Threat-modeling-guided Trust-based Task Offloading for Resource-constrained Internet of Things

MATTHEW BRADBURY, ARSHAD JHUMKA, and TIM WATSON, University of Warwick, UK
DENYS FLORES, Escuela Politécnica Nacional, Ecuador
JONATHAN BURTON and MATTHEW BUTLER, Meta Mission Data, UK

There is an increasing demand for Internet of Things (IoT) networks consisting of resource-constrained devices executing increasingly complex applications. Due to these resource constraints, IoT devices will not be able to execute expensive tasks. One solution is to offload expensive tasks to resource-rich edge nodes, which requires a framework that facilitates the selection of suitable edge nodes to perform task offloading. Therefore, in this article, we present a novel *trust-model-driven system architecture*, based on *behavioral evidence*, that is suitable for resource-constrained IoT devices and supports computation offloading. We demonstrate the viability of the proposed architecture with an example deployment of the Beta Reputation System trust model on real hardware to capture node behaviors. The open environment of edge-based IoT networks means that threats against edge nodes can lead to deviation from expected behavior. Hence, we perform a *threat modeling* to identify such threats. The proposed system architecture includes threat handling mechanisms that provide security properties such as confidentiality, authentication, and non-repudiation of messages in required scenarios and operate within the resource constraints. We evaluate the efficacy of the threat handling mechanisms and identify future work for the standards used.

CCS Concepts: • **Computer systems organization** → *Sensor networks*; • **Security and privacy** → **Trust frameworks;**

Additional Key Words and Phrases: Trust, computation offloading, Internet of Things, resource constrained, edge computing, threat modelling

ACM Reference format:
Matthew Bradbury, Arshad Jhumka, Tim Watson, Denys Flores, Jonathan Burton, and Matthew Butler. 2022. Threat-modeling-guided Trust-based Task Offloading for Resource-constrained Internet of Things. *ACM Trans. Sen. Netw.* 18, 2, Article 29 (February 2022), 41 pages.
https://doi.org/10.1145/3510424

This article is an extension of [14].
This work was supported by the PETRAS National Centre of Excellence for IoT Systems Cybersecurity https://petras-iot.org/ [EPSRC Grant EP/S035362/1].
Authors' addresses: M. Bradbury, School of Computing and Communication, Lancaster University, Lancaster, UK, LA1 4YW; email: M.S.Bradbury@lancaster.ac.uk; A. Jhumka, Department of Computer Science, University of Warwick, Coventry, UK, CV4 7AL; email: H.A.Jhumka@warwick.ac.uk; T. Watson, WMG, University of Warwick, International Manufacturing Centre, Coventry, UK, CV4 7AL; email: tw@warwick.ac.uk; D. Flores, Departamento de Informática y Ciencias de la Computación, Escuela Politécnica Nacional, Ladrón de Guevara E11 253, Quito, Ecuador, PO Box 17-01-2759; email: denys.flores@epn.edu.ec; J. Burton and M. Butler, Meta Mission Data, Malvern Hills Science Park, Geraldine Rd., Malvern, UK, WR14 3SZ; emails: {jonathan.burton, matt.butler}@mmd.meta.aero.

1550-4859/2022/02-ART29 $15.00
https://doi.org/10.1145/3510424

1 INTRODUCTION

Internet of Things (IoT) networks—comprising a large number of IoT devices—are being deployed in a variety of contexts including smart farming [29], healthcare [28], and smart cities [43]. IoT devices have typically been deployed as a distributed system that performs sensing of their environment. Recently, there has been interest in these devices performing more complex tasks (including actuation), such as in **Industrial IoT (IIoT)**. An issue is that many of these devices are resource constrained, with limited processing power, data storage, energy storage, and other constraints.

Due to these resource constraints, it is infeasible for IoT devices[1] to perform computationally expensive tasks such as machine learning. To enable the IoT devices to execute such compute-intensive applications, a support network, called an *edge network*, is used that consists of resource-rich (powerful) nodes called *edge nodes*.[2] Thus, these tasks will need to be sent from IoT devices to the edge network in order to be executed, in a process called *computation offloading*. In vehicular and cellular networks, the task offloading problem is referred to as Multi-access Edge Computing [47]. Targets for offloading may be accessed via the internet (e.g., a Cloud service) or via edge nodes that exist within the same local network as the IoT devices. For a large class of applications, offloading to edge nodes is preferred, such as when the network has no access to Cloud services or when latency is important.

To meet request demand and support application diversity, multiple edge nodes should be provisioned in the network. However, the scale and openness of these networks is such that edge nodes can become attack targets, which can cause edge nodes to deviate from their normal operation. A crucial selection problem then arises: which edge node should an IoT node submit a task to? This can be addressed by *evidence-based behavioral trust* [66, 81], where the incidence of how well an edge node has correctly executed tasks in the past is recorded and is used as a predictor of how likely that edge node is to correctly execute future tasks. That behavioral evidence is captured through a trust model, which is solely used to select which edge node(s) will be most suitable to offload a task to. Trust is typically evaluated at the application level, and sufficient storage is needed to record information about each edge node of interest. However, trust models that require a large amount of memory or processing power to compute are not viable for IoT devices. Hence, simpler models, e.g., the Beta Reputation System [42] or **hidden Markov models (HMMs)** [27], can be used instead.

While a suitable trust model is vital, its correct deployment is equally important. Common internet infrastructure and resource-rich MEC clients are typically unsuitable due to the same resource constraints that necessitate task offloading. Therefore, in this article, we propose and describe a *novel* system architecture for trust-based task offloading of IoT tasks onto edge nodes. The architecture focuses on facilitating offloading even under the resource constraints of the IoT devices acting as clients. It is also designed to be generic by supporting arbitrary behavioral trust models to select which edge a task should be offloaded to and multiple applications running on both edge nodes and IoT devices. Our focus is to support decentralized behavioral trust assessment and decentralized edge node selection for task offloading. In order to showcase the usefulness of this system architecture and evaluate its efficiency, we make use of a known trust model to ease understanding and do not propose a novel trust model. However, the development of a trust model that can be efficiently implemented on resource-constrained devices is an area for future work.

Due to the open nature of such systems, the threats against such systems are multiple and varied. We thus perform a threat modelling exercise of the system to identify potential adversary goals,

[1] By IoT devices, we mean devices that are resource constrained that perform sensing and actuating.
[2] By edge nodes, we mean resource-rich nodes at the edge of the network.

threats, attack paths, and possible mitigations. These threats take into account the highly resource-constrained nature of the IoT devices that may take part in this network and the challenges this poses in developing mitigations. A risk analysis of these threats has not been performed, as the threat modeling is conducted without a specific scenario defined (other than for the purpose of task offloading), and due to the lack of information on attacks against the recent standards employed to create this system that would allow for quantitative analysis. This threat modeling also excludes threats to the trust model due to the large amount of work already performed analyzing them [17, 31, 36] and instead focuses on the proposed middleware that facilitates the offloading.

Once threats have been identified, we demonstrate the viability of our proposed architecture by performing a deployment on real hardware and implementing a simple example trust model using the Beta Reputation System [42]. We demonstrate the evolution of trust values of edge nodes over time.

In this article we make the following contributions:

(1) We propose a system architecture for performing trust-based task offloading for IoT devices.
(2) We perform threat modeling on this architecture to identify potential attacks it may face upon deployment and then identify appropriate mitigations to address these threats.
(3) We profile cryptographic operations on Zolertia RE-Motes and use this to inform the message protection strategy.
(4) We conduct a small deployment of an existing trust model (the **Beta Reputation System (BRS)**) using Zolertia RE-Motes to demonstrate the efficacy of the system. BRS is chosen due to its amenability for efficient implementation for resource-constrained devices.
(5) We use the output of the threat modeling to perform attacks against signature verification and identify future mitigations needed in implementations of the Group OSCORE standard.

The rest of this article is structured as follows: In Section 2 we present related work. Section 3 describes the problem statement and problem setting. In Section 4 the individual components of the system architecture are described, and Section 5 performs the threat modeling against this system. Section 6 describes the experimental setup used to obtain the results presented in Section 7. A discussion of the system is presented in Section 8 before concluding in Section 9.

2 RELATED WORK

There has been much work on standardizing the fundamentals of IoT device infrastructure. Typically, many protocols designed for general internet use-cases are too resource intensive and are designed for high performance and not minimizing costs in terms of RAM, flash, computation, and energy. So, alternative protocols for these resource-constrained systems have been developed, such as uIPv6 [24] for addressing, TSCH [78] for energy-efficient wireless medium access, and RPL [2] for packet routing. Higher-level protocols have been implemented on top of these, such as CoAP [64], which provides similar functionality to HTTP. These protocols have been implemented for IoT operating systems such as RIOT [6] and Contiki-NG [25].

While security protocols such as DTLS can be used to protect UDP traffic, there has been recent effort to standardize security protocols specific to CoAP. OSCORE [61] provides encryption and authentication of messages. Only a subset of headers are protected to facilitate proxying, which changes some CoAP fields. A benefit to OSCORE is that it has low overhead compared to DTLS [34, 35] and there remain unaddressed issues with multiple DTLS implementations [30]; however, OSCORE does not provide forward secrecy, and additional standards (such as EDHOC [62]) would be required to set up security contexts to do so. Other approaches can involve trusted execution environments such as ARM TrustZone [60].

2.1 Trust Assessment

Trust has also been used to solve a variety of problems in IoT applications. Primarily, the security protocols deployed typically need to provide identity trust, where one node can verify the authenticity of a message sent from another node. Trust has been used in a variety of areas, such as routing of messages in wireless sensor networks [19], attack detection (such as intrusion into the network) [8], and localization [32]. In these areas, trust is evaluated based on observations made about the device's behaviors.

Trust and Reputation Systems (TRSs) are important for systems where nodes do not always behave correctly. Nodes may exhibit selective behavior, where services are correctly performed only some of the time [75]. For example, to save energy, a node may choose not to forward all messages that are routed through it. In networks where trust is derived from reputation information, the impact of manipulations of this information needs to be mitigated. An example is when nodes bad-mouth other nodes by lying and indicating they have a low trust value for another node [75].

There has been much work on developing trust models to solve these problems beyond resource-constrained systems [66, 81]. In vehicle and cellular networks the task offloading problem (which we focus on in this article) is referred to as **Multi-access Edge Computing (MEC)** (previously Mobile Edge Computing) [47]. A variety of solutions have been proposed for trust-based offloading in MEC systems, typically involving machine learning approaches [53, 80], **linear programming (LP)** [20], or game theoretic approaches [63]. However, these solutions are typically unsuitable for use in resource-constrained systems. Many machine learning models, though not all, require a large amount of memory or are expensive to compute, and techniques such as LP or game theoretic approaches would require large amounts of data to be sent to a central location to be processed into a schedule, which is costly in terms of energy. This typically means that lightweight approaches are needed that are evaluated on resource-constrained IoT devices.

The seminal example of a lightweight trust model is the BRS [42]. The BRS maintains two counters, which are parameters to the Beta distribution, the number of good events observed (α), and the number of bad events observed (β). A trustor can calculate a trust value about a trustee via the expected value of this distribution (the ratio of good events to the total number of events). These values can be updated with more observations, allowing the belief in the trustee to be refined over time.

The BRS and other models such as those that use HMMs [27] can allow the representation of trust in a small amount of space in RAM. However, for these trust models to be effective in selecting a target for task offloading, they need a suitable system to feed them with observations and then deliver tasks to the chosen node.

2.2 Task Offloading for Resource-constrained IoT Devices

There are very few task offloading middlewares that have been developed with a focus on resource-constrained devices that this work investigates. MEC approaches more commonly target cellular and WiFi networks with low-resource devices typically being smartphones or connected vehicles; it is also uncommon for work to present an actual implementation and experiments performed on real-world devices.

Aura [37] facilitates offloading tasks from resource-rich mobile clients to resource-constrained IoT devices, which is the opposite direction to which this article considers offloading. Tasks also need to be able to be divided into small pieces as they will be executed via MapReduce on the IoT devices. Such an approach may raise the lifetime of a battery-powered resource-rich device (such as a smartphone), but it will also lead to a large decrease in the lifetime of battery-powered resource-constrained IoT devices tasks are offloaded to.

An MEC approach focusing on IIoT was presented in [38]. However, this system focuses on the resources provided by the edge node and does not consider how IoT devices will be suitably implemented to offload tasks within their resource constraints.

A task offloading solution for TinyOS was briefly presented in [67]. However, it lacked details on how another device would be selected for offloading and capabilities to evaluate the success of the offload. In [58] a system was developed for sharing task compute among a group of resource-constrained devices with no access to a resource-rich device to process the task(s). These approaches have not focused on the implementation of a task offloading system and the security threats it may face. Instead, there has been a focus on how to select which device a task should be offloaded to. In this article we present a middleware that is agnostic to the mechanism being used to select an offload target; however, it focuses on providing supporting information to task offloading approaches that utilize behavioral trust.

2.3 Threat Modeling

Much work has been performed investigating attacks against TRSs [36, 75, 77], including attacks such as (1) Bad-mouthing, (2) Good-mouthing, (3) Collusion, (4) Selective Behavior, and (5) Self-promotion. However, the system that *uses* a TRS to perform task offloading is also important to secure. Threat modeling approaches allow the exploration of attacks against a system, its components and their interactions, plus adversaries and their goals. Attack Trees [79] are commonly used to represent an attack scenario as a tree with the root labeled with the goal of the attacker and other nodes labeled as sub-goals or attacks.

Threats are typically classified into one of two frameworks, either the CIA security properties ((1) Confidentiality, (2) Integrity, and (3) Availability) or the STRIDE [65] security threats ((1) Spoofing, (2) Tampering, (3) Repudiation, (4) Information disclosure, (5) Denial of service, and (6) Escalation of privilege). There are many threat modeling methodologies [5, 10] where the usual process is to (1) identify the system and its information assets, (2) identify the adversaries, (3) identify how the adversaries will attack the system and which vulnerabilities they may exploit, (4) calculate a measure of risk (e.g., risk = impact $\times$ likelihood), and, finally, (5) identify mitigations to reduce risk.

Commonly used methodologies include (1) the Cyber Kill Chain [50], which focuses on threats and where controls block an attack in seven steps from reconnaissance to actioning objectives; (2) the **Process for Attack Simulation and Threat Analysis (PASTA)** [73, Chapter 7], which is a risk-centric analysis where risk is used to prioritize which threats mitigations should be developed for; (3) Trike [59], which takes a risk management perspective of a limited classification of threats (denial of service, escalation of privilege, and social responsibility); and (4) OCTAVE [1], which is a risk-based analysis of (a) operational risk, (b) security practices, and (c) the technology involved. The analysis focuses on assets (e.g., information and systems) key to an organization.

2.4 Previous Threat Analyses

There have been a variety of threat analyses of task offloading systems and the specific components that the system this article proposes builds upon. For example, a survey of MEC was performed in [57] that included offloading security considerations. However, as previously mentioned, MEC solutions are not directly applicable to resource-constrained IoT devices due to those resource constraints limiting the approaches to offloading and platforms that can be used.

There has been much work on investigating attacks and countermeasures for **Wireless Sensor Network (WSN)** and IoT systems [3, 9, 18, 71]. This involves attacks at different layers including the physical, communication, and application layers. Attacks at these layers will have requirements, different impacts, and varying ease in which the attacks can be performed. For example,

physical attacks may be difficult to perform (e.g., requiring physical access to the relevant devices) but can lead to large impacts such as via key stealing attacks or adversaries loading a custom firmware.

The majority of these analyses focus on network-level attacks, for example, attacks to routing protocols for packets in the network such as RPL [72]. Many attacks applicable to other wireless systems (e.g., eavesdropping, jamming, replay, and others) also apply to these resource-constrained systems; however, their unique aspects mean that additional attacks can have a large impact. For example, denial of sleep attacks [56] can prevent communication hardware sleeping, leading to exhaustion of energy stores and denial of service. Communication security layers such a DTLS [30] and OSCORE [61, Section D.1] have also been investigated due to systems depending on the protections these layers provide.

3 PROBLEM STATEMENT

In this section, the offloading problem that the system architecture needs to facilitate is defined. The system is modeled as a graph $G = (V, E)$, where

- $V = V_R \cup V_C \cup \{\rho\}$ is the set of nodes in the network, made up of resource-rich edge nodes (V_R), resource-constrained IoT nodes (V_C), and a resource-rich *root* node ρ, and
- $E \subseteq V \times V$ is the set of communication links between nodes in the network.

We assume that an IoT device $i \in V_C$ exists within the network to perform sensing and actuation. They are typically battery powered with limited CPU power, RAM, energy storage, and potentially no stable storage. A representative device is the Zolertia RE-Mote [83], which has a 32 MHz CPU, 32 KiB of RAM, 512 KiB of programmable flash, a 800 mAh battery, and support for an optional SD card. Another is the nRF52840 [51] SoC, which has a 64 MHz CPU, 256 KiB of RAM, and 1 MiB of programmable flash. Communication in these devices is typically performed using IEEE 802.15.4, Bluetooth Low Energy, or LoRaWAN. An edge node $e \in V_R$ is a device with extensive computing capabilities. Edge nodes support the IoT nodes by executing tasks that either are too expensive for the IoT device or require access to data unavailable to the IoT devices. The special root node is equipped with similar resources to edge nodes and performs dedicated tasks for the system.

