Tailoring Onion Routing to the Internet of Things: Security and Privacy in Untrusted Environments

Jens Hiller*, Jan Pennekamp*, Markus Dahlmanns*, Martin Henze†, Andriy Panchenko‡, Klaus Wehrle*
*Communication and Distributed Systems, RWTH Aachen University, Aachen, Germany
{hiller, pennekamp, dahlmanns, wehrle}@comsys.rwth-aachen.de
†Cyber Analysis & Defense, Fraunhofer FKIE, Bonn-Bad Godesberg, Germany · martin.henze@fkie.fraunhofer.de
‡IT Security, Brandenburg University of Technology, Cottbus, Germany · andriy.panchenko@b-tu.de

***Abstract*—An increasing number of IoT scenarios involve mobile, resource-constrained IoT devices that rely on untrusted networks for Internet connectivity. In such environments, attackers can derive sensitive private information of IoT device owners, e.g., daily routines or secret supply chain procedures, when sniffing on IoT communication and linking IoT devices and owner. Furthermore, untrusted networks do not provide IoT devices with any protection against attacks from the Internet.**

Anonymous communication using onion routing provides a well-proven mechanism to keep the relationship between communication partners secret and (optionally) protect against network attacks. However, the application of onion routing is challenged by protocol incompatibilities and demanding cryptographic processing on constrained IoT devices, rendering its use infeasible.

To close this gap, we tailor onion routing to the IoT by bridging protocol incompatibilities and offloading expensive cryptographic processing to a router or web server of the IoT device owner. Thus, we realize resource-conserving access control and end-to-end security for IoT devices. To prove applicability, we deploy onion routing for the IoT within the well-established Tor network enabling IoT devices to leverage its resources to achieve the same grade of anonymity as readily available to traditional devices.

I. Introduction

In trusted environments, e.g., smart homes, the establishment of security for IoT devices is well understood. However, the emergence of other private and industrial *mobile* use cases, such as pet monitoring [1], [2] or container and parcel tracking [3]–[5], involve IoT devices that operate in untrusted networks, e.g., using public Wi-Fi or ad-hoc networks for Internet access. Inevitably, new risks come up in these environments. First, the relationship between communicating parties reveals private or secret routines [6] and, combined with location data, eases theft of valuable goods [5]. Second, IoT devices in untrusted networks lack either a globally routable IP address [7] or traditional defense mechanisms such as firewalls and intrusion detection systems [8].

Current networking paradigms offer two alternatives to tackle these problems: *Cloud computing*, which is often used to access statistics and functionality of IoT devices via smartphone apps [9], introduces a point of indirection which hides the relation of devices that exchange information, establishing a mutually reachable point of contact. However, severe concerns challenge the trust in cloud providers due to their ever-increasing data gathering in all areas of life [10], incidents of data theft and misuse [11], and a general lack of control on data handling by the cloud [12]–[14]. Thus, to omit this single point of trust, we desire a direct communication with or between IoT devices [15]. In such a setting, anonymous communication using *onion routing* likewise promises to prevent a linkage of communication partners [16]–[19] and to ensure reachability coupled with access control [16], [20].

However, resource constraints of IoT devices challenge the use of traditional anonymous communication systems. Specifically, limited processing capabilities of IoT devices impede the execution of heavyweight cryptographic public key operations [21]–[24] used for connection establishment. The problem of involved processing exacerbates for mobile IoT devices as it depletes limited energy resources [25]. Furthermore, resource constraints and operation in lossy wireless networks force IoT devices to implement special protocols [22]–[24] which are incompatible to traditional anonymous communication, which depends on a reliable transport and is thus incapable of dealing with the frequent packet loss and reordering in the IoT.

In this paper, we close the gap that hinders the application of anonymous communication in the IoT domain. To this end, we tailor onion routing to the IoT using a *delegation* paradigm. More specifically, our contributions are as follows:

- We demonstrate the computational infeasibility of establishing onion routing on IoT devices, highlight protocol incompatibilities between anonymous communication and the IoT domain, and emphasize the need for proper access control to limit the attack surface.
- We realize onion routing for the IoT by delegating processing intensive tasks during connection setup to trusted entities. Our mechanism can leverage the benefits of existing, widely deployed onion routing networks such as Tor without requiring changes to the vast majority of existing onion routers and our implementation is public [26], [27].
- We thoroughly analyze the performance and applicability of our approach. Our delegation method results in only a modest 1.7 s latency increase for connection establishment and realizes proper access control with 1 s latency overhead. We achieve these reasonable circuit setup times as delegation strikingly decreases the local processing at IoT devices from several seconds or minutes to well-manageable 53 to 146 ms. Our approach leads to a negligible overhead for only a small number of nodes in an existing onion network.

Author manuscript.

 DOI: 10.1109/ICNP.2019.8888033

II. IoT Scenarios and Attacker Model

An increasing number of scenarios relies on the IoT to deliver core functionality. In the following, we discuss the resulting communication models and use cases for Internet-connected IoT devices and define our attacker model as a foundation to provide security in these scenarios.

A. IoT Scenarios & Communication Models

We distinguish two primary communication models in the IoT. Interactive communication with end-user devices, e.g., smartphones, (cf. Figure 1a) realizes direct data access and remote control. Complementary, communication between IoT devices (cf. Figure 1b) enables automated workflows. An increasing number of scenarios depend on *mobile* IoT devices that rely on (public) wireless access points or cellular uplinks:

Supply chain, container, parcel, and delivery robot tracking [3]–[5] enable shipper and consignee to track the current location of freight. IoT devices also can report environmental conditions [28] of perishable or fragile cargo, e.g., to trigger temperature and humidity changes by other IoT devices or to enable evaluation of transport conditions by the consignee.

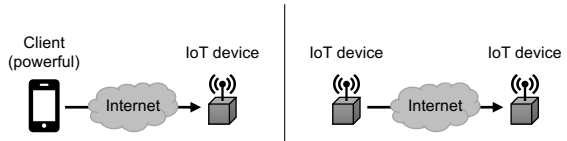
Pet or wildlife monitoring [1], [2] enables pet owners and rangers to interactively track movement, life signs, and even live footage or videos of animals. Other IoT devices automatically unlock smart doors, control food dispensers, or inform rangers of poaching to protect endangered species.

Similarly, *smart home devices on travel*, e.g., alarm systems for hotel rooms, as well as *smart travel equipment*, such as smart bags, enhance user experience on travel (in untrustworthy networks) with functionality similar to traditional smart home applications and tracking of luggage.

B. Attacker Model for IoT Scenarios

To keep cloud providers oblivious of information derived from IoT usage (Section I), we consider it inevitable that IoT devices communicate *directly* with each other (cf. Figure 1). However, direct communication still risks security and privacy of users as attackers can link communication partners [29] which enables surveillance and tracking, e.g., an attacker can link goods in supply chains or pets with the respective communication partner and owner. This data can reveal private daily routines, secret business processes, and, combined with location information, ease theft of valuable goods [5].

Furthermore, direct communication introduces an attack surface as IoT devices provide a globally reachable server [30], [31]. Thereby, IoT devices expose potentially vulnerable implementations to the Internet. Especially in mobile scenarios, they typically access the Internet via *untrusted networks*, which lack reliable in-network security mechanisms, e.g., firewalls and intrusion detection systems [32]. In particular, IoT devices use cellular networks of (foreign) providers [3], leverage free Wi-Fi hotspots, or establish ad-hoc networks, e.g., using infrastructure-based IoT networks for parking or traffic management systems. Consequently, IoT devices can be contacted by anyone such that attackers can exploit vulnerabilities for DoS or takeover. Furthermore, on-path attackers



(a) Interactive scenario (b) IoT to IoT communication

Fig. 1. The two different IoT communication models have to support mobile communication over untrusted networks.

can sniff on data packets to retrieve sensitive data, and link the end-points of communication to gain private meta information.

Its inherent property of protecting data transmission and achieving unlinkability of communication partners [33] makes anonymous communication a promising solution to protect IoT devices and their owners from such attacks.

