SGX-USB: Secure USB I/O Path for Secure Enclaves

Yeongjin Jang
Samsung Research America†
blue69420892@gmail.com

Sejin Keem
Portland State University
sejin@pdx.edu

† Conducted the work at the Georgia Institute of Technology.

Abstract

User input plays an essential role in computer security because it can control system behavior and make security decisions. Output from the system to users is also important because it often contains security-critical information that must be protected in terms of its integrity and confidentiality, such as passwords and user's private data. Despite the importance of user input and output (I/O), modern computer systems often fail to provide necessary security guarantees for them, which could result in serious security breaches. To establish trust in the user I/O in a commodity computer system, we present SGX-USB*, which combines three essential security properties: confidentiality, integrity, and authenticity, to ensure the assurance of user I/O.* SGX-USB *establishes a trusted communication channel between the USB controller and an enclave instance of Intel SGX. The implemented system supports common user input devices, such as keyboards and mice, over the trusted channel, ensuring the integrity of user input.*

Keywords: Confidential Computing, Secure Enclaves, Trusted I/O, and Intel SGX.

1. Introduction

Today's systems are very complex. Even a simple desktop computer consists of a huge software stack including the operating system, device drivers, system daemons and other applications. Thus protecting the entire software stack of a system is extremely difficult. One promising approach to protecting a system is to reduce the attack surface by isolating the execution runtime into a separate environment. The history of building secure operating systems (OS) and hypervisors [Seshadri et al., 2007, McCune et al., 2010] have evolved into many software-based approaches [Klein et al., 2009, Onarlioglu et al., 2013, Koeberl et al., 2014, Chen et al., 2016] to provide the trusted execution environment (TEE) in commodity systems.

Recently, Intel introduced a new hardware extension, Intel Software Guard Extension (SGX) [Hoekstra et al., 2013], which provides a hardware-based TEE, also called as a secure enclave. While software-based TEEs still require either a trusted hypervisor or a trusted operating system, this hardware TEE implementation offers a strong security guarantee of not trusting privileged software including operating systems and hypervisors, by isolating memory and registers at the hardware level [Costan and Devadas, 2016, Rozas, 2013].

Although Intel SGX is now available in the most of newly manufactured commodity x86 processors, this hardware TEE is still limited to server or daemon applications because Intel SGX cannot support trusted user input/output (I/O) to its enclave that is running in ring 3, due to the requirement that I/O handling must be done in ring 0. In order to get benefits from Intel SGX, we design SGX-USB, which can establish a secure I/O path between a USB device and an enclave. In particular, SGX-USB opens a secure channel that can support USB protocol, which enables support for a variety of user I/O devices including a keyboard, mouse, camera, speaker, and display, and even for non-user-facing devices such as a disk.

To enable a secure channel, SGX-USB places a proxy device that sits in the middle of the channel and establishes a secure communication channel between an I/O device and the enclave. The channel is designed to guarantee the authenticity of two end points, the application in the secure enclave and the proxy device. On top of that, the channel also guarantees the confidentiality and the integrity of the data that are passed through the channel.

Establishing a secure communication channel starts with a remote attestation process that authenticates the enclave and the proxy device, and shares a secret between these two at the same time. After authenticating end-points and sharing a secret between them, the proxy device opens a communication channel between a USB I/O device and an enclave. The proxy device protects the data transmitted in the channel by using a derived encryption key from the shared secret. Through the remote attestation process and application of encryption

URI: https://hdl.handle.net/10125/107279
978-0-9981331-7-1

over the channel, SGX-USB can guarantee the three key security properties: authenticity by remote attestation, and confidentiality and integrity by encryption. Thereby, SGX-USB provides the assurance of user input and allows the enclave instance to handle commands and data from the user securely.

While current applications of Intel SGX primarily ensure secure network and file I/O, this new design enables secure user I/O in the TEE so that Intel SGX can facilitate user-facing trusted applications, such as an authentication manager that securely processes passwords. Moreover, we show that constructing an end-to-end trusted I/O channel from one user to another over the Internet is possible with SGX-USB; for example, having a video chat over the Internet. SGX-USB can forward not only the user I/O devices but also general USB I/O devices through the established secure channel. Its overhead on the bandwidth is around 1%, and added latency is around 11 microseconds, all of which are negligible.

To summarize, we made the following contributions in this paper:

- We design SGX-USB, which enables the trusted user I/O to an enclave of Intel SGX by establishing a trusted I/O channel between a USB device and an enclave. The design of SGX-USB ensures the authenticity of channel end points and the confidentiality and the integrity of the data that flows through the established secure channel.
- We extended the Intel SGX remote attestation process to enable authentication and secret sharing between a remote device and an enclave.
- We implemented a prototype of SGX-USB with commodity hardware, a small board computer and a desktop computer to demonstrate SGX-USB's feasibility in securely delivering keyboard input to an enclave. Moreover, we present a potential interesting use case of SGX-USB for video chat, which establishes a user-to-user trusted I/O channel over the Internet.

2. Background and Related Work

Intel SGX. Intel Software Guard Extensions (SGX) [Hoekstra et al., 2013, Intel Corporation, 2013, Intel Corporation, 2014, Rozas, 2013] is an extension of the x86 instruction set architecture (ISA). It offers a trusted execution environment (TEE), known as enclaves, operating at the user-level.

SGX Threat Model. The trusted computing base (TCB) of SGX encompasses only the processor hardware and the program running within an enclave. To preserve the TCB without relying on an operating system, SGX grants an enclave isolated memory space and independent execution runtime [Costan and Devadas, 2016]. SGX architecturally prevents any access to the enclave's memory and registers from any other execution domain. This strict isolation even applies to the operating system kernel. Moreover, SGX encrypts all data within an enclave to assure its confidentiality and integrity, safeguarding against physical attacks like cold-boot and bus snooping. Given this combination of isolation and encryption, an enclave can remain impervious to potential threats from operating systems, kernel device drivers, and other processes.