There is a set of applications $\mathcal{A}$ deployed in the network. Each application $a \in \mathcal{A}$ has two parts: (1) one part that is deployed on at least one IoT device $c \in V_C$ and (2) a second part that is deployed on at least one edge node $v \in V_R$. The set of IoT device applications is denoted by $\mathcal{A}_C$, while the set of applications running on edge nodes is denoted by $\mathcal{A}_R$. We assume a bijection $f_{app} : \mathcal{A}_R \to \mathcal{A}_C$ from the edge node applications to IoT device applications. We assume two functions: (1) $A_C : V_C \to 2^{\mathcal{A}_C}$ that returns the set of applications on an IoT device and (2) $A_R : V_R \to 2^{\mathcal{A}_R}$ that returns the set of applications on an edge node.

Tasks that are generated on an IoT device $c \in V_C$ for application $a \in A_C(c)$ will need to be delivered to an edge node $v \in V_R$ that hosts the corresponding application and then a result returned to the IoT device that submitted the task. The set of edge nodes that can process these tasks of application a is given by

$$V_R^a = \left\{ r \mid r \in V_R \wedge \left(\exists a' \in A_R(r) \, , f_{app}(a') = a \right) \right\}. \tag{1}$$

Definition 3.1 (Task Offloading Problem). Given an IoT device $c \in V_C$ and a task t for application $a \in A_C(c)$ that c needs to offload, select an edge node $r \in V_R^a$ to which the task t should be offloaded to such that (1) the task t will be accepted by r, (2) r returns a result within some finite deadline d, and (3) the result of executing t on r is the correct output.

We assume that application tasks may not always be executed correctly by edge nodes and that there may be failures in any of the three dimensions referred to in Definition 3.1 due to edge nodes choosing to intentionally behave badly (e.g., wanting to prefer some capabilities over others) or due to other failures (e.g., transient network failures) that may cause packets to be lost. The deadline of a task is application specific; some applications may be critical and have an early deadline, whereas non-critical applications will have later (possibly flexible) deadlines.

We propose to use a measure of *behavioral trust* as one approach to solve the Task Offloading problem. The metric captures the likelihood of a node to correctly execute an offloaded task. However, in order to capture this behavioral trust, a system must first provide (1) identity trust and confidentiality, where messages between nodes can be authenticated and protected; (2) mechanisms to facilitate the discovery of edge nodes and their capabilities; and (3) mechanisms to submit tasks, receive responses, and make observations about these actions. Depending on the trust model in use, it may also be necessary to (4) provide stereotype information about nodes to bootstrap trust and (5) facilitate the dissemination of reputation (i.e., indirect trust) information. This article does not provide a solution to the Task Offloading problem but instead presents a system architecture that facilitates the deployment of trust models that can help solve the Task Offloading problem.

4 SYSTEM ARCHITECTURE

In this section we present a high-level description of our architecture to perform trust-based task offloading before describing in detail the individual components. The system relies upon uIPv6 [24] and RPL [2] for message routing, and CoAP [64] for reliable messaging. As CoAP uses UDP, this avoids the RAM cost of including a TCP stack. CBOR [12] is used to encode the contents of CoAP messages. For the security layer, OSCORE [61] provides encryption and authentication of CoAP messages. We plan for the use of Group OSCORE [70] to secure messages that require no encryption but require being digitally signed. As no working implementation is currently available for the draft standard, we have performed an implementation using Contiki-NG [25].

Our implementation uses Contiki-NG; however, the system architecture can be implemented on other IoT OSes that support the required features, such as RIOT [6], Zephyr [68], OpenThread [33], and others. The Contiki-NG operating system uses a coroutine-based cooperative scheduling model [26] instead of a multi-threaded model. The impact of this design is that the multiple applications need to ensure that they behave well to avoid impacting other applications and tasks. For example, they will need to yield often enough to allow other coroutines to execute. Due to Contiki-NG's implementation of RPL, only one border router acting as a root node is supported. On this single root node, a CoAP server (implemented using aiocoap [4]) will be used to provide services to the network. An overview of the system is shown in Figure 1.

We make the following assumptions as part of the development of this system architecture:

(1) As IoT devices have finite lifetimes, they may be retrieved and have batteries swapped, at which point firmware updates may be performed.
(2) The multiple applications running on a single device are assumed to be *mutually trusted* [76], where one application does not intentionally aim to negatively impact another.
(3) IoT devices will generate tasks that they lack the computational ability or knowledge to calculate. These tasks will be submitted to resource-rich devices that will calculate the result. These tasks are independent, so a resource-rich device does not need to receive multiple task submissions before it can begin processing a task.
(4) A measure of behavioral trust in an edge node is evaluated locally on the IoT devices.

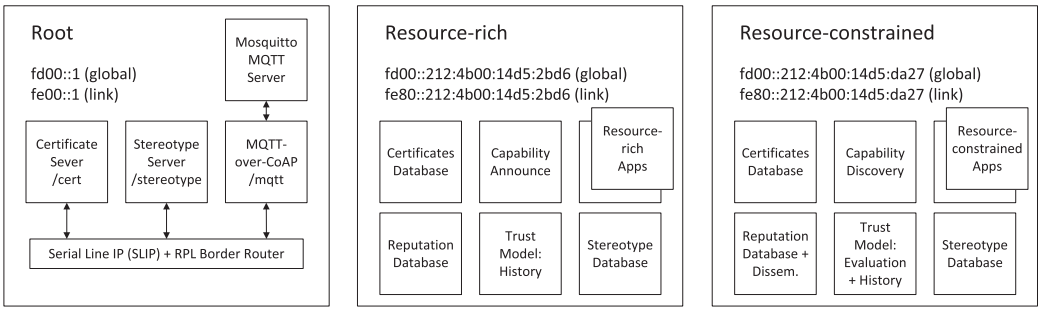


Fig. 1. System functionality across the three device classes with example addresses.

(5) Due to Contiki-NG's RPL limitations, it is not possible to provision redundant root nodes. Therefore, we assume the single root is trustworthy and reliable. The implications of a single root node are discussed in Section 8.1.

These assumptions have been made based on the following reasoning. Assumption (1) is made as there will be limited resources for over-the-air firmware updates [82] after the implementation of our task offloading middleware. A solution to this would be to use more capable devices (such as the nRF52840) instead of Zolertia RE-Motes. Assumptions (2) and (5) are required due to feature limitations of Contiki-NG. Other IoT operating systems such as Tock OS [45] and Zephyr [68] are working toward support for mutually distrusted applications, but support for our desired security layer (OSCORE) is lacking. Assumption (3) is made as we do not expect a task submitted by one IoT device to depend on a different task submitted by another IoT device. If data from other devices is required, then it can be obtained via an additional MQTT service if a centralized approach is acceptable or via a suitable ad hoc data aggregation approach [55] without requiring task dependencies. Finally, Assumption (4) is made because there are additional energy and time costs involved with the transmission of multiple observations, the device evaluating trust would need to be assumed to behave well, and there are additional security threats that would need to be considered with a centralized evaluation of trust [36].

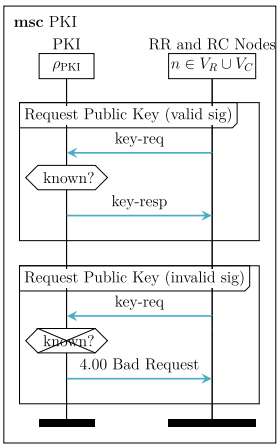
4.1 Public Key Infrastructure

This system is primarily focused on providing evidence of actions taken in a behavioral trust model. The trust model is then used to select an appropriate edge node to offload a task to via an evidence-based evaluation of the edge's past behavior. However, in order to provide a foundation for assessment of behavioral-based trust, it is necessary that nodes in the network have trust in the identities of other devices in the network.

Each IoT device is pre-deployed with a root certificate, their own certificate, and their secret key. This implementation uses the NIST P-256 **elliptic curve (EC)** (also known as secp256r1) for ECC keys because it provides a good level of security and the keys and signatures both take up a small amount of space (64 B) compared to keys required for RSA at comparable bits of security [44]. However, a downside is that ECC operations are time consuming to compute; therefore, we aim to minimize the use of ECC operations where appropriate. Due to the large size of X.509 certificates, we use a CBOR-encoded certificate similar to XIOT [39], whose contents are shown in Figure 2. While the certificates support including the time at which they are valid, not all systems may be capable of checking the validity. This is because there may be no time synchronization protocol in use that allows IoT devices to align their local clock with a global clock. So in this system, certificates can be purged from the root node once IoT devices are expected to have run out of battery.

```
Certificate = [
    tbscertificate  : TBSCertificate,
    signature       : bytes .size 64
]
TBSCertificate = [
    serial_number   : uint,
    issuer          : bytes .size 8,
    validity        : [notBefore: uint, notAfter: uint],
    subject         : bytes .size 8,
    stereotype_tags : StereotypeTags,
    public_key      : bytes .size 64
]
StereotypeTags = [
    device_class    : uint
]
```

(a) CDDL definition of lightweight certificate



(b) PKI Protocol

Fig. 2. Specification of lightweight certificate and Certificate Request Protocol.

To minimize the number of ECC operations that IoT devices perform, a shared secret is provided to an OSCORE context, so for the majority of operations AES-CCM is used to encrypt and provide authentication of messages. In order for a node n_1 to create an OSCORE context with another node n_2, **elliptic curve Diffie-Hellman (ECDH)** is first used to generate a shared secret using n_2's public key and n_1's secret key. Therefore, ECC operations are only required when (1) deriving the OSCORE context, (2) digitally signing/verifying specific messages, and (3) verifying certificates.

At network deployment not all edge nodes may be known; this means there is a need for IoT devices to be able to request keys for unknown nodes from a key server on the trusted root node. The protocol for this is shown in Figure 2(b).

4.2 Resource-rich Capability Discovery

IoT devices need to discover edge nodes and their capabilities (i.e., what applications they are running). Discovery of these capabilities aligns with a publish-subscribe model where IoT devices subscribe to announcements of edge nodes publishing their capabilities. MQTT [7] is a pub-sub protocol designed for IoT devices; however, it has a number of downsides when integrating with this system. Primarily, MQTT uses TCP to provide reliability, which means that there would be an additional RAM cost by including a TCP library. The use of MQTT would also mean that the security mechanisms protecting CoAP messages could not be applied to MQTT messages. To mitigate this overhead on the IoT devices, we instead implement MQTT-over-CoAP, where an application on the root node translates CoAP messages into MQTT messages and vice versa. The MQTT-over-CoAP translator application communicates with a Mosquitto [46] server that provides the MQTT functionality.

There are four phases to resource-rich capability discovery, which are shown in Figure 3. The first requires IoT devices to `subscribe` to four topics: (1) `edge/+/announce`, (2) `edge/+/unannounce`, (3) `edge/+/capability/+/add`, and (4) `edge/+/capability/+/remove`. The first wildcard entry (represented by `+`) is the edge node's EUI-64 in hexadecimal, and the second wildcard entry is the name of the capability.

The `announce` topic is used for edge nodes to announce themselves to others in the network. Their lightweight certificate is included in the message, so receivers do not need to request it. The

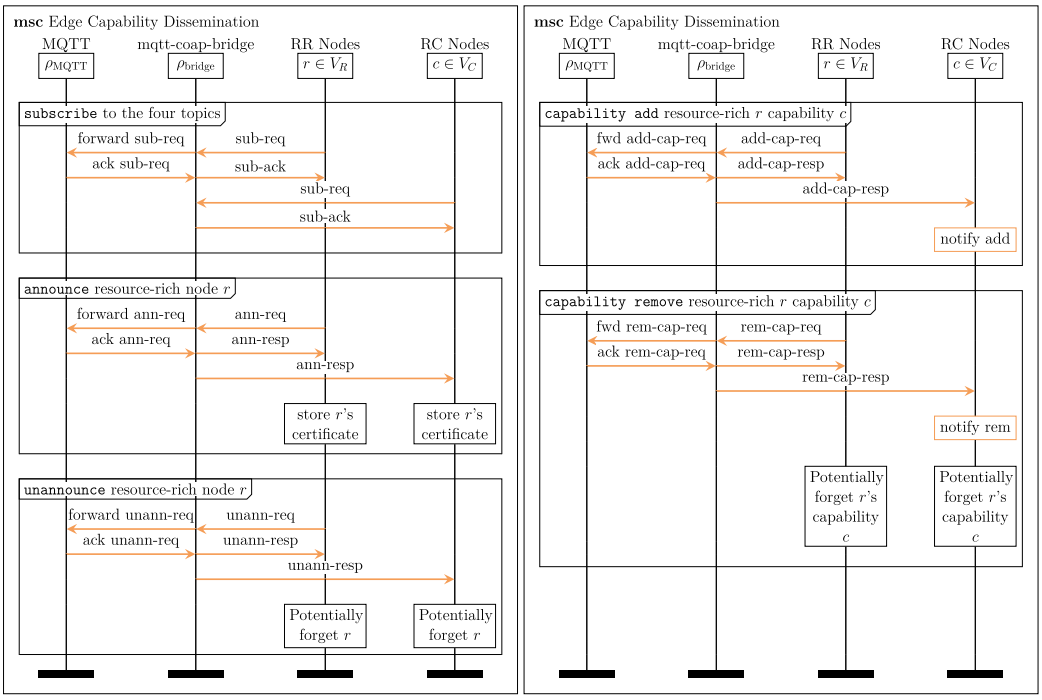


Fig. 3. Resource-rich Capability Discovery Protocol.

receivers will validate the certificate upon reception. The `capability add` topic is used for edge nodes to inform subscribers that a specific capability is being provided by that node.

The `unannounce` and `capability remove` topics are used by well-behaved edge nodes to inform subscribers that the node or a capability is unavailable, respectively. Malicious nodes may not publish these messages, so IoT devices will need to be able to handle this scenario when encountered.

4.3 Resource-rich Stereotype Request

An issue in trust-based selection is that when the system is starting or a new entrant joins, there is little opportunity for historical data to have been gathered and used to build a trust model. Therefore, in order to facilitate better initial decisions, stereotypes can be provided as a starting point to bootstrap trust models [66]. When an edge node announces itself, the certificate it sends contains a set of tags that provide an abstract description of the node. Once these tags are received, the stereotype for this set of tags is requested from the root node as shown in Figure 4. When choosing which IoT device to submit a task to, the stereotype with the closest set of matching tags may be used in the process of calculating the trust value for that edge node.

4.4 Reputation Dissemination

Trust models may incorporate a measure of *reputation* into their evaluation of the trustworthiness of an edge node. The reputation of a trustee is the beliefs held by other trustors in the system and it is stored in the same format as the trust model held by other trustors. When a trust model incorporates reputation, each of the IoT devices need a mechanism to disseminate their beliefs. It is important to provide non-repudiation for messages containing reputation of trustees so nodes

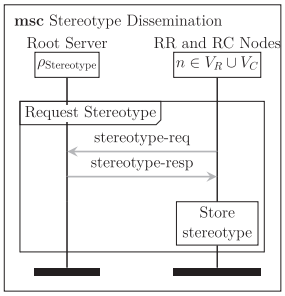


Fig. 4. Resource-rich stereotype request.

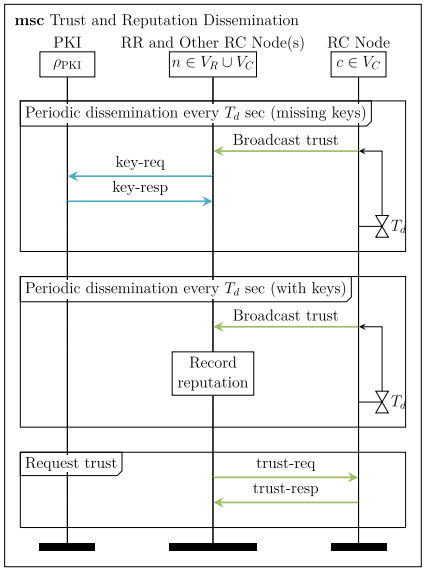


Fig. 5. Trust Dissemination Protocol.