III. Anonymous Communication

Anonymous communication unlinks communication partners and protects data transmission in *high-latency* (without real-time constraints, e.g., email [34]) and *low-latency* (real-time communication, e.g., web browsing or instant messaging) settings. Since connections to IoT devices are often interactive, we focus on low-latency communication. The achievable level of anonymity depends on the number of participants, motivating the use of already well-deployed anonymization networks.

a) Establishing Anonymous Connections: A common approach to achieve sender anonymity is called *onion routing* [16], [17]. It utilizes distributed trust across multiple nodes by sending packets on a detour over several hops, called onion relays (ORs). At each of these hops, one layer of encryption is added or removed, altering each packet. The client establishes a virtual tunnel, called *circuit*, to the destination over typically three ORs: guard, middle, and exit. To this end, the sender negotiates a separate symmetric key with each OR to encrypt the application data in multiple layers. Tor [16], [35] is the de-facto standard of such an approach.

In a Tor circuit, each OR only knows (i) which peer has sent it data and (ii) to which peer it is relaying data. Here, a circuit length of three constitutes a reasonable trade-off between security and performance, where the middle OR hinders the *exit* to get to know the *guard* (first OR) and vice versa [16].

b) Hiding Service Locations: Onion services (OSes) additionally provide receiver anonymity, thus protecting against (distributed) denial-of-service (DoS) and physical attacks [36]. As a side effect, OSes can operate even if the server does not own a public IP address or is located behind a firewall. These properties are attractive for IoT devices (cf. Section II-B).

To offer an onion service, a server selects three ORs, called *introduction points* (InPs), and builds a separate circuit to each of them. Next, the server generates an *OS descriptor* [20] and publishes it anonymously to *OS directory servers* (OSDir) to announce the presence of the service and how to connect to it. To access an OS in Tor, the client uses the matching *onion address* to fetch the OS descriptor and thus the respective InPs from an OSDir. Then, the client creates a circuit to a randomly selected OR, called *rendezvous point* (RP), and sends an arbitrary nonce, called *rendezvous cookie*, to that OR. Following, the client builds a new circuit to one of the

InPs to inform the OS about the selected RP and cookie. If the OS accepts the client's request, it establishes a new circuit to the selected RP and sends the cookie. The RP recognizes the cookie and, finally, starts to relay encrypted packets between client and onion service. The server can also be configured to only grant access to authorized clients [20].

IV. Challenges for Onion Routing in the IoT

To enable secure and privacy-preserving communication in the IoT, we propose to interconnect IoT devices using onion routing. However, realizing onion routing for the IoT is challenging due to infeasible computational overheads and protocol incompatibilities. In the following, we detail these challenges and distill resulting requirements.

a) Infeasible Public Key Overhead: The setup of encryption layer keys (cf. Section III) requires public key cryptography which significantly burdens resource-constrained IoT devices w.r.t. processing resources, channel establishment times, and energy resources [22]. Already a single Curve25519 Diffie-Hellman key agreement on a Zolertia Z1 (MSP430 16-bit CPU@16 MHz) takes 117 s. A Tor circuit establishment requires multiple such key agreements which sum up to 11 min of public key processing (even 32 min to connect to an onion service). Even a comparably powerful OpenMote B (ARM Cortex-M3@32 MHz) requires 2.8 s or 7.7 s (cf. Figure 5b).

The resulting high connection setup times render the use of onion routing infeasible for most use cases and the large amount of computations and involved communication depletes limited energy resources of mobile IoT devices. Thus, we strive for a solution that relieves IoT devices from public key computations by offloading them to more powerful devices.

b) Incompatible Protocols & Deployability: IoT devices employ network and security protocols that better account for their resource constraints as protocols typically used on the Internet. While TCP is prevalent in the Internet, IoT devices use UDP to avoid bookkeeping and ROM overheads [37]. As Transport Layer Security (TLS) does not operate on the unreliable transport provided by UDP, IoT devices rely on Datagram TLS (DTLS). This selection leads to incompatibilities when striving to connect IoT devices to existing onion networks, e.g., Tor requires clients to connect using TLS on top of TCP. Thus, to allow connections from IoT devices, onion relays must support IoT protocols. However, onion routing networks are slow in adapting proposed changes [38]. Thus, a need for gradual deployment of IoT support in onion routing networks exists, i.e., allowing IoT devices to profit from onion routing benefits even if only a few ORs adopted the required changes.

A different challenge for IoT onion routing results from unreliable wireless links. IoT networks involve a high risk of loss or reordering of cells which is neither addressed by the unreliable UDP transport nor handled by deployed onion routing systems, such as Tor, which assume the use of a reliable transport protocol. Thus, the use of onion routing in the IoT domain requires us to bring together the lightweight, unreliable communication protocols of IoT domains with the guarantees expected by widely deployed onion routing networks.

c) Resulting Requirements: From these challenges and our scenarios and attacker model (cf. Section II), we distill requirements for security and privacy in untrusted IoT environments. Besides providing *encrypted communication* to prevent sniffing by on-path attackers, e.g., untrusted network providers used by mobile IoT devices for Internet access, a main requirement is to *prevent attackers from linking communication partners*. Realizing this requirement without requiring trust in cloud providers is important, i.e., *independent of a trusted third party* that the IoT device owner cannot control. We address these requirements by tailoring onion routing to the IoT.

Doing so, we have to consider that IoT devices must *provide access* also if (i) the network only provides an *unroutable network address or blocks inbound connections*, or (ii) a mobile IoT device *repeatedly changes its network address*. As IoT devices in untrusted networks are not (necessarily) protected by firewalls or intrusion detection systems (cf. Section II-B), an IoT device should *expose only a minimal set of functionality to unauthorized communication partners* to prevent takeovers and limit the potential of DoS attacks. Furthermore, as such IoT devices can in principle be contacted by anyone, they must *enforce access control* to only grant legitimate peers access to their functionality. Mechanisms that tackle these requirements further have to *consider resource constraints* of IoT devices.

Our approach enables resource-constrained (mobile) devices, as prevalent in the IoT, to use anonymous communication while ensuring proper encryption, access control, and restriction of functionality if exposed to potential attackers.

V. Tailoring Onion Routing to the IoT

The core idea of our approach to tailor onion routing to the IoT is to assist resource-constrained IoT devices by deploying dedicated support functionality at the edge of legacy onion routing networks as shown in Figure 2: *IoT entry nodes* (1) provide dedicated entrance points into the anonymization network that specifically care for the unique requirements of IoT devices. To this end, IoT entry nodes translate between network and security protocols used in traditional onion routing and protocols specifically designed for resource-constrained IoT devices, make IoT devices globally reachable, and enforce universal access control. *IoT devices* (2) execute local access control and enforce end-to-end security. (1) and (2) already enable a *powerful client* (3a), that is permitted to learn the IoT device's network location, to anonymously communicate with the IoT device. Offloading to a *delegation server* (3b), e.g., operated at the IoT device owner's home router, further assists IoT devices in establishing anonymous communication themselves, especially by unburdening them from computationally expensive cryptographic operations. Furthermore, it allows IoT devices to hide their location from only partially trusted clients by operating as IoT-enabled onion services.

In the following, we first explain the functions of IoT entry nodes and IoT devices, allowing powerful clients to anonymously communicate with IoT devices, before we extend our design with offloading to allow constrained devices to establish circuits and act as location-hidden onion service.

A. IoT Entry: Connecting the IoT to Onion Routing Networks

To support resource-constrained IoT devices in using onion routing, we introduce the functionality of an *IoT entry*. More specifically, we elevate a subset of (long-running) guards in a traditional onion routing network to (i) support IoT protocols to translate between onion routing and IoT communication protocols and (ii) let them serve as stable point for establishing connections with IoT devices. Furthermore, we (iii) realize a resource-conserving in-network access control at the IoT entry. In the following, we discuss how we can achieve this functionality for already deployed onion routing networks, allowing IoT devices to benefit from their performance and anonymity. The majority of ORs are *not* extended with IoT functionality and remain unchanged (cf. Figure 2). For Internet connectivity (required to connect to the IoT entry), IoT devices continue to use gateways at the IoT network edge [39]–[42].

