LOIS: Low-cost Packet Header Protection for IoT Devices

Minmei Wang
minmei.wang@uconn.edu
University of Connecticut
USA

Shouqian Shi
sshi27@ucsc.edu
University of California, Santa Cruz
USA

Xiaoxue Zhang
xzhan330@ucsc.edu
University of California, Santa Cruz
USA

Song Han
songhan@uconn.edu
University of Connecticut
USA

Chen Qian
cqian12@ucsc.edu
University of California, Santa Cruz
USA

ABSTRACT

The widely deployed IoT devices in various applications, such as smart homes and smart factories, pose new privacy concerns. IoT devices typically capture users' activities or collect information from their surroundings and then send the information to remote cloud servers, exposing private information to passive adversaries by looking at the packet headers. Thus, in an enhanced IoT security protocol, protecting privacy also requires hiding packet headers and other traffic metadata. This work presents the LOIS framework, a packet-level packet header protector based on efficient one-time keystreams. LOIS allows IoT devices to efficiently hide the IP and port information in packet headers while allowing the cloud to recover the original packet headers. Besides, LOIS can easily integrate with existing IoT traffic padding algorithms to hide traffic patterns. We implement LOIS on commodity servers running in a public cloud. Our experimental results show that LOIS only introduces moderate overhead. For example, results show that LOIS only incurs about 250–365 ns end-to-end latency on average for the upload traffic, which is 80%–90% less than that of IPsec.

ACM Reference Format:
Minmei Wang, Shouqian Shi, Xiaoxue Zhang, Song Han, and Chen Qian. 2023. LOIS: Low-cost Packet Header Protection for IoT Devices . In *International Conference on Internet-of-Things Design and Implementation (IoTDI '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3576842.3582380

1 INTRODUCTION

The Internet-of-Things (IoT) market has dramatically expanded over the recent years and various IoT devices have been deployed in different scenarios, such as smart homes, smart factories, smart cities, smart health, and smart transportation [18][10]. Most IoT devices (such as smart sensors) communicate in a passive and on-demand way: they transmit data to other devices or remote servers as necessary with minimum user involvement. For example, many IoT devices collect data from their surroundings or capture the users' activity information, then automatically transmit the data to the remote cloud servers for data analysis services.

However, as most of the sensing data contain sensitive information about the users, this cloud-based IoT service framework posts severe privacy leakage concerns [31]. One primary privacy concern is that some users' activities and their surrounding environment information are exposed to unknown parties, such as the passive adversaries in the network path, even though the payload data is encrypted with TLS/SSL. A successful attack can be as simple as *looking at the packet headers*! Researchers have found that the encrypted IoT traffic can still be used to infer the device identity or the related user activities by analyzing the packet headers and traffic patterns [25][9][19]. For example, when devices monitor users' sleep, or when devices surreptitiously record users' activity data such as audio [2], they need to send the data packets to the specific destination IP addresses (also called the service IPs) and ports of the cloud servers with a specific pattern. Hence, by observing the IP addresses, ports, time, and frequency of those encrypted data packets, the passive adversaries can successfully obtain some sensitive information of the users, such as the type and function of IoT devices [22], the type of the activity, and the communication pattern information [6, 9]. Therefore, to enhance the data security of these IoT services, it is of significant importance to protect not only the privacy of the service data but the *headers* of the packets generated during the services.

In this work, we focus on an important yet challenging problem: hiding sensitive packet header information in IoT traffic to protect user privacy. The requirements to achieve this goal are summarized as follows.

1) **Oblivious service IPs.** The destination IP of an IoT packet needs to be oblivious to a passive adversary. The packet header cannot reveal which application or service the packet is used for. In particular, a cloud may host many IoT services, and each service is usually assigned a dedicated IP address called the service IP [13, 16]. Hence the service IP should be hidden from the passive attackers in the network path.

2) **Hide device identity and activities.** Adversaries cannot link the identity of an IoT device to the packets its sent.

These requirements are challenging for two reasons. First, packet header fields, such as IP addresses and port numbers, are used to

IoTDI '23, May 09–12, 2023, San Antonio, TX, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0037-8/23/05…$15.00
https://doi.org/10.1145/3576842.3582380

identify the packets by the upper layer applications and route traffic to their destinations. Hiding this information without influencing its function is challenging. Second, IoT devices are resource-constrained. Hence, to make it friendly for those devices, the protection method should incur light overhead.

An intuitive solution for protecting packet headers is to use virtual private networks (VPNs), as suggested in a recent study [7]. A VPN tunnel can wrap all traffic through the tunnel by encrypting the packets, including their headers. However, using VPNs has several limitations. 1) VPN performs encryption and decryption operations on both packet headers and payloads, causing considerable performance degradation, as shown in our evaluation in Sec. 7. In fact, the majority of IoT packets are small packets [19], and the payloads have already been encrypted by TLS/SSL [24, 31]. Hence the overhead is not necessary. 2) VPN technology cannot protect the device's identity (IP address) when building a VPN tunnel between the device and the remote server.

This work presents a system for **L**ightweight **O**blivious **I**oT **S**ervices called LOIS, which achieves the above requirements of protecting packet headers and hence user privacy. The main idea is built upon the fact that some major cloud providers, such as Amazon and Google, host a large number of IoT applications – either by themselves or their customers. Hence, each such cloud can offer a unified IP for all the supported services, which can be the destination IP address for all the packets to the cloud. The unified IP to server IP translation can be performed on the cloud load balancers, which are currently doing virtual IP (VIP) to server IP translation [21]. Then the sensitive fields in the packet header are encrypted using *stream cipher* with a one-time keystream chosen from a list of keystreams of the requested service, which can protect device identities on the entire path to the server with much less overhead compared to VPNs.

The proposed LOIS framework consists of the following main modules: the keystream management module and the packet header modification module. In addition, LOIS can effectively integrate the stochastic traffic padding (STP) algorithm [7] in the traffic analysis defense module to defend against traffic analysis attacks. We implement LOIS using the Intel data plane development kit (DPDK) on commodity servers and compare it to IPsec and the pure forwarding method (Vanilla). We find that LOIS incurs a small end-to-end overhead for the bidirectional traffic, around 250–365 ns on average for the upload traffic and around 164–250 ns for the download traffic, which is 80%–90% less compared with IPsec. In addition, in contrast to IPsec, LOIS can directly send the download packets from the server to the device, which significantly saves the network bandwidth for the load balancer.

The rest of this paper is organized as follows. We analyze characteristics of IoT traffic in Section 2. Section 3 introduces models and gives the problem specification. Section 4 introduces the overview of LOIS. We illustrate the design of the keystream management module in Section 5 and introduce the design of the packet header modification module in Section 6. We show the evaluation results in Section 7. We discuss some concerns related to the deployment and the privacy of LOIS in Section 8. Finally, we summarize the related work in Section 9 and conclude this work in Section 10.

2 CHARACTERISTICS OF IOT TRAFFIC

In this section, we analyze some characteristics of IoT traffic. We adopt a public-available IoT traffic data set [26], which contains the network traffic of 28 unique IoT devices in a smart home representing six different categories: cameras, switches, triggers, hubs, air quality sensors, electronics, and healthcare devices. We download the 20 days of IoT traffic data for analysis.

We first find that it is very easy for a passive adversary to infer which IoT application a packet is used for because **the destination IP and port of the packet headers are completely visible** although the payloads are encrypted by TLS/SSL. We then choose one device from each category to study the distribution of the packet sizes. Fig. 1 shows the statistics result. We can find that IoT devices tend to exchange a small amount of data in each packet. Most of the packet sizes are less than 250 bytes.

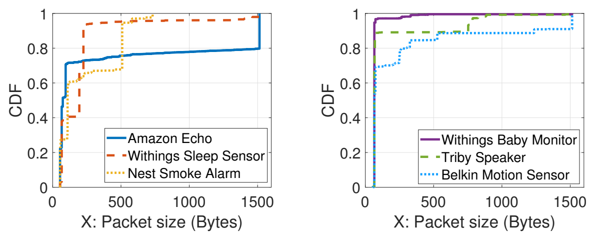


Figure 1: Distribution of IoT packet size

We further measure the average number of packets generated or received by each device in one day. Fig. 2 shows the results of twelve different IoT devices. We can find that the average number of packets in one day varies significantly among various IoT devices.