Remote attestation. Beyond isolation and encryption, SGX offers protocols for both local and remote attestations, ensuring the integrity of an enclave instance's code and its loading parameters. An enclave can produce a status report, detailing the measurements (e.g., hash) of its loaded code and security specifics.

Local attestation facilitates the validation of one enclave by another on the same hardware platform. The target enclave's report is signed using a hardware key, which another enclave can verify. For remote attestation, a remote verifier assesses the status of an enclave. Intel supports this with the Quoting Enclave and Intel Attestation Service (IAS). The Quoting Enclave can produce a *quote*, a signed version of an enclave's report, similar to local attestation. This quote is endorsed with an Intel-issued hardware key. The IAS then offers APIs to validate a quote. Hence, after receiving a quote, a remote verifier submits it to IAS to ascertain its legitimacy.

I/O handling in SGX. Given that SGX's design situates the enclave in user-space (i.e., ring 3), it cannot directly process I/O requests, which generally demand kernel (i.e., ring 0) privileges [Costan and Devadas, 2016]. Instead, an external application works in tandem with the operating system to manage I/O requests for the enclave, mirroring how typical processes interact with the OS.

As the actual I/O management is left to the untrusted OS without inherent protection, the enclave must independently secure its I/O channels. This is vital to maintain the confidentiality and integrity of data in transit. Hence, Intel suggests leveraging the remote attestation protocol, combined with the Diffie-Hellman key exchange [Johnson et al., 2016, Brickell and Li, 2007, Anati et al., 2013], to craft a secure communication link between an enclave instance and a distant host. To permanently store data produced by an enclave on a disk, Intel recommends the *sealing* feature [Anati et al., 2013], which ensures authenticated encryption using a hardware key. Furthermore, many research initiatives that depend on Intel SGX for data protection have adopted the Transportation Layer Security (TLS) to provide authenticated encryption for network communication channels [Baumann et al., 2014, Kim et al., 2015, Shih et al., 2016, Hunt et al., 2016, Arnautov et al., 2016, Schuster et al., 2015].

User I/O in SGX. Unlike network connections that allow the negotiation of security parameters across their channels, user I/O devices like keyboards and mice are constrained. Termed "dumb I/O devices", they lack the capability to negotiate encryption schemes to establish secure communication with an enclave. This limitation is not exclusive to user I/O devices. Other general I/O devices face similar challenges. For instance, graphic

display devices, such as GPUs, rely on direct memory access (DMA) for data processing. However, they can't securely interact with an enclave due to the absence of protocols designed for safe communication.

Intel does offer a solution for audio and video outputs with its protected audio and video path (PAVP) [Intel, 2008]. This is integrated into the chipset and ensures data transmitted over bus channels is encrypted. Yet, this solution has its own limitations. The protocol behind PAVP is proprietary, making it compatible only with Intel HD Graphics devices—an integrated GPU tailored for the processor. As a result, standard I/O devices are left without the means to utilize this protocol.

3. Overview

SGX-USB is designed to establish a trusted communication channel, enabling user I/O devices to connect seamlessly with an Intel SGX enclave instance. Notably, SGX-USB pioneers a secure channel tailored for USB devices. Given the versatility of USB, this encompasses a range of user I/O devices, from keyboards, mice, and cameras to speakers and displays. It even extends to non-user-facing devices like disks. A challenge arises since standard USB devices aren't inherently equipped to negotiate or implement security parameters on their I/O channels.

SGX-USB introduces an intermediary—a proxy device positioned within the channel. This device's role is to facilitate secure communication between the I/O device and the enclave. This secure dialogue is initiated via a remote attestation process, which not only authenticates the enclave and proxy device but simultaneously shares a confidential secret between them. Post authentication and secret-sharing, the proxy device activates a communication channel linking the USB I/O device with the enclave. Data transmission within this channel is safeguarded using an encryption key derived from the shared secret.

The salient feature of this remote attestation process is the assurance it offers concerning the genuineness of both channel endpoints. Coupled with authenticated encryption, it guarantees the secure conveyance of I/O requests to and from the intended USB device. In essence, the channel is so fortified that no system component outside the enclave can access the I/O data.

3.1. Security Guarantees

SGX-USB offers the following assurances for the secure communication channel it establishes between an enclave and the USB Proxy USB Proxy Device:

- **Authenticity**: The security protocol of SGX-USB allows channel establishment only upon verification of its endpoints—the enclave and the USB Proxy Device. Channel I/O requests are encrypted with a unique key, exclusively shared between the authenticated entities.
- **Confidentiality**: All I/O requests on the channel are encrypted using a key, securely derived from a shared secret. The robustness of the remote attestation process ensures that only the two communication channel endpoints are privy to this encryption key. This effectively bars any potential attackers from accessing the raw data of I/O requests.
- **Integrity**: SGX-USB also ensures the integrity of data. By employing an authenticated encryption model that is resilient against data tampering, replay, and reordering attacks, the system effectively thwarts any malicious data injections.

