Hide Your Hackable Smart Home from Remote Attacks: The Multipath Onion IoT Gateways

Lei Yang[1], Chris Seasholtz[2], Bo Luo[2], and Fengjun Li[2(✉)]

[1] Amazon LLC., Seattle, WA, USA
yanglei@amazon.com
[2] The University of Kansas, Lawrence, KS, USA
{seasholtz,bluo,fli}@ku.edu

Abstract. The rapid expansion of IoT-enabled home automation is accompanied by substantial security and privacy risks. A large number of real-world security incidents exploiting various device vulnerabilities have been revealed. The *Onion IoT gateways* have been proposed to provide strong security protection for potentially vulnerable IoT devices by hiding them behind IoT gateways running the Tor hidden services, in which the gateways can only be accessed by authorized users with the *.onion* addresses of the gateways and correct credentials. However, the limited bandwidth of Tor makes this approach very impractical and unscalable. To tackle this issue, we present two novel designs of *multipath Onion IoT gateway* and *split channel Onion IoT gateway*. The first design implements a customized multipath routing protocol in Tor to construct a multi-circuit anonymous tunnel between the user and the Onion gateway to support applications that require low latency and high bandwidth. The second scheme splits command and data channels so that small-sized command packets are transmitted through the more secure channel over the Tor hidden service, while the less secure data channel over the public network is used for outbound very-high-bandwidth data traffic. Experiment results show that the proposed approaches significantly improve the performance of Onion IoT gateways, so that they can be practically adopted to securely transmit low-latency and high-bandwidth data, such as HD video streams from home surveillance cameras. We also prove the security guarantees of the proposed mechanism through security analysis.

Keywords: IoT security · Smart homes · Tor hidden service

1 Introduction

By connecting billions of smart devices to the Internet, the Internet-of-Things leads to a pervasive deployment of intelligence into our daily life with innovative applications. In a recent estimation, approximately 8.4 billion IoT devices are connected to the Internet worldwide in 2017 – a 31% increase from 2016. By

J. Lopez et al. (Eds.): ESORICS 2018, LNCS 11098, pp. 575–594, 2018.
https://doi.org/10.1007/978-3-319-99073-6_28

2020, the number of connected device will reach 20.4 billion, resulting in a global market of $2 trillion [22]. One of the fastest growing IoT fields is smart home systems, sometimes referred as *home automation*, in which smart appliances such as baby monitors, security cameras, smoke alarms, smart locks, smart lights, and smart switches/plugs are connected to the home network and remotely controllable through the Internet. Beyond convenience, the smart home technology also provides tangible benefits such as safety and energy-efficiency.

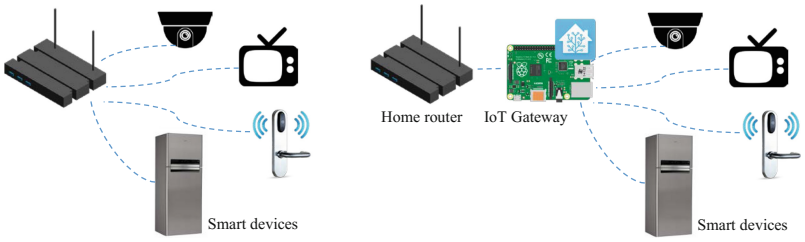
While we are witnessing a rapid expansion of IoT-enabled home automation, the increasing use of the networked IoT devices is accompanied by substantial security and privacy risks [29,31,35], which in some cases could lead to chilling safety consequences since the smart devices in home automation are monitoring our personal activities at home. For example, burglars can hack into our surveillance system [13] or analyze our electricity consumption [43] to observe our life pattern, and get into our homes with the help of our smart locks [18]. To make things worse, the compromised devices can be turned into bots to launch a DDoS attack. For example, the Mirai botnet compromising millions of cameras and digital video recorders took down the Dyn DNS servers in 2016 and caused a massive Internet outage as well as up to $110 million economic loss [17].

As security and privacy has become a most important consideration in the design and implementation of the smart home technology, various security solutions have been proposed to secure light-weight IoT communication protocols (e.g., DTLS [27,32] for RPL [38], 6LoWPAN [33] and CoAP [34]), enhance authentication [28,30] and access control [25,26,36], attest operational status of remote devices and detect intrusions, etc. However, over the recent years, a large number of real-world attack incidents have been revealed by academia, security firms and individual researchers, which have exploited various types of vulnerabilities in consumer IoT devices and applications involving the use of surveillance cameras [4,13] and baby cameras [1], smart locks and garage openers [18], smart appliances [2,9], thermostats [3], plugs and light bulbs [29], etc.

A root cause of these vulnerabilities is that the manufacturers have been lax in adopting appropriate or even basic security measures. For example, the D-LINK DCS2132L Internet cameras require no credential to access the management interface [4,35], and the WeMo devices allow mobile Apps to access them through an unencrypted SOAP API [31]. The lack of security protection is due to several reasons. First, it is difficult to extend conventional security schemes to IoT devices that are usually resource-constrained. Moreover, implementing security measures on IoT devices especially on the low-end ones requires skills and resources, and thus increases design and development costs. Finally, manufacturers are under business pressures to hit the market so that security is not their priority.

Consider the large number of heterogeneous IoT devices, manufacturers' general lack of incentives to adopt appropriate security practices, and the slow progress in IoT security standardization/regulation, it is difficult, if not completely impossible, to develop security solutions for each individual device, nor to force each device vendor to ensure a flawless implementation or adopt adequate

security protections. Recognizing the fact that security vulnerabilities always exist in IoT devices, the question we pose is *in what strategy the chance of adversaries attacking vulnerable devices can be reduced and where this protection should be deployed*?



(a) Direct access model (b) Gateway-based access model

Fig. 1. Home automation operational models

This naturally leads to an isolation-based approach that uses a dedicated IoT gateway to separate the private network, in which the IoT devices are deployed, from the public network, and secure the perimeter of the private network at the gateway. As shown in Fig. 1b, the IoT gateway coordinates the connected home automation devices and isolates them from direct access from the public Internet. Open-source platforms such as *Home Assistant* [5] and *open Home Automation Bus* (openHAB) [8] are introduced to support the interconnection of devices in different types and from different manufacturers. In this work, we developed our secure IoT gateways on the Home Assistant platform, but our design can be easily extended to other platforms. Since all devices are managed and controlled through the gateway, individual devices no longer offer interfaces for remote control and thus are not directly exposed to remote adversaries. However, the gateway, which may have its own security vulnerabilities, becomes the new target of interest to the adversaries, and also the single point of failure.

