Establishing Trusted I/O Paths for SGX Client Systems With Aurora

Hongliang Liang, *Member, IEEE*, Mingyu Li, Yixiu Chen, Lin Jiang, Zhuosi Xie, and Tianqi Yang

***Abstract*— Today users' private data in edge computing devices (desktops, laptops, and tablets, etc.) is at high risk because they run applications on potentially compromised or malicious systems. To address this problem, hardware vendors propose Trusted Execution Environment (TEE). Particularly, Intel has released a new processor feature called Software Guard eXtension (SGX), and provisions shielded executions (i.e., enclaves) for security-sensitive computations. Regrettably, Intel SGX's design objectives omit trusted I/O paths. Without such guarantees, it is unlikely for an enclave to fulfill its security and privacy purposes because the source or sink of data may have been corrupted. To this end, we propose a novel architecture called AURORA to provide trusted I/O paths for enclave programs even in the presence of untrusted system software. Specifically, AURORA exploits two commercial-off-the-shelf features (System Management Mode, SMM and SGX) and establishes a secure channel between an enclave program and target device. Furthermore, we design and implement trusted paths for HID keyboard, serial port printer, hardware clocks, and USB mass storage, respectively. Leveraging these trusted paths, we protect real-world applications including OpenSSH client, OpenSSL server/client and SQLite database. Security and performance evaluations show that AURORA mitigates several kinds of I/O related attacks and introduces acceptable overheads. Our framework has been open-sourced and is available to the security community.**

***Index Terms*— Trusted path, hardware trust, Intel SGX, system security.**

I. INTRODUCTION

TRUSTED computing [1] encompasses six key technologies including: 1) endorsement key, 2) protected execution, 3) sealed storage, 4) secure input and output, 5) remote attestation, and 6) trusted third party. To this end, Intel provisions Software Guard eXtensions (SGX) [2] to establish trusted execution environment that protects the integrity and confidentiality of desired computation. Intel SGX enforces strong isolation in memory for security-sensitive compartments in a user-level application, called *enclaves*, from untrusted privileged systems. Many SGX-based protection architectures [3]–[8] have already been proposed to secure data and code on untrusted servers.

Unfortunately, Intel SGX enclaves have no direct access to any hardware resources because Intel SGX by design does not support any secure input/output mechanisms according to its Software Developer's Manual [9]. Concerning I/O requests, an enclave has to rely upon the untrusted compartments through system call interfaces. This unreliable dependency is susceptible to attacks such as memory-based Iago attacks [10] and I/O-based traffic analysis [11].

A trusted path between an enclave program and a target I/O device is a protected channel that assures the secrecy and authenticity of transferred data. It is extremely critical to user's privacy on personal computers, such as health information, financial account and personal documents, etc. The private data can be protected inside an enclave during runtime, but it is impossible to protect it at the time of being printed since typically printer devices only understand plaintext data and the corresponding drivers are under the control of the untrusted kernel. Similarly, the textual input from users such as keystrokes can be recorded on a compromised computer, which is a very serious problem when it comes to password security [12]. Previous research like SafeKeeper [13] only protects the password on the server side, but cannot protect the user's password from keylogging threats on the client side.

Intel SGX enclaves also suffer from the loss of a high-precision trusted clock. State-of-the-art library OSes such as Haven [3], Graphene-SGX [14] and Panoply [15] rely on the clock value from untrusted systems. As a result, these systems are vulnerable to time deception attacks [16]. Blockchain-oriented systems like Town Crier [17] depend on a remote server for a trusted time; however, this clock latency is high (usually hundreds of milliseconds) and uncertain. Intel Management Engine (ME) offers a trusted clock service for SGX enclaves, but the time value is coarse-grained (second-resolution) and not absolute at all [18]. It is not satisfactory for those enclaves that request the timestamp information at a high frequency, (e.g., tens or hundreds of times per second). Hence, to achieve millisecond-resolution time, Intel SGX SSL [19] has to use *ftime*, offered by the OS, which contradicts original SGX threat model. Cryptographic libraries such as TaLos [20], mbedTLS-SGX [21], WolfSSL-SGX [22] encounter the same problem as well. Additionally, another trusted service named Intel monotonic counters are slow and the NVRAM used by it only supports limited times of writes [23].

Manuscript received January 16, 2019; revised July 4, 2019 and August 26, 2019; accepted September 27, 2019. Date of publication October 4, 2019; date of current version January 16, 2020. This work was supported by the National Natural Science Foundation of China (NSFC) under Grant U1713212 and Grant 91418206. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Jean-Luc Danger. *(Corresponding author: Hongliang Liang.)*

The authors are with the Trusted Software and Intelligent System Laboratory, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: hliang@bupt.edu.cn; maxul@bupt.edu.cn; chenyixiu@bupt.edu.cn; jianglin@bupt.edu.cn; xiezhuosi@bupt.edu.cn; yangtianqi@bupt.edu.cn).

Digital Object Identifier 10.1109/TIFS.2019.2945621

Today's SGX ecosystem has not yet come up with an off-the-shelf solution to trusted I/O paths. Enclaves themselves cannot authenticate whether they are communicating with trusted devices or not. It is significantly important to solve the problem on ***how to establish trusted I/O paths for enclaves on untrusted systems***, as almost all SGX-based projects exclude the underlying OS from the trusted computing base (TCB) while awkwardly depending on it for I/O requests.

On Intel platforms, we observe that 1) all peripheral I/O devices are connected with south-bridge to I/O Advanced Programmable Interrupt Controller (I/O APIC), 2) I/O interrupts can be rerouted to System Management Interrupt (SMI) via configuring I/O APIC and 3) the SMI handler is initialized by Unified Extensible Firmware Interface (UEFI) which contains kinds of device drivers; thus, we hold the insight that *the SMI handler* is a perfect candidate to provide trusted I/O paths for security-critical applications.

In this paper, we present AURORA, a novel architecture that safeguards I/O paths of SGX enclaves. To directly communicate with hardware devices without depending on the untrusted software, AURORA introduces System Management RAM (SMRAM) as a special enclave. SMRAM is a tamper-proof memory region used only in System Management Mode (SMM), much like the enclave region with SGX protection in user mode. AURORA delegates I/O requests from enclaves to the SMI handler protected inside SMRAM. We call this particular SMI handler *SMVisor*. In hardware, SMVisor is safe from any corruptions by other privileged software. Therefore, Intel SGX enclaves can safely use I/O devices while remaining *transparent* to the underlying OS and hypervisor.

To summarize, this paper makes these contributions:

1) The notion of two types of trusted I/O paths: data-provider/-consumer path and data-storage/-transmitter (II-C) path, for security-sensitive applications (enclave programs).
2) A novel architecture named AURORA (III-A) leveraging two hardware features of Intel processors (SMM and SGX) to provide enclaves trusted paths transparent to untrusted OSes/HVs with a minimal trusted computing base (V-B).
3) Design and implementation of four types of trusted I/O paths (IV): keyboard, printer, clock, and storage, based on AURORA. To the best of our knowledge, we are the first to provide these *realistic* trusted I/O paths for SGX enclaves on the client side.
4) Three case studies: OpenSSH client, OpenSSL client, and SQLite database using trusted I/O paths provided by AURORA (VII). Experimental results show that AURORA can protect them from typical I/O related attacks (VI-B) with acceptable performance overhead (VI).

II. BACKGROUND AND PROBLEM DEFINITION

In this section, we first describe two architecture features of Intel's CPU, and then propose two types of trusted I/O paths based on the characteristics of devices. Finally, we describe the threat model and assumptions.

A. Software Guard Extension

Intel SGX [2] provides trusted execution environments called enclaves in user level. Enclaves' code and data reside in a hardware-protected memory named enclave page cache (EPC), which is inaccessible to any software including OS/HV or SMI handler. Enclave code can access the memory outside the enclave. As enclave code is only allowed to be executed in user mode, any interaction with devices must execute outside of the enclave and through untrusted system calls. SGX enables a threat model where users only trust the Intel CPUs and the code running inside the enclaves.

