A Secure Message Broker in an Untrusted Environment

Dylan Dreyer

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-91
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-91.html

May 21, 2019

A Secure Message Broker in an Untrusted Environment

by

Dylan Dreyer

A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor David E. Culler, Chair
Professor Raluca Ada Popa

Spring 2019

A Secure Message Broker in an Untrusted Environment

A Secure Message Broker in an Untrusted Environment

by Dylan Dreyer

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor David E. Culler
Research Advisor

5/16/19
(Date)

* * * * * * *

Professor Raluca Ada Popa
Second Reader

5/21/2019
(Date)

Abstract

A Secure Message Broker in an Untrusted Environment

by

Dylan Dreyer

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor David E. Culler, Chair

Publish/subscribe (pub/sub) is increasingly becoming a common communication paradigm in large-scale applications due to its flexibility and scalability. However, a common problem that has garnered a lot of research attention is how to secure these pub/sub systems as well as the data they handle. Especially of concern is the security risks when deploying pub/sub message routers in untrusted environments like the cloud. This thesis provides a brief discussion on current secure pub/sub systems and related technologies. We present an overview of a secure message broker as part of a pub/sub system that utilizes a state-of-the-art authorization system, an in-band key sharing mechanism, and trusted hardware execution environments to provide secure authorization and message confidentiality. We evaluate our system and show that the overhead of our added security mechanisms is negligible.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to recognize and thank many people for contributing to the completion of this thesis. First off, I would like to thank the graduate students in the BETS (Buildings, Energy, and Transportation Systems) research group for their mentorship, namely Michael Andersen, Gabe Fierro, and Sam Kumar. In particular, Michael Andersen deserves much credit for building the systems upon which I have conducted my research and guiding my efforts throughout the few years I have been affiliated with the BETS group. Professor David Culler deserves significant recognition for his guidance, advice, and mentorship of this project. I would also like to thank Professor Raluca Ada Popa for being a faculty reviewer for this work. Finally, I would like to acknowledge my family, friends, and the many peers whom I have interacted with throughout undergraduate and graduate school. They all have been a true source of inspiration, knowledge, and camaraderie.

Chapter 1

Introduction

1.1 Motivation

A communication paradigm known as publish/subscribe (commonly referred to as pub/sub) is becoming increasingly prevalent in large scale systems, as witnessed by the number of working pub/sub implementations generally available and in use [16, 27, 42, 54]. Popular big data messaging systems such as Kafka [33] and RabbitMQ [47] feature pub/sub functionality as well. The pub/sub paradigm features subscribers acting as consumers of data that publishers produce. However, the interaction between publishers and subscribers is decoupled in space, time, and synchronization through what Eugster et al. call an "event service" intermediary [21]. Less formally, pub/sub allows publishers and subscribers to make one connection to a central communication service instead of several individual point-to-point connections. By avoiding point-to-point communication, pub/sub greatly increases the scalability of large-scale applications and makes the paradigm very well suited for today's decoupled and distributed micro-service and cloud-based architectures.

Because these pub/sub systems are frequently handling sensitive data and deployed in untrusted domains such as the cloud, privacy of data and the underlying pub/sub system is of utmost concern. Data breaches are a risk in the cloud-based landscape today, and solutions to better secure data are eagerly sought after [15]. Furthermore, enterprises want to keep their data safe from other outside risks such as subpoenas [26] and insider attacks [20]. In general, two main approaches have been utilized to secure pub/sub: cryptographic techniques and trusted hardware execution environments.

Specific motivation for this work stems from the XBOS (eXtensible Building Operating System) project in the BETS (Building, Energy, and Transportation Systems) lab at UC Berkeley [23]. XBOS is a large-scale distributed operating system for smart buildings that aims to create a software-defined building infrastructure. XBOS is comprised of a collection of distributed services which are connected by a secure message bus, as can be seen in Figure 1.1. These services can range from embedded IoT (Internet of Things) devices to databases to machine learning models.

XBOS requires a scalable, robust, and secure communication (syndication) layer to transmit data that end services produce and consume. A secure pub/sub system is the natural fit for these requirements. Aside from protecting sensitive messages in the system, this syndication layer should also provide authentication and authorization along with necessary security in a built environment. The artifact that XBOS uses for pub/sub communication is WAVEMQ [2], a topic-based pub/sub system built over a decentralized authorization system called WAVE (Wide Area Verified Exchange) [3]. A more detailed overview of WAVE and WAVEMQ is provided in §2.3 and §2.4 respectively. WAVEMQ is well suited as a pub/sub system for XBOS because it provides secure authorization and end-to-end message encryption. As can be seen in Figure 1.1, WAVEMQ consists mainly of routers (site and designated) and services (end users). Crucial to the security of WAVEMQ is the security of the authorization layer, WAVE. However, WAVEMQ's current design does not fully protect WAVE when designated routers are operated in untrusted environments such as the cloud.

1.2 Problem Statement

In this work, we are specifically concerned with preserving the confidentiality of WAVE when a WAVEMQ designated router is deployed in untrusted environments such as the cloud. In WAVEMQ, all messages are accompanied by a WAVE proof which enumerates the permissions associated with the message. For example, the proof might confirm that a given message publisher indeed has "publish" permissions on some given topic being published to. To prevent malicious publishers in the system from flooding subscribers with invalid messages and to reduce the overhead of verifying WAVE proofs at the end users, WAVEMQ routers act as message brokers by examining the contents of the proof and verifying its validity. However, this proof reveals lots of sensitive information about the permissions it enumerates and also information about the underlying WAVE authorization layer (see §2.3 for more details on this). The confidentiality of WAVE permissions is at risk if these proofs are handled by a compromised router, as an attacker that gains access to a plaintext WAVE proof can view sensitive user data and permissions. Thus, the overarching problem that we consider in this work is keeping WAVE permissions and data confidential and secure in all environments.

Two main sub-problems arise from this scenario. The first is protecting the confidentiality of WAVE proofs when being handled in untrusted environments. The second is protecting the confidentiality of WAVE proofs *during decryption and validation on a WAVEMQ designated router*. Our goal is to provide additions to WAVEMQ that alleviate these problems.

1.3 Contributions

The contributions of this thesis are summarized as follows:

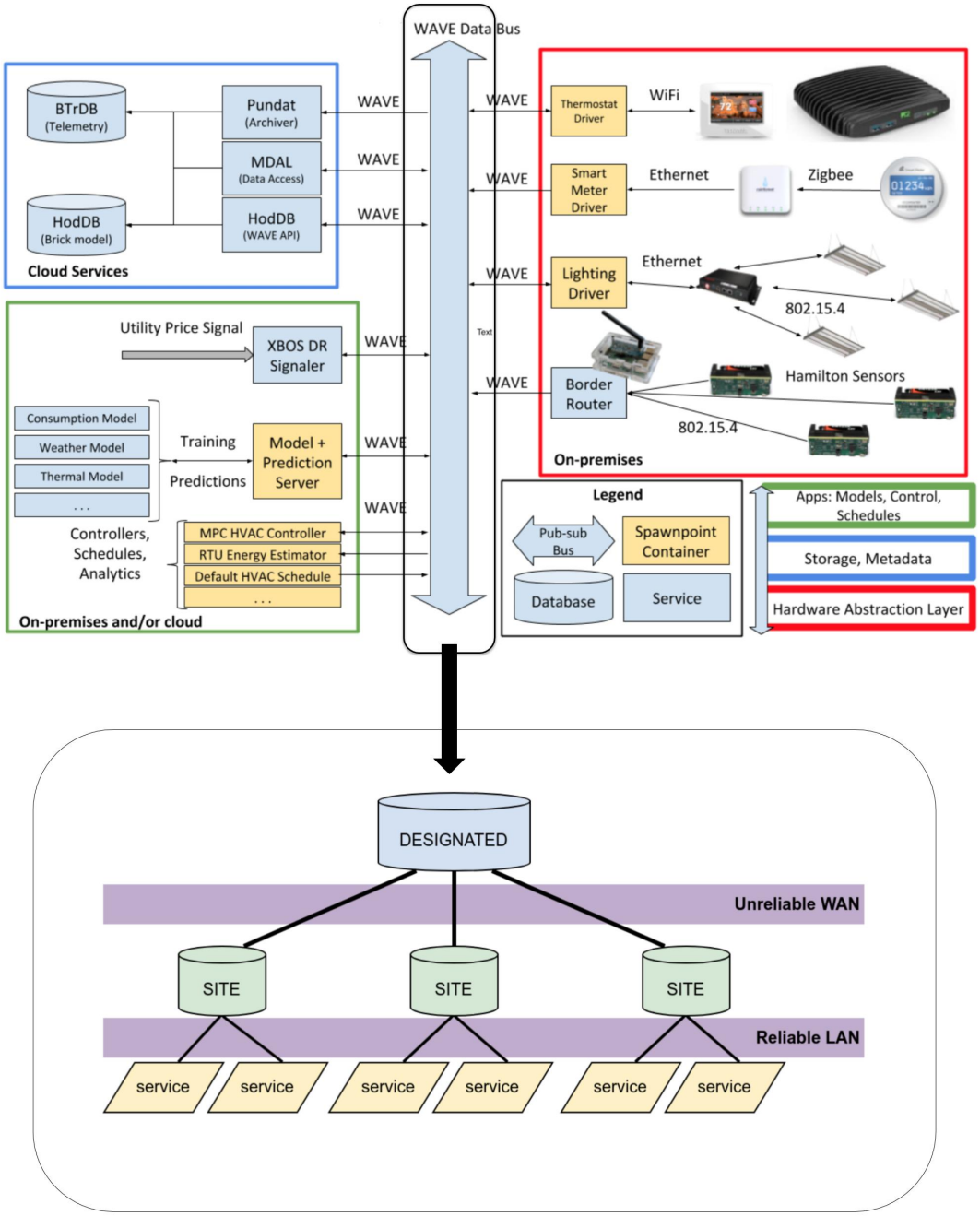


Figure 1.1: *Diagram of an example XBOS system with WAVEMQ operating as the data bus [2, 22]*

1. We analyze previous secure pub/sub systems in the literature and compare these works to our own.

2. We present a secure message broker and encryption mechanism for a pub/sub scheme, WAVEMQ, that protects the confidentiality of the underlying authorization system, WAVE, and features an in-band key sharing mechanism.