1) Bootstrapping, Connections & Access Control: To make the IoT entry act as a globally reachable point of contact and provide in-network access control for connections to IoT devices, we leverage a *cookie* (shared secret between IoT device and client). We use this cookie similar to traditional onion routing which employs cookies to interconnect clients and onion services (cf. Section III). In particular, client and IoT device connect to the same IoT entry and present the same cookie to allow the IoT entry to interconnect them.

a) Bootstrapping Cookie and IoT Entry Selection: Client and IoT device have to select the same cookie and IoT entry (cf. Figure 2). To this end, we perform an out-of-band exchange of a *master secret* and a list of IoT entries (guard nodes with IoT support) during a one-time bootstrapping phase. Based on the master secret, client and IoT device derive keys for the selection of IoT entries (k_{select}) and cookies (k_{cookie}).

To *independently select the same IoT entry*, client and IoT device use k_{select} to compute a keyed hash. They then use the result as index for the preshared list with IoT entries to agree on one. As input for the keyed hash, we use the current quarter and year (e.g., "Q2-2019") to implement the traditional onion routing strategy that keeps a once selected guard for a long time to decrease the risk of selecting a malicious guard [43]. To not disrupt connectivity if an IoT entry fails, IoT devices should connect to multiple IoT entries in parallel or fall back on another in the list (e.g., using inputs such as "Q2-2019-1").

The problem of selecting the same IoT entry is conceptually similar to obtaining knowledge of introduction points of a (traditional) onion service (cf. Section III). However, onion services use a circuit to anonymously publish their introduction points in the onion service directory. This approach would require a constrained IoT device to establish a circuit itself, which is computationally challenging (cf. Section IV).

To *independently create the same cookie*, IoT device and client use k_{cookie} to calculate a keyed hash function on the IoT entry's IP address. We bind the cookie to the IoT entry to prevent the use of a purloined or leaked cookie at a different IoT entry, e.g., to launch an impersonation or DoS attack.

b) Interconnecting Clients and IoT Devices: To use an IoT entry as stable point of contact for an IoT device, we let the IoT device establish a DTLS connection to the IoT entry and send a *cookie* (Figure 3a (0)). A client that wishes to connect to the IoT device establishes a circuit to the IoT entry (Figure 3a (1)) and requests an extension of this connection to the registered IoT device. To this end, we introduce an *IoT connect cell* which is sent from the client to the IoT entry and contains the same cookie as used by the IoT device (Figure 3a (2)). The IoT entry uses the cookie to determine the target IoT device. It then forwards the IoT connect cell to provide the IoT device with the contained HMAC for (local) access control and secrets for end-to-end security (details in Section V-B). Subsequently, the IoT device notifies the acceptance of the circuit to the IoT entry which passes this information on to the client and relays following onion cells between both connections, like a traditional OR.

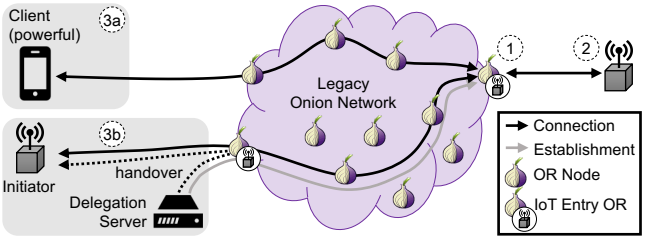
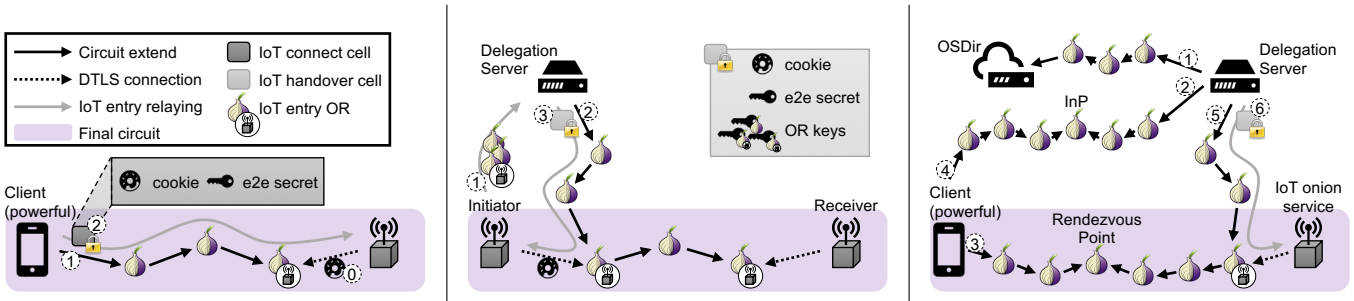


Fig. 2. We compose IoT onion routing out of (1) IoT entry nodes that realize IoT protocol support and reachability as well as resource-efficient in-network access control, (2) complementary access control and enforcement of end-to-end encryption by IoT devices, and (3) offloading of expensive cryptographic operations to a (3a) capable client or (3b) trustworthy delegation server.

c) Using Cookies for In-Network Access Control: To ensure that only authorized entities establish circuits to IoT devices, we keep cookies secret and let IoT entries only forward IoT connect cells with a valid cookie. Specifically, we send cookies only over encrypted and authenticated connections to the IoT entry, i.e., DTLS or onion routing connections (cf. Figure 3a (0) and (2)). Thus, only client, IoT device, and IoT entry know the cookie required to connect to the IoT device.

This access control does not put additional burden on the IoT entry as we show in Section VI-A1. Nevertheless, the cookie mechanism provides only a first layer of defense. A malicious or malfunctioning IoT entry may still enable adversaries to connect to the IoT device. Hence, we complement it with local access control at the IoT device (cf. Section V-B1).

2) Accounting for Protocol Incompatibilities: To allow resource-constrained IoT devices to use existing onion routing networks, the IoT entry *translates* between IoT and Internet protocols. We achieve this by equipping the IoT entry with UDP and DTLS support. Furthermore, our *cell acknowledgment and retransmission scheme* allows IoT entry and IoT device to exchange onion cells over unreliable IoT protocols and (partly wireless) connections. As ORs use implicit counters per circuit to de-/encrypt layers of onion cells, cell loss or reordering leads to wrong counters. As traditional ORs can neither detect nor repair wrong counters, circuits break if cells get reordered or lost [44]. To address this problem, we extend cells between the IoT entry and IoT device with cell numbers to afford reordering and acknowledgment of cells.



(a) Connection Setup & Access Control (b) Delegated Circuit Establishment (c) Location-Hiding IoT Onion Service

Fig. 3. For IoT onion routing, we introduce a *cookie* based connection and in-network access control mechanism (a) which already enables powerful clients to establish circuits to constrained IoT devices. Offloading the circuit setup to a *delegation server* allows IoT devices to initiate circuits themselves (b), and to efficiently operate as location-hiding *IoT onion service* (c).

B. IoT Device: Lightweight E2E Security and Access Control

We design the IoT device to use *lightweight* security mechanisms for those tasks that it can offload neither to the untrusted IoT entry (cf. Section V-A) nor to trusted communication partners or delegation servers (detailed in Section V-C). In particular, we (i) implement local access control to protect against attackers that can bypass in-network access control at IoT entries. Furthermore, we (ii) enforce end-to-end security to prevent on-path entities to read or modify transmitted data.

1) Local Access Control: We design IoT devices to verify the legitimacy of all connection attempts themselves to detect and counter ineffective in-network access control caused by a malicious or malfunctioning IoT entry (cf. Section V-A1c). To this end, we base the local access control on a secret key k_{access} that only the IoT device and legitimate clients know. Specifically, we derive k_{access} from the master secret established between IoT device and client during the one-time bootstrapping phase (cf. Section V-A1a). We instruct the client to use k_{access} to calculate an HMAC to authenticate the content of the IoT connect cell sent via the IoT entry to an IoT device to establish a connection (cf. Section V-A1). Upon reception of the IoT connect cell, the IoT device validates the HMAC and only accepts the circuit if the HMAC is correct. Thus, to connect to the IoT device, an adversary requires knowledge of k_{access} (only known by IoT device and client).

2) End-to-End Security: To protect transmitted data against sniffing or modification, IoT devices enforce end-to-end encryption and data authentication. To this end, we implement an authenticated *Diffie-Hellman (DH) key exchange* [45] within the cells that client and IoT device exchange during circuit establishment via the IoT entry (Figure 3a (2)), similar to a traditional circuit extension. Thus, client and IoT device share perfect forward secret keys for end-to-end security.