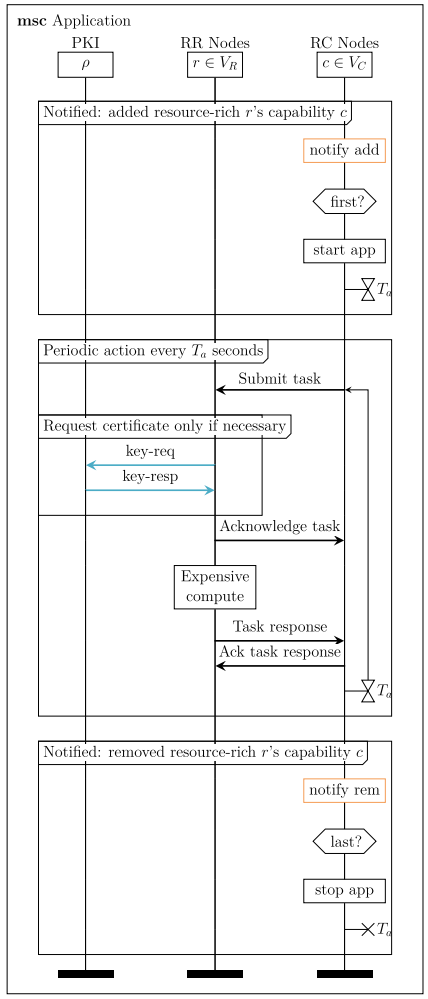


Fig. 6. Application Protocol.

cannot claim they had a different trust value in the past. There is also no need for confidentiality, meaning the messages can be signed and sent unencrypted.

There are multiple options for implementing dissemination of reputation information, such as (1) performing a network-wide multicast, (2) targeting specific nodes, (3) performing a one-hop broadcast, or (4) allowing nodes to request reputation information. In this implementation we focus on (3) and (4), where IoT devices perform a periodic dissemination of trust values and also allow other nodes to request reputation information from arbitrary nodes. Both of these actions are shown in Figure 5.

4.5 Application

The messages that applications on IoT devices send to edge nodes (the task request and, vice versa, the task response) will be protected by OSCORE via encryption and authentication of the message. While some applications may not require confidentiality, a generic layer needs to encrypt the messages in order to facilitate applications that do require it. An example of the application protocol

is shown in Figure 6 for an application that periodically sends a task every T seconds. When a `capability add` for this application is received, the application is started if it is not running.

For the first periodic action, an edge node may be selected that lacks the IoT device's certificate. The edge node will not acknowledge the message and instead request the certificate, either when the IoT device retries whether the certificate has been retrieved or when the IoT device will eventually time out. Subsequent actions will not need to repeat this as the edge nodes will cache the certificate. In the case of failure, applications may choose to resubmit a task to an alternate edge node. This is left up to the application as it will require buffering the task to facilitate retrying it.

5 THREAT MODELING

In this section, a threat modeling is performed on the proposed system architecture for trust-based task offloading from resource-constrained to resource-rich devices. This involves identifying (1) the threat actors who may attack the system, (2) the components of the system and how they interact, and (3) how individual attacks may be combined so an adversary can reach one of their goals.

To aid in performing threat modeling of this system, a **data flow diagram (DFD)** of the system is shown in Figure 7 using De Marco notation [22]. The DFD has been customized to highlight additional features of interest. Data flows within an entity are shown with solid lines, and data flows from one entity to another are shown as dashed lines. These dashed lines are at greater risk of impact from an adversary. The importance of the OSCORE security contexts for communication is highlighted by including a data flow from the context database to the boundary of the entity. Two entities cannot interact without both having appropriate security contexts (except for reputation dissemination, which only requires the receiver to have an appropriate context for the sender).

5.1 Threat Actors

In this section we describe three threat actors that the system may be subject to: (1) a malicious resource-rich edge, (2) a malicious resource-constrained IoT device, and (3) an external threat actor. The edge and IoT devices may be malicious (e.g., when owned by competing organizations) or become malicious (e.g., via compromise). We describe the adversaries along the following six dimensions. Only the Tactics of TTPs are included in the description of adversary goals, as Techniques and Procedures will depend on the attacks being performed.

Goals: What is the threat actor trying to achieve?
Motivations: Why is the threat actor trying to achieve their goals?
Resources and Capabilities: What equipment/tools/finance/personnel/etc. does the threat actor have? What actions are they able to perform with these resources?
Knowledge: What information does the adversary have? How does this impact the way in which they perform attacks?
Presence: Where is the adversary (or their equipment) located? What is their mobility?
Tactics, Techniques, and Procedures (TTPs): These "describe the behaviour of an attacker. Tactics are high-level descriptions of behaviour, techniques are detailed descriptions of behaviour in the context of a tactic, and procedures are even lower-level, highly detailed descriptions in the context of a technique" [41]. For example, is the attack likely to be slow and silent, requiring persistence on the system, or quick and noisy? How will attacks likely be conducted?

5.1.1 Malicious Resource-rich Edge Node. The goals and motivations for a malicious resource-rich edge node are shown in Table 1. These devices could be malicious for a variety of reasons. This may include compromise by an external adversary, or the edge node may act in a manner

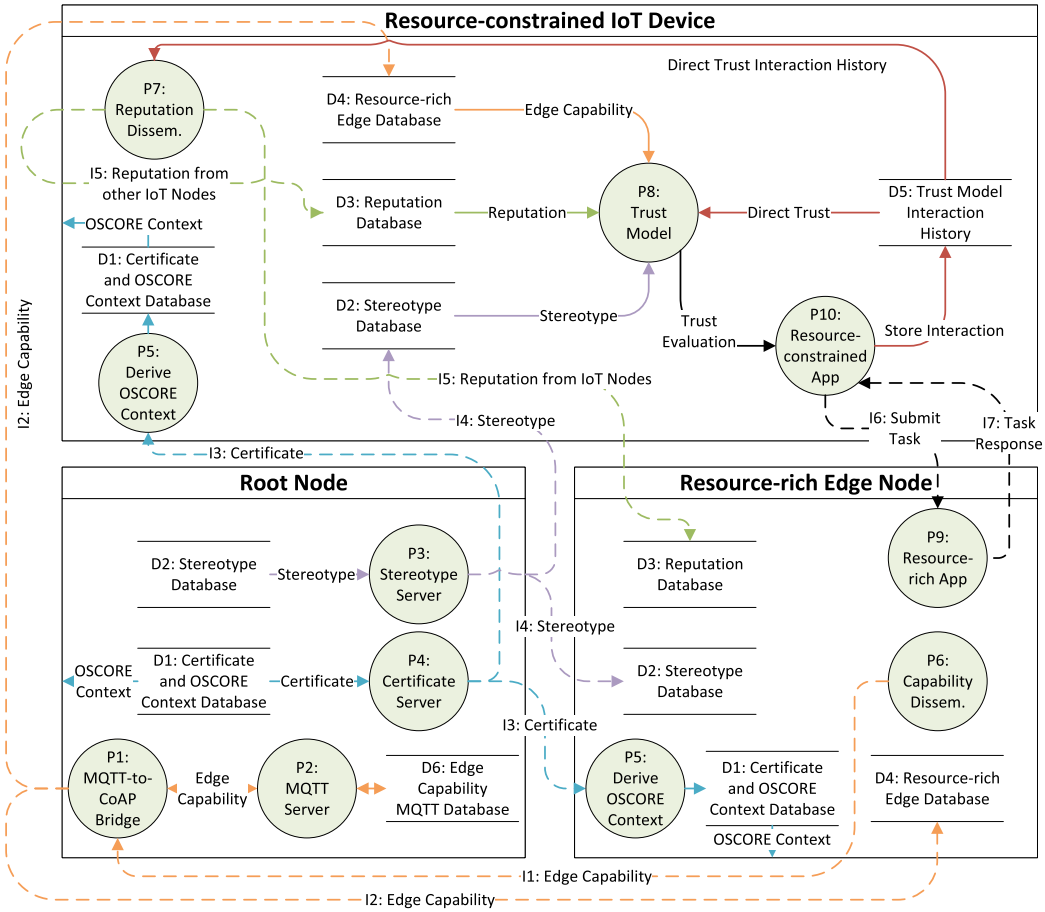


Fig. 7. Data flow diagram of system, where dashed lines represent data flows that cross entity boundaries and solid lines are internal to the entity.

that could be considered malicious to manipulate the actions of the network to its organization's advantage. Edge nodes will have a relatively large amount of resources available to them and their operators will have internal knowledge of the many operations of the network that are required to allow the edge node to communicate with the network. They will exist at the boundary between the ad hoc IoT network and other networks (such as the internet or private clouds). These devices will have access to the network for its entire lifetime.

5.1.2 Malicious Resource-constrained IoT Node. There are many similarities between this adversary and the malicious resource-rich edge, as both adversaries will have knowledge of network operation. A malicious resource-constrained node may also have been deployed with malicious behavior, or may become compromised. A difference is that node compromise is more challenging than for resource-rich devices due to the intentional low flexibility of the OSs that run on these devices. While firmware updates can be performed **over the air (OTA)**, there is a resource cost and the update mechanisms also pose security risks [82]. IoT devices will typically have poor physical security, so attacks where the firmware is dumped, modified, and flashed are possible [74]. Goals

Table 1. Malicious Resource-rich Edge Node

Goal	Motivations	Requirements	Tactics
G1: Alter trust values of other edge nodes	• Reduce/increase task requests	• Able to emulate an IoT node (reputation dissemination) • Able to impact edge node's ability to execute tasks	• Slow manipulation over a long time • Rapid manipulation over a short time
G2: Delegate tasks to other edges	• Be rewarded for other edges' work	• Able to emulate an IoT node	• Manipulation over a long period of time
G3: Prevent IoT nodes submitting tasks or obtaining result	• Prevent certain nodes in the system from submitting tasks or obtaining result	• Spoof other edge nodes • Manipulate message routing	• Quick action for periodic tasks • Long-term manipulation for aperiodic tasks
G5: Avoid correctly executing tasks	• Save resources in order to prioritize preferred nodes/applications	• Be sufficiently trusted for tasks to be sent to this edge	• Utilize knowledge of beliefs to perform incorrectly at key moments

Table 2. Malicious Resource-constrained IoT Node

Goal	Motivations	Requirements	Tactics
G6: Alter trust values of edge nodes	• Reduce/increase task requests • Increase capacity for our tasks at preferred edge nodes	• Peer-provided reputation is used in trust model	• Slow manipulation over a long time
G7: DoS edge nodes	• Prevent certain edge nodes from performing valid tasks	• Generate more tasks than an edge can process	• Rapid task generation • Potentially obvious attack

and motivations for this adversary are shown in Table 2. These devices exist within the ad hoc network and have a potentially shorter lifespan than the network if battery powered.

5.1.3 External Adversary. An external adversary will have limited knowledge toward the functioning of the network but may be able to gain an understanding through passive eavesdropping of network operations. They can exist both within and outside the ad hoc network, and can be mobile when within the ad hoc network. This adversary will typically not be limited by resource constraints (unless deploying their own similar network to perform attacks) and will be able to outlive devices in the network. Goals and motivations for this adversary are shown in Table 3.

5.2 Threat Summary

This section will discuss threats to interactions, data stores, and processes in general, and the full details of the threat modeling are shown in Appendix A. It is important to understand the details of a threat as to simply identify that an interaction is vulnerable to a denial-of-service attack is not sufficient to analyze mitigations to that threat.

As much of this system involves wireless communication, it means that typical threats to this kind of communication exist to interactions between different entities in the system. A spoofing threat, where an adversary impersonates another entity in a crafted packet, may have an intended impact to interact maliciously with the system as if the adversary was a genuine entity. In a

Table 3. External Adversary

Goal	Motivations	Requirements	Tactics
G8: Prevent function of network	• Reduce/increase task requests • Increase capacity for our tasks at preferred edge nodes	• Local presence to jam IoT communications • Remote presence to DoS edges or Root	• Obvious attack
G9: Eavesdrop to obtain confidential information	• Obtain sensitive/valuable information	• Local presence to overhear IoT communications	• Stealthy and long term

tampering threat, an adversary performs a **man-in-the-middle (MITM)** attack or replays a previous packet. This could involve modifying information such as capability dissemination or manipulating reputation via old recorded messages. For some communications, there is a threat of repudiation, where an entity may claim they sent different information than they actually did in the past. This threat is typically associated with reputation dissemination, as a resource-constrained device should not be able to lie about reputation information it had previously sent. Due to the wireless communication, an adversary with the necessary equipment and presence can eavesdrop on communications to learn sensitive information. Also due to the wireless communication, it is relatively easy for an adversary to cause a DoS by jamming the medium to prevent communication.

There are additional threats that arise due to the resource constraints of the IoT devices. One key example that is common to data stores on resource-constrained devices is the threat posed by buffer exhaustion. As these buffers are relatively small (due to the low amount of memory available) for a large network, it will not be possible to store information on all resource-rich and resource-constrained devices. So, there arises multiple threats: an adversary will attempt to fill a buffer with low-quality information, meaning a resource-constrained device is unable to perform an offload or is unable to make a good choice about whom to offload to. Alternatively, if an eviction strategy is in use, then an adversary can take advantage of it to attempt to replace high-quality information with low-quality information.

Finally, DoS threats to resource-rich systems via a large number of requests have additional impacts on resource-constrained devices due to the cost to process these requests (which will be investigated in Section 7.2). A high computation cost to perform cryptographic operations means that fewer messages are needed to obtain a DoS and that an impact on one aspect of the system can potentially impact others (which will be investigated in Section 7.5).

5.3 Additional General Threats

There exist a variety of additional general threats to this system that are not specific to one interaction, data store, or process. One of the key general threats is that weaknesses are discovered in the implementation of the system or in the cryptographic algorithms used after deployment. These potential vulnerabilities could lead to a wide variety of attacks being performed. In resource-rich systems it is expected that software updates will be deployed to resolve these issues; however, in resource-constrained systems an OTA update mechanism is not always included. This is due to the Flash and RAM cost of supporting OTA updates and also the communication cost of performing the update. Draft standards such as SUIT [48] present a serialization format for software updates in resource-constrained devices, with a prototype implementation available for RIOT.[3] Providing

[3]https://riot-os.org/api/group__sys__suit.html.

OTA updates presents a trade-off in terms of both the increased attack surface and the overhead to support updating, which may reduce resources available to applications or trust models.

As OSCORE is used to protect CoAP messages, it means that limitations of OSCORE apply to this system. The main threats are that CoAP headers are not confidential, as OSCORE only protects the integrity of a subset of headers and does not encrypt them. This has the potential to reveal context information to an adversary about operations being performed in the system.

We have proposed pre-deploying a public/private key pair to IoT devices lasting their lifetime. This simplifies key management and reduces the cost of managing and exchanging keys. A downside is that using ECDH to derive a shared secret once (i.e., for the lifetime of the devices) does not provide forward secrecy and no key revocation has been implemented. If required, then future standards (such as EDHOC [62]) that facilitate ECDHE will be necessary to set up OSCORE contexts.

5.4 Mitigations and Notes

This section describes the mitigations and notes about specific threats. Depending on the required characteristics of alternative applications, assumptions that have been made may not provide suitable protection. For example, it has been assumed that reputation should be public; however, some deployments may wish to keep reputation private. Alternatively, these notes may describe mitigations that are outside of the scope of the middleware that performs task offloading, such as jamming mitigations in low-powered wireless networks.

5.4.1 OSCORE (O). For many of the identified threats OSCORE is an appropriate mitigation, as "OSCORE is intended to protect against eavesdropping, spoofing, insertion, modification, deletion, replay, and man-in-the middle attacks" [61, Section D.1]. However, OSCORE is not intended to protect against all traffic analysis attacks. For example, the packet length, timing of packet generation, and various other CoAP attributes (e.g., token, block options, and CoAP headers) may reveal important context information to an attacker [61, Section D].

5.4.2 Group OSCORE (GO). For threats that require non-repudiation, Group OSCORE can provide this security property due its use of digital signatures. However, due to implementation limitations (that are elaborated on in Section 8.4), messages that require Group OSCORE only have their payload signed in this implementation. This means CoAP headers are not protected.

5.4.3 Certificates Protected with a Digital Signature (CS). As certificates are signed by the root node, they can be authenticated. The root's private key would need to be disclosed for an adversary to create new identities, i.e., new certificates with valid signatures.

5.4.4 Edge Redundancy (ER). It is expected that multiple edge nodes are provisioned for each capability in the system in order to provide redundancy for the services required by the applications on the resource-constrained devices.

5.4.5 Root Is a Trusted Third Party (RTTP). The Root node is expected to behave as a Trusted Third Party, in terms of its behavior in responding to requests for certificates and stereotypes, plus correctly recording capability subscriptions and forwarding publications.

5.4.6 OSCORE and Repudiation (O-R). OSCORE uses a HMAC to provide guarantees about the authenticity and integrity of a message. This does not provide repudiation as either party could have generated the HMAC. This is relevant to a third party who may observe a dispute (i.e., an intentional act to manipulate the reputation of an agent). However, this system does not use witnesses to provide information about observed events in order to build reputation. If it became

important to provide repudiation, then Group OSCORE could be used instead; however, the additional cost of signing and verifying additional messages would need to be taken into account.

5.4.7 Trust Model Repudiation (TMR). The trust model may make decisions based on repudiated information. Currently, protection is only included for reputation and stereotypes but not edge capabilities, as the trust model will be able to ascertain the accuracy of this information.