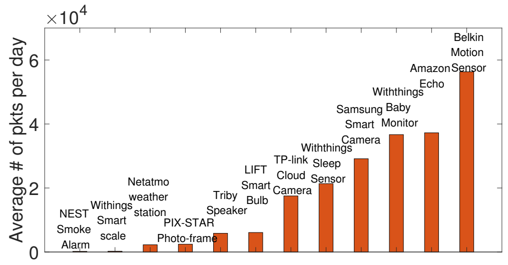
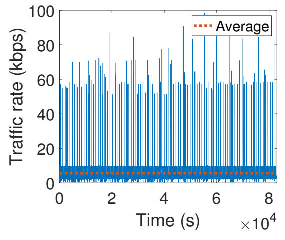
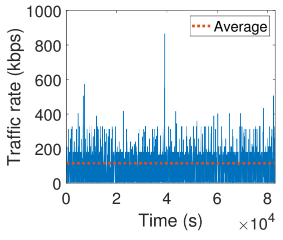


Figure 2: Average number of packets

Next, we use one-day traffic data to measure the average traffic rate for different devices. The data were collected from 7:00 am. We sum the traffic size in one second to calculate the average traffic rate per second for each IoT device. Fig. 3 shows the results of Amazon Echo and Belkin Motion Sensor. We can find that different devices have different traffic patterns. For example, the average traffic rate is around 6 kbps for Echo and 115 kbps for the Belkin motion sensor. In addition, we find there are many clear pattern changes (such as the temporal traffic peaks) that can tell some specific user activities for both applications. Passive adversaries can utilize the traffic pattern to infer device type and user behaviors. For example, an attacker can use machine learning models to classify the traffic types and detect or recognize the user's activities from the encrypted IoT packet trace data [20, 28].



(a) Amazon Echo



(b) Belkin Motion Sensor

Figure 3: Average traffic rate

Take-aways. The headers of the IoT packets are completely not protected, and they can reveal user privacy. The sizes of IoT packets are small, mostly under 250 bytes. Traffic patterns of different IoT devices reveal user privacy.

3 MODELS AND PROBLEM STATEMENT

3.1 Network model

We consider the scenarios where the IoT devices communicate with the cloud servers for some service tasks such as sensor data reporting and analysis. The IoT device either directly connects to the Internet through an access point, or the IoT device is located in a smart community that consists of different types of IoT devices that are managed by a gateway. One common use case is a smart home scenario with sleep monitors, security cameras, smart door locks, etc. Another use case can be an industrial IoT network such as healthcare industrial IoT [17], or an organization/building network with various IoT devices. Fig. 4 shows the network model, which consists of the following four main components.

1) IoT devices. An IoT device (or "device" in short) is an object with sensors or actuators, which has constrained computing, memory, and power resources. The devices can sit in a smart community and connect to the Internet through an access point. The devices capture and collect data, then transmit the data to the cloud. The devices may also request services from the cloud.

2) Access points. An access point is a network device that helps IoT devices to connect to the Internet. The access point usually connects to a router as a standalone device. In some network setups, the access point is an integral component of the router.

3) Servers. The servers provide various services for IoT devices, and are typically located in the cloud.

4) Load balancers. A load balancer, which is deployed in a cloud, manages the traffic to the cloud. The load balancer should process every packet to the cloud in both current practice and the LOIS system. It is worth noting that packets sent from servers to devices do not need to pass through the load balancer. Load balancers in the cloud provide a unified cloud IP address called CIP to serve as the destination IP address for all the traffic sent to the cloud. In addition, load balancers use unique service IDs (SID) to distinguish services for different traffic. Load balancer are also responsible for translating the unified IP address and the service ID to the destination server IP and the port number.

3.2 Threat model

We assume the traffic between IoT devices and remote cloud servers is encrypted using the TLS/SSL protocol. Thus, the traffic packet

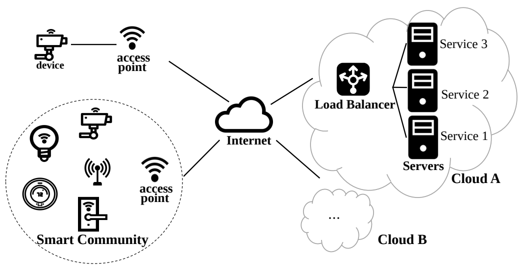


Figure 4: Network model

content is not accessible to the entities except for the sender and the receiver. The servers and the load balancer in the cloud are trusted.

We are concerned with the passive adversaries that can collect network traffic and infer the user's private information from the traffic data. Although the passive adversaries cannot view the packet payload of encrypted traffic, they can easily view the source and destination IP addresses and the port numbers from the packet headers, and infer the device identity and requested service information. Furthermore, the passive adversaries can get traffic rates, inter-packet intervals, and packet size information to infer more sensitive information, such as the device type and users' activities that trigger the traffic. The passive adversaries can reside along the path from the device to the access point, from the access point to the cloud, or within the cloud. We divide the adversaries into two different categories.

1) Local adversaries. Local adversaries are the passive adversaries that are located on the path from the device to the access point. Or, if the IoT device is deployed in a smart community, local adversaries sit in the local area network (LAN). Local adversaries with access to the Wi-Fi network can view all packet headers. Local adversaries without access to the Wi-Fi network can only know the sending time of the packet and view the link layer header , including MAC address, sizes of Wi-Fi packets, while other information like IP headers and transport layer headers are encrypted.

2) External adversaries. External adversaries can view the traffic only after packets leave the access point. They are either located along the path from the access point to the cloud (on-path adversaries) or sit within the cloud (cloud adversaries). So external adversaries can view all the IP header and transport layer header information. But they cannot get the MAC address of the packets. One representative external passive adversary is the Internet Service Provider (ISP). We assume the cloud provider and the servers are trusted, but there could be a passive adversary in the cloud network, such as a compromised router. If the servers (the receiver) are not trusted, all possible protections of packet headers will fail.

We assume adversaries do **not** have the power to act as **global passive adversaries** that can observe both the traffic inside the cloud and outside the cloud. They can only sit in one place of the network path and view part of the traffic.

3.3 Problem specification

The detailed objectives of LOIS are: 1) Hidden service. If a passive adversary is a local adversary or a on-path adversary, it cannot know the services IPs and corresponding ports of the packet. 2) Sender anonymity. Passive adversaries cannot discover the identity of the IoT device (IP address) for the upload traffic sent from the device to the cloud. 3) Packet unlinkability. Passive adversaries cannot determine whether the packets belong to the same connection by directly viewing packet information. This property helps to defend against traffic analysis attacks or user tracking based on packet header information. We also consider the traffic statistic analysis attack based on the traffic pattern information. We assume that passive adversaries can gain prior knowledge about the characteristics of IoT traffic. Thus, they can utilize this knowledge to identify the device type and infer related user activities. We integrate the existing stochastic traffic padding (STP) algorithm [7] to defend this attack.

There are **two main challenges** to achieve the above objectives. First, since we rely on keystreams to hide sensitive packet header information, how to efficiently manage keystreams and make keystreams correspond to provided services. Second, making the packet's sender get the keystream efficiently and transfer the used keystream information to the receiver without exposing it to the passive adversaries is challenging.

3.4 VPN is not an optimal approach for our goals

In this section, we discuss the feasibility of utilizing the VPN technology to defend against passive adversaries. The load balancer scales out services hosted in the cloud by mapping packets destined to a provided service with a virtual IP address to a pool of servers with multiple direct IP addresses [21]. We build two VPN tunnels: Tunnel 1 wraps all traffic between the IoT device and the load balancer, and tunnel 2 wraps all traffic between the load balancer and a server. For the upload traffic sent from the device to the server, the requested service is hidden by tunnel 1, and the device identity and service type are protected by tunnel 2. Since servers that provide services dynamically change, the client cannot directly build a VPN tunnel between the device and the target server. Therefore, the download traffic also needs to pass through the load balancer and be protected by tunnel 1 and tunnel 2.

However, there are limitations of utilizing VPN technology to solve the problem. 1) VPN tunnel 1 cannot provide sender anonymity, exposing the device's IP address to passive adversaries. On-path adversaries can simply collect traffic with the same IP address to a cloud within a period, linking these packets to a connection. Thus, VPN technology cannot achieve our objectives. 2) For bidirectional traffic, the VPN technology poses two encryptions and two decryptions on every packet, causing considerable computational overhead. 3) The download traffic also needs to pass through the load balancer, bringing extra computational overhead and bandwidth consumption on the load balancer.