3. We evaluate our system and show that it introduces negligible overhead.

To address the problems in WAVEMQ's current design, we propose two major additions. First, we look at how to incorporate hardware enclaves into WAVEMQ designated routers to protect WAVE proofs during validation. Second, we look at how to protect the confidentiality of WAVE proofs when being handled in WAVEMQ through an encryption scheme leveraging WAVE itself. Opposed to many other secure pub/sub encryption schemes, our scheme allows publishers, subscribers, and routers to learn the keys needed to encrypt and decrypt WAVE proofs *without an out-of-band channel*. We call WAVEMQ with these additions *Secure WAVEMQ*.

We also discuss what constitutes our *trusted compute base (TCB)* [1], what components of our secure message broker run inside of an enclave, and how our secure message broker integrates into Secure WAVEMQ. Finally, we provide an evaluation of our implementation of Secure WAVEMQ.

1.4 Paper Overview

The paper is outlined as follows. Chapter 2 covers background on the core components that make up our system, including hardware enclaves, identity-based encryption, WAVE, and WAVEMQ. Chapter 3 provides an overview of related work on secure pub/sub and SGX-based cloud systems. In chapter 4, we give an overview of the design of our system, Secure WAVEMQ. In chapter 5, we look at several evaluations of our implementation, and we conclude with a summary of our work as well as directions for future research in chapter 6.

[1] The trusted compute base of a computer system is the set of components that are assumed to be running securely and if compromised could directly affect the security of the system.

Chapter 2

Background

2.1 Hardware Enclaves

Compute devices are ubiquitous in everyday life [39]. From the cloud to the Internet of Things, more and more compute devices are sensing, processing, and transmitting sensitive data in various domains. Typically in computer systems, data integrity is enforced by operating system isolation between applications. Furthermore, by owning and operating their own hardware, users can assert another level of security over their data. However, today's large scale applications running in datacenters handle lots of sensitive information on hardware that is under the control of external entities, such as cloud providers. These untrusted environments are a privacy hazard to sensitive user data, as untrusted hardware could be under the control of malicious entities. Hardware enclaves are an emerging technology that provide a trusted execution environment in the face of compromised system software or even a malicious operating system, essentially allowing an application to be shielded while running. Most existing enclave technology has come about from proprietary projects like ARM TrustZone [49], HyperWall [56], and Intel SGX [40]. Similarly, Apple now incorporates several secure enclaves into their iOS and mobile hardware architectures [5]. However, the open source community has recently started to develop hardware enclave technology as well [30]. In this paper, we focus on Intel SGX because of the widespread use of Intel processors in common cloud infrastructures and existing support for application development.

Intel Software Guard Extensions (SGX)

Intel Software Guard Extensions (Intel SGX) are a set of special instructions built into newer Intel processors that provide a trusted execution environment for applications [17, 40]. These instructions allow for a secure execution environment (called an enclave) to be instantiated and run in an Intel processor. Specifically, Intel SGX enclaves maintain confidentiality and integrity of application data and code by executing in a special processor mode that performs integrity checks on memory and code and uses a memory encryption engine to encrypt all data stored in memory. It prevents the application from switching privilege levels in case of a

malicious OS; any system call, interrupt, or fault that occurs during enclave code execution can only be serviced after the processor executes a special enclave exit procedure that securely saves the state of the enclave in enclave memory. Intel SGX also supports detection of code manipulation attacks, isolation between separate enclave instances, and replay prevention. These safeguards prevent a running application in an enclave from being tampered with or disrupted. The root of cryptographic trust in Intel SGX stems from two hardware-infused keys that are provisioned at manufacture time, a "Root Provisioning Key" and a "Root Sealing Key". From these two keys, an Intel SGX-enabled processor derives several other keys that are used for many different purposes such as memory encryption, data sealing, enclave signing, and remote attestation.

Due to these constraints, any application running in an enclave must adhere to strict conventions. The enclave must be instantiated successfully and checked by the processor before any application code is run. Interfaces for entering (*ecall*) and exiting (*ocall*) the enclave must be strictly defined by the developer. Code running in the enclave that needs to make a system call or other such invocation of untrusted code must make an *ocall* to have this code run outside of the enclave. Once completed, an *ecall* can be made to the enclave to resume execution. Upon completion of the program, the enclave is destroyed through a teardown procedure.

Though several side channel vulnerabilities have been discovered in Intel SGX ([11, 13, 35, 59], and most notably Meltdown [38] and Spectre [32]), we still view it as a promising technology that enhances the security of computing on sensitive data in untrusted environments.

Sealing

To protect the privacy of data stored on an untrusted machine, Intel SGX enclaves provide a functionality called *sealing*. This allows Intel SGX enclaves to store ("seal") sensitive enclave data in untrusted memory of host machines by encrypting the data with a sealing key protected in the processor. Enclaves can then retrieve ("unseal") this data at a later time (i.e. upon startup) and decrypt it for use. This is especially useful for storing sensitive data that should persist even if the enclave is shut down or stops operation for any reason.

Remote Attestation

Another feature of importance in Intel SGX is *remote attestation* [41]. This is the process of an enclave proving to a remote party that it is running a specific version of code. To do so, a particular running enclave produces a signed measurement (also called a *quote*) of the code it is running and delivers it to the remote party. The remote party can then verify the quote's authenticity and integrity. This quote contains sensitive information that could be used by an attacker to feign the operation of an enclave. Thus, this delivery procedure involves several steps of communication between the enclave and the remote party to instantiate a secure channel to transmit the sensitive quote. A remote party can ensure through remote

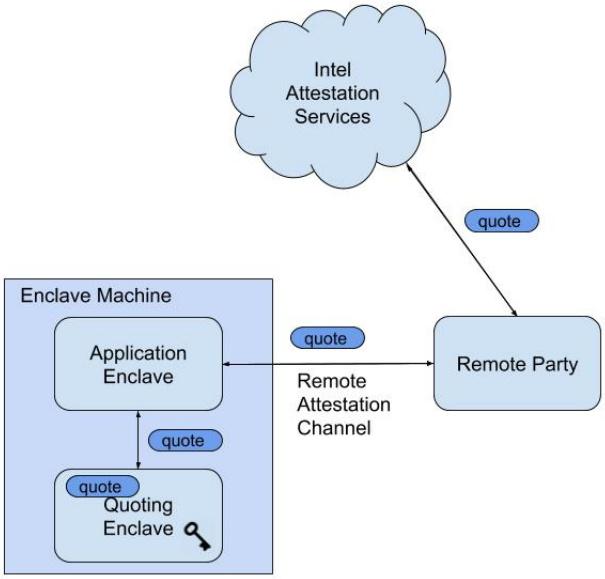


Figure 2.1: *Overview of the general steps of EPID-based remote attestation between an enclave and a remote party.*

attestation that the enclave it is communicating with is indeed running a specific version of code. This is especially important when provisioning secrets to an enclave that will be handling sensitive data i.e. handing over a secret key to an enclave such that it can decrypt sensitive data.

The primary method of Intel SGX remote attestation uses what is called Intel Enhanced Privacy Identifier (EPID). Figure 2.1 shows a general overview of this EPID-based remote attestation procedure. A quoting enclave running on the same machine as the enclave signs an enclave measurement to form a quote with an EPID key. This key, derived from one of the hardware infused root keys, creates an EPID-based signature, which is a group signature scheme based off of Direct Anonymous Attestation technology [12] that does not uniquely identify the platform. The EPID-based signature requires the remote party to communicate with Intel Attestation Services (IAS) servers to retrieve a group public key to verify that the signature was indeed created by an Intel-manufactured processor and that the signature has not been revoked for any reason [29].

However, due to demand from Intel SGX users wanting to host their own attestation infrastructure and not rely on IAS, Intel has recently been developing another mechanism for remote attestation via the Elliptic Curve Digital Signature Algorithm (ECDSA) [50]. ECDSA-based remote attestation allows for a remote party to produce their own verifiable quotes and thus not need to communicate with IAS. Though we use EPID-based remote attestation in our system, it could be useful to investigate in the future if ECDSA-based remote attestation could be implemented in Secure WAVEMQ.

