Enhancing Security and Privacy of Tor's Ecosystem by Using Trusted Execution Environments

Seongmin Kim, Juhyeng Han, and Jaehyeong Ha, *Korea Advanced Institute of Science and Technology (KAIST);* Taesoo Kim, *Georgia Institute of Technology;* Dongsu Han, *Korea Advanced Institute of Science and Technology (KAIST)*

https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kim-seongmin

This paper is included in the Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17).

March 27–29, 2017 • Boston, MA, USA

ISBN 978-1-931971-37-9

Open access to the Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation is sponsored by USENIX.

Enhancing Security and Privacy of Tor's Ecosystem by using Trusted Execution Environments

Seongmin Kim, Juhyeng Han, Jaehyung Ha, Taesoo Kim*, Dongsu Han
KAIST Georgia Tech*

Abstract

With Tor being a popular anonymity network, many attacks have been proposed to break its anonymity or leak information of a private communication on Tor. However, guaranteeing complete privacy in the face of an adversary on Tor is especially difficult because Tor relays are under complete control of world-wide volunteers. Currently, one can gain private information, such as circuit identifiers and hidden service identifiers, by running Tor relays and can even modify their behaviors with malicious intent.

This paper presents a practical approach to effectively enhancing the security and privacy of Tor by utilizing Intel SGX, a commodity trusted execution environment. We present a design and implementation of Tor, called SGX-Tor, that prevents code modification and limits the information exposed to untrusted parties. We demonstrate that our approach is practical and effectively reduces the power of an adversary to a traditional network-level adversary. Finally, SGX-Tor incurs moderate performance overhead; the end-to-end latency and throughput overheads for HTTP connections are 3.9% and 11.9%, respectively.

1 Introduction

Tor [35] is a popular anonymity network that provides anonymity for Internet users, currently serving 1.8 million users on a daily basis [13]. Tor provides sender anonymity through multi-hop onion routing/encryption as well as responder anonymity using "rendezvous points" that allow the operation of *hidden services*. It is a volunteer-based network in which world-wide volunteers donate their computation and network resources to run open-source Tor software. At the time of this writing, Tor consists of 10,000 relays, with some relay nodes even known to be run by a variety of law enforcement agencies around the world [5, 15]. However, it is not without limitations.

Fundamentally, Tor is vulnerable when an attacker controls a large fraction of relays; anonymity (or privacy) can be broken if all relays in a circuit are compromised because Tor relays can identify the circuit using its identifiers. To prevent malicious relays from entering the system, Tor exercises a careful admission and vetting process in admitting new relays and actively monitors their operation. At the same time, to make traffic analysis more difficult, Tor relies on having a large number of relays and tries to keep a diverse set of relays spread out world-wide [33, 34], which helps to decrease the chance of selecting two or more relays controlled by an adversary. However, having a large network and keeping all relays "clean" are fundamentally at odds in a volunteer-based network. This is exemplified by the fact that, by design, Tor relays are not trusted; in operation they are carefully admitted and their behaviors are examined by a centralized entity [27, 35].

Even having control over a relatively small number of Tor relays still gives significant advantages to attackers. For example, a malicious adversary can change the behavior by running a modified version of Tor, compromise keys, and/or have access to other internal information, such as the circuit identifier, header, and hidden service identifiers. In fact, many *low-resource attacks* (i.e., attacks that do not require taking control of a large fraction of the network) heavily rely on adversaries acquiring internal information or being able to modify the behavior of Tor relays. These low-resource attacks utilize a combination of multiple factors, such as being able to demultiplex circuits, modify the behavior, and access internal data structures. For example, harvesting hidden service identifiers [27] requires access to a relay's internal state, a sniper attack [43] requires sending false `SENDME` cells, and tagging attacks [60] require access to header information. Selective packet drop [27, 43] or circuit closure [28], used by many attacks, also requires being able to demultiplex circuits with circuit identifiers.

This paper aims to address the current limitations of Tor and practically raise the bar for Tor adversaries by using Intel SGX, a commodity trusted execution environment (TEE) available on the latest Skylake and Kaby Lake

microarchitectures. We ask ourselves three fundamental questions: *(1) What is the security implication of applying TEE on Tor? (2) What is its performance overhead?* and *(3) Is it deployment viable in the current Tor network?*

To this end, we design and implement SGX-Tor, which runs on real SGX hardware. We show that it can effectively reduce the power of Tor adversaries to that of a network-level adversary that cannot see the internal state of Tor components. Specifically, we protect private Tor operation, such as TLS decryption and circuit demultiplexing, from adversaries, so that only the TLS-encrypted byte stream is exposed to them, unlike the original vanilla Tor. We further argue that this has far-reaching implications on the trust model and operation of Tor:

- **Trust model:** Currently, Tor relays are semi-trusted. While they are monitored and vetted during admission and operation, their behaviors are not fully verified. In fact, many attacks are discovered and reported after the fact [20, 43, 60, 68]. With SGX-Tor, behaviors are verified through attestation, and private information is securely contained without being exposed to untrusted parties. This simplifies the vetting and monitoring process, allowing Tor to grow its capacity more easily. This will provide a stronger foundation for Tor's privacy (anonymity).
- **Operation and deployment:** SGX-Tor has significant implications in Tor operation. First, because we can both prevent and detect code modification and forging false information, many attacks can be prevented. Second, because SGX-Tor controls the information exposed to the external world, it helps operational privacy. For example, we can ensure that the consensus document, which lists Tor relays and their states [35], does not leave the secure container (i.e., enclave). This effectively turn all relays into bridge relays, a set of relays that not publicly listed [18]. Finally, SGX-Tor can be easily deployed because it uses commodity CPUs and can even be incrementally deployed.

In summary, we make the following contributions:

1. We analyze the assumptions and components used in existing attacks on Tor and discuss how the use of Intel SGX nullifies them to disable the attacks.
2. We present the first design and implementation of Tor that run on real SGX hardware.
3. We demonstrate that SGX-Tor limits the power of Tor adversaries to that of a network-level adversary.
4. We characterize the performance of Tor-SGX through extensive micro- and macro-benchmarks.

Organization: § 2 provides background on Intel SGX and Tor. § 3 describes our approach and the attacks SGX-Tor can defend against. § 4 and § 5 provide the system design and implementation, which we evaluate in § 6. § 7 discusses remaining issues and concerns. § 8 presents related work, and § 9 concludes our work.

2 Background

This section provides key features of Intel SGX and an overview of the Tor anonymity network.

Intel SGX: Intel SGX provides isolated execution by putting and executing the code and data of an application inside a secure container called an *enclave*. It protects sensitive code/data from other applications and privileged system software such as the operating system (OS), hypervisor, and firmware. The memory content of the enclave is stored inside a specialized memory region called Enclave Page Cache (EPC). The EPC region of memory is encrypted within the Memory Encryption Engine (MEE) of the CPU and is hardware access controlled to prevent snooping or tampering with the enclave page content.

Intel SGX instructions consist of privileged instructions and user-level instructions. Privileged instructions are used for loading application code, data, and stack into an enclave. When the enclave is loaded with appropriate memory content, the processor generates the identity of the enclave (i.e., SHA-256 digest of enclave pages) and verifies the integrity of the program by checking the identity that contained a signed certificate (EINIT token) of the program. If the verification succeeds, the CPU enters the enclave mode and the program within the enclave starts to execute from a specified entry point. User-level instructions are used after the program loads.

SGX also provides remote attestation and sealing functions. Remote attestation allows us to verify whether the target program is executed inside an enclave without any modification on a remote SGX platform [24]. Finally, sealing allows us to store enclave data securely in a non-volatile memory by encrypting the content using a `SEAL_KEY`, provisioned by SGX CPU [24]. Unseal restores the content back into the enclave. Intel white papers [39, 40, 53] describe the specifications in detail.

Tor network: The Tor network is a low-latency anonymity network based on onion routing [35]. Tor consists of three components: clients (Tor proxies), directory servers, and relays. Suppose that Alice uses Tor proxy to communicate with Bob through the Tor network. By default, Alice's proxy sets up 3-hop (entry, middle, exit) onion-encrypted circuit to ensure that any single Tor component cannot identify both Alice and Bob (e.g., entry relay knows the source is Alice, but does not know who Alice is talking to). Directory servers are trusted nodes that provide signed directories, called the consensus document. They consist of nine computers run by different trusted individuals and vote hourly on which relays should be part of the network. Relays are provided by volunteers who donate the hosting platform and network bandwidth.

Relays maintain a TLS connection to other relays and clients and transmit data in a fixed-size unit, called a cell.

Each relay maintains a long-term identity key and a short-term onion key. The identity key is used to sign the router descriptor and TLS certificates, and the onion key is used for the onion routing. The directory server also uses an identity key for TLS communication and a signing key for signing the consensus document.

