AFRL-AFOSR-VA-TR-2019-0359

Writing and Securing Peer-to-Peer Computation

Stephen Chong
HARVARD COLLEGE PRESIDENT & FELLOWS OF

11/15/2019
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
22-11-2019	Final Performance	15 Aug 2016 to 14 Aug 2019

4. TITLE AND SUBTITLE
Writing and Securing Peer-to-Peer Computation

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-16-1-0351

5c. PROGRAM ELEMENT NUMBER
61102F

6. AUTHOR(S)
Stephen Chong

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
HARVARD COLLEGE PRESIDENT & FELLOWS OF
1350 MASS AVE STE 600
CAMBRIDGE, MA 02138-3846 US

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AF Office of Scientific Research
875 N. Randolph St. Room 3112
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/AFOSR RTA2

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-AFOSR-VA-TR-2019-0359

12. DISTRIBUTION/AVAILABILITY STATEMENT
A DISTRIBUTION UNLIMITED: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This grant aims to make it easier to write and secure peer-to-peer computation: direct sharing of data and code between devices without a trusted server to mediate their interaction. The ability for peers to send data and computation directly to each other enables exciting applications in settings where a central trusted server is not available (due to either connectivity issues or insufficient trust in available servers).
The objectives of this research lie in three primary directions.
1. Efficient enforcement of security and resource usage on code received from an untrusted device.
2. Efficient verification of code executed by an untrusted device.
3. Design and implementation of the Calder programming language, a language for sharing data and computation among heterogeneous resource-constrained devices.
In this third year of the project, we continued the design and implementation of the Calder programming language. We have completed implementation of a simulator for execution of Calder programs and furthered development of Calder's runtime system to dynamically decide where to execute mobile code to optimize resource usage and to enforce resource constraints. Due to the departure of key personnel, we pivoted and spent significant effort this year on the use of trusted execution environments to enable efficient enforcement of security on code between peers. Specifically, we developed the DFLATE calculus, which provides high-level abstractions for using Trusted Execution Environments (such as Intel's SGX enclaves) to enable secure sharing of data and computation. This work was published at the IEEE Computer Security Foundations Symposium, a premier venue for foundational research on computer security.

15. SUBJECT TERMS
Language-based Security, Untrusted Devices, Peer-to-Peer Computation, Verified Computation, Secure Mobile Code, Information Flow

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			NGUYEN, TRISTAN
Unclassified	Unclassified	Unclassified	UU		19b. TELEPHONE NUMBER *(Include area code)* 703-696-7796

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

AFOSR Final Report

Grant Title:	**Writing and Securing Peer-to-Peer Computation**
Grant Number:	**FA9550-16-1-0351**
PI:	**Dr. Stephen Chong**
	chong@seas.harvard.edu
Organization	**Harvard University**
Program Manager	**Dr. Tristan Nguyen**
Dates covered	**August 15 2018–August 14 2019**

Abstract:

This grant aims to make it easier to write and secure *peer-to-peer computation*: direct sharing of data and code between devices without a trusted server to mediate their interaction. The ability for peers to send data and computation directly to each other enables exciting applications in settings where a central trusted server is not available (due to either connectivity issues or insufficient trust in available servers).

The objectives of this research lie in three primary directions.

1. Efficient enforcement of security and resource usage on code received from an untrusted device.
2. Efficient verification of code executed by an untrusted device.
3. Design and implementation of the Calder programming language, a language for sharing data and computation among heterogeneous resource-constrained devices.

In this third year of the project, we continued the design and implementation of the Calder programming language. We have completed implementation of a simulator for execution of Calder programs and furthered development of Calder's runtime system to dynamically decide where to execute mobile code to optimize resource usage and to enforce resource constraints.

Due to the departure of key personnel, we pivoted and spent significant effort this year on the use of trusted execution environments to enable efficient enforcement of security on code between peers. Specifically, we developed the DFLATE calculus, which provides high-level abstractions for using Trusted Execution Environments (such as Intel's SGX enclaves) to enable secure sharing of data and computation. This work was published at the IEEE Computer Security Foundations Symposium, a premier venue for foundational research on computer security.

Participants

Stephen Chong (PI)
Pablo Buiras (Postdoctoral Fellow)
Anitha Gollamudi (Graduate student)

Publication list for report period:

"Information Flow Control for Distributed Trusted Execution Environments" by Anitha Gollamudi, Stephen Chong, and Owen Arden. In *Proceedings of the 32nd IEEE Computer Security Foundations Symposium* (CSF), June 2019.