One approach to secure the IoT gateway is to connect it to a back-end cloud server that relays all the commands from the cloud so that it can utilize the existing security mechanisms provided by the cloud. Samsung's SmartThings [11], Apple's HomeKit and Google's Brillo are several examples. However, the cloud-based approach needs to store IoT data on the cloud and thus yields a serious privacy issue if the cloud service provider is not fully trusted to view our private IoT data. In fact, users have expressed serious security and privacy concerns due to data breaches and various types of data abuses [20].

Therefore, we propose to integrate the Tor hidden service onto the potentially vulnerable IoT gateway so that it is protected from being directly exposed to remote adversaries. This is because in most network attacks, a critical step is to identify vulnerable, Internet-facing nodes through reconnaissance. By hiding the gateway behind the Tor network, the adversaries, without knowing the

gateway's *.onion* address, cannot directly scan or access the gateway. In this way, the Tor hidden service acts as an additional security buffer between the smart home applications and the adversaries. This idea was first introduced by Nathan Freitas from the Tor Project in [21], which described a straightforward approach of obtaining an *.onion* address for the IoT gateway and running the Tor hidden service directly on it. However, in practice, this approach suffer from a well-known performance problem of the Tor network, in which Tor users often experience very high delays [41,42]. The poor performance of the IoT gateway running the Tor hidden service affects any IoT applications with realtime requirements. Moreover, to prevent congestion, Tor actively throttles high-bandwidth applications. Consequently, IoT gateways with high-bandwidth services such as video streaming will be blocked.

To tackle the performance problem, we propose two novel and practical designs of multi-path Onion IoT gateways, namely *IoT gateway over multipath Tor hidden services*, and *IoT gateway over split channels*. Both designs provide strong security protection by hiding the IoT devices behind the gateway running the Tor hidden services. In the first solution, we extended the multipath routing protocol *mTor* [42] and customized it to construct an end-to-end tunnel consisting of multiple Tor circuits between the user and the proposed Onion IoT gateway. By applying a self-adaptive scheduling scheme, the tunnel transfers data over multiple circuits efficiently and thus achieves a good throughput to support IoT applications that require low network latency. Since the traffic is routed through the anonymous tunnel, this scheme provides a same security protection as the original Tor-based approach. Therefore, it fits the user who requires strong security protection and a reasonable performance.

We further improved the performance of the Onion gateway in our second design by using split channels for command and data transmission. In particular, the IoT gateway running the Tor hidden service maintains two separate channels: the command channel handles requests and responses, which are tiny messages, through the hidden service ports over the Tor network, and the data channel is only used to send high-bandwidth traffic to remote users over the public network. By separating the command and data channels, we provide a good security protection by hiding the security-sensitive command interface behind Tor, while avoiding injecting high-bandwidth traffic into the Tor network.

The main contributions of this work are as follows:

- We present a general security solution to safeguard IoT devices with potential security vulnerabilities by hiding them behind the specially designed IoT gateways running the Tor hidden services.
- We propose two novel designs of Onion IoT gateways to provide strong security protection by integrating the Tor hidden services on the IoT gateway with optimized performance to support high-bandwidth and low-latency IoT applications.
- To our best knowledge, the proposed Onion IoT gateway design is the first practical solution to integrate Tor hidden service with secure IoT gateways.

The rest of this paper is organized as follows. We first introduce the preliminaries in Sect. 2, and then present our IoT gateway designs, namely *multi-path Onion IoT gateway* and *split channel Onion IoT gateway*, in Sects. 3 and 4, respectively. We evaluate the performance of the proposed designs through experiments in Sect. 5 and analyze their security in Sect. 6. Finally, we discuss the related work in Sect. 7 and conclude this work in Sect. 8.

2 Preliminaries

2.1 One Instance of Smart Home Gateway: Home Assistant

Home Assistant (HA) is an open-source home automation platform running on Python 3, which is able to automatically discover, monitor, control and automate various consumer IoT devices [5]. It can run on major operating systems (e.g., Linux, Windows, OS X) and hardware modules ranging from PCs to micro-controllers such as Raspberry Pi. Home Assistant offers a web interface and allows users to remotely access it through web browsers or mobile applications.

We choose Home Assistant as our implementation platform for several reasons. First, HA is an open-source platform supporting the major brands of IoT devices, and thus is widely used in home automation application development. It also has good community supports. Moreover, Freitas implemented the Tor-based gateway on HA [21], so it is fair to compare his scheme with ours on the same platform. It worths noting that our designs do not rely on the HA platform. In particular, the first design using the customized multipath Tor routing protocol is platform-independent, and the second design can be easily implemented in other platforms with a small effort.

2.2 Tor and Tor Hidden Service

The Tor network [19] is an overlay network consisting of Onion Routers (ORs) contributed by volunteers to support anonymous communication over the Internet. To do so, the client's proxy, known as the Onion Proxy (OP), randomly selects three routers to establish a Tor circuit to the destination. It then encrypts the data in layers, packs them into 512-byte cells and sends data cells through the circuit. Each router along the circuit peels off one layer of encryption and forwards the cell to the next router until it reaches the last relay (known as "exit"), which further forwards the data to the original destination. Each hop only knows who has sent the data (predecessor) and to whom it is relaying (successor) due to the layered encryption.

Tor hidden services use rendezvous points (RPs) to allow service operators to offer TCP-based services, such as web or instant messaging servers, without revealing their real IP addresses. Service operators can enable it by setting up Tor as the proxy for their services. Figure 2 illustrates the basic components of Tor hidden services: (1) The hidden server (HS) randomly selects several routers as its introduction points (IPs) and builds onion circuits to them. (2)

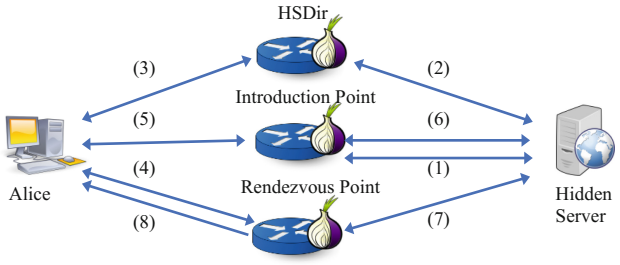


Fig. 2. Tor hidden services architecture

HS uploads its service descriptor to the hidden service directory (HSDir), where the descriptor along with HS's public key and the set of IPs is signed by HS's private key. Now, HS is ready to accept connections from clients. (3) To connect to the hidden service, a client (e.g., Alice) contacts HSDir to retrieve the service descriptor of HS using its onion address, which Alice learns out of band. (4) With the set of IPs and HS's public key from the service descriptor, Alice randomly selects a router as her RP, gives it a rendezvous cookie (RC) which is a one-time secret, and builds a circuit to it. (5) Alice sends an introduce message to one of the IPs and (6) asks it to forward the message to HS, which contains the rendezvous cookie, RP address and the first part of a Diffie-Hellman (DH) handshake encrypted by HS's public key. (7) After decrypting the introduce message, HS establishes a new circuit to Alice's RP and sends a rendezvous cell to it, containing RC and the second part of DH handshake. (8) RP relays the rendezvous cell to Alice. (9) After verifying RC and generating the end-to-end session key, Alice and HS start communicating through RP, which relays data cells between the two circuits without change.