3.2. Assumptions and Threat Model

In conceptualizing SGX-USB for a fortified I/O channel linking a USB device to an SGX enclave, we operate under several assumptions:

- We assume the computer's operating system that hosts enclaves as untrustworthy. This assumption extends to potential compromise scenarios, including non-enclave OS applications, system libraries, drivers, and the kernel.
- Our trust is exclusively placed on the Intel SGX hardware component (the processor) and Intel's remote attestation service (IAS). Such trust enables the verification of program integrity within enclaves and facilitates secret key sharing between the enclave and the USB Proxy Device.
- we trust the private key of the remote attestation service provider that signs the public key of the USB Proxy Device. In this regard, an enclave can verify the validity of the UPD by checking if the public key of the UPD that signs its ECDHE parameter is issued by the Remote Attestation Service Provider (RASP), by verifying its signature.
- Complete faith is vested in the software stack constituting the USB Proxy Device (UPD), which includes the UPD's OS kernel, the `usbip` driver [Márton, 2011], system libraries, and the proxy application operating on the device.
- Our model assumes that attackers lack physical access to any components of SGX-USB. Furthermore, the hardware integral to the SGX-USB framework is assumed trustworthy. This implies that attackers cannot directly interface with USB devices (like keyboards) or implant any hardware backdoors.

4. Design of SGX-USB

4.1. Architecture

To provide the required security properties to the channel that is established to an enclave, the SGX-USB system consists of following components: an enclave program, the Remote Attestation Service Provider (RASP), and the USB Proxy Device (UPD). Figure 1 illustrates how the SGX-USB components are connected.

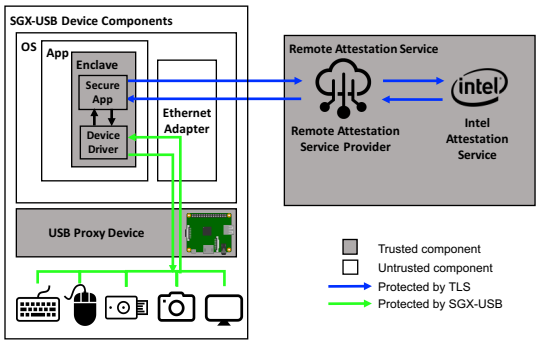

Figure 1: A diagram that illustrates the architecture of SGX-USB. An application that handles I/O runs in the enclave. The enclave will authenticate with the remote attestation service provider (RASP) through the Intel SGX remote attestation process. Intel Attestation Service (IAS) will provide the verification of a quote generated for an enclave, to verify the authenticity of an enclave. The USB Proxy Device (UPD) will receive the signed quote then verifies the signatures of the quote, and then establishes a secure communication channel with the enclave and forward USB I/O devices.

Enclave Program. In SGX-USB, the program that will process I/O must be run in an SGX enclave. This program can be any application that utilizes the secure I/O channel. For example, on utilizing a secure I/O channel as a secure method of processing password, a program that handles the authentication process with user's password will be running in the enclave.

Because an enclave of SGX cannot directly handle I/O requests, the enclave communicates through the untrusted part of the program (i.e. `ocall`) that handles (untrusted) I/O requests such as networking and exchanging unencrypted traffics with the USB Proxy Device. Over the untrusted channel, an enclave and the USB Proxy Device wrap the channel with an encryption layer to provide security guarantees on the confidentiality and the integrity of the data that they stream through the channel.

To share an encryption key and to verify the authenticity of the channel end point, SGX-USB utilizes the remote attestation process provided by Intel (through Intel IAS) to prove its authenticity and integrity of the program in the enclave and verifying the USB Proxy Device. This process is handled by the remote attestation service provider (RASP).

Remote Attestation Service Provider (RASP). The RASP handles the verification of the authenticity and integrity of an enclave program through the remote attestation protocol of Intel SGX. The RASP is a server program that resides on the network (i.e., on the Internet) and verifies whether or not the current enclave program is intact. By communicating with the Intel Attestation Service (IAS) and the enclave, the RASP receives a quote that is generated by the enclave, which indicates the launching status of the enclave, and sends the quote to the IAS to get a signed quote. Subsequently, the RASP signs the quote by its private key to make sure the authenticity of the ECDHE security parameter in the quote, which will be used for establishing a secure communication channel between the USB Proxy Device and the enclave.

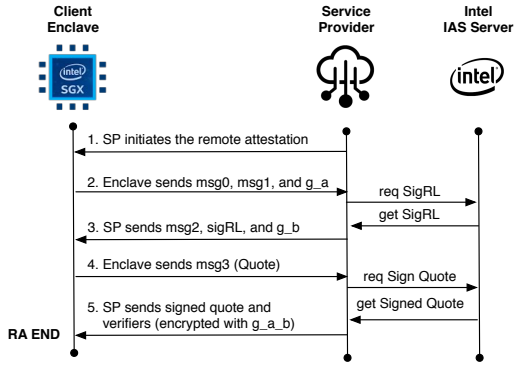

Figure 2: The remote attestation process of Intel SGX.

Intel Attestation Service (IAS). Intel Attestation Service is a part of the remote attestation infrastructure of Intel SGX. The job of the IAS is to verify a quote generated by the Quoting Enclave, which is a signed data of a measurement report of an enclave. Because all the quotes of enclaves are protected by a secret key that is fused in the processor and only Intel knows, only the IAS can verify the legitimacy of the quote. The RASP verifies the quote received from an enclave using the IAS to ensure the authenticity of the enclave.

USB Proxy Device (UPD). The USB Proxy Device is a proxy that forwards packets from USB I/O devices to an enclave through secure communication channels. The UPD sits between USB I/O devices and the enclave, and it acts as a middle man that creates secure I/O channel and forwards the I/O requests. To establish the secure channel, the UPD first shares a secret with the enclave program by following the remote attestation process. After sharing a secret, the UPD derives an encryption key and apply an encryption layer to the channel between an enclave and itself to make the channel secure. After establishing a secure communication channel protected by encryption, the UPD forwards USB packets from the target USB device to an enclave, and from an enclave to the target device, *vice versa*.