Tor also provides receiver anonymity. It allows Bob to run a hidden service behind a Tor proxy and serve content without revealing his address. To publish a service, Bob's Tor proxy chooses a relay that will serve as an introduction point (IP) and builds a circuit to his IP. It then creates a hidden service descriptor containing its identifier (ID), IP, and public key and publishes the information in the Tor network. The descriptor is stored in a distributed hash table called a hidden service directory. Using the descriptor obtained from the directory, Alice establishes a circuit to its IP and specifies a rendezvous point (RP) for Bob. The IP then relays this information to Bob. Finally, the RP forwards communication between Alice and Bob.

3 Approach Overview

This section describes our assumptions and threat model, presents high-level benefits of applying SGX to Tor, and analyzes how SGX-Tor prevents many attacks.

3.1 Scope

In this paper, we focus on attacks and information leakage that target Tor components. Because Tor is a volunteer-based network, an attacker can easily add malicious relays and/or compromise existing relays. Subversion of directory authorities seriously damages the Tor network, which needs to be protected more carefully. We also consider attacks and information leakage that require colluding relays. Obtaining control over Tor nodes is relatively easier than manipulating the underlying network [66] or having wide network visibility [29, 44, 56]. We follow Tor's standard attack model [35] and do not address attacks that leverage plain text communication between client and server and network-level adversaries (e.g., traffic analysis and correlation attacks [44]).

Threat model: We take a look at how Tor's security model can be improved with SGX. Instead of trusting the application and the system software that hosts Tor relays, SGX-Tor users only trust the underlying SGX hardware. We assume an adversary who may modify or extract information from Tor relays. Following the threat model of SGX, we also assume an adversary can compromise hardware components such as memory and I/O devices except for the CPU package itself [53]. In addition, any software components, including the privileged software (e.g., operating system, hypervisor, and BIOS), can be inspected and controlled by an attacker [53]. DoS attacks are outside the scope of this paper since malicious system software or hardware can simply deny the service (e.g., halt or reboot). Also, side channel attacks, such as cache timing attacks on SGX, are also outside the scope. Both assumptions are consistent with the threat model of Intel SGX [53] and prior work on SGX [26, 61]. Finally, software techniques for defending against attacks that exploit bugs [62, 64] (e.g., buffer overflow) in in-enclave Tor software is out-of-scope.

3.2 SGX-Tor Approach and its Benefits

Main approach: First, our approach is to enclose all private operation and security-sensitive information inside the enclave to make sure that it is not exposed to untrusted parties. We make sure that private or potentially security-sensitive information, such as identity keys and Tor protocol headers, does not leave the enclave by design, relying on the security guarantees of the SGX hardware. This ensures that volunteers do not gain extra information, such as being able to demultiplex circuits, by running a Tor node other than being able to direct encrypted Tor traffic.

Second, we prevent modification of Tor components by relying on remote attestation. When Tor relays are initialized, their integrity is verified by the directory servers. Thus, directory servers ensure that relays are unmodified. Directory servers also perform mutual attestation. We also extend this to attest SGX-enabled Tor proxies (run by client or hidden server) for stronger security properties. Unless otherwise noted, we primarily consider a network in which all Tor relays and directory servers are SGX-enabled. We explicitly consider incremental deployment in §4.2. In the following, we summarize the key benefits of the SGX-Tor design and its security implications.

Improved trust model: Currently, Tor relays are semi-trusted in practice. Some potentially malicious behaviors are monitored by the directory server, and others are prevented by design. However, this does not prevent all malicious behaviors. The fundamental problem is that it is very difficult to explicitly spell out what users must trust in practice. This, in turn, introduces difficulties to the security analysis of Tor. By providing a clear trust model by leveraging the properties of SGX, SGX-Tor allows us to reason about the security properties more easily.

Defense against low resource attacks: To demultiplex circuits, low resource attacks often require node manipulation and internal information that is obtained by running Tor relays. Examples include inflating node bandwidth [27], sending false signals [43], injecting a signal using cell headers [21, 27], and packet spinning attack [58]. SGX-Tor prevents modifications to the code and thus disables these attacks (see §3.3).

Leakage prevention of sensitive information: Directory servers and Tor relays use private keys for sign-

ing, generating certificates, and communicating each other through TLS. Directory servers are under constant threats [32]. If directory authorities are subverted, attackers can manipulate the consensus document to admit or to direct more traffic to malicious relays. Multiple directory authorities have been compromised in practice [32]. This caused all Tor relays to update their software (e.g., directory server information and keys). Relays also contain important information, such as identity keys, circuit identifiers, logs, and hidden service identifiers. By design, SGX-Tor ensures that data structures contained inside the enclave are not accessible to untrusted components, including the system software.

Operational privacy: The consensus document distributed by the director servers lists Tor relays. However, keeping the information public has consequences. It is misused by ISPs and attackers to block Tor [11], to infer whether the relay is being used as a guard or exit [28, 57], and to infer whether Tor is being used [56]. The information also has been used in hidden server location attacks [27, 57]. As a counter-measure, Tor maintains bridge relays. Currently, users can obtain a small number of bridge addresses manually. When all Tor nodes, including Tor proxies, are SGX-enabled, one can keep the list of all relays private by sending the consensus document securely between the directory and user enclaves to enhance the privacy of the Tor network.

3.3 Attacks Thwarted by SGX-Tor

Attacks on Tor typically use a combination of multiple attack vectors. To demonstrate the benefit of SGX-Tor, we analyze existing attacks on Tor and provide a security analysis for SGX-Tor. First, we show attacks that require node modification and how SGX-Tor defeats them.

A bandwidth inflation [25, 27, 57] attack exploits the fact that clients choose Tor relays proportional to the bandwidth advertised in the consensus. This provides malicious relays an incentive to artificially inflate their bandwidths to attract more clients [27]. Bandwidth inflation has been one of the key enablers in low resource attacks [25]. When a relay is first introduced in the network, it reports its bandwidth to the directory servers, allowing the relay to falsely report its bandwidth. To prevent the relays from cheating, directory servers scan for the relay's bandwidth. However, the bandwidth probing incurs pure overhead. It can also be evaded by throttling other streams to make scanners misjudge [27] the bandwidth of relays. Leveraging this, Biryukov et al. [27] inflated the bandwidth report more than 10 times. SGX-Tor simplifies bandwidth reports because of the enhanced trust model. A relay just needs to report the sum of bandwidth that it uses to serve Tor traffic. Because it can be trusted, we do not need an external bandwidth scanner. Note SGX-Tor also defeats replay attacks that might be mounted by an untrusted OS (e.g., duplicate messages) by generating a nonce and keeping it within the enclave during the TLS connection between the relay and directory server.

Controlling hidden service directories [27]: Tor relays that have an HSDir flag set serve as hidden service directories by forming a distributed hash table to which hidden services publish their descriptors. To use a hidden service, clients must fetch the hidden service descriptor that contains the descriptor ID, the list of introduction points, and the hidden service's public key. Biryukov et al. [27] demonstrated an attack in which the attacker can control access to any hidden services. First, malicious relays become hidden service directories for the target hidden services. This amounts to generating a public key that falls into an interval in which the hidden service descriptor ID belongs. After this, malicious hidden service directories can see the requests for the target hidden service descriptors, which they can drop to deny the service. SGX-Tor prevents this because 1) untrusted components in the relay do not see the descriptor; and 2) relay behaviors cannot be altered.

A tagging attack is a type of traffic confirmation attack that changes the characteristics of a circuit at one end (e.g., exit relay) for it to be recognized by the other (e.g, entry guard). This is used to effectively de-anonymize communication between two parties. These attacks require modification of relays. For example, a cell counting attack [27, 50], replay attack [60], and relay early traffic confirmation attack [21] send more command cells, duplicate existing cells, or adjust how many cells are sent over the network at a time to create a distinct pattern. SGX-Tor prevents them because these attacks require relay modification. Note that tagging has been used to de-anonymize hidden services. Biryukov et. al. [27] modified the rendezvous point to send 50 `PADDING` followed by a `DESTROY` cell. Clients use the rendezvous point and the attacker can reliably determine if the hidden service uses its relay as the entry by counting the number of cells. If a pattern is found, the previous node to the entry is marked as the hidden service.

Consensus manipulation in directory server: By taking over directory servers, attackers can manipulate the consensus by accessing the memory content of directory servers. This allows them to admit malicious relays, cast a tie-breaking vote, or steal keys [35]. Especially, the admission of malicious relays is very dangerous; it increases the possibility of various low-resource attacks using malicious Tor relays. During the vetting process, the directory authority creates a "status vote," which contains the information of relays such as its liveness and bandwidth information. The authorities then collect the voting result and generate a consensus document. If more than half of the authorities are manipulated, they can publish the consensus document that contains many malicious

relays [35]. SGX-Tor not only prevents attackers from accessing the content by placing the information inside the enclave, but also detects modified directory servers.

Second, some attacks do not require node modification, but break the privacy by leveraging a relay's internal information. SGX-Tor prevents this by limiting the information available to the attackers.

Collection of hidden service descriptors [27, 51]: This attack collects all hidden service descriptors by deploying a large number of relays that serve as hidden service directories (HSDir). Obtaining service descriptors is easy because one can just dump the relay's memory content. It is shown that with careful placement of HSDirs in the distributed hash table, 1,200 relays is are enough to harvest the entire list [27], which is used to launch other attacks, such as opportunistically de-anonymizing hidden services. The use of SGX-Tor prevents this, as all potentially security-sensitive information, including the hidden service descriptor, is stored only inside the enclave.