Intel SGX SDK provides a function call mechanism for enclaves via ECALL and OCALL. Thus, an application can invoke an enclave's code via an ECALL and get the return values. The enclave can invoke an OCALL to execute a function in the untrusted portion of the application and receive a return value. SGX provisions remote attestation and local attestation [24]. With SGX's remote attestation ability, one can gain confidence that the target application is securely running within an enclave. Local attestation allows an enclave to prove to one another that it is running on the same processor.

B. System Management Mode

System Management Mode (SMM) [9] is the most privileged mode for handling system-wide functions like power management, etc. Upon a system management interrupt (SMI), the CPU saves the system context and switches to SMM, and executes a predefined logic (SMI handler) in system management memory (SMRAM). We deem SMRAM as a special *enclave*, because SMRAM cannot be accessed by OS/HV after initialization, and therefore protects the confidentiality and integrity of SMI handler. An *RSM* (resume) instruction is executed to switch back to the protected mode.

The SMI handler is initialized by UEFI, and SMRAM can be locked by configuring *D_OPEN* and *D_LCK* bits in SMRAM control register. Therefore, the SMI handler is safe from tampering after boot. Moreover, the SMI handler can access all CPU registers, I/O devices, and physical memory (except for SGX EPCs), and thus we expand the SMI Handler to design AURORA's SMVisor in III-A.

C. Trusted I/O Paths

We define *trusted I/O paths* in trusted execution environments (TEEs, e.g. SGX) as follows. Given a trusted application (e.g. an SGX enclave) and a trusted peripheral device, a trusted I/O path is a secure channel of data transfers between them, even in the face of software adversaries such as host kernel, hypervisor, guest VM and kernel drivers. We classify trusted I/O paths into two categories based on their characteristics:

1) **Data-provider/-consumer type.** Although SGX provides user-space strong isolation for program end-points (i.e. enclaves), most of the devices provide and/or consume plaintext messages. Cases vary from human input data (e.g., keyboard scan code) to outputted data (e.g., a stream of bits to printer/display), which cannot guarantee the confidentiality and integrity in an untrusted

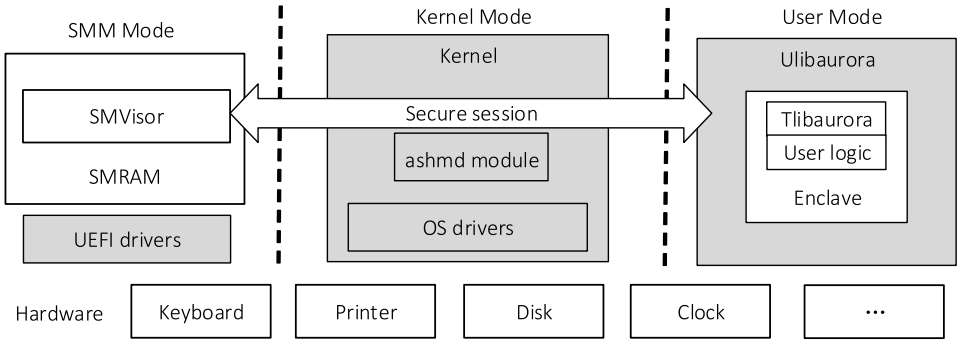

Fig. 1. AURORA's architecture. The gray areas denote untrusted parts, and the white ones represent AURORA's trusted computing base.

or compromised system. Therefore, for trusted paths of this kind, AURORA encrypts/decrypts in SMRAM the data from/to devices for enclaves to protect the data from the untrusted system.

2) **Data-storage/-transmitter type.** Storage disks and network adapters are typical storage/transmitter devices. Both kinds of devices can deal with encrypted data without knowing the semantics of the data. However, both devices need a pile of software stacks (e.g., file system, network protocol) to work correctly and remain compatible. Building trusted paths for them is very challenging. One possible approach is to decompose existing software stack and port them into the trusted applications, which will greatly swell the TCB and attack vectors [25]. As an alternative, AURORA leverages a *verify-then-use* method that reuses existing software stacks in a secure manner (III-C).

The main difference between these two types is that *data-provider/-consumer* is either the sink or the source of data and must handle data in plain-text, which requires careful isolation or confidential protection; Whereas the *data-storage/-transmitter* type acts as a medium for data at rest or in transition. It focuses on the freshness and integrity of data.

D. Threat Model and Assumptions

We consider an adversary who can compromise the system via software attacks. He can directly attack enclave interfaces visible to the OS, and actively reconfigure any devices (e.g., modify MMIO region, or change the operating mode of a device) and perform arbitrary operations (e.g., trigger interrupts, issue DMA write requests) using any I/O commands.

AURORA requires a modern Intel platform with SGX and TPM support. We assume the hardware and SMRAM firmware (the microkernel we implement) of the host machine is trusted. We do not trust the UEFI drivers but verify whether they operate following their specifications, or perform unintended operations: e.g., intercept bus traffic, or write to an address that is not specified in DMA commands. Our assumptions reduce the TCB on the target host as modern commodity system usually contains a large software stacks (e.g., UEFI, possibly hypervisor, kernel and other applications) when dealing with I/O devices. Physical attacks on devices, side-channel attacks [26], [27], information leakage (I/O size and request/response frequency) and denial-of-service attacks to Aurora's framework are not considered. The adversary cannot access the physical machine and modify its hardware setup.

III. DESIGN

To provide trusted I/O paths, we employ the combination of SGX and SMM, both of which are processor enforced protection features. Because they are offered at the opposite ends of Intel privilege-level model, we name our framework after AURORA.[1]

Establishing a trusted path in AURORA consists of the following three steps. First, AURORA builds a secure session between SMVisor and an enclave that performs an I/O request. Second, SMVisor receives the request via the secure session without leaking knowledge to the untrusted system. Third, SMVisor interacts with the target device on behalf of the enclave, and performs desired I/O operations for the request. The procedure is conceptually similar to a *syscall* or *vmcall*. We name this an *smcall* for brevity.

A. Architecture

AURORA is composed of three components: SMVisor, libaurora and a secure session established between SMVisor and libaurora, as shown in Figure 1.

1) SMVisor takes charge of the target devices from the untrusted kernel. It dispatches device interrupts to either kernel or enclaves. SMVisor securely invokes UEFI drivers outside of SMRAM (III-C).
2) Libaurora consists of two parts. The trusted part (Tlibaurora) complements unsupported I/O APIs in Intel SGX SDK library, helping developers secure enclave's I/O paths. The untrusted part (Ulibaurora) is responsible for forwarding *smcalls* to SMVisor.
3) The secure session exchanges encrypted messages between SMVisor and an enclave via a shared memory provided by a kernel module named ashmd. AURORA leverages a 5-tuple: *(ENCLID, DEVID, OPTYPE, PAYLOAD, MAC)* for each message. The first two arguments indicate the program endpoint and device endpoint respectively. The third stands for the type of I/O operation. The fourth carries the message data and

[1] Aurora takes place in the polar regions, *i.e.,* south and north of the earth.

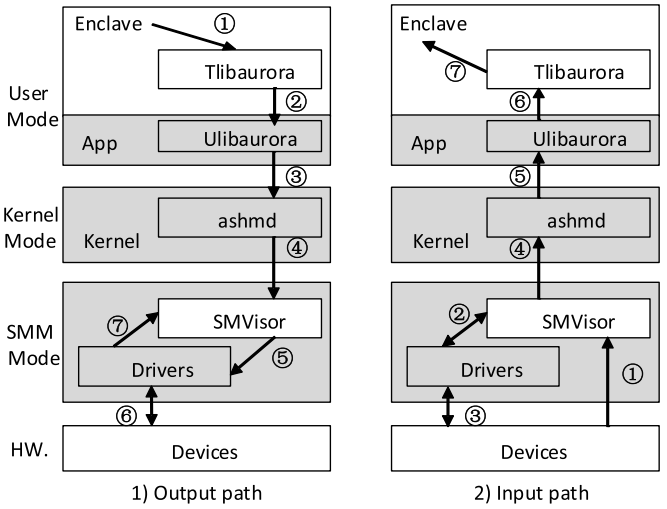

Fig. 2. The workflow of a trusted I/O path in AURORA. 1) An **output** request is triggered by an enclave as an exception: ① The user logic in an enclave issues an output request by invoking Tlibaurora's APIs; ② Tlibaurora marshals and encrypts the request's arguments, and invokes Ulibaurora outside the enclave to relay the request; ③ Ulibaurora *ioctls* the ashmd module to trigger an *smcall*; ④ SMVisor is notified and obtains the encrypted requests from the secure session; ⑤ SMVisor decrypts the request and invokes the UEFI driver; ⑥ The driver processes output data in PCI space or MMIO buffer; ⑦ The driver returns a result code to SMVisor. In the end, SMVisor executes *RSM* to return to protected mode. 2) An **input** request from an enclave causes a device interrupt: ① The target device triggers an interrupt, and the control flow is rerouted to SMVisor which has the interrupt handler; ② SMVisor invokes the UEFI driver to handle the input request; ③ The driver gets the raw input data and sends it to SMVisor; ④ SMVisor encrypts the data inside SMRAM, and sends it to the target enclave via the secure session; ⑤ The system switches back to protected mode and the ashmd module uses a signal to notify Ulibaurora; ⑥ Ulibaurora invokes Tlibaurora to receive the encrypted data; ⑦ The enclave decrypts the encrypted data and finally obtains the raw input data.

the last is the message authentication code used for integrity protection. All the arguments are marshaled and encrypted.

