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th International ACM Conference on Computer and Communications Security*.

[18] Stanley Chow, Philip Eisen, Harold Johnson, and Paul C. Van Oorschot. 2002. White-box cryptography and an AES implementation. In *Proceedings of the International Conference on Selected Areas in Cryptography*. Springer.

[19] OpenFog Consortium. 2018. IEEE approved draft standard for adoption of OpenFog reference architecture for fog computing. *IEEE P1934/D2.0* (Apr. 2018).

[20] Robert H. Deng, Jian Weng, Shengli Liu, and Kefei Chen. 2008. Chosen-ciphertext secure proxy re-encryption without pairings. In *Proceedings of the International Conference on Cryptology and Network Security*. Springer.

[21] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe. 2015. High-speed Curve25519 on 8-bit, 16-bit and 32-bit microcontrollers. *Des. Codes Cryptogr.* 77, 2 (2015).

[22] Luca Ferretti, Michele Colajanni, and Mirco Marchetti. 2014. Distributed, concurrent, and independent access to encrypted cloud databases. *IEEE Trans. Parallel Distrib. Syst.* 25, 2 (2014), 437–446.

[23] GNU Project. 2018. The GNU Multiple Precision Arithmetic Library. Retrieved from https://gmplib.org/.

[24] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. 2004. Universal re-encryption for mixnets. In *Proceedings of the IACR Cryptographers Track at the RSA Conference*. Springer.

[25] Google Cloud Platform. 2018. Google Cloud IoT. Retrieved from https://cloud.google.com/iot/docs/.

[26] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. 2006. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*.

[27] René Hummen, Hossein Shafagh, Shahid Raza, Thiemo Voig, and Klaus Wehrle. 2014. Delegation-based authentication and authorization for the IP-based internet of things. In *Proceedings of the IEEE International Conference on Sensing, Communication, and Networking*.