4 OVERVIEW OF THE LOIS FRAMEWORK

Fig. 5 shows the overview of the LOIS framework. LOIS is installed as three types of interfaces: 1) the device interface called LOIS-DI; 2) the interface on the cloud load balancer called LOIS-CI; and 3) the server interface LOIS-SI. Each interface is responsible for certain operations on packet headers.

Each LOIS-CI is assigned an IP address (CIP) which is served as the destination IP for all the services it manages. Simultaneously,

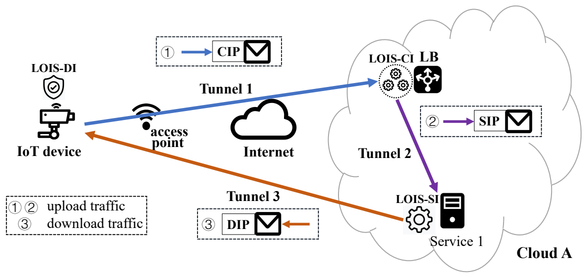


Figure 5: Overview of the LOIS framework

LOIS-CI provides a service ID (SID) to distinguish every managed service. CIP and SID are used to identify the requested service by setting them to be the destination IP address and the destination port number of a packet sent to the cloud. LOIS-CI is implemented on the load balancer. LOIS-DI can be directly implemented on the device, and the IP address of this device is called DIP. LOIS-SI is implemented on each server that provides services. Each LOIS-SI is assigned with an IP address called SIP to denote as the destination IP address that provides services.

For the upload traffic which is sent from the device to the remote cloud server, LOIS performs the following steps.

Step 1. LOIS-DI hides all the sensitive information in the packet header. Adversaries can only view that the packet is sent to the LOIS-CI with CIP.

Step 2. LOIS-CI only recovers the IP address and the port number of the target server. Other fields are hidden to prevent passive cloud adversaries from inferring sensitive information from the packets, such as the source device and the flow-related information. Then, the packet is sent to the target server with SIP. After the server receives the packet, it will recover all the hidden fields.

For the download traffic, LOIS performs as follows.

Step 3. LOIS-SI directly hides all sensitive fields and sends it to the destination device with DIP. Adversaries can only view that the packet is sent to the DIP.

The main idea behind the LOIS framework to hide the packet metadata is to use *a one-time keystream* to encrypt sensitive fields in the packet header for each packet. There are three main modules which are listed as follows.

(1) **Keystream management module** generates, stores, and updates keystreams. Simultaneously, a LOIS handshaking process is proposed for LOIS-DI and LOIS-CI to create the same keystream table.

(2) **Packet header modification module** modifies the packet header to protect the packet metadata information.

(3) **Traffic analysis defense module** hides the traffic pattern and prevents passive adversaries from inferring sensitive information according to the traffic pattern.

As the traffic pattern exposes times and user activities, the LOIS framework integrates the existing traffic padding algorithms called STP [7] in the traffic analysis defense module to hide it. The traffic pattern is hidden for the traffic in tunnel 1, tunnel 2, and tunnel 3.

LOIS-DI can also be implemented on a local server or any programmable network middlebox, such as Wi-Fi access point or access gateway router. Multiple devices can sit behind LOIS-DI, and LOIS-DI protects those managed devices by preventing external adversaries *from sensing sensitive information with less overhead compared to the VPN method.*

5 DESIGN OF THE KEYSTREAM MANAGEMENT MODULE

Each packet requires one unique keystream to encrypt the sensitive fields to hide them in packet headers, thus protecting user privacy. Keystreams are pre-generated and stored on LOIS-DI, LOIS-CI, and LOIS-SI instead of real-time generation due to security and efficiency requirements. Becuase generated keystreams need another process of deduplication to avoid key reuse attack, which costs time. This section introduces the design and the management of keystreams. For each packet, the receiver needs to know which keystream is used without exposing the keystream itself. Hence we use part of each keystream as an identifier. The identifier is sent in plain text and used by the receiver to identify the keystream used in the packet and recover the original packet header. Identifiers are in plain text, but passive adversaries cannot infer the corresponding keystream from an identifier without knowing detailed identifier-to-keystream mappings, since identifiers and keystreams are randomly generated and there is no relation between an identifier and a keystream. These mappings are shared secret by the IoT devices, cloud load balancer, and servers. Besides knowing the keystream for an identifier, its related service ID also needs to be stored to obtain the service information from an identifier efficiently. In LOIS, the identifier is the key k, and the keystream is the value v for key-value lookups. The identifier is used for two purposes: 1) to retrieve the keystream from the local mapping table at the packet recovery side and decrypt the packet header; 2) to get the service ID.

There are three requirements for the design of the keystream management module, which are summarized as follows. 1) Each communicating LOIS pair, which contains one LOIS-DI and one LOIS-CI, owns the same list of keystreams for all the requested services by the device. Also, LOIS-DI and LOIS-CI need to distinguish keystreams for different services. 2) Keystreams cannot be used twice to prevent adversaries from recovering encrypted fields by the reused key attack. 3) Efficiently obtaining the keystream for each packet is required.

5.1 Keystreams generation

Keystreams are generated using a pseudorandom number generator by agreeing on a seed and a nonce. Then the long keystream is cut into segments. We set 64 bits for identifiers, 128 bits for keystreams, and 16 bits for service IDs in LOIS. Passive adversaries cannot infer the corresponding keystream according to the identifier unless they know identifiers-to-keystreams mappings. Since the traffic are bidirectional and packets in the upload traffic and the download traffic both need keystreams to protect packet metadata, each keystream table contains keystreams for both request packets and reply packets.

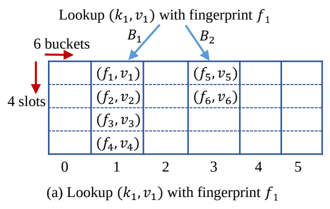
In this work, we propose a LOIS handshaking process to generate the same list of keystreams and their identifiers for a LOIS pair. We assume LOIS-DI and LOIS-CI have already built a secure channel based on the public key infrastructure (PKI). The LOIS handshaking protocol is run on the built secure channel. The purpose of the protocol is to agree on the seed, the nonce, requested service information, and the number of keystreams for each service through the secure channel. The LOIS handshaking process is initialized by LOIS-DI when it tends to request services from a cloud. LOIS-DI sends a hello message to LOIS-CI, which includes a nonce number r_d (64 bits), requested services information, the number of keystreams k for each service. After LOIS-CI receives the hello message, it also generates a nonce r_c (64 bits) and sends it to LOIS-DI. The seed s is generated by a PRNG based on $< r_d, r_c >$, which is unique for different LOIS pairs. After the message has been exchanged in the handshaking process, both LOIS-DI and LOIS-CI generate keystreams using the same seed s.

5.2 Keystreams storage

The LOIS framework needs to use **a key-value table** to store keystreams information **with unique requirements**: 1) It is well known that using two identical keystreams is dangerous for a stream cipher. Thus, each identifier in the table of a device should be unique. 2) Each cloud serves many devices simultaneously. Hence **global duplication** should be applied for these tables; 3) The table should cost very small memory and support fast lookup and update operations.

5.2.1 ***Core algorithms.*** To meet the requirements, our innovation is to improve a recent tool called Vacuum filters [30], which is an enhanced version of cuckoo filters [14] and achieve $O(1)$ lookup time and amortized $O(1)$ insertion time for approximate membership queries. We *enhance the original Vacuum filter design for a space-efficient and fast key-value store with deduplication across the tables on different devices.* Instead of storing the complete key, our design stores the fingerprint of each key to save memory. Thus, we call the designed key-value store as a partial key Vacuum table. We adopt 16 bits fingerprints in the experiments. Fig. 6 shows the (2,4)-partial key Vacuum table (hereinafter called the (2,4)-PK Vacuum table). A (2,4)-PK Vacuum table consists of a number of buckets. Each bucket has four slots. (2,4)-PK Vacuum table stores the l-bit digest of k, which is represented as f, to save memory. Every (f, v) pair is stored in one slot of the two candidate buckets B_1 and B_2 based on the hash values of the key k.