To help understand the workflow of a trusted path in AURORA, we break down the process of the output path and input path respectively into sequential phases, as illustrated in left and right part of 2.

Like library OSes [3], [14], [15], AURORA has few interfaces with the untrusted OS. The data plane of an enclave is protected by advanced encryption schemes accelerated by the processor (*i.e.,* AES-GCM with AES-NI support), and the data flows between the enclave and SMVisor are completely invisible to the OS. Unlike library OSes, which delegate I/O operations to the untrusted OS, AURORA ships security-sensitive parts of these operations into SMVisor, thus eliminating the reliance on untrusted OS.

B. SMVisor

SMVisor is the core component in AURORA. It may take charge of multiple devices simultaneously, therefore it needs to determine which device is being requested and then invokes the corresponding driver.

1) Interrupt Handler: To deal with a device interrupt, SMVisor reroutes the interrupt to SMI. Specifically, SMVisor intercepts device events by configuring the *Redirection Table* defined in I/O APIC and modifying the destination of the device event to SMI. SMVisor distinguishes the interrupt source and thus notifies a certain enclave or the untrusted kernel. In order to forward the interrupt to upper software layer, it issues an inter-processor interrupt (IPI) by writing the interrupt command request (ICR) register in the local APIC.

To perform I/O operations correctly, a driver should know the exact I/O ports of the target device. At the time of system boot, SMVisor collects all device information and records their memory-mapped I/O (MMIO) base addresses into a *PCI BAR Table*, thereafter the matching drivers can manipulate the mapped PCI configuration space according to the table.

2) Sanitization Module: SMVisor verifies the validity of parameters passed to the drivers (e.g. a PCI structure pointer should point to one of the entries in the *PCI BAR Table* within SMRAM) and checks the return values according to the device specifications (e.g. each driver function has a specific range of error codes) as well as its functional correctness. When the parameters or return values are invalid, SMVisor notifies the enclave that a potential attack from OS may occur, and rejects the subsequent requests. Moreover, SMVisor checks the memory boundary (i.e. starting address and region range) that the driver tries to access to ensure that it can not write data outside of SMRAM. Furthermore, a driver may require more memory space during runtime (e.g. for DMA with devices), therefore SMVisor maintains a dynamic heap with a sanity-checking feature [28]. To avoid possible memory leaks, the manager frees all allocated memory for the driver before returning to the protected mode.

C. SMM Drivers

In this section, we describe how SMVisor safely reuses existing drivers. In modern monolithic systems, OS drivers are coupled with the kernel. These drivers lack of effective isolation from the rest of the untrusted systems, therefore we do not use them. Instead, since existing open-source UEFI firmware, such as Coreboot[2] and Project Mu,[3] contains various drivers in their Driver eXecution Environment (DXE), and the code of these drivers are significantly scrutinized compared to the proprietary code, we reuse these drivers and forward the device interrupt for them to handle. This OS-neutral design allows AURORA to support different kinds of OS, e.g., Linux, Windows, or macOS.

Furthermore, we reuse UEFI drivers in a secure manner. At the booting stage, SMVisor calculates the hash checksum of each driver and stores its integrity signature inside SMRAM. When receiving an I/O request, SMVisor verifies the integrity of the corresponding driver to ensure it is not tampered with. When executing a driver, SMVisor puts its data, heap, stack segments inside SMRAM, and thus ensures them inaccessible to OS/HV. Moreover, SMVisor uses its sanitization module to validate the behavior of the driver, as described in III-B.2.

[2]https://www.coreboot.org/
[3]https://microsoft.github.io/mu/

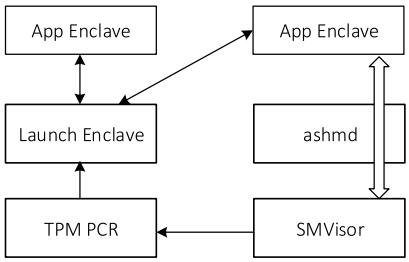

Fig. 3. Mutual attestation and key agreement between SMVisor and an application enclave. The single line arrows denote the attestation direction, the double line arrow denotes the following key agreement.

D. Kernel Module (ashmd)

SGX memory model is asymmetric: the processor prohibits code outside of enclave to access the enclave, whereas the enclave code can access addresses outside the enclave. AURORA leverages shared memory to bridge SMVisor and enclaves. For instance, we use a kernel module *ashmd* to allocate contiguous physical memory as shared memory. For an enclave program, Ulibaurora opens and mmaps this device into its address space. As a result, Tlibaurora can directly operate on this memory in enclave mode. If the kernel launches memory-based Iago attacks [10] such as mapping to an illegal location, Tlibaurora will verify whether the whole mapped memory is outside of the enclave, otherwise the secure channel will not be established.

In order to distinguish interrupts to enclaves from others, ashmd module allocates an IRQ from the system. When ashmd receives an IPI from SMVisor, it knows this is a response to an enclave. Moreover, to prevent OS from modifying I/O redirection table, ashmd module spawns a kernel thread which holds a copy of I/O APIC table and continually check the status of I/O APIC table. Once it detects a modification by OS, it tries to restore it with the reserved one. If the attempt fails, ashmd immediately notifies the target enclave with an error code. We treat the denial of ashmd thread scheduling as a kind of DoS attack, which is beyond the scope of the paper.

E. Secure Session

We design a secure session that protects the message exchange between SMVisor and enclaves. The initial stage of the secure session is to ensure whether both parties are trusted or not. On one hand, the untrusted privileged system may launch SMI-specific fuzzing attacks on SMVisor. On the other hand, the privileged system may fake the identity as SMVisor since it can emulate an SMI and try to *handshake* with an enclave. To mitigate such a man-in-the-middle attack, we introduce *mutual attestation*. When each of both parties trusts the other, they can exchange a symmetric secret key using *key agreement*, as depicted in 3.

1) Mutual Attestation: During a measured boot, Intel Boot Guard [29] hashes the firmware and stores its cryptographic hash in the TPM Platform Configuration Register (PCR) [30]. The final PCR value reflects the whole boot process. If the process is tampered with, the PCR value will differ. In order to ease SMVisor attestation, we use the launch enclave provided by the Intel SGX Platform Software to attest SMVisor for one-time effort. Afterwards, any application enclave running on the same machine can query the launch enclave to know if SMVisor attestation has succeeded. To attest SMVisor, the launch enclave needs to verify the PCR value by requesting a TPM quote, which includes a cryptographic signature over the PCR value alongside with a fresh nonce. This ensures the integrity of the PCR value and prevents replay attacks. Besides, an application enclave attests itself to the launch enclave using Intel's local attestation.

2) Key Agreement: After the mutual attestation, the SMVisor and an enclave establish an authenticated secure channel via an ephemeral Diffie-Hellman key exchange. We then use AES-GCM encryption scheme for further message exchange in order to prevent replay attacks. Intel processors with SGX feature support Advanced Encryption Standard New Instructions (AES-NI) to accelerate AES encryption and also can mitigate side-channel attacks [9].

