S2Net: Preserving Privacy in Smart Home Routers

Seung-Seob Lee, *Student Member, IEEE*, Hang Shi, Kun Tan, *Member, IEEE*, Yunxin Liu, *Senior Member, IEEE*, SuKyoung Lee, *Member, IEEE*, and Yong Cui, *Member, IEEE*

Abstract—At present, wireless home routers are becoming increasingly smart. While these smart routers provide rich functionalities to users, they also raise security concerns. Although the existing end-to-end encryption techniques can be applied to protect personal data, such rich functionalities become unavailable due to the encrypted payloads. On the other hand, if the smart home routers are allowed to process and store the personal data of users, once compromised, the users' sensitive data will be exposed. As a consequence, users face a difficult trade-off between the benefits of the rich functionalities and potential privacy risks. To deal with this dilemma, we propose a novel system named Secure and Smart Network (S2Net) for home routers. For S2Net, we propose a secure OS that can distinguish and manage multiple sessions belonging to different users. The secure OS and all the router applications are placed in the secure world using the ARM TrustZone technology. In S2Net, we also confine the router applications in sandboxes provided by the proposed secure OS to prevent data leakage. As a result, S2Net can provide rich functionalities for users while preserving strong privacy for home routers. In addition, we develop a crypto-worker model that provides an abstraction layer of cryptographic tasks performed by a heterogeneous multi-core system. The other important role of crypto-worker is to parallelize the computations in order to resolve the high computation cost of cryptographic functions. We report the system design of S2Net and the details of our implementation. Experimental results with benchmarks and real applications demonstrate that our implementation is capable of achieving high performance in terms of throughput while mitigating the overhead of S2Net design.

Index Terms—Smart home router, ARM TrustZone, private data protection, transport layer security, heterogeneous multi-core architecture, secure operating system

1 Introduction

Owing to the proliferation of smart devices, such as smartphones, tablets, and other Internet-of-Things (IoT) devices, wireless home routers have become the center of our homes. In addition to basic network connectivity, home routers are now providing a wider range of functionalities for a connected smart home, including network acceleration, parental control, multimedia streaming, personal cloud storage, and a central hub for IoT devices. These network functionalities have been traditionally provided by internet service providers (ISPs). However, in conjunction with the rising concerns about privacy and autonomy, more users are focusing on taking control of how personal data is processed away from remote entities, such as cloud servers or middleboxes, through their own home routers [1], [2], [3]. A number of companies have already released several router products for smart homes, for example, OnHub from Google [4].

While smart routers provide more functionalities to users, they also raise security concerns. Since smart routers process and store personal data of users, which include photos taken by family members, personal documents, or even web browsing history, once compromised, these sensitive data will be exposed. Unfortunately, the existing operating systems (OS) for wireless home routers, which are mostly derived from embedded Linux, are far from secure [5]. Several attacks that specially target home routers have already been reported [6]. Adding more *smart* functions may even worsen this situation, as these new applications (apps) may have vulnerabilities and can be compromised as well. For example, a personal photo manager can expose user's private photos, if it becomes compromised. Unfortunately, this situation cannot be easily mitigated using conventional methods like OS patching, because OS updates cannot eliminate all the (potential) vulnerabilities.

Indeed, market research has shown that more than 63 percent of users are concerned about the security of their home networks, and a home router is one of the weakest links [7]. Coincidently, end-to-end encryption (i.e., HTTPS) is widely used for Internet traffic, in response to the rising concerns over the privacy of users. Since all the payloads sent over HTTPS are encrypted, end-to-end encryption discourages even the traditional network functions (like web-proxy and parental control) on the intermediate nodes such as middleboxes installed by ISPs – not to mention self-deployed, less managed home routers. For example, deep

- *S.-S. Lee and S. Lee are with the Department of Computer Science, Yonsei University, Seoul 03722, South Korea. E-mail: {shsym, sklee}@yonsei.ac.kr.*
- *H. Shi and Y. Cui are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: shi-h15@mails.tsinghua.edu.cn, cuiyong@tsinghua.edu.cn.*
- *K. Tan is with the Huawei Technologies Co., Ltd., Beijing 100085, China. E-mail: cohen_tan@hotmail.com.*
- *Y. Liu is with the Microsoft Research Asia, Beijing 100080, China. E-mail: yunxin.liu@microsoft.com.*

Manuscript received 23 Dec. 2018; revised 28 Mar. 2019; accepted 21 May 2019. Date of publication 24 June 2019; date of current version 13 May 2021. (Corresponding author: SuKyoung Lee.)
Digital Object Identifier no. 10.1109/TDSC.2019.2924624

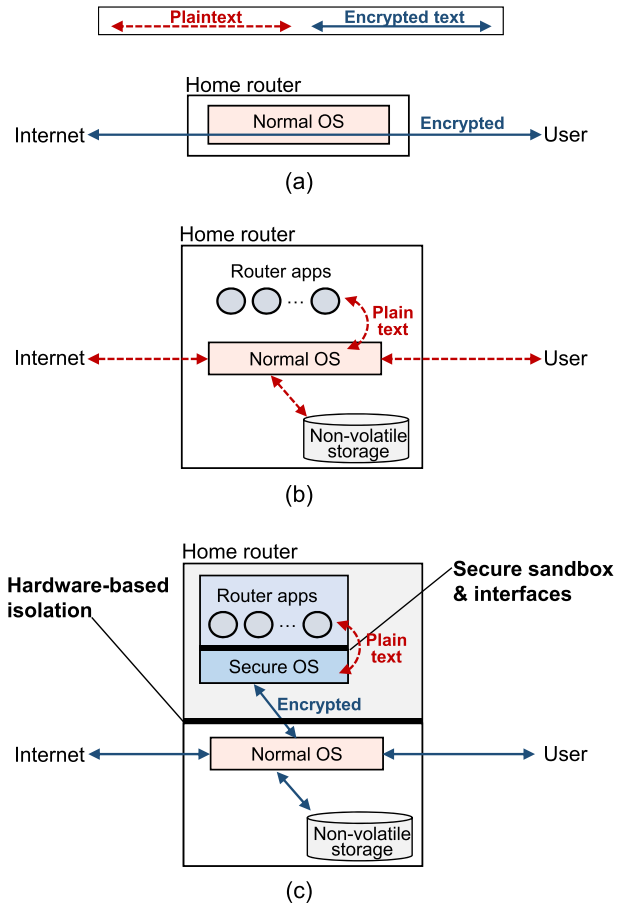


Fig. 1. Data flow of user data in a) traditional router, b) smart router, and c) S2Net.

packet inspection (DPI)-based filtering, which is used to prevent access to illegal/harmful contents, cannot work with HTTPS connections [8]. Therefore, customers are facing a difficult dilemma: *Should we embrace the new features of smart routers while putting our privacy at risk? Or, should we conservatively preserve our privacy while sacrificing functionality?*

Resolving this dilemma, however, is challenging. One particular difficulty lies in that both router apps and the underlying OS are untrusted. Private data can be leaked by the compromised OS (e.g., network stack) even when none of the apps are malicious, and vice versa. Consequently, the previous approaches in trustworthy computing, which assume that either the OS or the apps are trusted, cannot be applied directly, even though some of them rely on modern hardware security technologies such as Intel SGX and ARM TrustZone [9], [10], [11], [12].

This paper resolves the above dilemma by using a novel system named Secure and Smart Network (S2Net), which is designed for ARM-based smart home routers. The objective of S2Net is allowing the router apps to process personal data while ensuring that any personal data are not leaked by either the router OS or the apps. To achieve this goal, S2Net has the following three key features:

- We propose a secure OS in S2Net, and place the proposed secure OS and untrusted router apps in the secure world by leveraging ARM TrustZone technology [13]. S2Net confines the vulnerable router apps (called applets) in sandboxes to prevent malicious behaviors. In S2Net, we also design secure interfaces through with the router apps can interact with the secure OS.
- The secure OS is designed to include a session manager to empower the home router to distinguish and manage multiple sessions that belong to different users and share the same domain. We also propose a novel Same-Origin-plus-User Policy (SOUP) so that the session manager can protect the personal data of an individual, which is not possible when using Same-Origin Policy (SOP).
- We develop a crypto-worker model that provides an abstraction layer of cryptographic tasks performed by a heterogeneous multi-core system. Accordingly, the implementation details related to the heterogeneous processors are isolated from the session manager. Note that S2Net can create a bottleneck at the home router because every data packet is required to be decrypted before being processed and needs to be re-encrypted afterward. Based on this notion, the crypto-worker also distributes cryptographic tasks to multiple computation modules considering their loads.

By using the aforementioned features, S2Net can preserve the privacy of user data in home routers, as shown Fig. 1. All private data is supposed to be encrypted when exchanged between end nodes (i.e., users and remote servers) as shown in Fig. 1a. If the private data is decrypted in smart router OS, the data can be leaked easily (Fig. 1b). However, the private data in S2Net can only be decrypted inside a secure OS and then passed on to router apps through secure interfaces (Fig. 1c).

To the best of our knowledge, S2Net is the first of its kind system that enables both strong privacy and rich functionalities in home routers. Our contributions in this paper are summarized as follows:

- We characterize the dilemma in choosing between functionality and privacy of smart home routers, and then show that the current mechanisms fall short in resolving it. (Section 2).
- We present the design of S2Net, a novel system for home routers, and its key components. (Section 4).
- We describe our implementation of S2Net on ARM and the techniques developed to optimize the system performance. (Section 5).
- We report the evaluation results of various benchmarks and real apps, to show that our implementation is able to mitigate the overhead of S2Net and meet the requirements of various router apps. (Section 6).

2 Motivation and Background

In this section, we introduce the smart home router and ARM TrustZone as the background and motivation for our work.