(2,4)-PK Vacuum table insertion. For any (k, v) pair, (2,4)-PK Vacuum table stores its fingerprint and value (f, v) in an empty slot in bucket $B_1(k)$ or $B_2(k)$. If neither $B_1(k)$ nor $B_2(k)$ has an empty slot, the PK Vacuum table will perform the **eviction** process. It chooses a non-empty slot in bucket B (B is $B_1(k)$ or $B_2(k)$). The (f', v') stored in the slot ((f_1, v_1) in Fig. 6) will be removed and replaced by (f, v). Then (f', v') will be placed to a slot of its alternate bucket. If the alternate bucket is also full, the PK Vacuum table recursively evicts an existing stored pair (f'', v'') to place (k', v'), and looks for an empty slot for (f'', v''). When the number of recursion process exceeds a predefined threshold, this insertion is failed and a reconstruction of the whole table is required. The detailed algorithm is shown in Algorithm 1. Here H, H', and H'' are uniform hash functions. Function `Alt` is the multi-range alternate function. Each item individually chooses an alternate range from the four alternate ranges (ARs) to calculate the index of the alternate bucket. L_x is the chosen alternate range (AR) for item x . The four ARs are denoted as L_0, L_1, L_2, L_3 ($L_0 \geqslant L_1 \geqslant L_2 \geqslant L_3$), which



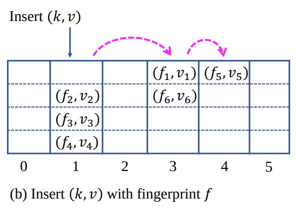


Figure 6: Partial key vacuum table

are automatically calculated by the algoritm reported in Vacuum filter [30].

Algorithm 1: PK Vacuum table insertion: Insert(k,v)

```
$f = H''(k)$;
$B_1(k) = H(k) \mod m$ ;
$B_2(k) = \texttt{Alt}(B_1(k), f)$);
$\texttt{Alt}(B_1(k), f)) = B_1(k) \oplus (H'(f) \mod L_k)$;
if $B_1(k)$ or $B_2(k)$ has an empty slot then
    put $(f, v)$ in an empty slot ;
    return Success;
Randomly select a bucket $B$ from $B_1(k)$ and $B_2(k)$ ;
for $i = 0; i < MaxEvictions; i$++ do
    foreach fingerprint $f'$ in $B$ do
        if Bucket $\texttt{Alt}(B, f')$ has an empty slot then
            put $(f, v)$ to the original slot of $f'$ ;
            put $(f', v')$ to the empty slot ;
            return Success ;
        Randomly select a slot from bucket $B$ ;
        Swap $f$ and the fingerprint $f'$ in the chosen slot;
        $B = \texttt{Alt}(B, f)$ ;
return Fail;
```

(2,4)-PK Vacuum table lookup. The lookup of value for a key k is to fetch the two candidate buckets and match the fingerprint f of the key in all eight slots until a fingerprint matches f. Then, the corresponding value of the key k is obtained. Otherwise, the item is not stored in the table.

In this work, we utilize the (2,4)-PK Vacuum table to achieve high memory utilization and fast lookup throughput. One particular requirement is that no duplicate identifiers and no keystreams exist in the table. The PK Vacuum table can detect the duplication of identifiers by first querying the table with the identifier. If the result is negative, the identifier is unique. To detect the duplication of keystreams, we store fingerprints of the existing keystreams in a Vacuum filter. Therefore, to successfully insert a key-value item (k, v), the first step is to query the Vacuum table with the identifier k and query the filter with the keystream. If both results are negative, this key-value item can be inserted into the PK Vacuum table.

Update of keystreams table. Keystreams are dynamic in the LOIS framework with new keystreams that are continuously inserted. A single PK Vacuum table and the corresponding Vacuum filter are not sufficient because the table's size may not be big enough to store all the key-value items with continuous insertions after the table is created. In this work, we design a dynamic partial key Vacuum table (DVT), which is inspired by dynamic cuckoo filter [11]. The DVT leverages the PK Vacuum table as the building block and consists of a number of s linked homogeneous PK Vacuum tables. Initially, a DVT consists of only one PK vacuum table. The DVT will extend its capacity by linking a new PK Vacuum table. Each PK Vacuum table in the DVT has the same number of buckets. We also adopt a dynamic Vacuum filter (DVF), which consists of several linked Vacuum filters, to store fingerprints of keystreams in the dynamic environment with an expansion set of keystreams. We maintain consistency between the DVT and DVF, in which the fingerprints of keystreams that are stored in the i-th linked PK Vacuum table are stored in the i-th linked Vacuum filter. To insert a keystream, its identifier and the keystream will first need to be queried in the DVT and the DVF, respectively, to check duplication. If both results are negative, this key-value item can be inserted into the DVT, and the keystream is inserted into the DVF. The insertion algorithm of the DVT and DVF is similar to that in the dynamic cuckoo filter.

Lookup of keystreams. The lookup of a keystream by its identifier from the DVT requires to probe every PK Vacuum table in the DVT. If a matched fingerprint of the identifier is found, the corresponding keystream is returned. Since keystreams are consumed, and each keystream is only used once, we also maintain a consumption rate for each PK vacuum table in the DVT. Once a keystream is used by querying it with the identifier, the PK Vacuum table's consumption rate that stores this key-value pair is updated. If the consumption rate of one PK Vacuum table reaches 100%, this PK Vacuum table is deleted from the DVT. Simultaneously, the corresponding Vacuum filter storing fingerprints of keystreams is deleted from the DVF.

5.2.2 ***Storage.***

Since LOIS-CI serves many LOIS-DIs simultaneously, LOIS-CI maintains much more keystreams than each LOIS-DI. After agreeing on the same seed and the nonce between a LOIS pair to generate a number of keystreams for both request packets and reply packets, LOIS-CI may detect that some keystreams cannot be used due to duplication. Thus LOIS-CI will send duplicate keystreams to LOIS-DI to notify LOIS-DI not to choose these keystreams, making consistency of keystreams without duplication on LOIS-DI and LOIS-CI. Fig. 7 shows the storage of keystreams for LOIS. LOIS-CI stores mappings from identifiers to <keystream, SID> pairs for the upload traffic. LOIS-DI stores unused <identifier, keystream> pairs of each service for the upload traffic. Besides, LOIS-DI stores mappings from identifiers to <keystream, SID> pairs in a DVT for the download traffic of all required services.

Each LOIS-SI builds a smaller dynamic PK Vacuum table that only contains mappings from identifiers to <keystream, SID> pairs of its provided services for the upload traffic for all ongoing served LOIS-DIs. To construct the smaller DVT for LOIS-SI, LOIS-CI sends metadata including seed, the identity of each LOIS-DI called DIP, number of keystreams, and duplicated keystreams to LOIS-SI. Then, LOIS-SI generates the same list of keystreams as LOIS-CI and locally updates the DVT. Additionally, LOIS-SI keeps unused <identifier, keystream> pairs for the download traffic of different LOIS-DIs.

5.3 Obtaining the keystream for each packet

LOIS-DI needs to hide packet headers for the upload traffic, and LOIS-SI needs to hide those of the download traffic. Since LOIS-DI

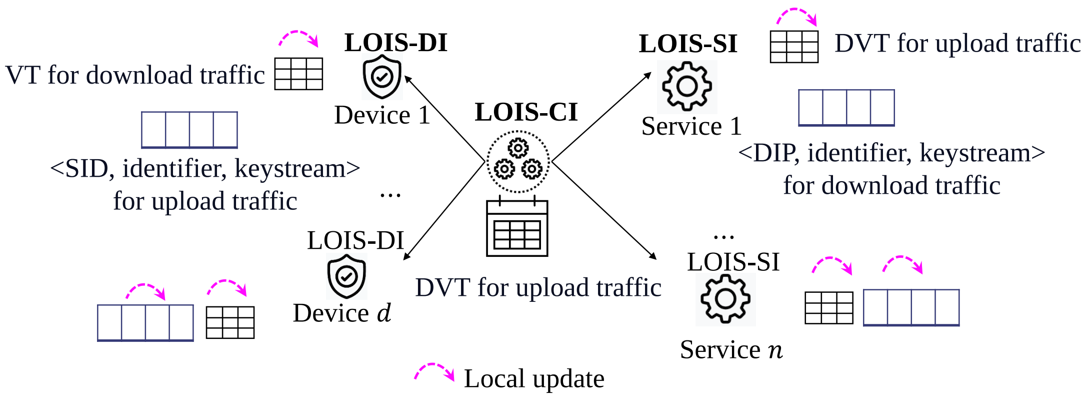


Figure 7: LOIS keystreams management

and LOIS-SI store unused <identifier, keystream> pairs of requested services for the upload traffic and the download traffic, respectively, they directly extract an unused pair for an incoming packet.