3) Termination: When an enclave finishes its requests, its Tlibaurora logic will notify SMVisor to terminate the secure session and release resources. When there remains no live session, SMVisor will disable its interrupt routing and thereafter make no impact on the system.

F. Security Enhancement and Performance Optimization

To provide better security and performance, AURORA introduces *hardware encryption*, *data obliviousness* and *batch mechanism for smcalls*.

1) Hardware Encryption: Intel extends x86 ISA with AES-NI and claims to prevent known side-channel attacks. We make use of it to address the same concern and use a constant-time AES-128-GCM algorithm to defeat cache timing attacks. Nonces are used for replay proof. This hardware accelerated encryption also decreases the preemption time in SMM.

2) Data Obliviousness: In an established session, if an enclave requests the same I/O path twice and the results happen to be the same, the encrypted messages will also be identical because the session key is not changed. An adversary can infer secrets by observing such side channels. To prevent such secrecy leakage, AURORA pads all messages with random values to the same length (4KB, same size as one EPC page), which restricts possible information leakage at page-level. The maximum length of a message is 4080 Bytes, leaving at least 16 bytes for random padding.

3) Batched SMCALLs: To reduce the context switches between SMVisor and enclaves, AURORA supports a batched *smcalls* mechanism using two dedicated, lock-free ring buffers for input and output respectively. Enclaves can set a desired threshold, so Tlibaurora will coalesce these *smcalls*. For asynchronous I/O requests, the mechanism can benefit the enclave with higher throughput.

IV. TRUSTED I/O PATHS BETWEEN ENCLAVES AND DEVICES

As described in section II-C, trusted I/O paths should assure that the users' data is transferred by a secure session.

In this section, we illustrate how to build trusted I/O paths for data-provider/consumer devices, e.g. human-interface-device (HID) keyboards, serial printers, and hardware clocks, and data-storage/transmitter devices, e.g. portable USB disk.

A. Trusted HID Keyboard Input

Trusted input ensures that third-party software on the same computer has no access to the user's input intended for an enclave application. In other words, trusted input paths exclude other untrusted software including the underlying OS from sharing the input device. We use AURORA to secure user's private textual inputs from the keyboard to the enclave.

1) Trusted Input Mode: When an enclave issues a trusted input request, SMVisor routes the keyboard interrupt (IRQ 1 by default) to SMI. All of the subsequent keyboard scan codes are then cached inside SMRAM. We call this case *trusted input mode*. After an *enter* key is hit and released, SMVisor recovers the I/O APIC redirection table and exits trusted input mode. Finally, SMVisor encrypts the input and sends it to the secure channel. The desired enclave obtains it from the channel and decrypt it.

2) User Verification: Attackers can use keyloggers to steal passwords or credit card information. Most keyloggers are not stopped by HTTPS encryption which only protects data in transit between computers. For example, they are frequently implemented as rootkits that subvert the operating system kernel to gain unauthorized access to the hardware. To mitigate this threat, we leverage the PC speaker to indicate that the trusted path has been established. When the system first boots, the network is disabled by the SMVisor, and we assume that the loaded kernel is clean due to TPM's measure boot. AURORA asks a user to store a short, arbitrary MIDI-format melody (around 5 seconds) inside SMRAM via a configure enclave, so that every time when a trusted path is set up, the PC speaker will play the melody, and then the network is enabled by the SMVisor. Since the melody is a pre-shared secret only known by SMVisor and the user, it is impossible for the attackers to learn and mimic this process.

B. Trusted Serial Printer Output

Shared print services are very common today, by which colleagues in a unit or students in a college print their documents using the same physical machine. Users' private or sensitive data may also be printed using such infrastructures. Malware or compromised systems running on these shared machines can steal these data for malicious purposes like industrial espionage or illegal data acquisition. To mitigate this threat, we build trusted serial-port path for shared print services.

When an enclave requests the trusted printer service, it encrypts the output packet and sends the encrypted packet to SMVisor. SMVisor receives the packet and decrypts it to plaintext packet. As the packets are usually held with printer-specific format, SMVisor issues the corresponding serial port driver and adds this job into its task queue. Afterwards, SMVisor returns the status code from the printer devices to the enclave end-point.

C. Trusted Hardware Clock

The integrity and non-forgeability of the clock value is much more vital. In this section, we describe how to extend AURORA for trusted, absolute, high-resolution and attack-aware clock.

1) Absolute Value: We use Real-Time Clock (RTC) to provide the absolute wall-clock time (i.e. epoch) for enclaves. To avoid incorrect values due to hardware updating, SMVisor keeps obtaining the RTC values until the last two are consistent. After obtaining a valid wall-clock, SMVisor then obtains the rest timers. The order is critical because RTC time retrieval is the most expensive amongst others and will break the freshness of other timers' values.

2) High Resolution: When a time value with high resolution (e.g. micro-second) is required, AURORA uses invariant time stamp counter (TSC) and High Precision Event Timer (HPET). For instance, AURORA uses the following formulae to compute the *tv_usec* value in *timeval* data structure: $tv\_usec = (HPET\_value/HPET\_Hz)/1,000,000$; where $(HPET\_value/HPET\_Hz)$ computes the absolute time in micro-seconds. SMVisor references invariant TSC to estimate the latency of hardware response and adjust the value. Invariant TSC ensures that the duration of each clock tick is uniform.

3) Attack-Awareness: Though hardware clocks can be controlled and modified by the malicious OS, diverse timers in Aurora offer enclaves the ability to validate the credibility of time values. Tlibaurora records last values read from all hardware clocks and obtains current values to calculate their differences respectively, and then uses clock frequencies to compute the actual elapsed time. When any of the measured time violates the monotonic rule, we assume a time attack.

For the case where the hardware clocks are altered in a consistent way, that is, all the timers are altered in the same offset, we propose a novel algorithm to detect such an attack. We use a dedicated counting thread [31] guarded inside an enclave occupying a logical thread on the same CPU core. Such a sibling thread will block the possibility of side-channel attacks when hyper-threading is active [32]. The counting thread provides dT and the difference of hardware timers provides dt, we use first-order derivative dt/dT to detect the possible time attack. Currently, we set the confidence interval to 0.05, which denotes the possible altering in the timers' rising slope. If the slope changes much, we assume an attack.

Aurora's trusted hardware clock is immune to the TOCTOU attack because it leverages SMM which guarantees that the operations of check and use are atomically performed.

D. Trusted USB Storage

Storage software usually supports block devices and file system. A compromised OS may silently discard data written by the application or return fabricated data during a read operation. It is a serious issue when secure data storage is necessary for ensuring correct operation (e.g., secure logging, APT monitoring and compliance). We extend AURORA to support secure USB massive storage devices as an example. We believe that AURORA's approach is suitable to support other storage systems such as SATA HDD, NVMe SSD, etc.

1) Secure Storage Protocol: When an enclave issues a request for a USB storage device, SMVisor leverages I/O interrupt rerouting mechanism to monitor and deal with every operation of the USB device, including transferring data and control commands via a secure session in AURORA. In Linux, the data is by default transferred using bulk mode and is exchanged based on DMA mechanism, which stores the DMA address and buffers location in a scatter-gather list. The ashmd module obtains the information from the USB driver.

After capturing the USB Request Blocks (URBs), SMVisor passes them to the USB device via a secure session. Note that SMVisor leverages AES-GCM scheme to encrypt transferred blocks between enclaves and devices III-A. All data blocks from the enclave are encrypted by default, and a message authentication code (MAC) is generated for each block. SMVisor checks the MAC of each block to validate its authenticity and integrity. If it checks an inconsistent MAC, SMVisor will return an error to the enclave.

V. IMPLEMENTATION

A. Components

We modified the SMI handler of CoreBoot to implement AURORA's SMVisor, and modified the drivers of CoreBoot project to export interfaces which can be invoked by SMVisor. We disable the caching mechanism in SMM and check the access permission using the memory type range registers (MTRR) to ensure SMVisor's protection. For the sake of compatibility, Tlibaurora supports standard POSIX APIs, so enclave developers need not write OCALLs.

