Securing Publish/Subscribe

by

Javier Munster

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Securing Publish/Subscribe

Javier Munster

Master of Applied Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2018

Increased public scrutiny has led to calls for greater security regarding user data. As a widely used many-to-many communication paradigm, publish/subscribe (pub/sub) has received a significant amount of attention from researchers regarding ways in which to secure user data. In this thesis, we present the current state-of-the-art of securing pub/sub systems. We categorize the existing research, presenting what is currently achievable and identify gaps and potential areas of research. One such gap is the assumption of an initial established security parameter, shared secret or key between communicating clients. We propose a novel scheme called HyShare that does not require an unilaterally trusted, universally available, out-of-band service for the dissemination of a secret.

Contents

Chapter 1

Introduction

1.1 Motivation

Publish/Subscribe (pub/sub) is a simple communication paradigm that allows loosely coupled clients to communicate in an event-based manner. Due to its efficiency, pub/sub has emerged as an efficient many-to-many communication framework. As a result, pub/sub has become prevalent in a wide variety of application domains, including social networking (e.g., Facebook's Wormhole [97]), music streaming (e.g., Spotify [94]), healthcare [72, 7, 101, 102], cyber-physical systems (e.g., location-based services [122], sensor networks [120, 78]), business process management [58], policy management [118] and stock quotes dissemination [114]. The prevalence of pub/sub is further reflected in the large number of open and closed source implementations available, including Amazon Simple Notification Service [2], Apache Pulsar [5], Google Cloud Pub/Sub [40], HedWig [42], Hermes [1], MQTT [69], PADRES [53], Redis [91] and Siena [15].

Despite pub/sub – and communication paradigms in general – being largely invisible to end users, concerns about the handling of personal data are increasingly under public scrutiny. Supporting examples of this are too numerous to list [14, 22, 73]. We will briefly highlight some notable recent cases. One recent incident involved Facebook and Cambridge Analytica. In what has become a very public scandal, the personal data of 50 million Facebook users was gathered by Cambridge Analytica without the knowledge or consent of these users [63]. The announcement of an Federal Trade Commission

investigation into this matter prompted a sharp decline in Facebook stock value of almost $50 billion USD [35] and culminated with Facebook CEO Mark Zuckerberg testifying before both the U.S. Congress and the European Parliament. The increasing prominence of privacy in the public is also fueled by additional prominent examples. This includes the Equifax data leak of financial information, the Uber breach of personal data of 57 million users, the credit card data leak of Whole Foods, etc. [22]. The increased public scrutiny has led to increased regulation in how sensitive data is to be handled [31].

Parallel to the public scrutiny over the handling of their data are domain-specific requirements dictating that particular data must have specific security properties. The need for confidentiality is immediately clear in some domains (e.g., consumer credit card records) and sometimes required by law (e.g., healthcare data must be encrypted before online transmission [8]). In this domain, policy is used to determine what kind of data publishers are allowed to transmit (e.g., spam control measures might set a limit to the publishing rate of publishers) and which subscribers are allowed to receive it (e.g., a family doctor should normally not be receiving health records for people who are not their patients). Whistle-blowers and political dissidents often have an interest in retaining their anonymity. Service providers have an interest in making their service available and so must ensure the integrity of the pub/sub system.

In this thesis, we present the current state-of-the-art in terms of pub/sub security, highlighting what is currently achievable and what is not. In addition, we present our own novel work towards the sharing of a secret in pub/sub.

1.2 Problem Statement

The goal of this thesis is to present a detailed analysis of how pub/sub is secured in the academic literature. We argue that there is a lack of knowledge regarding what problems have been solved, and what problems have yet to be solved. Our analysis identifies gaps in the current literature and potential areas of future research. Since pub/sub handles information dissemination, it is subject to serious security concerns about how this potentially sensitive information is handled. These concerns are wide-ranging and

critical for the adoption of pub/sub in broader applications [114]. Several application domains require the consideration of some security aspect, if not multiple. Indeed, the entire choice of communication paradigm might hinge on the security guarantees that can be achieved. We limit the scope of our overview to those approaches that have been directly applied to pub/sub, as otherwise our overview would grow out of bounds.

