Bootstrapping trust in software defined networks

Nicolae Paladi[1,*], Christian Gerhmann[2]

[1]RISE SICS, Sweden
[2]Lund University, Sweden

Abstract

Software-Defined Networking (SDN) is a novel architectural model for cloud network infrastructure, improving resource utilization, scalability and administration. SDN deployments increasingly rely on virtual switches executing on commodity operating systems with large code bases, which are prime targets for adversaries attacking the network infrastructure. We describe and implement TruSDN, a framework for bootstrapping trust in SDN infrastructure using Intel Software Guard Extensions (SGX), allowing to securely deploy SDN components and protect communication between network endpoints. We introduce *ephemeral flow-specific pre-shared keys* and propose a novel defense against *cuckoo attacks* on SGX enclaves. TruSDN is secure under a powerful adversary model, with a minor performance overhead.

1. Introduction

Renewed and widespread interest in virtualization – along with proliferation of cloud computing – has spurred a series of innovations, allowing cloud service providers to deliver on-demand compute, storage and network resources for highly dynamic workloads. Consequently, more hardware and virtual components are added to already large networks, complicating network management. To help address this, SDN emerged as a novel network architecture model. Separation of the *data* and *control* planes is its core principle, allowing network operators to implement high-level configuration goals by interacting with a single *network controller*, rather than configuring discrete network components. The controller applies the configuration to the *network edge*, i.e. to its global view of the data plane [16]. Data and control plane separation in SDN challenges network infrastructure security best practices evolved in the decades since packet-switched digital network communication gained popularity [14, 20, 26].

In the cloud infrastructure model, SDN allows tenants to configure complex topologies with rich network functionality, managed by a network controller. The availability of a global view of the data plane enables advanced controller capabilities – from pre-calculating optimized traffic routing to managing applications that replace hardware middleboxes. However, these

capabilities also make the controller a valuable attack target: once compromised, it yields the adversary complete control over the network [33]. The global view itself is security sensitive: an adversary capable of impersonating network components may distort a controller's global view and influence network-wide routing policies [17].

Virtual switches are another category of security sensitive components in SDN deployments. They execute on commodity operating systems (OS) and are often assigned the same trust level and privileges as hardware switches – specialized network components with compact embedded software [34] or application-specific integrated circuits. Commodity OS are likely to contain security flaws which can be exploited to compromise virtual switches. For example, their configuration can be modified to disobey the protocol, breach network isolation and reroute traffic to a malicious destination or compromise other network edge elements through lateral attacks. Such risks are accentuated by the extensive control a cloud provider has over the infrastructure of its tenants.

Security and isolation of tenant infrastructure can be strengthened by confining select SDN components to trusted execution environments (TEE) and attesting their integrity before provisioning security-sensitive data. TEEs with strong security guarantees can be built using SGX, a set of recently introduced extensions to the x86 instruction set architecture and related hardware [1, 22]. Earlier work used SGX to protect computation in cloud

*Corresponding author. Email: nicolae.paladi@ri.se

environments, by executing modified OS instances in SGX enclaves [2] or a data processing framework in a set of SGX enclaves [39]. However, while both of the above efforts *highlighted* the need to secure network communication, they did not *address* it.

1.1. Contribution

This paper makes the following contributions:

- We present TruSDN, a framework to bootstrap trust in SDN infrastructure.

- We introduce flow-specific pre-shared keys for communication protection.

- We propose a defense against cuckoo attacks [30], based on properties of the enhanced privacy ID (EPID) scheme [4] used for remote enclave attestation.

- We describe the implementation and a performance evaluation of TruSDN.

1.2. Organization

We introduce the system model in Section 2, describe the adversary model in Section 3 and the design of TruSDN in Section 4. In Section 5 we provide a security analysis, describe the prototype implementation and performance evaluation in Section 6 and review the related work in Section 7. We discuss future work in Section 8 and conclude in Section 9.

2. System Model

In this section we describe the SDN architectural model and the SDN deployment layers. Furthermore, we describe the use of TEEs based on Intel SGX.

2.1. Software Defined Networking

In this paper we target SDN in infrastructure cloud deployments. The system model follows the architecture presented in [5] and depicted in Figure 1.

The *data plane* includes hardware and software switch implementations. *Software switching* is used in cloud deployments due to its scalability and configuration flexibility. Figure 2 illustrates the software switching approaches for communication between two collocated endpoints. In a typical switch implementation, its kernel-space component is optimized for forwarding performance, lacks decision logic and only forwards packets matching rules in its *forwarding information base* (FIB) [24]. The FIB comprises packet forwarding rules deployed to satisfy network administrator goals. Mismatching packets are discarded or redirected to the *control plane* through the *southbound API*. While the data plane uses complementary functionality of

both virtual and physical switches, the role of the latter is often reduced to routing IP-tunneled traffic between hypervisors [31]. In this paper we do *not* address control of hardware switches and traffic routing between hosts; we assume that the physical network provides uniform capacity across hosts, based on e.g. equal-cost multi-path routing [18], such that if multiple equal-cost routes to the same destination exist, they can be discovered and used to provide load balancing among redundant paths. Overlay networks – e.g. VLANs or GRE [13] – are used for communication between endpoints. In this work, we focus exclusively on software switching and use the term "switch" to denote a virtual, software implementation. We refer to hardware switch implementations as "hardware switches".

In the *control plane*, high-level network operator goals are translated into discrete routing policies based on the *global network view*, i.e. a graph representation of the virtual network topology. The main component of a control plane is the *network controller*, which we define as follows:

Network Controller (NC) is a logically centralized component that manages network communication in a given deployment by updating the FIB with specific forwarding rules. The NC compiles forwarding rules based on three inputs: the dynamic global network view, the high-level configuration goals of the network operator, and the output of the network management applications.

The NC is typically implemented as part of a logically centralized *network OS*, which builds and maintains the global network view and may include a *network hypervisor*, to multiplex network resources among distinct virtual network deployments.

Southbound API is a set of vendor-agnostic instructions for communication between data and control planes. It is often limited to flow-based traffic control of the data plane, with management done through a configuration database [31].

Network operators use *network management applications* (NMAs), e.g. firewalls, traffic shapers, etc., to configure the network using high-level commands.

2.2. Deployment layers

We next describe the deployment layers of SDN infrastructure (Figure 3).

The *hardware layer* includes infrastructure for data transfer, processing and storage and is comprised of network hardware (including hardware switches and communication channels), hardware server platforms and data storage.

The *infrastructure layer* includes software components for virtualization and resource provisioning to infrastructure users, referred to as *tenants*. For network resources, this layer includes the network hypervisor, which creates *network slices* by multiplexing physical network
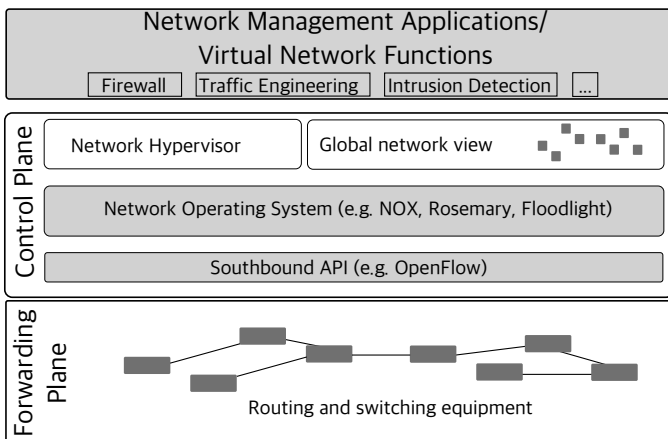
Figure 1. The SDN architectural model.

Figure 2. Communication paths between collocated endpoints: (1) virtual switch; (2) host-local, e.g. native bridging; (3) virtual queues in the NIC.

infrastructure between tenants. Infrastructure providers expose a *slice* (i.e. a quota) of network resources to the tenants.

The *service layer* includes components controlled by tenants. Network components operated by tenants are grouped into *network domains*, comprising the virtual network resources and topologies that logically belong to the same organizational unit and network slice, and perform related tasks or provide a common service. The *network hypervisor* ensures that a tenant's control plane can only control switches in its own slice. Within their slice, tenants have exhaustive creation, destruction and configuration privileges over components, such as instances of switches, the NC, NMAs and network domains. We define three *logical* communication segments (Figure 4): between the network controller and switches (α segments); among the switches on each host (β segments); between host-local switches and network endpoints (γ segments).