[28] Michaela Iorga, Larry Feldman, Robert Barton, Michael J. Martin, Nedim S. Goren, and Charif Mahmoudi. 2018. *NIST SP 500-325: Fog Computing Conceptual Model*. NIST SP. DOI: https://doi.org/10.6028/NIST.SP.500-325
[29] D. McGrew J. Salowey, A. Choudhury. 2008. *RFC5246: AES Galois Counter Mode (GCM) Cipher Suites for TLS*. RFC.
[30] B. Kaliski. 2000. *RFC2898: PKCS 5: Password-Based Cryptography Specification Version 2.0*. RFC.
[31] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography*. CRC Press, Boca Raton, FL.
[32] Hugo Krawczyk. 2010. Cryptographic extraction and key derivation: The HKDF scheme. In *Proceedings of the IACR International Conference on Advances in Cryptology (CRYPTO'10)*. Springer.
[33] LoRa Alliance. 2018. LoRaWAN. Retrieved from https://www.lora-alliance.org/.
[34] Wil Michiels. 2010. Opportunities in white-box cryptography. In *Proceedings of the IEEE International Conference on Security and Privacy* (2010).
[35] Niels Moller. 2018. Nettle: A low-level cryptographic library. Retrieved from https://www.lysator.liu.se/~nisse/nettle/.
[36] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. 2014. Reconsidering generic composition. In *Proceedings of the IACR International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT'14)*. Springer.
[37] Moni Naor, Benny Pinkas, and Omer Reingold. 1999. Distributed pseudo-random functions and KDCs. In *Proceedings of the International Conference on Theory and Applications of Cryptographic Techniques*.
[38] Erick Nascimento, Julio López, and Ricardo Dahab. 2015. Efficient and secure elliptic curve cryptography for 8-bit AVR microcontrollers. In *Proceedings of the International Conference on Security, Privacy, and Applied Cryptography Engineering*.
[39] Andrew Nash, William Duane, and Celia Joseph. 2001. *PKI: Implementing and Managing E-security*. McGraw-Hill, New York, NY.
[40] B. Clifford Neuman and Theodore Ts'o. 1994. Kerberos: An authentication service for computer networks. *IEEE Commun. Mag.* 32, 9 (1994), 33–38.
[41] Kim Thuat Nguyen, Nouha Oualha, and Maryline Laurent. 2016. Authenticated key agreement mediated by a proxy re-encryptor for the internet of things. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS'16)*. Springer.
[42] National Institute of Standards and Technology. 2013. *FIPS 186-4: Digital Signature Standard (DSS)*. NIST Pubs. https://doi.org/10.6028/NIST.FIPS.186-4
[43] David Nuñez, Isaac Agudo, and Javier Lopez. 2016. Attacks to a proxy-mediated key agreement protocol based on symmetric encryption. In *Proceedings of the 31st IFIP International Conference on Data and Applications Security and Privacy*.
[44] OASIS. Sep. 2018. *Message Queuing Telemetry Transport (v3.1.1)*. OASIS Standard.
[45] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. 2016. Confidentiality-preserving publish/subscribe: A survey. *Comput. Surv.* 49, 2 (2016), 27.
[46] OpenID. 2018. OpenID: The Internet Identity Layer. Retrieved from https://openid.net/.
[47] Nicholas Pippenger. 1980. On the evaluation of powers and monomials. *SIAM J. Comput.* 9, 2 (1980), 230–250.
[48] Joost Renes, Peter Schwabe, Benjamin Smith, and Lejla Batina. 2016. μKummer: Efficient hyperelliptic signatures and key exchange for microcontrollers. In *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems*.
[49] Rodrigo Roman, Jianying Zhou, and Javier Lopez. 2013. On the features and challenges of security and privacy in distributed internet of things. *Comput. Netw.* 57, 10 (2013), 2266–2279.
[50] I. Liusvaara S. Josefsson. 2017. *RFC8032: Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC.
[51] L. Seitz, F. Palombini, M. Gunnarsson, and G. Selander. Sep. 2018. *OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework*. Internet-draft.
[52] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Sep. 2018. *Authentication and Authorization for Constrained Environments (ACE) Using the OAuth 2.0 Framework (ACE-OAuth)*. Internet-draft.
[53] G. Selander, J. Mattsson, and F. Palombini. Sep. 2018. *Object Security for Constrained RESTful Environments (OSCORE)*. Internet-draft.
[54] C. Sengul, A. Kirby, and P. Fremantle. 2018. *MQTT-TLS Profile of ACE*. Internet-draft.
[55] Jun Shao and Zhenfu Cao. 2009. CCA-secure proxy re-encryption without pairings. In *Proceedings of the International Workshop on Public Key Cryptography*.
[56] Ivan Stojmenovic and Sheng Wen. 2014. The fog computing paradigm: Scenarios and security issues. In *Proceedings of the IEEE Conference on Computer Science and Information Systems*.
[57] Amril Syalim, Takashi Nishide, and Kouichi Sakurai. 2011. Realizing proxy re-encryption in the symmetric world. In *Proceedings of the International Conference on Informatics Engineering and Information Science*. Springer.
[58] Open Whisper Systems. 2017. The XEdDSA and VXEdDSA Signature Schemes. Retrieved from https://signal.org/docs/specifications/xeddsa/.

[59] Lu Tan and Neng Wang. 2010. Future internet: The internet of things. In *Proceedings of the IEEE 3rd International Conference on Advanced Computer Theory and Engineering*.

[60] Luis M. Vaquero and Luis Rodero-Merino. 2014. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Comput. Commun. Rev.* 44, 5 (2014), 27–32.

[61] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan. 2016. Sieve: Cryptographically enforced access control for user data in untrusted clouds. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*.

[62] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward A Lee, and John Kubiatowicz. 2015. The cloud is not enough: Saving IoT from the cloud. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Cloud Coputing*.

[63] Zhi-Kai Zhang, Michael Cheng Yi Cho, and Shiuhpyng Shieh. 2015. Emerging security threats and countermeasures in IoT. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*.

Received December 2017; revised September 2018; accepted October 2018