Type	Name	TCB Size	Languages Supported
SDK	Intel SGX SDK	Small	C/C++
Framework	Asylo	Small	C/C++
Container	SCONE	Medium	Python, Java, C/C++, Golang, Node, Rust, R
	Panoply	Medium	C/C++
	Graphene	Large	Python, Java, C/C++, R
	Haven	Large	Unknown

Table 2.1: *Comparison of Intel SGX development tools and frameworks. TCB Size is measured in a relative manner.*

Intel SGX Development

As a result of our work, we have gained familiarity with many current Intel SGX development tools and frameworks. From our experience, we have found that because Intel SGX is a fairly new technology, many frameworks available to developers are either limited in functionality and/or language support or early in development. Furthermore, new frameworks are constantly appearing, making the development landscape hard to navigate. We want to specifically point out that lack of support for popular languages hinders application development. In the rest of this section and in Table 2.1, we summarize our experiences and findings when developing applications for Intel SGX.

Intel provides an Intel SGX SDK in C/C++ which includes all of the primitives needed to develop enclave applications [28]. However, this low level SDK does not provide higher level abstractions that could be useful to the casual developer and thus increases the overhead of developing for Intel SGX.

There have been some recent projects that support the same powerful primitives that the Intel SGX SDK contains while providing better abstractions and APIs for the developer. The main project in this realm is Asylo [24], an open source project from Google which provides a higher-level and more flexible C/C++ abstraction for general, platform-agnostic enclave development. Google has also recently released a similar higher-level abstraction for platform-agnostic remote attestation. [25].

Another direction that has been explored for Intel SGX development is secure containers. There have been several works that utilize Intel SGX to protect container processes by leveraging different points in the design space. This design space revolves around what system functionality should be included inside of an enclave to enable secure, containerized Linux processes. The works that we encountered were SCONE [7], Graphene [58], Panoply [55], and Haven [9]. In terms of TCB size and functionality inside of the enclave, Panoply and SCONE target a middle ground that provides an interface to bridge between the SGX C/C++ API and an untrusted operating system. On another hand, Haven and Graphene feature a full library operating system running inside an enclave that can support full application functionality.

Finally, open source enclave technology, such as the Keystone project [30], is starting to

appear. The aim is to build an open source hardware enclave, however the technology is in very early stages of development.

We choose to use the Intel SGX SDK for our implementation of Secure WAVEMQ because of its flexibility and support for the functionality we desire. However, we believe it could be a barrier to entry for the casual developer and look forward to the further advancement of developer-friendly tools for Intel SGX. We are encouraged by the growth of the Intel SGX community to date and excited for the prospect of more advanced SGX tools and frameworks that will make SGX development more accessible in the future.

2.2 Identity-Based Encryption

In Secure WAVEMQ, we use an encryption scheme also employed in WAVE called Identity-Based Encryption (IBE). Originally proposed in 1984 by Shamir [53], IBE is a form of public/private key cryptography based on allowing the public key of the recipient to be any arbitrary string. The original motivation for this was to allow for two communicating parties to discover public keys without a certificate authority in an email system. The classic IBE example involves two users, Alice and Bob. Alice wants to send a message to Bob so she simply encrypts the message under Bob's email address (e.g. `"bob@company.com"`). Bob must contact a third party, a Private Key Generator (PKG), to obtain the private key corresponding to the public key Alice used to encrypt the message. Bob can then decrypt the message and follow the same procedure in reverse to send encrypted messages to Alice. Though this scheme is beneficial in the sense that the two communicating parties never have to exchange public keys, it also suffers from an inherent problem known as key escrow, where a third party (the PKG) knows users' private keys.

Specifically, WAVE uses a special form of IBE encryption called LQ-IBE [37]. Similar to many other IBE schemes, LQ-IBE involves a pair of keys: a master public key and public ID (any string of bytes) paired with a specific private key for that ID. However, opposed to a more general IBE scheme [10], LQ-IBE is a hybrid encryption scheme[1] that is more efficient because it weakens the expensive public key encryption yet still ensures strong ciphertext security properties when combined with so-called "IND-CCA2"[2] secure symmetric key encryption schemes. We leverage LQ-IBE through WAVE to encrypt proofs in Secure WAVEMQ. We describe LQ-IBE's use in WAVE in §2.3 and LQ-IBE's use in Secure WAVEMQ in §4.4.

[1] A hybrid encryption scheme typically combines public key (asymmetric) encryption and symmetric key encryption to leverage the efficiency of symmetric key encryption while maintaining the convenience of public key encryption. This is commonly used in practice because public key encryption generally has much higher overhead than symmetric key encryption.

[2] IND-CCA2 (indistinguishability under adaptive chosen ciphertext attack) is a term that describes the highest level of ciphertext security.

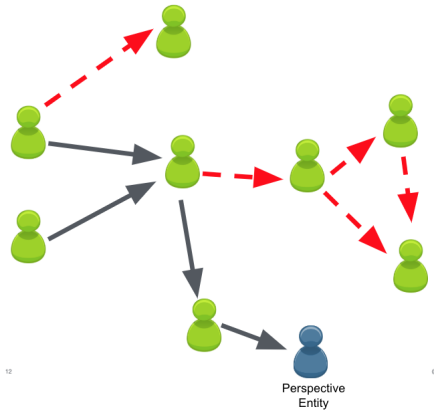


Figure 2.2: *This is an example WAVE Delegation of Trust (DoT) graph [4]. The arrows represent attestations that have been issued in the WAVE system. From the viewpoint of the perspective entity, only attestations that lie along a path that ends at the perspective entity can be discovered. This is called the perspective sub-graph. The perspective entity can use these attestations to form proof of permissions.*

2.3 WAVE

Wide Area Verified Exchange (WAVE) [3] is an authorization engine that provides secure access control. However, what separates WAVE from other authorization systems is that it is a fully autonomous, decentralized access control system which is built around allowing users to express and enforce access control securely without relying on a third party. WAVE permissions are cryptographically secured and are self-discoverable through an untrusted scalable storage backend while also allowing for revocation without a centralized server. WAVE is in essence a secure authorization framework with no central point of trust, and we leverage several of its desirable properties in our work.

WAVE revolves around entities, attestations, and proofs. Entities are users participating in the WAVE system and are represented by the figure-like objects in Figure 2.2. These entities can interact with one another by delegating permissions via attestations. An attestation is essentially a certificate granting permissions from one entity to another, and is represented by the arrows in Figure 2.2. Attestations form a graph structure, called a *Delegation of Trust (DoT)* graph, that indicate what permissions have been granted among entities in the WAVE system. Finally, entities can generate WAVE proofs to prove indisputably that they have received some permissions from another entity. In broad terms, a proof contains the attestations along the path from the entity who granted permissions to the entity trying to prove it has permissions. This path of attestations can be checked by any other entity participating in the WAVE system to ensure that the required permissions were indeed granted and valid.

Key Features of WAVE

WAVE is utilized as the underlying authorization and authentication layer for our Secure WAVEMQ pub/sub system. We list and describe particularly relevant features of WAVE from the perspective of Secure WAVEMQ in this section.

1. **Secure permissions**: Permissions are granted in WAVE through attestations. The actual permission policy in the attestation is encrypted under a special type of IBE scheme, WKD-IBE [1]. Each WAVE entity maintains its own WKD-IBE system instead of having a global PKG for the whole WAVE system, and encrypts permissions under the WKD-IBE public key and ID of the receiving entity's WKD-IBE system. A third party, malicious or not, cannot glean any information about permissions in WAVE by looking at arbitrary attestations because they are encrypted. Thus, the owner of a WAVEMQ topic can be assured that granting "publish" or "subscribe" permissions on the topic is secured.

2. **Delegation and revocation of permissions**: WAVE allows for *transitive delegation of permissions*. Users can re-delegate any subset of permissions as well as specify expiry policies for these permissions. WKD-IBE allows for fine-grained, hierarchical delegation of permissions through regex. WAVE allows for fine-grained control over permission **revocation** as well. Finally, there are **no ordering or timing constraints** on these permissions i.e. if one delegation of a permission expires, other delegations of that permission are unaffected. Granting of permissions can be asynchronous due to the fact that users can encrypt permissions under WKD-IBE public keys before the corresponding private keys even exists, and there is no need for an out-of-band communication channel to exchange keys.

3. **Offline participants and discoverability**: WAVE functions even when users are offline. The system allows for users to discover permissions granted to them at any time by listening for new attestations that add to their perspective DoT subgraph. Users can form proofs through a process that WAVE calls *reverse-discoverable encryption* and is made possible by how attestations are constructed in WAVE. Though we do not go in depth on how these attestations are constructed, one component of the attestation that is notable for our work is that it contains an IBE private key (separate from WKD-IBE) corresponding to the attestation issuer's IBE public key and ID set as the issuer's hash. This key is encrypted under a direct decryption key of the subject of the attestation, which allows any subject that has received permissions from an issuer to get access to this IBE private key. We will utilize this in Secure WAVEMQ as a key sharing mechanism.