To avoid machine check exception within SMM, we enable the Streaming SIMD Extensions (SSE) bit in control register CR0. With SSE and AES-NI, we boost encryption performance. For example, with AES-NI support, AES-256-GCM encryption and decryption on 4KB data in SMM costs 9us on average, for comparison, without AES-NI, AES-128-GCM implementation of Intel IPP Crypto library [33] costs 597us.

We implement a specific kernel model ashmd. Like Linux SGX driver, we use but do not trust them. The ashmd module is intended to relay the communications between enclaves and SMVisor.

B. Trusted Computing Base

AURORA's trusted components (i.e., SMVisor, ashmd, Tlibaurora, Cryptography) have 3393 lines of C code, and untrusted component (i.e., Ulibaurora) has 273 lines of C code. To avoid exploitable vulnerabilities, we use static analysis tools, e.g., Clang Static Analyzer,[4] over AURORA's code base and fix all found bugs. Besides, as AURORA is quite small and its complexity is obviously lower than that of the formally verified microkernel [34], we believe it is reasonable to put AURORA within reach of formal verification.

On our platform, the system-wide EPC limit is approximately 93MB and the SMRAM is 4MB by default. Both are scarce memory resource. We therefore measure the final size of the resulted stripped binary images. Our modification on CoreBoot only adds 3.9% on its total size (120.2KB V.S. 115.7KB). The enclave.so linked against Tlibaurora is 696KB in size and 1MB including runtime stack and heap manager. Smaller codebase denotes smaller TCB introduced.

TABLE I
MEASURING AURORA'S OVERHEAD BY CALCULATING PI

Interval	Time cost (s)		Overhead ratio	
	70us	150us	70us	150us
Baseline	8.545	8.635	0	0
1000ms	9.045	9.025	5.85%	4.52%
100ms	9.605	9.590	12.40%	11.06%
10ms	10.180	10.455	18.90%	21.08%

C. Limitation

Although SMM-based mechanism can be used to route many peripheral I/O events, it itself cannot be used to protect the framebuffers and therefore fails to provide a trusted display. This is because SMRAM area by default starts at the same address as the VGA space, therefore SMVisor cannot obtain the framebuffer. It can be solved by redirecting SMRAM to another higher memory area without overlaying the VGA space. However, as SMM-based protection has performance implications [35], we recommend that users choose Intel PAVP (assisted by Intel ME) and SGX [36] for real-time scenarios such as video chattings or conferences.

VI. EVALUATION

In this section, we evaluate the performance of AURORA framework and the trusted I/O paths. Our experiments are carried out on an Intel Core i7-7700HQ 2.8GHz CPU with 16GB of memory, running Ubuntu 16.04 LTS and SGX SDK 2.4 and SGX driver 1.9.

A. Framework Overhead

We select pi_css5[5] program, which has no interaction with the external system, to measure the performance of AURORA with as little noise as possible. We ran the program till achieving 4,000,000 digits of precision, and used two durations (70us and 150us) to emulate different real-world environments.

Specifically, we ran two threads: one thread (T1) running Pi number calculation, the other thread (T2) requesting the ashmd module to trigger an SMI in a given interval. Firstly, we ran T1 without the interference of T2 and took the time cost as a baseline. Then we started T2 to see the slowdown of T1. From Table I, we observe that when the interval decreases, the overhead increases. The worst case of overhead is around 21%, where *smcalls* are requested too frequently. Such a situation is rare in real world applications because an enclave has to execute its own logic. We observe that intensive requests may impact the time-sensitive tasks on the same system. Since *smcalls* are forwarded by ashmd module, it can use a threshold to block a malicious enclave from abusing. In our current setting, we allow an enclave to request

[4]http://clang-analyzer.llvm.org/index.html

[5]http://myownlittleworld.com/miscellaneous/computers/piprogram.html

TABLE II
SERIAL-PORT BENCHMARK COMPARISON BETWEEN LINUX AND AURORA

Payload Size(KB)	Linux(s)	Aurora(s)	Speedup(%)
4	0.03	0.03	0.0
64	0.32	0.24	25.0
256	1.08	0.87	19.4
1024	3.95	3.43	13.2

up 10,000 times per second, which introduces about 11% overhead to the system.

B. Security Evaluation

We evaluate AURORA's reliability with several security tests.

1) Keylogging Attack: To evaluate the effectiveness of AURORA's trusted input path, we deployed multiple spyware to get a user's inputs under trusted input mode, including a stealthy root keylogger,[6] a rootless X11-based keylogger ixkeylog,[7] a kernel-level rootkit keylogger,[8] and a remote keylogger.[9] None of them can obtain the user's inputs.

2) Output Corruption Attack: We leveraged a DMA-based NIC device to launch output corruption attack when using Aurora's trusted serial printer service. Since SMRAM and EPC RAM are strongly isolated by architectural protections, only meaningless content (0xFF) is obtained. No secrets are leaked. We then conducted another DMA attack to corrupt SMM drivers outside of SMRAM. SMVisor detected its integrity tampered and rejected subsequent requests, then notified the enclave application. Moreover, we conducted another DMA attack to corrupt the content of shared memory of the secure session. Both SMVisor and Tlibaurora can verify the broken MAC and restart a new session.

3) Time Deception Attack: To evaluate the trusted clock, we intentionally modified the RTC when an enclave issued a request for trusted clock. Tlibaurora detected the attacks successfully and indicated a failure. For instance, `gettimeofday()` returns -1 and sets the global variable *errno* appropriately. Existing C library specification does not have a flag indicating time attacks, the enclave logic may keep requesting until obtaining a valid time value. We support failure handling in Tlibaurora to help enclaves deal with such situations.

4) Storage Deception Attack: We modified the kernel driver of SCSI device and randomly dropped several disk I/O requests intentionally. SMVisor detected such accidental conditions and notified the Tlibaurora of corresponding enclave. In our current setting, the Tlibaurora will automatically try issuing the lost I/O requests three times so as to avoid storage failure. In our experiments, the enclave finally received the attack warning from the Tlibaurora and logged these events for further forensic analysis. Note that denial-of-service to Aurora's kernel module is not considered in this work.

[6]https://github.com/vim2meta/keylogger
[7]https://github.com/dorneanu/ixkeylog
[8]https://github.com/arunpn123/keylogger
[9]https://github.com/EinBaum/Totally-Not-A-Virus

TABLE III
STORAGE THROUGHPUT COMPARISON BETWEEN LINUX AND AURORA

Payload Size(MB)	Linux (MB/s)	Aurora (MB/s)	Slowdown (%)
4	32.9	31.8	3.3
64	40.9	39.2	4.2
256	42.7	40.4	5.3
1024	53.2	47.5	10.7

TABLE IV
BREAKDOWN OF THE TRUSTED CLOCK SERVICE

Action Step	Time Cost(us)
EPC encryption	2
Copy to DRAM	2
Switch to SMM	13
Copy to SMRAM	0
SMM decryption	3
Clock Service	44
SMM encryption	3
Copy to DRAM	0
Return and enter SGX	12
Copy to EPC	3
EPC decryption	2

C. Serial Port Output Benchmark

It is rather difficult to measure the overall time during which a printer accepts and finishes a print job, because we have to re-flash the printer firmware to hook its time function. Therefore, we redirected the serial port to a local file. To compare the serial output performance between native Linux and AURORA, we measured the time cost of writing the test data of different sizes to the serial-port address in Linux and in AURORA, respectively. Table II shows the experimental results. AURORA performs better than native Linux because AURORA uses batch mechanism (III-F.3) to reduce the frequent mode transitions while Linux has more user/kernel mode transitions when writing large data to a serial port. Note that, for the larger output data, the speedup decreases due to cryptographic computations in AURORA.

D. Trusted USB Storage Benchmark

To compare the overhead introduced when applying AURORA's trusted I/O path on USB storage devices, we wrote variable sizes of contents to a USB storage device on native Linux system as the baseline, by using `dd` program with options `fsync` and `direct`. To avoid the side-effect of any buffer mechanism, we use Linux raw device to mount and initialize the USB device. The throughput results are shown in Table III where the size of each block is set to 4KB. AURORA achieves approximately 90% of the baseline throughput when writing 1GB to the device. The major overheads are contributed by SMM/Protected mode transitions and cryptographic computations on each block.