A second objective of this work is to address the problem of efficiently sharing a secret among communicating publishers and subscribers without relying on a unilaterally trusted, universally available, out-of-band service to do this. The existing literature almost universally assume the existence of such a service to establish some initial secret key (e.g., symmetric-key encryption [59, 16, 24]), public key (e.g., asymmetric-key encryption [18, 71, 55]) or shared secret (e.g., attribute-based encryption [107, 49, 124, 109]) which is later used for key generation. The service is unilaterally trusted to provide accurate information, universally available to service any newly joining entities to the pub/sub system at any time so that they may receive the necessary keys and out-of-band in its need for additional resources that operate independently of the existing pub/sub system. There are additional implementation-specific features which this service is often used to provide. This includes the generation of new keys as arbitrarily requested by publishers and subscribers, the regeneration of keys whenever a subscriber subscribes or unsubscribes to an event stream and the periodic refreshing of all keys to support forward and backward secrecy. We argue that such a service is not needed in the process of sharing a secret in pub/sub.

1.3 Approach

We have categorized the existing pub/sub security literature as fitting into one of four security domains: confidentiality, authorization, anonymity, and integrity.

Confidentiality is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes [52]. This property is critical for the adoption of pub/sub systems in applications that must disseminate private or otherwise sensitive information. Unfortunately, many of the benefits of the pub/sub paradigm

come at the cost of increased exposure of the data being transmitted. For example, in the traditional pub/sub paradigm, brokers filter events based on subscriber interest. In order to match a received event to an interested subscriber, it follows that a broker must know about the subscriber's interests to determine if it is indeed interested in the event and it must know the event's attributes in order to verify that they do indeed match the subscriber's interests. Unfortunately, depending on the application, both the event's attributes and subscriber's interests is often sensitive data that must be protected from untrusted brokers.

Authorization deals with the property of being authorized to perform various actions within the pub/sub system. More specifically, authorization covers the techniques used for arbitrarily controlling the flow of messages through pub/sub beyond the regular routing operations normally performed by a broker. In general, this means how arbitrary policy and rules dictate which entities within the system can send, receive, and – where applicable – decrypt specific messages. Using authorization techniques within the system can help mitigate against several attacks once the source is identified by limiting the propagation of the attack within the system. In addition, as with confidentiality, authorization is necessary for the adoption of pub/sub into wider application domains. Indeed, most solutions that offer confidentiality require some form of authorization. Without any control over which entities can decrypt a given message, the confidentiality techniques are trivially defeated. The reverse is not necessarily true as authorization is broader in scope than merely the control of which entities within the system can decrypt messages.

Anonymization encompasses the hiding of an individual's identity within the system. Broadly speaking the anonymity literature falls into two sub-categories. One is *communication anonymity*, which hides the identity of clients during the transmission of messages. This is desirable for users that wish to retain privacy when transmitting content over the pub/sub network, like political dissidents, whistle-blowers, etc. The other anonymity sub-category is *data anonymity*, which involves the removal of an individual's identifying information from the data itself. This is useful when auditing or otherwise analyzing a collection of data that has been transmitted through the pub/sub network. Data anonymity ensures that the data being analyzed does not reveal private or oth-

erwise sensitive information to those doing the analysis. For example, in a healthcare system that transmits patient records, it may be useful for researchers to gather data on the trends and prevalence of particular diseases without necessarily knowing the names of the patients that have the disease. Data anonymity is used to strip data of identifying information so that it may be gathered and analyzed without the identification of specific patients.

Integrity deals with the overall system integrity. This includes attacks that may adversely affect the functioning of the pub/sub system, attacks that gather information about the pub/sub system in preparation for a more complicated attack and mitigation strategies for the various attacks. The attacks we survey are not numerous, likely due to a lack of research in this area as well as the lack of a standard approach to addressing security issues rather than the overall coverage of this subject by the existing literature.

In addition, in this thesis, we present *HyShare*, a hybrid broker network that leverages recent developments in trusted execution environments to share a secret in pub/sub without the need for a service to do this for us. We extend the *Iterated Secret Share Propagation Scheme* (*ISSPS*) [123] through the use of *Intel's Software Guard Extension* (*SGX*) [61] to efficiently share a secret end-to-end in a pub/sub system. To the best of our knowledge, this is the first time trusted execution environments have been leveraged within the context of pub/sub. This work is published in [70]. The reason for extending ISSPS is due to its impracticality. ISSPS requires a prohibitively large number of messages in order to share even a single secret. The number of messages needed to share a secret grows exponentially with the path length between publishers and subscribers. By leveraging trusted execution environments, we reduce the overall number of messages needed to share a secret in pub/sub.