4. **End to end encryption**: The actual permission policies that are contained in WAVE attestations are encrypted using WKD-IBE. This same mechanism can be used to encrypt arbitrary data, and we can employ this in WAVEMQ to secure user data end-to-end. Thus, WAVE provides not just secure authorization but also message content

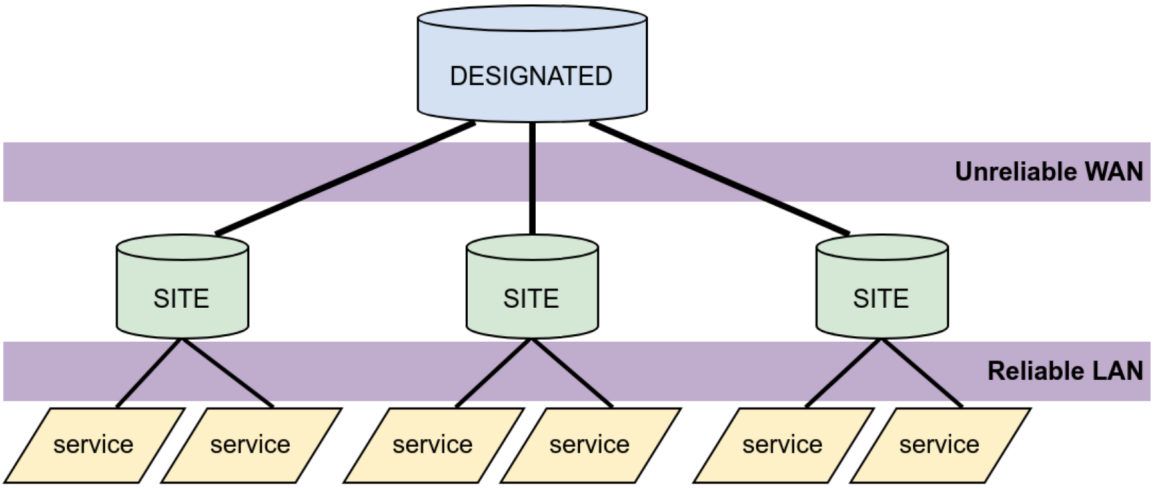


Figure 2.3: *Diagram of a theoretical WAVEMQ deployment, taken from [2].*

security in WAVEMQ. For users, this allows for *self-enforced authorization*, as they can give (through attestations) as many or as few individuals the ability to decrypt their own data.

2.4 WAVEMQ

WAVEMQ [2] is a topic-based pub/sub system that utilizes WAVE in order to provide authorization and message security. It can be deployed in systems such as XBOS (see §1.1), but also can be extended to any general pub/sub use case. As pictured in Figure 2.3, WAVEMQ features a tiered architecture with persistence in message queues in the face of an unreliable network. WAVEMQ also provides built in transport layer security and end-to-end encryption of message data. A message router called the "designated router", usually located in the cloud, routes messages between sites. The main components at these sites are publishers/subscribers (services) which are serviced by locally deployed site routers. In this thesis, we mainly focus on the "publish" and "subscribe" functionality that WAVEMQ provides users, although WAVEMQ does also provide "query" functionality by utilizing local message queues on site routers and persisting messages to disk on the designated router.

Chapter 3

Related Work

We group related cloud-based pub/sub systems to our work under two main catagories: hardware-centric and non hardware-centric approaches. We briefly give an overview of non hardware-centric works in §3.1, but focus on Intel SGX-based hardware-centric approaches in §3.2 and §3.4. The hardware-centric subcategories we focus on are SGX-based pub/sub systems and other cloud-based SGX systems including secure middleboxes. Table 3.1 presents a high level overview of a few important related systems to Secure WAVEMQ. Finally, §3.3 covers previous work on key sharing mechanisms in pub/sub schemes.

System	Paradigm	Key sharing	Data E2EE	Confidential
SCBR [46]	content-based	out-of-band	Yes	Yes
PubSub-SGX [6]	content-based	out-of-band	Yes	Yes
HyShare [43]	N/A	in-band	Yes	N/A
Secure WAVEMQ	**topic-based**	**in-band**	**Yes**	**No**[1]

Table 3.1: *Comparison of related works to Secure WAVEMQ.*

3.1 Secure Pub/Sub Systems

Secure pub/sub has become a popular area of research, where several techniques have been proposed covering both software-based and hardware-based solutions [45]. These works mainly concern data security and pub/sub confidentiality. Software-based techniques mostly revolve around various encryption schemes (symmetric, asymmetric, homomorphic, attribute-based, and more) [14, 18, 36, 44]. However, all of these schemes share similar weaknesses; they are either limited in functionality or require a globally trusted key sharing mechanism. In the rest of this section, we focus on works that utilize a hardware-centric approach to secure pub/sub and support broad functionality, similar to our work.

[1] We believe that a confidential WAVEMQ system is possible with further work. See §6.1.

3.2 SGX-Based Pub/Sub Systems

There are a couple existing SGX-based pub/sub systems that employ content-based routing. Pires et al. [46] propose a scheme with a cloud-based router that filters and routes messages based on a header, which is similar in design to our work. Their system, Secure Content-Based Routing (SCBR), requires a potential subscriber to contact a publisher directly to verify their subscription. The publisher then encrypts this subscription with a symmetric key shared beforehand with the cloud-based message router. The router decrypts and stores this message inside an index in the enclave such that when the publisher publishes a message, it can look up this subscription securely. SCBR does provide pub/sub confidentiality, but it requires out-of-band communication for its several key sharing steps. Pires et al. do not elaborate on how this key sharing mechanism would work, deeming it out of scope. We believe that the built-in key sharing mechanism in our system make it better suited for deployment in the real world.

Another SGX-based content-based routing system, PubSub-SGX [6], expands on the capabilities of SCBR by presenting a two-stage system architecture. PubSub-SGX adds a load balancer stage which load balances subscriptions across a set of SGX-enabled message brokers. Thus, the entire subscription space is split relatively evenly across many message brokers, allowing for parallelization and scalability. However, it does not provide secure authorization and end-to-end encryption of data.

3.3 Pub/Sub Key Sharing Mechanisms

As we have noted, a common weakness in many secure pub/sub systems is the lack of a practical key sharing mechanism to allow users to communicate securely end-to-end. In response to this, there has been some recent works specifically on key sharing schemes for pub/sub [60]. One in particular, HyShare [43], utilizes Shamir secret sharing [52] and Intel SGX enclaves to securely disseminate secrets between publisher and subscriber. However, this scheme requires some brokers in the system to be honest and uses SGX enclaves for disseminating secrets, not for message routing.

3.4 Other SGX-based Systems

Another type of system that also utilizes a message router in the cloud similar to pub/sub is network middleboxes. Duan et al. present a system called LightBox [19], which guarantees full-stack protection at native speeds inside of Intel SGX enclaves. Other works in the space include ShieldBox [57] and SafeBricks [48]. These systems use Intel SGX to build secure message brokers, but do not include the pub/sub functionality that we desire.

Intel SGX has also been used in anonymous networking schemes such as Tor [31] and in large scale systems for distributed computing [51, 61]. However, these systems also lack pub/sub functionality and practical key sharing mechanisms.

Chapter 4

System Overview

Secure WAVEMQ involves two major additions to baseline WAVEMQ. Secure WAVEMQ incorporates a hardware enclave into the WAVEMQ designated router that can verify WAVE proofs. To further ensure the confidentiality of WAVE proofs when being handled untrusted environments, Secure WAVEMQ also features a proof encryption scheme leveraging WAVE. We describe both of these additions and how they integrate into WAVEMQ in this section.

4.1 Threat Model

We formalize the threat model for Secure WAVEMQ here:

- We assume that end services (publishers and subscribers) and site routers are not compromised by any attacker, as they are assumed to be running on hardware under the control of a WAVEMQ user.
- The designated router is assumed to be running in an unreliable hardware environment such as the cloud. We assume a strong malicious attacker that could have fully compromised the designated router and could have physical access to the hardware itself. More specifically, we assume an adversary who potentially has superuser/root access to the system as well as access to the OS, kernel, hypervisor and/or other system software. This adversary could also potentially view and modify memory, the file system, system code, and/or network traffic.
- We assume that Intel SGX fully protects the code and data in an enclave and that remote attestation establishes a secure channel. As a result, we deem all side channel attacks on Intel SGX (such as the ones discussed in §2.1) out of scope for this work.
- We assume that WAVEMQ already implements transport layer security that protects against any network attacks.

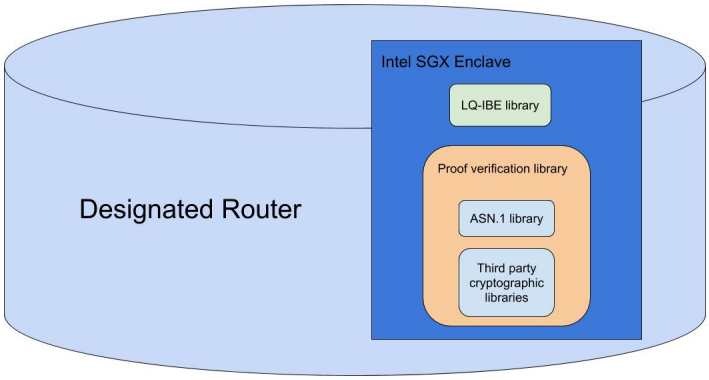


Figure 4.1: *Overview of components in a Secure WAVEMQ designated router message broker.*

- We assume that WAVEMQ already encrypts message contents end-to-end to protect message payloads from any possible attackers.
- We assume that the integrity and functionality of the underlying WAVE authorization system serving WAVEMQ is uncompromised.

4.2 Security Guarantees

The main security guarantee that Secure WAVEMQ provides on top of baseline WAVEMQ is that it guarantees the confidentiality of WAVE proofs on the WAVEMQ designated router. Specifically, Secure WAVEMQ guarantees that WAVE proofs attached to messages in WAVEMQ remain confidential on the designated router during handling (through encryption) **and** during decryption/verification (through hardware enclaves). Overall, this guarantees that the designated router will not leak any information about permissions in the Secure WAVEMQ system and thus ensure the security of the underlying authorization system, WAVE.