In our design, the IoT gateway built on the Home Assistant platform is running Tor hidden services, so it can only be accessed by its .onion address with an optional authentication token shared between authorized users. By applying multiple-hop onion routing and the end-to-end encryption, Tor hidden services provide strong protection to traffic flows and the location of the hidden server. This prevents the remote adversaries from knowing the IP address of the IoT gateway by probing or scanning the network, or even the existence of the IoT gateways, and thus reduces the risks of remote exploitation.

Flow Control. Tor uses a two-layer window-based end-to-end flow control scheme to guarantee a steady flow between two ends. Since multiple streams multiplex a circuit, the outer layer performs a circuit-level control which restricts the number of cells transmitted over a circuit for all streams. The inner layer enforces a stream-level control for individual streams. At both ends of a circuit, two OPs (one for sender and one for receiver) control the speed of data cells entering and leaving the circuit by keeping track of the circuit and stream windows. By default, a circuit window starts with 1000 cells and a stream window is initialized to 500 cells. When a data cell is sent, both windows decrease by

one. When a stream window becomes empty, the sender stops sending from this stream; when a circuit window reaches zero, the sender stops sending from all streams on this circuit. Windows are increased when the corresponding acknowledgment cell known as SENDME is received. For every 100 cells received on a circuit, the receiver sends a circuit SENDME to inform the sender to forward another 100 cells from this circuit. For every 50 cells received from a stream in this circuit, the receiver sends a stream SENDME to request another 50 cells from this stream.

The Performance Problem of Tor. There are about 7,000 onion routers in the Tor network, among which the majority is low-bandwidth relays. So, the donated bandwidth resource is relatively scarce comparing to the large user scale (i.e., almost 2.5 million users per day). Besides, due to the current path selection scheme, many users tend to select relays from a very small set of high-bandwidth relays when constructing the circuits, which causes frequent congestions on these relays. When congestion happens, a congested relay in a 3-hop circuit will greatly degrade the performance of the entire circuit. Hence, the problem becomes worse in hidden services, in which the circuit connecting the user to the hidden server contains six relays. In [21], Freitas proposed to directly deploy Home Assistant over the Tor hidden service, therefore, this scheme inevitably suffers the same performance problem stated above.

3 IoT Gateway over Multipath Tor Hidden Services

To overcome the performance problem of the current deployment of Tor for IoT gateway, we extend the *m*Tor approach in [42], and customize it into an end-to-end multipath routing scheme to support Tor hidden services, namely *m*TorHS.

As illustrated in Fig. 3, *m*TorHS constructs an anonymous *tunnel* consisting of m circuits, where m is a client-specified parameter. While the capacity of each circuit is dynamic over time, our proposed *m*TorHS scheme can adaptively distribute traffic onto m circuits in proportion to their dynamic capacities, and thus avoid the communication being blocked by a congested circuit and achieve an optimal overall performance. *m*TorHS is transparent to the Tor network, that is, no modification needs to be made on regular Tor relays. Only the two Tor Onion Proxies (OPs) on the user side (for users who choose to use *m*TorHS) and the hidden server side (i.e., Tor OP for Home Assistant) need to be updated. In particular, new functions are added for associating multiple circuits to a client stream, adding sequence number to data cell, reordering out-of-sequence cells, and scheduling cells across multiple circuits. Next, we will elaborate the process of tunnel construction and data transmission. For the ease of presentation, we denote user's Onion Proxy as OP or Alice interchangeably, and call hidden server's OP as HS in the following sections.

3.1 Tunnel Construction

In our scheme, the server establishes hidden service and client retrieves service descriptors in the same way as the conventional Tor hidden service (i.e., *step 3–6* in Sect. 2.2). Our modification starts from *step 4*.

Tunnel Initialization. Different from the current scheme which randomly selects one router as the rendezvous point (RP), the user Alice chooses m routers and constructs m circuits of 3 hops, each ending at a distinct router. Then, Alice gives m different rendezvous cookies (RC) to the RPs, which will be used to link the joining circuits established from the hidden server. We denote the first established circuit as the *primary circuit* and the other $m-1$ circuits as the *auxiliary circuits*. Once m circuits are established, Alice does the same thing as *steps 5–6* in Sect. 2.2. In particular, Alice sends an introduce1 message to an introduction point, which will forward it to the hidden server with an introduce2 message. The message contains the rendezvous cookie, RP address and the first part of a DH handshake for the primary circuit. We add two new fields to this message, namely, *is_multipath* and *tunnel_width*, which indicate the request is to build a multipath tunnel with tunnel width m. After receiving the introduce2 message, HS checks if *is_multipath* is set. If so, HS generates a unique 32-bit tunnel identifier (TID); otherwise, HS follows the original Tor protocol. HS establishes a new circuit to the RP of the primary circuit and sends Alice a rendezvous1 cell containing RC, the second part of DH handshake, and TID. RP relays the content of the rendezvous1 cell to Alice with a rendezvous2 cell. Once Alice receives it and successfully verifies RC, she extracts TID and generates the end-to-end session key. A 6-hop circuit is established between Alice and HS. Now, Alice and HS can communicate with each other through the primary circuit. It is worth noting that while two 3-hop anonymous circuits (between client and RP and between server and RP) join at the RP conceptually, we re-design the entire circuit construction protocol to enable end-to-end encryption between the client and the server so that no intermediate router can observe the clear traffic in Tor.