Introduction

The objective of this grant is to make it easier to write and secure *peer-to-peer computation*: direct sharing of data and code between devices without a trusted server to mediate their interaction. The ability for peers to send data and computation directly to each other enables exciting applications in settings where a central trusted server is not available (due to either connectivity issues or insufficient trust in available servers).

Peer-to-peer computation poses several challenges: How can programmers easily express robust and bug-free computation that can be sent to other devices? How can a device ensure that a received computation is safe to execute, in terms of information security and resource usage? How can a device ensure that a result computed on another device is correct?

There are three research directions:

1. Explore efficient enforcement of security and resource usage on code received from an untrusted device.
2. Explore efficient verification of code executed by an untrusted device.
3. Design and implementation of the Calder programming language, a language for sharing data and computation among heterogeneous resource-constrained devices.

In this third year of the project, we continued the design and implementation of the Calder programming language. However, due to the departure of key personnel, we pivoted and spent significant effort this year on the use of trusted execution environments to enable efficient enforcement of security on code between peers.

In the remainder of the report, we expand on the progress made on the design and implementation of Calder, and on the use of trusted execution environments for peer-to-peer computation.

Calder: Language Design and Simulator Implementation

We are developing Calder, a programming language for sharing data and computation among heterogeneous resource-constrained devices. Calder will support mobile code and resource usage optimization.

The language is a simply-typed lambda calculus extended with support for controlling the usage of two kinds of resources: built-in resources and language-level resources. Built-in resources include time, memory usage and energy consumption, all of which can have explicit bounds specified by the programmer. Language-level resources are declared by the programmer and include references to values with a public interface, e.g. a data structure or a function used to read from a hardware sensor. Moreover, these resources include an explicit annotation that represents whether the resource can be copied to other nodes and the cost of this operation.

In Calder, the programmer can transparently express computations that involve language-level resources in different nodes. The Calder runtime takes care of partitioning the computation in a way that optimizes resource usage, i.e., minimizing global cost by either moving code to where the resources are or vice versa, depending on what is cheaper in each case. This analysis also takes into account the bounds on basic resource usage, and might involve static and/or dynamic enforcement mechanisms.

In the third year of the grant, we completed implementation of a simulator for execution of Calder programs and furthered development of Calder's runtime system.

Calder moves code and data between devices at run time to optimize and enforce resource usage. The optimization relies on type inference to determine the resource usage of code. The optimization can be performed statically (i.e., at compile time) or on-the-fly (i.e., during run time), and thus it is necessary to be able to do type inference at run time. The type system is based on the resource constraint type system of Resource Aware ML (http://raml.co/), but our implementation specializes the constraints for the language features of Calder.

Trusted execution environments for peer-to-peer computation

Trusted Execution Environments (TEEs)— such as Intel's SGX enclaves—provide hardware mechanisms to protect code and data from other code within the same system, including privileged code such as the system's kernel. In addition, TEEs are able to provide remote attestation: providing a cryptographic proof to a remote party of the contents of a TEE, and thus a proof of what code is executing on the system. Although current TEE implementations suffer from security vulnerabilities, the fundamental ideas of TEEs enable exciting use cases, such as clients able to execute code on an untrusted cloud, and—once convinced that the correct code is executing in the protected environment of a TEE—sending sensitive data to the TEE without the untrusted cloud being able to access the sensitive data. TEEs can also enable trustworthy mobile computation between peers, allowing Peer A to send code to Peer B and receive a guarantee that Peer B is executing the code that was sent, without modification.

However, although TEEs can provide strong protection against powerful attackers, TEEs by themselves do not guarantee security. For example, code executing in a TEE could inappropriately release confidential information, or inappropriately rely on untrusted data. Or vulnerabilities in code executing in a TEE could be exploited to allow an attacker to gain complete control of a TEE.

We have developed a calculus DFLATE that provides high-level abstractions that reflect both the guarantees and limitations of the underlying security mechanism of TEEs in a distributed setting. This makes it easier for programmers to write distributed programs with strong information security guarantees: programmers can address the application-level security requirements and DFLATE will ensure that the program uses TEEs effectively and correctly to enforce the security requirements, including supporting mobile code in the form of function closures.

We have designed the DFLATE calculus and proven that it provides strong application-level security guarantees. That is, peers are able to send computations and data to each other, including to untrusted peers, and receive assurances about the security of computations and data.

The work on DFLATE appeared at the 32nd IEEE Computer Security Foundations Symposium, a premier venue for foundational research on computer security.