LOIS-CI and LOIS-SI need to recover hidden fields for the upload traffic in packet headers. They query their DVT to get the keystream using the extracted identifier from the packet header, which will be illustrated in Section 6. LOIS-DI performs the same way for the download traffic. Querying a DVT using the identifier does not leak information because attackers cannot infer the keystream from the identifier.

6 DESIGN OF THE PACKET HEADER MODIFICATION AND THE TRAFFIC ANALYSIS DEFENSE MODULE

This section introduces the detailed design of LOIS to hide packet headers and the traffic pattern information.

To hide sensitive information in the packet header, LOIS first classifies the header's fields into four groups.

(1) **Endpoint-related fields** contains the source address, the destination address, source port, and destination port fields. Destination address and destination port number expose the service type requested by IoT devices. Importantly, passive adversaries can group traffic for different endpoints according to endpoint-related fields. Thus, these fields need to be hidden.

(2) **Flow-related fields** contains the sequence number, the acknowledgment number, URG, ACK, PSH, RST, SYN, and FIN fields. Sequence numbers are related in each flow, leaking flow information. SYN and FIN fields indicate the start and end of a TCP flow, respectively. After adversaries identify the flow by combining endpoint-related fields and flow-related fields, they can get flow volume, flow duration, and pattern information further to infer devices' identity and the requested service. Thus, LOIS needs to hide these flow-related fields to avoid leaking flow information.

(3) **Affected fields** contain total length, header checksum, data offset, checksum, and TCP options fields. Although these fields reveal no sensitive information, they need to be changed due to the modification of other fields.

(4) **Unchanged fields** contains all the remaining fields. They do not need to be modified in the LOIS framework.

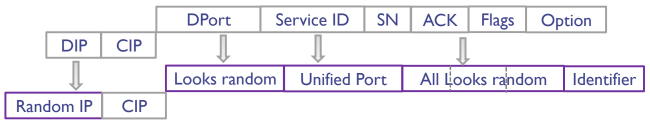


Figure 8: Packet header modification for the upload traffic

Unlike the VPN technique that encrypts all the fields in packet headers, LOIS carefully hides sensitive fields by applying two different schemes: replacement and XOR encryption. The replacement scheme is to replace the field with a unified one that cannot be distinguished. The XOR encryption scheme uses the XOR operator to encrypt the plaintext P by a keystream Q to get ciphertext C, which means that $C = P \oplus Q$, where P and Q have the same length. Specifically, P can be easily recovered by $C \oplus Q$. The keystream is generated and managed by the keystream management module. In this section, we assume that the keystream and its identifier are successfully generated for each packet.

Packet header modification for the upload traffic. We now describe the details of step 1 and step 2 presented in Section 4 and Fig. 5. **Step 1.** Fig. 8 shows the comparison of the main fields in the packet header before and after modification. In our design, the destination IP address and the destination port number of the packet are the unified cloud IP (CIP) and the service ID, respectively. However, the destination port number will expose the service information. Thus, it is then replaced by a unified port number to hide the service information. Therefore, adversaries cannot get the target service information by only directly viewing packet headers. There are two cases when dealing with source addresses and port numbers. 1) The device is located in a smart community and sits behind NAT. In this situation, LOIS-DI also integrates the NAT function by building a NAT translation table. LOIS-DI will negotiate with the gateway router to assign a unique port for this device. Therefore, LOIS-DI will first rewrite the source IP address of all the traffic from the device to a public IP of the smart community and use the assigned port number to the flow. The rewritten IP address and the port number are DIP and DPort in Figure 8. Then, these two fields will be encrypted by the keystream. 2) The device is assigned a public IP to connect to the Internet directly. Then, LOIS-DI encrypts the source IP and source port number of the upload traffic to hide

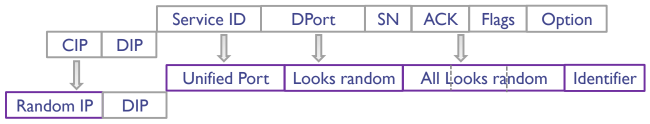


Figure 9: Packet header modification for download traffic

these sensitive fields. LOIS-DI also encrypts flow-related fields and modifies affected fields by the keystream. Here, LOIS-DI assigns an unused keystream for the requested service as the target keystream. To notify LOIS-CI of the keystream without exposing it, we insert the keystream's identifier in the TCP option field. Finally, the total length field and the data offset field are adjusted due to the option field's insertion. Due to the modification of packet headers, LOIS-DI recomputes IP header checksum and checksum in the TCP header and then sends the packet to the cloud. **Step 2.** LOIS-CI only needs to set the IP address and the port number of the service server. After LOIS-CI receives the packet, it extracts the identifier and gets the keystream and the service ID by querying the DVT. The LOIS framework should also be compatible with the function of the load balancer. We assume the load balancer uses 5-tuple to select the SIP, which is a typical design for the load balancer [21]. Thus, LOIS-CI needs to decrypt the source IP and the source port number using the keystream, while these two fields remain hidden in the packet. LOIS-CI utilizes 5-tuple and the service ID information to translate CIP and the service ID to the target server IP address and the port number, then LOIS-CI modifies the destination IP address and the port number of the packet and sends the packet to its target server. After the server receives the packet, it queries local DVT to get the keystream after extracting the identifier and recovers all the hidden fields of the packet.

Packet header modification for the download traffic. We now describe the details of step 3. As we have introduced, the traffic between IoT devices and remote servers typically contains a request packet and a reply packet. Besides, remote servers may proactively communicate with IoT devices, such as sending the command. Thus, packet headers for both bidirectional traffic need to be hidden. Otherwise, passive adversaries can infer sensitive information if we only hide packets in one direction. **Step 3.** LOIS-SI gets an unused identifier and the corresponding keystream by looking at the list that stores unused <identifier, keystream> pairs of the target service for DIP. Figure 9 shows how to hide endpoint-related fields and flow-related fields. DIP, which serves as the destination IP address, remains unchanged. Other fields, including source IP address, destination port number, and the flow-related fields, are encrypted by the keystream. A unified port replaces service ID to hide the service information. Then the identifier of the keystream is inserted in the TCP option field. Similar to step 1, other affected fields will be adjusted. Packets with modified packet headers will be directly sent to the LOIS-DI, not passing through the load balancer. Passive adversaries only view that packets are sent to the device with DIP. When LOIS-DI receives the packet, it extracts the identifier and queries the local DVT to get the keystream and the service ID. Then, LOIS-DI uses the keystream to recover all encrypted fields. The service ID then replaces the unified port number. If LOIS-DI integrates the NAT function, LOIS-DI also rewrites the DIP and DPort to the private IP address and the port number.

In the practical deployment, each IoT device may connect to several clouds to request services. To support multi-cloud scenarios, each client independently maintains keystreams for different clouds, which are generated by different seeds using the LOIS handshaking protocol. More connected clouds consume more memory on LOIS-DI to store keystreams.

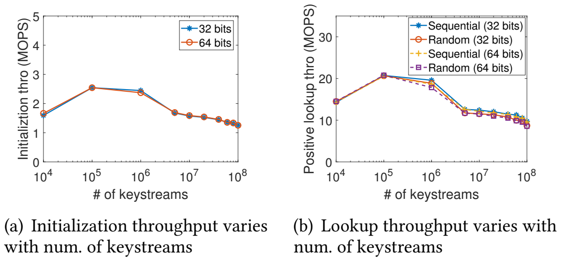
Traffic analysis defense module. The main idea is to use the link padding method to shape upload and download traffic to hide the traffic pattern. We adopt existing link padding methods in this work. The simplest way is constant rate padding, which pads the traffic to fixed-size packets with constant interpacket intervals. This padding method brings huge overhead data, as reported in [7]. In this work, we utilize the stochastic traffic padding (STP) algorithm [7]. One metric called adversary confidence is introduced in this paper to measure the expected ratio of correct activity inferences to total attempted activity inferences by passive adversaries when traffic rate metadata is protected by some techniques. STP imposes no additional network latency and can achieve low adversary confidence for relatively little bandwidth overhead.