4.3 Secure Message Broker

We added several components to the WAVEMQ designated router to form the secure message broker of Secure WAVEMQ. We first implemented a WAVE proof verification library in C/C++. This library includes some third-party cryptographic libraries and auto generated ASN.1 C/C++ code[1]. We then needed to modify this proof verification library such that it

[1]WAVE objects are defined in ASN.1 (Abstract Syntax Notation One) notation, a standard descriptive language for serializing and deserializing objects across platforms. Our library contains auto generated code

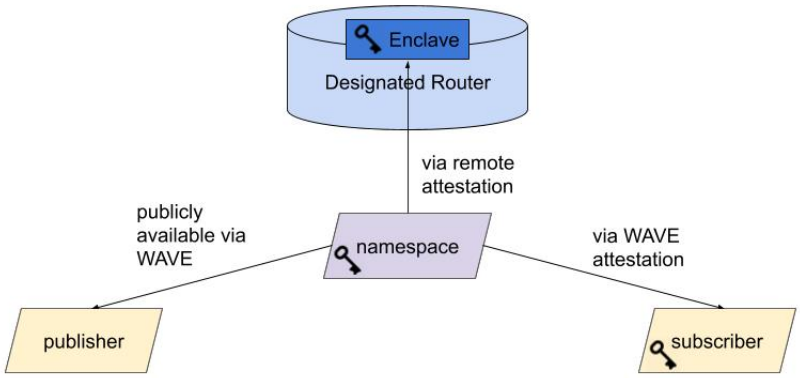


Figure 4.2: *Overview of how a namespace's private and public IBE keys are shared.*

could be compatible with Intel SGX, including changes to some of the auto generated ASN.1 C/C++ code such that it avoids using syscalls. We wrapped this modified proof verification library with an outer Intel SGX library to allow for proof verification to be done in an enclave. Finally, we incorporated an LQ-IBE library [34] into our Intel SGX library to support proof decryption inside of an enclave. This whole Intel SGX proof verification library runs on the Secure WAVEMQ designated router. Figure 4.1 shows the various components of our secure message broker.

4.4 Proof Encryption Scheme

We provide an overview of our proof encryption scheme that utilizes WAVE to ensure proof confidentiality and support in-band key sharing.

The WAVE mechanism we take advantage of is the sharing of IBE private keys through WAVE attestations. Once granted permissions in WAVEMQ from a namespace, a user has access to the IBE private corresponding to the namespace's IBE public key and ID that is set to the namespace's hash. Thus, a publisher, when publishing a message, can encrypt the accompanying proof with the IBE public key of the receiving namespace with the ID set as the public hash of the namespace. This encrypted proof can be decrypted by any subscriber that has been granted "subscribe" permissions in the namespace, as they will be possession of the corresponding IBE private key. This fits perfectly in Secure WAVEMQ, since any subscriber who has been granted "subscribe" permissions should also have the ability to view message proofs to verify them. WAVE's IBE key sharing mechanism ensures that the IBE private key will only be shared with entities who should be allowed to be in possession of it. The only way this key could be compromised is if an entity leaked the IBE private key or they themselves got compromised. Furthermore, any curious user participating in the system that has been granted permissions in a namespace would not be able to view any

from an ASN.1 to C/C++ compiler that allows us to deserialize WAVE objects into C/C++ objects.

other proofs other than ones explicitly routed to it unless it compromised one of the other components of the system. Our threat model makes no such probalistic assumption.

We implemented this proof encryption scheme by adding RPC calls to WAVE that allow WAVEMQ users to encrypt and decrypt proofs under a corresponding namespace. To allow the enclave running on the designated router to decrypt proofs and validate them, the namespace's IBE private key must be delivered to the enclave in a secure manner. This is achieved through a remote attestation procedure which we implemented and is described more in detail in the following section, §4.5.

All parties that need to verify a proof sent in WAVEMQ (publishers, subscribers, and brokers) are able to obtain the decryption key to do so via an in-band channel. A summary of how keys are shared can be seen in Figure 4.2. We argue that WAVEMQ's in-band key sharing mechanism utilizing WAVE separates our system from similar ones such as [46].

4.5 Secure WAVEMQ

Secure WAVEMQ builds upon baseline WAVEMQ and consists of several major components that all interact with one another. These components are one or more publishers, one or more subscribers, one or more namespaces, one or more site routers, a designated router with an Intel SGX enclave running on it, and a WAVE system. At a high level, the interaction of these components can be summarized as follows:

1. Namespaces give out publish/subscribe permissions on namespace topics (also known through WAVE as universal resource identifiers (URIs)).
2. Publishers publish messages to namespace topics.
3. Subscribers subscribe to namespace topics.
4. Site routers serve publisher and subscriber requests via RPCs and route traffic to and from the designated router.
5. The designated router routes traffic between site routers.

In this section, we discuss how we incorporate our two main additions, a secure message broker and proof encryption, into baseline WAVEMQ to form Secure WAVEMQ. Because site routers act at the behalf of publishers and subscribers and are assumed to be trusted, we do not mention them specifically in the rest of this chapter. We assume that they act as one with publishers and subscribers.

Designated Router Provisioning

The core feature of Secure WAVEMQ, a designated router with a running enclave, must be deployed and provisioned with the IBE decryption key before it can be operational. This

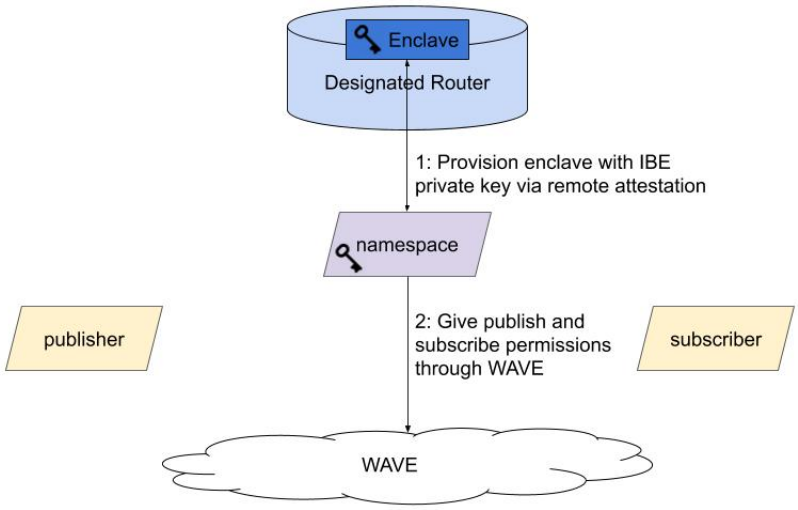


Figure 4.3: *Overview of provisioning a Secure WAVEMQ designated router.*

process is outlined in Figure 4.3. This is done via the EPID-based remote attestation procedure described in §2.1 between a namespace and the enclave that is validating messages on the designated router for the namespace. This is consequently followed by the enclave securely storing the received key on the designated router using the sealing procedure described in §2.1. This only needs to be done once per namespace, because the enclave can unseal the provisioned key if it crashes for any reason on restart. Note that the enclave only needs an IBE key from each namespace, not from each publisher and subscriber, which greatly reduces the number of keys that the enclave needs to manage.

After the designated router is provisioned, the namespace can delegate "publish" and "subscribe" permissions on namespace topics via WAVE and the designated router can start routing messages accordingly.

Subscribe

A basic subscribe procedure as seen in Figure 4.4 is as follows:

1. A subscriber obtains "subscribe" permissions to a topic from the namespace.
2. The subscriber sends a subscription request that also contains a proof of "subscribe" permissions. In Secure WAVEMQ, this proof is encrypted with the IBE public key and ID of the namespace.
3. The designated router receives the subscribe request and accompanying proof. In Secure WAVEMQ, the proof is decrypted with the provisioned IBE private key of the namespace and verified to validate the subscribe request inside of the enclave. Proof decryption and verification inside of an enclave on the designated router protects the confidentiality of user permissions that baseline WAVEMQ does not.