E. Trusted Clock Benchmark

For this experiment, we measured the complete trusted clock procedure for 1,000 times and used the average value.

TABLE V

COMPARISON OF EXISTING CLOCK SERVICES THAT CAN BE USED FOR SGX ENCLAVES. CLOCK SERVICES INCLUDE NETWORK TIME PROTOCOL (NTP), PRECISION TIME PROTOCOL (PTP), AND THE PLATFORM SOFTWARE (PSW) SUPPORTED BY INTEL MANAGEMENT ENGINE (ME), MONOTONIC COUNTER SUPPORTED BY BYZANTINE FAULT TOLERANCE (BFT) PROTOCOL

Clock Provider	Approach	Type	Resolution	Request Cost	Latency	Security	Use Cases
Remote Clock	NTP/PTP	absolute	1s	>100ms	high	trusted	Town Crier [17]
Intel ME	PSW	relative	1s	10.3ms	medium	trusted	SGX-Tor [37]
OS Kernel	OCALL	absolute	1ns	6us	low	untrusted	Panoply [15]
Distributed Network	BFT	N/A	N/A	1–2ms	medium	trusted	ROTE [23]
Hardware Clock	*smcall*	absolute	1ns	69us	low	trusted	Aurora-OpenSSL VII-B

Table Table IV shows the time cost for each step in a trusted clock path. We can see that the clock service takes most of the time. This is because 1) the RTC driver has to read clock value twice to check if the clock is updating its time data; 2) The RTC involves several `INS` instructions to read each field of a calendar time, and costs 14us to obtain a complete wall-clock time in one request; 3) The other 4 timers cost roughly 10us in sum. The total time cost (84us) satisfies the requirement of real-world trusted timestamps.

For comparison, we measured the performance of other clock providers. The results are shown in Table V. Among existing approaches, AURORA outperforms remote clock (the 2nd row) and Intel's reference clock (the 3rd row) by introducing very low latency, while achieving the same resolution (nanosecond-level) as OS services (the 4th row).

VII. CASE STUDIES

Aurora provides uniform abstractions (standard POSIX APIs) for the SGX applications, which need not be modified, to maintain platform-independence and thus achieve nice usability. We implemented trusted I/O paths for three real-world applications to illustrate the usability of AURORA.

A. Secure OpenSSH Login

Secure Shell (SSH) is widely used for shell access on UNIX-like systems. An SSH server uses a username and a password to authenticate an SSH client. If the client is infected with malware, the login credential can be stolen by key-loggers. To demonstrate the effectiveness of AURORA's trusted input path, we modified OpenSSH 7.7 to protect its client login process. Specifically, we ported passphrase related logic into an enclave, and used AURORA's *trusted input path* to get passwords which users input. We call the enhanced client as Aurora-OpenSSH client.

Aurora-OpenSSH client first requests SMVisor to intercept the user's keyboard events. This interception adds around 13% overhead per keystroke. Only after SMVisor routes the keyboard interrupt into itself and acknowledges to the client, will the prompt `"username@hostname password:"` be displayed on the screen. While the user is typing the passphrase, the client does not display any content on the screen, achieving the same effect as in the native OpenSSH client.

The user's input is buffered inside SMRAM. When the user releases `Enter` key, the password is then encrypted and sent to the secure channel, and SMVisor exits trust input mode. Aurora-OpenSSH client reclaims the password message and sends it to the server. Note that the plaintext password is only available in SMRAM and enclave. The path between SMRAM and enclave is protected by AURORA's secure channel and the path between Aurora-OpenSSH client and OpenSSH server is protected by TLS session.

TABLE VI

PERFORMANCE COMPARISON BETWEEN NATIVE SGX-OPENSSL AND AURORA-OPENSSL

	SGX-OpenSSL	Aurora-OpenSSL	Slowdown(%)
Certificate Generation	77.4ms	102.5ms	32.4
TLS Handshaking	2.8ms	3.4ms	21.4

B. Secure OpenSSL Session

SSL/TLS protocols are in widespread use in web browsing, email, instant messaging, and voice-over-IP, etc. An SSL/TLS session starts with SSL/TLS handshake protocol, including authentication, key exchange, etc. The procedures rely on system date and time. The SSL/TLS server needs to provide a certificate with a timestamp indicating the expiration time, and the SSL/TLS client needs to verify this certificate. Most of all, both parties require a reliable time source to detect possible timeout. For instance, both *ClientHello* and *ServerHello* messages use the system time to prevent replay attacks. If the time is emulated by an untrusted source, it may compromise the security of OpenSSL sessions.

One of the in-enclave SSL/TLS projects is SGX-OpenSSL[10] from SGX-Tor project [37]. It does not trust the system storage, therefore its server side creates the certificate inside the enclave each time it receives a new connection request from the client side. We enhanced the TLS handshake by using AURORA's trusted clock. Specifically, we replaced time-related OCALLs in the original project with AURORA's libaurora. We measured the time cost of certification generation at server side and TLS handshake at client side for 10,000 runs and take the average value. We also measured the native SGX-OpenSSL using original OCALLs. As shown in Table VI, in our LAN setting, Aurora-OpenSSL introduced about 32% latency in certificate generation and 21% in the TLS handshaking because both of them use AURORA's trusted clock. We anticipate the overhead ratio to decrease in a WAN environment, because the delay of a geographically distributed public network should be larger than our LAN setting.

[10]https://github.com/sparkly9399/SGX-OpenSSL

TABLE VII
TPC-H BENCHMARK COMPARISON BETWEEN NATIVE SGX-SQLITE AND AURORA-SQLITE

Query	SGX-SQLite(s)	Aurora-SQLite(s)	Slowdown(%)
02	4.68	5.10	8.97
11	1.29	1.42	10.08
16	1.08	1.10	5.30
17	91.26	93.30	2.23
18	24.98	25.91	3.72
19	93.15	98.25	5.47
21	39.68	43.16	9.04

C. Secure SQLite Database

Databases are widely used to store financial transaction records and other sensitive information. Some databases are built upon block devices, while maintaining its own caching mechanisms and storage policies without support of OS's file system, e.g. SQLite database.

We integrated AURORA's Tlibaurora into existing SGX-SQLite projects[11] and added code to support standard file system calls such as *open*, *read*, *write*, *stat*, *lstat*, *fstat*, *fsync*, *fcntl*, *ftruncate* and *unlink*. We then eliminated the file-related OCALLs from the untrusted system. To protect the data, Tlibaurora's shim layer uses a symmetric key derived from user password and in-enclave nonces to encrypt the file payloads. To avoid password leak, we use AURORA's trusted input path to obtain users' passwords from keyboard devices. Moreover, AURORA's secure session protects the confidentiality and integrity of data between SQLite enclaves and SMVisor. When SQLite invokes Tlibaurora file APIs, it opens the USB storage device in direct I/O mode and reads/writes data using 4KB block size. Finally, we set the synchronize flag to `FULL` to avoid the uncertainty of I/O caching.

We evaluated the enhanced SQLite performance by running standardized TPC-H benchmark[12] over a large database. These queries include *select from, order by, group by* benchmarks. We make minor changes to these query scripts to make them possible to be executed in SQLite. Table VII shows the completed queries[13] and their results of native SGX-SQLite and AURORA-SGLite respectively. In our experiments, no more than 10.08% overhead is introduced because of the extra encrypt/decrypt computation and mode transitions in AURORA. Note that these are the worst-case overheads (because we have set `PRAGMA synchronous` to `FULL`) as almost all databases have their own optimized caching mechanisms for better performance.

VIII. RELATED WORK

Our technique was inspired by Scotch [38] which is the first work that combines SGX and SMM to audit VM resource usage. Its motivation differs from ours. Similar to AURORA's SMVisor, previous research efforts such as HyperCheck [39], HyperSentry [40] and IOCheck [41], also employ a dedicated SMI handler to protect the integrity of the OS kernel, hypervisor or firmware, respectively. We next discuss related work of other trusted I/O architectures and trusted paths.