2.1 Smart Home Router

In recent years, the home-router industry has reinvented home routers to be *smart*. Compared to traditional routers,

these new smart home routers have more powerful hardware, much richer functionalities, and are programmable – allowing the addition of new applications [4], [14], [15]. These applications can be either *network middle-box services* or *local services*, such as:

- *Network and service acceleration* This type of service includes technologies such as Web caching, prefetching, or application-specific proxies. For example, a router may download a movie on behalf of a user, and the user can immediately watch the movie when back home. This is called *offline downloading* [16].
- *Unified security for home* Smart home routers may inspect the network traffic to detect phishing, virus, and malware for *all* home devices (e.g., DPI-based filtering). This is particularly useful for small-factor devices such as low-end phones and home IoT devices that often do not have or cannot run sophisticated security software [17].
- *Home IoT hub* With the emergence of various home IoT devices such as smart light bulbs and thermostats, there is a need for a local, persistent hub to control and manage all the home IoT devices. Many smart home routers are designed as such a home IoT hub. They also provide remote access, which enables users to use a smartphone to check and control their home IoT devices from outside [14], [15].
- *Home entertainment center* With internal or external storage, smart home routers may also be a file server for family photos, videos, music, and movies, and can also act as a home entertainment center to stream various media content to other devices [15].

In order to support such rich and flexible functionalities, smart home routers run a complex, full-fledged OS (e.g., Linux, OpenWrt [18], or Android) and provide rich frameworks for developing third-party plugins. However, the openness and flexibility of smart home routers impose big security problems; these adopted OSes have vulnerabilities that can be compromised. We have already seen that vulnerabilities have been reported frequently in these OSes, from kernel and sub-systems to other key libraries [5]. Moreover, third-party plugins (router apps) may also contain vulnerabilities and thus may not be fully trusted. In conclusion, the reality of smart homes is that both the OS and applications cannot be trusted. Therefore, previous works on trustworthy computing, which assume that at least one of them is trusted, cannot be directly applied [9], [12], [19].

Coincidently, the rise of privacy concerns even calls for end-to-end encryption for all Internet traffic, which, however, effectively disables any new functions on packet processing. mcTLS [10] and BlindBox [11] have been proposed to enable network functionalities even on encrypted data, assuming that middleboxes are trusted. However, home routers could be compromised more easily, unlike professionally administered middleboxes. Consequently, users face a difficult dilemma between embracing the new features of smart home routers and putting their privacy at risk.

2.2 ARM TrustZone

ARM TrustZone is the hardware architecture of ARM for trusted computing [13]. It is widely available in many ARM processors including Cortex-A and Cortex-M processors. The heart of ARM TrustZone is a partitioning of hardware and software resources into two "worlds" – a *Secure world* for the security subsystem and a *Normal world* for everything else. Each physical processor core provides two virtual cores that respectively run in the two worlds, in a time-sliced fashion. The two worlds are separated by hardware access control. Each world is an isolated runtime environment, and has its own resources including memory, registers, cache, controllers, and interrupts. As a result, software programs running in different worlds are strongly isolated. Depending on the configurations of the system, a resource can be partitioned between the two worlds (e.g., memory), shared by them, or assigned to one world only (e.g., peripherals).

When an ARM system boots, it always enters the Secure world first. This design ensures that a secure bootloader can provision the system resources before any untrusted code (e.g., an OS for the Normal world) gets a chance to run. For example, the secure bootloader allocates a range of physical memory for the Secure world only, programs the interrupts and DMA controllers, and loads a secure OS. Subsequently, the secure code yields to the Normal world to run an untrusted commodity OS.

3 Threat Model and Assumptions

System Architecture. We assume that a smart home router is ARM-based and supports ARM TrustZone.[1] It is assumed that the ARM system-on-chip (SoC) hardware including ARM TrustZone is correctly implemented and the booting process is trusted. Thanks to the hardware-based protection provided by ARM TrustZone, we trust that the secure OS running in the Secure world is not compromised. On the other hand, we do not trust the commodity OS (e.g., Linux, OpenWrt, or Android) running in the Normal world. We assume that the applets can be comprised even though they are not designed to be malicious.

Cryptographic Functionalities. We assume that the apps in the clients support cryptographic protocols when they are communicating with the router or a remote server. As a default protocol, the transport layer security (TLS) used by HTTPS and datagram TLS (DTLS) are considered in this paper.

Range of Protection. We do not prevent Denial-of-Service (DoS) attacks. A malicious OS in the Normal world may drop network packets, reset network connections, or delete stored data. As long as we allow the untrusted OS in the Normal world to access system resources, these attacks cannot be prevented. We also assume that attackers do not have physical access to a router, and therefore our system is vulnerable to certain physical attacks such as memory attacks.

The attack scenarios considered in this paper are as follows:

1. ARM TrustZone was first introduced and implemented in ARM1176JZ(F)-S processor, which was based on ARMv6Z architecture [20]. Its successor, ARMv7-A architecture, was already the most widely deployed architecture for mobile devices.

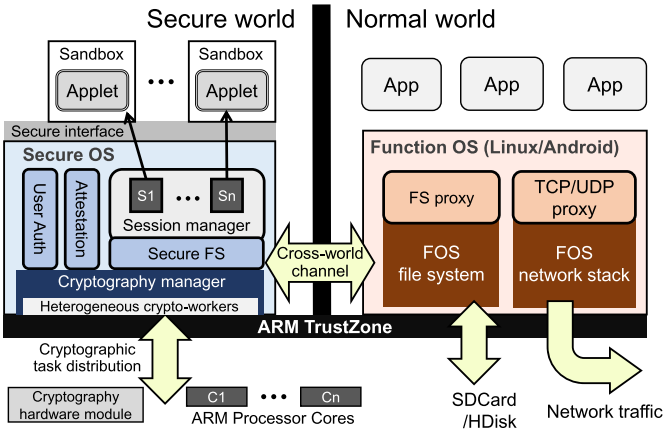


Fig. 2. System architecture of S2Net.

- *Memory disclosure and code injection*: A compromised OS may try to read data from the memory space allocated to an app in an inappropriate way.
- *Malicious network stack*: Since the private data processed by apps is passed to a network stack, the data can be leaked by a malicious network stack.
- *Malicious file system*: The data written by an app can be leaked by a compromised file system.
- *Leaking data to remote server*: A compromised app may try to send private data to remote servers.

4 S2Net System Design

In this section, we present the S2Net system design, including its architecture and key components.

4.1 System Architecture

As shown in Fig. 2, S2Net is based on ARM TrustZone, which partitions software and hardware into Secure and Normal worlds. The conventional router OS, e.g., Linux or Android, is placed in the Normal world; hereafter, we refer to it as the *function OS* (FOS). FOS retains most of the responsibilities of a conventional router OS, namely driving all peripherals (e.g., hard disks, SD cards, and network interfaces), maintaining Internet connectivity, and performing packet forwarding. FOS is usually fat and vulnerable to attacks.

Inside the Secure world, S2Net employs a thin *secure OS* (SOS) such as a microkernel,[2] which is securely booted into the Secure world by the processor. The main components of S2Net deployed in the SOS leverage FOS's network stack and file system to communicate with clients/servers and to manage persistent storage devices, respectively. Although personal data may be passed to the FOS or stored in a storage device managed by FOS, they are always *shielded* by encryption. Therefore, all the personal data can be decrypted and processed inside the SOS. To do so, a client should involve a secure protocol when accessing a service in an S2Net router, e.g., TLS/DTLS. Non-TLS/DTLS traffic will not be passed to the SOS; instead, it will be routed to its original destination as done by traditional routers.

2. A microkernel has a small Trusted Computing Base (TCB) and thus can be secured through formal proof [21].

Next, we elaborate the main components in S2Net, namely the cryptography manager, narrow secure interface, session management, secure storage, and attestation.

4.2 Cryptography Manager

The cryptography manager internally maintains multiple crypto-workers, which provides an abstraction layer of cryptographic tasks performed by a heterogeneous multi-core system, with the aim of hiding the implementation details related to the heterogeneous processors from the session manager and the SFS. The other important role of crypto-worker is to parallelize the computations in order to reduce the high computation cost of cryptographic functions. The cryptography tasks requested by the session manager or the SFS are distributed to the workers by considering the current computational load of the workers. The implementation details of the cryptography manager on a heterogeneous multi-core system are described in Section 5.2.

As a cryptographic secure protocol, we use TLS/DTLS protocol because TLS/DTLS is widely used for Internet services. It comprises a *handshake protocol* for session establishment and a *record protocol* for data exchange. It realizes the following three security properties: 1) *Entity authentication.* During the handshake, the client authenticates the server by verifying that a valid certificate links the server's domain name and the public key. 2) *Data secrecy.* The TLS/DTLS end-points negotiate a symmetric *session key* during the handshake and use this session key to encrypt/decrypt *records* (application data blocks). 3) *Data integrity and authentication.* The session key is also used to generate a message authentication code (MAC) for each TLS/DTLS record. The endpoints can use this MAC to verify the origination of a record and its integrity.

4.3 Session Manager

The session manager is the core component of S2Net in a smart home router. In the session manager, we design and implement the following two important functions: 1) establish the TLS/DTLS connection to the client/the remote server; and 2) manage the life cycles and data flows of applets over the TLS/DTLS connections to prevent malicious behaviors of applets.

4.3.1 Connection Establishment

To separate the en/decryption process of TLS/DTLS from FOS, a *TCP/UDP proxy* is installed in the FOS as shown in Fig. 2. The job of the proxy is simple. It listens to the pre-configured TCP/UDP ports (e.g., 443 for HTTPS) and accepts a TCP/UDP connection from a client. Then, it forwards all payload data to the session manager in SOS through a *cross-world channel*. The entire payload data is annotated with the TCP/UDP connection metadata, i.e., the 5-tuple of [src-ip, src-port, dst-ip, dst-port, protocol]; thus, the session manager can have enough information to demultiplex data for different TCP/UDP connections.