However, for severely resource-constrained IoT devices, already a single DH key agreement leads to significant processing overhead, e.g., 117 s on a Zolertia Z1 (MSP430 16-bit CPU@16 MHz). To speed up connection establishment for this device class, we *optionally replace the DH key exchange*. To this end, the client replaces its DH share in the IoT connect cell with a nonce. Client and IoT device use this nonce and their preshared master secret to derive a fresh secret s_{e2e} which they use in place of the otherwise DH-derived secret. As detailed in our security discussion (Section VII), this removes perfect forward secrecy (PFS) against IoT entries but the onion encryption layers still achieve PFS against all other entities.

C. Offloading to a Delegation Server

As public key operations involve high processing overheads for IoT devices, we design a delegated circuit establishment to enable IoT onion routing between two constrained IoT devices (Figure 3b). Afterward, we adapt this approach to allow IoT devices to hide their location from communication partners by operating as *IoT onion service* (Figure 3c). Besides public key operations for circuit creation, our design requires a single public key operation by the IoT device to establish a DTLS connection to the IoT entry (cf. Section V-A1b). IoT devices only rarely establish this long-term DTLS connection. Still, we refer readers to delegation-based DTLS [25] as method to also relieve IoT devices from DTLS public key overhead.

1) Circuit Establishment with a Delegation Server: Since the establishment of circuits involves high processing overhead for resource-constrained IoT devices, we introduce the concept of a *delegation server*, a trusted entity operated by the IoT device owner. As untrusted networks used by IoT devices lack such a trusted entity, our design allows for deployment of the delegation server anywhere in the Internet, e.g., on the IoT device owner's home router or web server. As we show in Figure 3b, the task of the delegation server is to *establish circuits* on behalf of the IoT device (2), and *hand over* the circuit to the IoT device via the IoT entry (3).

Our design intentionally relies on the delegation server only for connection establishment. The alternative, i.e., relaying all data through the delegation server, would incur a performance penalty resulting from longer circuit lengths and load on the delegation server. In the following, we describe the delegation approach when connecting two IoT devices, but the approach likewise allows an IoT device to establish circuits to traditional Internet devices and onion services or IoT onion services.

To *trigger a delegated circuit establishment*, an IoT device and its delegation server maintain a long-term IoT onion routing circuit. To unburden the IoT device from establishing this circuit itself, the powerful delegation server establishes the circuit to the IoT device (cf. Figure 3a). We (explicitly) use a

circuit as control channel to hide the relationship between the IoT device and its delegation server, and thus its owner.

To start the delegated circuit setup to another IoT device, the initiating IoT device (initiator) uses the long-term circuit to inform the delegation server about the desired target (Figure 3b (1)). The delegation server then *establishes a circuit* to the target (Figure 3b (2)). Intuitively, this circuit needs to flow from the IoT entry of the initiator (Figure 3b left) over some middle node to the IoT entry of the target. This behavior guarantees unlinkability between initiator and target.

To further conceal the relationship between delegation server and initiator, the delegation server temporarily adds two additional ORs to the start of the circuit, i.e., before the IoT entry of the initiator. This step is important as any link between these two devices would reveal the identity of the initiator's owner. The two additional ORs are removed from the circuit when it is handed over to the initiating IoT device.

To actually *hand over* the circuit to the initiating IoT device, we introduce an IoT handover cell (Figure 3b (3)) which extends the IoT connect cell used by powerful clients. The delegation server sends this IoT handover cell to the IoT entry (third OR). The IoT entry applies the same cookie-based access control as for IoT connect cells and forwards the IoT handover cell to the IoT device (cf. Section V-A1). Besides the secret for end-to-end security, the IoT handover cell provides the IoT device with the keys to create onion encryption layers for the third, fourth, and fifth OR. To finish the handover, the IoT device instructs the IoT entry to close the circuit towards the delegation server and relay any further onion cells between the IoT device and the target (Figure 3b bottom).

By introducing the delegation server, we unburden IoT devices from any public key overhead for circuit establishment.

2) Hiding IoT Device Location – IoT Onion Service: IoT devices should not only be able to prevent outsiders to link them with communication peers, but also hide their location from clients. For example, researchers, rangers, or insurances are interested in live status information such as medical condition of IoT-equipped pets and wildlife, or attrition of machines. While owners may provide such access for the common wealth or financial benefits, the data gathering purpose does often not justify access to location information which would invade the privacy of the IoT device owner. As traditional onion routing hides device locations with onion services, we also strive to allow IoT devices to operate as *IoT onion service*. To this end, we (i) *offload the traditional onion service circuit setup* (cf. Section III) to the delegation server and (ii) *hand over the final circuit* to the IoT device that acts as IoT onion service.

Our delegated circuit setup (Section V-C1) in principle already affords to hide the location of IoT devices by delegating any circuit setup required to operate as onion service. However, to relieve resource-constrained IoT devices from processing multiple handovers, we enable them to offload the complete onion service circuit setup to their delegation server.

Figure 3c shows the connection setup to an IoT onion service. Up to Step (5) we use the design of traditional onion services. Specifically, the delegation server—on behalf of the IoT device—advertises itself as onion service in the OSDir (1) and establishes circuits to introduction points (2). To connect, a client first selects and establishes a circuit to a rendezvous point (3) and then connects to one of the delegation server's introduction points, thereby sending a traditional introduce cell which contains information on the rendezvous point (4).

Deviating from the traditional design, the delegation server uses our delegated circuit establishment to connect to the rendezvous point (5) and hands over the circuit to the IoT device (6). The handover again includes the cookie-based access control to authenticate the delegation server. Furthermore, it provides the IoT device with keys for the circuit up to the rendezvous point and—for onion services mandatory—end-to-end security. Additionally, the delegation server extends the IoT handover cell with a precomputed traditional rendezvous cell. Returning to the traditional design, the IoT device forwards this rendezvous cell to the rendezvous point to let it interconnect the circuits created in Steps (3) and (5). As the rendezvous cell contains the public DH share for end-to-end security negotiated by the delegation server (henceforth used by the IoT device) the rendezvous point relays it to the client. Subsequently, client and IoT device can exchange payload. As important aspect of our design, clients do not even notice that the onion service is operated by an IoT device as the delegation server and IoT entry handle all IoT specific mechanisms.

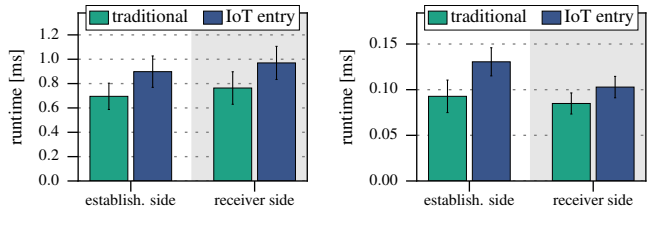
As for scenarios without hidden location, we require clients to legitimate their access to IoT devices. To this end, the delegation server applies—traditionally optional—*onion service client authorization* [20], i.e., among other things, it adds encrypted introduction points to the onion service directory.

VI. Evaluation of IoT Onion Routing

To show the applicability of our design and evaluate its performance, we implemented it for the widely deployed onion routing network Tor and performed measurements both in a controlled local testbed and the real Tor network.

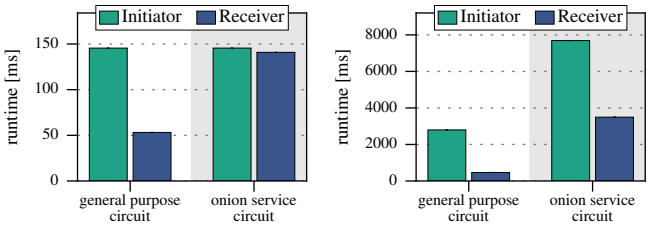
a) Implementation for Tor and Contiki-NG: We implemented IoT onion routing for Tor 0.3.2 [16], [35]. Specifically, we enhanced the onion relay implementation with IoT entry functionality and adapted the Tor client to support our connection establishment with cookies. By further enhancing client functionality with our circuit handover approach, we realized a delegation server for Tor. With similar adaptations to Tor's onion service code, we extended the delegation server to also assist IoT devices to operate as IoT onion service.