4.2. Verifying Authenticity and Sharing Secret through Remote Attestation

Before establishing a secure communication channel between the UPD and an enclave, both components authenticate each other to check if the each end point of the channel is intact. Because the regular remote attestation protocol provided by Intel only allows us to verify an enclave from the RASP, we extended the protocol to let the UPD verify an enclave and sharing a secret between them.

Intel SGX Remote Attestation. Intel SGX provides a way of attesting the launching status of an enclave through the remote attestation protocol. Figure 2 illustrates how this process works. In the following, we describe each step of the protocol.

1. The service provider (the RASP), which is a remote party that requests the verification of an enclave, initiates the remote attestation process.

2. The enclave that is being attested gets the request then send `msg0`, which contains group ID of an enclave and `msg1`, which contains the public key (i.e., `g_a`) parameter of the Elliptic Curve Diffie–Hellman Ephemeral (ECDHE) protocol that will be used for sharing a secret with the service provider at the end of the remote attestation process.

3. Next, on receiving both `msg0` and `msg1`, the service provider verifies the group ID (must be 0) in the `msg0`. If the group ID is zero, then the service provider requests a revocation list (i.e., `SigRL`) from the Intel IAS. This `SigRL` is signed by Intel IAS and will be used by the enclave for verifying the validity of the service provider. After processing the messages, the service provider generates `msg2`, which contains its public key parameter for the ECDHE key exchange (i.e., `g_b`).

4. After receiving `msg2`, the enclave generates a report and gets a quote for the report by the Quoting Enclave (QE). A report of an enclave includes the measurements (i.e., hash) of the launching status of an enclave that only Intel Attestation Service can verify as well as both of public key parameters (`g_a` and `g_b`) for the ECHDE key exchange. The Quoting Enclave, an enclave that is developed by Intel, will sign the report, and the enclave sends this quote to the service provider as `msg3`.

5. Subsequently, the service provider receives `msg3` and send it to the IAS to verify whether the quote is valid or not. Only for the valid quote, the IAS will return a signed quote with a signature generated by Intel's private key. The service provider verifies this signature; if it is valid, the service provider generates a shared secret and then send the signed quote to the enclave as `msg4`, by encrypting the signed quote with the secret key.

6. The enclave also calculates a shared secret and derives an encryption key; then it decrypts the quote from `msg4` using the key and verifies the signature of the quote. In consequence, the enclave can ensure that it has shared a secret with a service provider that is certified by Intel (because they cannot get the correct signature from IAS unless Intel does not certify them), and the service provider can ensure that it has shared a secret with a legitimate enclave instance (because IAS will not sign the quote if an enclave instance is not legitimate). Note that both the enclave and the RASP have shared a secret (i.e., `g_a_b`) through the ECHDE protocol.

SGX-USB Remote Attestation. To share a secret between the UPD and an enclave, we extended the remote attestation process of Intel SGX. Figure 3 shows the process of the remote attestation with our extension. We describe the extended part of the process in the following.

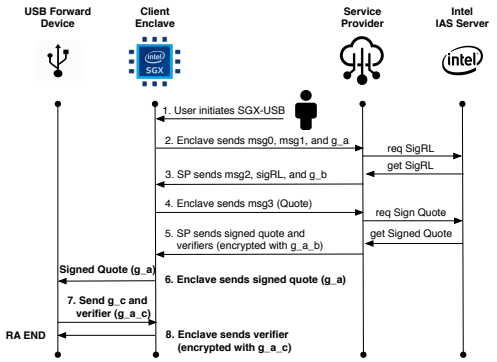

Figure 3: An extended remote attestation process for SGX-USB. Steps from 1 to 5 remain the same as the regular remote attestation of an enclave. Procedures marked with the bold face (Steps 6, 7, and 8) indicate additional procedures for attesting an enclave from the USB Forwarding Device.

- For the step 1, we changed the process to be user initiated instead of the service provider.
- After the step 5, the enclave sends the signed quote (decrypted from `msg4`) to the UPD.
- On receiving the signed quote, the UPD verifies if the quote is correctly signed by the Intel's private key and the private key of the RASP. Only if both signatures are verified, the UPD generates an ECDHE parameter and send the public part (i.e., `g_c`) to the enclave as `msg5`, along with the signature of this public parameter and the public key that can verify the signature. Note that the UPD presents a public key as a certificate that contains its signature, signed by the RASP.
- On receiving `msg5` from the UPD, the enclave verifies the signature of the ECDHE parameter (i.e., `g_c`) using the public key of the RASP, to check if the RASP has certified the signing key. As a result of the process, the UPD has verified that both the IAS and the RASP have signed the quote (so it is valid), and the enclave has verified that the signature of the public key parameter (i.e., `g_c`) is generated by a public key that is certified by the RASP (so the UPD is certified one). Only if all signatures are verified, both the enclave and the UPD calculates a shared secret (i.e., `g_a_c`) and derives an encryption key that will be used for securing the I/O channel.

4.3. User Verification

User Verification. Although the public key infrastructure and cryptographic operations can guarantee the authenticity, the confidentiality, and the integrity of communication channel, the establishment of the channel must go through the user verification process to ensure that the use of the channel follows the user's intent.

To this end, the UPD explicitly display the identity of an enclave (e.g., application name), the device that the

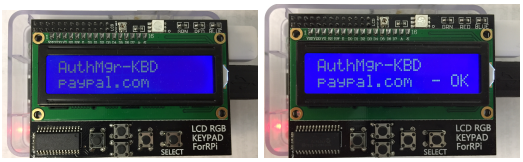

Figure 4: The user interface for verifying an enclave and its usage, presented in the USB Proxy Device. Figure on the left shows how the UPD displays the request for establishing a secure channel to a keyboard from an enclave. The information displayed on the LCD screen indicates the name of an enclave (i.e., AuthMgr), the name of the requested device (i.e., Keyboard), and application specific information for indicating the usage of the input (i.e., `paypal.com`). After clicking the `SELECT` button (i.e., the user approves), the screen will show the 'OK' sign at the end of the second line to indicate that the secure channel is established.