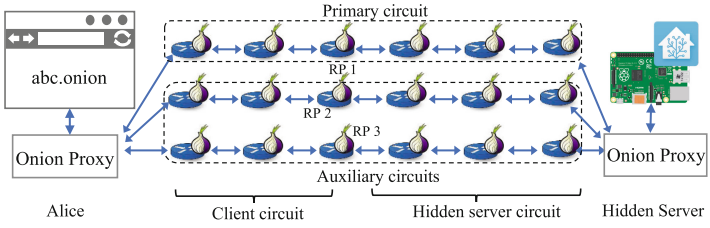


Fig. 3. An example of mTorHS architecture where $m = 3$.

With TID, Alice adds the remaining auxiliary circuits to the tunnel by sending $m-1$ next_rp_m[1] messages to HS along the primary circuit. The format of

[1] To distinguish from the commands in current Tor, all the newly added commands in mTorHS have a suffix m.

each next_rp_m message is similar to the introduce1 message, which contains TID and RP's address used in auxiliary circuits. In response to the next_rp_m message, HS builds a new circuit connecting to the corresponding RP, and acknowledges each successful joining with a rendezvous1 to that RP. HS associates all circuits with the same TID to form a tunnel for Alice. After Alice receives all m rendezvous2 cells including 1 cell from the primary circuits and $m-1$ cells from the auxiliary circuits, a multipath tunnel is successfully constructed.

Tunnel Management. Atop circuit management of Tor, mTorHS introduces additional tunnel management to oversee circuits in the tunnel. mTorHS manages the multipath tunnel dynamically according to the congestion status of member circuits over time. If OP detects that the transmission on a member circuit becomes very slow, OP will construct a new circuit to replace it (will be elaborated in the next subsection). The slow circuit closing scheme provides OP the ability of responding to real-time network dynamics, and prevents a slow circuit from becoming a bottleneck of the entire tunnel. In particular, OP can add new auxiliary circuits or tear down any existing circuit at any time. In particular, a new auxiliary circuit can be added by sending a next_rp_m command to inform HS of the new RP address. To tear down a circuit, OP informs HS to drop it using a drop_m message. After receiving drop_m cell, HS immediately stops sending on this circuit and responds to OP using a dropped_m message with the number of cells that have already been sent on this circuit (denoted as n_s). Once OP receives n_s cells on this circuit, it terminates the circuit.

3.2 Data Transmission

When Alice's data stream arrives at OP via SOCKS, OP spawns the client stream (denoted as the *parent* stream) to m subflows and appends them to the tunnel by associating each subflow with a circuit. Each subflow has its own stream window and inherits a common stream ID from the parent. Next, OP sends a relay_begin cell through a random member circuit to start the access.

Scheduling and Data Cell Allocation. Conceptually, data cells can be forwarded through any member circuit in the tunnel. However, if the number of allocated cells on a particular circuit exceeds its capacity, it will become congested. Since the overall performance of a tunnel is bounded by the slowest circuit, two endpoints of a tunnel need to cooperate to schedule cells across multiple circuits based on the capacities of individual circuits. A naive approach for cell allocation is to probe the capacity of each circuit after it is initiated and schedule traffic according to the probed capacity.

However, the method is problematic in practice. First, it will introduce a large amount of probing traffic to the Tor network. Moreover, the capacity of each circuit may change dramatically after probing. Therefore, the static scheduling scheme cannot adapt to network dynamics so that it is ineffective. In [14], Alsabah et al. presented an opportunistic probing approach to estimate the round-trip-time (RTT) of a circuit based on Tor's circuit-level congestion control scheme. The RTT-based approach is reactive to network dynamics, but

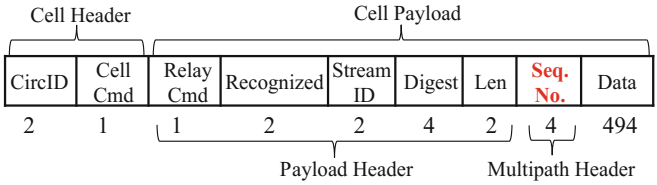


Fig. 4. *m*TorHS cell format: a new filed, *sequence number* marked in red, is added as the multipath header, representing the sequence number of sent-out cells. (Color figure online)

it is still not very accurate because the congestion feedbacks are received infrequently [15]. We argue that RTT-based approach may not be a good choice in cross-layer scheduling, which is also recognized in multipath TCP design [16].

In *m*TorHS, we adopt a "pulling" scheduling scheme – instead of pushing data cells to circuits by a scheduler, we let each subflow actively pull data from a shared send buffer, whenever its stream window becomes nonempty. Initially, each subflow has a stream window of 500 cells. As described in Sect. 2.2, the stream window decreases by one when sending a cell out and increases by 50 when receiving a stream-level SENDME. Consequently, a subflow stops sending cells when its stream window size drops to zero and resumes when it receives a SENDME. When the circuit to which a subflow is appended becomes congested, cells will be moving much slower towards the receiver, resulting in delayed stream-level SENDMEs and long waiting at the sender end. Whereas, subflows on fast circuits will send out data cells fast and steadily. In this way, the "pulling" scheduling is subflow self-adaptive without accurate explicit circuit RTT measurements. When multiple subflows have a nonzero stream window, we adopt a FIFO (first-in-first-out) queue to schedule them.

Slow Circuit Detection. Another challenge in *m*TorHS design is the detection of slow circuits. To avoid a congested circuit becoming the bottleneck of the entire tunnel, OP will replace slow circuits with new ones. We adopt a distance-based outlier detection approach to determine whether a circuit is congested based on a sliding window of 50 cells. In particular, we measure the time of receiving every 50 cells and find the lower and upper quartiles (Q_1 and Q_3) of ten most recent records to calculate the interquartile range (IQR) where $IQR = Q_3 - Q_1$. If a new measurement falls out of the range of $(0, Q_3 + 1.5IQD]$, it is considered as an outlier indicating the circuit is experiencing a congestion. To increase detection reliability, OP considers a circuit as congested if at least three consecutive outliers occur. Once detected, OP and HS will collaborate to tear down the congested circuit and replace it with a new one. This can be done through the tunnel management discussed above.

Data Re-ordering. Combining the self-adaptive "pulling" scheduling and active congestion detection schemes, *m*TorHS is able to adapt to network dynamics, which potentially prevents slow circuits from degrading the overall

performance of multipath tunnel. However, due to dynamic scheduling, data cells may arrive at the receiver out of order. To solve this issue, we have to modify the format of Tor data cell to incorporate a 32-bit sequence number in the multipath data packets. As shown in Fig. 4, the first four bytes of data payload is reserved for this purpose. Moreover, we add a new *relay_subdata_m* command to indicate a data cell is multipath data. When OP receives a multipath data cell from a subflow, it first checks if the sequence number is expected. An expected cell is immediately forwarded to the application stream, while an out-of-order cell is stored in a buffer and ordered according to its sequence numbers.