The network components operated by tenants are grouped into *network domains*, which comprise the virtual network resources and topologies which logically belong to the same organizational unit, belong to the same network slice, and perform related tasks or provide a common network service. Tenants create, modify, destroy domains and manage the communication flows between the network endpoints in their domain. A tenant's control plane can only communicate with switches in its global network view, which is limited to the respective network slice. Furthermore, on their control plane, tenants can deploy network management applications which operate on their global network view.

The *user layer* includes endpoint consumers of network services, e.g. virtualization guests, containers and applications in a network domain.
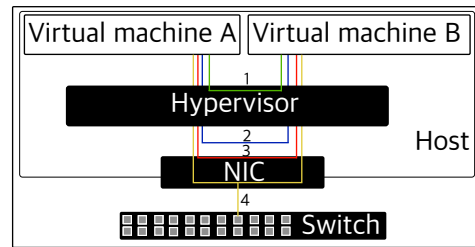
2.3. Trusted Execution Environments

The proposed solution relies on TEEs that *both* provide strong isolation and allow remote code and data integrity attestation. Such a TEE can be created using Intel SGX enclaves (introduced in [1, 22]) during OS runtime and relies for its security on a trusted computing base (TCB) of code and data loaded at build time, processor firmware and processor hardware. At build time, the CPU measures the loaded code, data and memory page layout. At initialization time, the CPU produces a final measurement, after which the enclave becomes immutable and cannot be externally modified. The CPU maintains the measurement throughout the enclave's lifetime to later assert the integrity of the enclave contents. Processor firmware is the root of trust (ROT) of an enclave. It prevents access to the enclave's memory segment by either the platform OS, other enclaves, or other external agents. Enclaves operate in a separate memory region inaccessible to non-enclave processes, called the enclave page cache (EPC). Multiple mutually distrusting enclaves can operate on the platform. The processor enforces separation of memory access among enclaves based on the layout in the *EPC map*. Program execution within an enclave is transparent to both the underlying OS and other enclaves.

Remote attestation allows an enclave to provide integrity guarantees of its contents [1]. For this, the platform produces an attestation assertion with information about the identity of the enclave and details of its state (e.g. the mode of the software environment, associated data, and a cryptographic binding to the platform TCB making the assertion). For *intra-platform attestation* (i.e. between enclaves on the same platform), the reporting enclave (*reporter*) invokes the EREPORT instruction to create a REPORT structure with the assertion and calculate a message authentication code (MAC), using a *report key*, known only to the target enclave (*target*) and the CPU. The structure contains a
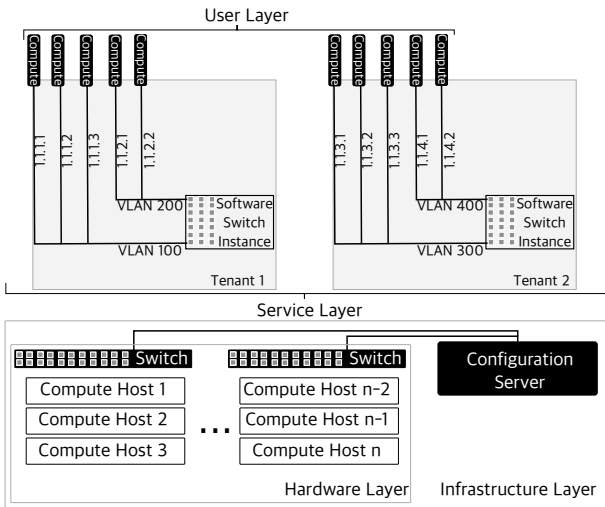
Figure 3. Deployment Layers.

Figure 4. Logical communication segments: α: between the NC and switches; β: among the switches on each host; γ between host-local switches and network endpoints.

user data field, where the reporter can store a hash of the auxiliary data provided. The target recomputes the MAC with its report key to verify the authenticity of the structure, and compares the hash in the user data with the hash of the auxiliary data, to verify its integrity. Enclaves then use the auxiliary data to establish a secure communication channel. For *inter-platform attestation* the remote verifier first sends a challenge to the enclave platform, where the challenge is complemented with the indentity of a *quoting enclave* (QE) and forwarded to the reporter, which appends the challenge response to the REPORT and attests itself to the QE. The QE verifies the structure, signs it with a platform-specific key using the *enhanced privacy ID group signature scheme* (EPID) [4] and returns it to the verifier, to check the authenticity of the signature and the report itself [1]. The use of the EPID scheme is part of the SGX implementation and allows to maintain the privacy of the platform which hosts the enclave.

3. Threat Model

We now describe the adopted adversary model, as well as the core security assumptions on which we base our design. The adversary model we adopt can be described by the capabilities of the adversary at the *network* and *platform* levels respectively (overview in Table 1).

3.1. Network infrastructure

For SDN infrastructure, we adopt the adversary model introduced in [9] and extended with SDN-specific attack vectors in [26]. We assume a powerful adversary ($\mathcal{Adv}$), which controls the cloud deployment network infrastructure; it can intercept, record, forge, drop and replay any message on the network, and is only limited by the constraints of the employed cryptographic
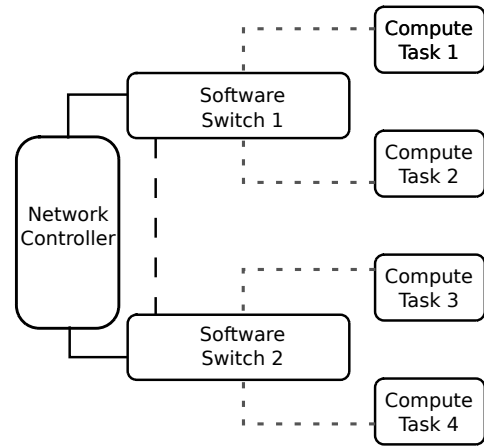
methods. Particularly, the $\mathcal{Adv}$ may forge messages that do not match any of the rules installed in the FIB. Furthermore the $\mathcal{Adv}$ may create own instances of switches and launch Sybil attacks [10] and launch other types of topology poisoning attacks [17] to distort the global network view. Finally, $\mathcal{Adv}$ can store arbitrary quantities of intercepted communication and attempt its decryption with encryption keys intercepted or leaked at a later point. It can analyze the traffic patterns in the network through passive probing and may disrupt or degrade network connectivity to achieve its goals [8]. We explicitly exclude Denial-of-Service attacks on the SDN infrastructure.

3.2. Platform

For platform security, we consider a powerful adversary, similar to [2, 39], that may control the entire software stack in the cloud provider's infrastructure.

On the hardware level, we assume the processor is correctly implemented and remains uncompromised; furthermore, we assume a reliable and secure source of random numbers (which can be provided by the CPU). $\mathcal{Adv}$ has full control over the remaining hardware, including memory, I/O devices, periferials, etc. Similarly, $\mathcal{Adv}$ fully controls the software stack, including the platform OS and the hypervisor. This implies that $\mathcal{Adv}$ may pause indefinitely the execution of the code in the TEE and return arbitrary values in response to OS system calls. However, a deployment orchestrator and NC execute under tenant control, on a fully trusted platform and software stack. We exclude side-channel attacks. While some side-channel attacks – e.g. timing, cache-collision, controlled channel attacks – can be mitigated through software modification [42], preventing other side-channel attacks – such as power analysis – requires hardware modifications. An $\mathcal{Adv}$ with advanced

Table 1. Summary of the $\mathcal{A}dv$ capabilities in relation to the adversary model.

Type	Network	Platform
Included	Intercept, record, forge, drop, replay messages; Analyze the traffic patterns; Disrupt or degrade network connectivity; Launch topology poisoning attacks	Control non-processor hardware; Control software stack OS, hypervisor; Pause execution; Deploy arbitrary software components; "Cuckoo attack": Forward function calls to compromised SGX enclaves; Return arbitrary values to system calls
Not included, mitigations known		Side-channels: cache-collision, controlled channel; Attacks on shielded execution;
Excplicitly excluded	Denial-of-Service (DoS) attacks	Side-channels: power analysis; DoS attacks

capabilities may leverage its full control over the OS to utilize the class of known attacks on shielded execution; while we do not address such attacks, they have known countermeasures [2, 6].