M := | Key ID (4 bytes) | Shared Secret (32 bytes) |
|---|---|
| Alg ID (4 bytes) | "SGXRAENCLAVE\0" (13 bytes) | "SGXRASERVER\0" (12 bytes) |

KEY := SHA256 (M)
SMK := KEY[0..16]
SK := KEY[16..32]

Figure 5: The data format for deriving secret key from a shared secret. The key derivation function uses the SHA-256 message digest algorithm to derive a 16 bytes secret key from a shared secret.

enclave connects to (e.g., keyboard), and the usage of the device (e.g., that domain name that the keystrokes will be submitted).

Figure 4 shows an example of the user verification process of SGX-USB on using the system as a password authentication manager. On the screen, the first line displays the application name as *AuthMgr* and the device name as *KBD* to indicate that the AuthMgr enclave would like to talk to the keyboard device. In the second line, the UPD will display how the user input will be used for, in other words, displays the domain name `paypal.com` to indicate that the password typed by the user will be submitted to the `paypal.com`.

To authorize the access, the user can click 'OK' button on the device (indicated as *Select* in Figure 4). The channel will be established only if there is a user approval; otherwise, the UPD will not make the connection to the device.

4.4. Integrity and Confidentiality: Encrypted Communication Channel

To protect the communication channel between an enclave and the UPD, SGX-USB wraps the channel with an encryption layer protected by the key that is exchanged during the remote attestation process. In short, SGX-USB applies the AES-128-GCM scheme, which is an authenticated encryption with associated data (AEAD) that can protect both data confidentiality and data integrity.

Key derivation. After finishing the remote attestation process, both an enclave and the UPD have shared a 256-bit secret through ECDHE protocol using the NIST P-256 curve. To derive a 128 bit key for an AES encryption, SGX-USB followed the same way on how Intel derives a secret key in their SDK example; the scheme uses the SHA-256 message digest algorithm. Figure 5 illustrates how the key derivation function works. By hashing the Key ID (0 in this case), the 32 bytes shared secret, the Algorithm ID (0 in this case), and two string literals `SGXRAENCLAVE` and `SGXRASERVER`, the derivation function generates a 32 bytes message digest and uses the latter 16 bytes (*SK* in Figure 5 for the encryption key.

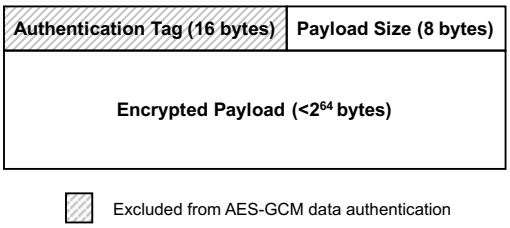

Figure 6: The header format for delivering encrypted payload on trusted I/O channel in SGX-USB. Authentication Tag will be used for verifying the integrity of both the size field and encrypted payload. While the AES-128-GCM encryption applied only to the payload, the size field is supplied as additional data for AES-128-GCM data authentication; thus the encryption scheme protects the integrity of both encrypted payload and the size field.

Encryption Scheme. SGX-USB uses `AES-128-GCM` for the encryption scheme for the secure channel. Since the GCM (Galois Counter Mode) is an authenticated encryption with associated data (AEAD) encryption scheme, we can use one key for protecting both the confidentiality and the integrity of the data. To encapsulate a plaintext USB packet into an encrypted packet, we attach a 24 bytes header on the payload followed by the encrypted packet payload. Figure 6 shows how the header of a packet in the channel composed.

To send a plaintext USB packet over the secure channel, we first identify the size of the packet. To protect the integrity of both encrypted text and the size field, we encrypt the packet payload using the AES-128-GCM encryption scheme. At the same time when the encryption is being processed, we put the size (8 bytes) field as the additional data to be authenticated. In this way, the scheme allows us to detect any forgery on both encrypted data and the size field. As a result of an encryption routine, the scheme will generate a 16 bytes authentication tag that will be verified when decrypting the data to check if the data is intact. We put this tag at the top of the header to deliver the tag to the other end point.

We process the decryption in a reverse way. After receiving the header data, we initialize a decryption engine with the secret key, the authentication tag in the header, and the size field in the header as the additional data to authenticate. The encryption scheme will return true only if when the secret key and the authentication tag is matched.

Another point that is essential on applying the

AES-128-GCM encryption scheme is the setting of initialization vector (IV). To securely use the encryption scheme, one must not reuse the IV for one secret key. To follow such a secure scheme, we set a 12 byte (96bit) integer counter value starting with zero value for each of sending and receiving side, and increment IV counter per each en(de)cryption operation. In this regard, the IV will not be reused if the channel sends less than 2^{96} packets, which is a practically unreachable number. Additionally, because the IV counter is monotonically increasing on each sending and receiving side, the scheme is resistant to the replay and the reordering attack.

5. Implementation

We implemented our prototype of SGX-USB using a desktop machine that supports Intel SGX (a quad-core Intel Core i7 6700K@4.0Ghz) and Raspberry Pi 3 model B device, which is a small board computer, and securely forwarded a keyboard device to an enclave.