5.4.8 OSCORE Implementation (IMPL-O). The implementation of any security layer is expected to be well tested. However, due to the early state of the implementation, there is the potential for vulnerabilities to exist in the OSCORE layer. Alternatively, OSCORE may expose latent vulnerabilities in other layers of the OS (such as CoAP) that previously could not be exploited. One approach to mitigate this risk is supporting OTA firmware updates. However, this will likely (1) include an additional RAM and Flash cost, which may reduce space available to trust models, and (2) provide an additional attack vector.

5.4.9 Eviction Strategies (ES). Buffers with limited sizes storing information on trust, reputation, stereotypes, and credentials may need to evict items to make room for new items. There is preliminary work on eviction strategies [13] and attacks against eviction strategies [40] under the context of a trust model; however, further work is needed in this area.

5.4.10 RPL Implementation (IMPL-RPL). Contiki-NG's implementation of RPL used for message routing in the ad hoc network only allows there to be a single root node in the RPL tree.

5.4.11 Jamming (J). A large amount of work has been performed on mitigating jamming in these kinds of networks [49]. This includes at the MAC level, where, for example, **time-slotted channel hopping (TSCH)** manages the impact of noise on packet delivery unreliability by hopping between different frequencies. Some variants of TSCH [69] have been proposed that mitigate selective jamming; however, there are methods for an adversary to reverse-engineer the TSCH schedule in order to selectively jam [21].

5.4.12 Reputation Is Public (RP). Reputation information is intentionally broadcast in the clear with a digital signature. This provides non-repudiation and reduces the need to re-transmit reputation encrypted for multiple different targets.

5.4.13 OSCORE Network Analysis (O-NA). The OSCORE threat model does not attempt to protect against all types of network analysis. Headers are sent in the clear and other context information (e.g., timing) may reveal information about the operations being performed. The techniques required to provide protection will depend on the context threats that need to be protected against.

5.4.14 Verify Digital Signature (VDS). Verifying digital signatures is expensive, and due to the resource constraints, very few can be verified per second. Appropriate techniques need to be implemented to mitigate an adversary causing a DoS by sending many messages to verify.

5.4.15 DoS Mitigation (D). Due to the limited amount of memory to process received packets over the network, it will be possible for an adversary to cause packets to be dropped under a relatively high network rate. Techniques can be applied to filter packets believed to only be from a DoS attack [18]. For specific actions, such as requesting a response (e.g., submitting a task), these too can be filtered if believed to be a DoS attack; however, it will be important to consider the impact of false positives on genuine requests.

5.4.16 Stereotypes (S). Stereotypes are used to bootstrap trust and provide a starting point for how edge resources are expected to behave. If stereotypes are unavailable, trust evaluation will need to start from an initial state that makes no assumptions about behavior. Trust-based task

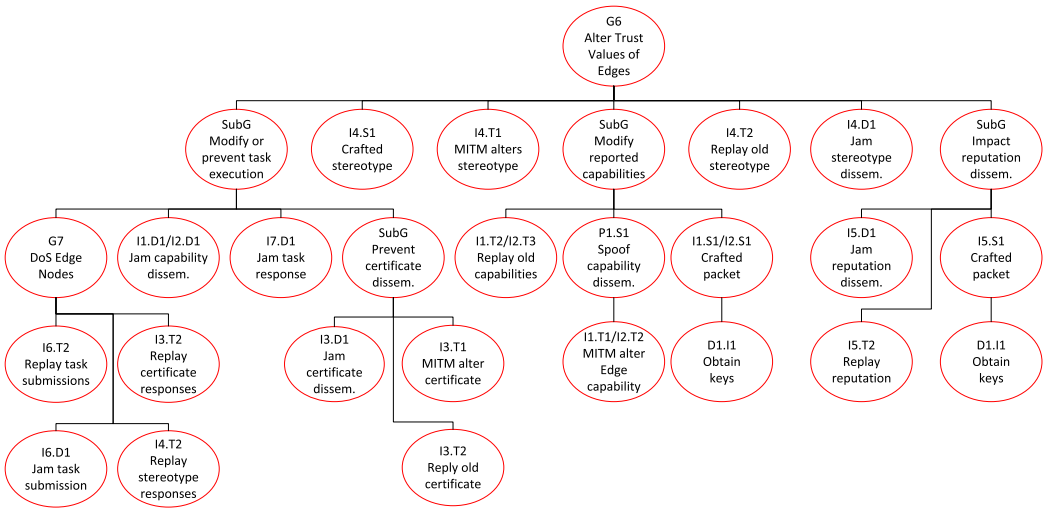


Fig. 8. Attack tree of a malicious resource-constrained IoT node's goal G6 and G7.

offloading can occur; however, sub-optimal offloading decisions may be made. If an adversary learns of stereotypes, then there is the potential for them to be able to better manipulate a trust model. However, this impact will be limited as the trust model gains a record of interactions.

5.4.17 Open Source Software (OSS). The implementation of the offloading infrastructure will be open source, so there are no confidentiality threats to the implementation. Application implementation may be confidential.

5.4.18 Out-of-scope (OOS). Any threat marked with "–" for both the mitigation and note is deemed to be out of the scope of the threat model analysis. Reasons for being out of scope may include that if the adversary has the capability to perform this attack, then other more impactful threats will also be possible. For example, if an adversary can modify, add, or remove contents directly in data stores on IoT devices, then they will also be likely in a position to flash custom firmware.

5.5 Attack Trees

We now examine attack trees for a malicious resource-constrained IoT device's goals G6 and G7 in Figure 8 and a malicious resource-rich edge's goal G3 in Figure 9. To save space we do not include mitigations, as these are included in Appendix A. Each node in the attack tree is either a goal, a threat that a component or interaction is subject to, or a sub-goal (marked with "SubG") to group threats. These attack trees are not intended to be exhaustive, but to include representative threats.

Figure 8 shows an attack tree with the root goal of a resource-constrained IoT node being to alter the trust values of resource-rich edge nodes. In this attack tree there are three main groups of attacks. The first involves modifying or preventing reputation dissemination for trust models that incorporate reputation. The second involves modifying the capabilities that edge nodes report. The rationale is that well-behaved resource-constrained devices will attempt to use edge nodes for unsupported capabilities and then record bad interactions that decrease trust. The third is to manipulate the stereotypes for that edge node. Finally, tasks can be prevented from being submitted, executed, or having results delivered back to the relevant resource-constrained IoT devices.

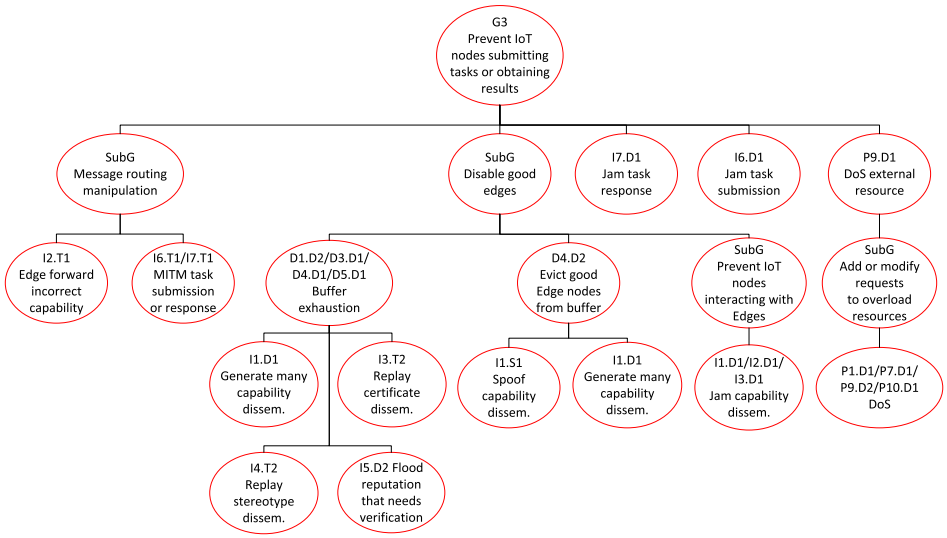


Fig. 9. Attack tree of a malicious resource-rich edge node's goal G3.

Many of these threat are mitigated by the use of either OSCORE or Group OSCORE due to the provision of integrity, authenticity, replay protection, and either confidentially (OSCORE) or non-reputation (Group OSCORE). Various mitigations exist for threats involving communication jamming (e.g., channel hopping) and DoS via the quantity of tasks submitted (e.g., filtering of tasks believed to be an attack); however, both mitigations have their limitations and will not protect against a persistent attack from a high-resource adversary.

Figure 9 shows an attack tree with the root goal of a resource-rich edge node preventing resource-constrained nodes from submitting or obtaining task results. This falls under four main categories: (1) denying service to any external resources an edge may depend upon, (2) jamming or intercepting task submission and/or responses, (3) preventing resource-constrained IoT devices from storing records of well-behaved edge nodes, and (4) manipulating the capabilities that well-behaved edges report. Again, many of these threats are protected against via OSCORE and Group OSCORE. However, threats that involve buffer exhaustion need additional mitigations, such as eviction strategies that take into account ways in which they can be attacked [13, 40].

6 EXPERIMENTAL SETUP

Experiments were performed using a deployment of six Zolertia RE-Motes, which have hardware acceleration for SHA2, AES-CCM-16-64-128 (used by OSCORE), and 256 bit ECC. The key benefit is hardware support for ECC operations that take a long time to compute. Contiki-NG's implementation allows the CPU to execute other (potentially time-sensitive) tasks while a message is being signed or verified. Each RE-Mote was attached to a Raspberry Pi, which logged output. Two of the Raspberry Pis acted as edge nodes, performing expensive computation for applications.

The system frequently publishes capabilities (every 2 min), disseminates trust (every 2 min), generates a monitoring task (every 1 min), and generates a routing task (every 2 to 3 min). These rates are higher than typical in order to obtain results in a reasonable time and would need to be adjusted based on the deployment performed.

To avoid memory fragmentation, fixed-sized pools are allocated at compile time. Table 4(a) shows the default maximum number of different types of objects that can be allocated and their RAM cost. These values would need to be adjusted for deployments with different network sizes.

6.1 Example Trust Model

To illustrate the operation of this system we implement an example trust model using the BRS. The trust model is solely intended to obtain a set of edge nodes that are believed to be suitable for offloading a task to, where the task will be successfully executed, and a result will be successfully returned to the IoT device within a deadline. The trust value for each metric m, edge node r, and application a is beta-distributed $\mathcal{T}_m(r,a) \sim \text{Beta}(\alpha,\beta)$ (application specific) or $\mathcal{T}_m(r) \sim \text{Beta}(\alpha,\beta)$ (application agnostic), where α is the number of successful interactions and β is the number of unsuccessful interactions. The expected value of the distribution summarizes the number of successful events that have occurred as shown in Equation (2). For any application-agnostic metric, m: $\mathcal{T}_m(r,a) = \mathcal{T}_m(r)$.

$$E[X] = \frac{\alpha}{\alpha+\beta} \text{where } X \sim \text{Beta}(\alpha,\beta) \tag{2}$$

Each IoT device c maintains three sets of Beta distributions that summarize interactions with edge node r and application a:

- $\mathcal{T}_{\text{sub}}(r)$ — Did r inform c that a task was received and will be executed?
- $\mathcal{T}_{\text{res}}(r)$ — Did r provide a result for a task?
- $\mathcal{T}_{\text{corr}}(r,a)$ — Was the result that r provided for application a *correct*?

Correctness is an application-specific and best-effort attempt to validate if a result for a task conforms to expected aspects of the result. As c will not execute the task and compare results, it must be expected that the correctness evaluation will contain false positives when evaluating malicious responses.

These three Beta distributions are updated when relevant events occur. These events need to be hooked into by the trust model. The three main events are (1) task submission, (2) task response, and (3) response quality. Other events can be defined and hooked into as needed by alternate trust models, such as in challenge-response-based trust models [15].

The overall trust value of an edge node is summarized by a weighted mean over the expected values of these distributions as shown in Equation (3), where (1) $M(a)$ is the set of metrics that relate to application a, and (2) $0 \leq \varphi_{a,m} \leq 1$ is the weight that application a gives metric m. Applications use the weight to specify the relative importance of metrics, with $1 = \sum_{m \in M(a)} \varphi_{a,m}$.

$$\mathcal{T}(r,a) = \sum_{m \in M(a)} \varphi_{a,m} E[\mathcal{T}_m(r,a)] \tag{3}$$

If a stereotype $\mathcal{S}_m(r)$ is available for edge node r and metric m, then the trust model for that metric is adjusted before calculating the summarized trust. As these trust models are initialized as $\text{Beta}(1,1)$, 1 is subtracted from α and β.

$$\mathcal{T}_m'(r) = \begin{bmatrix} \mathcal{T}_m(r).\alpha - 1 + \mathcal{S}_m(r).\alpha \\ \mathcal{T}_m(r).\beta - 1 + \mathcal{S}_m(r).\beta \end{bmatrix} \tag{4}$$

The individual distributions are updated as per Algorithm 1, where $f^{\text{opp}}_{a,m}$ is an application a and metric-specific m function that evaluates the *opinion* IoT device c has about a situation and interaction. The situation details which task was submitted, and the interaction contains information about the last interaction with the edge node. For example, a situation may be "Request route from a to b" and the interaction may be "r timed out returning a response."

This trust model utilizes reputation information disseminated to demonstrate the cost that it incurs. As we are not investigating threats against trust models (such as peers providing false reputation information), we perform a simple combination of the peer-provided reputation information. Due to the potential for reputation to contain partial information, care must be taken in

ALGORITHM 1: Update state based on a situation and interaction

▷ a is an application, s is a situation, i is an interaction
function UPDATE(a, s, i)
  for $m \in M(a)$ **do** ▷ Iterate over each metric
    if RELEVANTINTERACTION(a, s, i, m) **then**
      $o \coloneqq f^{\text{opp}}_{a,m}(s, i)$ ▷ What opinion is held about i happening under s by a for m?
      if o = Successful **then**
        $\mathcal{T}_m(e, a).\alpha \coloneqq \mathcal{T}_m(e, a).\alpha + 1$ ▷ Increment *good* event counter
      else
        $\mathcal{T}_m(e, a).\beta \coloneqq \mathcal{T}_m(e, a).\beta + 1$ ▷ Increment *bad* event counter

ALGORITHM 2: Select which edge node to offload to

▷ a is an application, $V_R^{a\prime}$ are the known edge nodes that support a
trust_band $\coloneqq 0.25$
function CHOOSEEDGE(a)
  max_trust $\coloneqq \max_{r \in V_R^{a\prime}} \mathcal{TR}(r, a)$
  choices $\coloneqq \{\, r \mid r \in V_R^{a\prime} \wedge \mathcal{TR}(r, a) \geq \text{max\_trust} - \text{trust\_band} \,\}$
  return RANDOMCHOOSE(choices) ▷ Randomly choose one of the options

summarizing it. Equation (5) produces the reputation provided by n for resource-rich node n and application a. This is summarized in Equations (6) and (7), where peers that provide reputation on no relevant metrics are excluded. Finally, the impact peer-provided reputation has on the overall result is weighted by a factor ψ that is in the range $[0, 1]$ in Equation (8).

$$\mathcal{R}(r, a, n) = \frac{\sum_{m \in M(a)} \varphi_{a,m} \begin{cases} E\left[\mathcal{R}^n_m(r, a)\right] & \text{if } \mathcal{R}^n_m(r, a) \neq \perp \\ 0 & \text{otherwise} \end{cases}}{\sum_{m \in M(a)} \varphi_{a,m} \begin{cases} 1 & \text{if } \mathcal{R}^n_m(r, a) \neq \perp \\ 0 & \text{otherwise} \end{cases}} \tag{5}$$

$$\mathcal{R}_s(r, a) = \{\, \mathcal{R}(r, a, n) \mid n \in V \wedge (\exists m \in M(a), \mathcal{R}^n_m(r, a) \neq \perp) \,\} \tag{6}$$

$$\mathcal{R}(r, a) = \frac{1}{|\mathcal{R}_s(r, a)|} \sum \mathcal{R}_s(r, a) \tag{7}$$

$$\mathcal{TR}(r, a) = (1 - \psi)\mathcal{T}(r, a) + \psi\mathcal{R}(r, a) \tag{8}$$

6.2 Example Edge Node Selection

To choose which edge node to submit a task to, that edge node must support the application that originated the task and also have a sufficiently high trust value. A subset of candidate edge nodes that support the application $V_R^{a\prime} \subseteq V_R^a$ may be known. This could be due to a lack of space in memory or IoT nodes not processing or being unable to process the relevant announcement from an edge node. For this example model we implemented a banded approach, where a sufficiently high trust value is any trust value within some distance from the maximum trust value (set to 0.25 in this work). The chosen edge node is selected randomly from the edge nodes that meet these criteria as shown in Algorithm 2.