To enable IoT devices to use onion routing, we implemented IoT device functionality for the operating system contiking [46]. We use tinyDTLS [47], [48] to realize DTLS connections to IoT entries. Based on these connections, we realized the cookie registration at IoT entries. Furthermore, we ported the cell processing of Tor and added support for our IoT-specific connect and handover cells. Finally, we implemented acknowledgments to retransmit and reorder cells that IoT entry and IoT device exchange over unreliable UDP/DTLS.



(a) Establishment (b) Payload

Fig. 4. Processing at **IoT entries** is mostly indistinguishable from traditional ORs. Hence, they can support IoT functionality without performance penalties.



(a) IoT onion routing (b) Traditional onion routing

Fig. 5. Offloading relieves **IoT devices** from high Diffie-Hellman processing times ((a) vs. (b)). The remaining overhead (a) depends on the number of onion layer de-/encryptions in addition to end-to-end security.

b) Deploying Onion Routing for the IoT: To evaluate the performance, we deployed our implementation in a local testbed as well as interconnected it with the real Tor network.

To realize all three scenarios (cf. Figure 3), we run our IoT device implementation on two OpenMote B (ARM Cortex-M3@32 MHz, 32 kB RAM) which provide connectivity via an 802.15.4 network. Furthermore, we used four desktop grade machines (Intel i5@3.3 GHz, 16 GB RAM) to deploy two IoT entries, one delegation server, and one (powerful) client. For our local testbed, we deployed four additional desktop grade machines with Tor's traditional onion relay implementation.

Furthermore, to derive runtime performance for extremely constrained IoT devices, we measured onion routing processing on a Zolertia Z1 (MSP430 16-bit CPU@16 MHz).

To not impede the security of users in the real Tor network, we tested our implementation in our isolated local testbed and only moved to the real Tor network once our implementation proved to operate correctly. When deploying our IoT entries in the real Tor network, we ensured to only perform measurements on our own circuits. To this end, we marked our own circuits with a special cell sent from our client to the IoT entry. Thus, we neither altered any traffic of real Tor users nor performed any measurements on their circuits, as demanded by the Tor guidelines for ethical research [49], [50].

A. Local Testbed: Influence on Performance

To gain a detailed view on the factors influencing performance, we measured both circuit establishments and payload transfer in a controlled local testbed. In particular, we compare runtimes of our implementation with the performance of traditional Tor. We show that onion routing in the IoT is feasible for all participating devices, i.e., onion relays, IoT devices, and delegation servers. As latencies between onion relays vary, we further analyze the impact of latency on circuit establishment and payload transfer. We report on the arithmetic mean of 30 runs and show 99% confidence intervals.

1) Marginal Overhead for Tor Network: The performance of onion relays determines the performance of the whole Tor network as they make up circuits and forward communication data. Thus, we first analyze the overhead for *onion relays*, i.e., we compare the runtime overhead of traditional guard and exit nodes with our IoT entries in Figure 4. The key message is that processing on IoT entries when establishing IoT onion routing circuits is *indistinguishable* (no statistical significance) from traditional onion routing (ranging in only tens to hundreds of *micro*seconds for a single circuit) regardless of the scenario, i.e., independent of if the IoT entry forwards an IoT connect cell or participates in a handover of a circuit from a delegation server to the IoT device. Similarly, IoT entries face almost no additional overhead for processing cells that transport payload. Hence, IoT entries can readily provide the functionality for tailoring onion routing to the IoT without performance penalties.

2) IoT Devices: One Order of Magnitude Less Processing: To show that our design makes onion routing viable for IoT devices, we analyze their processing when establishing IoT onion routing circuits (Figure 5a). We put these results into perspective with the infeasible public key processing on IoT devices required for traditional onion routing (Figure 5b).

First considering (non-location hiding) general purpose circuits, an IoT device requires about 146 ms of processing to establish a circuit—assisted by its delegation server—as shown in Figure 5a. This time splits almost evenly between handling the handover ticket and processing encryption layers for onion cells that finish the circuit establishment. In comparison, an IoT device that accepts an incoming circuit (either from a traditional client or a delegation-assisted IoT device) only has to take care of end-to-end security (about 53 ms).

Considering the overhead when using (location-hiding) onion service circuits, a circuit establishing IoT device requires the same processing as for a general purpose circuit, as additional processing is offloaded to the delegation server. An IoT device that acts as IoT onion service, i.e., accepts incoming onion service circuits, faces higher processing. More specifically, it is assisted by a delegation server to establish its part of the circuit up to the rendezvous point and also has to create the encryption layers for this part of the circuit.

Overall, our work decreases the former infeasible processing overhead at IoT devices to well manageable overheads. As shown in Figure 5b, already the cryptographic processing for circuit establishment alone would take an OpenMote B 2.8 s, compared to 141 to 146 ms with our IoT onion routing approach. Establishment of a circuit to an onion service would even require 7.7 s of processing at the IoT device, whereas IoT onion routing still only requires about 146 ms. This benefit increases distinctly for even more constrained IoT devices. For example, a Zolertia Z1 would need roughly 11 min of local processing to establish a circuit itself. Our delegation approach reduces this to well manageable 601 ms, making onion routing feasible for this device class. Similarly, accepting a circuit as

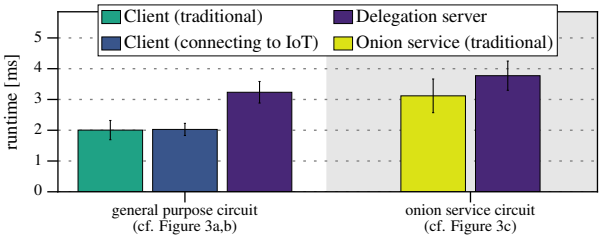


Fig. 6. IoT connect and handover via cookies causes unnoticeable overhead at IoT communication partners. Delegation servers face additional processing for establishing longer circuits that hide their relation to the IoT device.

location-hidden IoT onion service only requires about 639 ms of processing (instead of 32 min without delegation).

3) Viable for Delegation Servers: Next, we analyze the overhead for (powerful) clients that connect to IoT devices via IoT entries as well as the overhead for delegation servers that assist IoT devices in establishing circuits.

As detailed in Figure 6, a *powerful client* faces no overhead when connecting to an IoT device compared to a connection to a traditional server via onion routing. A *delegation server* that establishes a circuit on behalf of an IoT device faces only a well manageable overhead of 1 ms to ensure its unlinkability to the IoT device utilizing 5 ORs instead of 3 (cf. Figure 3b).

Studying the location-hiding onion service case, delegation servers can easily manage the marginal overhead when assisting IoT devices in operating as onion service (0.7 ms compared to a traditional onion service; again due to a longer circuit)

4) Reasonable Connection Establishment Times: As the Tor network is globally distributed, high latencies between onion relays and towards end-points influence circuit establishment and payload transfer times. To obtain insights into the influence of latency, we use *netem* to artificially apply 0, 50, 100, and 150 ms of latency between each pair of machines in our testbed (excluding 802.15.4 networks). Figure 7 details the resulting circuit establishment times for our three IoT scenarios (left) and the two traditional scenarios without IoT devices (right). As our offloading reduces IoT device processing from infeasible several seconds to milliseconds, local processing is almost negligible. The remaining time for networking increases proportionally with the latency and circuit length. Compared to the powerful client scenario, a delegated circuit setup requires more networking due to two low-power 802.15.4 networks (cf. Figure 3b). While our IoT onion service scenario involves only one 802.15.4 network, it faces more networking due to an additional and a longer circuit for introduction and rendezvous point, respectively. Putting our results into perspective with traditional onion routing (right), our approach faces some overhead, mostly due to the inevitable use of 802.15.4 networks. Importantly, our approach yields reasonable circuit setup times (cf. real world results in Section VI-B) especially as our offloading strikingly decreases the local processing at IoT devices (cf. Section VI-A2).

5) Round-Trip-Times for Application Payload: To assess the feasibility of application payload exchange with IoT onion routing, we measured the round-trip-time (RTT) of a single payload cell (512 byte) for our IoT and the traditional settings.