6. Evaluations

We evaluate SGX-USB by answering the following questions:

- How secure is the I/O communication channel established by SGX-USB? (§6.1)
- How much overhead does SGX-USB incur on delivering I/O packets in terms of throughput and latency? (§6.2)
- How long does it take to establish the secure I/O channel through the remote attestation process? (§6.2)

6.1. Security

Attacks against enclave instances. The first avenue for an attack is to thwart the protection provided by an SGX enclave instance. Because Intel SGX isolates all memory access from the entire domain controlled by attackers including operating system, attackers cannot obtain nor alter the runtime data in the enclave's memory. Additionally, attackers cannot change the behavior of an enclave instance because the remote attestation process ensures that the integrity of the code in the enclave. One possible way would be launching a malicious enclave instance and establishing secure I/O channel using this enclave. However, because the remote attestation process of SGX-USB not only includes the IAS but also utilizes the RASP, the RASP will not generate a signed quote if the enclave instance (and its measurement) is not pre-registered to the service. By only using the set of pre-registered enclave applications, an attacker could launch a disguising attack that invokes a different enclave instance with the enclave what user wants to use. In such a case, at the final verification stage of SGX-USB, the user will notice that the name of the malicious enclave instance is not the enclave that user has his/her intention, so request for establishing secure I/O channel will be rejected.

Attacks on the remote attestation procedure. Attackers could try to forge an acceptable message during SGX-USB's remote attestation process. First, an attacker could attempt to generate a fake quote; for example, the attacker could present a valid quote message with a legitimate measurement for a registered enclave while executing a malicious enclave instance. Nonetheless, this is strictly protected by the Intel SGX hardware and the IAS. All the measurement reports must be generated by the secret key fused into the processor hardware, and the quote can only be verified by the Quoting Enclave, which is created by Intel. Thus, the attacker cannot either generate valid quote with forged message or pass the verification process of the IAS.

Finally, an attacker could attempt to build a fake USB Proxy Device to inject arbitrary I/O message to an enclave instance. Although we set our threat model to exclude attackers with any physical access to the device, the attacker could obtain an instance of the USB Proxy Device if the device available in public. Because current prototype implementation builds the USB Proxy Device as a small computer instance, the attacker who can obtain the device can disassemble to leak the private key signed by the RASP and use the key to build a fake instance of the USB Proxy Device. However, we believe that this is just an implementation issue; this can be protected by implementing the UPD with the other TEE or entirely in hardware. We further discuss on other trusted ways of building the USB Proxy Device in §7.

Attacks on the secure channel. Attackers could attempt to decrypt or inject data on the established secure channel. Unfortunately, the secure channel is protected by a symmetric encryption scheme, `AES-128-GCM`, which is an authenticated encryption with associated data (AEAD). The correct use of the scheme guarantees that no attackers can decrypt or alter the encrypted data without obtaining the encryption key. To achieve this guarantee, SGX-USB follows the same way in how Transportation Layer Security (TLS) utilizes the same scheme as AEAD (e.g., use decrypted data only if tag matches, encrypt only short block, does not reuse the same IV, etc.). Moreover, because ECDHE key exchange scheme securely derives the key, attackers can obtain the key only if by breaking the scheme or by forging the key exchange message, all of which are impossible in SGX-USB construction.

Despite the fact that SGX-USB can guarantee the authenticity of channel end points and the integrity and the confidentiality of the data on the channel, SGX-USB cannot guarantee the availability of the channel. We further discuss on this limitation in §7.

6.2. Performance

We evaluated SGX-USB for the performance of I/O channel in terms of throughput and latency. Moreover, we provide timing information how long does the establishment of a secure I/O channel through remote

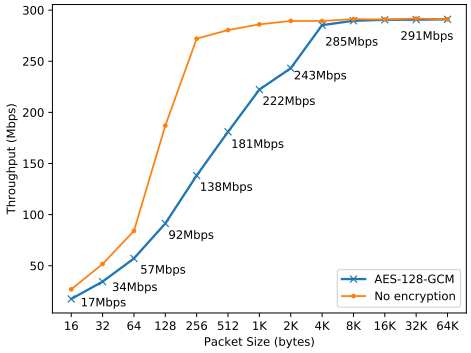

Figure 7: The measured throughput of the secure I/O channel for various packet size.

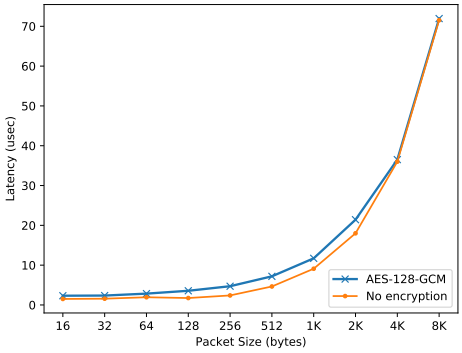

Figure 8: The measured average latency of the secure I/O channel for various packet size, in five seconds of transmission.

attestation takes.

Throughput and Latency. To evaluate the performance of the secure I/O channel established by SGX-USB, we measure the throughput and latency of the channel for various packet sizes. To help understand the result, we note that the maximum packet size of the USB protocol is 1 KB and typical packet size for USB HID devices is 32–512 bytes. Although we use a Gigabit Ethernet adapter (max bandwidth 1Gbps) for the communication channel, the adapter is connected to a USB 2.0 port (max bandwidth 480Mbps) due to the hardware limitation on the USB Proxy Device so the maximum bandwidth of the channel is around 310Mbps without any encryption or encapsulation.

Figure 7 depicts the throughput of the secure I/O channel for various packet size and Table 1 lists detailed numbers.

The smaller packet size incurs more overhead on both encryption process (CPU) and size (bandwidth). Because SGX-USB applies a separate instance of `AES-128-GCM` encryption (i.e., using a different IV) per each packet, the number of required encryption initialization process is increased for the smaller packet size. Moreover, because SGX-USB adds small header data (24 bytes) per each packet for transmitting data authentication tag (16 bytes) and indicating payload size (8 bytes), delivering smaller packet would incur more size overhead. Furthermore, we deliver USB packets over a TCP connection on the Ethernet link, so additional 50 bytes size overhead is applied per each 1426 byte payload (1426 = 1500 (MTU) - 50 (TCP/Ethernet) - 24 (header)).