The main idea of the STP algorithm is that the upload and download traffic during the time interval that devices communicate with remote servers is shaped equivalently. So passive adversaries cannot infer the type of user activities. Additional periods of equivalent shaping are injected randomly for bidirectional traffic. In particular, STP divides time into discrete periods of length T. STP performs a random draw from a fixed Bernoulli distribution with probability q to decide whether to shape traffic during that period in the beginning. If STP decides to pad the traffic, it first randomly draws an offset time from $[0, T]$ and starts padding the traffic after the offset time. The total padding time is T, and STP will pad the traffic to a predetermined rate R.

To defend against local passive adversaries, the STP algorithm is run on each device to pad the upload traffic. Thus, adversaries cannot infer sensitive information from the traffic pattern. For the download traffic, the STP algorithm is run on LOIS-SI. The predetermined traffic rate R is set to be the same for the devices in one smart community, making adversaries unable to infer the device type from the traffic rate. T is set to be long enough to cover complete bidirectional communications (request packets and reply packets) in one TCP connection. Similarly, T is the same for one smart community if the device locates in a community. It's suggested to set T to be the longest flow duration for all the devices in the smart community. LOIS randomizes sensitive fields in the packet header, making it harder to link packets to the same connection. We will utilize this property to design a more efficient padding algorithm to hide traffic patterns in future work.

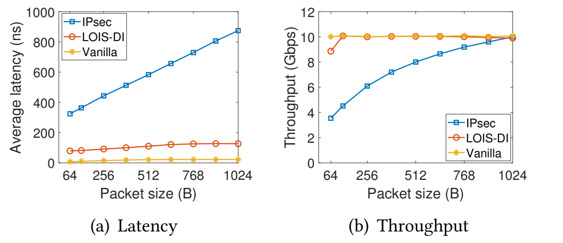
7 IMPLEMENTATION AND EVALUATION

We implement the LOIS framework using Intel Data Plane Development Kit (DPDK) [3] in a public cloud experimental environment, CloudLab [1]. DPDK bypasses the complex network stack in the Linux kernel and processes packets in the userspace. CloudLab is a testbed for researchers running experiments with cloud architectures [1]. We use c220g2 nodes in the Wisconsin cluster to evaluate the performance of the LOIS framework. Each node is equipped with one Dual-port Intel x520 10Gbps NIC, with 8 lanes of PCIe



(a) Initialization throughput varies with num. of keystreams (b) Lookup throughput varies with num. of keystreams

Figure 10: Performance of Keystream Benchmark



(a) Latency (b) Throughput

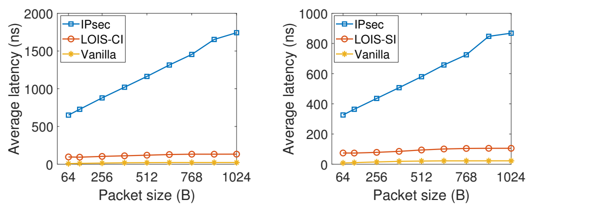
Figure 11: Performance of DI varying with packet size

V3.0 connections between the CPU and the NIC. And each node has two Intel E5-2660 v3 10-core CPUs at 2.60 GHz. The Ethernet connection between every two nodes is 2x10 Gbps. Logically, node 1 uses the DPDK official packet generator Pktgen-DPDK [5] to generate packets or get packets from a real IoT traffic dataset. Node 2 and node 3 work as the LOIS-DI and the LOIS-CI, respectively. Node 4 works as the LOIS-SI. In addition, we implement LOIS-DI on a Raspberry Pi 3 with one single 1.4 GHz processor and 1 GB RAM, which works as an example of a wide spectrum of devices that can use LOIS. LOIS can be easily implemented on less powerful IoT devices if they have available memory to store keystreams and the corresponding DVT, as shown in Fig. 19(a) (E.g., the device requires about 0.15 MB for 10^4 keystreams).

We compare our proposed LOIS with IPsec [4] – a protocol used in most VPNs – implemented using DPDK, and pure forwarding algorithm (hereinafter called Vanilla). We choose AES-CBC-128/SHA1-HMAC for the IPsec algorithm. Two metrics are used to evaluate performance. (1) **Average latency** measures the average time caused by operations of LOIS-DI, LOIS-CI, and LOIS-SI. (2) **Throughput** measures the number of processed bits per second. Unless otherwise mentioned, we conduct five production runs, LOIS handles more than one million packets on each run.

7.1 Keystream benchmark

This section evaluates the efficiency of the keystream generation and lookups in the LOIS framework. The initialization process generates keystreams for request packets and reply packets, and creates required data structures. We measure the throughput in millions of operations per second (**MOPS**). In the initialization experiments, each operation is to generate a keystream and insert it to the table. In the lookup experiments, each operation is a lookup of the keystream based on the identifier.



(a) LOIS-CI (b) LOIS-SI

Figure 12: Performance of LOIS-CI/SI varying with packet size

We evaluate the initialization performance and positive lookup performance, as shown in Figure 10. 32 bits identifiers and 64 bits identifiers are generated. Figure 10(a) shows the initialization performance. We can find that the initialization throughput decreases with more number of keystreams. The total initialization throughput is about 1-3 MOPS. Figure 10(b) shows the positive lookup performance varies with the number of keystreams in one PK Vacuum table. We evaluate the performance of sequential access and random access. Results show that the PK Vacuum table has good positive lookup throughput – a positive lookup means the keystream does exist in the table. Different access pattern of identifiers on one Vacuum table has little influence on the result.

7.2 Evaluation of LOIS

Performance of DI for the upload traffic. We evaluate the performance varying with the packet size of LOIS-DI for the upload traffic. Smaller packet size means more packets generated per second under 10 Gbps bandwidth. For the upload traffic, the node running LOIS and IPsec hides sensitive information on packet headers. Figure 11 shows the results. In this experiment, the number of total requested services by LOIS-DI is 50. The service for each packet is uniformly chosen from 50 services. We can find that LOIS brings small overhead compared to the pure forwarding algorithm, which shows the efficiency of LOIS. LOIS only accesses packet headers and modifies headers either by replacement or XOR encryption; both operations are efficient. IPsec encrypts original packets, causing larger latency with larger packets. The result shows that LOIS-DI outperforms IPsec, only causing 0.14x-0.25x latency compared to IPsec. Furthermore, latency for LOIS-DI with different packet sizes is relatively stable, while packets with larger sizes require larger processing time for IPsec. Figure 11(b) shows that LOIS-DI achieves 1.0x-2.5x throughput compared to IPsec.

Performance of CI for the upload traffic. We evaluate the performance varying with the packet size on the load balancer for the upload traffic, which is shown in Figure 12(a). LOIS-CI extracts the identifier and queries the keystream from the DVT, but LOIS-CI only needs to recover the service ID and then set the destination IP address and the port number according to the service ID. For IPsec, it needs to first decrypt the packets, set the destination IP address and the port number, and then encrypt the whole packet, causing

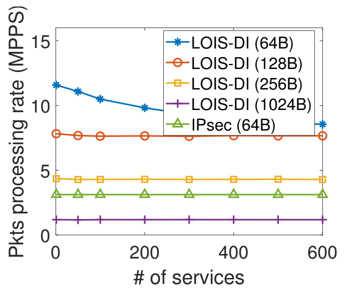


Figure 13: Performance varying with # of services

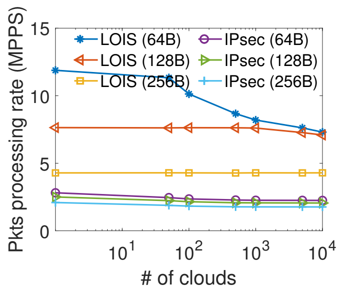


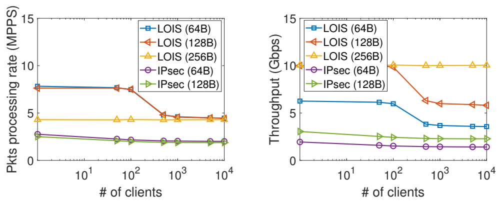
Figure 14: Performance varying with # of clouds

higher overhead. Results show that LOIS-CI only incurs about 0.1x overhead compared to IPsec.