A. Trusted I/O Architectures

SGXIO [25] discusses how to provide trusted generic I/O paths for SGX enclaves. It introduces a trusted hypervisor to assist the establishment of the trusted path, and requires users to port drivers into enclaves. However, SGXIO is a conceptual design without any implementation and evaluation. It is hard for users to port commodity drivers into an enclave because 1) enclave mode does not support privileged instructions; 2) it is unclear how to partition drivers. In contrast, AURORA's drivers are invoked in SMM with no instruction restricted. Drivers are regulated by SMVisor protected inside SMRAM. This reusing technique allows AURORA to be deployed easily.

Zhou et al. [42] introduce a microhypervisor named Wimpy to build trusted paths between a program and a device. This work requires the program to contain the driver associated with the device. It admittedly claimed that running drivers in user model (Ring 3) is difficult. AURORA and Wimpy share many common security goals, e.g. interrupt isolation and MMIO protection. AURORA has smaller TCB than Wimpy.

Intel platform services (PSW) [18] provisions trusted time and monotonic counters by running a coprocessor (Intel Management Engines). It brings separated hardware (e.g., NIC, RTC and SPI flash) that cannot be controlled by untrusted host system. However, since Intel ME runs concurrently with host CPUs, it is difficult and insecure to implement secure human-interface inputs as AURORA does; the host CPUs controlled by untrusted OS and the ME coprocessor controlled by trusted firmware will race a DMA access to the keyboard buffer, which may corrupt or leak user's confidential input. By contrast, AURORA's SMVisor in nature preempts the untrusted OS.

B. Specific Trusted Paths

1) Trusted Input Paths: On the desktop side, TrustLogin [43] uses SMM to protect user passwords. It caches the password inside the SMRAM; when the user releases the `Enter` key, TrustLogin places the password inside the packets of the NIC buffer. In contrast, AURORA does not need to intercept OS networking services, thus decreasing overall impact surfaces. Fidelius [44] leverages a Raspberry Pi to implement a secure channel between a keyboard and a hardware enclave. The specific LED is turned on when the secure channel is established. AURORA does not need any specialized I/O devices and reuse existing hardware devices. On the server side, SafeKeeper [13] leverages SGX to protect the password databases on untrusted clouds. It uses several strategies to mitigate attacks such as phishing. As Intel SGX is also available on personal laptops and mini PCs (i.e., NUCs), we take advantage of SGX to protect a use's password when he logs in his client computer, which differs from their scenarios. On the mobile side, TrustUI [45] exploits ARM TrustZone to

[11] https://github.com/yerzhan7/SGX_SQLite
[12] http://www.tpc.org/tpch/
[13] Note that not all queries can be executed due to the partial porting of the original project.

build a trusted path between a user and a service provider, because ARM TrustZone can isolate physical memory and peripheral interrupts. In design, SMVisor is similar to the trusted OS inside TrustZone's secure world.

2) Trusted Clock Paths: Intel PSW [18] supports coarse-grained trusted time but does not provide a wall-clock time. AURORA provides high resolution and absolute clock. Moreover, though SGX v2 provides the ability to access RDTSC for enclaves, the malicious kernel can still change the TSC value to make it untrusted. Aurora provides an attack-detection method which can also benefit for SGX v2. ShieldBox [46] achieves high-precision yet low-latency clock service by on-NIC PTP clock, however, the time source is not secure because enclaves cannot detect if NIC-Timer is tampered. By contrast, AURORA can detect time attack via a validation algorithm. Déjà Vu [16] implements a reference clock thread using transactional synchronization extensions (TSX) to protect a trustworthy source of time measurement. Déjà Vu's goal is to detect interrupt-based attacks, while AURORA can provide a general time service for enclaves. Compared to our prior work [47], which provides a trusted clock to enclaves, AURORA proposes a unified framework for different I/O devices and presents more evaluation on real-world applications. For distributed enclaves, ROTE [23] proposes a protocol to provide a monotonic counter, which has high latency than AURORA's trusted clock service.

3) Trusted Storage Paths: AuditedIO [48] leverages SGX and a kernel module to implement verifiable data storage for disk I/O operations. It requires a programmable SSD device and modifies the firmware. By contrast, AURORA aims for a trusted-path framework which can be used to support general storage devices. Jang [49] builds a trusted path between a USB proxy device and an enclave. The proxy device uses LED display for user verification. It plays the role of AURORA's SMVisor while AURORA does not need to introduce dedicated hardware. BASTION-SGX [50] modifies the Bluetooth firmware and establishes trusted paths between Bluetooth devices and enclave programs. It has a similar threat model with AURORA and rules the privileged software (OS, drivers, etc.) out of the TCB. Pesos [51] builds a trusted path between enclaves and cloud-based storage, and supports policy-based object protection. Still, Pesos requires a programmable storage HDD named Kinetic Open Storage, while AURORA does not.

IX. CONCLUSION

We present AURORA, a novel architecture which provides trusted I/O paths on client Intel SGX platforms. Based on AURORA, we design and implement several trusted I/O paths for SGX enclaves, including HID keyboard, serial-port printer, USB mass storage and the hardware clock. To the best of our knowledge, we are the first to provide these *realistic* trusted I/O paths for enclaves. Our implementation demonstrates that AURORA is extensible and transparent with underlying commodity systems. Security and performance evaluations with real-world applications show that AURORA can mitigate several I/O related attacks and provide trusted I/O paths with low overhead.

We made AURORA open-source,[14] so that the community can explore on top of it new secure systems.

ACKNOWLEDGEMENT

The authors would like to thank Kai Huang from Intel Corporation and Shweta Shinde from National University of Singapore for their generous help and constructive advice. The authors also thank the anonymous reviewers for their valuable comments that help improve the article.

REFERENCES

[1] E. W. Felten, "Understanding trusted computing: Will its benefits outweigh its drawbacks?" *IEEE Security Privacy*, vol. 1, no. 3, pp. 60–62, May/Jun. 2003.

[2] F. McKeen *et al.*, "Innovative instructions and software model for isolated execution," in *Proc. 2nd Workshop Hardw. Archit. Support Secur. Privacy (HASP)*, Tel-Aviv, Israel, Jun. 2013, pp. 1–8.

[3] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Trans. Comput. Syst.*, vol. 33, no. 3, pp. 8:1–8:26, Aug. 2015.

[4] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, Savannah, GA, USA, Nov. 2016, pp. 533–549.

[5] S. Arnautov *et al.*, "SCONE: Secure linux containers with intel SGX," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, Savannah, GA, USA, Nov. 2016, pp. 689–703.

[6] H. Nguyen and V. Ganapathy, "Engarde: Mutually-trusted inspection of SGX enclaves," in *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 2458–2465.

[7] H. Nguyen *et al.*, "Cloud-based secure logger for medical devices," in *Proc. IEEE 1st Int. Conf. Connected Health, Appl., Syst. Eng. Technol. (CHASE)*, Jun. 2016, pp. 89–94.

[8] F. Schuster *et al.*, "VC3: Trustworthy data analytics in the cloud using SGX," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA, May 2015, pp. 38–54.

[9] *Intel64 and IA-32 Architectures Software Developer's Manual, Volume 3D*, Intel Corp., Santa Clara, CA, USA, 2016.

[10] S. Checkoway and H. Shacham, "Iago attacks: Why the system call API is a bad untrusted RPC interface," in *Proc. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Houston, TX, USA, Mar. 2013, pp. 253–264.

[11] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *Proc. 18th ACM Conf. Comput. Commun. Secur. (CCS)*, Chicago, IL, USA, Oct. 2011, pp. 215–226.

[12] D. Silver, S. Jana, D. Boneh, E. Chen, and C. Jackson, "Password managers: Attacks and defenses," in *Proc. 23rd USENIX Secur. Symp.*, San Diego, CA, USA, Aug. 2014, pp. 449–464.

[13] K. Krawiecka, A. Kurnikov, A. Paverd, M. Mannan, and N. Asokan, "SafeKeeper: Protecting Web passwords using trusted execution environments," in *Proc. World Wide Web Conf. (WWW)*, Lyon, France, Apr. 2018, pp. 349–358

[14] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *Proc. USENIX Annu. Tech. Conf., (USENIX ATC)*, Santa Clara, CA, USA, Jul. 2017, pp. 645–658.