Demultiplexing and finger-printing: Tor multiplexes multiple circuits in a single TLS connection and multiple streams (e.g., TCP flows) in a single circuit. Many attacks rely on being able to identify circuits and streams. For example, cell counting attacks [27, 50] and circuit and website finger-printing attacks [47] take advantage of the relay's ability to identify and count the number of cells in a circuit. Traffic and timing analysis used by Overlier et. al. [57] leverages circuit-level information to strengthen the attack. In a vanilla Tor circuit, demultiplexing is trivial because each relay decrypts the TLS connection. In contrast, SGX-Tor hides circuit-level information, including identifiers, from the rest of the world. Note that this means running Tor relay does not give any more information than being a network-level adversary that observes traffic. This makes traffic/finger-printing analysis attacks much more difficult because now an adversary must rely on an additional layer of inference (e.g., timing analysis) for circuit demultiplexing. This forces adversaries to take more time and resources to successfully mount an attack and increase the false positive rates for finger-print attacks, especially in a heavily multiplexed environment [17, 22, 34, 59]. Thus, it enhances the privacy of Tor users.

Bad apple attack [48]: Making circuit identification non-trivial also raises the bar for the bad apple attack. In Tor, multiple TCP streams from a user share the same circuit because it improves efficiency and anonymity [35]. However, this means that even if one TCP stream's source address is revealed, the source address of all TCP streams within a circuit is revealed. The attack takes advantage of this and uses "insecure" applications to de-anonymize secure applications within the same circuit [48]. With SGX-Tor, the attack is not as straightforward because an SGX-enabled exit node that observes many TCP streams cannot easily associate the streams with their circuit. Even when the node is observing all packets, circuit association is difficult on a highly multiplexed exit node (e.g., even if the predecessor is the same for two packets, they may belong to different circuits). A more involved traffic analysis and long running TCP sessions may be required.

Finally, clients (Tor proxies) have also been used to launch attacks. We discuss how SGX-Tor can protect Tor against existing attacks with SGX-enabled Tor proxies.

A sniper attack [43] is a destructive denial-of-service attack that disables Tor relays by making them to use an arbitrarily large amount of memory [43]. A malicious client sends a `SENDME` signal through the circuit without reading any data from it. `SENDME` causes the exit relay to send more data, which exhausts memory at the entry, eventually causing it to be terminated by the OS. This attack requires Tor proxy (client) modification, which can be prevented when the proxy uses SGX. When using SGX-enabled proxies, directory servers or entry guards can verify their integrity. When there is a mix of non-SGX and SGX clients, an effective counter-measure would be to kill circuits [19]. when an entry guard is short of memory, but it can deprioritize circuits to the SGX-enabled proxies when looking for victims because they can be trusted.

Malicious circuit creation: Congestion [37] and traffic analysis attacks [54, 55] use throughput information as a side channel to break the anonymity of Tor (e.g., de-anonymize relays offering a hidden service, identify guards or relays used by a flow). These attacks commonly modify clients to create 1-hop circuits or circuits with a loop to inject measurement traffic to target relays. An SGX-enabled Tor proxy can prevent this by enforcing a minimum hop for circuits (e.g., 3) and disallowing loops when a proxy creates a circuit. Without creating a loop, throughput finger-printing is made much more difficult, less accurate, and more expensive.

Hiding consensus document: As explained in §3.2, SGX-enabled clients and directory servers can keep the list of relays private by enclosing the consensus information inside the enclave to enhance operational privacy.

4 Design

SGX-Tor ensures, by design, the confidentiality of security-sensitive data structures and the integrity of Tor nodes. In addition to the direct benefits of applying SGX, SGX-Tor is designed to achieve the following goals:

Trustworthy interface between the enclave and the privilege software: Although Tor must rely on system software (e.g., system calls) for operation, the interface between the enclave and operating system (OS) must not be trusted. A malicious or curious operating system (or even firmware) can compromise applications running on

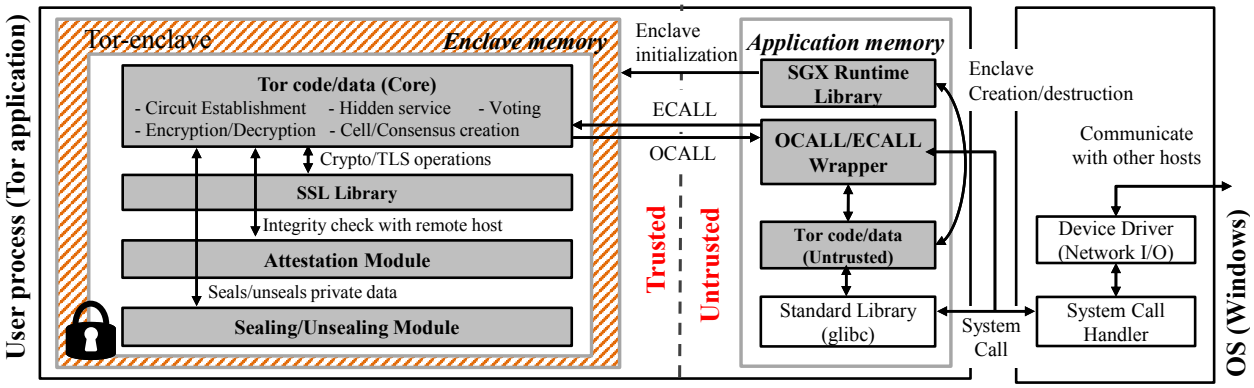


Figure 1: The architecture of SGX-Tor. Gray-colored boxes indicate modified or newly added components from the original Tor. The address space of the Tor application is divided into: *enclave memory* and *application memory*. The enclave communicates only with untrusted software through a well-defined interface.

a secure enclave by carefully manipulating interface between them (e.g., return values in system calls [30]). To reduce such an attack surface, we define a narrow interface between the untrusted and trusted components of SGX-Tor, making the interface favorable to verification or formal proof [26]. For example, SGX-Tor relies on minimal system support for networking, threading, and memory allocation and performs sanity-checking for input/output arguments inside the enclave when requesting services to untrusted system software.

Reducing performance overhead: Utilizing SGX causes inevitable performance degradation for two main reasons: 1) context switches occurring when entering and leaving the enclave mode require TLB flushes and memory operations for saving/restoring registers, and 2) memory accesses (not cache accesses) require additional encryption or decryption by a Memory Encryption Engine (MEE). In addition, the small EPC region (e.g., 128 MB in Intel Skylake [6]) can limit the active working set and thus requires further management of enclave memory that incurs additional performance overhead. Since SGX-Tor protects all security-sensitive data structures and operations within the enclave, large data structures (e.g., list of router descriptors whose size is 10 MB) easily deplete the EPC capacity, in which case the kernel evicts EPC pages through SGX paging instructions (e.g., EWB and ELDB/U) [53], leading to performance degradation. To reduce the overhead, we minimize copying already encrypted data to EPC, such as encrypted packets, and explicitly stage out from EPC large data structures that are used infrequently while ensuring their confidentiality with sealing (see §5).

Deployability and Compatibility: We make practical recommendations to achieve compatibility with existing Tor. Our design facilitates incremental deployment of SGX-enabled Tor components. Note that some features such as remote attestation must have SGX-enabled directory servers, and some properties are only achieved when all components are SGX-enabled. In this paper, we discuss potential issues and benefits of incremental deployment of the SGX-enabled Tor ecosystem.

4.1 SGX-Tor: Architecture

Figure 1 shows the overall architecture of SGX-Tor, shared by all components. The memory region is divided into two parts: the hardware-protected enclave memory region and the unprotected application memory region. Tor-enclave, staged inside an enclave, contains the core components, such as directory authorities, onion routers, and client proxy, which are protected. The untrusted components in the application memory implement an interface to system calls and non-private operations, such as command line and configuration file parsing.

Tor-enclave also contains essential libraries for Tor applications. The SSL library handles TLS communication and the cryptographic operations (e.g., key creation) required for onion routing. The remote attestation module provides APIs to verify the integrity of other Tor programs running on the remote side. The sealing module is used when sensitive information such as private keys and consensus documents must be stored as a file for persistence. SGX-Tor uses the sealing API to encrypt private keys and consensus documents with the seal key provided by the SGX hardware. The enclosed file is only decrypted within the enclave through the unsealing API.