Table 4. Buffer Size Configuration and Overall Flash and RAM Cost

(a) Constants and RAM Cost

Name	#	Entry (B)	Total (B)
Certificates	12	288	3 456
Stereotypes	5	24	120
Edges	4	52	208
Edge Caps.	12	32	384
Peers	8	32	256
Peer Edges	32	32	1 024
Peer Edge Caps.	96	16	1 536
Reputation Tx	2	500	1 000
Reputation Rx	2	260	520
Sign Buffer	3	24	72
Verify Buffer	3	24	72

(b) IoT Device Flash and RAM Usage

Category	Flash		RAM	
	(B)	(%)	(B)	(%)
applications/monitoring	1 388	1.2	384	1.3
applications/routing	3 968	3.3	505	1.7
contiki-ng	7 232	6.0	826	2.8
contiki-ng/cc2538	14 572	12.1	2 356	7.9
contiki-ng/coap	8 774	7.3	2 388	8.0
contiki-ng/net	27 080	22.5	8 236	27.8
contiki-ng/oscore	5 652	4.7	1 010	3.4
newlib	26 415	22.0	2 534	8.5
system/common	3 420	2.8	37	0.1
system/crypto	7 022	5.8	5 173	17.4
system/mqtt-over-coap	1 494	1.2	503	1.7
system/trust	13 106	10.9	5 724	19.3
Total Used	120 123	100	29 676	100
Total Available	524 288		32 768	

6.3 Example Applications

To illustrate the operation of this system we implement two example applications: (1) environment monitoring and (2) vehicle routing. The environment monitoring application generated a task every 1 min, which involves sending sensor data to an edge node. The vehicle routing application generated a task every 2 to 3 min containing source and destination coordinates and expected to receive a response containing the route between those points within 2 min. The routing application performs a correctness check of the result by checking that the source and destination are the first and last items respectively in the provided path. The trust model weights for these two applications are $[\,1\ 0\ 0\,]$ for Environment Monitoring and $[\,\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}\,]$ for Routing, where the vector contains $[\,\varphi_{a,sub}\ \varphi_{a,res}\ \varphi_{a,cor}\,]$. ψ for both applications was set to 0.25.

7 RESULTS

This section presents results analyzing (1) the RAM and Flash costs of the implementation; (2) the cost of cryptographic operations, highlighting the trade-offs made; (3) the runtime performance of the system with example applications; and (4) an example attack on the system.

7.1 RAM and Flash Usage

The RAM and flash usage of the implementation (shown in 4) was generated using `nm` on the compiled binary. This only shows the cost of defined symbols such as static variables and functions; it does not include strings. Symbols have been classified into categories to identify where the RAM and flash costs are incurred. The implementation is limited by the RAM of the IoT hardware because dynamic memory allocation is typically avoided with embedded systems, as long-term use can lead to memory fragmentation, which prevents future allocation requests from succeeding. Therefore, fixed-size buffers are chosen at compile time. In our implementation 65% of the RAM utilization comes from the buffers required to implement network access (contiki-ng/net), certificate storage and digital signatures (system/crypto), and the trust model (system/trust).

Table 5. Performance of Cryptographic Operations

Operation	Mean Cost	Units
SHA256	637 ± 11.6	ns B^{-1}
ECC Sign (sepc256r1)	360 ± 0.04	ms
ECC Verify (sepc256r1)	711 ± 0.03	ms
ECDH	344 ± 0.02	ms
AES-CCM-16-64-128 Encrypt	0.94 ± 0.01	µs B^{-1}
AES-CCM-16-64-128 Decrypt	1.01 ± 0.01	µs B^{-1}

7.2 Cryptographic Operations Cost

In this section, the relevant cryptographic operation costs are profiled on the Zolertia RE-Mote to understand the trade-offs of using different message protection approaches. The hardware on which these tests were performed has a clock with 32,768 ticks per second, meaning timers have a resolution of 30.5 µs (3 s.f.). The average costs are shown in Table 5 with 95% confidence intervals.

Results for SHA256 and ECC operations were gathered by generating a random plaintext with a random length from 1 to 1 024 B and then signing and verifying that plaintext. As SHA256 is performed as part of the sign operation, each sign and verify operates on a constant number of bytes, so the results are not shown per byte. Results for AES-CCM encryption and decryption were gathered by generating a random plaintext with a random length from 1 to 1 024 B, a random 35 B of additional authenticated data (the maximum supported by OSCORE), a random 16 B key, and a random 13 B nonce. The plaintext was encrypted and an 8 B authentication tag was generated, which was then decrypted and authenticated.

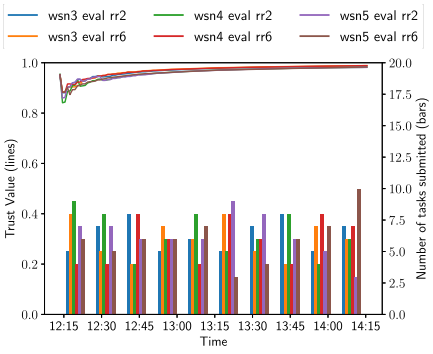
These results highlight the cost difference between AES-CCM operations and the ECC operations on this IoT hardware. Performing an AES-CCM operation on a 100 B message is three orders of magnitude faster than an ECC operation. This performance difference is why ECC operations are only used to derive a shared secret for OSCORE and to disseminate signed reputation information, whereas AES-CCM is used to protect all other messages.

7.3 Task Submission

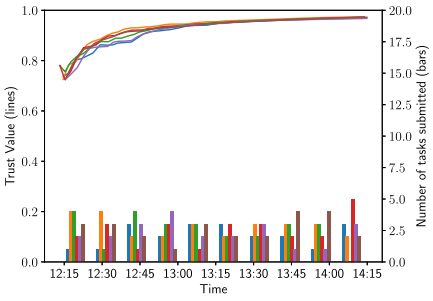
We performed a deployment with three IoT devices (wsn3, wsn4, wsn5) and two edge nodes (rr2 and rr6), where both edge nodes perform the monitoring application correctly. Two different experiments were performed, one where both edge nodes performed correctly for the vehicle routing application and another where rr2 performs correctly and rr6 always performs incorrectly for the vehicle routing application. Incorrect behavior is randomly chosen from (1) not sending a response, (2) sending an invalid response claiming the response is correct, or (3) sending a response indicating a failure. The system was run for long enough for trust values to begin to converge. Results are shown in Figure 10 when both edges are good and Figure 11 when rr2 is good and rr6 is bad. The graphs show (1) the IoT devices evaluate their trust that the edge node will execute the task (lines) and (2) the number of tasks an IoT device submits to an edge node over a time period (bars).

Figure 10(a) and 11(a) show results for the monitoring application, where trust values start high (due to stereotypes) and remain high. There are instances where trust values decrease, which may be due to transient failures such as edge nodes failing to acknowledge a task submission. The tasks submitted by the three IoT devices are distributed across the two edge nodes as no edge node has a sufficiently low trust value for them to be excluded from task submission.

Figure 10(b) and 11(b) show results for the vehicle routing application. The trust values begin at a high value due to stereotype information; however, in Figure 11(b) the trust in the two edge nodes quickly diverges due to the differing behavior. While rr6 has a trust value that is still within

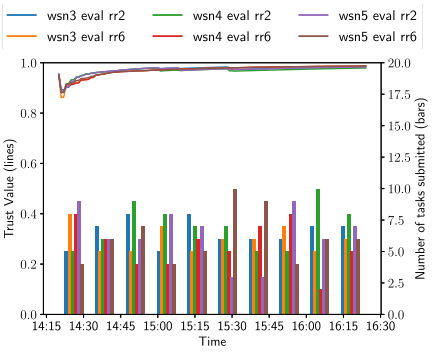


(a) Monitoring

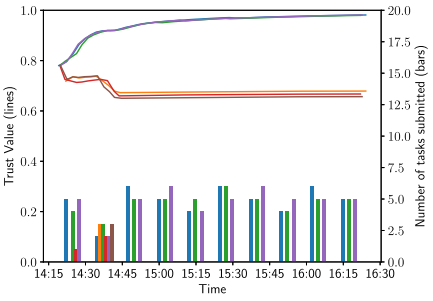


(b) Routing

Fig. 10. Trust values over time and nodes selected to execute tasks for two different applications when both edges are good.



(a) Monitoring



(b) Routing

Fig. 11. Trust values over time and nodes selected to execute tasks for two different applications when rr2 is good and rr6 is bad.

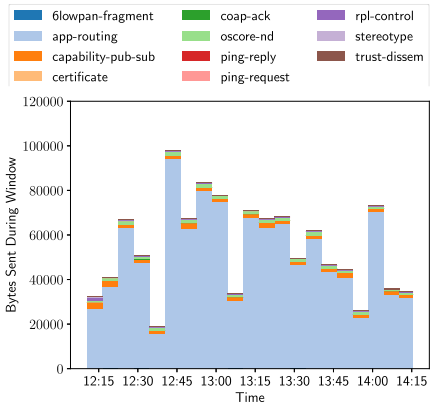
the maximum distance from the highest trust value, it can be chosen to execute tasks (as described in Section 6.1). However, after the trust value becomes sufficiently low, rr6 is excluded from being selected and the IoT devices only send tasks to the well-behaving rr2.

7.4 Message Cost

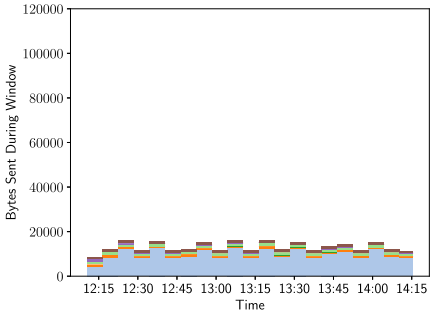
Results showing the number of bytes transmitted and received are shown in Figures 12 and 14 Figures 13 and 15, respectively. Messages have been grouped into 6 min windows. The results for IoT devices wsn4 and wsn5 are omitted as they show a similar pattern to wsn3.

The messages have been categorized where possible. Due to issues with analysis tools, not all OSCORE contexts can be decrypted, so valid messages will appear as "oscore-nd." Not all 6LoWPAN fragments could be reassembled, so are shown as "6lowpan-fragment." Packets marked as "oscore-nd" were for a variety of purposes, including the two applications and potentially other categories where messages could not be decrypted. "trust-dissem" packets are intentionally not protected with OSCORE, as they need to be signed and not encrypted. The implementation currently manually includes a digital signature, which will be the case until Group OSCORE is supported (as will be described in Section 8.4).

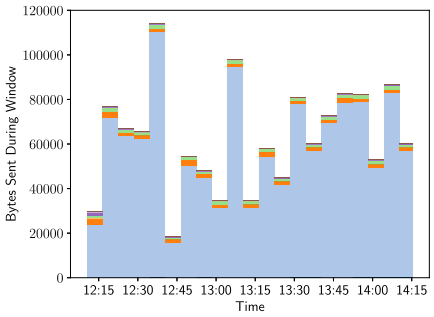
Comparing the two edge nodes rr2 (always good) and rr6 (always bad) shows why an edge node may choose to perform maliciously, as there is a greatly decreased cost in delivering correct application functionality. Edge node 2 has a higher number of messages sent and received than rr6 because by performing correctly, it needs to deliver the result of the application. For the



(a) rr2

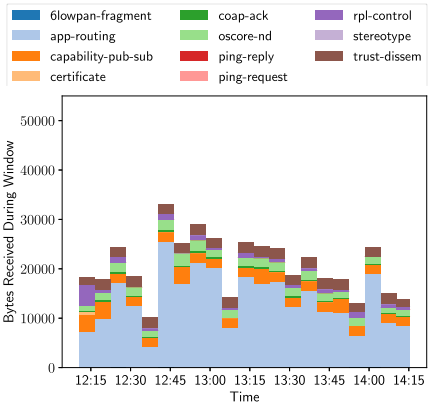


(b) wsn3

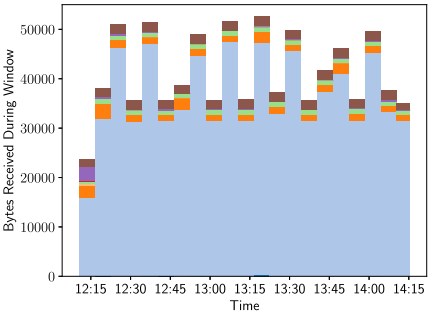


(c) rr6

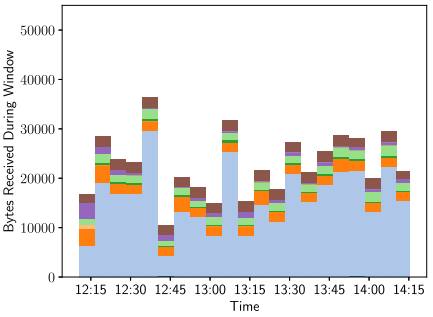
Fig. 12. Length of messages sent over 6 min windows when both edges are good.



(a) rr2



(b) wsn3

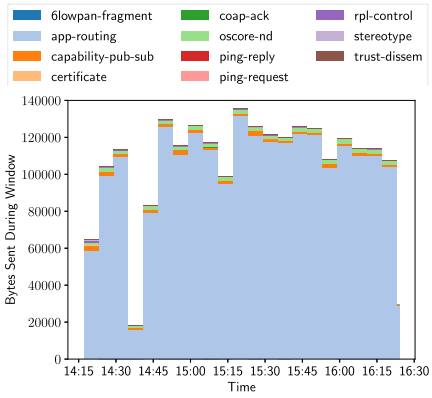


(c) rr6

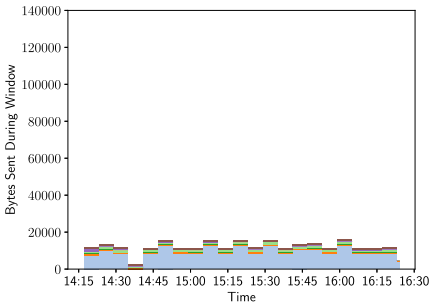
Fig. 13. Length of messages received over 6 min windows when both edges are good.

vehicle routing application task, this means that a result of 9 to 10 KiB needs to be delivered back to the IoT device, which involves sending 53 CoAP messages and receiving the same number of acknowledgments in the best case.

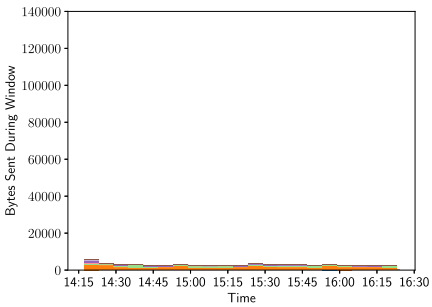
The vehicle routing application has the largest proportion of bytes sent (>74%) and received (>87%) for the three IoT devices. Trust dissemination and subscribing to capabilities are also expensive, costing 6–8% of bytes sent and 4–5% of bytes received. Packets that our analysis tools could not process (marked as “oscore-nd”) took up 7–8% of bytes transmitted and 2–3% of bytes



(a) rr2

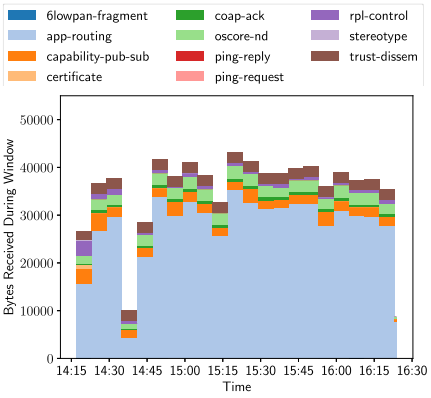


(b) wsn3

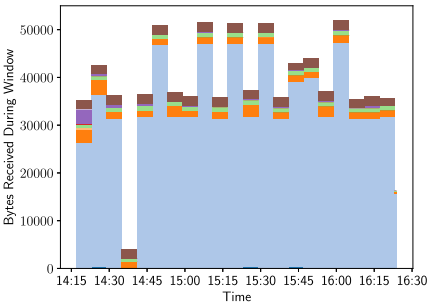


(c) rr6

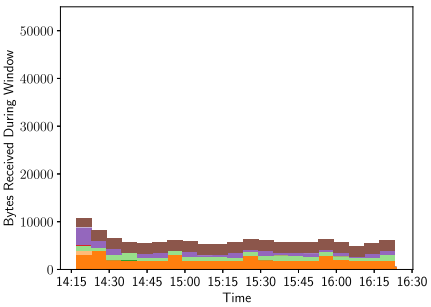
Fig. 14. Length of messages sent over 6 min windows when rr2 is good and rr6 is bad.



(a) rr2



(b) wsn3



(c) rr6

Fig. 15. Length of messages received over 6 min windows when rr2 is good and rr6 is bad.

received. This indicates a worst-case 26% overhead in transmitted bytes and 13% overhead in received bytes to facilitate trust-based task offloading. In reality for the sample applications these overheads will be lower, as some application packets were categorized as "oscore-nd." The overhead will differ depending on the frequency of reputation and capability dissemination, frequency of tasks, and the payload sizes of tasks and their responses. Applications that require less data to be sent and received will lead to their being a greater overhead to disseminate information required for trust-based task offloading. Deployments would need to adjust the rate at which reputation and capability information is disseminated based on application needs.