Performance of SI for the upload traffic. We evaluate the performance varying with the packet size on the server for the upload traffic. Figure 12(b) shows the average latency. We can find that LOIS-SI brings low overhead on the server, while IPsec incurs higher overhead because the server needs to decrypt the whole received packet. The result shows that LOIS-SI only takes 0.12x-0.22x of IPsec's time for the operation on the server. Combining the total latency on LOIS-DI, LOIS-CI, and LOIS-CI, LOIS only needs 0.10x-0.20x of IPsec's time for the upload traffic.

Number of services. We evaluate the performance of LOIS-DI for the upload traffic varying with the different number of services. We generate packets with target services using a uniform distribution. The packet size is set to be 64 bytes, 128 bytes, 256 bytes, and 1024 bytes. Figure 13 shows the results. We can find that the packets processing rate is stable for packets with 128 bytes, 256 bytes, and 1024 bytes, showing that LOIS scales well with the number of services for larger packets. For small packets with 64 bytes, the performance slightly decreases with more services. Because more packets need to be processed per second on LOIS-DI, accessing identifiers and keystreams with more services increases the processing time. LOIS-DI still outperforms IPsec on small packets with 64 bytes.

Number of clouds. This part evaluates the performance for the upload traffic varies with the number of clouds. For LOIS, the client requests a total eight services from each cloud. Each service is assigned 1024 periodically updated keystreams. For IPsec, the client creates sessions with every cloud. Figure 14 shows the performance for packets with 64 bytes, 128 bytes, and 256 bytes. Results show that LOIS outperforms IPsec, because IPsec needs to encrypt packet headers and payload, bringing considerable computation overhead.



(a) Packets processing rate (b) Throughput

Figure 15: Performance of LOIS-CI varying with # of clients

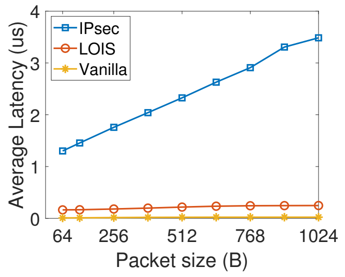


Figure 16: Average latency for the download traffic

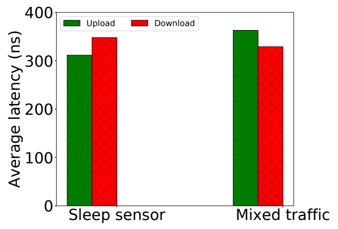


Figure 17: Average latency on Raspberry Pi 3

LOIS with 64 bytes has a performance degradation when the number of clouds is larger than 100, because obtaining keystreams with more clouds will influence the performance when the number of packets to be processed is large.

Number of clients. Since each cloud serves many clients concurrently, this part evaluates the performance varying with the different number of clients. For LOIS, each client requests a total of eight services. Each service is assigned 1024 periodically updated keystreams. As the packet size for IoT traffic is small, we show packets' performance with 64 bytes, 128 bytes, and 256 bytes. Figure 15 shows the result. We can find that LOIS outperforms IPsec, achieving >2x packets processing rate and throughput. More clients influence the table size in LOIS-CI. The lookup throughput decreases with more keystreams in the table, as shown in Figure 10(b). Therefore, for small packets (64 bytes and 128 bytes), the performance will decrease with more clients (>100). Because querying the table to get the corresponding keystreams to recover packets will dominate the performance when the number of packets that need to be processed per second is large with smaller packet sizes. Although the performance of LOIS on 64-byte packets and 128-byte packets has an oblivious decrease with a larger table, it still outperforms IPsec by a big margin.

Average latency for the download traffic. This part evaluates the total latency caused by LOIS and IPsec for the download traffic. For LOIS, the packet is sent from the server and then directly sent to the target device. LOIS-SI hides sensitive fields, and LOIS-DI recovers the packet. The packet needs to pass through the load balancer to get protection from VPN tunnel 1 and VPN tunnel 2, incurring larger overhead for IPsec. Results are shown in Figure 16. We can find that LOIS requires 0.10x-0.20x time compared to IPsec, showing the efficiency of LOIS. And the download traffic of LOIS does not pass through the load balancer, saving network bandwidth on the load balancer.

7.3 Performance on IoT traffic

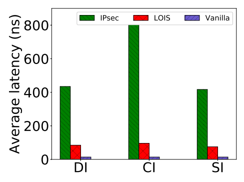
This section evaluates the performance on the real IoT traffic, of which the packet size is small. Figure 18(a) shows the result on the traffic of Withings sleep sensor. Its distribution of packet size is shown in Figure 1. Besides, we adopt one-day IoT traffic data from [26], which contains network traffic of 28 unique IoT devices. Then we utilize the packet size information from this mixed traffic to evaluate the performance, as shown in Figure 18(b). We evaluate the average latency for these two datasets on LOIS-DI, LOIS-CI, and LOIS-SI respectively. Both results show that LOIS bring small computational overhead and outperforms IPsec on real IoT traffic. In addition, we test the average latency of LOIS-DI for the upload traffic and the download traffic using the real IoT traffic dataset on the Raspberry Pi 3 testbed and present the result in Figure 17. Result show that LOIS incurs about 300 ns–365 ns average latency on the device for one packet.

7.4 Memory cost

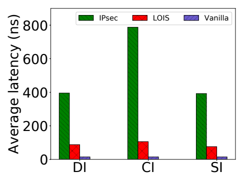
This part evaluates the memory cost for LOIS, as shown in Figure 19. We consider keystreams for the bidirectional traffic. LOIS-DI stores unused identifiers and keystreams of its requested services for the upload traffic. Besides, LOIS-DI stores identifier-to-keystream mappings for the download traffic because it needs to recover hidden fields for the download traffic. Since LOIS-DI only needs to store a small number of keystreams, LOIS introduces a low memory overhead on LOIS-DI, requiring about 0.15MB for 10^4 keystreams, as shown in Figure 19(a). In LOIS, CI stores mappings from identifiers to keystreams and services in the DVT, its memory cost is shown in Figure 19(b). Since each keystream is only used once, one solution to further reduce the memory cost on CI is to generate a small number of keystreams for each client and periodically update keystreams. Here, we do not count the memory cost on LOIS-CI for data structures used for keystream deduplication because these data structures are only used in the keystream generation stage, which can be stored in the slow memory. SI stores identifier-to-keystream mappings for the upload traffic and unused <identifier, keystream> pairs for the download traffic. However, SI only stores a small number of keystreams for its provided service, decreasing the memory cost in practice. Besides, memory is not a limited resource on servers.

8 DISCUSSION

Security analysis. Since passive adversaries try to infer sensitive information from the traffic, the first step is to separate the traffic

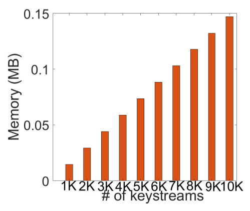


(a) Withthings sleep sensor

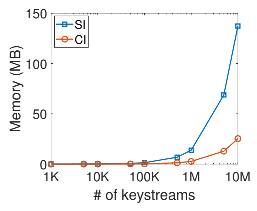


(b) Mixed IoT traffic

Figure 18: Performance on IoT traffic



(a) Memory cost on LOIS-DI



(b) Memory cost on CI and SI

Figure 19: Memory cost

for different IoT devices. In the LOIS framework, we hide the source IP and the packet header's target service information, preventing on-path adversaries from separating the traffic directly from packet headers. Simultaneously, flow information is hidden. Thus, on-path adversaries cannot map packets into different flows according to packet headers. For the local adversaries, packets are contained in an inner layer of an encrypted 802.11 frame. They can group the traffic for devices but cannot view more detailed information, such as the service type of the traffic. For cloud adversaries, LOIS can be adopted to prevent cloud adversaries from separating packets. LOIS hides sensitive fields by keystreams, making it harder to separate packets for different devices and different flows. VPN technology exposes more information because it cannot protect the sender's IP address.

Deployment of LOIS. LOIS deploys the device interface LOIS-DI directly on an IoT device, requiring modifications of end devices and increasing the deployment difficulty for the real applications. We first argue that the deployment of LOIS is not complex since we only need a module to generate and manage keystreams and a module to access and modify packet header information. Furthermore, suppose we do not have access to modify end devices. In that case, LOIS-DI can be deployed on a local server or any programmable network middlebox, such as a Wi-Fi access point or gateway router. In this setting, LOIS-DI will manage multiple IoT devices. LOIS-DI performs the same steps as that is deployed on the device. But LOIS does not prevent attacks against local adversaries that sit on the path from the device to the network device that deploys LOIS-DI. At the same time, if we want to adopt the VPN technology, we will also deploy the client side on a local server or a middlebox. Compared to the VPN-based method, LOIS brings a small overhead and thus achieves a better throughput, as discussed in the evaluation part.

Information leakage from keystreams. Keystreams are generated by pseudorandom number generators, not leaking sensitive information to passive adversaries.

9 RELATED WORK

As IoT devices typically collect sensitive data and communicate with their service providers, user privacy attracts a lot of attention. Ren *et al.* [24] perform a multidimensional analysis of information exposure from 81 devices in labs environment. Mazhar *et al.* [19] present a measurement study of smart home IoT devices in the wild by collecting real-world network traffic from more than 200 devices, showing that smart home IoT devices are susceptible to user behavior tracking.

IoT device classification and identification. Researchers classify and identify IoT devices by analyzing the IoT traffic [20, 22, 23, 25–27]. Paper [26] develops a multi-stage machine learning based classification algorithm to identify specific IoT devices based on the IoT traffic from a smart environment containing 28 different IoT devices. Paper [23] introduces a probabilistic framework for device identification. PINGPONG [27] automatically extracts packet-level signatures for device events from IoT traffic to detect the device or specify corresponding events.

Traffic analysis attack. The analysis of IoT traffic brings the possibility of attacks to IoT devices [7, 9, 12]. Paper [9] finds that IoT traffic rates can reveal potentially sensitive user interactions even when the traffic is encrypted, leading attackers to detect user behaviors. Paper [7] presents a user activity inference attack by which a passive adversary can infer user behaviors from analyzing traffic metadata. To defend against traffic analysis attack, paper [8] proposes four strategies to protect smart home privacy from passive adversaries. One strategy is to tunnel all smart home traffic through a virtual private network (VPN) to prevent device identification and user behavior inference attack. One of the limitations of a VPN-based strategy is that it cannot protect user privacy against passive adversaries after VPN endpoints. And the VPN technology decreases the packet processing rate. Another strategy is to apply traffic shaping to hide the traffic pattern. The independent link padding algorithm (ILP) [15, 29] can shape bidirectional traffic rates to a predetermined rate or schedule. Padding traffic to form fixed-size packets with constant packet intervals is the simplest form of ILP. A recent work [7] proposes a stochastic traffic padding (STP) algorithm to hide traffic pattern.

10 CONCLUSION

We present a keystream-based LOIS framework to protect user privacy by hiding IoT packet headers. LOIS includes the keystream management module, the packet header modification module, and the traffic analysis defense module. We implement the LOIS framework on commodity servers running in a public cloud. Results show that LOIS achieves a better throughput compared with IPsec, and brings small overhead for every packet. In addition, we implement LOIS-DI on a Raspberry Pi 3 to evaluate the computation overhead on LOIS-DI for bidirectional traffic. Results show that LOIS-DI incurs about 300 ns–365 ns latency on the device for one packet.

11 ACKNOWLEDGEMENT

M. Wang is supported by University of Connecticut startup funding. M. Wang, X. Zhang, S. Shi, and C. Qian were partially supported by National Science Foundation Grants 1750704, 1932447, and 2114113. And C. Qian was partially supported by the Army Research Office and was accomplished under Grant Number W911NF-20-1-0253. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. S. Han was partially supported by National Science Foundation Grant CNS-1932480.

REFERENCES

[1] [n. d.]. CloudLab. https://www.cloudlab.us/.
[2] [n. d.]. Google admits its new smart speaker was eavesdropping on users. https://money.cnn.com/2017/10/11/technology/google-home-mini-security-flaw/index.html.
[3] [n. d.]. Intel DPDK: Data Plane Development Kit. https://www.dpdk.org.
[4] [n. d.]. IPsec. https://doc.dpdk.org/guides-16.04/sample_app_ug/ipsec_secgw.html.
[5] [n. d.]. Pktgen-DPDK. https://github.com/Pktgen/Pktgen-DPDK/.
[6] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-Boo: I see your smart home activities, even encrypted!. In *Proc. of ACM WiSec.* 207–218.
[7] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. 2019. Keeping the smart home private with smart (er) iot traffic shaping. *Proc. of PoPETs* 2019, 3 (2019), 128–148.
[8] Noah Apthorpe, Dillon Reisman, and Nick Feamster. 2017. Closing the blinds: Four strategies for protecting smart home privacy from network observers. *arXiv preprint arXiv:1705.06809* (2017).
[9] Noah Apthorpe, Dillon Reisman, and Nick Feamster. 2017. A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. *arXiv preprint arXiv:1705.06805* (2017).
[10] Hamidreza Arasteh, Vahid Hosseinnezhad, Vincenzo Loia, Aurelio Tommasetti, Orlando Troisi, Miadreza Shafie-khah, and Pierluigi Siano. 2016. Iot-based smart cities: a survey. In *Proc. of IEEE EEEIC.* 1–6.
[11] Hanhua Chen, Liangyi Liao, Hai Jin, and Jie Wu. 2017. The dynamic cuckoo filter. In *Proc. of IEEE ICNP.* 1–10.
[12] Bogdan Copos, Karl Levitt, Matt Bishop, and Jeff Rowe. 2016. Is anybody home? Inferring activity from smart home network traffic. In *Proc. of IEEE SPW.* IEEE, 245–251.
[13] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A Fast and Reliable Software Network Load Balancer. In *Proc. of USENIX NSDI.*
[14] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo filter: Practically better than bloom. In *Proc. of ACM CoNEXT.* 75–88.
[15] Xinwen Fu, Bryan Graham, Riccardo Bettati, Wei Zhao, and Dong Xuan. 2003. Analytical and empirical analysis of countermeasures to traffic analysis attacks. In *Proc. of IEEE ICPP.* 483–492.
[16] Rohan Gandhi, Hongqiang Harry Liu, Y. Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. 2014. Duet: Cloud scale load balancing with hardware and software. *Proc. of ACM SIGCOMM.*
[17] M Shamim Hossain and Ghulam Muhammad. 2016. Cloud-assisted industrial internet of things (iiot)–enabled framework for health monitoring. *Computer Networks* 101 (2016), 192–202.
[18] Minhaj Ahmad Khan and Khaled Salah. 2018. IoT security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems* 82 (2018), 395–411.
[19] M Hammad Mazhar and Zubair Shafiq. 2020. Characterizing Smart Home IoT Traffic in the Wild. *arXiv preprint arXiv:2001.08288* (2020).
[20] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. 2017. ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis. In *Proc. of ACM SAC.* 506–509.

[21] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proc. of ACM SIGCOMM*. 15–28.

[22] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. 2017. Iot sentinel: Automated device-type identification for security enforcement in iot. In *Proc. of IEEE ICDCS*. 2177–2184.

[23] Jorge Ortiz, Catherine Crawford, and Franck Le. 2019. DeviceMien: network device behavior modeling for identifying unknown IoT devices. In *Proc. of ACM/IEEE IoTDI*. 106–117.

[24] Jingjing Ren, Daniel J Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Proc. of ACM IMC*. 267–279.

[25] Mustafizur R Shahid, Gregory Blanc, Zonghua Zhang, and Hervé Debar. 2018. Iot devices recognition through network traffic analysis. In *Proc. of IEEE BigData*. IEEE, 5187–5192.

[26] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2018. Classifying IoT devices in smart environments using network traffic characteristics. *Proc. of IEEE TMC* 18, 8 (2018), 1745–1759.

[27] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. [n. d.]. Packet-Level Signatures for Smart Home Devices. *Signature* 10, 13 ([n. d.]), 54.

[28] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. 2019. PingPong: Packet-Level Signatures for Smart Home Device Events. *arXiv preprint arXiv:1907.11797* (2019).

[29] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proc. of ACM SOSP*. 137–152.

[30] Minmei Wang, Mingxun Zhou, Shouqian Shi, and Chen Qian. 2019. Vacuum filters: more space-efficient and faster replacement for bloom and cuckoo filters. *Proc. of the VLDB Endowment* 13, 2 (2019), 197–210.

[31] Serena Zheng, Noah Apthorpe, Marshini Chetty, and Nick Feamster. 2018. User perceptions of smart home IoT privacy. *Proc. of ACM HCI* 2, CSCW (2018), 1–20.