Because of these overhead characteristics, SGX-USB demonstrated 57.1 Mbps of throughput for 64 bytes packets, which is around 18% of the maximum throughput. However, for 4K bytes packets, the bandwidth became saturated, and the overhead is negligible.

Encapsulating the packet and applying encryption on the packet also incurs overhead on the channel latency. Figure 8 depicts the throughput of the secure I/O channel for various packet size and Table 2 lists detailed numbers. Although the latency increases as the packet size increases, the absolute value of the latency in maximum USB packet size (i.e., 1 KB) is around 11 microsecond, which is fairly negligible.

Authentication speed. The remote attestation process of SGX-USB requires:

- Two round trips between the enclave and the RASP for delivering `msg0`, `msg1`, and `msg2`; and `msg3` and `msg4`,
- Two round trips between the RASP and the IAS, one for requesting and receiving `SigRL` and the other for and the signed quote,
- One round trip between the enclave and the UPD for exchanging ECHDE parameter (using the signed quote).

To model a realistic use case, we setup the connection between the enclave and the UPD as a local network connection, place the RASP on the remote network using the Google Cloud Platform and using the test IAS server provided by Intel.

The total round trip time for the remote attestation for SGX-USB (Figure 3) from step 1 to step 7, it took in average 553 milliseconds, with standard deviation 31ms for 100 times of remote attestation trials. This overhead is not that much because the entire process of remote attestation is one-time cost per each channel; it only happens when the USB Proxy Device establishes a new secure communication channel with an enclave.

7. Discussions

In this section, we discuss on how SGX-USB can support general I/O, on the performance of the channel, on the feasibility of hardware implementation of the UPD, on authenticating the identity of an enclave, and the availability of the channel, which is an unprotected security property on the channel.

General I/O support with SGX-USB. On forwarding a USB device to a network device, the prototype design of SGX-USB borrows the implementation of the `usbip` [Márton, 2011] project that generally supports all kinds of USB devices, so SGX-USB is. Because the

Table 1: The measured throughput of the secure I/O channel for various packet size, in five seconds of transmission. The throughput measured by the amount of payload data transmitted on the channel without counting any additional data for encapsulation. *W/O encapsulation* indicates the channel throughput when we count the entire amount of data transmitted through the channel including header information. *No encryption* indicates the channel throughput when we applied payload encapsulation (i.e., adding of the header) but did not apply encryption.

Packet Size (Bytes)	**64**	**128**	**512**	**1024**	**4096**	**8192**
W/O encapsulation (Mbps)	181.3	295.1	309.6	306.8	294.6	292.2
No encryption (Mbps)	84.1	187.0	270.5	286.1	289.4	289.6
AES-128-GCM (Mbps)	57.1	91.5	181.0	222.3	285.3	289.4
Overhead (%)	-32.1%	-51.1%	-35.5%	-22.3%	-1.4%	-0.07%

Table 2: The measured average latency of the secure I/O channel for various packet size, in five seconds of transmission. *No encryption* indicates the latency incurred when we applied payload encapsulation (i.e., adding of the header) but did not apply encryption.

Packet Size (Bytes)	**64**	**128**	**256**	**512**	**1024**	**4096**	**8192**
No encryption (usec)	1.74	1.93	2.39	4.64	9.10	35.99	71.53
AES-128-GCM (usec)	2.85	3.51	4.71	7.19	11.71	36.51	71.94
Overhead (usec)	+1.11	+1.57	+2.32	+2.55	+2.61	+0.52	+0.41

USB protocol transmits its data as packets, delivering each USB packet as a packet over the IP can be done by only incurring transformation overhead. Moreover, since we use transmission control protocol (TCP), which is a reliable protocol, for the data transmission, so there will be no missing packet on the other end. Therefore, as long as the driver software can run in the enclave, SGX-USB can support any USB device by forwarding its packet to the enclave.

In addition to USB devices, we believe that SGX-USB can forward devices that support RDMA (remote direct memory access) protocol through established channel by implementing a driver counterpart in the enclave, because by design, data for RDMA can be delivered over the network.

Channel Performance. Performance evaluation result of SGX-USB shows that its latency is in a performant range, but suffers performance bottleneck due to the encryption process. However, the bottleneck can be removed if the processor supports hardware-based encryption engine. Starting from newer ARM processors, processor manufacturers other than Intel try to integrate hardware module that accelerate encryption speed.

Regarding the size overhead of the channel bandwidth, the higher bandwidth would mostly be used by USB at bulk transfer, which sends a large amount of data split in each 1K byte packet. In such a case, SGX-USB can set a buffer to consolidate multiple USB packets into a large chunk (e.g., merging 16 packets into a 16Kbytes packet) only for the bulk transfer then the overhead will be negligible.

Hardware implementation of the UPD. Although we implemented our prototype of SGX-USB using a Raspberry Pi, which is a small board computer, we believe that implementing the UPD in hardware or other TEE with smaller TCB is feasible. The hardware implementation of UPD may include a USB host controller to receive raw packets from I/O devices, a communication interface to the enclave device, (in any form, e.g. Ethernet or USB OTG guest device), a cryptographic engine that handles the remote attestation process and AES encryption, and a small storage that is loaded with trusted public keys and a firmware that controls the components.

