A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications

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

Abstract

Network applications and protocols are increasingly adopting security and privacy features, as they are becoming one of the primary requirements. The wide-spread use of transport layer security (TLS) and the growing popularity of anonymity networks, such as Tor, exemplify this trend. Motivated by the recent movement towards commoditization of trusted execution environments (TEEs), this paper explores alternative design choices that application and protocol designers should consider. In particular, we explore the possibility of using Intel SGX to provide security and privacy in a wide range of network applications. We show that leveraging hardware protection of TEEs opens up new possibilities, often at the benefit of a much simplified application/protocol design. We demonstrate its practical implications by exploring the design space for SGX-enabled software-defined inter-domain routing, peer-to-peer anonymity networks (Tor), and middleboxes. Finally, we quantify the potential overheads of the SGX-enabled design by implementing it on top of OpenSGX, an open source SGX emulator.

1. INTRODUCTION

Security and privacy has become one of the primary concerns both for users and application/protocol developers. Today, lack of security and privacy guarantees often acts as a deterrent in adopting new technologies [17, 39], and applications with stronger security and privacy guarantees remain more competitive. As a result, software technologies for security and privacy have seen wider adoption within the network over the years. For example, a large fraction of Internet traffic uses Transport layer security (TLS) [27], and anonymity networks, such as Tor, is becoming increasingly popular [21].

At the same time, significant progress in hardware-based protection and security have also made into the market. Modern subscriber identity modules in mobile handsets securely store authentication/encryption keys and perform cryptographic operations in hardware. More powerful hardware-based security solutions, such as trusted execution environments (TEEs) (e.g. ARM TrustZone), are also becoming popular.

Recently, Intel announced Software Guard Extensions (SGX), as an extension to the x86 instruction set architecture (ISA), which supports trusted execution [25]. One notable fact is that unlike other TEEs, its trusted computing base (TCB) only includes the processor and the application code inside the enclave. It is designed to make it easy to run arbitrary application code inside the enclave, while addressing the performance limitations of previous approaches [25]. This commoditization of TEE has drawn attention of the research community. A few pioneering studies [7, 23, 31] have shown that SGX can not only support traditional TEE applications, but also dramatically enhance the security and privacy of existing cloud applications.

We believe SGX can potentially have significant impact on networking, because many networking applications run on commodity hardware. Thus, this paper takes a first attempt at answering the following question: *what impact does commoditization of TEEs have on networking?* To take a more systematic approach, we look at three different types of networking software. To demonstrate practical implications, we apply SGX to problems that are currently relevant.

First, we show that SGX can be applied to solve policy privacy issues in software-defined inter-domain routing. Privacy is important in inter-domain routing because ISPs do not want to disclose their routing policies for security and commercial reasons [39]. Privacy is one of the main barriers to having a centralized controller for SDN-based inter-domain routing. A recent proposal uses secure multi-party computation (SMPC) to achieve policy privacy [17]. However, transforming arbitrary computation to SMPC is non-trivial, and it suffers from severe performance issues. We show that appropriately leveraging the hardware protection of SGX results in a more straight-forward design without significant impact on performance. Second, we explore the design space of applying TEE to secure the Tor anonymity network. Because Tor relies on volunteer-provided hardware, it is particularly vulnerable to malicious modifications by curious volunteers or by harmful attackers who have control over the machines [22, 35]. Also, by design, Tor places much trust on the directory authorities, who are also vulnerable to attacks [11]. In exploring the design space, we show that security and privacy of Tor can be strengthened, and SGX can enable a completely distributed service without directory authorities. Finally, we show that

HotNets '15 November 16–17 2015, Philadelphia, PA USA

http://dx.doi.org/10.1145/2834050.2834100

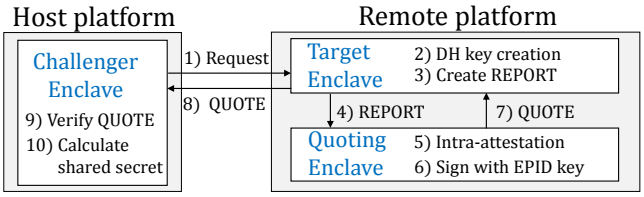

Figure 1: Bootstrapping secure channel during attestation

SGX can also be used to securely introduce in-network functionality into TLS sessions. To demonstrate the feasibility, we prototype these applications on OpenSGX [2], an open source emulator that emulates SGX instructions. We also estimate the overhead of our new design using our prototype. Our performance characterization shows that SGX-enabled applications introduce modest performance overhead.

2. BACKGROUND ON INTEL SGX

We briefly introduce key features of Intel SGX related to the design choices we make. Detailed specifications are described in Intel SGX white papers [18, 19, 25].

2.1 Intel SGX Overview

Intel SGX allows applications or part of an application code to be executed in a secure container, called an *enclave*. It protects them from other applications and privileged system software, such as the operating system (OS), hypervisor, and BIOS. To achieve protection against privileged software, memory content of the enclave is stored inside Enclave Page Cache (EPC), which is protected memory where encrypted enclave pages and Intel SGX data structures are stored. A page table that maps enclave pages onto EPC frames is managed by the OS. However, the OS cannot see the memory content because EPC region is encrypted by the memory encryption engine (MEE) within the CPU; only the enclave that is associated with the EPC page can access it. The processor maintains enclave page cache map (EPCM) to keep meta-data associated with each EPC page for access protection. Privileged software can only launch denial-of-service attacks, but otherwise the enclave is secured by hardware design.

Before an application can be executed inside the enclave, its code, data, and stack must be loaded into an enclave. Intel SGX supports a set of privileged instructions for this. After provisioning the enclave with appropriate memory content (e.g., code, data, and stack) is finalized, the hardware "measures" the identity of the software (i.e., a SHA-256 digest of enclave contents) inside the enclave, and enforce that only the software whose integrity is verified can be executed.[1]

Threat model: SGX assumes that an adversary can compromise any software components including the operating system, hypervisor, and firmware. Also, hardware components (e.g., memory and I/O devices) can be inspected by an attacker except for the CPU package itself. However, denial-of-service is out of scope; a malicious privilege software could easily crash or halt the system. We adopt the same threat model as SGX throughout this paper for all of our example applications. For example, in Tor network, we assume that an attacker can obtain full control of the system software and hardware (except the CPU package) of onion routers.

2.2 Remote Attestation

SGX also provides a remote attestation feature, in which an enclave can verify the integrity of a target enclave running on a remote, SGX-enabled platform [4]. Intel SGX provides EGETKEY and EREPORT instructions to enable this.

We first explain the intra-attestation procedure in which two enclaves, A and B, running on the same host verify the identity of the other. First, enclave A obtains the identity of B. Then, using the EREPORT instruction, it creates a REPORT data structure that contains the hash value of the two enclaves (enclave identities), public key of the signer who signed the identity of A (signer identity), some user data, and a message authentication code (MAC) over the data structure. The MAC is produced with a report key, only known to the target enclave and the EREPORT instruction on the same machine. Using EGETKEY, enclave B obtains the report key used to compute the MAC, and verifies that the REPORT is valid to ensure that they are running on the same host. It then verifies the identity of A (i.e., its content is not altered). Finally, enclave B reciprocates this process for mutual verification.

Remote attestation extends intra-attestation to attest the identity of a remote enclave. Intel SGX uses a specially provisioned enclave, called *quoting enclave*, whose identity is well-known, as shown in Figure 1. Only the quoting enclave can access the processor key used for attestation. Figure 1 illustrates the process. When a remote attestation request arrives from a challenger enclave, the target enclave performs (intra-)attestation with the quoting enclave residing in the same host. The quoting enclave then creates a signature of attestation result (QUOTE), using the private key of the CPU. [2] The enclave on the host platform can then verify the signature using the remote platform's public key. As part of remote attestation, two remote enclaves can bootstrap a secure channel by performing a Diffie-Hellman key exchange. For this, the host and target enclave embed Diffie-Hellman parameters in messages 1) and 8) to derive a shared secret, similar to TLS handshaking.

3. APPLICATION CASE STUDIES

This section demonstrates that the commoditization of trusted execution enabled new opportunities in networking, using several case studies that explores the new design space.

3.1 Software-Defined Inter-Domain Routing

Many new designs for inter-domain routing aim at providing new features (i.e., flexible route selection [16, 36, 37] and verifiability [26, 39]). However, in inter-domain routing, introducing new features often comes at the cost of privacy, which is something that ISPs are not willing to sacrifice [17, 39].

[1]The assumption is that the identify of the software is previously signed by an authority that a user trusts.

[2]Intel actually uses a group signature scheme (EPID [8]) for attestation and verification of SGX-equipped platform.

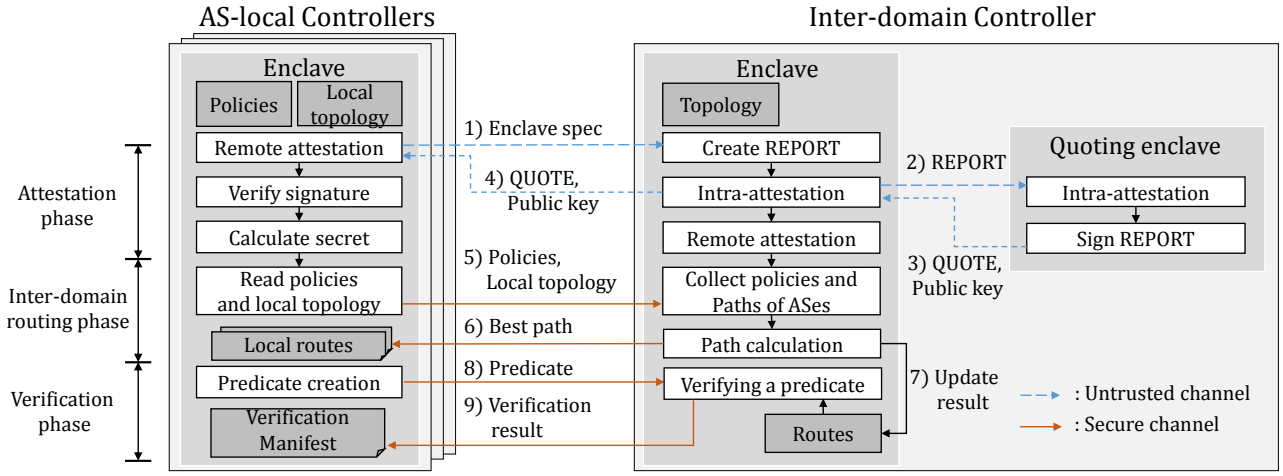

Figure 2: System overview illustrating the interactions between inter-domain controller and AS-controllers

Thus, providing rich features while ensuring the privacy has been a challenge in inter-domain routing [39].

Recently, software-defined networking approaches that utilize centralized decision making have demonstrated that they enable new properties and features, such as fast convergence, application-specific peering, and traffic engineering [16, 17]. However, SDN-based inter-domain routing exacerbates the privacy issues because it requires the logically centralized controller to learn the topologies and policies of the autonomous systems (ASes) [16]. A recent study proposes using secure multi-party computation (SMPC) to address this issues [17]. However, it requires non-trivial design, and the computational complexity of SMPC is prohibitively expensive.

TEEs provide new opportunities for ensuring privacy in SDN-based inter-domain routing. Our core idea is to enclose all private information inside the enclaves and allow all communication to happen between enclaves through a secure channel. Figure 2 shows an overview of our design in which two main components run inside the enclave: AS-local controller and inter-domain controller. ASes, who are part of the inter-domain routing system, maintain a common code base for the inter-domain controller that they agree upon. The ASes inspect the source code of the inter-domain controller to make sure that no private information (e.g., policy and topology) is explicitly sent to the outside world. We assume ASes follow a deterministic compilation process to obtain the correct identity of the inter-domain controller. At runtime, each AS-local controller verifies the integrity of the code using remote attestation, upon connecting to the inter-domain controller. Communication between the AS-local and inter-domain controller is done through a secure channel that is established during remote attestation. The AS-controllers and the inter-domain controller mutually authenticate to verify each others' identities. AS-local controllers send their policies and topology information to the inter-domain controller through the secure channel. The inter-domain controller then computes paths for all ASes and sends routes for each AS.

Policy verification: Inter-domain routing policies are determined by business contracts, such as peering agreements that contain "promises" that ASes make to others. For example, an AS may make a promise to one of the customers to prefer the customer's route over others [39]. However, verifying whether the outcome (or policy) stays up to the promises is difficult, and as a result these promises are not always kept in practice [39]. Existing approaches, such as SPIDeR [39], propose to verify whether ASes live up to their promises by collaboratively verifying the promise. However, SPIDeR relies upon the condition that all neighbors of a promise-breaking AS correctly follows their design to perform the verification. Furthermore, doing this in a privacy preserving fashion is non-trivial. In contrast, our design can enable such verification in a more direct and straight-forward manner while ensuring policy privacy.

To address the problem, we allow a query to be executed in the verification model inside the enclave of the inter-domain controller. The verification module executes the query against the decisions that the controller has made. The query is a Boolean condition that an AS wants to verify concerning the behavior of other ASes that it has a business relationship ship with. For example, two ASes, A and B, agree upon the condition to be verified, according to their business agreement (e.g., is the route announced by A most preferred by B?). The AS who was promised can verify whether the other is living up to its promise. The AS who made a promise can ensure that the query examines only the minimal condition required to verify the agreement, without leaking additional information. Following the idea of SPIDeR, the inter-domain controller verifies this over all routes that A receives, but does this securely within the enclave. The controller ensures that only the predicates agreed upon by the two ASes are verified. As a result, the verification process does not leak any extra information. Finally, during the entire process, privacy sensitive information does not leave the hardware-protected enclaves, as illustrated in Figure 2.

3.2 Tor Anonymity Network

Several attacks on Tor [20, 22, 35] assume that attackers have one or more compromised Tor onion routers (ORs). Because Tor relies on volunteer nodes, once they are admitted in the

system, it is easy for their owners to modify the software to launch attacks. For example, when the malicious Tor node is selected as an exit node, an attacker can modify the plain-text, unless end-to-end encryption is used by the end-user. It is shown that a single compromised node is sufficient to break the anonymity [22, 35].

Also, attacks on directory authorities are a constant threat to Tor [3]. Directory authorities perform admission control, determine the liveness of ORs, flag potentially malicious ORs, and even drop compromised ORs from Tor. To avoid a single point-of-failure and be robust against subversion attacks, Tor maintains multiple independent directory servers and builds consensus on active/legitimate ORs through majority vote. If directory authorities are subverted, attackers can admit malicious ORs or disable the Tor network. Note, multiple directory authorities have been actually compromised [11], which caused all nodes to update their software (e.g., directory server information and keys).

We take a look at how Tor's security model can be strengthened with respect to the deployment of SGX. We assume an incremental deployment model and discuss three deployment phases in the order of the ease of deployment. We assume that 1) the Tor source code is extensively verified by the community, 2) SGX-enabled nodes run Tor inside the enclave[3], and 3) the Tor foundation publishes a signed certificate of legitimate software that contains the identities.

SGX-enabled directory: Currently, Tor runs nine directory authorities [35]. If all of these directory servers are SGX-enabled, they can keep authority keys and list of Tor nodes inside the enclaves. This ensures that the attacks cannot steal the keys or data even if they compromise the host running the directory. Using SGX, directory servers can also attest each other to ensure the integrity and communicate through the secure communication channel. The attackers can still launch denial-of-service attacks (e.g., by killing the process), but they cannot alter the directory behavior (e.g., to launch tie-breaking attack or to admit malicious ORs [12]).

Incremental addition of SGX-enabled ORs: Once ORs are SGX-enabled, authorities can attest their integrity. Malicious Tor nodes fail to pass an enclave integrity check, because compromised OR executes additional operations, such as tampering or snooping. Another advantage is that admission of new ORs can be done automatically. This may serve as an incentive to deploy SGX-enabled ORs because currently addition of new ORs requires manual approval from majority of directory authorities, which is a bottleneck. We believe that SGX-enabled ORs will strengthen the trust model of Tor, because current model of manually admitting ORs essentially relies on trust on non-trustworthy volunteers [22, 35]. However, incremental deployment raises new issues, such as finding an interim solution that balances security and privacy with performance and efficiency in the Tor network.

[3]There are two approaches to run software in TEE. The traditional one is to put minimally required software components in TEE to reduce the trusted computing base. The other is to run unmodified software on TEE [7]. This is still an area of active research.

Fully SGX-enabled setting: If all Tor components, including authority nodes, Tor nodes and clients, are SGX-enabled, it is possible to defend all attacks mentioned above that change the behavior of director servers or ORs, such as the bad apple attack [20, 22, 35]. Using remote attestation, each Tor component can check 1) target program's integrity, 2) its certificate, and 3) whether it is running on the SGX-enabled platform or not. By inspecting these properties, each Tor component can trust each other because it verifies that the other is running the legitimate version of Tor and also establishes a secure communication channel with the other party. The directory authorities do not have to vote for detecting bad-exit nodes because problematic Tor nodes are excluded during the remote attestation. In fact, a new Tor design is possible that does not require directory authorities that manually maintain and check the membership, because verification is done by hardware through SGX. Tor can utilize a distributed hash table to track the membership, similar to other peer-to-peer systems [34]. We believe many research opportunities remain in realizing or extending our suggested models.

3.3 Secure In-network Functions

The use of in-network functions (or middleboxes) is popular on the Internet. However, widespread use of TLS protocol disrupts in-network processing since only endpoints of communication can access the plain-text. Existing approaches to this problem include augmenting the TLS protocol to explicitly include middleboxes [28], designing a new protocol and encryption schemes that allow operations over encrypted traffic [32], and passing session keys out-of-band.

SGX enables yet another possibility. The key idea is that endpoints use a remote attestation to authenticate middleboxes and give their session keys through the secure channel to in-path middleboxes. When both end-points are SGX-enabled, it can be used to allow only the middleboxes that both end-points agree upon decrypt/encrypt the TLS traffic. For example, the end-points can attest each other and the middlebox to achieve this. Passing session keys through the secure channel can be also done unilaterally by either of the two end-points, which enables several more use cases. For example, TLS traffic in enterprise networks can be sent to the SGX-enabled cloud for deep packet inspection. Or service providers can deploy middleboxes that inspect TLS traffic in a more secure fashion. We leave the detailed design and comparison with alternative approach as future work.

4. SECURE EXECUTION OF SHARED CODE

SGX provides isolation through the trusted CPU, and remote attestation through the cryptographic guarantee of a user's private key. Given the prior knowledge of a program running on the SGX and its signed key, the owner can remotely verify the integrity of the code in execution. Interesting though, there are many programs that are publicly available: for example, open source projects or programs shared by multiple entities. Thanks to modern code management systems, such as `git`, virtually everyone can validate the integrity of the

entire project by comparing its local history with the remote, shared repository. If there were an unauthorized change to the program's history, users or programmers can promptly flag the fraud.

Given the openness of the project and with the power of isolation provided the SGX, users now can privately and securely (e.g., no tampering and no explicit information leakage) run the program as long as they share the private key for the attestation. For example, Tor nodes can be launched, executed and verified by anyone who has the private key; for example, the Tor foundation can create and announce the shared key for attestation purposes. This allows anyone who obtains the valid code and the "open" private attestation key from the open project can: 1) build the Tor binary program [4]; 2) create a valid signature for the program; 3) validate the integrity of the remote Tor program leveraging the remote attestation feature of SGX. At the same time, Intel SGX's isolation property ensures that no one, including the donator of the hardware for the node, can subvert the running Tor programs.

We believe this idea can also be extended to any open projects. The SDN inter-domain controller of § 3.1 can benefit from this design as well. Participating ASes who connect to the controller can inspect and verify the controller's code to make sure that it does not leak any policy information on purpose (i.e., the code does not contain any logic to explicitly send the information to the outside world). They also share a private key for attestation purposes among themselves. Then, by relying on SGX's isolation and remote attestation properties, ISPs can ensure privacy.

5. PRELIMINARY EVALUATION

We evaluate the cost of using SGX functionality to improve the privacy and security of network application. In theory, enclaves running Intel SGX perform near to the native speed of a processor if no external communications or interrupts (e.g., asynchronous exits in SGX) are incurred. To characterize the potential performance penalty of adopting SGX in the absence of real SGX hardware platform, we provide a model (e.g., hardware parameters from recent literature) and measure how many of such operations are required in performing a high-level function of secured network applications. We characterize the overhead using a prototype of software-defined inter-domain routing (similar to that of Figure 2) developed using OpenSGX [2][5]. We use a Quad core Intel i5-4690 3.5GHz CPU machine running Linux 3.11.0.

Overhead characterization: We measure the overhead of two key elements that dominates the overhead of running Intel SGX: remote attestation and I/O inside the enclave. We provide a rough estimate of the cost of these operations in terms of CPU cycles, using our prototype. We assume each SGX instruction [7] takes 10K CPU cycles and measure the instructions per cycle (IPC) by executing applications natively without OpenSGX to estimate CPU cycles for normal instructions. We measure the run-time overhead excluding the cost launching an SGX application (e.g., loading code pages into EPC pages) because it is a one-time cost.

[4]For simplicity, we assume that the compilation process is deterministic [13, 29].

[5]OpenSGX is the only publicly available platform at the time of writing.

	Target		Quoting		Challenger	
	w/o DH	w/ DH	w/o DH	w/ DH	w/o DH	w/ DH
SGX(U) inst.	20	20	17	17	8	8
Normal inst.	154M	4338M	125M	125M	124M	348M

Table 1: Number of instructions during remote attestation. "w/ DH" denotes remote attestation with DH key exchange and SGX(U) stands for usermode SGX instructions.

	SGX (1 packet)		SGX (100 packets)	
	w/o crypto	crypto	w/o crypto	crypto
SGX(U) inst.	6	6	204	204
Normal inst.	13K	97K	136K	972K

Table 2: Number of instructions of a single packet transmission. "crypto" includes overhead of symmetric key encryption for secure communication.

Type	Number of attestations
Inter-domain routing	number of AS controller
Tor network (Authority)	number of reachable exit nodes
Tor network (Client)	number of authority nodes
TLS-aware middlebox	number of in-path middleboxes

Table 3: Number of remote attestations for each design

Cost of remote attestation: Table 1 shows the cost of remote attestation in terms of the number of instructions executed in the target, quoting, and challenge enclaves. As part of remote attestation, we can establish a shared secret using Diffie-Hellman (DH). Table 1 reports the overhead with and without DH. We set the DH parameter as 1024-bit and use AES-ECB mode as a symmetric key operation with 128-bit key using polarssl [5]. The challenger enclave consumes 626M cycles[6]. The remote platform, which executes the quoting and target enclave, consumes 8033M cycles. Note the Diffie-Hellman key exchange takes up 90% of the cycles. This suggests that the cost of each remote attestation is small.

Table 3 shows the number of remote attestations required for each of the three applications discussed in § 3. The number of remote attestations required is proportional to the size of each network. Note, remote attestation occurs only at the beginning when two parties communicate for the first time. Thus, the overhead of remote attestation is minimal.

Cost of I/O inside enclaves: We also measure the cost of packet I/O within the enclave. For this, we implement and run a simple server program which sends an MTU sized packet inside an enclave. Note each I/O operation incurs context switches (exit and resuming the enclave) and additional instructions. Table 2 shows number of instructions used to send a single packet and 100 packets in a batch with and without encryption for secure communication. It shows that while the cost of a single I/O operation is high, the cost can be amortized with batched I/O.

[6]$10k\times$ [# of SGX(U) instructions] + $IPC\times$ [# of normal instructions]. Here, the resulting average IPC is 1.8.

	Inter-domain		AS-local (avg.)	
	w/o SGX	w/ SGX	w/o SGX	w/ SGX
SGX(U) inst.	-	1448	-	42
Normal inst.	74M	135M	13M	24M

Table 4: Costs of SDN-based inter-domain routing. "w/o SGX" denotes executing applications natively without SGX and the result of AS-local is the average of 30 controllers.

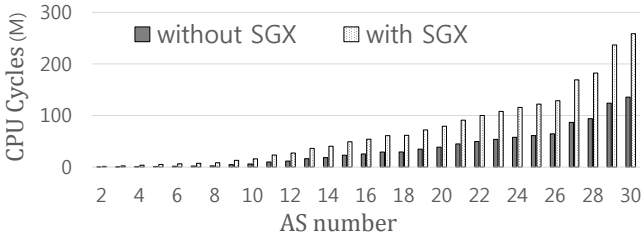

Figure 3: Number of CPU cycles consumed in the main controller as AS number increases

Software-Defined Inter-domain Routing: We quantify the overhead of the SGX-enabled software-defined inter-domain routing controllers. In our implementation, the AS-local controllers send their BGP-like policy to the inter-domain controller. We then create a random topology with 30 ASes with hypothetical business relationships. We model export rules according to their business relationship (i.e., peer, customer, and provider) and assume each AS has a local preference. The inter-domain controller then computes routing paths for all ASes using the rules of BGP. We verify the correctness of its output using GNS3 [1].

Table 4 shows the additional cost of using SGX. We compare the instructions executed with the controller running inside the enclave with OpenSGX and the one that run natively without using SGX. To quantify the overhead of normal, steady-state operation, we exclude the cost of enclave initialization and remote attestation that occurs only at the beginning. The inter-domain controller consumes 82% more normal instructions and AS-local controller requires 69% more on average than the non-SGX counterparts. The overhead is mainly due to in-enclave I/O and dynamic memory allocation that cause context switches.

Figure 3 shows the CPU cycles consumed by the inter-domain controller by varying the number of ASes. The overhead gradually increases as more ASes participate because it makes the topology more complex. SGX-enabled inter-domain controller consumes 90% more CPU cycles compared to the one without SGX support. Considering that our inter-domain controller performs only BGP-like path computation and supports no other desirable features of an SDN controller, this is modest overhead.

6. RELATED WORK AND DISCUSSION

TEE has been an effective way of guaranteeing safe execution of trusted application in mobile and PC environments. Many types of TEEs have been deployed, including Trusted Platform Module, ARM TrustZone, Intel TXT, and AMD SVM, over time. Traditional trusted computing relies on chain of trust, which increases the size of TCB. Flicker [24] dramatically reduces the TCB and allows piece of application code to run inside the isolated environment. However, TEE's usage has been rather limited due to performance issues. Intel SGX removes this limitations by supporting native performance for applications running inside enclaves.

One of the early applications of SGX came from the cloud [7, 10, 30, 31]. Recently, Haven [7] proposed a cloud software infrastructure that supports unmodified application binaries to run inside the enclave. VC3 [31] suggests using SGX for ensuring privacy in data analytics in the cloud. Earlier application of trusted computing to networking applications include establishing a stable pseudonym and secure channel in P2P networks [6], applying TPM for securing TLS [14, 15], and providing data-plane fault isolation in the network [38]. Our work focuses on identifying a wide range of new and practical networking applications that SGX enables. We explore their design space to demonstrate how SGX can be used to solve currently relevant problems in the field.

Discussions: This work identifies new design space that SGX enables. Many research issues still remain in securing enclave applications. For example, an enclave application can be subject to Iago attacks [9], if it blindly relies on external services (e.g., system call). The enclave program must verify/sanity check the return values and output parameters of system calls. The SGX specification also adopts this practice in its support for dynamic memory allocation [19]. Also, to minimize the attack surface, the system call interface to be used within the enclave should be narrow [7]. Finally, verifying the application behavior is also not trivial and currently requires manual effort. An important research issue to enable automatic verification of application source code (e.g., the program actually does not leak private data) [33].

7. CONCLUSION

Motivated by the recent advances in trusted execution environments, this paper explores new research opportunities that wide-spread use of TEE might enable. We demonstrate its practical implications by exploring the design space for SGX-enabled software-defined inter-domain routing, peer-to-peer anonymity networks (Tor), and middleboxes. Our design shows that SGX can not only improve the security and privacy of existing applications, but also enable new services. Our preliminary evaluation based on the prototype built on top of OpenSGX shows that SGX-enabled applications results in modest performance degradations compared to a version that does not utilize SGX, while benefiting from improved security and privacy.

ACKNOWLEDGEMENTS

This work was supported in part by the ICT R&D program of MSIP/IITP, Rep. of Korea (No.B0190-15-2011), Korea-US Collaborative Research on SDN/NFV Security/Network Management and Testbed Build and the Basic Science Research Program of NRF funded by MSIP, Rep. of Korea (NRF-2013R1A1A1076024). Taesoo Kim is partially supported by ONR N00014-15-1-2162, NSF DGE-1500084 and DARPA-15-15-TC-FP-006.

References

[1] Gns3 network simulator. http://www.gns3.com/.

[2] OpenSGX: An open platform for intel sgx. https://github.com/sslab-gatech/opensgx.

[3] Possible upcoming attempts to disable the tor network. https://blog.torproject.org/blog/possible-upcoming-attempts-disable-tor-network, December 2014.

[4] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *International Workshop on Hardware and Architectural Support for Security and Privacy*, pages 1–8, Tel-Aviv, Israel, 2013.

[5] P. Bakker. Polarssl project. http://polarssl.org/.

[6] S. Balfe, A. Lakhani, and K. Paterson. Trusted computing: providing security for peer-to-peer networks. In *Peer-to-Peer Computing, Fifth IEEE International Conference on*, pages 117–124, Aug 2005.

[7] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. In *USENIX OSDI*, 2014.

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

[9] S. Checkoway and H. Shacham. Iago attacks: Why the system call api is a bad untrusted rpc interface. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 253–264, 2013.

[10] C. Chen, H. Raj, S. Saroiu, and A. Wolman. cTPM: A cloud TPM for cross-device trusted applications. In *USENIX NSDI*, 2014.

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

[12] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.

[13] J. Edge. Lots of progress for Debian's reproducible builds. http://lwn.net/Articles/630074/, Jan. 2015.

[14] Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, and N. Asokan. Beyond secure channels. In *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*, STC '07, 2007.

[15] K. Goldman, R. Perez, and R. Sailer. Linking remote attestation to secure tunnel endpoints. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing*, STC '06, 2006.

[16] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett. SDX: A software defined internet exchange. In *ACM SIGCOMM*, 2014.

[17] D. Gupta, A. Segal, A. Panda, G. Segev, M. Schapira, J. Feigenbaum, J. Rexford, and S. Shenker. A new approach to interdomain routing based on secure multi-party computation. In *ACM HotNets*, 2012.

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

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

[20] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the tor network. Technical report, DTIC Document, 2014.

[21] T.-W. Johnny Ngan, R. Dingledine, and D. Wallach. Building incentives into tor. In R. Sion, editor, *Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 238–256. Springer Berlin Heidelberg, 2010.

[22] 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 *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats*, pages 2–2, 2011.

[23] Y. Li, J. M. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry. MiniBox: A Two-Way Sandbox for x86 Native Code. In *Proceedings of the USENIX Annual Technical Conference*, June 2014.

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

[25] 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 *International Workshop on Hardware and Architectural Support for Security and Privacy*, pages 1–8, Tel-Aviv, Israel, 2013.

[26] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra. Verifying and enforcing network paths with icing. In *ACM CoNEXT*, 2011.

[27] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The cost of the "S" in HTTPS. In *ACM CoNEXT*, 2014.

[28] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. Lopez, K. Papagiannaki, P. R. Rodriguez, and P. Steenkiste. multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *ACM SIGCOMM*, 2015.

[29] M. Perry. Deterministic Builds Part Two: Technical Details. https://blog.torproject.org/blog/deterministic-builds-part-two-technical-details, Oct. 2013.

[30] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-sealed data: A new abstraction for building trusted cloud services. In *USENIX Conference on Security Symposium*, 2012.

[31] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. Vc3: Trustworthy data analytics in the cloud. Technical Report MSR-TR-2014-39, Microsoft Research, February 2014.

[32] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM*, 2015.

[33] R. Sinha, S. Rajamani, S. A. Seshia, and K. Vaswani. Moat: Verifying confidentiality of enclave programs. In *ACM CCS*, 2015.

[34] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, 2001.

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

[36] X. Yang, D. Clark, and A. Berger. Nira: A new inter-domain routing architecture. *Networking, IEEE/ACM Transactions on*, 15(4):775–788, Aug 2007.

[37] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. Scion: Scalability, control, and isolation on next-generation networks. In *IEEE Symposium on Security and Privacy*, 2011.

[38] X. Zhang, Z. Zhou, G. Hasker, A. Perrig, and V. Gligor. Network fault localization with small tcb. In *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, Oct 2011.

[39] M. Zhao, W. Zhou, A. J. Gurney, A. Haeberlen, M. Sherr, and B. T. Loo. Private and verifiable interdomain routing decisions. In *ACM SIGCOMM*, 2012.