The unprotected application code provides support for Tor-enclave without handling any private information. It handles public data, such as RSA public keys, published certificates, and router finger-prints. Note that key pairs and certificates are generated in Tor-enclave. The SGX runtime library provides an interface to create or destroy an enclave. The untrusted part and the Tor-enclave run as a single process, communicating through a narrow and well-defined interface. A function that enters the enclave is called an `ECALL`, and a function that leaves the enclave is called an `OCALL` as defined in the Intel SGX

SDK [6]. Table 4 (in Appendix A) lists major `E/OCALL` interfaces. We use ECALLs to bootstrap Tor-enclave, while `OCALLs` are used to request services to the system software. The `OCALL` wrapper of SGX-Tor passes the request (e.g., `send()` system call) and arguments from the enclave (e.g., buffer and its length) to the system software and sends the results back to the enclave. Tor-enclave relies on the following system services:

- Network I/O (socket creation, send/recv packets)
- Threading and signal management for event handling
- Error handling
- Memory mapping for file I/O

Note, we rely on the I/O and resource allocation services provided by the system software. In addition to providing the narrow interface, we harden the interface; because the `OCALL` interface and its wrapper are untrusted, we validate the parameters and return a value of `OCALL`. For example, we perform sanity-checking for parameters of the `OCALL` interface to defend against attacks (e.g., buffer overflow) from the untrusted code/data by utilizing the Intel SGX SDK. For every system call used by Tor, we leverage this feature to validate the pointer variables provided by the untrusted OS by putting additional arguments (if needed) that specify the size of the input/output buffer [1].

4.2 SGX-Tor Components and Features

Figure 2 (a) describes the Tor components for providing sender anonymity and (b) illustrates the scenario for a hidden service (i.e., responder anonymity). SGX-Tor applies SGX to every component, including client proxy, directory authorities, onion routers, and hidden services. This section describes how each component is changed in SGX-Tor. We first present the design of four common features shared by all components, followed by the individual Tor components shown in Figure 2.

Initialization (common): All Tor components except client proxy create key pairs, a certificate, and the finger-print at initialization. For this, Tor provides a `tor-gencert` tool that creates RSA keys and certificate for directory authorities and Tor relays. Directory servers create private keys for signing and verifying votes and consensus documents. A Tor relay creates an onion key to encrypt and decrypt the payload for onion routing. Both directory and relay have an identity key pair to sign TLS certificates and consensus document/router descriptor. The original Tor saves the key pairs as a file, which can be leaked once the privilege software is compromised.

[1] For example, `ocall_sgx_select()`, an `OCALL` for `select(int nfds, ..., struct timeval *timeout)` has an additional parameter "**int tv_size**" to specify the buffer size of **"timeout"** (See Appendix). The value is filled in inside the enclave and the sanity-checking routine provided by the SDK inspects the input/output buffer within the enclave.

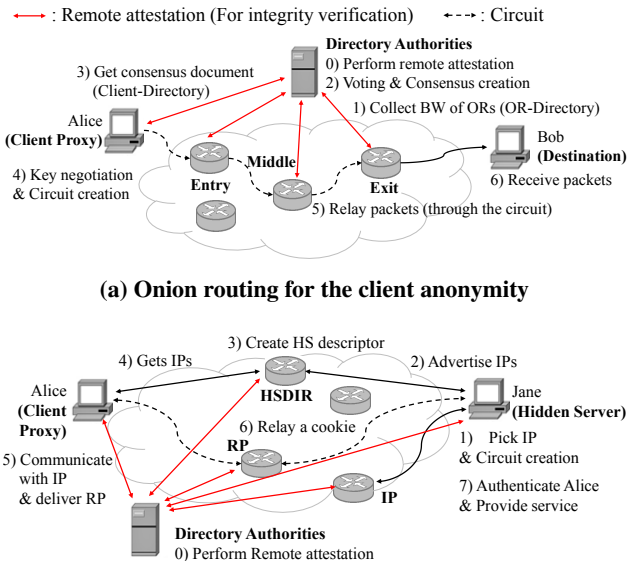


Figure 2: Overview of SGX-Tor in action. The remote attestation verifies the integrity of each other.

SGX-Tor protects the cryptographic operations and seals private keys before storing them in a file.

TLS communication (common): When Tor forwards packets between relays within a circuit, it uses TLS to encrypt the application-level Tor protocol, such as circuit ID, command, cell length, and payload. Currently, this information is visible to relays and system software that hosts the relays. Thus, security-sensitive information, including session key and related operations for establishing TLS connection (e.g., handshaking, encryption, and decryption), must be protected from the untrusted software. In SGX-Tor, the TLS communication is executed within the enclave, leaving the payload of the Tor protocol protected. The system software only handles the network I/O of packets that are encrypted within the enclave.

Sealing of private information (common): When a Tor application terminates, it stores the cached consensus documents and private/public key pairs to a second storage space for later use. To securely perform such an operation, SGX-Tor utilizes the sealing function. When the Tor-enclave must store private information in a file, it encrypts the data using a seal key provided by SGX hardware. The stored data can be loaded and decrypted when the same program requests an unseal within an enclave. Based on the sealing/unsealing interface in the SGX SDK, we develop a high-level API to store the important data of the directory authorities and client proxies. The sealed data is never leaked, unless the CPU package is compromised.

Supporting incremental deployment (common): So far, we explained the system, assuming that all parts

are SGX-enabled. However, we also support interoperability; e.g., it is possible to establish a circuit with an entry-guard that runs SGX-Tor while the middle and exit relays run the original Tor. We add configuration options in the Tor configuration file (`torrc`) to enable remote attestation. The `EnableRemoteAttest` option is set by directory authorities to indicate whether it supports remote attestation. It also has `RelaySGXOnly` and `ClientSGXOnly` options to only admit relays and clients that pass attestation. The relays and clients can set the `RemoteAttestServer` option to request attestation to the directory. For the SGX-Tor client proxy, we add an option to get the list of validated relays from the SGX-enabled directory server. Without these options, SGX-Tor behaves like an ordinary Tor without attestation.

Directory authority: The directory authority manages a list of Tor relays from which the client proxy selects relays. The consensus document, containing the states of Tor relays, is generated by directory servers through voting that occurs every hour. The voting result (i.e., consensus document) is signed by the directory authority to ensure authenticity and integrity. SGX-Tor creates a consensus document and performs voting inside the enclave. For example, data structures for keeping the relay's bandwidth information (`networkstatus_t`), voter list, and voting results are securely contained inside the enclave.

Onion router (relay): Tor relays perform encryption/decryption of the cell content. Relays periodically rotate the private onion keys used for onion routing. SGX-Tor encloses such operations inside the enclave so that security-sensitive information, such as circuit identifiers, cannot be manipulated by an attacker. Because TLS communication is also performed inside the enclave, untrusted components cannot observe Tor commands, unlike in the original Tor. Finally, bandwidth measurement, stored in the `routerinfo_t` data structure, is done securely by calculating the sum of bandwidth inside the enclave so that it cannot be inflated or falsely reported.

Client proxy: The client's circuit establishment and key negotiation process with Tor relays are securely executed inside the enclave. Also, the consensus document and the list of relays are enclosed within the enclave, unlike in the original Tor, where they are transmitted using TCP unencrypted. We modified it to use TLS inside the enclave so that keys and consensus documents are not exposed to untrusted components. We also disallow clients creating a loop in a circuit. For hidden services, SGX-Tor securely manages relevant data structures, such as the hidden service descriptor, address of rendezvous point, and circuit identifier for hidden services to prevent any unintended information leakage.

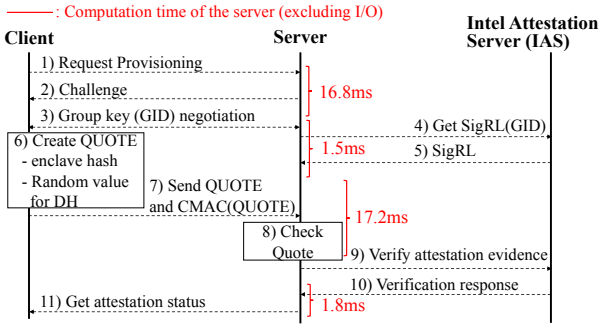


Figure 3: Remote attestation procedure. QUOTE contains the hash of the target enclave in the client.

4.3 Remote Attestation

SGX-Tor uses remote attestation to detect and remove modified nodes from the network. Currently, the same Tor binary serves as the directory authority, Tor relay, and client proxy. Only their configurations are different. This means that every Tor component has the same measurement.

Figure 3 shows the attestation procedure between a client in which an enclave program runs and a remote server that is verifying its integrity. Intel provides Intel Attestation Server (IAS) [41], whose role is similar to a certificate authority, to aid the process; it provisions to the remote server a public key used to authenticate the attestation report. It also issues an endorsement certificate for each SGX processor's attestation key to ensure that the key is stored within the tamper-resistant SGX CPU [31]. During the remote attestation, the remote server checks the `QUOTE` data structure that contains the hash value (code/data pages) to verify the integrity of the client. The remote server then signs the `QUOTE` using the Enhanced Privacy ID (EPID) group key and forwards it to the IAS. The IAS verifies the signature and `QUOTE` and sends the attestation report to the remote server. Here, the EPID group key provides anonymity and also supports unlinkability [41]. Finally, the server verifies the report signature and sends the attestation status to the client.

We use remote attestation for a) integrity verification of relays during onion router (or relay) admission, b) mutual attestation between authorities, and c) sending the list of relays to the trustworthy client. SGX-Tor provides high-level APIs for each remote attestation cases.