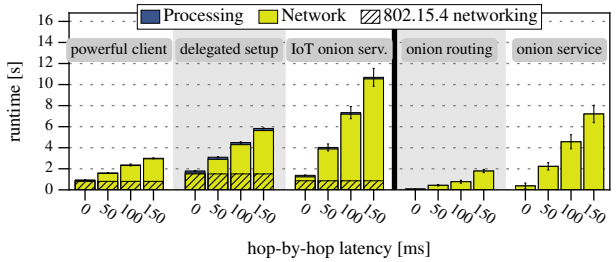


Fig. 7. IoT networking limits the performance compared to cable-based networks. Delegation approaches increase **connection setup** times due to temporarily longer circuits but initially realize feasible connection establishment.

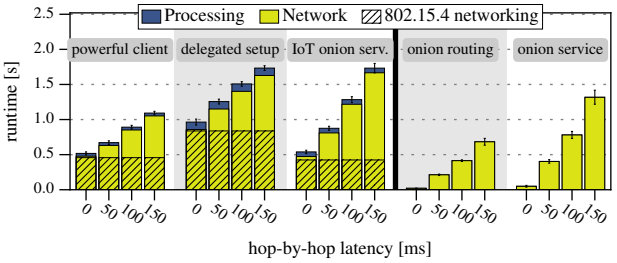


Fig. 8. **Payload** round-trip-times moderately increase due to lossy, wireless networks of IoT devices, especially when Tor interconnects two IoT devices.

As shown in Figure 8, the RTT is again influenced by the number of 802.15.4 networks (powerful client vs. delegated setup) or by the length of the circuit (powerful client vs. IoT onion service). Again, IoT scenarios face overhead mostly due to the use of 802.15.4 networks, especially as we design them to use the same circuit lengths as in traditional onion routing.

In summary, the results obtained in our controlled local testbed show negligible overheads for ORs that serve as IoT entries and only small overheads for devices that assist IoT devices. Furthermore, circuit setup and payload transfer is dominated by the inevitable use of 802.15.4 networks.

B. Performance in Real Tor Network

To prove the actual *applicability* of our approach for real onion routing networks, we interconnected our IoT entries with the live Tor network. To approximate authentic latencies between IoT devices and IoT entries, we set the latency between them drawing from a realistic Tor latency distribution [51].

a) Real World Connection Establishment: Figure 9a shows that connections for our IoT onion routing scenarios in the real Tor network can indeed be established within 1.4 to 4.2 s, with deviations between the three scenarios similar to the observations in our local testbed. These numbers match the time required for establishing circuits in the unmodified Tor network. For instance, the Tor metrics project [52] reports median circuit build times of up to 2 s for circuits of length three. Considering that delegated circuit setup for IoT devices and connections to onion services requires circuits with more than three hops to hide the link between IoT device and delegation server, the performance of establishing connections to and from IoT devices in the real Tor network is well in line with the performance generally achieved in the Tor network.

b) Real World Payload Round-Trip-Times: Also for the exchange of application payload, we obtain similar results as in our local testbed. We visualize in Figure 9b that the transfer

of one cell (512 byte) roughly takes 633 ms when exchanging payload between a powerful client and an IoT device, 1 262 ms for a delegated circuit with two low-power 802.15.4 networks involved, and 855 ms for an IoT onion service. Considering median circuit round-trip latencies in the Tor network of up to 630 ms for connections to public servers and up to 800 ms towards onion servers as report by the Tor metrics project [52], the performance of IoT onion routing is clearly sufficient for real-world deployment in IoT scenarios.

Summarizing, our IoT onion routing design overcomes previously prohibitive overheads, enabling resource-constrained IoT devices to benefit from anonymous communication. In the live Tor network, we even achieve a similar performance for IoT devices as already observed for more powerful devices.

VII. Security Discussion

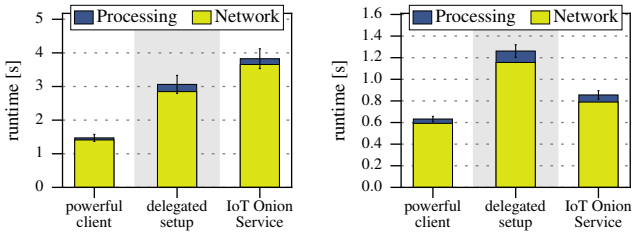
We now discuss how our approach to IoT onion routing addresses the security requirements identified in Section IV.

a) Circuit Security: The inability of attackers to break the encryption of circuits depends on the encryption layer keys derived during circuit extensions. IoT onion routing uses the traditional onion routing circuit extension and thus achieves the same desirable perfect forward security (PFS).

However, when offloading circuit establishment to a delegation server (cf. Section V-C1), encryption keys for the circuit are sent from the delegation server to the IoT device encrypted but without PFS (as the bootstrapping master secret lacks PFS). This is an often selected trade-off for highly constrained devices [25] and only the IoT entry has access to the non-PFS secure encryption of the keys, as the circuit keys are additionally enclosed by PFS-secure onion encryption layers. Thus, we still achieve PFS against all other potential attackers and provide security against malicious IoT entries as sensitive data is—in addition to onion layers— protected by secrets derived during bootstrapping between client and IoT device.

b) Unlinkability: The main goal of onion routing (in the IoT) is to prevent linkage of communication partners, e.g., to keep supply chains private [29]. Since we construct circuits between sender and receiver as in traditional onion routing, we inherit security against non-global observers w.r.t. linkability. To also prevent linkability of an IoT device to its delegation server (and thus device owner), the delegation server uses circuits to communicate with IoT entry and IoT device.

c) Access Control and DoS Protection: Access control prevents unauthorized entities to access data or functionality of an IoT device. In IoT onion routing, a benign IoT entry already blocks (malicious) connections that lack a valid cookie for the IoT device. Thus, our approach establishes an efficient in-network defense against (DoS) attacks that target IoT devices. Consequently, an IoT device only faces malicious connections attempts if a malicious or misconfigured IoT entry does not block unauthenticated attempts or (inadvertently) leaks cookies. In this case, a resource efficient HMAC validation scheme enables IoT devices to detect unauthenticated connection attempts (cf. Section V-B). Additionally, by recognizing



(a) Establishment (b) Payload

Fig. 9. IoT onion routing in the **real Tor network**. Runtimes are influenced by slow IoT networks (left/right: 1, middle: 2) and processing of onion layers at IoT devices. Results for 100 runs (99% confidence intervals).

forged connection attempts, an IoT device can deduce misbehavior of the IoT entry and react by changing to another IoT entry. To not disrupt connectivity due to such changes, IoT devices should negotiate fall-back IoT entries with their communication partners or use multiple IoT entries in parallel.

d) End-to-End Security: To prevent on-path attackers from sniffing or modifying payload in transit, IoT onion routing enforces encryption and authentication based on keys derived during bootstrapping. As end-to-end security with PFS is challenging for resource-constrained IoT devices, they can optionally replace the PFS-secure Diffie-Hellman key exchange with a non-PFS secure key derivation (cf. Section V-B2). Even when replacing the Diffie-Hellman exchange, all data, i.e., nonces for the establishment of keys and the payload, is protected by PFS-secure onion layers. Security against a malicious IoT entry is still established with secrets from the bootstrapping between client and IoT device.

e) Limitations of Malicious IoT Entries: IoT entries, just as guards in traditional onion routing, have a privileged position as they are directly connected to IoT devices. Most notably, this position enables an IoT entry to track the IP address of an IoT device (including network changes in case of mobility) and to mount correlation or traffic analysis attacks [53]–[56]. As the security of traditional onion routing is likewise challenged by malicious guard nodes, extensive research has been conducted to investigate corresponding security issues [57]–[59]. This research shows that sticking to one guard node for a long time yields less security risk than rapid changes, prompting networks such as Tor to keep guards for several months or years [43]. To achieve the same level of security, we follow a similar strategy for selecting IoT entries (cf. Section V-A1a). Importantly, a malicious IoT entry can only track the IoT device but neither identify its communication partners nor the delegation server.

f) Security of IoT Entries: IoT entries are mainly concerned with potential new DoS attack vectors resulting from added functionality. As establishing circuits that include IoT devices requires only a modest overhead compared to establishing traditional circuits, existing DoS mitigations still apply. Likewise, mitigations available for TLS as used by traditional onion relays can—due to the similarity of protocols—be transfered to DTLS as used by IoT entries. Similar considerations hold for mitigating potential attacks on the buffering and reordering of packets necessary to create a reliable link in

the absence of TCP. Finally, our cookie mechanism does not introduce a new attack vector as it is conceptually indistinguishable from cookies that onion relays use to interconnect connections when acting as rendezvous point (cf. Section III).