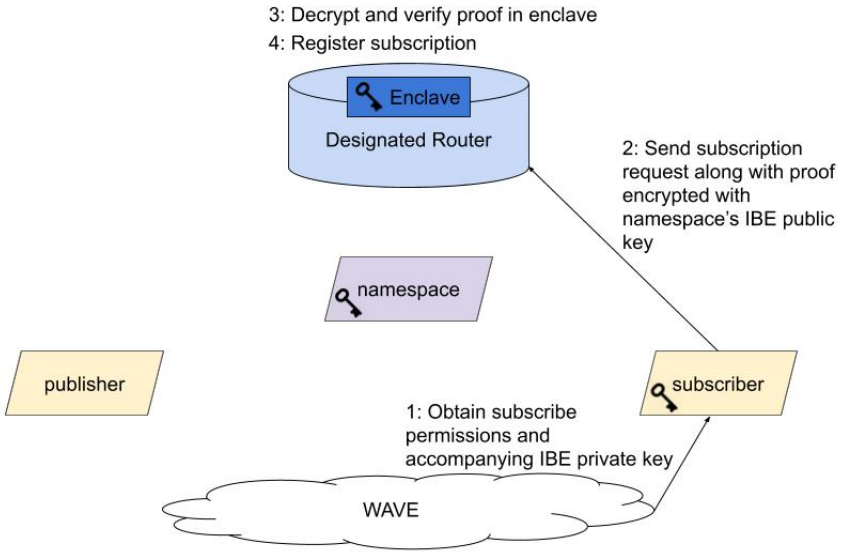


Figure 4.4: *Overview of a Secure WAVEMQ subscribe action.*

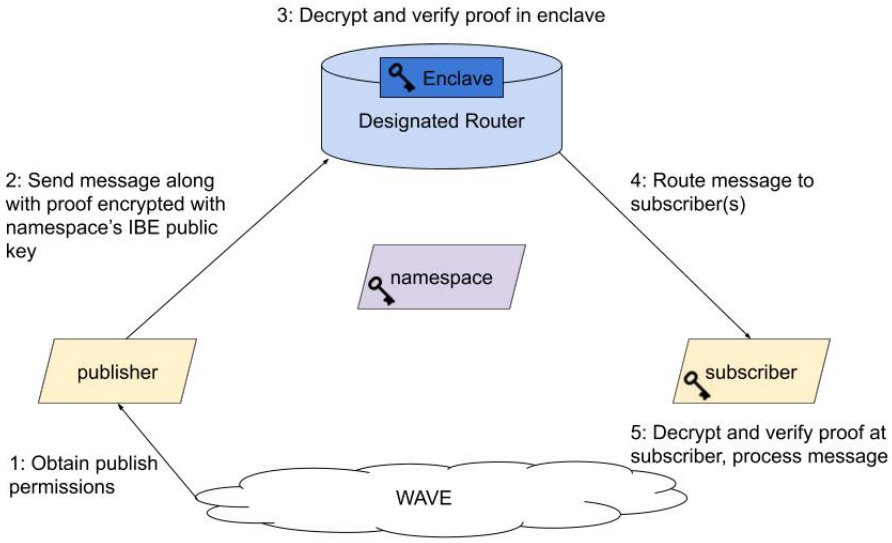


Figure 4.5: *Overview of a Secure WAVEMQ publish action.*

4. The designated router registers this valid subscription request and allocates the appropriate resources to serve the subscriber.

Publish

A basic publish procedure as seen in Figure 4.5 is as follows:

1. A publisher obtains "publish" permissions to a topic from the namespace.

2. The publisher publishes a message on this topic, encrypting the message contents with WAVE end-to-end encryption such that only the subscribers can decrypt the contents. In Secure WAVEMQ, along with the message, the publisher encrypts the accompanying WAVE proof of "publish" permissions with the IBE public key and ID of the namespace.

3. The designated router receives the publish message and accompanying proof. In Secure WAVEMQ, the proof is decrypted with the provisioned IBE public key of the namespace and verified to validate the publish message inside of the enclave. Proof decryption and verification inside of an enclave on the designated router protects the confidentiality of user permissions that baseline WAVEMQ does not.

4. After the publish message is verified by the enclave, the message is routed to any subscribers that have subscribed to the namespace topic being published to.

5. The subscriber receives the publish message and accompanying proof. In Secure WAVEMQ, the subscriber must decrypt and verify the proof to validate the message. Once the proof is validated, the subscriber can decrypt and process the message contents.

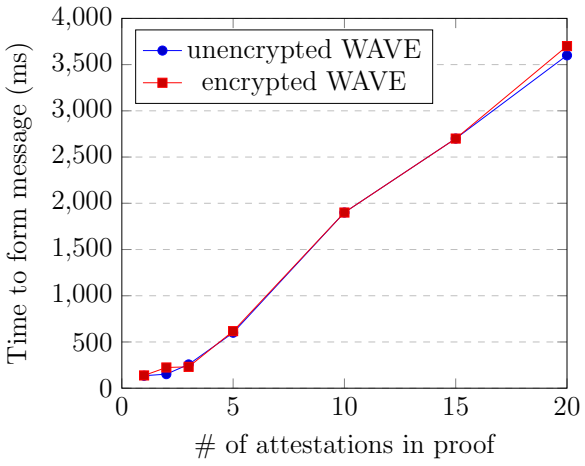
Chapter 5

System Evaluation

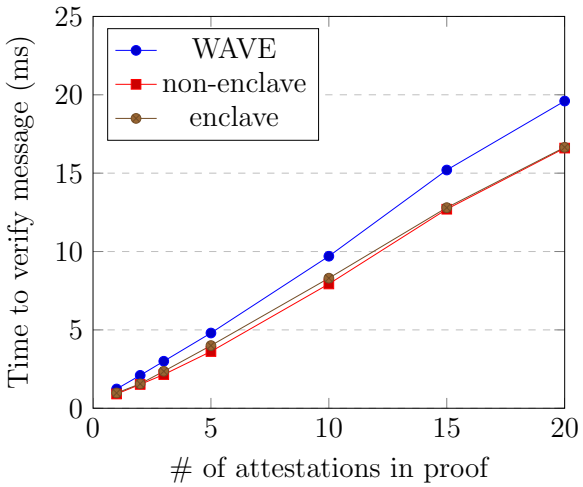
In this section, we present several metrics of our implementation of Secure WAVEMQ. Our code contributions include a WAVE proof verification library implemented in C/C++ and a parent library that encapsulates this proof verification library using the Intel SGX SDK. These libraries can be found at https://github.com/ddreyer/wave-verify and https://github.com/ddreyer/wave-verify-sgx respectively. The WAVE proof verification library contains approximately 24k LoC (lines of code) in total. 20k of these LoC consist of third party libraries and auto generated code. The parent proof verification library using the Intel SGX SDK contains approximately 4k LoC. The modifications to the WAVE and WAVEMQ source code to interface with these libraries and support proof encryption/decryption involve approximately 200 LoC. We ran our evaluations on a machine with a 2 core SGX-enabled Intel Kaby Lake i7-7567U CPU with a 4MB cache and 16GB of RAM. We also note that although the Intel SGX SDK supports running applications in "simulation" mode, all of our evaluations are run in "hardware" mode on actual SGX-enabled hardware.

5.1 Microbenchmarks

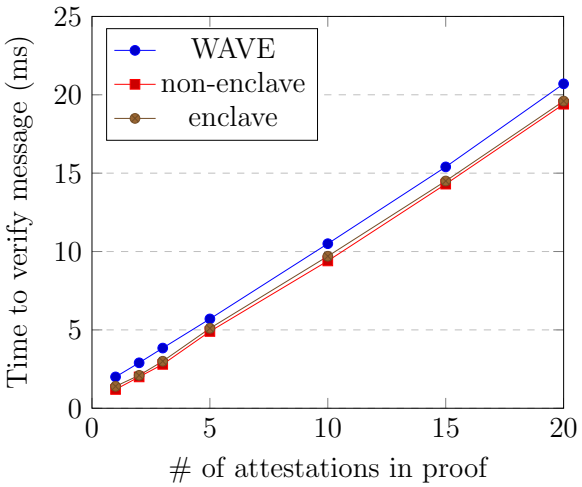
In this section, we look at the overhead induced by Secure WAVEMQ as compared to WAVEMQ on the two main components of the pub/sub system that it affects: message formation in publishers and message verification in brokers and subscribers. Message formation involves building a proof of permissions, encrypting the proof and message contents, and creating a signature over the message. Message verification involves verifying the message signature followed by decrypting and validating the proof of permissions. We believe these are meaningful benchmarks of our system because they form the bulk functionality of the main publish and subscribe actions and are not impacted by deployment-specific factors such as hardware, network, number of publishers/subscribers etc. We specifically are interested in how various-sized WAVE proofs affect the performance of message formation and verification. Our evaluation consists of measuring the overhead of message formation and verification on several different scenarios while varying the accompanying WAVE proof's



(a) Message formation overhead.



(b) Message verification overhead on unencrypted proofs.



(c) Message verification overhead on encrypted proofs.

Figure 5.1: *Overhead of message formation and verification across various scenarios.*

size. We varied proof size by adjusting the length of the chain of attestations in the proof. To do this, we created a chain of WAVE entities that all granted the same permission down the chain, with the last entity in the chain building a proof for this permission. The scenarios we evaluated include using encrypted/unencrypted proofs and verifying proofs using baseline WAVE (written in Golang [8]) versus our C/C++ verification library outside and inside an enclave. Although WAVEMQ does cache the results of proof building and verification for the duration of the proof's validity, we are interested in evaluating the worst case, a cold cache miss.

Message Formation

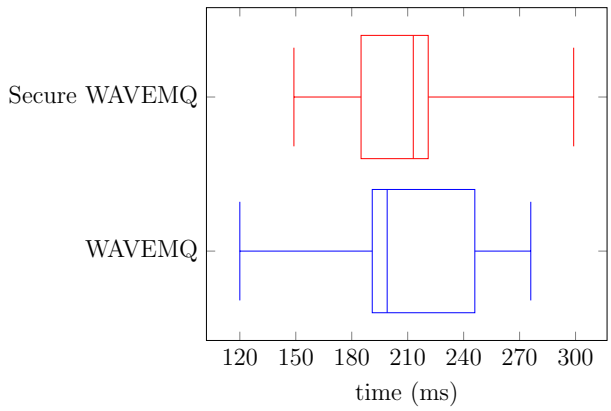
In message formation, we are interested in the overhead that proof encryption would have on message publishers. In our evaluations, we measure the message formation process in baseline and Secure WAVEMQ without any message content to isolate the proof creation step. Note that in messages with message content, message formation overhead would expect to increase due to the need to encrypt the message content. From the results in Figures 5.1a, we see that encryption has minimal overhead and that the overhead is linear with respect to the number of attestations in the proof.