Integrity verification of relays (Dir-to-Relay): When a relay contacts directory authorities to register itself in the Tor network, it requests remote attestation (taking the client role in Figure 3) to the directory authorities (server role). The directory server admits only "clean" relays and filters out suspicious ones that run modified Tor programs. Non-SGX relays will also fail to pass the attestation.

Mutual attestation of directories (Dir-to-Dir): The directory authorities mutually perform remote attestation to detect modified servers. A modified directory might try

Component	Lines of code (LoC)	% Changed
Tor application	138,919 lines of C/C++	2.5% (3,508)
EDL	157 lines of code	100% (157)
OCALL wrapper	3,836 lines of C++	100% (3,836)
Attestation/Sealing	568 lines of C/C++	100% (568)
OpenSSL	147,076 lines of C	1.0% (1,509)
libevent	21,318 lines of C	7.0% (1,500)
zlibc	8,122 lines of C	-
Scripts	262 lines of python	-
Total	321,470 lines of code	3.4% (11,078)

Table 1: Lines of code for SGX-Tor software.

to admit specific Tor relays (possibly malicious), launch tie-breaking attacks to interrupt consensus establishment, or refuse to admit benign relays [35]. However, because malicious directory servers will fail to pass the attestation with SGX-Tor, it helps the Tor community to take action quickly (e.g., by launching a new reliable authority). When the codes related to the relay admission policy or algorithm of the directory server are patched, a new measurement can be distributed to check its validity through remote attestation. Note that the existing admission control mechanisms for Tor relays can still be used.

Trustworthy client (Client-to-Dir): To detect modified client proxies, the directory authority attests clients and transmits only the consensus document when they pass attestation. This filters out modified clients that might perform an abnormal circuit establishment such as creating a loop [37, 54, 55] and also keeps the consensus document confidential. Therefore, only benign SGX-enabled Tor clients obtain the consensus document, which contains the list of relays verified by directory authorities. In summary, if all relays and clients are SGX-enabled, we can 1) keep the list of relays private and 2) block malicious clients.

5 Implementation

We developed SGX-Tor for Windows by using the SGX SDK provided by Intel [6]. In total, it consists of 321K lines of code and approximately 11K lines of code are modified for SGX-Tor, as broken down in detail in Table 1. As part of this effort, we ported OpenSSL-1.1.0 [8], zlib-1.2.8 [16], and libevent-2.0.22 [7] inside the enclave. Note that the porting effort is non-trivial; for one example, OpenSSL libraries store the generated keys to files, but to securely export them to non-enclave code, we have to carefully modify them to perform sealing operations instead of using naive file I/Os. Furthermore, because enclave programs cannot directly issue system calls, we implemented shims for necessary system calls with an `OCALL` interface. However, to minimize the TCB size, we ported only required glibc functions, such as `sscanf()` and `htons()`, instead of embedding the entire library. As a result, our TCB becomes dramatically reduced compared to other SGX systems such as Graphene [67] or Haven [26] (more than 200 MB) that implements a whole library OS to support SGX applications; SGX-Tor results in 3.8 times smaller TCB compared to Graphene (320 K vs. 1,228 K LoC). The source code is available at `https://github.com/kaist-ina/SGX-Tor`.

Managing enclave memory: To work with the limited EPC memory, SGX-Tor seals and stores large data structures *outside* of the enclave. If required, it explicitly loads and unseals the encrypted data into the enclave. For example, `cached-descriptors` that contain the information of reachable relays (e.g., finger-print, certificate, and measured bandwidth), are around 10 MB for each, which is too big to always keep inside the enclave. Unlike the original Tor, which uses memory-mapped I/Os to access these data, SGX-Tor loads and unseals them into the EPC only when it has to update the list of relays, which essentially trades extra computation for more usable EPC memory. Similarly, certain system calls such as `recv()` are implemented to save the enclave memory; they get a pointer pointing to the data (e.g., encrypted packets) outside the enclave instead of copying them to the enclave memory.

Sealing and unsealing API: We implemented sealing and unsealing API to substitute file I/O operations for private keys. SGX-Tor uses C++ STL map to store generated private keys in the enclave memory. The key of the map is the name of the private key, and the value of the map is a structure that contains the contents and length of a private key. When SGX-Tor needs to read a generated key, it finds the key contents by the key name through the map. The application side of SGX-Tor can request the private keys using sealing API to store it in the file system. SGX-Tor uses sealing before sending the keys outside the enclave. In reverse, the application side of SGX-Tor also can request to load the sealed private keys using unsealing API. SGX-Tor decrypts sealed key by unsealing it and stores it in the map. These sealing and unsealing mechanisms are easily usable because they are implemented as macros.

Securely obtaining entropy and time: The vanilla OpenSSL obtains entropy from the untrusted underlying system through system calls, like `getpid()` and `time()`, that make the enclave code vulnerable to Iago attacks [30, 42]; for example, a manipulated time clock can compromise the logic for certification checking (e.g., expiration or revocation). To prevent such attacks, we obtain entropy directly from the trustworthy sources: randomness from the `rdrand` instruction (via `sgx_read_rand`) and time clocks from the trusted platform service enclave (PSE) (via `sgx_get_trusted_time`) [6][pp. 88-92, 171-172].

Data structure	Tor	Network-level adversary	SGX-Tor (Component)
TCP/IP header	V	V	V
TLS encrypted bytestream	V	V	V
Cell	V	**N**	**N** (R)
Circuit ID	V	**N**	**N** (R)
Voting result	V	**N**	**N** (D)
Consensus document	V	**N**	**N** (D/R/C)
Hidden service descriptor	V	**N**	**N** (H)
List of relays	V	**N**	**N** (C)
Private keys	V	**N**	**N** (D/R/C)

Table 2: Information visible to adversaries who run SGX-Tor and original Tor and network-level adversaries. "V" denotes visible; "N" denotes non-visible. Component "D" denotes a directory authority, "R" relay, "C" client, and "H" hidden service directory.

6 Evaluation

We evaluate SGX-Tor by answering three questions:

- What types of Tor attacks can be mitigated?
- What is the performance overhead of running SGX-Tor? How much does each component of SGX-Tor contribute to the performance degradation?
- How compatible is SGX-Tor with the current Tor network? How easy is SGX-Tor adopted?

Experimental setting: We set up two evaluation environments for SGX-Tor: 1) by admitting SGX-Tor onion router in the real Tor network and 2) by constructing a private Tor network where all components, including directories and client proxies, run SGX-Tor. We used nine SGX machines (Intel Core i7-6700 3.4GHz and Intel Xeon CPU E3-1240 3.5GHz). The private Tor network consists of a client proxy, five relays, and three directory servers. Note that directory servers also work as relays. We extend the work of Chutney [23] to configure and orchestrate our private SGX-Tor network.

6.1 Security Analysis

Table 2 summarizes security- and privacy-sensitive data structures that are available to three types of adversaries: 1) an adversary who controls relays running original Tor, 2) an adversary who controls the platform running SGX-Tor, 3) and a network-level adversary. V marks the visible information to an adversary, whereas N denotes non-visible ones. An adversary who controls the vanilla Tor can access a great deal of sensitive information, attracting more adversaries to run malicious Tor relays. In contrast, an attacker who even controls the platform running SGX-Tor cannot gain any information other than observing the traffic, just like a network-level adversary. This indicates that the power of Tor adversaries is reduced to that of network-level adversaries with SGX-Tor. Among the attacks in §3.3, we choose three well-known classes of attacks considering their reproducibility and severity. We replicate these attacks (and their key attack vectors) in a lab environment and evaluate if SGX-Tor correctly mitigates them.

Bandwidth inflation: To demonstrate this attack, we modify the Tor code to advertise inflated bandwidth of a relay to directory servers. The directory server performs bandwidth scanning to check whether a relay actually serves the bandwidth advertised by itself [10]. During the scanning, the directory server creates a 2-hop circuit, including the target relay, and downloads the file from particular hosts to estimate the bandwidth of the relays. If a malicious relay is selected as non-exit, it can directly see which connection is originated from the directory server [27]. By throttling other traffic, the compromised relay inflates the measured bandwidth and gets a fast flag, which is given to a high bandwidth relay indicating that they are suitable for high-bandwidth circuits, in the consensus document [25, 27, 57]. However, with SGX-Tor, modifying the Tor code is not fundamentally possible due to the measurement mismatch during the attestation.

Circuit demultiplexing: Being able to decrypt cell Tor headers and demultiplex circuits and streams in relays is a common attack vector exploited in cell counting attacks [27, 50], traffic analysis [57], website finger-printing attacks [47], bad apple attacks [48], replay attacks [60], relay early attacks [21], and controlling access to hidden services [27]. With a modified relay, we were able to dump Tor commands, circuit IDs, and stream IDs; count cells per stream [27, 50]; duplicate cells [60]; and even selectively drop particular circuits and streams [28]. However, with SGX-Tor, the modified relay failed to be admitted due to attestation failure. With the attested SGX-Tor relay, it is not possible to dump the EPC memory outside the enclave unless the code inside the enclave is compromised due to an exploitable bug (e.g., buffer overflow). Even inferring the cell boundary was not trivial, let alone observing decrypted cell headers.