In summary, our design of IoT onion routing aligns well with the security design of traditional onion routing. Furthermore, we rely on well-established security concepts to reduce the attack surface of our added functionality for IoT support.

VIII. Related Work

Providing security for resource-constrained IoT devices has been studied from various directions. JEDI [21] and Droplet [60] enable efficient end-to-end encryption in many-to-many publish-subscribe scenarios using untrusted third parties. However, they do not hide who publishes to or reads from a data stream. To relieve IoT devices from the expensive setup of DTLS connections, different approaches propose to offload processing or key establishment to trustworthy, more powerful devices [23]–[25], [61], [62]. Another line of research shifts tasks from IoT devices to trusted network elements, e.g., for DoS protection in heterogeneous environments [22] or centralized responses to attacks [63]. While not providing the strong security guarantees of anonymous communication, these approaches provide valuable input for realizing cryptographic operations at constrained IoT devices and improving the performance of our approach, e.g., to realize onion routing without any public key cryptography at the IoT device.

Still, related work shows that basic security functionality, e.g., encryption and authentication, is not sufficient to protect privacy in the IoT. Fachkha et al. [64] show the risks of globally reachable IoT devices by studying probes directed at them. From another angle, encrypted traffic still reveals private information [65], e.g., sleep cycles [66]. While onion routing is generally vulnerable to traffic analysis [53]–[55], we hide the identity of devices and hence provide anonymity.

Moving towards anonymous communication for the IoT, approaches for smart homes introduce trusted, Tor-enabled local gateways to provide anonymous access to IoT functionality [31], [67]. In contrast to our delegation server which can be located anywhere in the Internet, these approaches require an on-site trusted gateway which is an unrealistic assumption in untrusted networks and mobile settings. Yang et al. [8] address resulting latency and bandwidth performance problems in gateway-based IoT deployments by using multiple circuits in parallel. This optimization could also be adapted to our approach where we do not require a trusted on-path gateway to establish circuits on behalf of resource-constrained IoT devices. To gain even more performance, non-anonymous channels [8] instantiate high bandwidth transmissions via Tor circuits. This approach is unsuitable for many IoT scenarios as the non-anonymous channel identifies communication partners and degrades onion routing to a method for access control. Finally, Davoli et al. [68] build an anonymization network directly on top of IoT devices. In contrast to our work, they do not relieve IoT devices from processing overheads but instead put even more load on tightly resource-constrained devices.

Focusing on enhancing anonymous communication and not specifically considering IoT devices, Baumann et al. [69] propose to use Tor to enable device-to-device communication even behind firewalls. However, this approach does not address the resource-constraints of IoT devices or the use of special IoT protocols. To address latency problems in Tor resulting from interference of circuits, Reardon and Goldberg [70] propose to use DTLS (on top of UDP) for communication between onion routers. Contrary, our approach leaves the onion network unchanged and instead deploys DTLS at the edge to enable IoT devices to communication anonymously.

IX. Conclusion

As more and more IoT deployments involve (mobile) devices that use untrusted networks for Internet connectivity, the need for providing security and privacy for these devices becomes paramount. Most importantly, these deployments demand to hide the link between client and IoT device to protect private habits and business secrets. While onion routing prevents this linkage of communication partners, applying it in the IoT domain is challenged by its large use of public key cryptography, overburdening processing capabilities of resource-constrained IoT devices. Furthermore, protocol incompatibilities hinder IoT devices to use well-deployed onion routing networks with large anonymity sets such as Tor.

To tailor onion routing to the specific requirements of the IoT, we introduce IoT entries that enable comparably powerful clients such as smartphones to anonymously communicate with resource-constrained IoT devices. Notably, we only require changes at a small number of nodes at the edge of an existing onion routing network to provide IoT support. Thus, we can leverage the large pool of existing onion routers in already deployed anonymity systems. By further introducing delegation servers, we enable IoT devices to establish onion routing circuits despite their resource constraints and even empower IoT device to provide location-hidden onion services. Our implementation is publicly available [26], [27].

To show the applicability of onion routing for the IoT, we conducted measurements in a local testbed and the real Tor network. Our results show that we can tailor onion routing to the IoT without diminishing security. The processing overheads imposed on IoT devices and delegation servers to establish anonymous connections are well-manageable. Importantly, we observe only negligible overheads for upgrading existing onion relays with IoT functionality. Thus, a sufficient number of onion relays can be enhanced with IoT functionality, easing the deployment of our approach. To further strengthen security, we envision to realize delegation servers as secure network functions with trusted execution environments, e.g., SGX. This would allow us to operate the delegation server even on untrusted infrastructure to further ease deployability.

By tailoring onion routing to the IoT, we enable resource-constrained IoT devices to benefit from the high level of security and privacy offered by anonymous communication.

X. Acknowledgments

The authors thank the German Research Foundation (DFG) for the kind support within the Cluster of Excellence "Internet of Production" (IoP) under project ID 390621612.

References

[1] C.-M. Own, H.-Y. Shin, and C.-Y. Teng, "The Study and Application of the IoT in Pet Systems," *Advances in Internet of Things*, vol. 3, no. 1, 2013.

[2] S. Ramnath, A. Javali, B. Narang, P. Mishra, and S. K. Routray, "IoT based localization and tracking," in *Proceedings of the 2017 International Conference on IoT and Application (ICIOT)*. IEEE, 2017.

[3] Z. D. R. Gnimpieba, A. Nait-Sidi-Moh, D. Durand, and J. Fortin, "Using Internet of Things Technologies for a Collaborative Supply Chain: Application to Tracking of Pallets and Containers," *Procedia Computer Science*, vol. 56, 2015, the 10th International Conference on Future Networks and Communications (FNC) / The 12th International Conference on Mobile Systems and Pervasive Computing (MobiSPC) Affiliated Workshops.

[4] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, 2014.

[5] Franklin Heath Ltd, "LPWA Technology Security Comparison (White Paper)," https://fhcouk.files.wordpress.com/2017/05/lpwa-security-white-paper-1_0_1.pdf, 2017, accessed on August 16, 2019.

[6] J. Pennekamp, R. Glebke, M. Henze, T. Meisen, C. Quix, R. Hai, L. Gleim, P. Niemietz, M. Rudack, S. Knape, A. Epple, D. Trauth, U. Vroomen, T. Bergs, C. Brecher, A. Bührig-Polaczek, M. Jarke, and K. Wehrle, "Towards an Infrastructure Enabling the Internet of Production," in *Proceedings of the 2nd IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2019.

[7] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators," in *Proceedings of the 2005 USENIX Annual Technical Conference (ATC)*. USENIX Association, 2005.

[8] L. Yang, C. Seasholtz, B. Luo, and F. Li, "Hide Your Hackable Smart Home from Remote Attacks: The Multipath Onion IoT Gateways," in *Proceedings of the 23rd European Symposium on Research in Computer Security (ESORICS)*. Springer, 2018.

[9] M. Henze, J. Pennekamp, D. Hellmanns, E. Mühmer, J. H. Ziegeldorf, A. Drichel, and K. Wehrle, "CloudAnalyzer: Uncovering the Cloud Usage of Mobile Apps," in *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*. ACM, 2017.

[10] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of "big data" on cloud computing: Review and open research issues," *Information Systems*, vol. 47, 2015.

[11] D. Chen and H. Zhao, "Data Security and Privacy Protection Issues in Cloud Computing," in *Proceedings of the 2012 International Conference on Computer Science and Electronics Engineering (ICCSEE)*. IEEE, 2012.

[12] E. Fernandes, J. Jung, and A. Prakash, "Security Analysis of Emerging Smart Home Applications," in *Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016.

[13] Intel IT Center, "Peer Research: What's Holding Back the Cloud?" Intel, Tech. Rep., 2012.

[14] M. Henze, "Accounting for Privacy in the Cloud Computing Landscape," Ph.D. dissertation, RWTH Aachen University, 2018.

[15] M. Brachmann, S. L. Keoh, O. G. Morchon, and S. S. Kumar, "End-to-End Transport Security in the IP-Based Internet of Things," in *Proceedings of the 2012 21st International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2012.