Message Verification

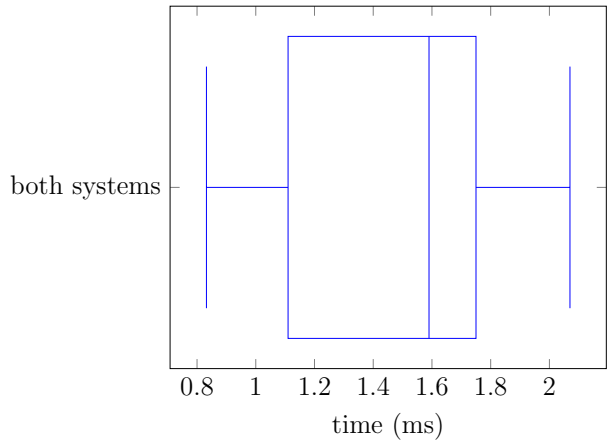
In message verification, we are interested in the overhead that proof decryption and validation have on message verifiers. Specifically, designated routers decrypt and validate proofs in Intel SGX, while subscribers decrypt and validate proofs using the baseline WAVE implementation. However, in our evaluation we include all possible in-between scenarios including unencrypted/encrypted proofs running on baseline WAVE and our C/C++ verification library running inside and outside an enclave. The results in Figures 5.1b and 5.1c confirm the linear relationship between the number of attestations in the proof and the time needed to verify for all setups. It also shows that there is a minimal fixed cost of running our verification code inside of an enclave, the cost per attestation is more or less the same, and that the overhead of proof decryption is minimal. Finally, we attribute our C/C++ proof verification library outperforming WAVE's verification module to the superior performance of C/C++ when compared to Golang.

5.2 Functional Benchmarks

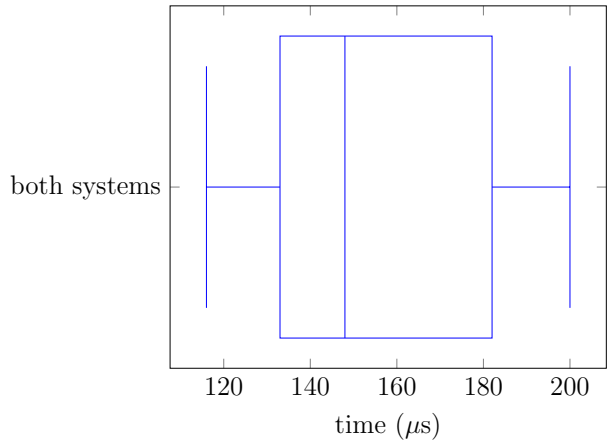
We next evaluate the two main actions that users of Secure WAVEMQ can invoke: publish and subscribe. We compare the time it takes to publish and subscribe from the perspective of the client between baseline WAVEMQ and Secure WAVEMQ. We use a very basic setup that involves one namespace giving permissions to one publisher and one subscriber. In our publish messages, we include no message payload such that there is no message content encryption overhead added in. Figure 5.2a shows the expected minimal overhead that proof encryption has in Secure WAVEMQ when there is a cache miss. Figure 5.2b shows the overhead of a publish action on a cache hit, which inherently makes the overhead of Secure WAVEMQ equivalent to baseline WAVEMQ since cache misses are infrequent. A cache miss occurs when there is not a valid proof built for a publish message for a given topic. WAVEMQ caches the results of proof formation for a given topic for the duration of the proof's validity. Figure 5.2c shows the overhead of the subscribe action regardless of setup. The reason it is system independent is because the client receives a handle to a message stream on a subscribe RPC call rather than waiting for the result of the action.



(a) Overhead of user publish action on cache miss.



(b) Overhead of user publish action on cache hit.



(c) Overhead of user subscribe action.

Figure 5.2: *Distributions of overhead of user publish and subscribe actions on our experimental setups.*

Chapter 6

Conclusion

6.1 Future Work

A natural direction for this work in the future is exploring a confidential version of WAVEMQ, similar to other confidential pub/sub systems. Currently, the enclave running in the designated router only protects the confidentiality of WAVE proofs and the WAVEMQ authorization layer. Confidential WAVEMQ would entail running the whole WAVEMQ designated router in an enclave to protect the confidentiality of the subscriptions and publications themselves. We believe this is possible using a secure container framework such as SCONE, which currently supports compiling Golang inside of a trusted execution container. However, there are questions of performance and scalability when going in this direction, mainly due to enclave overhead and the heavy use of memory by WAVEMQ routers to store message queues. Enclave memory is limited, so the working set of a WAVEMQ designated router could cause significant overhead issues.

We also see investigating scaling WAVEMQ to larger deployments and possibly utilizing multiple designated routers to handle large WAVEMQ systems. We also see investigating the use of ECDSA remote attestation to remove the need to rely on Intel services for remote attestation. Finally, the vulnerability of Intel SGX to numerous side channel attacks could pose significant security concerns and limit the effectiveness of our system.

6.2 Final Remarks

In this thesis, we present the design and implementation of a secure message broker in a pub/sub system, Secure WAVEMQ, which features Intel SGX hardware enclaves deployed on message routers in the cloud. Secure WAVEMQ also leverages a state-of-the-art authorization system, WAVE, to create an in-band key sharing mechanism between publishers, subscribers, and routers. Our design overall looks to protect the authorization layer of Secure WAVEMQ from powerful attackers that may have uninhibited access to message routers in

the cloud. Finally, we show that our improvements incur a minimal overhead to all components of the system, making it suitable for large-scale deployment.

Our work, coupled with other related secure SGX-based pub/sub systems, highlights the security benefits that trusted hardware execution environments have on pub/sub communication. More generally, recent developments in trusted hardware execution environments securing applications in the cloud point to a movement of "migrating the cloud into the enclave". Combined with innovations in other confidential computing techniques, we see our work as taking one step closer to building secure applications in the growing cloud ecosystem. With the growth of large-scale applications operating in secure environments, enterprises and users will feel more secure when computing on sensitive data in the cloud, further advancing the impact and benefit of cloud computing.

Bibliography

[1] Michel Abdalla, Eike Kiltz, and Gregory Neven. "Generalized Key Delegation for Hierarchical Identity-Based Encryption". In: *Computer Security – ESORICS 2007*. Ed. by Joachim Biskup and Javier López. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 139–154. ISBN: 978-3-540-74835-9.

[2] Michael P Andersen. *WAVEMQ*. https://github.com/immesys/wavemq. 2018.

[3] Michael P Andersen et al. *WAVE: A Decentralized Authorization System for IoT via Blockchain Smart Contracts*. Tech. rep. UCB/EECS-2017-234. EECS Department, University of California, Berkeley, Dec. 2017. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-234.html.

[4] Michael Andersen, Sam Kumar, and Gabe Fierro. *WAVE: Decentralized global authorization without a trusted party*. 2017. URL: https://docs.google.com/presentation/d/1Dvn5pGE5sXwenFCLk25xEd1YClqlgYqjGhdA58BvO9I/edit?usp=sharing.

[5] Apple Inc. *iOS Security, iOS 12.1, November 2018*. Tech. rep. 2018.

[6] Sergei Arnautov et al. "PubSub-SGX: Exploiting Trusted Execution Environments for Privacy-Preserving Publish/Subscribe Systems". In: Oct. 2018, pp. 123–132. DOI: 10.1109/SRDS.2018.00023.

[7] Sergei Arnautov et al. "SCONE: Secure Linux Containers with Intel SGX". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 689–703. ISBN: 978-1-931971-33-1. URL: http://dl.acm.org/citation.cfm?id=3026877.3026930.

[8] Go Authors. *Golang*. 2019. URL: https://golang.org/ (visited on 04/20/2018).

[9] Andrew Baumann, Marcus Peinado, and Galen Hunt. "Shielding Applications from an Untrusted Cloud with Haven". In: *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*. OSDI'14. Broomfield, CO: USENIX Association, 2014, pp. 267–283. ISBN: 978-1-931971-16-4. URL: http://dl.acm.org/citation.cfm?id=2685048.2685070.

[10] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '01. London, UK, UK: Springer-Verlag, 2001, pp. 213–229. ISBN: 3-540-42456-3. URL: http://dl.acm.org/citation.cfm?id=646766.704155.

[11] Ferdinand Brasser et al. "Software Grand Exposure: SGX Cache Attacks Are Practical". In: *11th USENIX Workshop on Offensive Technologies (WOOT 17*.

[12] Ernie Brickell, Jan Camenisch, and Liqun Chen. "Direct Anonymous Attestation". In: *Proceedings of the 11th ACM Conference on Computer and Communications Security*. CCS '04. Washington DC, USA: ACM, 2004, pp. 132–145. ISBN: 1-58113-961-6. DOI: 10.1145/1030083.1030103. URL: http://doi.acm.org.libproxy.berkeley.edu/10.1145/1030083.1030103.

[13] Jo Van Bulck et al. "Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1041–1056. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck.

[14] Sunoh Choi, Gabriel Ghinita, and Elisa Bertino. "A Privacy-Enhancing Content-Based Publish/Subscribe System Using Scalar Product Preserving Transformations". In: *Database and Expert Systems Applications*. Ed. by Pablo García Bringas, Abdelkader Hameurlain, and Gerald Quirchmayr. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 368–384. ISBN: 978-3-642-15364-8.

[15] Privacy Rights Clearinghouse. *DATA BREACHES*. 2019. URL: https://www.privacyrights.org/data-breaches (visited on 04/27/2019).

[16] Google Cloud. *CLOUD PUB/SUB*. 2019. URL: https://cloud.google.com/pubsub/ (visited on 03/20/2019).