Table 6. Counts of Different Status of Three Operations When Not Under Attack (2G, GB) and When a DoS Attack Is Performed on Signature Verification (A2/3, A2/2)

(a) Result of Sending Reputation

Node	Result	2G	GB	A2/3	A2/2
wsn3	SUCCESS	62	63	26	25
wsn4	SUCCESS	62	63	26	25
wsn5	SUCCESS	62	63	–	–

(b) Result of Receiving Reputation

N	Source	Result	2G	GB	A2/3	A2/2
wsn3	wsn5 /adv.	SUCCESS	56	61	3 521	3 346
		NO_KEY	1	1	4	17
		OOM	0	0	7 042	6 667
		VER_OOM	0	0	0	9
	wsn4	SUCCESS	60	58	1	2
		NO_KEY	1	1	1	8
		OOM	0	0	14	11
		VER_OOM	0	0	0	0
wsn4	wsn5 /adv.	SUCCESS	47	60	3 525	3 342
		NO_KEY	1	2	5	47
		OOM	0	0	7 030	6 662
		VER_OOM	0	0	0	11
	wsn3	SUCCESS	60	61	0	0
		NO_KEY	1	1	1	15
		OOM	0	0	22	8
		VER_OOM	0	0	0	0
wsn5	wsn4	SUCCESS	61	60	–	–
		NO_KEY	1	1	–	–
	wsn3	SUCCESS	59	62	–	–
		NO_KEY	1	1	–	–

(c) Result of Adding Certificate

N	Source	Result	2G	GB	A2/3	A2/2
wsn3	root	SUCCESS	1	1	1	1
	wsn5 /adv.	SUCCESS	1	1	2	1
		VER_OOM	0	0	0	1
	rr2	SUCCESS	1	1	1	0
		VER_OOM	0	0	0	8
	wsn4	SUCCESS	1	1	1	1
		VER_OOM	0	0	0	7
	rr6	SUCCESS	1	1	1	1
		VER_OOM	0	0	0	5
wsn4	root	SUCCESS	1	1	1	1
	wsn5 /adv.	SUCCESS	1	1	1	1
		VER_OOM	0	0	0	1
	rr2	SUCCESS	1	1	1	1
		VER_OOM	0	0	0	1
	wsn3	SUCCESS	1	1	1	1
		VER_OOM	0	0	0	14
	rr6	SUCCESS	1	1	1	0
		VER_OOM	0	0	0	8
wsn5	root	SUCCESS	1	1	–	–
	rr2	SUCCESS	1	1	–	–
	wsn4	SUCCESS	1	1	–	–
	wsn3	SUCCESS	1	1	–	–
	rr6	SUCCESS	1	1	–	–

7.5 Example Attack: Signature Verification DoS

In this section we investigate the impact of an attacker attempting to cause a denial of service of signature verification in order to prevent resource-constrained devices from being able to process reputation information disseminated from other resource-constrained devices. The adversary on startup signs a packet containing reputation information in a valid format and broadcasts it to its neighbors every 300 ms. The packet is correctly generated each time with correct CoAP headers.

In this experiment we investigate the presence of an adversary instead of wsn5. rr2 and rr6 act as well-behaved resource-rich nodes, and wsn3 and wsn4 are well-behaved resource-constrained nodes. We include results from where there are two good resource-rich nodes (2G) with an experiment duration of 2 hours and 5 minutes and one good and one bad resource-rich node (GB) with an experiment duration of 2 hours and 7 minutes for comparison.

The results in Table 6(a) show that the attack has no impact on devices being able to *send* reputation information. This is because the attack focuses on the buffers used to verify *received* packets, even though both share hardware used to accelerate cryptographic operations.[4] The impact on

[4]As part of the testing of this attack a bug was discovered that did prevent verification. Due to Contiki-NG's use of coroutines there is no preemptive scheduling, so when the signer and verifier coroutines did not yield after com-

receiving reputation information is shown in Table 6(b), where two configurations are investigated under attack. A2/3 is an attack on the system where both resource-rich nodes are good with a duration of 53 minutes, the size of the reputation receive buffer is 2, and the verify buffer is 3 (as shown in Table 4(a)). Here, it can be seen that the two resource-constrained nodes receive a large number of reputation messages to verify from the attacker. However, not all of the messages are successfully verified (`SUCCESS`), some fail as the relevant certificate is missing (`NO_KEY`), and others fail because there is no space in the reputation receive buffer (`OOM`). A high proportion of messages from the attacker are verified (33%) compared to reputation messages from other resource-constrained devices (0–6%), indicating that a DoS attack against this buffer is feasible by an obvious attack.

However, there is an additional impact that can be achieved if the system is poorly configured. For each reputation message that is queued to be verified it consumes space in two buffers, one for receiving reputation and a second for verifying messages. By default (Table 4(a)), the size of the verify buffer is larger than the reputation receive buffer, but if they have the same length (the A2/2 configuration with a duration of 50 minutes), there is an additional impact achievable. Here, an attacker can cause a DoS on both buffers and verifying a message can fail when the verify buffer is full (`VER_OOM`). However, as adding certificates also depends on signature verification, it means that reputation dissemination can potentially prevent or delay this. Delays in verifying and adding certificates ranged from 2 min to 34 min; however, these delays may be higher in systems with lower rates of reputation and capability dissemination. These delays prevent OSCORE contexts from being created between parties and mean that no secure communication can occur between them. This is shown in Table 6(c) where only A2/2 fails to verify certificates as the verify queue is full (`VER_OOM`). For the majority of pairs this leads to an increase in time for a certificate to be verified, but some nodes are prevented from establishing an OSCORE context in these experiments.

This attack highlights vulnerabilities the combination of limited memory and processing power can lead to. It also raises the need for implementations of security protocols such as Group OSCORE (that will make significant use of digital signatures) to build in appropriate mitigations. For example, Group OSCORE APIs should allow CoAP endpoints to disallow Group OSCORE messages when not expected or required. Applications that only need the guarantees provided by OSCORE should be able to indicate that Group OSCORE protected messages will be rejected. Secondly, endpoints that do accept Group OSCORE projected messages will need to include protection. Heavyweight techniques (such as those used to mitigate (D)DoS in cloud environments [11]) will likely be unsuitable, so lightweight techniques [52] will need to be used instead. However, the implementation of these techniques will need to take care that the memory trade-off is appropriate.

7.6 Example Attack: Remove Bad Interactions

For our second example attack, we investigate a resource-rich node who always performs badly and periodically publishes a `capability remove` or an `unannounce` with the intent for resource-constrained devices to remove the low-trust observations from their database. This is because when these messages are received (as shown in Figure 3), a resource-constrained device may remove the information—as the resource-rich device has claimed it is no longer relevant—in order to make space for new edges with capabilities matching the resource-constrained device's.

pleting their action, they could immediately proceed to sign/verify another message instead of yielding to the other coroutine to allow them to do work. This was resolved in https://github.com/MBradbury/iot-trust-task-alloc/commit/c6c1b1cd36101a7155b908325fb48fc136b61995.

In these experiments rr6 always performs the vehicle routing application incorrectly and we use different resetting behaviors to examine how much trust rr6 can regain. We investigate three different scenarios: (1) rr6 publishes `capability remove` and the IoT devices eagerly remove the capability trust information for the vehicle routing application, (2) rr6 publishes unannounce and the IoT devices eagerly remove the trust information on rr6 and all its capabilities, and (3) rr6 publishes unannounce and the IoT devices mark all the trust information for removal if needed (e.g., to make room for a new edge). Due to the low trust value of the rr6, it would typically be excluded quickly as the default parameterization of selecting which edge to offload a task to is to select from an edge with a capability where the trust value is within 0.25 of the highest trust value, so this band has been set to be within 0.5 of the highest trust value in these experiments. Reputation dissemination is also disabled to highlight the changes to the direct trust values.

The graphs in Figures 16 and 17 respectively show the results for the monitoring and vehicle routing applications in these experiments. It can be seen that in Figure 17(a), when `capability remove` is used to reset trust, the trust in rr6's routing capability decreases over time with periodic spikes of higher trust. The spike is due to the data stored by $\mathcal{T}_{\text{corr}}(rr6, routing)$ being reset; however, as $\mathcal{T}_{\text{sub}}(rr6)$ and $\mathcal{T}_{\text{res}}(rr6)$ are not reset, any incorrect results that are stored here persist and lead to the overall continuing decrease in trust.

When unannounce is used to reset trust in Figure 17(b), this gradual decrease is not observed. Instead, there is a periodic restart to the initial trust value, which decreases before the next reset. This is caused by all of $\mathcal{T}_{\text{corr}}(rr6, routing)$, $\mathcal{T}_{\text{sub}}(rr6)$, and $\mathcal{T}_{\text{res}}(rr6)$ being deleted and reset to their initial values. One additional impact that an unannounce has is that it resets trust values for applications that *are* behaving well, i.e., the monitoring application. This leads to the trust in the monitoring application decreasing while the trust in the vehicle routing application rises. Potentially to discourage this attack, the stereotypes for the monitoring application should start lower so the attack causes a greater decrease in the trust in monitoring when an unannounce is published.

In Figure 16(c) and 17(c) a lazy approach is taken to the removal of edges and capabilities after an unannounce. Instead of immediately removing the information, it is marked as inactive. When the space is needed for new edges or capabilities and there is no available memory, it will be freed. These results show that publishing an unannounce has no impact, as the trust history is not removed in this scenario as no new resource-rich edges are discovered. A downside is that lazy removal complicates logic throughout the middleware, as extra checks need to be performed to ensure an edge or capability is active and objects representing edges and capabilities need to be used carefully such that memory is not used after they are freed.

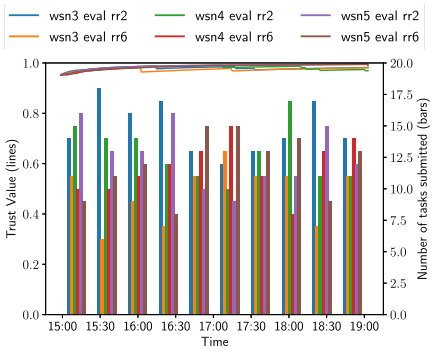
These results show that the architecture in which trust offloading is performed can have an impact on the values calculated by trust models. This will be important to consider for resource-constrained devices, as network deployers will need to make a decision about information management and the ways that adversaries will be able to take advantage of it.

8 DISCUSSION

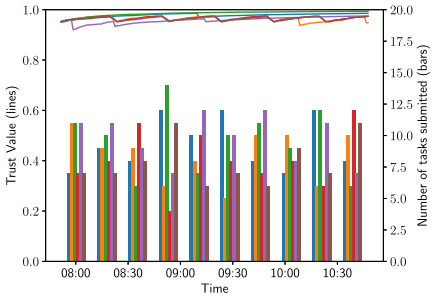
We now discuss considerations with using this architecture, including the impact of design decisions that were made due to limitations in available software libraries or to simplify the implementation.

8.1 Single Root Node

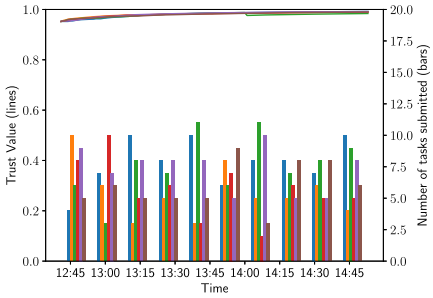
In this system there only exists a single root node, which is problematic from a security viewpoint as this presents a single target that can prevent functioning of the system. The reason for this single root node is due to implementation limitations in Contiki-NG, where their implementation of RPL



(a) Publishing `capability remove` with eager removal

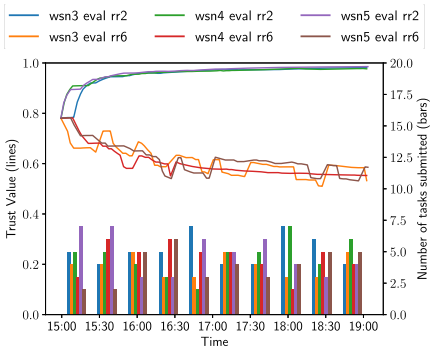


(b) Publishing unannounce with eager removal

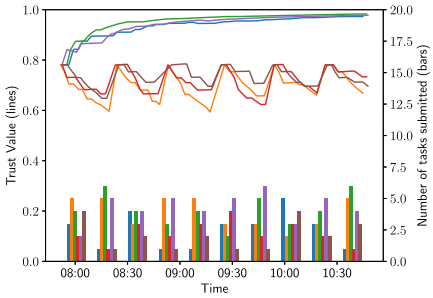


(c) Publishing unannounce with lazy removal

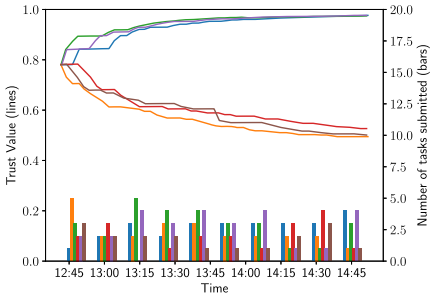
Fig. 16. Impact of a malicious resource-rich edge publishing `capability remove` or unannounce to reset trust values for the monitoring application.



(a) Publishing `capability remove` with eager removal



(b) Publishing unannounce with eager removal



(c) Publishing unannounce with lazy removal

Fig. 17. Impact of a malicious resource-rich edge publishing `capability remove` or unannounce to reset trust values for the vehicle routing application.

(RPL-lite) only supports a single border router. While there is an alternative RPL implementation (RPL-classic), Contiki-NG's documentation recommends using RPL-lite.[5]

There are two approaches to resolving this issue. The first would be to switch to using an RPL implementation that supported multiple border routers. In this case there would need to be consensus among the multiple root nodes when edge nodes publish their capabilities. The second

[5]https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-RPL.

approach would be to distribute the provision of the information that the root nodes provide, including stereotypes, certificates, and pub/sub of edge capabilities. There exist a number of distributed database techniques for these kinds of resource-constrained systems [23]; however, such an approach would need to consider the impact of losing central control over this information. For example, stereotypes would not be able to be updated and there would be no central authority on identities, so additional threats would need to be considered (e.g., Sybil attacks).

8.2 Alternate Operating Systems

With multi-tenant IoT devices there are risks posed by per-application interaction. Operating systems such as Tock [45] (which is written in Rust) provide useful guarantees—such as memory isolation—at the language level. However, Tock currently lacks support for higher-level functionality needed to build complicated systems, such as support for RPL [2] or CoAP [64].

8.3 Use of MQTT Retain Flag

An optimization to reduce the cost of announce and capability messages would be to use the MQTT retain flag. This means when a message is published, that message is saved and delivered to nodes that subscribe in the future. However, a retained message may contain outdated information (e.g., an edge crashed or a capability becomes unavailable). Therefore, in this work we have chosen to be conservative and have edges periodically publish information.

8.4 Implementation Limitations

Due to the use of recently published standards, there are some features of the libraries being depended upon that are not yet implemented. First, the implementation of OSCORE for Contiki-NG does not yet implement RFC 8613 Appendix B.1 [61], which means that when nodes reset, they will be unable to restart communication via OSCORE. Second, the Group OSCORE draft standard [70] does not yet have a fully working implementation for Contiki-NG, so signed and unencrypted trust packets cannot be protected by Group OSCORE. To work around this, the payload is signed; however, this will not protect the CoAP headers that Group OSCORE protects.

8.5 Explanation of Trust Values

In this system, IoT devices evaluate a measure of behavioral trust that an edge node will execute a task and return a correct result within a deadline. Edge nodes will discover the opinions others have for them due to reputation dissemination. However, edge nodes lack context for why these beliefs are held. An edge node will not know which tasks and observations that IoT devices made that led to a poor trust value. Edge nodes should be able to request (or be informed) of the reasoning behind their trust values, facilitating explainable AI. However, this must be balanced with the difficulty of maintaining a large history of observations on the IoT nodes due to limited RAM.

8.6 Limitations of Threat Modeling

In this work we have decomposed the system into individual components and interactions between those components to perform threat modeling. This has allowed the identification of ways in which such a system would be attacked and what mitigations have been provided. However, threat modeling will not identify every threat, and because we have taken a situation-agnostic viewpoint, it will not include those situation-dependant threats. Our threat modeling is also subject to the biases, relative experience, and relative knowledge of the people who have undertaken the threat modeling. To mitigate this we have involved people from both academic and industrial backgrounds to perform the threat modeling.

8.7 Alternate Trust Models

This middleware has been designed to be agnostic to the trust model in use that selects an appropriate edge node to offload a task to. Users of the system must choose an appropriate edge selection approach, which may not even be a trust model. The example trust model used in this work evaluates trust by building beta distributions about three aspects of task offloading. This was chosen in order to provide granularity to evaluate general edge behavior and specific behavior for applications. Alternate trust models that provide either a lower or higher granularity could both be effectively used as part of this middleware. Support is provided for stereotypes and reputation, but they do not need to be used. If reputation is used and there are concerns about threats such as collusion, then suitable collusion-resistant trust models [54] will need to be used. Further work may be needed to develop low-memory variants of these trust models.