A more flexible design would be utilizing ARM TrustZone. In this case, by implementing the `usbip` driver on a small and secure TEE OS for TrustZone, we can significantly reduce the size of TCB. Moreover, in conjunction with using TPM, we can securely store the code and the private key of UPD with the data sealing feature; so the attackers with a possession of the UPD cannot alter the code nor retrieve the private key of the device.

Availability of the channel. We exclude the availability from the security property that SGX-USB should guarantee for the communication channel. This is an inherent limitation due to the adoption of the threat model of Intel SGX because Intel SGX excludes the operating system, which runs as a higher privilege than an enclave, from the threat model.

Although guaranteeing availability cannot be possible under current threat model, untrusting the operating system, the user will directly be notified at least when the availability issue happens (i.e., the device does not work at all). The operating status of the channel will either be fully working or not and cannot be half-working status because missing any of USB packet will break the

encryption status of the secure channel.

Trusting the USB Proxy Device. An attacker could deploy a rogue USB proxy device with the intent to maliciously control key negotiation, data encryption/decryption, and data integrity verification. Although this paper does not directly address such threats——our primary focus is on demonstrating the solution's feasibility——these rogue devices can be combated using a digital signature for the device. This signature could be endorsed by a recognized vendor, such as Intel, or another open-source consortium to ensure it is not proprietary. In today's context, trust in device drivers is fortified by verifying their code-signature. Similarly, a TEE vendor might issue a certificate that endorses the proxy device's public key, all within the framework of the public-key infrastructure (PKI). We leave such efforts as future work.

8. Conclusion

We have constructed a trusted I/O channel leveraging the trusted execution environment to ensure vital security properties. By facilitating a trusted user I/O pathway, Intel SGX can now support user-centric applications like authentication managers and end-to-end secure video chats. Looking ahead, an immediate next step involves the real-world hardware implementation of the SGX-USB device, further cementing a trusted I/O path to Intel SGX.

References

[Anati et al., 2013] Anati, I., Gueron, S., Johnson, S., and Scarlata, V. (2013). Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13.

[Arnautov et al., 2016] Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M. L., et al. (2016). Scone: Secure linux containers with intel sgx. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Savannah, GA.

[Baumann et al., 2014] Baumann, A., Peinado, M., and Hunt, G. (2014). Shielding applications from an untrusted cloud with haven. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 267–283, Broomfield, Colorado.

[Brickell and Li, 2007] Brickell, E. and Li, J. (2007). Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 21–30.

[Chen et al., 2016] Chen, Y., Reymondjohnson, S., Sun, Z., and Lu, L. (2016). Shreds: Fine-grained execution units with private memory. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (Oakland)*, pages 56–71, San Jose, CA.

[Costan and Devadas, 2016] Costan, V. and Devadas, S. (2016). Intel sgx explained. Technical report, Cryptology ePrint Archive, Report 2016/086. http://eprint.iacr.org.

[Hoekstra et al., 2013] Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., and Del Cuvillo, J. (2013). Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, pages 1–8, Tel-Aviv, Israel.

[Hunt et al., 2016] Hunt, T., Zhu, Z., Xu, Y., Peter, S., and Witchel, E. (2016). Ryoan: A distributed sandbox for untrusted computation on secret data. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Savannah, GA.

[Intel, 2008] Intel (2008). Graphics Drivers Blue-ray Disc* Playback On Intel Graphics FAQ. http://www.intel.com/support/graphics/sb/CS-029871.htm#bestexperience. Accessed: 05/04/2015.

[Intel Corporation, 2013] Intel Corporation (2013). Intel Software Guard Extensions Programming Reference (rev1). 329298-001US.

[Intel Corporation, 2014] Intel Corporation (2014). Intel Software Guard Extensions Programming Reference (rev2). 329298-002US.

[Johnson et al., 2016] Johnson, S., Scarlata, V., Rozas, C., Brickell, E., and Mckeen, F. (2016). Intel software guard extensions: Epid provisioning and attestation services. *White Paper*.

[Kim et al., 2015] Kim, S., Shin, Y., Ha, J., Kim, T., and Han, D. (2015). A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets)*, Philadelphia, PA.

[Klein et al., 2009] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. (2009). sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220.

[Koeberl et al., 2014] Koeberl, P., Schulz, S., Sadeghi, A.-R., and Varadharajan, V. (2014). Trustlite: A security architecture for tiny embedded devices. In *Proceedings of the Ninth European Conference on Computer Systems*, page 10. ACM.

[McCune et al., 2010] McCune, J. M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., and Perrig, A. (2010). TrustVisor: Efficient TCB Reduction and Attestation. In *Proceedings of the 31th IEEE Symposium on Security and Privacy (Oakland)*, pages 143–158, Oakland, CA.

[Márton, 2011] Márton, N. (2011). USBIP protocol documentation. https://lwn.net/Articles/449509/.

[Onarlioglu et al., 2013] Onarlioglu, K., Mulliner, C., Robertson, W., and Kirda, E. (2013). Privexec: Private execution as an operating system service. In *Proceedings of the 34th IEEE Symposium on Security and Privacy (Oakland)*, pages 206–220, San Francisco, CA.

[Rozas, 2013] Rozas, C. (2013). Intel software guard extensions. http://www.pdl.cmu.edu/SDI/2013/slides/rozas-SGX.pdf.

[Schuster et al., 2015] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M. (2015). VC3: Trustworthy Data Analytics in the Cloud using SGX. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA.

[Seshadri et al., 2007] Seshadri, A., Luk, M., Qu, N., and Perrig, A. (2007). Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. *ACM SIGOPS Operating Systems Review*, 41(6):335–350.

[Shih et al., 2016] Shih, M.-W., Kumar, M., Kim, T., and Gavrilovska, A. (2016). S-nfv: Securing nfv states by using sgx. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 45–48. ACM.