3.3 Discussions

By applying congestion detection, users of multipath hidden services can route traffic through multiple circuits (some may be lightly occupied), and thus improve the overall performance. Such a congestion avoidance scheme also benefits single-path users, since *m*TorHS stops using the congested paths to give bandwidth to others' traffic. To further balance the usage on high-bandwidth relays between multipath users and general users, we can force multipath users to use the low-bandwidth relays to establish their tunnel and still achieving an acceptable performance, since most Tor relays are low-bandwidth and they are under-utilized in the current Tor [42]. We can bundle these idle, low-bandwidth relays to effectively serve the multipath users without hurting general users. In terms of security, this solution, namely IoT gateway over multipath Tor hidden services, transmits all traffic, including incoming and outgoing traffic, on gateway through Tor network, so the IP address of gateway is still hidden from the public Internet and thus the adversary cannot scan and attack the gateway.

*m*TorHS improves the network utilization by employing low-bandwidth relays. However, it will not increase the overall bandwidth of Tor network. If millions of users access their gateways via Tor, especially for bulk traffic like watching camera videos, the huge demand on bandwidth may exceed the capacity of Tor. Therefore, IoT gateway over *m*TorHS is best for users who have very strong security requirement but only need an acceptable performance. For the majority users who want to achieve a better balance between security and performance, we propose an alternative solution, namely *IoT gateway with split command and data channels*, which will be presented in next section.

4 IoT Gateway with Split Command and Data Channels

Tor provides very strong security protection but limited bandwidth, while the public Internet has the opposite – high bandwidth but weak security protection. To combine the advantages, we propose a novel scheme, namely *IoT gateway with split command and data channels*, which leverages the security of Tor and the good performance of the public Internet. More specifically, the IoT gateway only accepts incoming traffic from the Tor channel, while responding to the remote client with encrypted (data) traffic through the Internet channel.

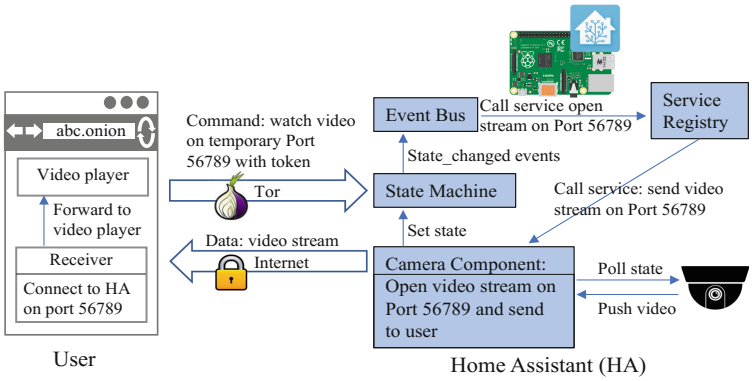


Fig. 5. An example of IoT gateway with split command and data channels: user sends request to watch camera video on port 56789 through the Tor channel, while Home Assistant temporarily opens port 56789 to deliver encrypted video through Internet.

With this scheme, the IoT gateway can still defend against vulnerability scanning by refusing to respond to any request coming from the Internet, such as ICMP ping, telnet, or HTTP request. The adversary obtains nothing by scanning the gateway with IP address. Besides, since the onion address of hidden service is only known by the user himself, the adversary cannot scan the gateway through Tor. Therefore, the security of the IoT gateway is still well protected. From the efficiency perspective, the incoming command traffic to the IoT gateway is usually transient and very small, so transmitting the commands through Tor will not introduce much overhead to Tor. In response to the command coming from the Tor channel, the gateway sends out bulk traffic, such as camera videos, to the user through the Internet channel. Since the video stream is transfered through the public Internet, we need to ensure strong encryption and mutual authentication between the user and the IoT gateway.

An overview of the protocol is shown in Fig. 5. Next, we deliberate this protocol with an example in which a user remotely requests video stream from a camera behind the IoT gateway.

Step 1: Connection Initialization Through Tor. The user connects to the IoT gateway, such as Home Assistant, using its onion address (e.g., abc.onion). At the client side, the user selects a device (e.g., a camera) and specifies a random high TCP port p_d for the data stream. Meanwhile, a 256-bit random token r_A, a 1024-bit random number x and its corresponding g^x (for Diffie-Hellman) will be generated. The random token r_A will be used for the later mutual authentication, and g^x will be used to generate the session key. Then, user submits this configuration to IoT gateway (e.g., Home Assistant) through Tor.

Step 2: Service Initialization at IoT Gateway. In a conventional IoT gateway, data will be disseminated to the user through the same channel as the

incoming request. For example, video from the camera will be sent back to the user on the web interface, transmitted through Tor, if the request comes from Tor HS. In our scheme, we redesign the interfaces of the data-intensive devices, such as the Camera Component, so that the data stream is re-directed to a temporary port specified by the user, and transmitted to the user through the public Internet channel. To do so, we add a new functionality to the Camera Component and register this service (i.e., camera through Internet) to the Service Registry. Since this new service is implemented at the component level, it can work with different types of cameras seamlessly.

When Home Assistant receives user's request from the Tor channel, it generates a 256-bit random token r_{HS}, a 1024-bit random number y and its corresponding Diffie-Hellman number g^y. HA uses the received g^x and y to generate the session key g^{xy} and responds to user with the random token r_{HS} through the Tor channel. Meanwhile, HA opens port p_d, which is specified by the user, and waits for the connection for data dissemination, e.g., video streaming.

Step 3: Initialization of Data Channel Through Internet. After user receives the token r_{HS} and the second half of Diffie-Hellman handshake g^y through Tor, he will generate the session key g^{xy}, and send a request containing r_{HS} to Home Assistant through the data channel, which is encrypted with g^{xy}.

Step 4: Data Dissemination Through Internet. Once user's request for video stream arrives at port p_d through the Internet channel, Home Assistant decrypts it with the session key and verifies the token r_{HS}. If the authentication succeeds, Home Assistant sends r_A followed by the subsequent data to the user. The communication is encrypted with the session key. Meanwhile, HA discards requests that fail the authentication. Finally, when the user receives the reply, he decrypts it to get r_A to verify the server. If the authentication succeeds, he will accept the subsequent data stream. Otherwise, the connection will be closed.