8.8 Alternate Edge Selection Models

In this work we focused on IoT devices being able to select which edge node a task should be offloaded to. This was intentional in order to eliminate threats that a centralized task scheduler would pose. However, the system could be configured such that a single edge is advertised to the IoT devices and that edge performs a subsequent offload based on the tasks that are submitted to it. This is not a recommend configuration of this system.

In addition, we proposed a single method via which an edge node would be selected based on a trust model in Section 6.2. Our implementation[6] can be configured with a variety of alternate methods for choosing an edge node a task should be offloaded to based on the information provided by the trust model.

8.9 Scalability

The middleware that we have developed is scalable insofar as the memory constraints of IoT devices allow. In Table 4(a) we specify the constants used, such as allocating space for 12 certificates and space for trust and reputation information for four edge nodes. These constants can be configured for different-size networks. If more IoT devices or edge nodes exist than there is capacity for in memory, then there are two options: (1) more capable IoT devices are required with more RAM or (2) data can be evicted. When evicting information, it is important to ensure that IoT devices can continue to offload tasks. Our future work is investigating appropriate eviction strategies for this information [13] and threats to eviction strategies [40].

9 CONCLUSIONS AND FUTURE WORK

We have presented a system for facilitating trust-based task offloading for multiple applications on IoT devices. Through two case studies and an example instantiation of an existing trust model, we show how a suitable edge node is selected as the destination for offloading. In our example, it took six rounds of task submissions over nearly 30 min for a permanently bad node to be excluded and at worst a 50% overhead in transmitted bytes and 28% overhead in received bytes. The implementation applies recent IoT security standards such as OSCORE and will make use of future standards such as Group OSCORE to provide security guarantees. We have also developed the building blocks to enable the use of more complex trust models that involve the use of disseminated reputation information and stereotypes. For future work, this system architecture that provides the ability to offload tasks using trust models could be modified to support additional information used to build those models. For example, witness statements could be provided for actions that have occurred.

[6] https://github.com/MBradbury/iot-trust-task-alloc/tree/master/wsn/common/trust/choose.

The middleware could also support generating attestation reports of the interactions that led to a specific trust value. However, the resource constraints are likely to limit what is possible to cache and report.

APPENDIX

A THREAT MODELING DETAILS

This appendix contains the full details of the identified threats, showing threats against interactions in Table 7, threats against processing components in Table 8, and threats to data stores in Table 9. Any threat where both the mitigation and note is marked with "–" is deemed out of scope.

A.1 Interaction Threats

Table 7. Threats against Component Interactions

Attack	Impact	Mitig.	Note
I1: Capability Dissemination: Edge to Root			
S_1: Crafted packet	Adversary can manipulate capability state to be different than intended	O	–
T_1: MITM	Modify reported edge capabilities	O	–
T_2: Replay	Replay outdated edge capability reporting	O	–
R_1: Repudiate edge capability	Edge can claim different capability information was provided than it did previously	–	O-R
I_1: Eavesdropping	Adversary can learn of capabilities	O	–
D_1: Jam communications	Edge cannot be selected by IoT to submit tasks	–	J
E_1: Inject malformed capability dissemination	Aim to encounter parsing vulnerabilities	–	IMPL-O
I2: Capability Dissemination: Root to Edge/IoT			
S_1: Crafted packet	Adversary can manipulate capability state to be different than intended	O	–
T_1: Modify data	Root can forward different contents than supplied by edge	–	RTTP
T_2: MITM	Modify reported edge capabilities	O	–
T_3: Replay	Reply outdated edge capability reporting	O	–
R_1: Repudiate edge capability	Root can claim different capability information was provided than it did previously	–	RTTP
I_1: Eavesdropping	Adversary can learn of capabilities	O	–
D_1: Jam communications	Edge cannot be selected by IoT to submit tasks	–	J
E_1: Inject malformed capability dissemination	Aim to encounter parsing vulnerabilities	–	IMPL-O
I3: Certificate Dissemination: Root to Edge/IoT			
S_1: Crafted packet	Adversary can alter certificate for edge/IoT nodes, allowing malicious nodes to impersonate them	O	–
T_1: MITM	Modify certificates disseminated	O	–
T_2: Replay	Reply certificate dissemination	O	–
R_1: Repudiate certificate	Root can claim different certificate information was provided than it did previously	–	RTTP

(Continued)

Table 7. Continued

Attack	Impact	Mitig.	Note
I_1: Eavesdropping	Adversary can learn of identities and certificates	O	O-NA
I_2: Eavesdropping	Adversary can learn of certificate requests and responses	O	O-NA
D_1: Jam communications	Prevent certificate dissemination, edge/IoT potentially cannot communicate with some edge/IoT nodes	–	J
D_2: Inject new certificate requests	Overload edge/IoT by sending certificate responses	–	VDS, D
E_1: Inject malformed certificate requests	Aim to encounter parsing vulnerabilities	–	IMPL-O
I4: Stereotype Dissemination: Root to Edge/IoT			
S_1: Crafted packet	Adversary can manipulate task allocation	O	–
T_1: MITM	Modify stereotypes disseminated	O	–
T_2: Replay	Reply stereotype dissemination	O	–
R_1: Repudiate stereotype	Root can claim different stereotype information was provided than it did previously	–	O-R
I_1: Eavesdropping	Adversary can learn of stereotypes and device classes	O	–
D_1: Jam communications	Quality of task allocation may be low as the system starts	–	J
D_2: Inject new stereotype requests	Overload edge/IoT by sending stereotypes responses	–	D
E_1: Inject malformed stereotype requests	Aim to encounter parsing vulnerabilities	–	IMPL-O
I5: Reputation Dissemination: IoT to Edge/IoT			
S_1: Crafted packet	Adversary can generate fake reputation packet to alter reputation of edge nodes	GO	–
T_1: MITM	Modify reputation disseminated	O	–
T_2: Replay	Adversary can replay reputation dissemination to alter reputation of edge nodes	GO	–
R_1: Repudiate reputation sent	IoT can claim different reputation information was provided than it did previously	GO	–
I_1: Eavesdropping	Adversary can learn of reputation beliefs	–	RP
I_2: Privacy loss	Trust models contained in reputation may reveal sensitive information about interaction history	–	RP
D_1: Jam communications	Quality of task allocation may be low as reputation information will be unavailable	–	J
D_2: Fake signed messages	Verifying digital signatures is expensive, so an adversary may send a large number to reduce the number of valid signed messages that can be processed on an IoT device	–	VDS
E_1: Inject malformed reputation dissemination	Aim to encounter parsing vulnerabilities	–	IMPL-O
I6: Task Submission: IoT to Edge			
S_1: Crafted packet	Edge nodes at risk of DoS with large number of fake tasks	O	D
T_1: MITM	Modify or add to a task submission	O	–
T_2: Replay	Replay task submission	O	–
R_1: Repudiate task submitted	IoT can claim different task was submitted than it did previously	–	O-R

(Continued)

Table 7. Continued

Attack	Impact	Mitig.	Note
I_1: Eavesdropping	Adversary can learn of private information within tasks	O	–
D_1: Jam communications	Task will not reach Edge and will not be executed	–	J
E_1: Inject malformed task submissions	Aim to encounter parsing vulnerabilities	–	IMPL-O
I7: Task Response: Edge to IoT			
S_1: Crafted packet	Adversary can generate malicious response different to valid response	O	–
T_1: MITM	Modify or add to a task response	O	–
T_2: Replay	Replay task responses	O	–
R_1: Repudiate task response	Edge can claim different response was provided than it did previously	–	O-R
I_1: Eavesdropping	Adversary can learn of private information within tasks	O	–
D_1: Jam communications	Task result will not reach IoT node	–	J
E_1: Inject malformed task responses	Aim to encounter parsing vulnerabilities	–	IMPL-O

A.2 Processing Component Threats

Table 8. Threats against Components (Processing)

Attack	Impact	Mitig.	Note
P1: MQTT-to-CoAP Bridge			
S_1: Spoof capability dissemination	IoT nodes receive different capability information than expected	O	–
I_1: Learn configuration	Read system files, configuration, and logs to learn of implementation, settings, and action history	–	–
D_1: DoS	Add or modify requests to overload resources	–	D
P2: MQTT Server / P3: Stereotype Server / P4: Certificate Server / P6: Capability Dissemination			
T_1: Configuration	Modify/add/remove system or configuration files	–	–
D_1: DoS	Add or modify requests to overload resources	–	D
P7: Reputation Dissemination			
D_1: DoS	Add or modify requests to overload resources	–	D
P8: Trust Model			
R_1: Repudiated information	Data sources (D2, D3, D4) may contain repudiated information.	–	TMR
E_1: Task submission	Manipulated data sources may lead to changes in whom tasks are submitted to	–	–
P9: Resource-rich Edge Application			
T_1: Configuration	Modify/add/remove system or configuration files	–	–
D_1: DoS	External resources required for application may become unavailable.	–	ER
D_2: DoS	Add or modify requests to overload resources	–	D
P10: Resource-constrained IoT Application			
T_1: Flash custom firmware (HW)	An adversary can capture the device and flash a custom firmware	–	–
D_1: DoS	Add or modify requests to overload resources	–	F

A.3 Data Store Threats

Table 9. Threats against Components (Data Stores)

Attack	Impact	Mitig.	Note
D1: Certificate Database			
T_1: Database poisoning	Adversary can modify/add/remove certificates for edge/IoT nodes, potentially allowing malicious nodes to impersonate real devices or for unexpected devices to join the network.	CS	–
I_1: Obtain certificate/keys	Adversary can obtain certificates and/or secret keys	–	–
D_1: Remove certificate	Adversary can remove certificates, preventing communication	–	–
D1: Certificate Database (Resource-constrained Specific)			
D_2: Buffer exhaustion	IoT has limited buffers and may be quickly exhausted by invalid malicious messages. Verifying digital signatures is slow, so processing message in these buffers will take a long time	–	ES
E_1: Key stealing (HW)	Adversary can obtain private keys if they can obtain hardware and dump firmware	–	–
D2: Stereotype Database			
T_1: Database poisoning	Adversary can modify/add/remove stereotypes, potentially allowing edges to be provided with an unexpected advantage/disadvantage	–	–
R_1: Repudiate sent stereotypes	Roots can claim to have previously sent different stereotypes than they truly had	CS	RTTP
I_1: Learn stereotypes	Learn stereotypes	–	S
D_1: Remove stereotypes	Adversary can remove stereotypes, preventing trust bootstrapping, potentially leading to worse offloading decisions	–	S
D3: Reputation Dissemination Receive on IoT			
T_1: Modify reputation	Adversary can modify/add/remove reputation information to manipulate who is selected for offloading	–	–
I_1: Learn reputation	Adversary can read confidential reputation information	–	RP
D_1: Buffer exhaustion	IoT has limited buffers and may be quickly exhausted by invalid malicious messages. Verifying digital signatures is slow, so processing message in these buffers will take a long time	–	ES
D4: Resource-rich edge Database			
T_1:	Adversary can modify/add/remove information about resource-rich edge nodes	–	–
R_1: Repudiate edge capability	Edges can claim to have previously published different capability information than they truly had	–	O-R
I_1: Learn edges	Adversary can read known resource-rich edges	–	–
D4: Resource-rich Edge Database (Resource-constrained Specific)			
D_1: Buffer exhaustion	IoT has limited buffers and may be quickly exhausted by invalid malicious messages. Fake or non-useful edges will consume space that may be better allocated to relevant edge nodes	–	ES

(Continued)

Table 9. Continued

Attack	Impact	Mitig.	Note
D_2: Evict good Edge nodes from buffer	Limited space to buffer information means that spoofed Edges can be used to fill the buffer, preventing good Edges from being available	–	ES
D5: Trust Model Interaction History			
T_1: Modify interaction history	Adversary can modify/add/remove historical interactions recorded	–	–
I_1: Learn interaction history	Adversary can read interaction history with different resource-rich nodes	–	–
D_1: Buffer exhaustion	Adversary can cause many interactions to be stored, leading to some needing to be freed	–	ES
D6: Edge Capability MQTT Database			
T_1: Modifiy capabilities	Adversary can modify/add/remove capabilities	–	–
I_1: Learn capabilities	Adversary can learn of resource-rich capabilities	–	–

DATA STATEMENT

The software used to generate these results can be found at https://github.com/MBradbury/iot-trust-task-alloc. The data gathered and presented in this article can be found at [16].

ACKNOWLEDGMENTS

The authors would like to thank Martin Gunnarsson and Krzysztof Mateusz Malarski at RISE for assistance with their work-in-progress OSCORE implementation.

REFERENCES

[1] Christopher J. Alberts, Audrey J. Dorofee, James F. Stevens, and Carol Woody. 2003. *Introduction to the OCTAVE Approach*. Technical Report. Software Engineering Institute, Pittsburgh, PA.

[2] Roger Alexander, Anders Brandt, J. P. Vasseur, Jonathan Hui, Kris Pister, Pascal Thubert, P. Levis, Rene Struik, Richard Kelsey, and Tim Winter. 2012. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550. https://doi.org/10.17487/RFC6550

[3] Saqib Ali, Taiseera Al Balushi, Zia Nadir, and Omar Khadeer Hussain. 2018. *WSN Security Mechanisms for CPS*. Springer International Publishing, 65–87. https://doi.org/10.1007/978-3-319-75880-0_4

[4] Christian Amsüss and Maciej Wasilak. 2013–. aiocoap: Python CoAP Library. http://github.com/chrysn/aiocoap/.

[5] Yassine Ayrour, Amine Raji, and Mahmoud Nassar. 2018. Modelling cyber-attacks: a survey study. *Network Security* 2018, 3 (2018), 13–19. https://doi.org/10.1016/S1353-4858(18)30025-4

[6] Emmanuel Baccelli, Cenk Gündoğan, Oliver Hahm, Peter Kietzmann, Martine S. Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. 2018. RIOT: An open source operating system for low-end embedded devices in the IoT. *IEEE Internet of Things Journal* 5, 6 (Dec. 2018), 4428–4440. https://doi.org/10.1109/JIOT.2018.2815038

[7] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta (Eds.). 2019. *MQTT Version 5.0*. OASIS Standard, Burlington, MA. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[8] Fenye Bao, Ing-Ray Chen, MoonJeong Chang, and Jin-Hee Cho. 2012. Hierarchical trust management for wireless sensor networks and its applications to trust-based routing and intrusion detection. *IEEE Transactions on Network and Service Management* 9, 2 (June 2012), 169–183. https://doi.org/10.1109/TCOMM.2012.031912.110179

[9] Zinaida Benenson, Peter M. Cholewinski, and Felix C. Freiling. 2008. Vulnerabilities and attacks in wireless sensor networks. In *Wireless Sensor Network Security*. Javier Lopez and Jianying Zhou (Eds.). Vol. 1. IOS Press, Amsterdam, The Netherlands, 22–43.

[10] Deborah J. Bodeau, Catherine D. McCollum, and David B. Fox. 2018. *Cyber Threat Modeling: Survey, Assessment, and Representative Framework*. Technical Report. The Homeland Security Systems Engineering and Development Institute, VA. MDPI. https://apps.dtic.mil/sti/pdfs/AD1108051.pdf.

[11] Adrien Bonguet and Martine Bellaiche. 2017. A survey of denial-of-service and distributed denial of service attacks and defenses in cloud computing. *Future Internet* 9, 3 (2017), 19 pages. https://doi.org/10.3390/fi9030043

[12] Carsten Bormann and Paul E. Hoffman. 2013. Concise Binary Object Representation (CBOR). RFC 7049. https://doi.org/10.17487/RFC7049

[13] Matthew Bradbury, Arshad Jhumka, and Tim Watson. [n.d.]. Buffer Management for Trust Computation in Resource-constrained IoT Networks. ([n. d.]). In Submission.