[15] S. Shinde, D. Le Tien, S. Tople, and P. Saxena. (2017). Panoply: Low-TCB Linux Applications With SGX Enclaves. Internet Society. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/panoply-low-tcb-linux-applications-sgx-enclaves/

[16] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with Déjá Vu," in *Proc. ACM Asia Conf. Comput. Commun. Secur. (AsiaCCS)*, Abu Dhabi, United Arab Emirates, Apr. 2017, pp. 7–18.

[17] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, Oct. 2016, pp. 270–282.

[18] Intel. (2016). *Trusted Time and Monotonic Counters With Intel SGX Platform Services*. [Online]. Available: https://software.intel.com/sites/default/files/managed/1b/a2/Intel-SGX-Platform-Services.pdf

[19] Intel. (2016). *Intel SGXSSL Library*. [Online]. Available: http://math.tntech.edu/rafal/cliff11/index.html

[14] https://github.com/Maxul/Aurora

[20] P.-L. Aublin. (2017). *TaLoS: Secure and Transparent TLS Termination Inside SGX Enclaves*. [Online]. Available: https://www.doc.ic.ac.uk/research/technicalreports/2017/DTRS17-5.pdf

[21] bl4ck5un. (2017). *MBEDTLS-SGX: A SGX-Friendly TLS Stack*. [Online]. Available: https://github.com/bl4ck5un/mbedtls-SGX

[22] Wolfssl. (2017). *Embedded SSL/TLS Library for Applications, Devices, IoT, and the Cloud*. [Online]. Available: https://www.wolfssl.com/

[23] S. Matetic *et al.*, "ROTE: Rollback protection for trusted execution," in *Proc. 26th USENIX Secur. Symp. (USENIX Security)*, Vancouver, BC, Canada, E. Kirda and T. Ristenpart, Eds. Berkeley, CA, USA: USENIX Association, Aug. 2017, pp. 1289–1306. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/matetic

[24] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd Workshop Hardw. Archit. Support Secur. Privacy (HASP)*, Tel-Aviv, Israel, Jun. 2013, pp. 1–7.

[25] S. Weiser and M. Werner, "SGXIO: Generic trusted I/O path for intel SGX," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, 2017, pp. 261–268. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3029806.3029822

[26] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Stealing Intel secrets from SGX enclaves via speculative execution," in *Proc. 4th IEEE Eur. Symp. Secur. Privacy*, Jun. 2019.

[27] S. van Schaik *et al.*, "RIDL: Rogue in-flight data load," in *Proc. S&P*, May 2019, pp. 1–18.

[28] S. Silvestro, H. Liu, C. Crosser, Z. Lin, and T. Liu, "FreeGuard: A faster secure heap allocator," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Dallas, TX, USA, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., Oct./Nov. 2017, pp. 2389–2403. doi: 10.1145/3133956.3133957.

[29] X. Ruan, *Platform Embedded Security Technology Revealed*. Berkeley, CA, USA: Apress, 2014.

[30] *Trusted Platform Module Library. Part 1: Architecture. Family 2.0. Revision 01.16*, TCG, Somerset, NJ, USA, 2014.

[31] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using SGX to conceal cache attacks," in *Proc. 14th Int. Conf. Detection Intrusions Malware, Vulnerability Assessment (DIMVA)*, Bonn, Germany, Jul. 2017, pp. 3–24.

[32] G. Chen, W. Wang, T. Chen, S. Chen, Y. Zhang, X. Wang, T.-H. Lai, and D. Lin, "Racing in hyperspace: Closing hyper-threading side channels on SGX with contrived data races," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2018, pp. 178–194.

[33] Intel Corporation, *Intel Integrated Performance Primitives Cryptography*, 2nd ed. Santa Clara, CA, USA: Intel, 2007.

[34] G. Klein *et al.*, "seL4: Formal verification of an OS kernel," in *Proc. 22nd ACM Symp. Oper. Syst. Principles (SOSP)*, Big Sky, MT, USA, Oct. 2009, pp. 207–220.

[35] B. Delgado and K. L. Karavanic, "Performance implications of system management mode," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Portland, OR, USA, Sep. 2013, pp. 163–173.

[36] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. del Cuvillo, "Using innovative instructions to create trustworthy software solutions," in *Proc. 2nd Workshop Hardw. Archit. Support Secur. Privacy (HASP)*, Tel-Aviv, Israel, Jun. 2013, pp. 1–8.

[37] S. Kim, J. Han, J. Ha, T. Kim, and D. Han, "Enhancing security and privacy of Tor's ecosystem by using trusted execution environments," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, Mar. 2017, pp. 145–161.

[38] K. Leach, F. Zhang, and W. Weimer, "Scotch: Combining software guard extensions and system management mode to monitor cloud resource usage," in *Proc. 20th Int. Symp. Res. Attacks, Intrusions, Defenses (RAID)*, Atlanta, GA, USA, Sep. 2017, pp. 403–424.

[39] J. Wang, A. Stavrou, and A. K. Ghosh, "HyperCheck: A hardware-assisted integrity monitor," in *Proc. 13th Int. Symp. Recent Adv. Intrusion Detection (RAID)*, Ottawa, ON, Canada, Sep. 2010, pp. 158–177.

[40] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky, "HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity," in *Proc. 17th ACM Conf. Comput. Commun. Secur. (CCS)*, Chicago, IL, USA, Oct. 2010, pp. 38–49.

[41] F. Zhang, H. Wang, K. Leach, and A. Stavrou, "A framework to secure peripherals at runtime," in *Proc. 19th Eur. Symp. Res. Comput. Secur. (ESORICS)*, Wroclaw, Poland, Sep. 2014, pp. 219–238.

[42] Z. Zhou, M. Yu, and V. D. Gligor, "Dancing with giants: Wimpy kernels for on-demand isolated I/O," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 308–323. [Online]. Available: http://ieeexplore.ieee.org/document/6956572/

[43] F. Zhang, K. Leach, H. Wang, and A. Stavrou, "TrustLogin: Securing password-login on commodity operating systems," in *Proc. 10th ACM Symp. Inf., Comput. Commun. Secur.*, 2015, pp. 333–344. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2714576.2714614

[44] S. Eskandarian *et al.*, "Fidelius: Protecting user secrets from compromised browsers," 2019, *arXiv:1809.04774*. [Online]. Available: https://arxiv.org/abs/1809.04774

[45] W. Li *et al.*, "Building trusted path on untrusted device drivers for mobile devices," in *Proc. Asia–Pacific Workshop Syst. (APSys)*, Beijing, China, Jun. 2014, pp. 8:1–8:7.

[46] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "ShieldBox: Secure middleboxes using shielded execution," in *Proc. Symp. SDN Res. SOSR*, Los Angeles, CA, USA, Mar. 2018, pp. 2:1–2:14.

[47] H. Liang and M. Li, "Bring the missing jigsaw back: TrustedClock for SGX enclaves," in *Proc. 11th Eur. Workshop Syst. Secur. (EuroSecEuroSys)*, Porto, Portugal, Apr. 2018, pp. 8:1–8:6. doi: 10.1145/3193111.3193119.

[48] N. Balakrishnan, L. Carata, T. Bytheway, R. Sohan, and A. Hopper, "Non-repudiable disk I/O in untrusted kernels," in *Proc. 8th Asia–Pacific Workshop Syst.*, Mumbai, India, Sep. 2017, pp. 24:1–24:6.

[49] Y. J. Jang, "Building trust in the user I/O in computer systems," Georgia Inst. Technol., Atlanta, GA, USA, Tech. Rep., 2017.

[50] T. Peters, R. Lal, S. Varadarajan, P. Pappachan, and D. Kotz, "BASTION-SGX: Bluetooth and architectural support for trusted I/O on SGX," in *Proc. 7th Int. Workshop Hardw. Archit. Support Secur. Privacy (HASPISCA)*, Los Angeles, CA, USA, Jun. 2018, pp. 3:1–3:9.

[51] R. Krahn, B. Trach, A. Vahldiek-Oberwagner, T. Knauth, P. Bhatotia, and C. Fetzer, "Pesos: Policy enhanced secure object store," in *Proc. 13th EuroSys Conf. (EuroSys)*, Porto, Portugal, Apr. 2018, pp. 25:1–25:17. doi: 10.1145/3190508.3190518.