Malicious circuit creation: By modifying the original Tor code, we successfully establish a 3-hop loop circuit. Creating a loop can be an attack vector for traffic analysis [54, 55] and congestion attack [37] with a long loop. In SGX-Tor, it is not possible to introduce loops because the directory authority can verify the integrity of the client proxies. Therefore, the modified Tor client fails to manipulate a circuit as it intended.

6.2 Performance Evaluation

End-to-end performance: To quantify the effect on performance in a wide-area network, we configure a private Tor network that consists of an entry guard and exit relay located in East Asia and the U.S. East, respectively. For SGX-Tor, every Tor component, including client proxy, except the destination server runs SGX, except for mid-

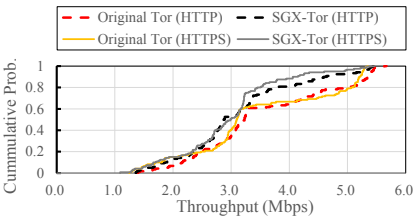


Figure 4: CDF of client throughput.

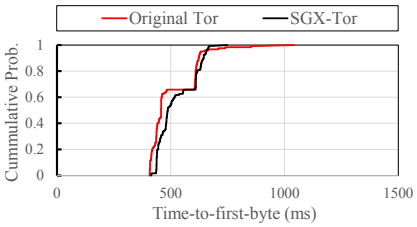


Figure 5: CDF of time-to-first-byte.

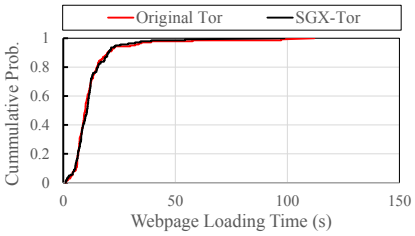


Figure 6: CDF of page loading time.

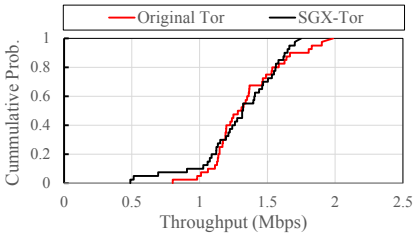


Figure 7: Hidden service throughput of SGX-Tor and original Tor.

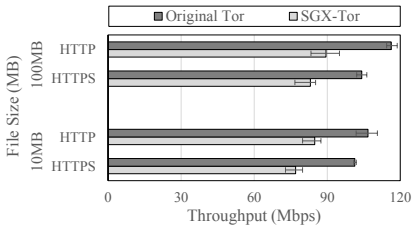


Figure 8: End-to-end client throughput in the private Tor network.

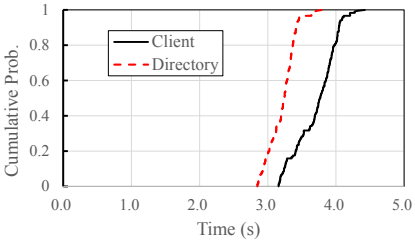
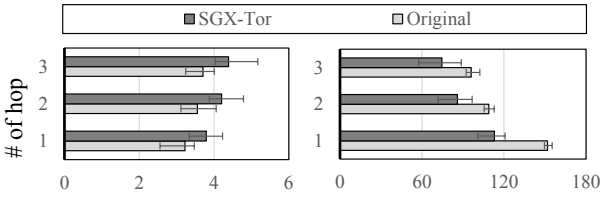


Figure 9: Remote attestation latency of the directory and the client.

Type	Time-to-fisrt-byte (ms)	Throughput (Mbps) 10 MB	50 MB
Original (baseline)	2.05	658	716
SGX-Tor (overhead)	2.19 (6.8%)	589 (8.8%)	651 (11%)

Table 3: Overhead of TLS communication.



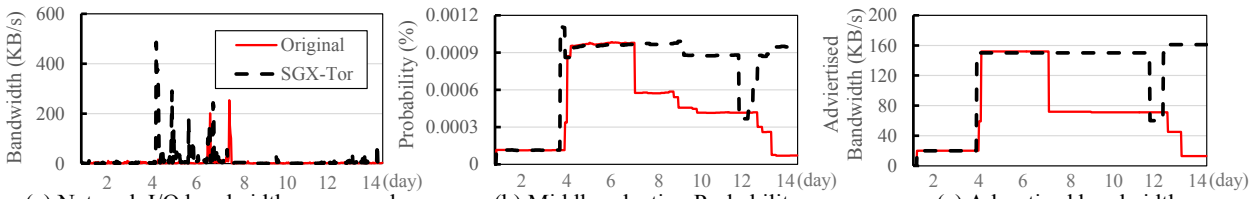
(a) Time-to-first-byte (ms) (b) Throughput (Mbps)

Figure 10: Overhead of onion encryption.

dle relays. To diversify the middle relay locations, we use Amazon EC2 [2] U.S. East, U.S. West, and Europe instances. Figure 4 shows the CDF of throughput while downloading a file (10 MB) via a HTTP/HTTPS server. The results are based on the average of 50 runs. SGX-Tor exhibits 11.9% lower throughput (3.11 Mbps) for HTTP and 14.1% (2.95 Mbps) lower throughput for HTTPS. Figure 5 shows the CDF of time-to-first-byte (latency) for HTTP transfer. SGX-Tor (525ms) only gives 3.9% additional delays compared to the original Tor (505ms). We also evaluate the web latency when a client connects to a website through Tor. Figure 6 shows the distribution of the web page loading time for Alexa Top 50 websites [1]. We measure the time from the initiation of a request until an onload event is delivered in the Firefox browser. Similar to time-to-first-byte, SGX-Tor gives 7.4% additional latency on average. We note that our SGX SDK allows compilation only in debug mode, since it requires an approved developer key provided by Intel to run an enclave in release mode. Thus, we used debug mode for all Tor performance measurements.

Hidden service: To quantify the overhead of running a hidden service with an SGX-Tor proxy, we run one on the real Tor network. At the client side, we use a Tor browser [12] that automatically picks a rendezvous point. For each measurement, we relaunch a Tor browser to establish a new circuit. We perform 100 measurements that transfer a 10 MB file from an HTTP file server running as a hidden service. Figure 7 shows the distribution of the throughput for SGX-Tor and the original Tor. The hidden service using SGX-Tor gives only a 3.3% performance degradation from 1.35 to 1.30 Mbps on average. We see a smaller gap because the performance is more network bottlenecked as packets from/to a hidden service traverse two 3-hop circuits and only the hidden service uses SGX.

Overhead of TLS and onion encryption: To quantify the overhead in a more CPU-bound setting, we create a private Tor network in which all components are connected locally at 1Gbps through a single switch. We measure the overhead of SGX-Tor starting from a single TLS connection without any onion routing and increase the number of onion hops from one to three. Table 3 shows the time-to-first-byte and throughput of TLS communication without onion routing. The result shows that SGX-Tor has 9.99% (716 to 651 Mbps) of performance degradation and has 6.39% additional latency (2.05 to 2.19 ms). Figure 10 shows the time-to-first-byte and throughput by increasing the number of onion hops for downloading a 10 MB file from the HTTP server. As the hop is increased, SGX-Tor has 17.7%, 18.3%, and 18.4% additional latency and the performance is degraded by 25.6%, 21.1%, 22.7%, respectively. Figure 8 shows the end-to-end client performance in the private Tor net-



(a) Network I/O bandwidth per second (b) Middle selection Probability (c) Advertised bandwidth

Figure 11: Compatibility test of SGX-Tor and original Tor while running as a middle relay. Both Tor relays started at the same time and acquire "Fast" and "Stable" flags during the evaluation [3].

work. The vanilla Tor achieves 106 Mbps for HTTP and 101 Mbps for HTTPS transfers, while SGX-Tor gives 85 Mbps and 77 Mbps respectively, resulting in a throughput degradation of 24.7% and 31.2%, respectively (for 10 MB).

Remote attestation: We quantify the latency and computation overhead of remote attestation. To emulate a Tor network running in a wide area setting, we introduce latency between the SGX-Tor directory and relays in our private Tor network to mimic that of the real network. The round-trip time between East Asia (where most of our SGX servers are) and nine directory authorities in Tor falls between 144ms (longclaw [3] in the U.S.) and 313ms (maatuska [3] in Sweden). The round-trip time between our directory and IAS was 310ms. We execute 30 runs for each directory while introducing the latency of a real Tor network. Figure 9 shows the CDF of the remote attestation latency when the directory authority verifies Tor relays. The average attestation latency of the entire process in Figure 3 is 3.71s.

SGX-Tor relays and clients request the remote attestation to the directory server once during bootstrapping. This means that this is a one-time bootstrapping cost and is relatively short compared to the client bootstrapping time, which takes 22.9s on average on a broadband link to download a list of relays [49]. We quantify the computational cost of remote attestation for the directory authority for verifying client proxies. For attestation, directory servers calculate the AES-CMAC of group ID for checking EPID group and ECDSA signature generation for QUOTE verification. On our i7 machine, the computation takes 37.3ms in total if the QUOTE verification succeeds, as annotated in Figure 3. When it fails, it takes 35.5ms. On a peak day, Tor has about 1.85 million users [13]. To quantify the amount of computation required, we perform AES-CMAC and ECDSA signature generation in a tight loop and measure the throughput. It gives 27.8 operations per second per core. Thus, with nine directory servers, assuming each has eight cores, the total computation time for remote attestation of 1.85 million daily clients will be about 15.4 minutes. We believe this is a moderate resource requirement for the directory authority.