[14] Matthew Bradbury, Arshad Jhumka, and Tim Watson. 2021. Trust assessment in 32 KiB of RAM: Multi-application trust-based task offloading for resource-constrained IoT nodes. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC'21)*. ACM, Virtual Event, 1–10. https://doi.org/10.1145/3412841.3441898

[15] Matthew Bradbury, Arshad Jhumka, and Tim Watson. 2021. Trust trackers for computation offloading in edge-based IoT networks. In *IEEE INFOCOM*, Vancouver, BC, Canada. IEEE, Virtual Event, 1–10. DOI: 10.1109/INFOCOM42981.2021.9488844

[16] Matthew Bradbury, Arshad Jhumka, Tim Watson, Denys Flores, Jonathan Burton, and Matthew Butler. 2021. *Dataset for: Threat Modelling Guided Trust-based Task Offloading in Resource-constrained Internet of Things Systems*. Zenodo. https://doi.org/10.5281/zenodo.4568700

[17] Diego De Siqueira Braga, Marco Niemann, Bernd Hellingrath, and Fernando Buarque De Lima Neto. 2018. Survey on computational trust and reputation models. *ACM Computing Surveys* 51, 5, Article 101 (Nov. 2018), 40 pages. https://doi.org/10.1145/3236008

[18] Ismail Butun, Patrik Österberg, and Houbing Song. 2020. Security of the Internet of Things: Vulnerabilities, attacks, and countermeasures. *IEEE Communications Surveys Tutorials* 22, 1 (2020), 616–644. https://doi.org/10.1109/COMST.2019.2953364

[19] Ing-Ray Chen, Fenye Bao, MoonJeong Chang, and Jin-Hee Cho. 2014. Dynamic trust management for delay tolerant networks and its application to secure routing. *IEEE Transactions on Parallel and Distributed Systems* 25, 5 (2014), 1200–1210. https://doi.org/10.1109/TPDS.2013.116

[20] Min Chen and Yixue Hao. 2018. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications* 36, 3 (2018), 587–597. https://doi.org/10.1109/JSAC.2018.2815360

[21] Xia Cheng, Junyang Shi, and Mo Sha. 2020. Cracking channel hopping sequences and graph routes in industrial TSCH networks. *ACM Transactions on Internet Technologies* 20, 3, Article 23 (July 2020), 28 pages. https://doi.org/10.1145/3372881

[22] Tom DeMarco. 1979. *Structured Analysis and System Specification*. Prentice-Hall, Englewood Cliffs, NJ.

[23] Ousmane Diallo, Joel J. P. C. Rodrigues, Mbaye Sene, and Jaime Lloret. 2015. Distributed database management techniques for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems* 26, 2 (2015), 604–620. https://doi.org/10.1109/TPDS.2013.207

[24] Adam Dunkels, Joakim Eriksson, Niclas Finne, Fredrik Österlind, Nicolas Tsiftes, Julien Abeillé, and Mathilde Durvy. 2012. Low-power IPv6 for the Internet of Things. In *9th International Conference on Networked Sensing (INSS'12)*. IEEE, Antwerp, Belgium, 1–6. https://doi.org/10.1109/INSS.2012.6240537

[25] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. 2004. Contiki - A lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*. IEEE, 455–462. https://doi.org/10.1109/LCN.2004.38

[26] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. 2006. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys'06)*. ACM, New York, NY, 29–42. https://doi.org/10.1145/1182807.1182811

[27] Ehab ElSalamouny, Vladimiro Sassone, and Mogens Nielsen. 2010. HMM-based trust model. In *Formal Aspects in Security and Trust*. Pierpaolo Degano and Joshua D. Guttman (Eds.). Springer, Berlin, 21–35.

[28] Atis Elsts, Xenofon Fafoutis, Przemyslaw Woznowski, Emma Tonkin, George Oikonomou, Robert Piechocki, and Ian Craddock. 2018. Enabling healthcare in smart homes: The SPHERE IoT network infrastructure. *IEEE Communications Magazine* 56, 12 (2018), 164–170. https://doi.org/10.1109/MCOM.2017.1700791

[29] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Kamran Abid, and Muhammad Azhar Naeem. 2019. A survey on the role of IoT in agriculture for the implementation of smart farming. *IEEE Access* 7 (2019), 156237–156271. https://doi.org/10.1109/ACCESS.2019.2949703

[30] Paul Fiterau-Brostean, Bengt Jonsson, Robert Merget, Joeri de Ruiter, Konstantinos Sagonas, and Juraj Somorovsky. 2020. Analysis of DTLS implementations using protocol state fuzzing. In *29th USENIX Security Symposium (USENIX Security'20)*. USENIX Association, Boston, MA, 2523–2540.

[31] David Fraga, Zorana Bankovic, and José Manuel Moya. 2012. A taxonomy of trust and reputation system attacks. In *11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom'12)*. IEEE, 41–50. https://doi.org/10.1109/TrustCom.2012.58

[32] Joaquin Garcia-Alfaro, Michel Barbeau, and Evangelos Kranakis. 2009. Secure localization of nodes in wireless sensor networks with limited number of truth tellers. In *7th Annual Communication Networks and Services Research Conference*. IEEE, 86–93. https://doi.org/10.1109/CNSR.2009.23

[33] Google. 2021. OpenThread. Retrieved January 26, 2021, from https://openthread.io/.
[34] Martin Gunnarsson, Joakim Brorsson, Francesca Palombini, Ludwig Seitz, and Marco Tiloca. 2020. Evaluating the performance of the OSCORE security protocol in constrained IoT environments. *Internet of Things* 13 (2020), 100333. https://doi.org/10.1016/j.iot.2020.100333
[35] Cenk Gündoğan, Christian Amsüss, Thomas C. Schmidt, and Matthias Wählisch. 2020. IoT content object security with OSCORE and NDN: A first experimental comparison. In *IFIP Networking Conference (Networking'20)*. IEEE, 19–27.
[36] Taha D. Güneş, Long Tran-Thanh, and Timothy J. Norman. 2019. Identifying vulnerabilities in trust and reputation systems. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*. International Joint Conferences on Artificial Intelligence Organization, 308–314. https://doi.org/10.24963/ijcai.2019/44
[37] Ragib Hasan, Mahmud Hossain, and Rasib Khan. 2018. Aura: An incentive-driven ad-hoc IoT cloud framework for proximal mobile computation offloading. *Future Generation Computer Systems* 86 (2018), 821–835. https://doi.org/10.1016/j.future.2017.11.024
[38] Xiangwang Hou, Zhiyuan Ren, Kun Yang, Chen Chen, Hailin Zhang, and Yao Xiao. 2019. IIoT-MEC: A novel mobile edge computing framework for 5G-enabled IIoT. In *2019 IEEE Wireless Communications and Networking Conference (WCNC'19)*. 1–7. https://doi.org/10.1109/WCNC.2019.8885703
[39] Joel Höglund, Samuel Lindemer, Martin Furuhed, and Shahid Raza. 2020. PKI4IoT: Towards public key infrastructure for the Internet of Things. *Computers & Security* 89 (2020), 101658. https://doi.org/10.1016/j.cose.2019.101658
[40] Arshad Jhumka and Matthew Bradbury. [n.d.]. Threats to Buffer Management Eviction Strategies for Trust Computation in Resource-constrained IoT Networks. ([n. d.]). In Submission.
[41] Chris Johnson, Lee Badger, David Waltermire, Julie Snyder, and Clem Skorupka. 2016. *Guide to Cyber Threat Information Sharing*. NIST Special Publication 800-150. National Institute of Standards and Technology, Gaithersburg, MD. https://doi.org/10.6028/NIST.SP.800-150
[42] Audun Jøsang and Roslan Ismail. 2002. The beta reputation system. In *15th Bled Electronic Commerce Conference*. University of Maribor Press, 14 pages.
[43] Latif U. Khan, Ibrar Yaqoob, Nguyen H. Tran, S. M. Ahsan Kazmi, Tri Nguyen Dang, and Choong Seon Hong. 2020. Edge computing enabled smart cities: A comprehensive survey. *IEEE Internet of Things Journal* 7, 10 (2020), 10200–10232. https://doi.org/10.1109/JIOT.2020.2987070
[44] Arjen K. Lenstra and Eric R. Verheul. 2001. Selecting cryptographic key sizes. *Journal of Cryptology* 14, 4 (2001), 255–293. https://doi.org/10.1007/s00145-001-0009-4
[45] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. 2017. Multiprogramming a 64kB computer safely and efficiently. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17)*. ACM, New York, NY, 234–251. https://doi.org/10.1145/3132747.3132786
[46] Roger A. Light. 2017. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software* 2, 13 (2017), 265. https://doi.org/10.21105/joss.00265
[47] Pavel Mach and Zdenek Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys Tutorials* 19, 3 (2017), 1628–1656. https://doi.org/10.1109/COMST.2017.2682318
[48] Brendan Moran, Hannes Tschofenig, Henk Birkholz, and Koen Zandberg. 2020. *A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest*. Internet-Draft draft-ietf-suit-manifest-11. IETF. http://www.ietf.org/internet-drafts/draft-ietf-suit-manifest-11.txt.
[49] Aristides Mpitziopoulos, Damianos Gavala, Charalampos Konstantopoulos, and Grammati Pantziou. 2009. A survey on jamming attacks and countermeasures in WSNs. *IEEE Communications Surveys Tutorials* 11, 4 (2009), 42–56. https://doi.org/10.1109/SURV.2009.090404
[50] Michael Muckin and Scott C. Fitch. 2019. *A Threat-driven Approach to Cyber Security*. Technical Report. Lockheed Martin Corporation. https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Threat-Driven-Approach.pdf.
[51] Nordic Semiconductor. 2019. *nRF52840*. Product Specification 4413_417. https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf. V1.1.
[52] Opeyemi A. Osanaiye, Attahiru S. Alfa, and Gerhard P. Hancke. 2018. Denial of service defence for resource availability in wireless sensor networks. *IEEE Access* 6 (2018), 6975–7004. https://doi.org/10.1109/ACCESS.2018.2793841
[53] Shengli Pan, Zhiyong Zhang, Zongwang Zhang, and Deze Zeng. 2019. Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach. *IEEE Access* 7 (2019), 134742–134753. https://doi.org/10.1109/ACCESS.2019.2942052
[54] Basit Qureshi, Geyong Min, and Demetres Kouvatsos. 2013. Countering the collusion attack with a multidimensional decentralized trust and reputation model in disconnected MANETs. *Multimedia Tools and Applications* 66, 2 (2013), 303–323. https://doi.org/10.1007/s11042-011-0780-7

[55] Sukhchandan Randhawa and Sushma Jain. 2017. Data aggregation in wireless sensor networks: Previous research, current status and future directions. *Wireless Personal Communications* 97, 3 (Dec. 2017), 3355–3425. https://doi.org/10.1007/s11277-017-4674-5

[56] David R. Raymond, Randy C. Marchany, Michael I. Brownfield, and Scott F. Midkiff. 2009. Effects of denial-of-sleep attacks on wireless sensor network MAC protocols. *IEEE Transactions on Vehicular Technology* 58, 1 (Jan. 2009), 367–380. https://doi.org/10.1109/TVT.2008.921621

[57] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. 2018. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems* 78 (2018), 680–698. https://doi.org/10.1016/j.future.2016.11.009

[58] Elisa Rondini, Stephen Hailes, and Li Li. 2008. Load sharing and bandwidth control in mobile P2P wireless sensor networks. In *2008 6th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'08)*. IEEE, 468–473. https://doi.org/10.1109/PERCOM.2008.41

[59] Paul Saitta, Brenda Larcom, and Michael Eddington. 2005. *Trike v.1 Methodology Document*. Technical Report. OctoTrike. https://www.octotrike.org/papers/Trike_v1_Methodology_Document-draft.pdf. Draft

[60] Carlos Segarra, Ricard Delgado-Gonzalo, and Valerio Schiavoni. 2020. MQT-TZ: Hardening IoT brokers using ARM TrustZone. In *39th International Symposium on Reliable Distributed Systems (SRDS'20)*. IEEE, 256–265. https://doi.org/10.1109/SRDS51746.2020.00033

[61] Göran Selander, John Mattsson, Francesca Palombini, and Ludwig Seitz. 2019. Object Security for Constrained RESTful Environments (OSCORE). RFC 8613. https://doi.org/10.17487/RFC8613

[62] Göran Selander, John Preuß Mattsson, and Francesca Palombini. 2021. *Ephemeral Diffie-Hellman Over COSE (EDHOC)*. Internet-Draft draft-ietf-lake-edhoc-12. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draftietf-lake-edhoc-12. Work in Progress.

[63] Ali Shakarami, Ali Shahidinejad, and Mostafa Ghobaei-Arani. 2020. A review on the computation offloading approaches in mobile edge computing: A game-theoretic perspective. *Software: Practice and Experience* 50, 9 (2020), 1719–1759. https://doi.org/10.1002/spe.2839

[64] Zach Shelby, Klaus Hartke, and Carsten Bormann. 2014. The Constrained Application Protocol (CoAP). RFC 7252. https://doi.org/10.17487/RFC7252

[65] Adam Shostack. 2008. Experiences threat modeling at Microsoft. In *Proceedings of the Workshop on Modeling Security (MODSEC'08)*. Jon Whittle, Jan Jürjens, Bashar Nuseibeh, and Glen Dobson (Eds.). CEUR-WS, 11 pages. http://ceur-ws.org/Vol-413/paper12.pdf.

[66] Phillip Taylor, Lina Barakat, Simon Miles, and Nathan Griffiths. 2018. Reputation assessment: a review and unifying abstraction. *Knowledge Engineering Review* 33 (2018), e6. https://doi.org/10.1017/S0269888918000097

[67] Sobit Thapa and Qijun Gu. 2013. Originator data security for collaborative task execution among weak devices. In *10th International Conference on Mobile Ad-Hoc and Sensor Systems*. IEEE, 421–422. https://doi.org/10.1109/MASS.2013.24

[68] The Linux Foundation. 2020. The Zephyr Project. Retrieved January 26, 2021, from https://zephyrproject.org.

[69] Marco Tiloca, Domenico De Guglielmo, Gianluca Dini, Giuseppe Anastasi, and Sajal K. Das. 2018. DISH: DIstributed SHuffling Against Selective Jamming Attack in IEEE 802.15.4e TSCH Networks. *ACM Transactions on Sensor Networks* 15, 1, Article 3 (Dec. 2018), 28 pages. https://doi.org/10.1145/3241052

[70] Marco Tiloca, Göran Selander, Francesca Palombini, and Jiye Park. 2021. *Group OSCORE - Secure Group Communication for CoAP*. Internet-Draft draft-ietf-core-oscore-groupcomm-13. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-13. Work in Progress.

[71] Ivana Tomić and Julie A. McCann. 2017. A survey of potential security issues in existing wireless sensor network protocols. *IEEE Internet of Things Journal* 4, 6 (2017), 1910–1923. https://doi.org/10.1109/JIOT.2017.2749883

[72] Tzeta Tsao, Roger Alexander, Mischa Dohler, Vanesa Daza, Angel Lozano, and Michael Richardson. 2015. A Security Threat Analysis for the Routing Protocol for Low-Power and Lossy Networks (RPLs). RFC 7416. https://doi.org/10.17487/RFC7416

[73] Tony UcedaVelez and Marco M. Morana. 2015. *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. John Wiley & Sons, Inc., Hoboken, NJ.

[74] Sebastian Vasile, David Oswald, and Tom Chothia. 2019. Breaking all the things—A systematic survey of firmware extraction techniques for IoT devices. In *Smart Card Research and Advanced Applications*. Begül Bilgin and Jean-Bernard Fischer (Eds.). Springer International Publishing, 171–185.

[75] Bo Wang, Mingchu Li, Xing Jin, and Cheng Guo. 2020. A reliable IoT edge computing trust management mechanism for smart cities. *IEEE Access* 8 (2020), 46373–46399. https://doi.org/10.1109/ACCESS.2020.2979022

[76] Jun Wang, Xi Xiong, and Peng Liu. 2015. Between mutual trust and mutual distrust: Practical fine-grained privilege separation in multithreaded applications. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'15)*. USENIX Association, 361–373.

[77] Yating Wang, Ing-Ray Chen, Jin-Hee Cho, and Jeffrey J. P. Tsai. 2017. Trust-based task assignment with multiobjective optimization in service-oriented ad hoc networks. *IEEE Transactions on Network and Service Management* 14, 1 (2017), 217–232. https://doi.org/10.1109/TNSM.2016.2636454

[78] Thomas Watteyne, Maria Rita Palattella, and Luigi Alfredo Grieco. 2015. Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement. RFC 7554. https://doi.org/10.17487/RFC7554

[79] Wojciech Wideł, Maxime Audinot, Barbara Fila, and Sophie Pinchinat. 2019. Beyond 2014: Formal methods for attack tree–based security modeling. *ACM Computing Surveys* 52, 4, Article 75 (Aug. 2019), 36 pages. https://doi.org/10.1145/3331524

[80] Dexiang Wu, Guohua Shen, Zhiqiu Huang, Yan Cao, and Tianbao Du. 2019. A trust-aware task offloading framework in mobile edge computing. *IEEE Access* 7 (2019), 150105–150119. https://doi.org/10.1109/ACCESS.2019.2947306

[81] Han Yu, Zhiqi Shen, Cyril Leung, Chunyan Miao, and Victor R. Lesser. 2013. A survey of multi-agent trust management systems. *IEEE Access* 1 (2013), 35–50. https://doi.org/10.1109/ACCESS.2013.2259892

[82] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. 2019. Secure firmware updates for constrained IoT devices using open standards: A reality check. *IEEE Access* 7 (2019), 71907–71920. https://doi.org/10.1109/ACCESS.2019.2919760

[83] Zolertia. 2016. *Zolertia RE-Mote Revision B Internet of Things hardware development platform, for 2.4-GHz and 863-950MHz IEEE 802.15.4, 6LoWPAN and ZigBee® Applications.* Datasheet ZOL-RM0x-B. Barcelona, Spain. V1.0.0.

Received May 2021; revised October 2021; accepted January 2022