SGX, similar to other trusted computing solutions, is vulnerable to *cuckoo attacks* [30]. In one attack scenario, malware on the target platform forwards the messages intended for the *local* SGX enclave (SGX_L^E) to a remote enclave under $\mathcal{A}dv$'s physical control (*malicious* enclave, SGX_M^E). Having physical access to SGX_M^E, $\mathcal{A}dv$ can apply hardware attacks to violate its security guarantees. As a result, $\mathcal{A}dv$ controls all communication between the verifier and SGX_L^E, with access to an oracle that provides all of the answers a benign SGX^E would, but without its expected security properties.

Briefly, the adversary model for platform security largely matches the remote administrator capabilities of an infrastructure cloud provider.

4. Solution Description

In this section we present TruSDN, a framework for bootstrapping trust in SDN deployments. Its goal is to allow tenants to securely deploy computing tasks and create virtualized network infrastructure deployments, given the adversary model defined in Section 3. To satisfy this goal, the framework must satisfy the following set of requirements:

- *Authentication:* communication in the domain must the authenticated, and a secure enrollment mechanism for data plane components must be in place.

- *Topology integrity:* the NC must be protected from network components that attempt to distort the global network view.

- *Component integrity:* integrity of switches must be attested prior to enrollment and the cryptographic material required for their network access must be protected with a hardware ROT.

- *Confidentiality protection of domain secrets:* network domain secrets – such as VPN session keys – should not be revealed to the $\mathcal{A}dv$.

- *Protected network communication:* network communication in the tenant domain must be confidentiality and integrity protected.

4.1. TruSDN overview

We begin by introducing the building blocks of TruSDN (Figure 5).

Trusted Execution Environments: TruSDN uses TEEs that guarantee secure execution in the given adversary model, assuming the CPU and executed code are correctly implemented.

Protected Compute Tasks: *Security sensitive* compute tasks (CT) are deployed in TEEs. Such tasks include all operations that tenants aim to protect from the $\mathcal{A}dv$. However, CTs rely on the untrusted OS for I/O and support functionality.

Protected Data Plane: Switches are deployed in TEEs – they route traffic between CTs according to forwarding rules communicated through secure channels and maintained in the FIB. The FIB of the switches, and the key material necessary to establish the secure channels are stored in TEEs.

Attested code in TEEs: An orchestrator under tenant control attests the TEEs during network infrastructure deployment, to ensure integrity of the deployed code and data before keys or key material are provisioned to the respective TEE.

In a typical deployment scenario, the tenant invokes an orchestrator to deploy a switch *bootstrap application* on the hosts in the tenant's domain. The bootstrap application invokes a host-local SGX driver to build an SGX enclave containing a switch. Next, the orchestrator attests the created enclave (as described in Section 2.3) prior to enrolling the switch with the NC. The
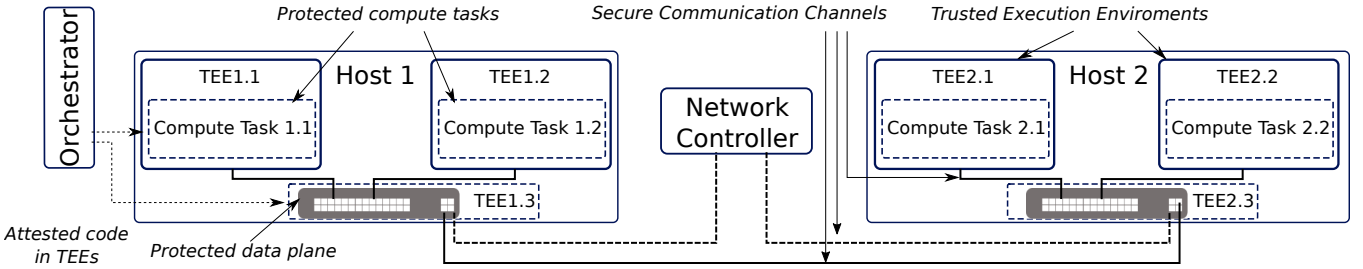
Figure 5. Illustration of core building blocks of TruSDN.

orchestrator uses the enclave's public key from the attestation quote to securely transfer the enclave-specific integrity and confidentiality protection session keys used to establish a protected communication channel between the NC and the TEE. Finally, the NC communicates any remaining security-sensitive payload to the created TEE, e.g. the initial FIB. Next, CTs are deployed in TEEs on the host and the switch forwards packets between the CTs, matching them against the rules in the FIB. Mismatching packets are forwarded to the NC, which may update the FIB with new rules. For clarity, we assume the orchestrator and NC are collocated on a platform under tenant control and view both as a single component, further referred to as "NC".

Secure Communication: TruSDN protects the communication between CTs, between switches and the NC, as well as among the switches, in the above adversary model. Communication security is ensured using confidentiality and integrity protection keys provisioned to authenticated network components and endpoints executing in TEEs. Furthermore, TruSDN leverages SDN principles to introduce a novel mechanism – per-flow communication protection using ephemeral flow-specific pre-shared keys (PSKs).

4.2. Cryptographic Primitives

We now define the cryptographic primitives and notations used in the remainder of this paper. We denote by $\{0,1\}^n$ the set of all binary strings of length n, and by $\{0,1\}^*$ the set of all finite binary strings. In a set U, we refer to the ith element as u_i, and use the following notation for cryptographic operations:

- Given an arbitrary message $m \in \{0,1\}^*$, we denote by $c = \mathsf{Enc}\,(K, m)$ a symmetric encryption of m using the secret key $K \in \{0,1\}^*$. The corresponding symmetric decryption operation is $m = \mathsf{Dec}(K, c) = \mathsf{Dec}(K, \mathsf{Enc}(K, m))$.

- We denote by pk/sk a public/private key pair for a public key encryption scheme. We denote by $c = \mathsf{Enc}_{\mathsf{pk}}\,(m)$ the encryption of message m with the public key pk, and the decryption by $m = \mathsf{Dec}_{\mathsf{sk}}(c) = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m))$.

- We denote a digital signature over a message m by $\sigma = \mathsf{Sign}_{\mathsf{sk}}(m)$ and the corresponding verification of a digital signature by $\nu = \mathsf{Verify}_{\mathsf{pk}}(m, \sigma)$, where $\nu = 1$ if the signature is valid and $\nu = 0$ otherwise.

- We denote a Message Authentication Code (MAC) using a secret key K over a message m by $\mu = \mathsf{MAC}(K, m)$.

We next describe key sharing and communication protection mechanisms on the identified logical segments. Table 2 summarizes the keys used by TruSDN.

4.3. SDN Trust Bootstrapping and Secure Communication

The first step in deploying a TruSDN infrastructure is to launch a set of trusted switches for connectivity and topology building. The NC requests the creation of switch enclaves to deploy switches in TEEs on hosts in its domain. Switches are deployed based on parameters provided by the NC in plaintext (application code and configuration). Next, the NC *attests* the integrity of switch enclaves and only *enrolls* the successfully attested ones (Figure 6). A TEE E_i is attested following the protocol introduced in [1]. With TruSDN however, the reporter generates an enclave-specific public-private keypair and submits its public key EK_i^{pk} along with the attestation data; a hash of the public key is stored in the `user data` field. The switch enclave is only enrolled to the global network view if its reported state matches the one expected by NC.

Having attested enclave E_i, NC communicates an Enrollment message (Table 3) with the enclave-specific pre-shared key K_i^{α} and domain-specific pre-shared key K_j^{β}, encrypted with an ephemeral key K_i'. Switches within a domain use K_j^{β} to protect communication on β segments. The NC appends a MAC of the message calculated with K_i'' and encrypts the keys K_i', K_i'' with EK_i^{pk}.

Once switches are deployed and enrolled, tenants may configure the network topology using the NC to update the switch FIBs. Communication on α segments – e.g. FIB updates or unmatched packets forwarded to the

Table 2. Summary of keys used in the TruSDN framework.