Overhead of key generation: Finally, we measure the overhead of in-enclave key generation for two RSA key pairs: identity key (3072-bit) and signing key (2048-bit) for the directory authority. SGX-Tor takes 12% longer than the vanilla Tor (2.90 vs. 2.59 ms), including the time for sealing keys and unsealing for key recovery.

6.3 Compatibility and Deployability

We demonstrate the compatibility of SGX-Tor by admitting a long-running SGX-Tor relay into the existing Tor network as a middle relay. For comparison, we also run a vanilla Tor side-by-side. We compare both relays in terms of (a) network I/O bandwidth per second, (b) probability to be selected as a middle relay, and (c) advertised bandwidth obtained from a published consensus document from CollectTor [4]. The bandwidth statistics are averaged over a 30-minute window. Figure 11 shows the result obtained for two weeks. In total, SGX-Tor served 10.5 GB of traffic. Both relays obtained fast and stable flag on the same day. The average advertised bandwidth of SGX-Tor relay is 119 KB/s. We see that SGX-Tor is compatible with the existing Tor network, and for all metrics SGX-Tor shows a similar tendency with the vanilla Tor.

7 Discussion

Deployment issues: The simplest way to deploy SGX-Tor in the existing Tor ecosystem incrementally is to use cloud platforms. Tor already provides default VM images to run bridges on Amazon EC2 cloud [13]. When SGX becomes available on cloud platforms, we envision a similar deployment scenario for SGX-Tor. As a recent patch provides support for SGX virtualization [9], we believe that deployment of SGX-Tor using cloud platform is feasible. Note that incremental deployment involves in security tradeoffs, as not all properties can be achieved as discussed in §4. As a future work, we would like to quantify the security tradeoffs and ways to mitigate attacks in the presence of partial deployment.

Limitation: Although SGX-Tor can mitigate many attacks against Tor components, attacks assuming network-level adversaries [44] and Sybil attacks [36] are still effective, as we mentioned in the §3.1. Additionally, SGX-Tor cannot validate the correctness of the enclave code itself. SGX-Tor can be compromised if the code contains software vulnerabilities and is subject to controlled side-channel attacks [69]. We believe these attacks can be

mitigated by combining work from recent studies: e.g., by checking whether an enclave code leaks secrets [65]; by protecting against side-channel attacks [62, 64]; or by leveraging software fault isolation [38, 62]).

8 Related Work

Software for trusted execution environments: Various TEEs such as TPM, ARM TrustZone, and AMD SVM have been used for guaranteeing the security of applications in mobile and PC environments. Since the traditional trusted computing technologies (e.g., hypervisor-based approach with TPM) rely on the chain of trust, it makes the size of TCB larger. Flicker [52] proposed an approach that executes only a small piece of code inside the trusted container, where it extremely reduces the TCB. Nevertheless, it suffers from performance limitations. Intel SGX removes this challenge by offering native performance and multi-threading. In addition, the cloud computing and hosting service providers, where Tor relays are often hosted [14], is predominantly x86-based.

Applications for Intel SGX: Haven [26] pioneered adopting Intel SGX in the cloud environment with an unmodified application. VC3 [61] proposed data analytics combined with Intel SGX in the cloud. Moat [65] studied the verification of application source code to determine whether the program actually does not leak private data on top of Intel SGX. Kim et al. [46] explores how to leverage SGX to enhance the security and privacy of network applications. These studies are early studies of SGX that rely on SGX emulators [42]. S-NFV [63] applies SGX to NFV to isolate its state from the NFV infrastructure and platform and presents preliminary performance evaluations on real SGX hardware. In contrast, we show how SGX can improve the trust model and operation of Tor and SGX-Tor run on real SGX hardware.

Attacks and security analysis on Tor: §3 discussed many attacks on Tor. SGX-Tor prevents modification of Tor binaries and limits attackers' ability; attackers can still launch attacks within the network outside Tor nodes. For example, attackers can still mount traffic analysis attacks or website finger-printing attacks. While Tor does not try to protect against traffic confirmation attacks [35], it aims to protect against general traffic analysis attacks. In particular, large-scale traffic correlation [29] and website finger-printing [45] attacks are believed to be very difficult in practice [22, 45] because those attacks require achieving an arbitrarily low false positive rate as the number of users becomes larger. Security analysis of Tor on a realistic workload is an ongoing research [34] area. In this work, we show how we can thwart known attacks against the Tor ecosystem by using a commodity trusted execution environment, Intel SGX.

9 Conclusion

Due to the wide adoption of the x86 architecture, Intel Software Guard Extensions (SGX) potentially has a tremendous impact on providing security and privacy for network applications. This paper explores new opportunities to enhance the security and privacy of a Tor anonymity network. Applying SGX to Tor has several benefits. First, we show that deploying SGX on Tor can defend against known attacks that manipulate Tor components. Second, it limits the information obtained by running or compromising Tor components, reducing the power of adversaries to network-level adversaries, who can only launch attacks external to the Tor components. Finally, this brings changes to the trust model of Tor, which potentially simplifies Tor operation and deployment. Our extensive evaluation of the SGX-Tor shows that SGX-enabled Tor components incur small performance degradation and supports incremental deployment on the existing Tor network, demonstrating its viability.

10 Acknowledgment

We thank the anonymous reviewers and our shepherd Phillipa Gill for their feedback. We also thank Will Scott, Chris Magistrado, Hyeontaek Lim, and Inho Choi for their comments on earlier versions. Dongsu Han is the corresponding author. This work was supported in part by the IITP funded by the Korea government (MSIP) [B0101-16-1368, Development of an NFV-inspired networked switch and an operating system for multi-middlebox services] and [B0190-16-2011, Korea-US Collaborative Research on SDN/NFV Security/Network Management and Testbed Build]; Office of Naval Research Global (ONRG); and NSF awards DGE-1500084, CNS-1563848, and CRI-1629851.

References

[1] Alexa: The top 500 sites on the web. `http://www.alexa.com/topsites`.

[2] Amazon Web Service. `https://aws.amazon.com/`.

[3] Atlas, flag:authority. `https://atlas.torproject.org/#search/flag:authority`.

[4] CollecTor. `https://collector.torproject.org/`.

[5] FBI's use of Tor exploit is like peering through "broken blinds". `http://arstechnica.com/tech-policy/2016/06/fbis-use-of-tor-exploit-is-like-peering-through-broken-blinds/`. Last accessed: Aug 2016.

[6] Intel Software Guard Extensions (Intel SGX) SDK. https://software.intel.com/en-us/sgx-sdk. Last accessed: Aug 2016.

[7] libevent-2.0.22. http://libevent.org/.

[8] OpenSSL-1.1.0. https://www.openssl.org/.

[9] Sgx virtualization. https://01.org/intel-software-guard-extensions/sgx-virtualization.

[10] Tor bandwidth scanner. https://trac.torproject.org/projects/tor/attachment/ticket/2861/bwauth-spec.txt.

[11] Tor: Bridges. https://www.torproject.org/docs/bridges.html.en. Last accessed: May, 2016.

[12] Tor Browser. https://www.torproject.org/projects/torbrowser.html.en.

[13] Tor Metrics. https://metrics.torproject.org/.

[14] Tor Project: GodBadISPs. https://trac.torproject.org/projects/tor/wiki/doc/GoodBadISPs.

[15] Tor:Overview. https://www.torproject.org/about/overview. Last accessed: Aug 2016.

[16] zlib-1.2.8. http://www.zlib.net/.

[17] Research problem: measuring the safety of the tor network. https://blog.torproject.org/blog/research-problem-measuring-safety-tor-network, 2011.

[18] Research problems: Ten ways to discover Tor bridges. https://blog.torproject.org/blog/research-problems-ten-ways-discover-tor-bridges, Oct 2011.

[19] Extend OOM handler to cover channels/connection buffers. https://trac.torproject.org/projects/tor/ticket/10169, 2014.

[20] How to report bad relays. https://blog.torproject.org/blog/how-report-bad-relays, July 2014.

[21] Tor security advisory: "relay early" traffic confirmation attack. https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack, July 2014.

[22] Traffic correlation using netflows. https://blog.torproject.org/category/tags/traffic-confirmation, Nov 2014.

[23] The chutney tool for testing and automating Tor network setup. https://gitweb.torproject.org/chutney.git, 2015. Accessed: 09/01/2016.

[24] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Proc. HASP*, pages 1–8, Tel-Aviv, Israel, 2013.

[25] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource Routing Attacks Against Tor. In *Proc. ACM Workshop on Privacy in Electronic Society*, 2007.

[26] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *Proc. USENIX OSDI*, 2014.

[27] A. Biryukov, I. Pustogarov, and R.-P. Weinmann. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *Proc. IEEE Symposium on Security and Privacy*, pages 80–94, 2013.

[28] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of Service or Denial of Security? In *Proc. ACM CCS*, 2007.

[29] S. Chakravarty, M. V. Barbera, G. Portokalidis, M. Polychronakis, and A. D. Keromytis. On the effectiveness of traffic analysis against anonymity networks using flow records. In *Proc. Passive and Active Measurement*, 2014.

[30] S. Checkoway and H. Shacham. Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface. In *Proc. ASPLOS*, 2013.

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

[32] R. Dingledine. Tor Project infrastructure updates in response to security breach. http://archives.seul.org/or/talk/Jan-2010/msg00161.html, January 2010.

[33] R. Dingledine. Turning funding into more exit relays. https://blog.torproject.org/blog/turning-funding-more-exit-relays, July 2012.

[34] R. Dingledine. Improving Tor's anonymity by changing guard parameters. https://blog.torproject.org/blog/improving-tors-anonymity-changing-guard-parameters, Oct 2013.

[35] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-generation Onion Router. In *Proc. USENIX Security Symposium*, 2004.

[36] J. R. Douceur. The Sybil Attack. In *Proc. IPTPS*, 2002.

[37] N. S. Evans, R. Dingledine, and C. Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *Proc. USENIX Security Symposium*, Berkeley, CA, USA, 2009. USENIX Association.

[38] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan:A Distributed Sandbox for Untrusted Computation on Secret Data. In *Proc. USENIX OSDI*, 2016.

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

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

[41] Intel. Intel Software Guard Extensions Remote Attestation End-to-End Example. `https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-example?language=de`, July 2016.

[42] P. Jain, S. Desai, S. Kim, M.-W. Shih, J. Lee, C. Choi, Y. Shin, T. Kim, B. B. Kang, and D. Han. OpenSGX: An Open Platform for SGX Research. In *Proc. NDSS*, San Diego, CA, Feb. 2016.

[43] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. In *Proc. NDSS*, 2015.

[44] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *Proc. ACM CCS*, pages 337–348. ACM, 2013.

[45] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A Critical Evaluation of Website Fingerprinting Attacks. In *Proc. CCS*, 2014.

[46] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han. A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications. In *Proc. ACM Workshop on Hot Topics in Networks*, page 7. ACM, 2015.

[47] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit fingerprinting attacks: passive deanonymization of Tor hidden services. In *Proc. USENIX Security Symposium*, pages 287–302, 2015.

[48] S. Le Blond, P. Manils, A. Chaabane, M. A. Kaafar, C. e. Castelluccia, A. Legout, and W. Dabbous. One Bad Apple Spoils the Bunch: Exploiting P2P Applications to Trace and Profile Tor Users. In *Proc. USENIX Conference on Large-scale Exploits and Emergent Threats*, pages 2–2, 2011.

[49] J. Lenhard, K. Loesing, and G. Wirtz. Performance Measurements of Tor Hidden Services in Low-Bandwidth Access Networks. In *International Conference on Applied Cryptography and Network Security*, pages 324–341, 2009.

[50] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia. A New Cell-Counting-Based Attack Against Tor. *IEEE/ACM Transactions on Networking (TON)*, 20(4):1245–1261, 2012.

[51] K. Loesing. *Privacy-enhancing technologies for private services.* PhD thesis, University of Bamberg, 2009.

[52] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proc. EUROSYS*, pages 315–328, 2008.

[53] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *Proc. HASP*, pages 1–8, Tel-Aviv, Israel, 2013.

[54] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. Stealthy Traffic Analysis of Low-latency Anonymous Communication Using Throughput Fingerprinting. In *Proc. ACM CCS*, pages 215–226, New York, NY, USA, 2011. ACM.

[55] S. J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *Proc. IEEE Symposium on Security and Privacy*, pages 183–195, Washington, DC, USA, 2005. IEEE Computer Society.

[56] S. J. Murdoch and P. Zieliński. Sampled Traffic Analysis by Internet-exchange-level Adversaries. In *Proc. Privacy Enhancing Technologies*, Berlin, Heidelberg, 2007. Springer-Verlag.

[57] L. Overlier and P. Syverson. Locating Hidden Servers. In *Proc. IEEE Symposium on Security and Privacy*, pages 100–114, 2006.

[58] V. Pappas, E. Athanasopoulos, S. Ioannidis, and E. P. Markatos. Compromising Anonymity Using Packet Spinning. In *Proc. Information Security Conference*, pages 161–174, 2008.

[59] M. Perry. A Critique of Website Traffic Fingerprinting Attacks. `https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks`, Oct 2013.

[60] R. Pries, W. Yu, X. Fu, and W. Zhao. A New Replay Attack Against Anonymous Communication Networks. In *Proc. IEEE International Conference on Communications*, pages 1578–1582, May 2008.

[61] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy data analytics in the cloud using SGX. In *Proc. IEEE Security and Privacy (SP)*, pages 38–54. IEEE, 2015.

[62] J. Seo, B. Lee, S. Kim, M.-W. Shih, I. Shin, D. Han, and T. Kim. SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs. In *Proc. NDSS*, 2017.

[63] M.-W. Shih, M. Kumar, T. Kim, and A. Gavrilovska. S-NFV: Securing NFV States by Using SGX. In *Proc. ACM International Workshop on Security in SDN-NFV*, 2016.

[64] M.-W. Shih, S. Lee, T. Kim, and M. Peinado. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs. In *Proc. NDSS*, 2017.

[65] R. Sinha, S. Rajamani, S. Seshia, and K. Vaswani. Moat: Verifying confidentiality of enclave programs. In *Proc. ACM CCS*, pages 1169–1184, 2015.

[66] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. RAPTOR: Routing Attacks on Privacy in Tor. In *Proc. USENIX Security Symposium*, 2015.

[67] C.-C. Tsai, K. S. Arora, N. Bandi, B. Jain, W. Jannen, J. John, H. A. Kalodner, V. Kulkarni, D. Oliveira, and D. E. Porter. Cooperation and Security Isolation of Library OSes for Multi-Process Applications. In *Proc. European Conference on Computer Systems*, page 9. ACM, 2014.

[68] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl. Spoiled onions: Exposing malicious tor exit relays. In *Proc. Privacy Enhancing Technologies*, pages 304–331. Springer, 2014.

[69] Y. Xu, W. Cui, and M. Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *Proc. IEEE Symposium on Security and Privacy*, 2015.

Appendix A

Type	Category	API	Description
ECALL	TOR	`void sgx_start_tor(int argc, char **argv, ...)`	Start SGX-Tor process in enclave
ECALL	TOR	`void sgx_start_gencert(char *tor_cert, ...)`	Create authority keys and certificate
ECALL	TOR	`void sgx_start_fingerprint(char *fingerprint, ...)`	Create finger-print
ECALL	TOR	`void sgx_start_remote_attestation_server(int port, ...)`	Start remote attestation server
ECALL	TOR	`sgx_status_t sgx_init_ra(int b_pse, ...)`	Create remote attestation context
ECALL	TOR	`sgx_status_t sgx_close_ra(sgx_ra_context_t context)`	Release remote attestation context
ECALL	TOR	`sgx_status_t sgx_verify_att_result_mac(sgx_ra_context_t context, ...)`	Verify the MAC in attestation result
OCALL	NET	`int ocall_sgx_socket(int af, int type, int protocol)`	Create socket descriptor
OCALL	NET	`int ocall_sgx_bind(int s, const struct sockaddr *addr, ...)`	Bind socket
OCALL	NET	`int ocall_sgx_listen(int s, int backlog)`	Make socket in listening state
OCALL	NET	`int ocall_sgx_accept(int s, struct sockaddr *addr, ...)`	Accept incoming connection
OCALL	NET	`int ocall_sgx_send(int s, char *buf, int len, int flags)`	Receive data from given socket
OCALL	NET	`int ocall_sgx_recv(int s, char *buf, int len, int flags)`	Send data through given socket
OCALL	NET	`int ocall_sgx_select(int nfds, void *rfd, ..., struct timeval *timeout, int tv_size)`	Examining the status of socket descriptor
OCALL	THREAD	`unsigned long long ocall_sgx_beginthread(void *args, ...)`	Create thread with given argument
OCALL	THREAD	`unsigned long ocall_sgx_TlsAlloc(void)`	Allocates a thread local storage index
OCALL	ERROR	`int ocall_sgx_GetLastError(void)`	Get the error code of calling thread
OCALL	ERROR	`void ocall_sgx_SetLastError(int e)`	Set the error code of calling thread
OCALL	MEM	`int ocall_sgx_CreateFileMapping(int hFile, ...)`	Create file mapping object
OCALL	MEM	`void * ocall_sgx_MapViewOfFile(int hFileMappingObject, ...)`	Mapping address space for a file

Table 4: Interface between the Tor enclave and the privilege software. SGX-Tor uses seven ECALLS to bootstrap the Tor-enclave and aid remote attestation. It uses a total of 57 OCALLs in four categories. We list representative OCALLs in each category.