[16] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *Proceedings of the 13th USENIX Conference on Security Symposium*. USENIX Association, 2004.

[17] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding Routing Information," in *Proceedings of Information Hiding: First International Workshop*. Springer, 1996.

[18] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman, "Mixmaster Protocol — Version 2," Internet Engineering Task Force: Internet-Draft, 2003.

[19] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a Type III Anonymous Remailer Protocol," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2003.

[20] "Tor Rendezvous Specification," https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt, 2019.

[21] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT," in *Proceedings of the 28th USENIX Conference on Security Symposium*. USENIX Association, 2019.

[22] R. Hummen, H. Wirtz, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Tailoring end-to-end IP security protocols to the Internet of Things," in *Proceedings of the 2013 21st IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2013.

[23] J. Granjal, E. Monteiro, and J. S. Silva, "End-to-end transport-layer security for Internet-integrated sensing applications with mutual and delegated ECC public-key authentication," in *Proceedings of the 2013 IFIP Networking Conference*. IEEE, 2013.

[24] R. Bonetto, N. Bui, V. Lakkundi, A. Olivereau, A. Serbanati, and M. Rossi, "Secure communication for smart IoT objects: Protocol stacks, use cases and practical examples," in *Proceedings of the 2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2012.

[25] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle, "Delegation-based authentication and authorization for the IP-based Internet of Things," in *Proceedings of the 2014 11th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2014.

[26] COMSYS, "tor4iot-tor on GitHub," https://github.com/COMSYS/tor4iot-tor, 2019, accessed on August 21, 2019.

[27] COMSYS, "tor4iot-contiki on GitHub," https://github.com/COMSYS/tor4iot-contiki, 2019, accessed on August 21, 2019.

[28] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, 2013.

[29] J. Pennekamp, M. Henze, S. Schmidt, P. Niemietz, M. Fey, D. Trauth, T. Bergs, C. Brecher, and K. Wehrle, "Dataflow Challenges in an Internet of Production: A Security & Privacy Perspective," in *Proceedings of the 5th ACM Workshop on Cyber-Physical Systems Security and PrivaCy (CPS-SPC)*. ACM, 2019.

[30] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proceedings of the 26th USENIX Conference on Security Symposium*. USENIX Association, 2017.

[31] N. Freitas, "The Internet of Onion Things," https://github.com/n8fr8/talks/blob/master/onion_things/Internet%20of%20Onion%20Things.pdf, 2016, accessed on August 16, 2019.

[32] M. Serror, M. Henze, S. Hack, M. Schuba, and K. Wehrle, "Towards In-Network Security for Smart Homes," in *Proceedings of the 2nd International Workshop on Security and Forensics of IoT (IoT-SECFOR)*. ACM, 2018.

[33] G. Danezis and C. Diaz, "A Survey of Anonymous Communication Channels," *Journal Of Privacy Technology*, 2008.

[34] D. L. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, vol. 24, no. 2, 1981.

[35] The Tor Project, "Tor," https://www.torproject.org/, 2002.

[36] A. Panchenko, A. Mitseva, M. Henze, F. Lanze, K. Wehrle, and T. Engel, "Analysis of Fingerprinting Techniques for Tor Hidden Services," in *Proceedings of the 16th Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2017.

[37] A. Sehgal, V. Perelman, S. Kuryla, and J. Schonwalder, "Management of resource constrained devices in the internet of things," *IEEE Communications Magazine*, vol. 50, no. 12, 2012.

[38] K. Loesing, "Measuring the Tor Network from Public Directory Information," in *Proceedings on Hot Topics in Privacy Enhancing Technologies (HoTPETS)*, 2009.

[39] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The Internet of Things Has a Gateway Problem," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile)*. ACM, 2015.

[40] "Zigbee Gateway," https://zigbee.org/zigbee-for-developers/zigbee-gateway/, accessed on August 19, 2019.

[41] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, 2012.

[42] M. Henze, B. Wolters, R. Matzutt, T. Zimmermann, and K. Wehrle, "Distributed Configuration, Authorization and Management in the Cloud-based Internet of Things," in *Proceedings of the 2017 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2017.

[43] R. Dingledine, "Improving Tor's anonymity by changing guard parameters," https://blog.torproject.org/improving-tors-anonymity-changing-guard-parameters, 2013, accessed on August 16, 2019.

[44] "Tor Protocol Specification," https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt, 2019.

[45] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, 1976.

[46] Contiki-NG, "Contiki-NG: The OS for Next Generation IoT Devices," https://github.com/contiki-ng/contiki-ng, 2017.

[47] Eclipse Foundation, "Eclipse tinydtls," https://projects.eclipse.org/projects/iot.tinydtls, 2016.

[48] Contiki-NG, "Contiki-NG: tinyDTLS," https://github.com/contiki-ng/tinydtls, 2017.

[49] K. Krauss, "Ethical Tor Research: Guidelines," https://blog.torproject.org/ethical-tor-research-guidelines, 2015, accessed on August 16, 2019.

[50] "Tor Research Safety Board," https://research.torproject.org/safetyboard.html, accessed on August 16, 2019.

[51] F. Cangialosi, D. Levin, and N. Spring, "Ting: Measuring and exploiting latencies between all tor nodes," in *Proceedings of the 2015 Internet Measurement Conference (IMC)*. ACM, 2015.

[52] "Tor Metrics," https://metrics.torproject.org/.

[53] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, "Website Fingerprinting at Internet Scale," in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2016.

[54] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective Attacks and Provable Defenses for Website Fingerprinting," in *Proceedings of the 23rd USENIX Conference on Security Symposium*. USENIX Association, 2014, pp. 143–157.

[55] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a Distance: Website Fingerprinting Attacks and Defenses," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*. ACM, 2012.

[56] J. Pennekamp, J. Hiller, S. Reuter, W. De la Cadena, A. Mitseva, M. Henze, T. Engel, K. Wehrle, and A. Panchenko, "Multipathing Traffic to Reduce Entry Node Exposure in Onion Routing," in *Proceedings of the 27th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2019.

[64] C. Fachkha, E. Bou-Harb, A. Keliris, N. Memon, and M. Ahamad, "Internet-scale Probing of CPS: Inference, Characterization and Orchestration Analysis," in *Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2017.

[57] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg, "Changing of the guards: A framework for understanding and improving entry guard selection in Tor," in *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2012.

[58] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2013.

[59] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on Tor by realistic adversaries," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2013.

[60] H. Shafagh, L. Burkhalter, S. Duquennoy, A. Hithnawi, and S. Ratnasamy, "Droplet: Decentralized Authorization for IoT Data Streams," arXiv preprint arXiv:1806.02057 [cs.CR], 2018.

[61] S. Raza, L. Seitz, D. Sitenkov, and G. Selander, "S3K: Scalable Security With Symmetric Keys – DTLS Key Establishment for the Internet of Things," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 3, 2016.

[62] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "DTLS based Security and Two-Way Authentication for the Internet of Things," *Ad Hoc Networks*, vol. 11, no. 8, 2013.

[63] J. Kuusijärvi, R. Savola, P. Savolainen, and A. Evesti, "Mitigating IoT Security Threats with a Trusted Network Element," in *Proceedings of the 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2016.

[65] N. Apthorpe, D. Reisman, and N. Feamster, "A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic," in *Workshop on Data and Algorithmic Transparency (DAT)*, 2016.

[66] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic," arXiv preprint arXiv:1708.05044 [cs.CR], 2017.

[67] Adafruit Industries, "Onion Pi," https://learn.adafruit.com/onion-pi/overview, 2013, accessed on August 19, 2019.

[68] L. Davoli, Y. Protskaya, and L. Veltri, "An Anonymization Protocol for the Internet of Things," in *Proceedings of the 2017 International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, 2017.

[69] F. W. Baumann, U. Odefey, S. Hudert, M. Falkenthal, and U. Breitenbücher, "Utilising the Tor Network for IoT Addressing and Connectivity," in *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER)*. SciTePress, 2018.

[70] J. Reardon and I. Goldberg, "Improving Tor using a TCP-over-DTLS tunnel," in *Proceedings of the 18th USENIX Conference on Security Symposium*. USENIX Association, 2009.