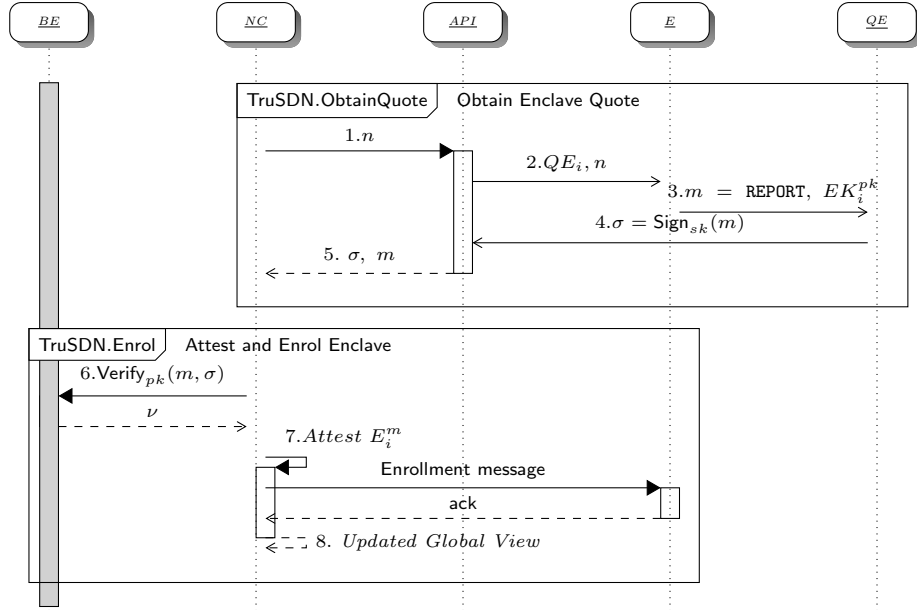| Key	Created by	Access	Usage
K_i^α	NC	NC, switch	Enclave-specific session, segment α
K_j^β	NC	NC , switch	Domain-specific session, segment β
K^j	NC	NC, switch	Ephemeral session key
K''	NC	NC, switch	Ephemeral MAC key
EK_i^{pk}	switch	public	Public key of the switch enclave
EK_i^{sk}	switch	switch	Private key of the switch enclave
CK_i^{pk}	CT	public	Public key of the compute task
CK_i^{sk}	CT	CT	Private key of the compute task
QE^{pk}	vendor	public	Public key of the quoting enclave
QE^{sk}	vendor	vendor, QE	Private key of the quoting enclave
SK_{ij}^γ	NC	NC, CT_i, CT_j	Ephemeral flow-specific pre-shared key

Figure 6. TruSDN enclave attestation and enrollment: (1.) Random nonce n is (2.) supplemented with the host QE identity; (3.) Quote m produced by the enclave is (4.) signed by the QE. (6.) The verifier checks the signature of the QE, (7.) attests the integrity of the enclave and (8.) only enrolls the enclave upon success. BE: back-end.

Table 3. Enrollment message sent by the NC upon switch enrollment.

$m = \mathsf{Enc}(K_i', (K_i^\alpha, K_j^\beta))$	$\mu = \mathsf{MAC}(K_i'', m)$	$\mathsf{Enc}(EK_i^{pk}, (K_i', K_i''))$

NC – is protected using the session key K_i^α (e.g. using TLS [12]), which never leaves the TEE.

Similarly, a secure channel is established among the switches within the same domain, using the pre-shared key K_j^β, to protect communication between switches on different hosts (e.g. TEEs 1.2 and 2.3 in Figure 5). K_j^β never leaves the TEEs, has a limited validity time and is periodically redeployed by the NC. On β segments, traffic may traverse multiple hardware switches, forwarded to the host over tunnels deployed on top of a standard routing protocol (e.g. [18]).

Next, the tenant may deploy CTs in TEEs and attest their integrity using the very same scheme and principles as for the switch deployment described above. The CTs and the network controller use the Enrollment message to establish a secure communication channel (e.g. TLS).

Once the NC has deployed and attested the TEEs with switches and CTs, intra-host communication (i.e. between two CT enclaves on the same host) is straightforward (Figure 7): when a packet m sent from C1 (e.g. a TLS ClientHello message) reaches the local host *switch A*, it attempts to match m against a FIB entry; if no suitable flow rule f is present, the switch forwards $\mathsf{Enc}(K_A^\alpha, m)$ to NC, which processes the packet, generates and deploys on the CTs C1, C2 a flow-specific pre-shared key SK_{12}^γ and finally updates the switch FIB with f, after which steps 2 and 3 are ignored; once the FIB is updated, the switch forwards m to C2, which
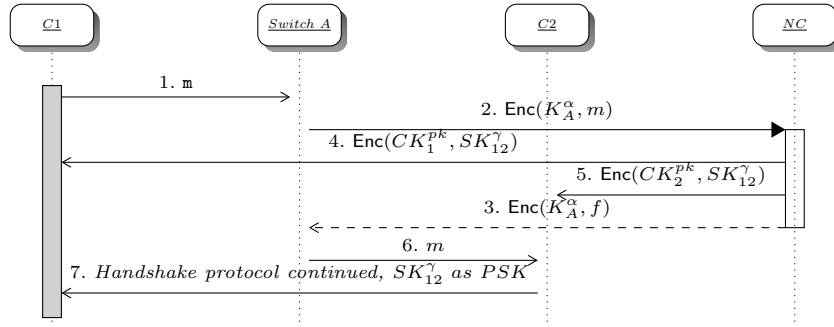
Figure 7. Intra-host communication with TruSDN.

continues the message exchange and uses SK_{12}^{γ} to protect the communication with C1, using e.g. TLS with a PSK ciphersuite[12].

Communication between CTs C1 and C3 deployed on distinct hosts is similar, with the only notable difference that the NC updates the FIB of the local switches on both hosts where C1, C3 are deployed.

In the above scenarios TruSDN leverages two aspects of the SDN model – **(1)** the deployment has a central authority (the NC) and **(2)** the first packet of a flow is forwarded to the central authority – to deliver on demand ephemeral PSKs to communication endpoints. This allows to relax the need for high-quality entropy being available to CTs (a known issue in virtualized environments [36]). Furthermore, this approach ensures communication security without compromising packet visibility – having control over the keys used to protect communication between the CTs allows the NC to maintain fine-grained insight into the traffic.

4.4. Preventing Cuckoo Attacks

To prevent cuckoo attacks [30], we propose a solution that leverages cryptographic properties of the EPID scheme used by the QE [4] and the *SIGn and Message Authentication* (SIGMA) protocol [41], which are both part of the Intel SGX implementation. The EPID scheme supports two signature modes: *fully anonymous mode* – the verifier cannot associate a given signature with a particular member of the group; *pseudo-anonymous mode* – the verifier can determine whether it has verified the platform previously. The unlinkability property distinguished in the two modes depends on the chosen *base*. A signature includes a pseudonym B^f, where B is the base chosen for a signature and revealed during the signature; f is unique per member and private. For a *random base* R, the pseudonym is R^f – in this case the signatures are unlinkable. For a *name base*, the pseudonym is N^f, where N is the name of the verifier – in this case the signatures remain unlinkable for *different* verifiers, while signatures with a common N can be linked. For privacy reasons, the EPID scheme currently implemented in Intel SGX accepts *name base*

pseudonyms only from verifiers authorized by the EPID authority [37], which is done by provisioning qualified verifiers with an X.509 certificate – e.g. an intermediate certification authority (CA) certificate – signed by the EPID authority acting as root CA.

We propose the following algorithm to prevent cuckoo attacks. At deployment time, the EPID authority issues, to an authorized verifier V_P, an intermediate CA verifier certificate for the platforms in the cloud provider's data center. Next, V_P attests its platforms following the SIGMA protocol and publishes a list of resulting platform EPID signatures and the signature name base, B_P^N. To guard against cuckoo attacks, tenants first request V_P to issue an X.509 certificate and enable them to become *authorized verifiers*. Next, tenants choose the same pseudonym base B_P^N (and a private f), follow the SIGMA protocol, and verify that the resulting signature is linkable to a signature in the published list. The cloud provider has multiple tools to protect platform privacy and prevent untrusted tenants from fingerprinting the platform infrastructure, e.g. limiting the validity of issued certificates, changing the name base, etc. Considering that the EPID scheme is currently not implemented in the SGX emulation software we used for prototyping, we intend to describe the implementation of the above algorithm in a follow-up report.

5. Security Analysis

In this section we analyze the security properties of the proposed framework in the adversary model described in Section 3. On the network level, many of the *Adv* capabilities are thwarted by first authenticating the switches deployed on the data plane, as well as the network edge (i.e. the compute tasks that generate or receive the network traffic), in combination with confidentiality and integrity protection of the traffic on the three identified segments. Authenticating the network components prevents topology poisoning attacks (a countermeasure mentioned in [17]), while confidentiality and integrity protection of all of the network traffic in the deployment prevents the *Adv*

from either learning the contents of the exchanged packets or successfully forging packets. The $\mathcal{A}dv$ may in this case still intercept and record messages. However, collecting encrypted traffic does not yield the $\mathcal{A}dv$ any more information about the contents of the exchanged packets. Similarly, the $\mathcal{A}dv$ does not gain an advantage by simply dropping or replaying messages, since these actions would at most simply reduce the channel capacity (as would the ability of the $\mathcal{A}dv$ to disrupt network connectivity). Finally, the proposed framework does not prevent the $\mathcal{A}dv$ analyzing the traffic patterns and does not prevent it from fingerprinting the components of the deployment, making it vulnerable to rule scanning and denial of service attacks. While the goals of TruSDN did not include this, such traffic analysis could be prevented using anti-fingerprinting techniques, as proposed in [3].

On the platform level, the security of the proposed framework relies to a large extent on the security properties of Intel SGX enclaves. This allows to protect the execution of switches and network edge components deployed in TEEs from the capabilities of an $\mathcal{A}dv$ controlling non-processor hardware, the software stack of the OS and the hypervisor. Similarly, pausing execution of switches executing in TEEs, while possible, would have no further effect than degrading network connectivity, already discussed above. While the $\mathcal{A}dv$ may attempt to deploy own arbitrary components on the data plane or the network edge in order to launch Sybill attacks, the integrity of such components would not be successfully attested, unless they are identical to legitimate components, which are assumed to be executing correctly – rendering Sybill behavior impossible. The $\mathcal{A}dv$ is prevented from launching cuckoo attacks by enabling tenants to verify the platforms, as described in Section 4.4. As presented in Table 1, several relevant classes of attacks are not addressed by TruSDN, but have known mitigations, namely cache-collision, controlled channel and attacks on shielded execution (addressed in [39, 42]). The capability of the $\mathcal{A}dv$ to return arbitrary values to system calls, while not addressed in this work, can be mitigated by a validation component as described in [2].

6. Implementation and Evaluation

We now describe the implementation and evaluation of TruSDN.

6.1. TruSDN Implementation

The TruSDN prototype deployment follows the design presented in Section 4 and is illustrated in Figure 8. *Host 1* and *Host 2* are instances of Ubuntu OS 15.04. In each instance, we deployed Linux Containers[1],

similarly based on Ubuntu OS 15.04. Containers create an environment with own process and network space, implemented using *namespaces*, with a distinct user ID, network stack, mount points, file systems, processes, inter-process communication, and hostname. We chose containers to facilitate prototype implementation, using their lightweight process isolation. Containers are part of the untrusted OS and this implementation choice is orthogonal to the security of TruSDN. Compute tasks are deployed in TEEs created using SGX enclaves (Figure 8): *enclaves* E1, E2, E4, E5 are placed respectively within containers C1, C2, C3, C4. The switches are deployed in TEEs created using SGX enclaves (enclaves E3, E6 in Figure 8).

Considering that platforms with hardware and software support for SGX were not publicly available at the time of writing, we used *OpenSGX* [19] to emulate the TEEs. It is a software SGX emulator and a platform for SGX development, implemented using binary translation of QEMU and emulating Intel SGX hardware components at instruction level. It includes emulated hardware and OS components, enclave program loader, the OpenSGX user libraries, debugging and performance monitoring support. The emulator allows to implement, debug, and evaluate SGX applications, but does not support binary compatibility with Intel SGX. Furthermore, OpenSGX does not implement all instructions, e.g. debugging instructions. While OpenSGX does not provide security gurantees, it allows us to obtain performance estimates for the proposed approach. We used $mbedTLS$[2] v1.3.11 (distributed with the emulator) for attestation of the SGX enclaves. We used OpenSSL v1.0.2d (distributed with the emulator) to set up protected communication channels between the CT enclaves and the local switches, and among switches within the same domain.

An SDN network controller is deployed in a third instance (*Host 3*). We used the Ryu[3] SDN framework, due to its flexibility and versatile APIs.

6.2. TruSDN Evaluation

We now analyze the performance impact, present evaluation results and discuss aspects that cannot be measured with the current prototype.

Sources of Performance Impact. TruSDN introduces several potential sources of performance impact (Table 4). We distinguish between *transient* performance overhead, which occurs occasionally (e.g. TLS key negotiation) and *continuous* performance overhead, present throughout the infrastructure operation. We *do not* consider the one-time cost of infrastructure deployment, e.g.

---

[1]Linux Containers Project Website: `https://linuxcontainers.org/`

[2]mbed TLS project website `https://tls.mbed.org/`
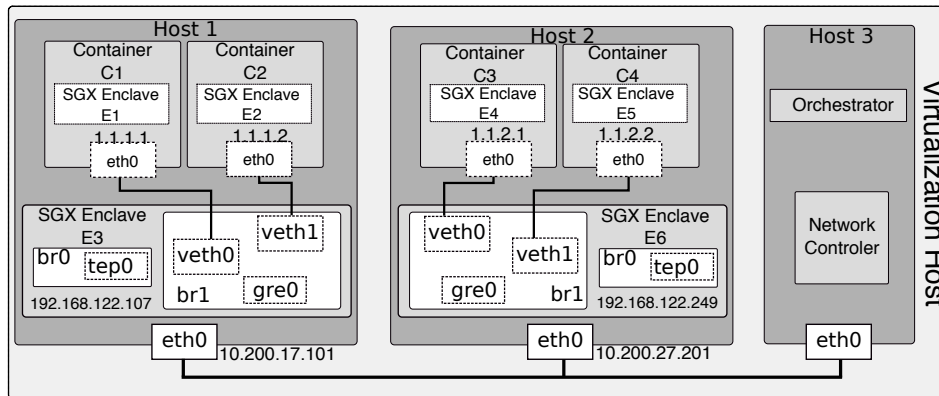[3]Ryu SDN framework: `https://osrg.github.io/ryu/`

Figure 8. Prototype deployment of TruSDN

provisioning the software, attesting TEEs and enrolling the components.

Measured Performance Impact. To evaluate the performance impact, we measured the footprint of establishing TLS sessions on α and γ segments. We used *iperf*, *openssl s_time* and an own Ryu application (Table 5).

TLS overhead on the α segment: We measured the round-trip latency of packets sent in plaintext and with TLS, over 1000 tests, each request sending messages of 100 bytes with the 80 bit OpenFlow header. Furthermore, we measured the data transfer rates for plaintext and TLS communication. Use of TLS increased total transfer time by 14.2% and reduced the transfer rate by 15.98%.

Figure 9 shows the round-trip latency of packets sent in plaintext and with TLS enabled, over 100 tests, each request sending 2 kB messages; furthermore, figure 10 compares the data transfer rates for plaintext and TLS communication in the test bed.

Figures 11 and 12 show respectively the round-trip latency and transfer rates for 10 MB messages. The impact of TLS is notably smaller, mostly due to the proportionally lower impact of the costly TLS key negotiation operations.

Delay on γ segment As mentioned above, the first packet of the flow is intercepted by the switch and forwarded to the NC in a `packet_in` message [7]. At this point the NC processes the flow and installs a flow rule on the switch. TruSDN *extends* this procedure by generating and distributing to the communicating CTs a pre-shared key, to be used for communication protection. Since this must be done prior to both forwarding the message to the destination CT and installing the flow rule, generating and distributing the PSK would normally delay the installation of the flow rule and increase the latency of the *first* packet (all subsequent

packets are forwarded according to the flow rule). To measure the introduced delay, we have sequentially established 1000 TLS sessions between compute tasks C1 and C2 (according to Figure 8). After each TLS session, we flushed the installed flow rules (with `ovs-ofctl del-flows br0`), which resulted in a `packet_in` message upon each new session. The latency of the first packet is shown in Figure 13, and compared against the latency of a first packet without the TruSDN extension.