[17] Victor Costan and Srinivas Devadas. *Intel SGX Explained*. Cryptology ePrint Archive, Report 2016/086. https://eprint.iacr.org/2016/086. 2016.

[18] Giovanni Di Crescenzo et al. "Privacy-Preserving Publish/Subscribe: Efficient Protocols in a Distributed Model". In: *Data Privacy Management and Autonomous Spontaneous Security*. Ed. by Joaquin Garcia-Alfaro et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 114–132. ISBN: 978-3-642-54568-9.

[19] Huayi Duan, Xingliang Yuan, and Cong Wang. "LightBox: SGX-assisted Secure Network Functions at Near-native Speed". In: *CoRR* abs/1706.06261 (2017). arXiv: 1706.06261. URL: http://arxiv.org/abs/1706.06261.

[20] A. J. Duncan, S. Creese, and M. Goldsmith. "Insider Attacks in Cloud Computing". In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. June 2012, pp. 857–862. DOI: 10.1109/TrustCom.2012.188.

[21] Patrick Th. Eugster et al. "The Many Faces of Publish/Subscribe". In: *ACM Comput. Surv.* 35.2 (June 2003), pp. 114–131. ISSN: 0360-0300. DOI: 10.1145/857076.857078. URL: http://doi.acm.org/10.1145/857076.857078.

[22] Gabe Fierro. *XBOS Overview*. 2019. URL: https://docs.xbos.io (visited on 03/07/2019).

[23] Gabriel Fierro and David E. Culler. *XBOS: An Extensible Building Operating System*. Tech. rep. UCB/EECS-2015-197. EECS Department, University of California, Berkeley, Sept. 2015. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-197.html.

[24] Google. *Asylo*. 2019. URL: https://asylo.dev/ (visited on 02/28/2019).

[25] Google. *Go-Attestation*. https://github.com/google/go-attestation. 2019.

[26] Google. *Transparency Report*. 2019. URL: https://transparencyreport.google.com/user-data/overview?user_requests_report_period=authority:US (visited on 04/27/2019).

[27] Amazon Inc. *PUB/SUB MESSAGING*. 2019. URL: https://aws.amazon.com/pub-sub-messaging/ (visited on 03/20/2019).

[28] Intel. *Intel Software Guard Extensions SDK for Linux*. 2019. URL: https://01.org/intel-softwareguard-extensions (visited on 02/28/2019).

[29] Simon Johnson et al. *Intel Software Guard Extensions: EPID Provisioning and Attestation Services*. Tech. rep. 2016. URL: https://software.intel.com/sites/default/files/managed/57/0e/ww10-2016-sgx-provisioning-and-attestation-final.pdf.

[30] *Keystone: Open-source Secure Hardware Enclave*. 2019. URL: https://keystone-enclave.org/ (visited on 03/26/2019).

[31] Seongmin Kim et al. "Enhancing Security and Privacy of Tor's Ecosystem by Using Trusted Execution Environments". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 145–161. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kim-seongmin.

[32] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *CoRR* abs/1801.01203 (2018). arXiv: 1801.01203. URL: http://arxiv.org/abs/1801.01203.

[33] Jay Kreps, Neha Narkhede, and Jun Rao. "Kafka: a Distributed Messaging System for Log Processing". In: 2011.

[34] Sam Kumar. *Embedded Pairing*. https://github.com/samkumar/embedded-pairing. 2019.

[35] Sangho Lee et al. "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 557–574. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-sangho.

[36] Jun Li, Chenghuai Lu, and Weidong Shi. "An Efficient Scheme for Preserving Confidentiality in Content-Based Publish-Subscribe Systems". In: (Feb. 2004).

[37] Benoît Libert and Jean-Jacques Quisquater. "Identity Based Encryption Without Redundancy". In: *Applied Cryptography and Network Security*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 285–300. ISBN: 978-3-540-31542-1.

[38] Moritz Lipp et al. "Meltdown". In: *CoRR* abs/1801.01207 (2018). arXiv: 1801.01207. URL: http://arxiv.org/abs/1801.01207.

[39] Knud Lasse Lueth. *State of the IoT 2018: Number of IoT devices now at 7B ffffdfffdffffd Market accelerating*. 2019. URL: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/ (visited on 04/28/2019).

[40] Frank McKeen et al. "Innovative Instructions and Software Model for Isolated Execution". In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. Tel-Aviv, Israel: ACM, 2013, 10:1–10:1. ISBN: 978-1-4503-2118-1. DOI: 10.1145/2487726.2488368. URL: http://doi.acm.org/10.1145/2487726.2488368.

[41] John Mechalas. *Code Sample: Intel Software Guard Extensions Remote Attestation End-to-End Example*. 2018. URL: https://software.intel.com/en-us/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example (visited on 04/25/2019).

[42] MQTT. *MQTT*. 2019. URL: http://mqtt.org/ (visited on 03/20/2019).

[43] Javier Munster and Hans-Arno Jacobsen. "Secret Sharing in Pub/Sub Using Trusted Execution Environments". In: *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. DEBS '18. Hamilton, New Zealand: ACM, 2018, pp. 28–39. ISBN: 978-1-4503-5782-1. DOI: 10.1145/3210284.3210290. URL: http://doi.acm.org.libproxy.berkeley.edu/10.1145/3210284.3210290.

[44] Mohamed Nabeel et al. "Privacy Preserving Context Aware Publish Subscribe Systems". In: *Network and System Security*. Ed. by Javier Lopez, Xinyi Huang, and Ravi Sandhu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 465–478. ISBN: 978-3-642-38631-2.

[45] Emanuel Onica et al. "Confidentiality-Preserving Publish/Subscribe: A Survey". In: *CoRR* abs/1705.09404 (2017). arXiv: 1705.09404. URL: http://arxiv.org/abs/1705.09404.

[46] Rafael Pires et al. "Secure Content-Based Routing Using Intel Software Guard Extensions". In: *CoRR* abs/1701.04612 (2017). arXiv: 1701.04612. URL: http://arxiv.org/abs/1701.04612.

[47] Inc. Pivotal Software. *RabbitMQ*. 2019. URL: https://www.rabbitmq.com/ (visited on 03/20/2019).

[48] Rishabh Poddar et al. "SafeBricks: Shielding Network Functions in the Cloud". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, 2018, pp. 201–216. ISBN: 978-1-931971-43-0. URL: https://www.usenix.org/conference/nsdi18/presentation/poddar.

[49] Nuno Santos et al. "Using ARM TrustZone to Build a Trusted Language Runtime for Mobile Applications". In: vol. 49. Feb. 2014, pp. 67–80. DOI: 10.1145/2541940.2541949.

[50] Vinnie Scarlata et al. *Supporting Third Party Attestation for Intel SGX with Intel Data Center Attestation Primitives*. Tech. rep. 2018. URL: https://software.intel.com/sites/default/files/managed/f1/b8/intel-sgx-support-for-third-party-attestation.pdf.

[51] F. Schuster et al. "VC3: Trustworthy Data Analytics in the Cloud Using SGX". In: *2015 IEEE Symposium on Security and Privacy*. May 2015, pp. 38–54. DOI: 10.1109/SP.2015.10.

[52] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[53] Adi Shamir. "Identity-based Cryptosystems and Signature Schemes". In: *Proceedings of CRYPTO 84 on Advances in Cryptology*. Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1985, pp. 47–53. ISBN: 0-387-15658-5. URL: http://dl.acm.org/citation.cfm?id=19478.19483.

[54] Yogeshwer Sharma et al. "Wormhole: Reliable Pub-Sub to Support Geo-replicated Internet Services". In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 351–366. ISBN: 978-1-931971-218. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/sharma.

[55] Shweta Shinde et al. "Panoply: Low-TCB Linux Applications With SGX Enclaves". In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/panoply-low-tcb-linux-applications-sgx-enclaves/.

[56] Jakub Szefer and Ruby B. Lee. "Architectural Support for Hypervisor-secure Virtualization". In: *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVII. London, England, UK: ACM, 2012, pp. 437–450. ISBN: 978-1-4503-0759-8. DOI: 10.1145/2150976.2151022. URL: http://doi.acm.org.libproxy.berkeley.edu/10.1145/2150976.2151022.

[57] Bohdan Trach et al. "ShieldBox: Secure Middleboxes Using Shielded Execution". In: *Proceedings of the Symposium on SDN Research*. SOSR '18. Los Angeles, CA, USA: ACM, 2018, 2:1–2:14. ISBN: 978-1-4503-5664-0. DOI: 10.1145/3185467.3185469. URL: http://doi.acm.org/10.1145/3185467.3185469.

[58] Chia-che Tsai, Donald E. Porter, and Mona Vij. "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX". In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 645–658. ISBN: 978-1-931971-38-6. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai.

[59] Wenhao Wang et al. "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX". In: *CoRR* abs/1705.07289 (2017). arXiv: 1705.07289. URL: http://arxiv.org/abs/1705.07289.

[60] Young Yoon and Beom Heyn Kim. "Secret Forwarding of Events over Distributed Publish/Subscribe Overlay Network". In: *PLOS ONE* 11.7 (July 2016), pp. 1–23. DOI: 10.1371/journal.pone.0158516. URL: https://doi.org/10.1371/journal.pone.0158516.

[61] Wenting Zheng et al. "Opaque: An Oblivious and Encrypted Distributed Analytics Platform". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 283–298. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng.