A Secure Computation Framework for SDNs

Nachikethas A. Jagadeesan
USC

Ranjan Pal
USC

Kaushik Nadikuditi
USC

Yan Huang
UMD, College Park

Elaine Shi
UMD, College Park

Minlan Yu
USC

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Centralized Networks

Keywords

Security; Software-Defined Networking

1. INTRODUCTION

Software Defined Networking (SDN) introduces a logically centralized control plane to run diverse management applications. In practice, a logically centralized control plane is realized using multiple controllers for scalability, reliability, and availability reasons. In fact, for various current and future networks of interest, it is practically infeasible to attempt a physically centralized SDN system. As SDN gains popularity, it is important to secure the SDN infrastructure to be resilient to potential attacks.

In SDN, controllers can become high-value and attractive targets for an adversary for the following reasons. First, controllers are sinks of information collected from different switches. This includes network topology and flow-counter values. Such information can be privacy sensitive. For example, an organization may wish to protect its internal network topology or hide what type of traffic is being routed through its network. In addition, privacy policies may prohibit information from flowing between one part of the organizational network to another. Second, controllers run full-fledged software stacks including an operating system and management applications. Therefore, they may expose a much larger attack surface than switches. Moreover, threats may arise from multiple sources. In addition to software vulnerabilities that may exist in the controller software stack, malicious insiders who have privileged access to the controllers may leak sensitive information or sabotage network operations. For example, the network operator wants to make sure that traffic flow counters in the controllers stay untouched by an adversary. Manipulation of these counters could allow DDoS attacks on hosts go undetected. As another example, the network operator wants to protect network topology information from adversaries in order to protect against DDoS attacks.

We borrow techniques from Multi-Party Computation (SMPC) [3] in cryptography to provide provable security guarantees against such threats. Given recent advances in making SMPC faster [2], we find it suitable to use multi-party computation to achieve the following security goals in SDNs: (i) When a subset of the controllers are compromised, no sensitive information such as network topologies are leaked (ii) The network's resilience to controller failure is improved. More formally, an operator should support guarantees on fault tolerant computation whereby for a threshold k, even if $k-1$ controllers become completely non-functional, the remaining controllers can execute the computation correctly.

2. SMPC FRAMEWORK

2.1 Background

Imagine a situation in which a group of mutually distrustful individuals are required to cooperate in order to achieve a common objective. Specifically, consider the case in which each individual possesses some private data and the group collectively attempts to

1. Compute some function that may depend on the private inputs of all the participating parties

2. Avoid revealing any individual data other that what may be deduced from the outcome of the computation.

SMPC is a field of cryptography that deals with the development of efficient protocols and algorithms to achieve these objectives.

Let us make this notion more precise. We have n parties, $C_1, \ldots, C_n$, each of whom possesses a private input. Let the input of C_i be denoted by x_i. *Our goal is to compute* $y = f(x_1, \ldots, x_n)$ *such that* C_i *learns only* y *and is ignorant about* $\{x_j \mid j \neq i\}$. It is known that there exists protocols that allow us to compute any arbitrary function f in this manner, provided we assume that not more than a certain fraction of the parties collude. Note that each party has an equivalent part to play in computing y. In other words, we discard the solution where the input set $\{x_i \mid i = 1, \ldots, n\}$ is sent to a trusted third party that computes y and sends it to $\{C_1, \ldots, C_n\}$.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author. Copyright is held by the owner/author(s).

HotSDN'14, August 21 - August 22 2014, Chicago, Illinois USA
ACM 978-1-4503-2989-7/14/08.
http://dx.doi.org/10.1145/2620728.2620768.

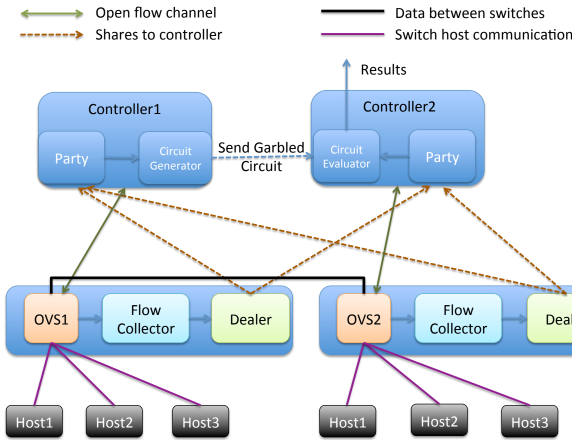

Figure 1: SMPC in SDNs

2.2 SMPC Design in SDN

The architecture of SDNs lends itself naturally to secure computation. The data plane switches assume the role of input providers to the controllers, where the secure computation is carried out. In our model, each input is divided into different parts (or *shares*) using an appropriate secret-sharing scheme. This division of inputs to different shares is done at the switches. As shown in figure 1, each switch has a component named *dealer* that collects the required input values and prepares them to be split across multiple controllers. Each share is given to a different controller. Similarly, each controller has a component named *party* that collects such data from the switches and engages in SMPC with other controllers. For example, consider the simple case where we have two controllers. One way of performing secure computation over them is by using Yao's garbled circuits [3]. In this model, a controller (say, 1) has a generator module that creates a boolean circuit that encodes his input and sends the same to controller 2. Controller 2 evaluates this boolean circuit using his input, thereby generating the result of our chosen computation, which is then published. The efficiency of this approach depends on our data and the function that we are trying to compute. We base the implementation of our algorithms in an efficient framework for running garbled circuits known as *fastGC* [2].

We now describe the challenges involved in enabling secure computation in SDNs. Firstly, any generally applicable SMPC framework for SDNs must address the issue of deciding whether an application is admissible for computation. SMPC only guarantees the *execution* of any given application in a secure manner and is conspicuously non-committal about what may be released by the results. Consider the trivial example of an identity function which simply publishes its inputs. The input, say topology information, to any application that realizes this design is inherently insecure. While the above may be an extreme example, it highlights the need for analysis that aids in deciding the admissibility of a function. In this paper, we *assume* the validity of our chosen applications.

Another challenge pertains to the handling of network state. Controllers are often heterogeneous in nature. We may have a cluster of servers that we trust more. Some of them may have more computational power than the others. Consequently, we may desire to pre-assign the division of state amongst controllers so that we extract the maximum efficiency. Details on how one might go about doing this are a subject for future discussion. For the purposes of this paper, we make the simplifying *assumption* that all state is handled equally by the controllers.

Finally, we have the constraints imposed by performance requirements. The benefits of secure computation are purchased by sacrificing some performance. The key question here is how much improvement can we make in this trade off between security and performance? SMPC algorithms that can handle a large class of functions tend to trade-off more performance to achieve their generality. It follows that applications that are of sufficient importance to our network or those that have strict latency requirements may warrant the development of custom algorithms that exploit the structure inherent in the problem to achieve the desired performance. Keeping that in mind, we posit that SMPC is generally better suited for applications that allow for offline processing of data as opposed to those that demand results in real time.

3. CASE STUDY: HH DETECTION

In this section, we describe our experimental setup and analyse the results obtained on implementing a prototype application, namely, heavy hitter (HH) detection. We define heavy hitters as the top k IP addresses that sent the maximum number of packets, for each window of a certain size (say t). One challenge we needed to overcome is to design the algorithm so as to ensure that each operation is *data oblivious*. In other words, we need to ensure that the memory access patterns of our algorithm does not reveal anything regarding the input data.

The algorithm has two main parts. Firstly, flow table entries from different switches need to be merged at the controllers. Secondly, the merged list is sorted to identify the heavy hitters. To ensure the data-oblivious property, both parts are realized using an oblivious sorting algorithm, namely, randomized shell sort [1]. Specifically, the flow data from all switches are concatenated to form a list that is sorted according their IP. The entries with the same IP are now clustered together which allows us to merge all of them in a single pass. This is followed by sorting the data by their flow table entries, which identifies the heavy hitters.

The above algorithm was implemented on the fastGC framework installed on a workstation running an Intel i7-3770 processor. The flow table entries which form the input to the controllers were taken from the CAIDA data set. For 4096 flow table entries, the runtime of our application was around 13.3 minutes. We plan to further improve the runtime with optimizations such as updates to the fastGC framework.

4. REFERENCES

[1] Michael T. Goodrich. Randomized shellsort: A simple data-oblivious sorting algorithm. *J. ACM*, 58(6):27:1–27:26, December 2011.

[2] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

[3] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science, 1986*, pages 162–167, October 1986.