5 Experiment and Performance Evaluation

To demonstrate the performance improvement of our proposed schemes, we implement all three Tor-based IoT gateway approaches, and perform experiments on the live Tor network. In particular, we compare network throughput for video streaming in the following settings: (1) the original Home Assistant without Tor hidden services (denoted by *HA-No-Tor*), (2) HA with single-path Tor hidden services (*HA-sTorHS*), (3) HA with multipath Tor hidden services (*HA-mTorHS*) and (4) HA with split command and data channel (*HA-Split*).

Setup. We deployed Home Assistant on a dedicated Raspberry Pi 2 [10] to simulate the proposed IoT gateway, which connects to a VStarCam IP camera [12]. The video stream is fed into Home Assistant via FFmpeg. The Raspberry Pi 2 is equipped with a 700 MHz ARM A6 microprocessor and 512 MB of RAM. The client accesses Home Assistant through a laptop with 2.5 GHz Intel i5 CPU, 8 GB RAM and OSX.

Experiments. To compute the throughput, we measure the overall time for the client to receive a 10 MB streaming data from Home Assistant. For *HA-Split*, we modify Home Assistant using Python 3 on the backend and Polymer on the frontend. For *HA-mTorHS*, we change the source code of Tor-v0.2.9.10. Note that all changes are made to the client's proxy and hidden server's proxy, so no change is needed on the Tor network. We compare two different multipath settings where the tunnel width m is set to 2 and 4, respectively. To eliminate the difference caused by circuits' capacities, we let the *HA-4TorHS* scheme to use the default path selection algorithm to choose relays for 4 circuits and record the used relays. Then, we let the *HA-2TorHS* scheme randomly choose 2 out of 4 circuits, while *HA-sTorHS* randomly chooses 1 from the 4 circuits. Each experiment is repeated 60 times over different time of a day, and the average transmission time for each approach is recorded.

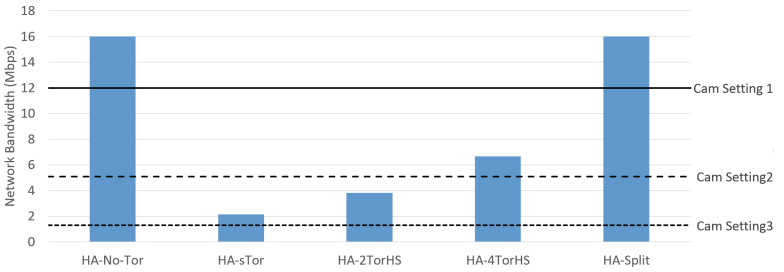


Fig. 6. Average network throughput provided by various approaches, compared with throughput needs of household IoT devices: Setting 1: two full HD (1280 × 1080) surveillance cameras, 30 fps, H.264 high quality compression; Setting 2: two HD 720p (1280 × 720) cameras, 30 fps, H.264 high quality compression; Setting 3: two HD 720p cameras, 15 fps, H.264 medium quality compression.

Results. Figure 6 compares the performance of different schemes. The Y-axis is the throughput measured from transmitting 10 MB video from the IoT camera. We can see that the baseline approach (HA-*s*Tor) that directly integrates Tor hidden services into Home Assistant achieves the worst performance, which is almost 7 times slower than the direct access to HA through the public Internet (HA-No-Tor).

Figure 6 also shows that the proposed multipath Tor hidden services can improve the performance significantly. In particular, the *HA-2TorHS* scheme adopting two multipath circuits is 1.7 times faster than *HA-sTorHS*, while *HA-4TorHS* is 3 times faster. In fact, *HA-4TorHS* is fast enough for a typical home surveillance setting with two HD 720p (1280 × 720) cameras capturing at 30 fps with H.264 high quality compression.

Lastly, *HA-Split* can achieve the same performance as accessing Home Assistant without using Tor with a comparably weaker security guarantee. More specifically, it provides the same security guarantee for the command channel

and the same anti-scanning feature for the IoT devices and HA. In summary, both *HA-mTorHS* (with $m > 3$) and *HA-Split* schemes can achieve an acceptable performance with enhanced security as we expect. We recommend users with larger throughput requirement to adopt the *HA-Split* scheme, which also avoids overloading the Tor network with a large amount of IoT data. While for users with higher security requirements, we recommend using the *HA-mTorHS* scheme, which hide both the IoT devices and the client behind Tor hidden service with a strong anonymity protection.

As we have discussed, a single Tor path can (and often) get congested. *m*TorHS with slow circuit detection and congestion control overcomes this problem. To demonstrate this, we conduct another experiment – a 6-hop path between the client and the hidden server is established using very fast relays measured by Tor, i.e., 3 relays used by client are 2391A, 92CFD and 3E13E, while 3 relay used by client are A7047, 96DAF and 8C23B. They are respectively Top 1%, 5%, 14%, 30%, 11% and 20% fastest relays among all Tor routers at the time of experiment (July 12, 2017). Unfortunately, the real throughput of this path consisting of very fast relays was lower than average: it takes 55 s to transmit the 10 MB video file. This poor performance is usually due to congestion on at least one of relays in the path [37]. Then, for *HA-2TorHS*, we keep the first path as is, and add another path with congestion control. The performance is improved to 16 s for transmitting a 10 MB file, since most traffic can be routed through the second path, which may also alleviate the congestion on the first path. For *HA-4TorHS*, we add 2 more paths with congestion control, and the performance is further improved to 10 s.

6 Security Analysis

In this section, we analyze the security of two proposed schemes in terms of authentication, encryption and anti-scanning.

Authentication. Home Assistant provides optional password-based authentication, but it is not required. Besides, such an authentication approach has several known weaknesses such as weak passwords, which is a commonly observed in many use cases. Moreover, password-based authentication is particularly vulnerable if adversaries are allowed unlimited attempts when guessing the password. As a result, the embedded authentication of Home Assistant is not reliable.

To tackle this problem, our proposed schemes offer two additional layers of authentication provided by Tor hidden services. First, adversaries cannot access Home Assistant over hidden services without knowing the onion address of the device behind the gateway. Since the onion address generated by Tor is an 80-bit number in base32, it is not easy for adversaries to predict the one used by a target. In addition, even if the adversary obtains the onion address by chance, Tor hidden services also require users to have a 132-bit authentication cookie in base64 to access the hidden server. It is very difficult for adversaries to guess the correct combination of onion address and authentication cookie. Therefore,

our schemes, by integrating Tor hidden services with the gateway, can provide a reliable authentication service to secure the IoT gateway.

Encryption. Home Assistant does not use HTTPS by default, so it is very insecure when accessing from remote. To address this problem, users are often suggested to set up additional link encryption using Let's Encrypt [7] for example. However, this requires a tedious process to configure the setup, especially for users who has little knowledge about networking and security. In practice, this usually discourages users from adopting secure configurations. On the contrary, our schemes are atop Tor hidden services, which have built-in onion encryption and end-to-end encryption. With a very simple configuration process during installation, we can set up the IoT gateway running over Tor hidden service. After that, all traffic that goes through Tor is well protected without any user involvement.

In the *HA-mTorHS* scheme, since all communication is over Tor, the data confidentiality completely relies on Tor's strong cryptography technologies. In the *HA-Separation* scheme, we use Tor channel for command and Internet channel for data. All commands are protected by Tor as discussed before, while data is also encrypted by an AES session key, which is negotiated through the command channel between user-end application and Home Assistant over hidden service.

Anti-scanning. Anti-scanning approach is an effective solution for cyberattacks against smart homes. The current Home Assistant running on the default port 8123 will respond to adversary's scanning on this port, so the adversary can find vulnerabilities and exploit them. In our *HA-mTorHS* scheme, all the access to Home Assistant must pass through the hidden services. Without knowing the onion address and the authentication cookie, adversary does not know the existence of Home Assistant, and thus cannot probe and access it. In our *HA-Separation* scheme, authentication and key negotiation are conducted through the command channel over Tor, so those vulnerability scanning techniques will not work any more. For data channel, it is still resistant to scanning when the port is temporarily open to the public Internet for data transmission due to two reasons: (1) On the data channel Home Assistant only responds to the connection request that contains the nonce encrypted by session key, which is sent to the user through Tor. Any other traffic will be dropped, so the scanning without the correct nonce will receive no response; (2) Since the port on Home Assistant is a random port and only temporarily open during data transmission, the adversary even may not have enough window of time to detect a open port via massive scanning, let alone a successful attack.

7 Related Work

Extensive research has been conducted to enhance the security of individual devices. For authentication, Liao et al. [28] propose a secure ECC-based RFID authentication scheme to realize the mutual authentication between devices.

Wu et al. [39] further improve the security by proposing lightweight private mutual authentication and private service discovery. For encryption, traditional cryptography can be applied to secure IoT. Dinu et al. introduce a benchmark framework to evaluate how well lightweight block ciphers, such as AES, RC5, Simon and Speck, etc., are suited to IoT devices. For communication, a bunch of dedicated protocols such as CoAP [34], RPL [38], and 6LoWPAN [33] and their variants have been proposed, which are not only lightweight for IoT devices but also security-oriented.

Another direction of approach that aims to achieve both security and efficiency is cloud-assisted IoT security designs. Since resource-constrained IoT devices usually cannot afford costly cryptographic techniques and large data storage, many schemes propose to solve this problem by leveraging the connected cloud, which provides powerful computation and storage capacity [23,24,40,44]. For example, [17,29] focus on cloud-assisted healthcare IoT, which mainly use the storage resources of the cloud. In [17], they proposed a scheme to add watermark into the collected data of a patient to avoid the privacy leakage on the cloud, while Yang et al. proposed a scheme that allows health service providers such as doctors to access and verify the encrypted medical records stored on the cloud by using a searchable encryption with forward privacy support [29]. In contrast, [28] utilizes the computation resources of the cloud to implement a data publishing scheme adopting attribute-based encryption, while [24] proposes a data access control scheme for constrained IoT devices and cloud computing based on hierarchical attribute-based encryption. The above solutions mainly focus on securing individual IoT devices, while IoT gives us a way to manage and secure a bunch of heterogeneous devices. For example, Intel IoT gateway can support comprehensive device protection with integrated McAfee, including secure boot, application integrity monitor, encrypted storage and more [6].

This work is also related to Tor routing optimization. Several multipath Tor schemes have been proposed to balance security and performance in Tor routing [14,41,42]. AlSabah et al. [14] first explored how to use multipath routing to improve Tor's performance, and Yang et al. [42] further analyzed the relay usage and proposed to use low-bandwidth relays to construct multiple circuits to improve performance and increase network utilization. Yang et al. [41] proposed a partial multipath routing scheme for Tor hidden services to enhance the resistance to traffic analysis. The tunnel is only built between the rendezvous point and the hidden server. They improve the anonymity based on the insight that traffic pattern are distorted by flow splitting and flow merging operations and by the multiple routes with different network dynamics. In contrast, with the goal of improving the performance, our proposed multipath Tor hidden services for IoT gateway adopts an end-to-end multipath structure, which leverages the end-to-end traffic management to work with network dynamics. Another notable difference is that our multipath scheme is transparent to Tor network, namely, no modifications are required on existing Tor routers except user proxy who is using multipath Tor, and thus, our scheme can be seamlessly adopted, while all other three schemes require new Tor routers to support their designs.

8 Conclusion

Security and privacy are critical issues in the adoption of IoT devices. IoT onion gateways provide strong security protection, but they suffer from the performance bottleneck caused by the limited bandwidth of Tor. To tackle this issue, we present a *multipath Onion IoT gateway*, which transmits IoT data stream through an anonymous tunnel with multiple Tor circuits with congestion control. We also present a *split channel Onion gateway*, which splits the command and data channels, to utilize the less-secure public Internet to route encrypted data streams. We have demonstrated the effectiveness, efficiency, and security guarantees of the proposed approach through experiments and security analysis.

Acknowledgment. This work is sponsored in part by the National Security Agency (NSA) Science of Security Initiative and the US National Science Foundation under NSF CNS-1422206 and DGE-1565570.

References

1. 9 baby monitors wide open to hacks that expose users' most private moments. https://arstechnica.com/security/2015/09/9-baby-monitors-wide-open-to-hacks-that-expose-users-most-private-moments/
2. Hack Samsung Fridge. https://www.pentestpartners.com/security-blog/hacking-defcon-23s-iot-village-samsung-fridge/
3. Hackers Make the First-Ever Ransomware for Smart Thermostats. https://motherboard.vice.com/en_us/article/aekj9j/internet-of-things-ransomware-smart-thermostat
4. Hacking 14 IoT Devices. https://www.iotvillage.org/slides_DC23/IoT11-slides.pdf
5. Home Assistant. https://home-assistant.io/
6. Intel IoT Gateway. https://www.intel.com/content/www/us/en/internet-of-things/gateway-solutions.html
7. Let's Encrypt. https://letsencrypt.org/
8. Openhab. https://www.openhab.org/
9. Ransomware Ruins Holiday By Hijacking Family's LG Smart TV on Christmas Day. https://www.yahoo.com/tech/ransomware-ruins-holiday-hijacking-familys-201136667.html
10. Raspberry Pi. https://www.raspberrypi.org/
11. Smartthings. http://www.samsung.com/us/smart-home/smartthings/hubs/f-hub-us-2-f-hub-us-2/
12. VStarCam Eye4. http://www.eye4.so/
13. Trendnet cameras - i always feel like somebody's watching me (2012). http://console-cowboys.blogspot.com/2012/01/trendnet-cameras-i-always-feel-like.html
14. AlSabah, M., Bauer, K., Elahi, T., Goldberg, I.: The path less travelled: overcoming Tor's bottlenecks with traffic splitting. In: De Cristofaro, E., Wright, M. (eds.) PETS 2013. LNCS, vol. 7981, pp. 143–163. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39077-7_8
15. AlSabah, M., et al.: DefenestraTor: throwing out windows in Tor. In: Fischer-Hübner, S., Hopper, N. (eds.) PETS 2011. LNCS, vol. 6794, pp. 134–154. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22263-4_8

16. Barré, S., Paasch, C., Bonaventure, O.: MultiPath TCP: from theory to practice. In: Domingo-Pascual, J., Manzoni, P., Palazzo, S., Pont, A., Scoglio, C. (eds.) NETWORKING 2011. LNCS, vol. 6640, pp. 444–457. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20757-0_35
17. Burke, S.: Massive cyberattack turned ordinary devices into weapons (2016). http://money.cnn.com/2016/10/22/technology/cyberattack-dyn-ddos/index.html
18. Coldewey, D.: Smart locks yield to simple hacker tricks (2016). https://techcrunch.com/2016/08/08/smart-locks-yield-to-simple-hacker-tricks/
19. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium, August 2004
20. Fernandes, E., Jung, J., Prakash, A.: Security analysis of emerging smart home applications. In: Proceedings of the 37th IEEE Symposium on Security and Privacy (2016)
21. Freitas, N.: Internet of onion things (2016). https://blog.torproject.org/blog/quick-simple-guide-tor-and-internet-things-so-far
22. Gartner Inc.: Gartner IoT forecast (2017). http://www.gartner.com/newsroom/id/3598917
23. Hossain, M.S., Muhammad, G.: Cloud-assisted industrial internet of things (IIoT)-enabled framework for health monitoring. Comput. Netw. **101**, 192–202 (2016)
24. Huang, Q., Wang, L., Yang, Y.: DECENT: secure and fine-grained data access control with policy updating for constrained IoT devices. World Wide Web **21**(1), 151–167 (2018)
25. Jia, Y.J., et al.: ContexIoT: towards providing contextual integrity to appified IoT platforms. In: Proceedings of The Network and Distributed System Security Symposium, vol. 2017 (2017)
26. Kim, J.E., Boulos, G., Yackovich, J., Barth, T., Beckel, C., Mosse, D.: Seamless integration of heterogeneous devices and access control in smart homes. In: 2012 8th International Conference on Intelligent Environments (IE), pp. 206–213. IEEE (2012)
27. Kothmayr, T., Schmitt, C., Hu, W., Brünig, M., Carle, G.: DTLS based security and two-way authentication for the internet of things. Ad Hoc Netw. **11**(8), 2710–2723 (2013)
28. Liao, Y.P., Hsiao, C.M.: A secure ECC-based RFID authentication scheme integrated with ID-verifier transfer protocol. Ad Hoc Netw. **18**, 133–146 (2014)
29. Ling, Z., Luo, J., Xu, Y., Gao, C., Wu, K., Fu, X.: Security vulnerabilities of internet of things: a case study of the smart plug system. IEEE Internet Things J. **4**(6), 1899–1909 (2017)
30. Ning, H., Liu, H., Yang, L.T.: Aggregated-proof based hierarchical authentication scheme for the internet of things. IEEE Trans. Parallel Distrib. Syst. **26**(3), 657–667 (2015)
31. Notra, S., Siddiqi, M., Gharakheili, H.H., Sivaraman, V., Boreli, R.: An experimental study of security and privacy risks with emerging household appliances. In: 2014 IEEE Conference on Communications and Network Security (CNS), pp. 79–84. IEEE (2014)
32. Raza, S., Shafagh, H., Hewage, K., Hummen, R., Voigt, T.: Lithe: lightweight secure CoAP for the internet of things. IEEE Sens. J. **13**(10), 3711–3720 (2013)
33. Shelby, Z., Bormann, C.: 6LoWPAN: The Wireless Embedded Internet, vol. 43. Wiley, Hoboken (2011)
34. Shelby, Z., Hartke, K., Bormann, C.: The constrained application protocol (CoAP) (2014)

35. Sivaraman, V., Chan, D., Earl, D., Boreli, R.: Smart-phones attacking smart-homes. In: Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 195–200. ACM (2016)
36. Sivaraman, V., Gharakheili, H.H., Vishwanath, A., Boreli, R., Mehani, O.: Network-level security and privacy control for smart-home IoT devices. In: 2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 163–167. IEEE (2015)
37. Wang, T., Bauer, K., Forero, C., Goldberg, I.: Congestion-aware path selection for Tor. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 98–113. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_9
38. Winter, T.: RPL: IPv6 routing protocol for low-power and lossy networks (2012)
39. Wu, D.J., Taly, A., Shankar, A., Boneh, D.: Privacy, discovery, and authentication for the internet of things. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 301–319. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_16
40. Yang, L., Humayed, A., Li, F.: A multi-cloud based privacy-preserving data publishing scheme for the internet of things. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 30–39. ACM (2016)
41. Yang, L., Li, F.: Enhancing traffic analysis resistance for tor hidden services with multipath routing. In: 2015 IEEE Conference on Communications and Network Security (CNS), pp. 745–746. IEEE (2015)
42. Yang, L., Li, F.: mTor: a multipath tor routing beyond bandwidth throttling. In: 2015 IEEE Conference on Communications and Network Security (CNS), pp. 479–487. IEEE (2015)
43. Yang, L., Xue, H., Li, F.: Privacy-preserving data sharing in smart grid systems. In: 2014 IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 878–883. IEEE (2014)
44. Yang, L., Zheng, Q., Fan, X.: RSPP: a reliable, searchable and privacy-preserving e-healthcare system for cloud-assisted body area networks. In: INFOCOM. IEEE (2017)