If the router participates in a TLS (or DTLS) session as a middlebox (e.g., a Web caching proxy), the session manager creates a TLS session to the genuine remote site on behalf of the client. If the session between the remote site and the router is established successfully, the session manager may

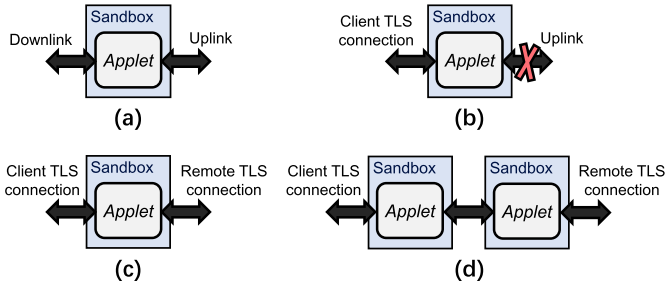


Fig. 3. Data flow of applets. (a) Basic communication model. (b) Single applet for a local service. (c) Single applet for a middlebox service. (d) A chain of applets for a middlebox service.

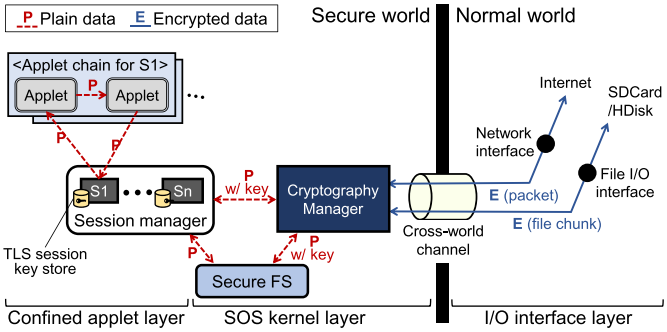


Fig. 4. En/decryption data flow in S2Net.

dynamically sign a certificate for the remote server domain (i.e., Server Name Indication [22]) using a router-specific certificate and return the signed certificate to the client. In this way, the S2Net router becomes "man-in-the-middle" of a TLS session between the client and the server.

The router-specific certificate is a unique certificate on each S2Net router, which is pre-installed in the SOS. We require the client to add the router-specific certificate into the trusted certificate authority (CA) list to accept the certificate signed by the router. Compared to the previous works (e.g., BlindBox [11]), by using the router-specific certificate, the computational overhead can be decreased significantly (see Section 6 for details). Moreover, no modification on the protocol stack is required. However, there are the essential prerequisites: the private key of the router-specific certificate must be secured, and the key should be managed by a trustworthy entity. In an S2Net router, the router-specific key is encrypted/stored inside the secure file storage (Section 4.5), and the SOS is verified/protected by ARM TrustZone (Sections 4.1 and 5.1). Thus, a user can start a secure communication by trusting the router-specific certificate.

4.3.2 Applets and Data Flow Management

Based on the domain name and the port number, the session manager looks up the corresponding configuration to load one or more applets and runs them in dedicated sandboxes (Section 4.4). The session manager also controls the data flow of applets by mapping their upstream/downstream channels to various entities. Fig. 3 illustrates a few typical examples. For a local service that is handled by a single applet, the session manager maps the downstream channel to the client TLS connection, while connecting the upstream channel to null (Fig. 3b). For a middlebox service, the processed user request may need to be forwarded to the genuine remote site. Hence, the upstream channel is mapped to the remote TLS connection (Fig. 3c). Finally, in a more complex case where the a middlebox service may contain a chain of several applets as shown in Fig. 3d, the session manager will connect the upstream channel of an applet to the downstream channel of an *uplink* applet. The downstream channel of the first applet is mapped to the TLS connection to the client, while the last applet may connect its upstream channel to the remote TLS connection.

Clearly, the S2Net session manager ensures that the personal data, after being processed by any applet, can only either be return to the client or be forwarded to the intended remote site as indicated in the original TLS/DTLS handshake. In other words, S2Net prevents an applet from establishing a connection to any other (potentially malicious) servers. Moreover, the personal data cannot be leaked as plain text. Fig. 4 shows three en/decryption layers that any data should pass through. Even if an applet might be compromised, all the data are encrypted in the SOS kernel layer by the session manager. Hence, the personal data passed to the untrustworthy I/O interface layer will be always encrypted, and the attacker cannot read any data. Consequently, an applet cannot cause sensitive information leakage even if it may be malicious.

By default, an applet instance is created when a TLS session is established and destroyed when the TLS session is torn down. However, a user may change this behavior by making an applet instance as *background*. A background applet is useful if a user wants to delegate some tasks to the router. For example, a user may want to download a large file and may delegate this job to the home router, so that the downloading will continue even if the user has disconnected and powered off the phone. One challenge for background applets is that later on when the user connects back, the same applet instance should be attached to the newly established TLS session. In this paper, we adopt a *Same-Origin-plus-User* policy (SOUP) to uniquely map later TLS sessions to the background applet of a user. We discuss SOUP in the next subsection.

4.4 Secure Interfaces

To isolate applets that process decrypted personal data, S2Net provides very simple and narrow secure interfaces with a sandboxing mechanism. As shown in Fig. 5, only two necessary interfaces are opened to an applet: one for communication and the other for file storage. Hence, the data processing pattern of an applet is extremely simple. Each applet has only two communication channels: a *downstream* channel to the client and an *upstream* channel to the remote server (see Fig. 3a). The file storage interface is for applets to access files on the local storage with strict access permissions (Section 4.5). Moreover, all the parameters are strongly typed and validated by SOS to eliminate any potential buffer overflow vulnerabilities. Since the interface channels can be assigned only by the session manager (Section 4.3), applets can read/write data only from/into the given channels. As a result, applets are strictly restricted, and can do just their data-processing work but no other unwanted things (e.g., making a new connection with a remote server).

Communication interface:

```
int read_channel (int direction, void* buffer,
                  int size);
int write_channel (int direction, void* buffer,
                  int size);
int set_type (bool background);
```

File storage interface:

```
int open (char* fname, char* mode);
int read (int fd, int& offset, int& length,
          void* buffer);
int write (int fd, int& offset, int& length,
          void* data);
int close (int fd);
```

Fig. 5. Interfaces of S2Net sandbox.

4.5 Secure File Storage and SOUP

S2Net provides a secure file storage (SFS) and allows an applet to store persistent data securely. The design of such a secure file storage needs to address the following two challenges: First, since SOS shares the same storage devices with the untrusted FOS, the stored personal data should not be able to be interpreted or modified by the FOS. Second, SFS should be able to manage the access rights, so that an unauthorized applet cannot the read personal data stored by another applet.

Sealed Storage. To prevent the FOS from accessing personal user data, we use cryptography and authentication to *seal* the data in storage. SFS is based on a simple *sealed storage* [23], which maps a file/folder in the SFS to a file/folder in the FOS file system. Every file transferred to the FOS's file system is first encrypted using a private storage key (derived from a root key stored in a hardware module, Section 5.1). Moreover, the SFS maintains metadata containing the hierarchical hash tree of the files to protect against unauthorized modification and replay attacks. Since the metadata file itself is also encrypted and signed by the SOS, FOS can neither read nor edit the personal data.

Same-Origin-Plus-User Policy (SOUP). To prevent unauthorized applets in the SOS from accessing the personal data, SFS needs to put all the files generated by one applet into a *compartment* and attach each compartment to a separate security label corresponding to the applet. One simple labeling method is using the domain name. This approach is similar to the *Same-Origin Policy* (SOP), which widely used in the existing Web browsers. However, SOP is not suitable for home routers, which are supposed to serve multiple users; even when two users access the same domain name, their personal data should not be mixed up. In S2Net, we address this issue by applying a new security policy called *Same-Origin-plus-User Policy* (SOUP). To label a compartment in SFS, SOUP uses the username or identifier and the domain name, together.

One way to obtain the authenticated username is to request a client certificate during the handshake phase. Although both TLS and DTLS are standardized as the channel security for computers, smart devices, and IoT devices [24], TLS/DTLS client certificates are not widely used in practical life. Therefore, in this paper, we leverage a local TLS/DTLS service (i.e., the User auth module in Fig. 2) to authenticate a user. Once the user is authenticated, a user label is generated for tagging all the data of the user (Section 5.1).

TABLE 1
Lines of Code (LoC) of our Implementation

Component or applet	LoC
Crypto-workers *(incl. CAAM driver)*	9,375
Session manager *(incl. proxy server/client)*	31,033
Secure storage *(incl. FS proxy and SFS library)*	8,875
Web caching applet	334
Content filtering applet	750
Home file server applet	240
Total	*50,607*

4.6 Attestation

We propose to use attestation [25] to detect whether an applet is compromised or not. For the attestation, the SOS calculates the hash of the memory block (e.g. SHA256 hash) and compares the hash to a pre-stored and certified record to decide whether the applet is compromised or not. An existing work [26] has shown that ARM TrustZone can be used for TPM functions, and therefore it can provide roots of trust required by the attestation.

5 Implementation and Optimizations

We implemented an S2Net prototype on the NXP SABRE Lite i.MX 6Quad development board, which has a 1 GHz ARM Cortex-A9 quad-core CPU and 1 GB RAM. The CPU supports ARM TrustZone and a set of other advanced security features including high assurance boot, cryptographic cipher engines, random number generator, secure RTC, and tamper detection [27]. We use Genode[3] release 15.02 as the SOS and Linux version 3.10.17 as the FOS, respectively. Table 1 shows the LoC of the key components and the sample applets.

5.1 S2Net System

TLS/DTLS Splitting and HTTP/2. To split a TLS/DTLS session between the Secure world and the Normal world, we port OpenSSL version 1.0.1i [28] onto Genode and run a TLS/DTLS proxy in the Secure world. We leverage the network stack of Linux in the Normal world to handle all TCP/UDP-layer activities. Before a client sends an HTTPS request to a Web server, it first tries to make a TCP connection with the Web server. The TCP/UDP proxy shown in Fig. 2 handles TCP handshakes and accepts the TCP connection on behalf of the Web server. When the first data packet (i.e., the TLS client *Hello* message) arrives, the TCP/UDP proxy forwards it to the TLS/DTLS proxy in the Secure world. The TLS/DTLS proxy handles all the TLS handshakes with the client. The TCP/UDP proxy in the Normal world also makes a new TCP connection to the Web server and forwards all the data packets to the TLS/DTLS proxy, which also does TLS handshakes with the Web server. After that, the client can send HTTPS requests to and receive HTTPS responses from the Web server. All the HTTPS data packets are decrypted and then re-encrypted by the TLS/DTLS proxy (i.e., by the secure protocol module) in the Secure world.

3. Even though Genode microkernel [21] has been used in this study, any other microkernel may also be compatible with S2Net.

Cross-World Communication. We use shared memory for the two worlds to communicate with each other. During initialization, the FOS allocates the memory block used for cross-world communication and tells the base address of the memory block by writing the base address into a register and calling the *SMC* instruction. Then, the SOS reads the base address from the register. The shared memory is organized as ring buffers with necessary metadata to describe the content in each data slot.

Secure File System. We build our secure file system using the Secure Block Device (SBD) library [29], an open-source library that applies cryptographic confidentiality and integrity protection to block devices. The library also includes a block cache to improve the performance. We ported and modified its code so that it can run in the SOS to provide file I/O interfaces to the session manager and work with our FS proxy in the Normal world over the cross-world communication channel. The FS proxy leverages the existing file system (ext4/fat32) of Linux to store the encrypted data into files. Among the authenticated encryptions supported by SDB, we choose to use Offset Codebook Mode (OCB) in our implementation due to its good performance [29].

Secure Booting and Key Management. Thanks to the High Availability Boot (HAB) library on i.MX family chipsets, the development board can authenticate a program image using a digital signature (first boot stage) [30]. Therefore, only an image with a certificate signed by the private key can be booted on the board while the public key is permanently burnt inside the board. In our prototype, U-Boot is used as a bootloader (second boot stage). U-Boot can be configured to authenticate the given kernel image before it boots on it (third boot stage). Hence, only the authenticated kernel image can pass the checks and boot up.

The i.MX6 processor has a permanent key named One Time Programmable Master Key (OTPMK), which is burnt by NXP before shipping their processors. The OTPMK is protected by, and is only accessible by, a hardware module named Cryptographic Acceleration and Assurance Module (CAAM). Based on the OTPMK, CAAM provides a protocol to protect personal user data. The data is encrypted using a randomly generated encryption key; the encryption key is also encrypted by a key derived from OTPMK. This encrypted data-key pair can be stored together inside the non-volatile storage. Since the encrypted data and key can be decrypted only by CAAM, we can securely store our root keys (such as encryption key for metadata of secure storage) and the router-specific certificate.

User Authentication. Since we do not trust the FOS, we cannot simply use the IP or MAC address of a client to authenticate the users, because it can be easily falsified by the FOS. Instead, we choose to use a method based on HTTP cookie to perform user authentication. The first time when a client sends an HTTP request through the S2Net system, a user login/register page is popped out. After the user logs in, a token is generated for the user and inserted into the cookie stored under the router's domain. After that, when the user sends an HTTP request, the S2Net system redirects the user to the router's domain. Then, the user authentication module retrieves the token from the cookie to identify the user. Only for the authenticated users, the S2Net system will generate a one-time authentication code (OTAC) and redirect the user to the remote server with the code (since the content is encrypted by HTTPS, OTAC cannot be read by the FOS or other intermediate nodes). Only when the S2Net system recognizes the valid OTAC, the HTTP request will be forwarded to the remote server. Since now the router would insert the token into the cookie stored under the domain of the remote server, further redirections will not be needed on the same domain. This cookie-based approach works well, as all browsers support cookies.

For any other devices/applications that are not compatible with the HTTP cookie-based authentication including IoT devices, a client certificate is requested to authenticate a user. IoT devices supporting DTLS (or TLS) are supposed to have their own client certificate and present the certificate when they do the DTLS handshake [24]. Therefore, the S2Net system can obtain the client certificate and authenticate the user during the handshake phase.

5.2 Performance Optimizations

S2Net introduces two major overheads: 1) world-switching between the Normal and Secure worlds caused by frequent cross-world communications, and 2) various computation-intensive cryptography operations, especially the decryption/encryption required by TLS/DTLS splitting. Thus, it is critical to optimize the system performance, particularly when running on ARM platform with a weak CPU. Our implementation can fully utilize the hardware capability of the i.MX 6Quad board to mitigate the overheads introduced by S2Net. To do so, we develop techniques to 1) minimize the number of world-switching, and 2) parallelize all the cryptography tasks.

Minimizing World-Switching. Switching from the Normal world to the Secure world or vice versa is not free. Each world-switching takes about 29 microseconds to save and restore the full switching context. In the S2Net system, the Secure world needs to receive every network packet from the Normal world and also needs to send them back to the Normal world. Therefore, if those cross-world communications are not properly handled, they may cause a large number of world-switching and thus incur a high overhead (e.g., meaningless world-switching should be prevented when there is no packet to send).

We minimize the number of world-switching through two techniques. First, we assign dedicated CPU cores to the two worlds to avoid CPU-core sharing among the two worlds. Since all computation-intensive data decryption and encryption happen in the Secure world, we assign more cores to the Secure world compared to the Normal world. Without loss of generality, in the prototype of S2Net, three and one cores are assigned to the Secure and Normal worlds, respectively.[4] Without sharing any CPU core, the FOS and the SOS can do their jobs simultaneously, and thus we are able to avoid unnecessary world switching caused by CPU-core sharing.

Second, we employ a *polling* mechanism in cross-world communications to further reduce the number of world-

4. The assignment of CPU cores can vary according to the number of total CPU cores, the number of applets the user wants to use and their computational load. A minimum requirement would be at least one CPU core per each world.

```
int request_crypto(unsigned action, // encryption, decryption, ...
                   unsigned char * data_buf, // data to be processed
                   unsigned size_data,
                   unsigned char *iv,
                   unsigned iv_size, // an initialization vector and its size
                   unsigned char *key,
                   unsigned key_size); // a key and its size
```

Fig. 6. Interfaces of the cryptography manager in S2Net.

switching. Instead of letting one world write data into the shared memory and then notify the other world to fetch the data, we use a dedicated thread in both worlds to monitor the changes of the shared memory. Such a thread keeps polling a flag of the shared memory. When the Secure or Normal world wants to send a data packet to the other world, it first writes the data packet into an empty data slot in the shared memory. Then, it changes the corresponding flag of the shared memory. As a result of the continuous polling in the other world, the other world can immediately detect the data-transmission intention and fetch the data packet from the shared memory for processing, without any world-switching.

Crypto-Worker. To optimize and speed up cryptographic operations in the S2Net system, we developed a crypto-worker model in the cryptography manager. The developed crypto-worker model provides an abstraction layer of cryptographic tasks performed by a heterogeneous multi-core system. Thus, the implementation details related to the heterogeneous processors are isolated from the session manager and the SFS, as shown in Fig. 2.

As we assign three cores to the Secure world, we create three homogeneous crypto workers (CW1) to serve all the cryptographic tasks, each running on one of the three cores. Examples of cryptographic tasks include encrypting or decrypting a data packet or a data chunk of a file. We also leverage the cryptographic hardware of the board, i.e., CAAM, to further boost the system performance. To enable cryptographic functions provided by CAAM, we port the CAAM driver for Linux into Genode. Since the system architecture of Linux and Genode are totally different, Linux kernels system calls in the CAAM driver are remapped to Genodes system function calls. By utilizing the CAAM driver, we run a different crypto worker (CW2) on the CAAM. When there is a request for cryptographic task from the session manager or the SFS, the crypto worker, CW2, makes a job descriptor including the type of cryptographic algorithm, address/size of the data buffer and key. Then, the job descriptor is passed to the CAAM. After the requested cryptographic job has been done by CAAM, the result is pulled and returned to the cryptography manager. By identifying the different computational powers of the heterogeneous crypto-workers, CW1 and CW2, the cryptography manager schedules and dispatches the cryptographic operations to these four crypto-workers (three CW1s and one CW2 on the three CPUs and CAAM). It keeps monitoring the load of the crypto-workers and dispatches a new cryptographic task to the crypto-worker that has the lowest load. With these parallel crypto-workers, we can significantly improve the performance of our S2Net system, as we will show in Section 6. Fig. 6 shows the interfaces used by the session manager to request the crypto workers to perform the cryptographic tasks.

5.3 Sample Applets

To demonstrate how to add new functionalities on our S2Net system, we build three sample applets: HTTPS Web cache, content filtering, and a home file server.

HTTPS Web Cache. We develop an HTTPS-Web-cache applet on S2Net. Under the SOUP policy, a user is unable to use the cached data already downloaded by other users. However, the SOUP policy can be useful in some cases where users do not want to share the cached data even between family members. For example, parents may not want to share their data or browsing history, which can potentially contain age inappropriate data, with their children.

When a user sends an HTTPS request to a remote server, the cache applet will intercept the request and check if a cached version of the requested document is available in the secure file storage. If yes, the cached version is returned to the user without going all the way to the remote server. Otherwise, the web-cache will forward the request to the remote server and get the response. If the response is cacheable (specified by the cache control header field), the applet stores it in the secure storage. According to SOUP, a cached file should be authenticated based on both the user token and the certified domain name of the remote server. Therefore, any personalized data such as web browsing history will not be leaked.

Content Filtering. Content filtering or parental control is widely used to protect people from visiting harmful/illegal websites. Unfortunately, content filtering services cannot work on HTTPS sessions, because the encrypted URLs and contents cannot be read (although the domain is plaintext). In the S2Net system, however, a content filtering applet can work with decrypted URLs and contents.

When a request arrives, the content filtering applet checks whether the URL of the request can be found in a local URL-blacklist consisting of all the websites that are not allowed to be accessed. If the URL is in the blacklist, the applet will block the request. We use a Bloom filter [32] based approach to speed up the URL matching over the blacklist. In addition, we also implement DPI filtering based on regular expression to recognize (potentially) harmful words and patterns.

Home File Server. The home-file-server applet acts as a local service without connecting to a remote server. It leverages the local secure storage of the router for family members to store and share various files from different devices. It provides a simple Web interface for uploading and downloading files.

6 Evaluation

In this section, we evaluate the security and performance of S2Net.

TABLE 2
Data Security Analysis of S2Net Compared to Related Works

Privacy threat model	S2Net	mcTLS [10]	BlindBox [11]	TrustShadow [12], [31]	Haven [9]	Genode [21]
Compromised FOS						
- Memory disclosure, code injection	●	○	○	●	●	●
- Malicious network stack	●	○	○	-	-	-
- Malicious file system	●	○	○	●	●	-
Compromised apps / MBs						
- Leaking data to remote server	●	○	○	-	-	-
Overhead	moderate (one encryption + decryption)	high (depends on # of contexts, MBs)	high (up to 24x)	moderate (one encryption + decryption for files)	low	low

Attacks can originate from both a compromised FOS and applications (apps) / middleboxes (MBs). (●: protected, ○: protected, but no protection provided for decrypted data, -: not protected).

6.1 Security Analysis

We first summarize the security of S2Net in Table 2 based on our threat model (Section 3). Similar to other approaches leveraging hardware-based protection [9], [12], S2Net can protect from memory disclosure and code injection attacks, thanks to the hardware-based memory isolation. Furthermore, in S2Net, malicious network stacks and file systems cannot leak private data. Since all the components related to the secure protocol are separated and isolated from the FOS, private data can only be decrypted in the secure world. Lastly, S2Net can prevent compromised router apps from exposing the private data due to the restricted interface of the secure sandbox and the session manager. Although router apps are allowed to process plaintext, they can neither pass the data to the FOS nor open a new connection to unauthorized entities.

6.2 Performance Evaluation

We evaluate the performance of our implementation on the i.MX 6Quad development board, focusing on the performance of the key operations of S2Net and the end-to-end performance of the sample applets.

Encryption and Decryption. We measure the throughput of encrypting (E-) and decrypting (D-) data chunks. All the results are averaged over 100 runs.

First, we compare the performance of a CW1 running on a CPU core and the performance of a CW2 running on CAAM. Fig. 7 shows the results for different data-chunk sizes. The performance of the CW2 running on CAAM is slightly lower than that of the CW1 in the case of 2 KB data chunk. However, the throughput of the CW2 increases when the data-chunk size is larger and becomes even higher than the throughput of the CW1 in the 8 and 16 KB cases (for DES algorithm). This is because the CW2 has an additional overhead caused by the CAAM driver (i.e., the time to make a job descriptor, push/pull the result into/from CAAM). Since the overhead increases in proportion to the number of data chunks, the CW2 can work more efficiently with data chunks of the bigger sizes.

Second, we evaluate how our parallel cryptography workers may speed up the total cryptography performance. To do this, we encrypt and decrypt 128 data chunks of 8 KB size to see the performance enhancement in parallel execution. At most, we have four crypto workers: three CW1s on three CPU cores, and one CW2 on CAAM. Fig. 8 shows the results with different numbers of CW1s, with/without the CW2 running on CAAM, in various cipher algorithms: AES using 128, 192, and 256 bits length keys running in CBC mode. It is shown that our implementation has negligible overhead in handling requests/responses between the crypto workers and the cryptography manager. Without counting the CW2 on CAAM, we can linearly increase the throughput using multiple cores, with at least a $\times 1.98$ or $\times 2.94$ performance improvement using two or three CW1s, respectively,

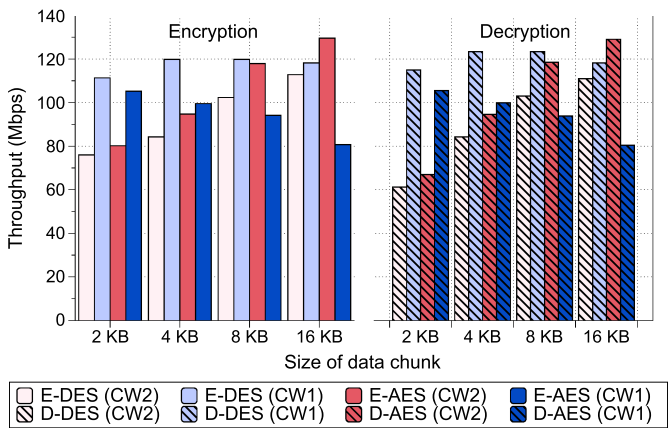


Fig. 7. Encryption and decryption performance of crypto-workers: CW1 on a CPU core versus CW2 on CAAM.

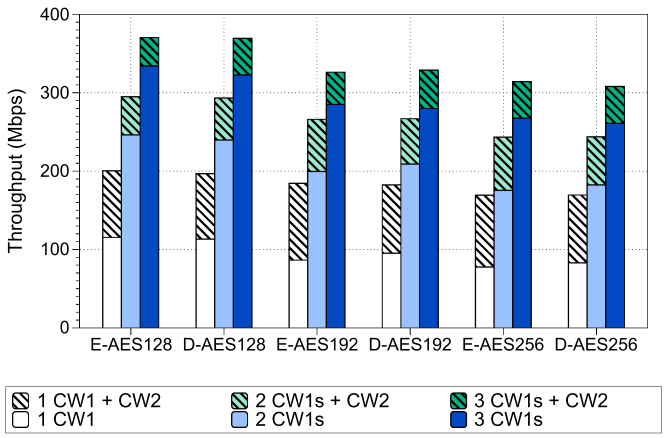


Fig. 8. Performance of heterogeneous crypto-workers in S2Net, up to three CW1s and one CW2.

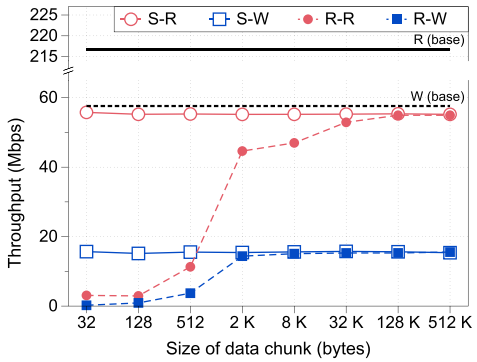


Fig. 9. Throughput of file I/O versus data chunk size.

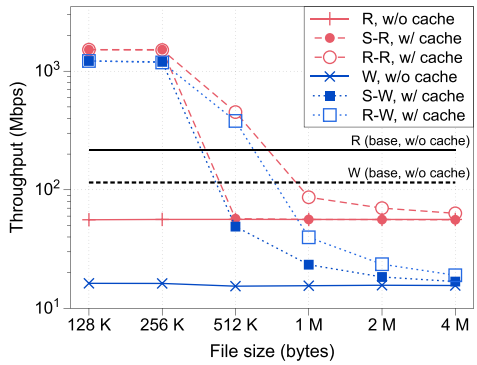


Fig. 10. Throughput of file I/O versus file size, with and w/o cache.

compared to the throughput of the single CW1. With the CW2 on CAAM, the performance of encryption (decryption) is further increased up to 118 percent (104 percent) of a single CW1. An interesting phenomenon observed from the result is that the throughput of CAAM cryptography is increased as the size of the key increases, because the CAAM module already has enough capability provided by its circuit to deal with large keys. On the other hand, the performance of SW-cryptography (i.e., CW1) decreases since more computations are needed for large keys. The benefit gained from using the CAAM module decreases as the number of total crypto workers increases due to the routing and scheduling overhead. However, the system is still able to provide a throughput higher than 360 Mbps, which is enough for practical home network scenarios.

Secure Storage. We measure the file I/O throughput by reading/writing a file from/into the secure storage to evaluate the overhead introduced by the S2Net system. For the evaluation, we use sequential and random access patterns for various sizes of files as in the benchmark applications [33] and [34]. Specifically, a test applet first reads (R) and writes (W) a file of 4 MB as a sequence of data chunks sequentially (S-) and randomly (R-). The size of the data chunk in each read/write call is varied in the experiment to investigate the impact of data chunk size on the throughput. We set the cache size and the data block size of the secure storage to 32 and 2 KB, respectively. In this test, a file access without any en/decryption is used as our baseline.

Fig. 9 plots the I/O throughput with different sizes of data chunks. It is shown that the performance of sequential R/W is almost the same over the entire range of chunk sizes. This is because the sequentially read/written chunks would be contained by the same data block, and therefore each block needs to be loaded from the disk (into the cache) only once. In the random R/W, however, the following R/W can request the data chunk stored in the non-adjacent block (i.e., not cached). Moreover, even the cached data block would be flushed soon due to the limited size of the cache. Therefore, since the same data block would be loaded from the disk multiple times, the random R/W achieves low throughput when the chunk size is small. When the chunk size is larger than the block size, the throughput increases dramatically, because each data block needs to be loaded only once. When the size of data chunk is large (512 KB), the performance of baseline is $3.9\times$ faster compared to the secure-storage, owing to the overhead of en/decryption and data block management in the secure storage.

To assess the impact of the cache in secure file storage, we conducted another experiment in which files of different sizes were used: 128 KB to 4 MB. In the experiment, the chunk size was fixed to 128 KB, which turned out to be an optimal value from the previous results, and the size of the cache was set to 512 KB. Fig. 10 shows the throughput for both sequential and random file accesses. The results of the sequential and random access have been combined in w/o cache cases, since the throughput is almost the same.

We observe from Fig. 10 that the throughput performance is almost the same irrespective of the file size when caching is not used. On the other hand, when caching is employed, the throughput is improved by up to $75.0\times$ for the files smaller than the cache. In particular, the throughput is higher than that of the baseline if the size of cache is large enough to contain the files (up to 512 and 256 KB of files for the sequential and random access patterns, respectively). However, when the file is larger than the cache, the performance decreases because cache misses occur more frequently, leading to loading of data from the disk and decrypting it. Even when the file size is the same as the cache size, some of the data blocks cannot be cached, since the secure file storage requires additional cache space for storing management blocks. Unlike in sequential access, some of the cached data blocks can be accessed again in random access; therefore, the throughput of random R/W is higher than that of sequential R/W.

Overhead of Attestation. To check the overhead of the attestation, we measure the throughput and processing time of SHA256 hashing that is used to generate a 256-bit length signature per applet. We vary the size of the applet up to 8 MB to see how much time will be required to generate a hash. Fig. 11 shows the result. Although the processing time depends on the memory size, the throughput is similar for different memory sizes, which is fast enough to check applets periodically.

Web-Cache Performance. We measure the page load time (PLT) of 11 popular websites in five cases: *w/o proxy*, *proxy w/o web-cache* (cold loading), and *proxy w/ web-cache* (warm loading) for both HTTP protocol version 1.1 and version 2. Fig. 12 shows that our Web-cache applet is able to reduce the PLT of the websites by up to 86.7 percent (82.6 percent) for HTTP/2 (HTTP/1.1) when the Web resources are cached. However, for cold loading, the applet results a longer PLT due to the extra handshake and decryption/encryption overhead.

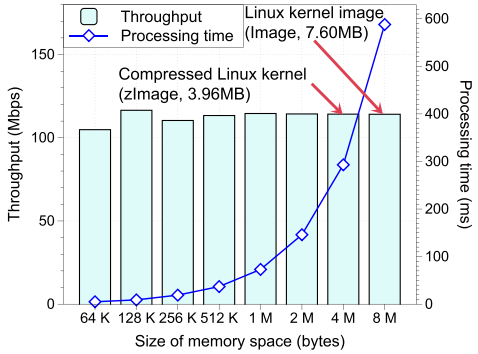


Fig. 11. Performance of calculating SHA256 hash.

One of the main differences between HTTP/2 and HTTP/1.1 is a new binary framing layer that enables multiple requests and responses to be sent over the same TLS connection (i.e., be multiplexed). In the warm loading case, HTTP/2 proxy is faster than HTTP/1.1 because the requests and responses to fetch non-cacheable contents can be served by the existing TLS connection without requiring additional TLS/TCP handshakes. In the cold loading case, however, the browser has to download all data including cacheable files, thus the transmission time increases and the relative benefit of multiplexing decreases, compared to the warm loading case. As a result, HTTP/2 proxy achieves similar performance to HTTP/1.1 proxy.

Content Filtering Performance. The performance of content filtering is measured in two ways: a blacklist-based URL matching and a DPI-based rule matching. Benefiting from the use of Bloom filter, our content filtering applet is able to perform URL matching very fast, independent of the size of the blacklist database. The throughput is as high as 1.3×10^5 URLs per second. It means that we can filter out a URL in less than 8 microseconds, which is negligible. Fig. 13 shows the performance of DPI over strings. The size of the strings varies from 128 B to 16 KB. Although the processing time of DPI for the 16 KB string with 1,000 rules is about just 1.4 seconds (on only one core), it would be efficient if the DPI is applied only on the major parts of the content (e.g., title and abstract of a Web page), because the processing time will increase as the size of a string increases.

File Server Performance. We measure the transmission speed of the file server applet by uploading/downloading a file of 100 MB. The baseline performance measured without SOS is 67 and 69 Mbps for upload and download, respectively. Although the uploading and downloading speeds of the file server applet, which are 16 and 60 Mbps, respectively, are slower than that of the baseline, they are still sufficient for real-time multimedia services such as streaming 4 K videos [35]. The difference between upload and download speeds of the file server applet are caused by the difference between R and W throughput of the secure file storage as shown in Figs. 9 and 10. The interesting observation from the results is that the performance gap is reduced by up to 94 percent compared to the evaluation for the secure storage itself (Fig. 10). This is because secure file transmission protocols, such as file transfer protocols (FTP) over TLS/SSL, basically require en/decryption, even when S2Net is not employed.

Effort to Develop an Applet. General programming on SOS, which is a microkernel-based operating system, would not be easy, because its system architecture is different from that of other popular OSes such as Linux. However, developing an applet for S2Net is not hard, because S2Net handles all the communication details, and therefore applets can just focus on data processing. As S2Net allows very limited interfaces (i.e., only communication and storage), we expect it is not hard for developers to learn to write S2Net applets, even if they are not familiar with microkernels.

7 Discussion

Run-Time Attestation. We implemented only a simple static attestation component, which just computes the hash of the binaries of applets and compares the computed hash to prestored ones in the local storage. Therefore, we can only detect whether the binaries of applets have been changed or not. Ideally, we should also do run-time attestation to check the integrity of the run-time states of applets, which is hard to do and causes additional overhead. However, thanks to the very narrow interfaces provided to applets, even if an applet is compromised at run-time, we can still ensure that the applet cannot leak personal user data.

Speculative Execution Attacks. Since speculative execution attacks, including Meltdown [36] and Spectre [37] threaten modern processors speculation methods, the ARM Cortex-A9 used in our prototype can be similarly affected by variant 1 and 2 of Spectre [38]. The variant 2 can be mitigated by applying kernel patch provided by ARM or invalidating the branch predictor [38]. Compared to the variant 2, the variant

Index	Website
1	google.com
2	youtube.com
3	facebook.com
4	yahoo.com
5	wikipedia.org
6	twitter.com
7	instagram.com
8	netflix.com
9	microsoft.com
10	tumblr.com
11	wordpress.com

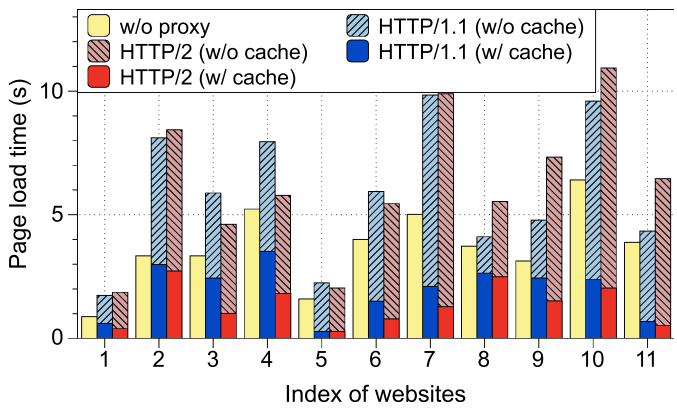


Fig. 12. Page load time of 11 popular websites (lower is better).

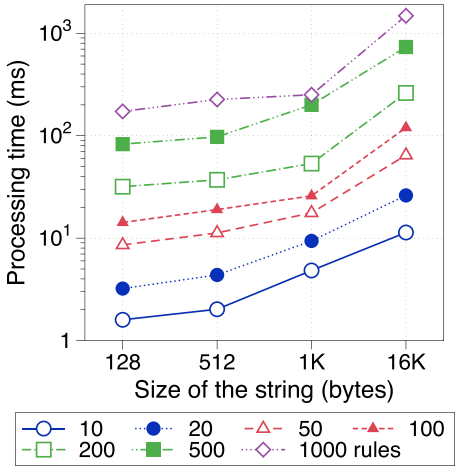


Fig. 13. Processing time of the DPI versus size of string.

1 is harder to mitigate, because vulnerable area needs to be found by manual inspection [39]. Software developers can use the speculation barrier headers provided by ARM to protect their private data from the attacks. As stated in [39] and [40], it is harder to exploit the vulnerabilities on secure space from the Normal world in practice, and any Spectre attacks on Trusted Execution Environment (TEE) are not identified yet. Also, it is difficult for applets to mount Spectre attacks because the applets are designed not to be malicious under the assumption made in Section 3 and attestation is performed. Further, secure sandboxing mechanism makes the attacks harder to succeed [41]. In addition, SOS can be patched more easily compared to the complicated FOSs due to its small codebase. Also, the updated SOS is always verified during the secure boot process to ensure the integrity of SOS.

Connection to Remote Servers. The S2Net system currently prohibits applets from opening any new connection to remote servers. It is useful for minimizing the attack surface; however, it also limits useful functions of applets such as downloading files from multiple servers in the background and automatic updates of applets. The easiest way to enable an applet to make a new connection to remote servers, while ensuring security, would be to launch one applet for each domain (e.g., a file). Another solution could be leveraging the client certificate the user signs a list of allowed domains for an applet with the private key. When the applet tries to open a new connection to a new server, the session manager can decrypt the list and check whether the domain is on the list. A third alternative is to provide an update service to applet. The developer of an applet, which needs periodical updates, can sign the domain and address of the update and attach them to the applet. Since the SOS can decrypt the address using the developer's certificate, it can establish a new HTTPS connection for downloading the update and verify the downloaded update before its use.

Fine-Grained Access Control for HTTP Content. The current S2Net design only supports coarse-grained access control. All applets have full access permissions to the HTTP content they receive from all the HTTP requests and responses. They can not only read but also arbitrarily change all the HTTP headers and bodies. As different applets may require different permissions, it is desirable to utilize fine-grained access control to further regulate the behavior of applets. For example, an intrusion-detection applet only needs to read the HTTP requests and responses but does not need to change them. To this end, we may introduce explicit permission management as mcTLS [10] does. We consider this as future work.

Load Balancing Between the Normal and Secure Worlds. In this paper, we assume that the computational load in the Secure world is higher than that in the Normal world, because the most computation-intensive task would be cryptography and the routing apps that process the user data will be running on SOS. In practice, however, the home router user may want to run applications requiring heavy computations such as machine learning in the Normal world. Hence, the allocation of CPU cores should be flexible and configurable. In the prototype implementation of S2Net, the number is configured in the SOS and the assignment of CPU cores can be modified but requires rebooting the system. Ultimately, it would be possible to assign CPU cores to each world in run-time, because all the CPU cores are originally designed to switch between the Secure and Normal worlds. However, further development is needed for switching worlds safely without any loss of data or security issues, thus we left this as a future work.

Configuration of System Parameters. There are some system parameters such as the number of crypto-workers, the period of attestation, and the number of CPU cores, that can affect the performance of S2Net. Under S2Net architecture, at least one CPU core per world, one crypto-worker should be assigned, and attestation must be performed once, when an applet is launched. Then, users and/or developers can adjust the system parameters considering the number of applets and their computational loads. Regarding the assignment of crypto-workers, the bandwidth of Internet connection and the number of simultaneous TLS sessions would be considered. For example, if a user has a 50 Mbps connection to the Internet, one crypto-worker could be sufficient, as shown in Fig. 8. However, if a higher bandwidth is provided with many TLS sessions, then more crypto-workers will be needed to accelerate the cryptography throughput. Similarly, users can decrease the period of attestation as the number of applets increases. Ultimately, the attestation period would be adjusted adaptively depending on the number of applets currently running on S2Net and their computational loads. Such dynamic optimization process would be able to be applied to other system parameters; we left it as a future work.

8 Related Work

Microkernel and LibOS. Including Genode, there are other microkernels [42] and LibOS [43] approaches that provide software-based isolation among applications. S2Net, however, leverages a hardware-based mechanism to separate the FOS (in the normal world) from the SOS (in the secure world), thus providing much stronger isolation. Additionally, S2Net can leverage the rich features of FOS for home routers, i.e., the entire TCP/IP stack, but without the need to trust FOS.

Trusted Computing. Besides ARM TrustZone, there are other secure hardware such as hardware security module

(HSM) [44], trusted platform module (TPM) [45], and Intel software guard extension (SGX) [46]. HSM and TPM are usually used for protecting secrets (e.g., keys) and for lightweight security computing such as a key generation. SGX is a new mechanism from Intel, which allows executing user-level code within a hardware-protected environment called *enclave*. This paper focuses only on ARM TrustZone, but the general principle developed here may also be suitable for the SGX platform.

A large body of literature has focused on protecting application data from an untrusted OS, by leveraging hardware security capability [9]. However, they assume that the application is trusted, and therefore do not restrict the application behavior. S2Net, however, not only shields user data from the untrusted FOS in the Normal world, but also confines untrusted applets in the SOS inside the secure world.

Secure Middleboxes. Recently, secure middleboxes were proposed to add network functions in HTTPS flows by making the middlebox visible to and controllable by users [10], [11], [47], [48], [49]. For example, Ericsson proposes using an explicit proxy certificate to indicate that a proxy is trusted [47]. Google requires that a client maintains a persistent TLS connection with a proxy to pass session keys out-of-band [48]. Cisco proposes a TLS extension to provide consistent security policies for the client, server, and proxy [49]. Without hardware support, all these solutions run within an untrusted commodity OS, and are thus less secure. They introduce extra complexity, and thus increase the attack surface.

Recently, mcTLS [10] extended TLS to support fine-grained access control with trusted in-network functions. In addition, BlindBox enables deep packet inspection over encrypted traffic through a new protocol and new encryption schemes [11]. All these works assume that the middlebox OS and applications are trusted. However, as we discussed earlier, such assumptions may be invalidated in current home router designs. Only one recently published work [50] considers (potentially) malicious OS and applications and provides a concept for the system design; however, it lacks evaluation for practical applications.

Sandboxing Mechanisms. Many sandboxing mechanisms at different software-stack levels have been proposed in literature. Examples include those from virtual machine [51] and library OS [52], to containers [53], [54], application virtualization [19], and Docker systems [55]. Our sandboxing approach is not fundamentally new compared to the previous ones. Instead, S2Net newly introduces restricted interfaces that are necessary for secure communication.

9 Conclusion

In this paper, we have proposed a novel system, named S2Net, for smart home routers, to preserve user privacy while providing rich functionalities. S2Net protects user privacy against both vulnerable router OS and applications by ensuring that the private data are always shielded from the Normal world and processed only in the Secure world of ARM TrustZone, and isolating data access among sessions with the proposed secure interfaces and SOUP. We could demonstrate via the experimental results that the implemented S2Net system was able to achieve high throughput performance, while significantly speeding up the cryptography performance and mitigating the overhead of S2Net by using the proposed crypto-workers for multiple CPU cores and a cryptographic hardware module. The results also showed that the S2Net enables the smart home router to provide various services such as web caching and content filtering to users without interruption, while protecting the personal data.

Appendix

A. Example Code of Applet

In this section, we present example code of DPI applet with the detailed explanation consisting of the following four phases: initialization, pull packet/data, process data, push packet/data. Although the exact shape of functions in the example code of the prototype could be different from that of the interfaces shown in Fig. 5, functionality of the prototype is same as the description stated in Section 4.4.

Example code of the initialization phase is shown in Fig. 14. In the initialize phase, the queues from session manager is initialized (from/to, up/downlink). Here we used four queues in total, however, developers can use only two queues if they want to consider only downlink data. The memory allocation is requested by calling API provided by Genode OS [21]. In Genode, which is based on micro kernel, the OS strictly restrict the resources allocated to each process.

Fig. 15 shows how the applet pulling and pushing packets from the buffers. In the while loop of function named, *entry()*, the applet checks whether there are newly arrived packets or not. If there are packets in uplink or downlink, the packets are pulled, processed, and pushed as shown in *entry_ps_to_pc()*. In this example, processes for uplink and downlink are exactly same (using the same *process_packet()* function). Some developers can differentiate these processes (e.g., applying different rules to uplink and downlink).

The packet processing phase of the deep packet inspection applet is shown in Fig. 16. In this phase, the data is

```
1  void Regex_box::Regex_box::initialize()
2  {

3      //initialize shared queue
4      _to_ps.initialize_queue((char*)"
           regex_box_to_ps");
5      _from_ps.initialize_queue((char*)"
           regex_box_from_ps");
6      _to_pc.initialize_queue((char*)"
           regex_box_to_pc");
7      _from_pc.initialize_queue((char*)"
           regex_box_from_pc");

8      //allocate memory for the buffer
9      if(_buf == (void *) 0)
10     {
11        _buf = (void *)Genode::env()->heap()->alloc
              (packet_size_max);
12     }

13     //initialize pool for generate random rules
14     init_char_pool();
15 }
```

Fig. 14. Example code of deep packet inspection applet. Phase 1) initialization.

```
1  void Regex_box::Regex_box::entry()
2  {
3     //start the regex box
4     while(1)
5     {
6        if (entry_ps_to_pc() || entry_pc_to_ps())
7           continue;

8        //if no packet, wait
9        _timer.usleep(10);
10    }
11 }

12 //pull packet from ps, process it, send it to pc
13 bool Regex_box::Regex_box::entry_ps_to_pc()
14 {
15    int pkt_size = _from_ps.pull_data(_buf,
          packet_size_max);
16    if(pkt_size <= 0)
17    {
18       //no data, return
19       return false;
20    }

21    //process packet
22    process_packet(_buf, pkt_size);

23    //return packet
24    while(_to_pc.put_data(_buf, pkt_size) <
          pkt_size){}

25    return true;
26 }
```

Fig. 15. Example code of deep packet inspection applet. Phase 2 and 4) pulling/pushing packets.

```
1  void Regex_box::Regex_box::process_packet(void *
       buf, int size)
2  {
3     //doing some regex here
4     unsigned matched = 0, unmatched = 0;
5     for(unsigned int i=0; i < _rules.size(); i++)
6     {
7        boost::xpressive::smatch what;
8        if(boost::xpressive::regex_search( std::
             string((const char*)buf), what, _rules[
             i].match_str))
9        {
10          matched ++;
11       }else{
12          unmatched ++;
13       }
14    }
15 }
```

Fig. 16. Example code of deep packet inspection applet. Phase 3) packet processing.

```
1  int main(void)
2  {
3     Regex_box::Regex_box dpi_box;

4     //initialize the dpi module
5     dpi_box.initialize();

6     //launch it
7     dpi_box.entry();

8     //return result
9     Genode::sleep_forever();
10    return 0;
11 }
```

Fig. 17. Example code of deep packet inspection applet – main procedure.

processed by pre-defined regular expression (regex) rules for pattern matching. In this paper, we ported Boost library into Genode for the regex processing [56]. Unlike the example, which just counts the number of matched rules, developers can make their own routine for the data processing. Finally, the main procedure for running applet is shown in Fig. 17.

Acknowledgments

This work was partially performed while Seung-seob Lee and Hang Shi were research interns at Microsoft Research Asia. This work has been supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2017R1A2B4002000).

References

[1] P. GarciaLopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.

[2] M. Henze, J. Hiller, O. Hohlfeld, and K. Wehrle, "Moving privacy-sensitive services from public clouds to decentralized private clouds," in *Proc. IEEE Int. Conf. Cloud Eng. Workshop*, Apr. 2016, pp. 130–135.

[3] A. M. Khan and F. Freitag, "On participatory service provision at the network edge with community home gateways," *Procedia Comput. Sci.*, vol. 109, pp. 311–318, Jan. 2017.

[4] Google OnHub smart home router. 2016. [Online]. Available: https://on.google.com/hub/

[5] Router bugs flaws hacks and vulnerabilities. 2019. [Online]. Available: http://routersecurity.org/bugs.php

[6] L. Constantin, "Large-scale attack hijacks your router through your browser". PCWorld, May 25, 2015. [Online]. Available: http://www.pcworld.com/article/2926312/large-scale-attack-hijacks-routers-through-users-browsers.html/

[7] Consumers want simple online protection. 2015. [Online]. Available: https://www.nominum.com/press_item/survey-shows-consumers-want-simpler-online-protection-from-internet-service-providers/

[8] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "S" in HTTPs," in *Proc. 10th ACM Int. Conf. Emerging Netw. Experiments Technol.*, 2014, pp. 133–140.

[9] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *Proc. 11th USENIX Symp. Operating Syst. Des. Implementation*, Oct. 2014, pp. 267–283.

[10] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. R. Rodriguez, and P. Steenkiste, "Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 199–212.

[11] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep packet inspection over encrypted traffic," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 213–226.

[12] L. Guan, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger, "TrustShadow: Secure execution of unmodified applications with ARM TrustZone," in *Proc. 15th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2017, pp. 488–501.

[13] ARM Ltd., "ARM security technology - building a secure system using TrustZone technology," ARM Technical White Paper, 2005–2009.

[14] Almond and Almond+ smart home routers. 2018. [Online]. Available: https://www.securifi.com/

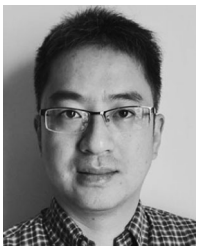
[15] A. Patil, "New range of smart home products accommodates early and new adopters of home automation," 2017. [Online]. Available: http://www.iotglobalnetwork.com/iotdir/2017/03/06/new-range-of-smart-home-products-accommodates-early-and-new-adopters-of-home-automation-4555/

[16] Z. Li, Y. Dai, G. Chen, and Y. Liu, *Offline Downloading: A Comparative Study*. Singapore: Springer, 2016.

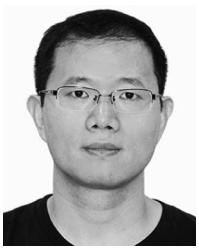
[17] M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow," in *Proc. 11th Int. Conf. Availability Rel. Secur.*, Aug. 2016, pp. 147–156.
[18] OpenWrt. 2018. [Online]. Available: https://openwrt.org/
[19] Microsoft, "Overview of Application Virtualization," Jun. 16, 2016. [Online]. Available: https://docs.microsoft.com/en-us/microsoft-desktop-optimization-pack/appv-v4/overview-of-applicationvirtualization
[20] ARM Ltd., "ARM security technology - building a secure system using TrustZone technology." 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf
[21] Genode Labs, "Genode operating system framework." 2019. [Online]. Available: https://genode.org/
[22] Transport layer security (TLS) extensions, RFC 3546. 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3546.txt
[23] The trusted computing platform alliance, 2003. [Online]. Available: https://www.trustedpc.com
[24] S. L. Keoh, S. S. Kumar, and H. Tschofenig, "Securing the internet of things: A standardization perspective," *IEEE Internet Things J.*, vol. 1, no. 3, pp. 265–275, Jun. 2014.
[25] S. Pearson and B. Balacheff, *Trusted Computing Platforms: TCPA Technology in Context*. Englewood Cliffs, NJ, USA: Prentice Hall, 2003.
[26] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon, M. Nystrom, D. Robinson, R. Spiger, S. Thom, and D. Wooten, "fTPM: A firmware-based TPM 2.0 implementation," Tech. Rep. MSR-TR-2015–84, Nov. 2015. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=258236
[27] NXP Semiconductors, "NXP i.MX 6Quad processors." 2018. [Online]. Available: http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-applications-processors/i.mx-6-processors/i.mx6qp/i.mx-6quad-processors-high-performance-3d-graphics-hd-video-arm-cortex-a9-core:i.MX6Q
[28] OpenSSL Software Foundation, "OpenSSL cryptography and SSL/TLS toolkit." 2019. [Online]. Available: https://www.openssl.org/
[29] D. Hein, J. Winter, and A. Fitzek, "Secure block device–secure, flexible, and efficient data storage for ARM TrustZone systems," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, 2015, pp. 222–229.
[30] Freescale Semiconductor Inc., " Secure boot on i.MX50, i.MX53, and i.MX 6 series using HABv4," Oct. 2015. [Online]. Available: https://www.nxp.com/files-static/32bit/doc/app_note/AN4581.pdf
[31] L. Guan, C. Cao, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger, "Building a trustworthy execution environment to defeat exploits from both cyber space and physical space for ARM," *IEEE Trans. Depend. Secure Comput.*, vol. 16, no. 3, pp. 438–453, May/Jun. 2019.
[32] T. Kocak and I. Kaya, "Low-power bloom filter architecture for deep packet inspection," *IEEE Commun. Lett.*, vol. 10, no. 3, pp. 210–212, Mar. 2006.
[33] ATTO Technology Inc., "Disk benchmark software." 2019. [Online]. Available: https://www.atto.com/disk-benchmark/
[34] J. Davis and S. Rivoire, "Building energy-efficient systems for sequential I/O workloads," Microsoft Res., Redmond, WA, USA, Tech. Rep. MSR-TR-2010–30, Mar. 2010.
[35] Google, "YouTube help, live encoder settings, bitrates, and resolutions." 2019. [Online]. Available: https://support.google.com/youtube/answer/2853702
[36] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," *In 27th USENIX Secur. Symp. (USENIX Secur. 18)*, pp. 973–990, 2018.
[37] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *arXiv:1801.01203*, 2018. [Online]. Available: https://arxiv.org/abs/1801.01203
[38] Arm Limited, "Speculative processor vulnerability." 2018. [Online]. Available: https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability
[39] J. Bech, A. Biesheuvel, M. Brown, and D. Thompson, "Linaro - implications of meltdown and spectre : Part 2," 2018. [Online]. Available: https://www.linaro.org/blog/meltdown-spectre-2/
[40] L. Guan, C. Cao, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger, "Building a trustworthy execution environment to defeat exploits from both cyber space and physical space for ARM," *IEEE Trans. Depend. Secure Comput.*, vol. 16, no. 3, pp. 438–453, May/Jun. 2019.
[41] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," *ACM Trans. Comput. Syst.*, vol. 35, no. 4, pp. 1–32, Dec. 2018.
[42] G. Heiser and K. Elphinstone, "L4 Microkernels: The lessons from 20 years of research and deployment," *ACM Trans. Comput. Syst.*, vol. 34, no. 1, pp. 1:1–1:29, Apr. 2016.
[43] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, "Rethinking the library OS from the top down," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 291–304, 2011.
[44] A. W. Services, "AWS CloudHSM getting started guide," Nov. 2013. [Online]. Available: http://aws.amazon.com/cloudhsm/
[45] T. C. Group, "TPM main specification level 2 Version 1.2, Revision 116," Mar. 2011. [Online]. Available: https://trustedcomputinggroup.org/resource/tpm-main-specification/
[46] Intel Corp., " Intel Software guard extensions programming reference," Oct. 2014. [Online]. Available: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf
[47] S. Loreto, J. Mattsson, R. Skog, H. Spaak, G. Gus, D. Druta, and M. Hafeez, "Explicit trusted proxy in HTTP/2.0. draft-loreto-httpbis-trusted-proxy20–01," IETF, Feb. 2014. [Online]. Available: https://tools.ietf.org/html/draft-loreto-httpbis-trusted-proxy20-01
[48] R. Peon, "Explicit Proxy in HTTP/2.0 draft-rpeon-httpbis-exproxy-00," IETF, Jun. 2012. [Online]. Available: https://tools.ietf.org/html/draft-rpeon-httpbis-exproxy-00
[49] D. McGrew, D. Wing, Y. Nir, and P. Gladstone, " TLS proxy server extension. draft-mcgrew-tls-proxyserver-01," IETF, 2012. [Online]. Available: https://tools.ietf.org/html/draft-mcgrew-tls-proxy-server-01
[50] S. Lee, H. Shi, K. Tan, Y. Liu, S. Lee, and Y. Cui, "Smart and secure: Preserving privacy in untrusted home routers," in *Proc. 7th ACM SIGOPS Asia-Pacific Workshop Syst.*, 2016, Art. no. 11.
[51] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, "Difference engine: Harnessing memory redundancy in virtual machines," *Commun. ACM*, vol. 53, no. 10, pp. 85–93, 2010.
[52] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, "Rethinking the library OS from the top down," in *Proc. 16th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2011, pp. 291–304.
[53] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," in *Proc. 2nd ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst.*, 2007, pp. 275–287.
[54] S. Bhattiprolu, E. W. Biederman, S. Hallyn, and D. Lezcano, "Virtual servers and checkpoint/restart in mainstream Linux," *SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 104–113, Jul. 2008.
[55] Docker. 2019. [Online]. Available: https://www.docker.com/
[56] E. Niebler, "Chapter 46. Boost.Xpressive." 2018. [Online]. Available: https://www.boost.org/doc/libs/1_69_0/doc/html/xpressive.html

Seung-Seob Lee received the BS degrees in computer science from Yonsei University, Seoul, South Korea, in 2011. He is currently working toward the PhD degree in the Department of Computer Science, Yonsei University, Seoul, South Korea. His research interests lie in the fields of fog/edge computing, especially for machine learning-based resource optimization and system-level network security for heterogeneous fog/edge devices. He is a student member of the IEEE.

Hang Shi is currently working toward the PhD degree in the Department of Computer Science and Technology, Tsinghua University. His supervisor is Prof. Y. Cui. His research interests include transport layer optimizations and mobile system.



Kun Tan received the BE, ME, and PhD degrees in computer science and engineering from Tsinghua University, Beijing, China, in 1997, 1999, and 2002, respectively. He joined Microsoft Research Asia, Beijing, after his graduation. He is currently a vice president of the Central Software Institute and the director and the chief architect with the Cloud Networking Lab, Huawei Technologies Co., Ltd. He has filed more than 30 pending patents and seven granted patents after he joined Microsoft Research Asia. His research interests include transport protocols, congestion control, delay-tolerant networking, and wireless networks and systems. He is a member of the IEEE.



Yunxin Liu received the BS degree from the University of Science and Technology of China, in 1998, the MS degree from Tsinghua University, in 2001, and the PhD degree from Shanghai Jiao Tong University, in 2011. He is a senior researcher with Microsoft Research Asia. His research interests include mobile and edge systems. He is a senior member of the IEEE.

SuKyoung Lee received the BS, MS, and PhD degrees in computer science from Yonsei University, Seoul, South Korea, in 1992, 1995, and 2000, respectively. From 2000 to 2003, she was with the Advanced Networking Technologies Division, National Institute of Standards and Technology, Gaithersburg, Maryland. She is currently a professor with the Department of Computer Science, Yonsei University, Seoul, South Korea. Her current research interests span wireless networking, mobile/edge systems, and vehicular networks. She is a member of the IEEE.

Yong Cui received the BE and PhD degrees in computer science and engineering from Tsinghua University, Beijing, China, in 1999 and 2004, respectively. He is currently a full professor with Tsinghua University and the co-chair of IETF IPv6 Transition WG Software. He has published more than 100 papers in refereed journals and conferences, holds more than 40 patents, and authored three Internet standard documents, including RFC 7040 and RFC 5565, for his proposal on IPv6 transition technologies. His major research interests include mobile wireless Internet and computer network architecture. He is a member of the IEEE.