The induced delay is primarily caused by two operations performed by the NC: *generating* a 256-bit PSK and distributing it to the CTs. Figure 14 displays a fine-grained picture of the induced delay. Key generation lasted on average 0.178 ms, while key distribution on average 0.54 ms (Table 5). We remind that the test environment is fully virtualized and posit that overhead of key generation can be reduced in a production environment, either by using pre-generated keys or with specialized hardware (e.g. crypto processors). In our tests, the duration of establishing a TLS session with ephemeral flow-specific pre-shared keys using the PSK-AES256-CBC-SHA cipher suite was 2.41% *less* compared to the use of e.g. ECDH-RSA-AES128-SHA256. Thus, TruSDN enables flexible use of pre-shared keys, which in turn reduces the duration of the TLS handshake, by avoiding expensive public key cryptographic operations [21]. Moreover, it reduces the CPU utilization for key derivation in CTs, at the cost of a minimal flow rule installation delay. The above approach may be applicable to other protocols. For example, none of the differences between the datagram TLS (DTLS) and TLS protocols specified in [35] indicate that the above approach is incompatible with DTLS. We leave further investigation for future work.

Unmeasured Performance Overhead. Implementing TEEs with OpenSGX limits the level of detail when it comes to performance evaluation, since: **(a)** the OpenSGX emulator *is not* binary compatible with

Table 4. Sources and types of performance overhead in TruSDN

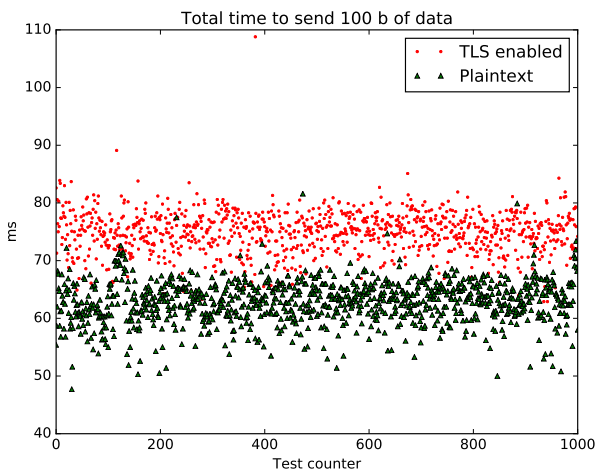| Source	Type	Clarification
TLS negotiation all segments	transient	Negotiate session keys for TLS
PSK distribution	transient	Distribute PSK for γ segments
TLS protection all segments	continuous	Overhead induced by TLS
Compute task execution in TEEs	continuous	Overhead induced by TEE
Switch execution in TEEs	continuous	Overhead induced by TEE

Figure 9. Throughput, fine-grained view.

Figure 10. Data transfer, fine-grained view
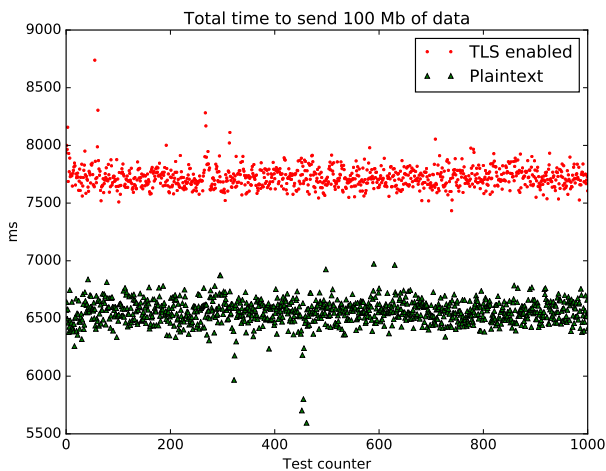

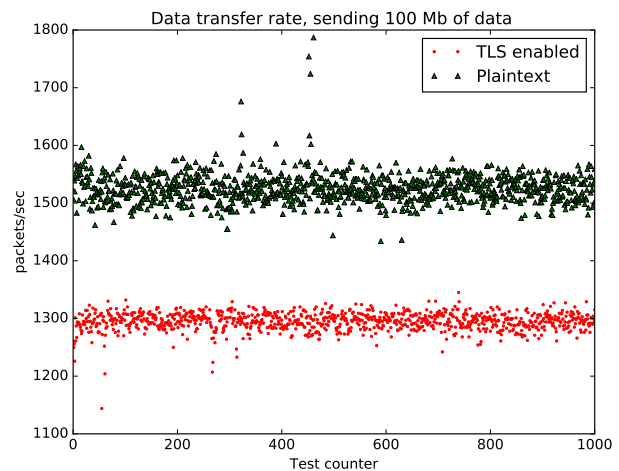**Figure 11.** Throughput, coarse-grained view.

Figure 12. Data transfer, coarse-grained view.

Intel SGX [19]; **(b)** in its current version[4] and unlike Intel's description of SGX [1], OpenSGX has yet to implement support multithreaded applications[5]. Thus, a fully accurate measurement on TruSDN performance cannot be done until Intel SGX hardware and software

is made available. However, we believe our experiments yield a fair picture of the expected performance impact.
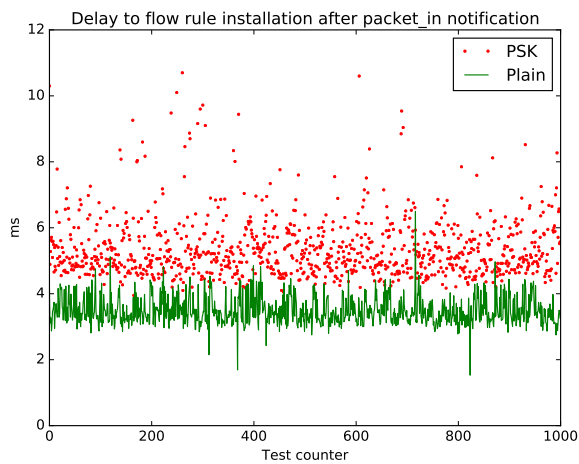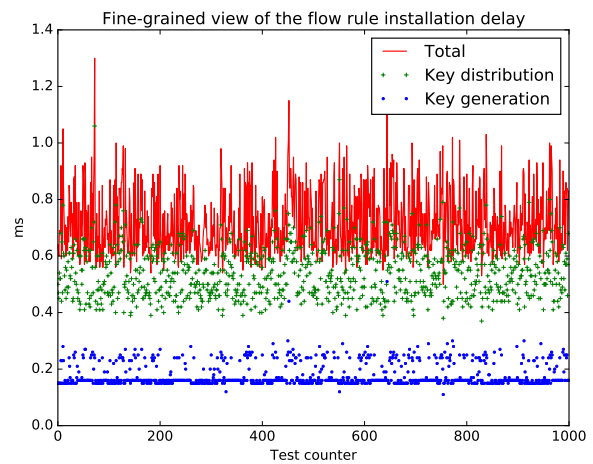
7. Related work

In this section we present related work on adversary models for SDN deployments, SDN controller security, and principles and software model of SGX.

---

[4]Commit e0713c7 on `https://github.com/sslab-gatech/opensgx`

[5]Issue #34 on `https://github.com/sslab-gatech/opensgx/issues/34`

Table 5. Summary of performance evaluation of TruSDN

Data	Minimum	Maximum	Mean	Median	Stddev
Total transfer time, ms	0.4	1.1	0.66	0.7	0.07
Total transfer time w. TruSDN, ms	0.5	7.1	0.8	0.8	0.22
TruSDN overhead, **total transfer time**			**21.2%**	**14.2%**	
Transfer rate, bytes per second	1225	2095	1595	1583	98.07
Transfer rate w. TruSDN, bytes per second	919	1589	1338	1330	64.86
TruSDN overhead, **transfer rate**			**16.11%**	**15.98%**	
First packet latency γ	1.53	6.50	3.48	3.38	0.42
First packet latency γ w. TruSDN	3.35	10.7	5.37	5.14	0.93
TruSDN overhead, **first packet latency**			**54.31%**	**52.07%**	
TLS handshake, ms	36.53	77.72	67.97	67.48	7.42
TLS handshake w. TruSDN, ms	52.35	76.44	67.15	66.53	3.93
TruSDN overhead, **TLS handshake**			**-2.21%**	**-2.41%**	
Key generation NC, ms	0.11	0.51	0.178	0.16	0.04
Key distribution γ, ms	0.37	1.06	0.54	0.53	0.08
Key total γ, ms	0.50	1.30	0.71	0.7	0.11

Figure 13. Coarse-grained view

Figure 14. Fine-grained view

7.1. Adversary models for SDN deployments

Despite the growing number of publications focusing on secure SDN architectures [17, 20, 26, 32, 34, 40], none of them directly addresses trust establishment in SDN deployments. Here, we review the related work on secure SDN architectures and provide a context for the work presented in this paper.

Kreutz et al. presented a list of seven attack vectors identified in SDN [20]: **a.** Forged or faked traffic flows; **b.** Attacks on vulnerabilities in switches; **c.** Attacks on control plane communications; **d.** Attacks on and vulnerabilities in network controllers; **e.** Lack of mechanisms to ensure trust between the controller and management applications; **f.** Attacks on and vulnerabilities in administrative stations; **g.** Lack of trusted resources for forensics and remediation. However, only part of the attack vectors described in [20] are exclusively relevant to SDN networks and no specific solutions are proposed. In this paper we focus on several attack vectors specific for SDN-based multi-tenant virtualized network architectures, namely vectors

c., e., g. of the above list. Furthermore, we describe in detail and implement a protocol that addresses the above issues.

Work in [26] introduced an adversary model, along with potential attack vectors and security requirements towards multi-tenant SDN infrastructure. The authors highlighted the need for security mechanisms on several layers of the SDN architectural model, in order to: **a.** Limit the effect of controller vulnerabilities on tenant domains; **b.** Authenticate the communication between the components of the SDN infrastructure; **c.** Enforce policy isolation between tenants; **d.** Verify integrity of network components prior to enrollment; **e.** Enforce quota isolation between tenants. In this work, we address points **a., b., d.** of the above list and contribute towards a comprehensive security framework for multi-tenant SDN deployments.

7.2. Secure SDN controllers

NOX is a network operating system (NOS) introduced in [16]. It presents network management applications

with a *centralized* programming model and a global view of the system state, allowing such applications to rely on simpler graph processing algorithms to compute the shortest paths and to operate with higher-level abstractions – e.g. users and host names – rather than MAC and IP addresses. NOX consists of several controller processes operating on a global network view built based on the process communication with the data plane. The global view is based on switch topology, user location, connected network components (e.g. hosts and middleboxes), as well as bindings between the names and addresses. Controller processes use the global view to make network management decisions and update the switch FIB over the OpenFlow API [23].

In [32], the authors presented FortNOX, a software extension for role-based authorization and security constraints enforcement for NOX. FortNOX detects rule conflicts, i.e. situations when candidate OpenFlow rules modify network flows specified by existing rules, and takes appropriate actions, depending on the authorization of the rule requestor. Role-based authentication is used to determine the security authorization of each rule producer, enforcing the principle of least privilege to ensure integrity of the mediation process. FortNOX consists of four components: a role-based source authentication module to validate signatures for each flow rule insertion request; a conflict analyzer to evaluate each new flow rule against existing active flow rules; a state table manager to track the current flow rules; and finally a flow rule time-out callback interface to update the aggregate flow table upon rule expiration. FortNOX includes a security directive translator for the *block, deny, allow, redirect, quarantine, undo, constrain,* and *info* directives; such directives are used for high-level threat mitigation which are in turn translated to flow rules to handle suspicious traffic. Neither NOX nor FortNOX address the problem of malicious network components, which can distort the global network view used by the NC. This can be addressed by extending the above mentioned network controllers with TruSDN.

"Rosemary" [40] is a NOS that improves network resilience in the presence of faulty or malicious applications by creating sandboxed environments for network management applications. Sandboxing (also called micro-NOS) is done by launching each application in a separate process context with access to all of the libraries that the application requires. Each micro-NOS also contains a resource monitor to supervise the applications and operates within the permission structure of Rosemary NOS. In turn, the 'Rosemary' NOS is an application running on a commodity Linux distribution. The isolation offered by the micro-NOS allows to improve robustness and prevent faulty or malicious applications from crashing the entire NOS. Furthermore, "Rosemary" prevents malicious components from accessing internal data structures of

other network management applications by interposing and verifying privileged system calls made by network applications. Despite its advantages, "Rosemary" does not address integrity and authentication of communication on the data plane. TruSDN complements this and creates a strong foundation for trusted deployment of a secure NOS.

In [17], the authors have addressed several attacks on network visibility in SDN and introduced a security extension to SDN controllers, aimed to detect network topology poisoning attacks. While the paper provides several SDN-specific infrastructure attack examples and highlights the urgency of addressing them, the suggested solution assumes a network deployment with exclusively physical switches and does not accommodate for the rapidly increasing number of *software*, virtualized switches. TruSDN complements this approach, is applicable to virtualized and multi-tenant network deployments and uses a hardware root of trust in order to ensure the integrity of virtual switches.

7.3. Software Guard Extensions

SGX was introduced in [22], with a description of the SGX software model, as well as of the extensions to the x86 ISA and the necessary hardware modifications for isolated execution in the SGX model; this was complemented by CPU based attestation for the SGX execution model, described in [1]; in [22] the authors describe several prototypes that use SGX to create trustworthy applications – namely a secure one-time-password generator, applications for enterprise rights management and secure video conferencing.

The work in [2, 39] describes application of SGX-based solutions in a cloud setting. In "Haven" [2] the authors describe the experience of porting a modified version of Windows 8 operating system to an SGX enclave and evaluate it with several unmodified applications – namely an Apache Web Server and an SQL Server, using synthetic data sets. In addition, the authors describe a mechanism to protect the enclave execution environment from the attacks of a malicious kernel, as well as a semantically secure data store protecting the confidentiality of data and file metadata against a malicious host. TruSDN protects confidentiality and integrity of network communication for an adversary model similar to [2]. While we deploy compute tasks in SGX enclave-based TEEs, the work in [2] is largely complementary, and similar "Haven"-like OSs could be used.

"VC3" [39] describes the deployment of a Map-Reduce framework in a cloud environment using SGX enclaves. Users implement *map* and *reduce* functions and compile them into private (encrypted) code and a generic public code that implements the key exchange and job execution protocols. Both private and public

code is initialized in enclaves and later attested by the users. After attestation, enclaves containing the public code perform the key exchange protocol and obtain the key to decrypt the private enclave code and process the distributed job execution protocol. Furthermore, to defend against cuckoo attacks, the authors propose that *cloud quoting enclaves* are created on each platform added in the cloud provider data centers. The cloud computing enclaves would then "countersign" the quotes produced by the quoting enclave. The approach described in [39] used to protect compute tasks for map-reduce jobs using SGX enclaves is largely complementary to protecting the communication between compute tasks with TruSDN. However, the defense against cuckoo attacks described in [39] both increases the complexity of the attestation protocol and does not prevent the adversary exploiting a compromised cloud quoting enclave outside of the physically secure datacenter perimeter. Instead, the approach described in Section 5 leverages the cryptographic properties of EPID signature scheme used by SGX enclaves, without complicating the attestation protocol.

8. Future Work

In this section we discuss the limitations of the presented solutions and outline several directions for future work.

Addressing the limitations of TruSDN Along with its advantages, particularly when it comes to providing security guarantees, TruSDN has a series of limitations, which stem in particular from its reliance on the SGX technology. While it is too early to speak of the performance overhead of the solution – given that SGX-enabled platforms are not yet widely available – we focus on the security limitations of TruSDN. In particular, the authors of [42] introduce *controlled-channel attacks*, a new type of side-channel attack that allows an untrusted operating system to extract large amounts of sensitive information from protected applications on systems. This side-channel attack has been successfully applied to Haven [2] and TruSDN in its current form could also be vulnerable to such side-channel attacks (note that we have explicitly excluded side-channel attacks from the adversary model in Section 3). According to [42], such attacks could be mitigated rewriting application such that its memory access pattern does not depend on sensitive data, prohibiting paging by the underlying OS or using techniques to obfuscate the memory access patterns.

Another limitation stems from the reliance on the platform vendor: by leaking the signing key of the Quoting Enclave, the platform vendor could create a "deniable back-door" allowing *person-in-the-middle* attacks during remote attestation [38]. This challenge remains unaddressed at this point.

Enabling Practical Deployments In future work we aim to modify a popular software switch implementation, such as Open vSwitch[6], in order to confine its security-sensitive components to an SGX enclave [27]. Placing only such security-critical components into a TEE can help reduce the performance overhead of the solution, improve its portability and facilitate its adoption.

Integrating storage protection mechanisms Related work on cloud storage protection [11, 15, 25] and secure virtual environment deployment [28] can be integrated with TruSDN in order to provide cloud infrastructure compute, storage and communication resources that are secure in the given adversarial model.

Applying TruSDN **to virtual network functions (VNFs)** A prototype applying the TruSDN approach to VNFs in an SDN deployment was described in [29]. Future challenges include designing an integrated prototype with component integrity attestation on the data plane, controller plane and management plane as well as leveraging TruSDN to secure communications throughout the deployment.

9. Conclusion

We described, implemented and evaluated TruSDN – a framework for bootstrapping trust in SDN infrastructure. It isolates network endpoints and switches in SGX enclaves, remotely attests their integrity, and establishes secure communication channels. We leveraged the principles of SDN to introduce *ephemeral flow-specific PSK* distributed at flow creation, which reduce the overhead of key derivation and reduce the total time to establish protected channels, at the cost of a minor delay in the flow rule installation. Finally, we leveraged the properties of the EPID scheme to propose an improved approach to prevent cuckoo attacks.

10. Acknowledgements

References

[1] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for CPU based attestation and sealing. In *Proc. 2nd International Workshop on Hardware and Architectural Support for*

---

[6]Open vSwitch Project website: `http://openvswitch.org/`

Security and Privacy, HASP '13, page 10. ACM, June 2013.

[2] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Transactions on Computer Systems*, 33:8:1–8:26, August 2015.

[3] R. Bifulco, H. Cui, G. O. Karame, and F. Klaedtke. Fingerprinting Software-Defined Networks. In *Proc. 23rd International Conference on Network Protocols*, ICNP '15, pages 453–459. IEEE, November 2015.

[4] E. Brickell and Jiangtao Li. Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities. *IEEE Transactions on Dependable and Secure Computing*, 9:345–360, May 2012.

[5] Martin Casado, Nate Foster, and Arjun Guha. Abstractions for Software-defined Networks. *Communications of the ACM*, 57:86–95, September 2014.

[6] Stephen Checkoway and Hovav Shacham. Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface. In *Proc. 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 253–264. ACM, March 2013.

[7] OpenFlow Switch Consortium. OpenFlow switch specification, version 1.3.5. Technical Report ONF TS-023, Open Networking Foundation, March 2015.

[8] Allan Cook, Michael Robinson, Mohamed Amine Ferrag, Leandros A. Maglaras, Ying He, Kevin Jones, and Helge Janicke. Internet of Cloud: Security and Privacy Issues. Technical Report arXiv:1711.00525, arXiv, November 2017.

[9] Danny Dolev and Andrew C Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29:198–208, March 1983.

[10] John R. Douceur. The Sybil Attack. In *Proc. 1st International Workshop on Peer-to-Peer Systems*, IPTPS '02, pages 251–260. Springer, March 2002.

[11] Rafael Dowsley, Antonis Michalas, Matthias Nagel, and Nicolae Paladi. A survey on design and implementation of protected searchable data in the cloud. *Computer Science Review*, In press, August 2017.

[12] Pasi Eronen and Hannes Tschofenig. Pre-shared key ciphersuites for transport layer security (TLS). RFC 4279, RFC Editor, December 2005.

[13] Dino Farinacci, P Traina, Stan Hanks, and T Li. Generic routing encapsulation (GRE). RFC 2992, RFC Editor, March 2000.

[14] Mohamed Amine Ferrag, Leandros A. Maglaras, Helge Janicke, Jiamin Jiang, and Lei Shu. Authentication Protocols for Internet of Things: A Comprehensive Survey. *Security and Communication Networks*, 2017:41, 2017.

[15] C. Gehrmann, F. Morenius, and N. Paladi. Procedure For Platform Enforced Storage in Infrastructure Clouds, March 2016. Patent Application WO2014185845 A1.

[16] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38:105–110, July 2008.

[17] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures. In *Proc. 2015 Network and Distributed System Security Symposium*, NDSS '15. Internet Society, February 2015.

[18] C Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, RFC Editor, November 2000.

[19] Prerit Jain, Soham Desai, Seongmin Kim, Ming-Wei Shih, JaeHyuk Lee, Changho Choi, Youjung Shin, Taesoo Kim, Brent Byunghoon Kang, and Dongsu Han. OpenSGX: An Open Platform for SGX Research. In *Proc. 2016 Network and Distributed System Security Symposium*, NDSS '16. Internet Society, February 2016.

[20] Diego Kreutz, Fernando Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *Proc. 2nd ACM SIGCOMM workshop on Hot topics in software defined networking*, HotSDN '13, pages 55–60. ACM, August 2013.

[21] Fang-Chun Kuo, Hannes Tschofenig, Fabian Meyer, and Xiaoming Fu. Comparison studies between pre-shared and public key exchange mechanisms for transport layer security. In *Proc. 25th IEEE International Conference on Computer Communications*, INFOCOM '06, pages 1–6. IEEE, April 2006.

[22] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *Proc. 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 10:1–10:1. ACM, June 2013.

[23] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38:69–74, April 2008.

[24] Thomas D. Nadeau and Ken Gray. *SDN: Software Defined Networks*. O'Reilly Media, Inc., Sebastopol, California, USA, 2013.

[25] N. Paladi, C. Gehrmann, and A. Michalas. Providing User Security Guarantees in Public Infrastructure Clouds. *IEEE Transactions on Cloud Computing*, 5:405–419, July 2017.

[26] Nicolae Paladi. Towards Secure SDN Policy Management. In *Proc. 8th International Conference on Utility and Cloud Computing*, UCC '15, pages 607–611, December 2015.

[27] Nicolae Paladi. *Trust but Verify: Trust Establishment Mechanisms in Infrastructure Clouds*. PhD thesis, Lund University, September 2017.

[28] Nicolae Paladi, Christian Gehrmann, Mudassar Aslam, and Fredric Morenius. Trusted Launch of Virtual Machine Instances in Public IaaS Environments. In *Proc. 15th International Conference on Information Security and Cryptology*, ICISC' 12, pages 309–323. Springer, December 2012.

[29] Nicolae Paladi and Linus Karlsson. Safeguarding VNF Credentials with Intel SGX. In *Proceedings of the SIGCOMM Posters and Demos*, SIGCOMM Posters and Demos '17, pages 144–146. ACM, August 2017.

[30] Bryan Parno. Bootstrapping Trust in a "Trusted" Platform. In *Proc. 3rd Conference on Hot Topics in Security*, HOTSEC'08, pages 9:1–9:6. USENIX, July 2008.

[31] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The Design and Implementation of Open vSwitch. In *Proc. 12th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '15, pages 117–130. USENIX, May 2015.

[32] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for OpenFlow networks. In *Proc. 1st Workshop on Hot topics in software defined networks*, HotSDN 12, pages 121–126. ACM, August 2012.

[33] Phillip Porras, Steven Cheung, Martin Fong, Keith Skinner, and Vinod Yegneswaran. Securing the software-defined network control layer. In *Proc. 2015 Network and Distributed System Security Symposium*, NDSS '15. Internet Society, February 2015.

[34] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proc. ACM Conference of the Special Interest Group on Data Communication*, SIGCOMM '13, pages 27–38, New York, NY, USA, 2013. ACM.

[35] Eric Rescorla and Nagendra Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, RFC Editor, January 2012.

[36] Thomas Ristenpart and Scott Yilek. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In *Proc. 2010 Network and Distributed System Security Symposium*, NDSS '10. Internet Society, February 2010.

[37] Xiaoyu Ruan. *Safeguarding the Future of Computing with Intel Embedded Security and Management Engine*. Apress, Berkely, CA, USA, 1st edition, 2014.

[38] Joanna Rutkowska. Thoughts on Intel's upcoming Software Guard Extensions (Part 2), 2013. [Online; March 2016].

[39] Felix Schuster, Manuel Costa, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-ruiz, and Mark Russinovich. VC3 : Trustworthy Data Analytics in the Cloud using SGX. In *Proc. 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 38–54. IEEE, May 2015.

[40] Seungwon Shin, Yongjoo Song, Taekyung Lee, Sangho Lee, Jaewoong Chung, Phillip Porras, Vinod Yegneswaran, Jiseong Noh, and Brent Byunghoon Kang. Rosemary: A Robust, Secure, and High-Performance Network Operating System. In *Proc. 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 78–89. ACM, November 2014.

[41] Jesse Walker and Jiangtao Li. Key exchange with anonymous authentication using DAA-SIGMA protocol. In *Proc. 2nd International Conference on Trusted Systems*, INTRUST '11, pages 108–127. Springer, December 2011.

[42] Y. Xu, W. Cui, and M. Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *Proc. 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 640–656. IEEE, May 2015.