1.4 Contributions

In Summary, the contributions of this thesis are:

1. The categorization and detailed analysis of the pub/sub security research. Overall, we find that the literature falls into four categories: confidentiality, authorization,

anonymity and integrity. We include a thorough evaluation of what is achievable in the current state-of-the-art.

2. An identification of various limits in the current academic literature, as well as potential areas of future research. We propose a number of promising techniques which, to the best of our knowledge, have not been adapted for use with pub/sub, and which we hope will be fertile ground for future research.

3. A thorough analysis of the Iterated Secret Share Propagation Scheme [123] showing its resilience to broker collusion and its weakness in terms of the number of messages needed to share a secret. Although the scheme's resilience to broker collusion was a benefit claimed by the authors of the scheme, it had not been proven.

4. HyShare, a novel pub/sub secret sharing solution. Secret sharing allows for the sharing of sensitive data between communicating publishers and subscribers in a network of untrusted brokers, and is often an assumed first step in the confidentiality literature. To the best of our knowledge, this is the first time trusted execution environments (including Intel's Software Guard Extensions) have been used with pub/sub.

5. A detailed evaluation of HyShare showing how our solution compares to the Iterated Secret Share Propagation Scheme. HyShare achieves a significant reduction in the number of messages needed to share a secret at the cost of using a trusted attestation service when entities first connect to the pub/sub network.

1.5 Organization

The remaining chapters of this thesis are organized as follows. Chapter 2 introduces the necessary background on pub/sub, SGX, SSPS and ISSPS. Chapter 3 describes the related work. Chapter 4 details the techniques available in the academic literature for achieving confidentiality in pub/sub. Chapter 5 presents the authorization solutions in the literature. Chapter 6 details the literature on anonymity. Chapter 7 details

the research on the system integrity of pub/sub, presenting both attacks on pub/sub, and existing mitigation strategies. Chapter 8 presents HyShare along with its threat model, hybrid broker network and broker placement strategies. Chapter 9 presents the experimental results of HyShare, including the observed reduction in messages needed to share a secret when compared to ISSPS and the latencies incurred by our solution.

Chapter 2

Background

In this section we present the background material required for our work on HyShare. HyShare extends existing work on secret sharing work by leveraging recent developments in trusted execution environments. Specifically, our implementation makes use of SGX, Shamir's Secret Sharing Scheme and the *Iterative Secret Share Propagation Scheme* (*ISSPS*). Since the iterative version logically follows from non-iterative version, we also present the *Secret Share Propagation Scheme* (*SSPS*). Both SSPS and ISSPS are also relevant to Chapter 4, on confidentiality.

2.1 Publish/Subscribe

Publish/subscribe is a communication paradigm where data sources (*publishers*) and data sinks (*subscribers*) communicate through messages (*events*). Events are disseminated by *brokers*, which route all events from publishers to a subset of interested subscribers. We collectively refer to publishers, subscribers and brokers as *entities*. The set of all publishers and subscribers is referred to as *clients*. The *broker network* refers to the set of all brokers.

Publishers indicate their intent to publish a stream of events through *advertisements*. Advertisements are propagated through the broker network. A subscriber expresses its interest in an event stream by *subscribing* to events. In *topic-based pub/sub*, advertisements indicate the event stream's topic which is used by subscriptions to identify a

particular event stream. In *content-based pub/sub*, events contain attribute-value pairs. Subscribers can further filter the events they receive by specifying boolean predicates over the attribute's domain of values they are interested in within subscriptions. Events also have a payload in both topic-based and content-based pub/sub.

Brokers are responsible for routing events from publishers to interested subscribers. A broker that is directly connected to any client is referred to as an *edge broker*. *Brokerless pub/sub* is an alternate pub/sub design that does not use brokers. Instead, clients form their own network and route all messages among themselves.

Pub/Sub is a desirable communication paradigm due to its decoupling in space, time and synchronization [30]. *Space decoupling* refers to the property of clients not needing to know or even be aware of one another. Publishers do not hold references to subscribers and vice versa. In Brokerless pub/sub, while clients are aware of their direct neighbours, they are not aware of which clients are the senders or receivers of events. *Time decoupling* refers to the property that the interacting clients do not need to be actively participating in the interaction at the same time. Publishers may publish an event while some subscriber is offline and likewise a subscriber may subscribe to an event stream while its publisher is offline. *Synchronization decoupling* refers to the property of clients not being blocked when publishing or receiving an event.

For the purposes of our secret sharing work, issues regarding access control are considered orthogonal to our secret sharing work. That is, we assume that any subscriber that can subscribe to events is authorized to receive them and that any publisher that can advertise an event is authorized to publish said event. As with the existing literature, we do not consider the issue of subscribers leaking secrets. We also assume that hop-to-hop communication channels are secure.

The literature we survey commonly uses the honest-but-curious threat model where all brokers do not deviate from the system specifications and protocols. Unless explicitly stated, we will do so as well. Our work on secret sharing considers a stronger threat model composed of honest-but-curious brokers and some number of brokers that behave arbitrarily by altering or dropping messages. We revisit this stronger threat model in Section 8.1.

2.2 Intel's Software Guard Extensions

Secure remote computation is the problem of executing software on a remote computer owned and maintained by an untrusted party, with integrity and confidentiality guarantees. SGX aims to solve this problem by leveraging trusted hardware in the remote computer. The trusted hardware establishes a secure container (referred to as an *enclave*) and the remote computation service user uploads the desired computation and data into enclaves. The trusted hardware protects the data's confidentiality and integrity while the computation executing [20].

SGX itself is a set of extensions to the Intel architecture that aims to provide integrity and confidentiality guarantees to security sensitive computation performed on a computer where all the privileged software (operating system, kernel, hypervisor, etc.) is potentially malicious [20]. SGX makes use of *attestation*, which proves to a user that they are communicating with a specific piece of software running in an enclave hosted by the trusted hardware.

We consider availability attacks on enclaves to be outside the scope of this thesis. Similarly, we assume that SGX is secure and do not consider side-channel attacks nor any other potential vulnerability of the SGX technology, including its cryptographic operations.

2.2.1 Attestation

Here, we present a high level description of the attestation protocol used by SGX. Attestation proves to a remote computer that it is communicating with a specific secure container hosted by a trusted platform [20]. Part of the protocol involves a Diffie-Hellman Key Exchange between the enclave and the remote computer that establishes a *shared key*. The shared key can be used to encrypt and decrypt messages sent to and from the enclave.

The attestation protocol is initiated by the remote computer in the form of a challenge to the enclave. In response, the enclave generates a cryptographic signature that certifies the hash of the enclave's contents (called an *attestation key*). The remote computer can

verify that the attestation key could only be generated by an enclave running the proper code by verifying the attestation key's validity by using the *Intel Attestation Service* (*IAS*) [47].

2.3 Secret Sharing in Publish/Subscribe

In this section we present Shamir's secret sharing scheme, the *Secret Share Propagation Scheme* (*SSPS*) and the *Iterated Secret Share Propagation Scheme* (*ISSPS*).

2.3.1 Shamir's Secret Sharing Scheme

The (k, n) secret sharing scheme introduced by Shamir [95] has two input parameters. The scheme splits a secret value D into n fragments, at least k of which are required to recover D. Without loss of generality, we assume D is an integer. The scheme is based on polynomial interpolation, where access to sufficient data points yields a valid polynomial that satisfies the available data. Conversely, insufficient data points yield an infinite number of polynomials, each one equally to be valid.

The following steps can be used to split a secret value D into n fragments:

1. Take a polynomial of degree $k-1$ in the form $q(x) = a_0 + a_1 x + ... + a_{k-1}x^{k-1}$.

2. Set coefficients a_i where $0 < i < k$ to random values (modulo a large prime).

3. Set a_0 to the original secret value D.

4. Decompose D into n fragments by evaluating:

$$D_1 = q(1), ..., D_i = q(i), ..., D_n = q(n)$$

To regenerate the original secret value D from any subset of k fragments, it is a matter of finding all coefficients a_i in $q(x)$ and evaluating $q(0)$. Solving this problem is equivalent to solving for k unknowns using k equations.

In this document, we use function $ssSplit(k, n, D)$ to denote the process of generating n fragments from a secret value D, at least k of which are required to regenerate D again using the (k, n) secret sharing scheme. This function outputs array $[D_0, D_1, ..., D_i, ..., D_n]$ where D_i represents the random polynomial evaluated at i for $0 \leq i \leq n$. Note that $D_0 = D$. We assume that $1 < k \leq n$. We also use function $ssJoin(F)$ to denote the process of regenerating the original secret D given a set of fragments F, where $|F| \geq k$. We make the simplifying assumption that $ssJoin(F)$ outputs D for both SSPS and ISSPS schemes depending on which one is being used in context, and despite the fact that the latter requires an additional number of fragments.

2.3.2 Secret Share Propagation Scheme

The *Secret Share Propagation Scheme* (*SSPS*) [123] transfers a secret from a publisher to a subscriber in the presence of honest-but-curious brokers and some number of Byzantine brokers. Publishers generate n fragments from an initial secret D, at least k of which are necessary to regenerate D using the (k, n) secret sharing scheme. Subscribers that receive at least k of these fragments are able to regenerate D. The fragments are transferred by the broker network in such a way that no broker ever has access to k or more fragments. This keeps D secure from the broker network.

To ensure that no broker ever has access to k or more fragments, the broker network is arranged in a topology with redundant paths from publisher to subscribers, limiting the broker overlap between paths. In other words, the redundant paths must have the following property: the intersection of the set of brokers belonging to each path from a given publisher to a given subscriber must be empty. We refer to paths where this property holds as *parallel paths*. This property is stronger than what is needed to securely share a secret using SSPS, as a broker may be in multiple paths without compromising D. The advantage of defining parallel paths in this way allows us to easily determine the number of compromised brokers supported by the system. Overall, up to $k - 1$ brokers may deviate from the system specifications and protocols by colluding and sharing fragments without compromising D.

Any starting topology can be transformed into one with an arbitrary number of

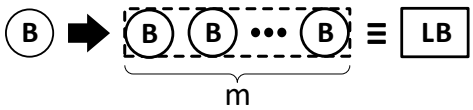


Figure 2.1: Transforming a single physical broker into a logical broker composed of m logically coupled physical brokers.

parallel paths by adding a sufficient amount of brokers. For each broker in the starting topology, replace it with a logical broker. A *logical broker* (LB) is a logical abstraction composed of m *physical brokers* (PB), where $k \leq m$. Usually, $m \leq n$. Note that we use the terms physical broker and broker interchangeably, preferring to use physical broker only when the emphasis adds clarity. We show this transformation on a single broker in Figure 2.1. If a broker in the starting topology is connected to an entity, then all brokers within an LB are connected to the entity in the new topology. This transformation immediately produces m parallel paths from any starting topology. We show such a transformation for a single publisher, broker and subscriber with $m = 3$ in Figure 2.2. Figure 2.3 illustrates an example of SSPS for $n = m = 3$ showing only a single publisher, a single subscriber and the subset of brokers that are directly involved in the transmission of all fragment for a single secret.

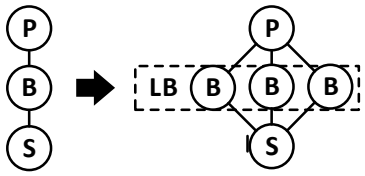


Figure 2.2: A simple topology transformation example for generating 3 parallel paths from a single path.

By adjusting k and n, a system administrator can adjust the number of tolerable colluding and Byzantine brokers in the broker network at the cost of a higher number of required parallel paths from publishers to subscribers. As n increases relative to k, a greater amount of redundant fragments are generated and transmitted, which increases the system's tolerance to dropped messages. Additionally, up to $k-1$ brokers may collude and share their fragments with one another without compromising D. Unfortunately, a single colluding broker in each of the k distinct parallel paths (i.e., k colluding brokers) is enough to compromise D.

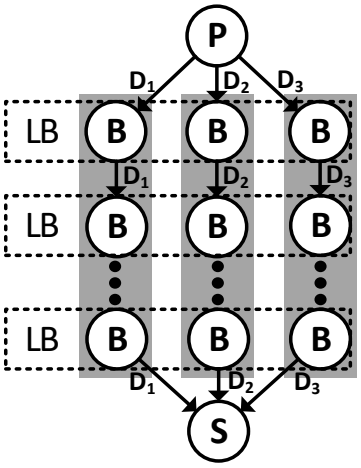


Figure 2.3: The propagation of fragments in SSPS for $n = m = 3$, with the parallel paths highlighted.

2.3.3 Iterative Secret Share Propagation Scheme

The *Iterative Secret Share Propagation Scheme* (*ISSPS*) [123] strengthens the problem of colluding brokers in multiple parallel paths. Although it does not improve on the worst case scenario, it changes which brokers must collude in order to compromise D. In SSPS, if k parallel paths contain a colluding broker then D is compromised. In ISSPS, the colluding brokers must all belong to a single LB in order for D to be compromised. Even if $k - 1$ brokers within all LBs involved in the sharing of a secret collude, they would not be able to compromise D. This comes at the cost of an explosion in the number of messages needed to share a secret. Overall, with ISSPS, the number of messages needed to share a secret scales exponentially with the number of hops between publisher and subscriber.

In ISSPS, each broker applies the (k, n) secret sharing scheme with each received fragment as if it where the secret D. The newly generated fragments are sent to the next hop LB, with each parallel path receiving a different fragment. To regenerate the original secret, the subscriber must regenerate each intermediate fragment. Figure 2.4 shows the dissemination of messages in ISSPS for $n = m = 3$ showing only the source publisher, sink subscriber and the subset of brokers that are involved in the transmission of any fragment for a single secret. Each broker-to-broker hop generates new fragments, each of which gets sent to a different physical broker within an LB. In ISSPS, even if a

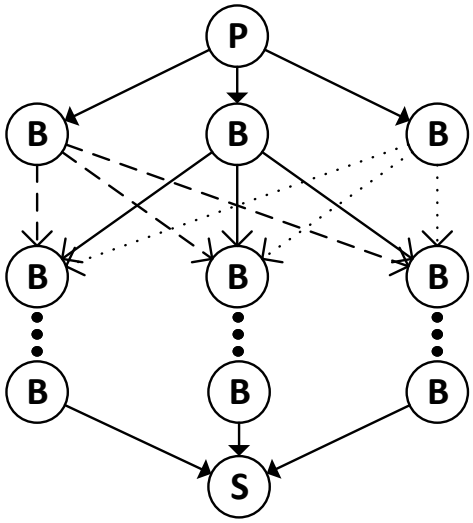


Figure 2.4: The propagation of fragments generated in ISSPS with $n = m = 3$.

single broker in each parallel path is compromised, it does not necessarily mean that the initial secret D is compromised. For this to happen, all brokers within an LB must be compromised. ISSPS is costly in the total number of messages that must be transmitted to share a single secret. If h denotes the average number of hops between publishers and subscribers then the number of fragments needed to share a secret is n^h. Figure 2.5 shows the magnitude of this problem for various n. The total number of messages needed to share a secret is even greater, as it requires n^h messages in the last hop alone of the transmission of a secret.

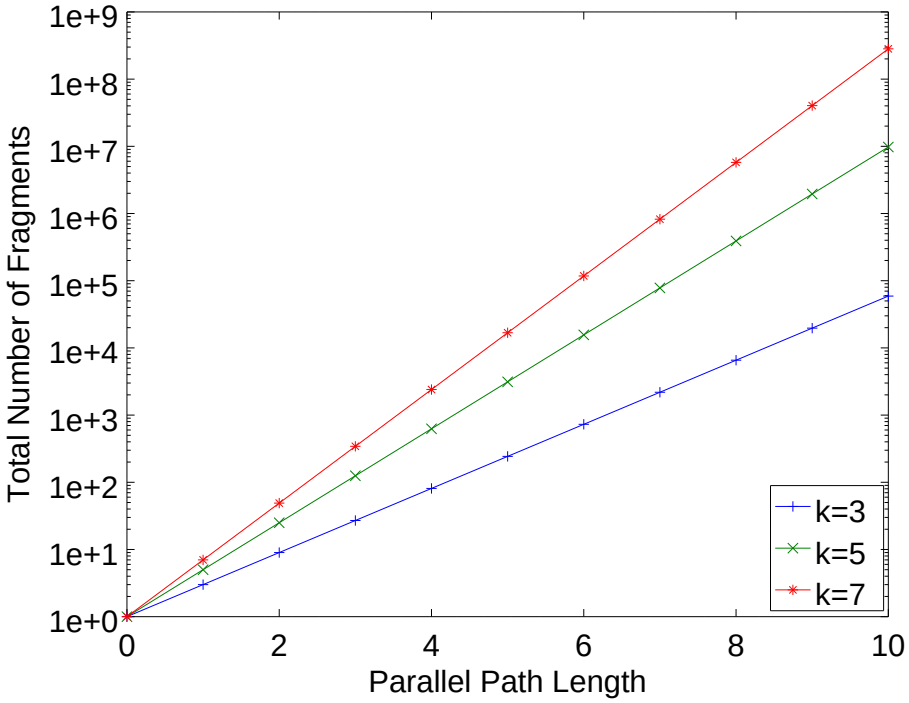


Figure 2.5: Number of fragments received by a subscriber on a log scale for the sharing of a single secret as the path length increases.

Chapter 3

Related Work

To the best of our knowledge, the need for confidentiality and privacy in pub/sub was first identified by Wang et al. [114]. The authors argue that the standard encryption and cryptographic techniques used to guarantee confidentiality outside of pub/sub are not directly applicable to pub/sub. This is due to the fact that brokers have the seemingly paradoxical task of routing events based on their confidential attributes. This is especially challenging in content-based pub/sub, where brokers must match events based on private event attribute-value pairs with private subscriptions. There have been a number of approaches developed since the release of that paper to do precisely this kind of matching in both topic-based and content-based pub/sub.

There has been a growing amount of academic research dedicated to achieving security in pub/sub. This large volume of work has never been categorized, summarized and presented as a whole before. Onica et al. [76] have constructed their own survey on confidentiality in pub/sub. That survey has some overlap with our own work, and it is best compared to our chapters on confidentiality (Chapter 4) and authorization (Chapter 5). Onica et al. [76] focus on a several distinct confidentiality and authorization techniques and go into great depth regarding each one. Instead, our own work focuses on breadth, covering a larger amount of the academic literature. In Chapter 4, out of the major papers covered (shown in Table 4.1), six of the entries (37.5%) do not appear in the survey. Similarly, in Chapter 5, out of the major papers we cover (shown in Table 5.1), four of the entries (40%) are not covered by the survey. In addition, our own work

is broader in scope, including additional developments in anonymity (Chapter 6) and integrity (Chapter 7) which are not in the survey.

HyShare is best compared with the confidentiality techniques that we present in Chapter 4. Although we compare these techniques to HyShare here, we do not reintroduce them as it would be redundant. These techniques include symmetric-key encryption [59, 89, 16, 24], homomorphic cryptosystems [71, 55], asymmetric scalar-product preserving encryption [18, 75], multiple layer commutative encryption [98], oblivious transfer [21] and attribute-based encryption [107, 49, 48, 124, 109]. A common thread to every solution we've observed in the literature is the need for an unilaterally trusted service that exists independently of the broker network (and the pub/sub system as a whole) but is still connected to all clients in order to disseminate keys, security parameters or some other form of shared secret among communicating clients before the various mechanisms can be applied. The need for this service is the primary problems addressed by HyShare. In addition, the existing solutions operate under the weaker honest-but-curious threat model. To the best of our knowledge, Yoon and Kim [123] (the work we enhance) is the first and only work in the literature to consider the dissemination of secrets through the existing broker network. Furthermore, they do so under stronger adversarial models than the honest-but-curious threat model, which we inherit by enhancing their work. Unfortunately, the number of messages their scheme needs to share a secret grows exponentially with the number of broker hops between clients. This is another problem that we address with HyShare. Our work can be considered supplementary to the aforementioned literature for sharing the initially required secrets or as a stand alone solution to disseminate all events within the system.

Content-based routing using SGX has been proposed before by Pires et al. [85]. The authors describe *SCBR*, a secure content-based routing engine that uses SGX for matching sensitive attributes within an enclave. SCBR differs from HyShare in that it is limited to single-broker systems. In addition, SCBR is restricted to a homogeneous hardware stack consisting of Intel hardware with SGX support.

Shamir's secret sharing scheme has been applied to pub/sub before by Minami et al. [67], but for a different use case. The authors secure the aggregation of various events

generated by various publishers. Our work applies Shamir's secret sharing scheme to solve the problem of secure dissemination of secrets through the pub/sub network. Shamir's secret sharing scheme has been used extensively in the broader context of security and distributed systems [28, 68, 11].

Chapter 4

Confidentiality

Confidentiality is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes [52]. This property is critical for the adoption of pub/sub systems in applications that must disseminate private or otherwise sensitive information. Unfortunately, many of the benefits of the pub/sub paradigm come at the cost of increased exposure of the data being transmitted. Honest-but-curious brokers must route messages based on their (potentially sensitive) content. In this section, we present the literature available for guaranteeing confidentiality in pub/sub systems in the presence of such brokers. We limit the techniques presented to those that have specifically been adopted for use with pub/sub systems. The focus of this chapter is on the techniques and mechanisms used to guarantee the privacy of information being disseminated through the system. Issues of authorization, access control and access rights are addressed in Chapter 5. We conclude this chapter with a discussion on key management techniques, which are used in tandem with other confidentiality solutions.

The standard encryption techniques and mechanisms used for achieving confidentiality in one-to-one communication (e.g., AES, RSA) cannot be directly applied to the pub/sub paradigm. The space decoupling property of pub/sub makes the sharing of secrets or keys between clients difficult since they must operate independently of one another. Additionally, if each client has its own key then a publisher would have to encrypt and send an event for each interested subscriber, bypassing the benefits introduced from broadcasting messages using the pub/sub paradigm. Another problem is that it is

difficult for brokers to route events based on their attributes (i.e., content-based routing) if the brokers are not trusted with the information contained within events (i.e., brokers are honest-but-curious). To ensure confidentiality, brokers have the seemingly paradoxical task of routing events to interested subscribers without being able to discern the event's attributes nor the interest expressed by any subscription. Subscription covering and other routing optimization techniques would be similarly difficult.

Since equality matching is preserved when using common encryption techniques (i.e., $m_1 = m_2 \iff E(m_1) = E(m_2)$), it is much easier to achieve topic-based routing where encrypted subscription's interests can be directly checked against individually encrypted event attributes. This is why most of the techniques in the literature seek to address the problem of confidentiality preserving content-based routing techniques. Specifically, this chapter focuses on the techniques available to secure the confidentiality of messages being transmitted through a pub/sub system with honest-but-curious brokers and unauthorized subscribers. Chapter 5 differs from this section by focusing instead on access control.

The need for confidentiality is dependent on the application domain. For some domains, confidentiality is absolutely critical due to the nature and sensitivity of the data being disseminated. For example, healthcare data is not only sensitive but often its confidentiality is mandated through government legislation and regulation. Without confidentiality, pub/sub could not be seriously considered as the mechanism for the dissemination of this kind of information.

Healthcare is a common motivating application for the need for confidentiality in pub/sub [50, 49, 103, 102]. Within this context, the events being generated contain patient data with details regarding the prescriptions they take, the results of medical exams, newly detected medical conditions, etc. Subscribers may be various medical entities like the patient's general practitioner, nursing home, medical researchers, auditors, etc.

Other applications with relevance to confidentiality that have been considered in the literature include payment systems [114], geolocation based services [78] and financial markets [114, 125]. Payment systems are those where subscribers pay for events generated by publishers. Subscribers pay for events that match their subscription and are delivered to them. Confidentiality is used to ensure only paying customers can receive

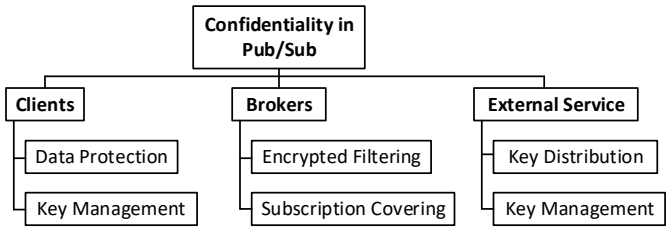


Figure 4.1: Components that are addressed by pub/sub confidentiality solutions.

and view events. Geolocation services are those where users publish events with their locations to the system, and subscribers provide services to those users with proximity to their location. In this case, users want to keep their location private to prying eyes. Financial systems involve stock quote dissemination systems where improper or insufficient privacy might leak a potential customer's investment strategy to an adversary resulting in financial loss.

In this chapter, we focus the confidentiality of the messages defined by the classic pub/sub paradigm: advertisements, events, subscriptions, unadvertisements and unsubscriptions. Pub/sub systems often have other forms of messages including heartbeats, control messages, etc. We do not discuss these latter messages as they are implementation dependent and their confidentiality requirements may differ widely from one another. The need for confidentiality in the former set of messages can be expressed through two terms: *Information Confidentiality* (IC) and *Subscription Confidentiality* (SC) [114]. IC refers to the information being sent by publishers (i.e., advertisements, events and unadvertisements), and specifically addresses whether the infrastructure can perform the necessary routing without the publishers trusting brokers with the content of their messages. SC refers to the information being sent by subscribers (i.e., subscriptions and unsubscriptions), and specifically addresses whether subscriber can obtain data from the pub/sub system without revealing their subscription interests to the publishers or the brokers. In both cases, the information contained within these messages may reveal sensitive information about clients using the system. Figure 4.1 lists the essential components that must be considered when developing a confidentiality solution to pub/sub. Clients are responsible for the protection of the data sent through the broker network and might be involved in the key management mechanism implemented. Brokers must be able to

filter events to interested subscribers without any knowledge of the sensitive data events or subscriptions may contain. Although strictly speaking optional, subscription covering is a routing optimization that should be addressed in order to have a scalable pub/sub implementation. Finally, the literature often makes use of an external service to handle key distribution and management. This service tends to be trusted.

Table 4.1 summarizes the solutions covered in this chapter. Categorized by the major technique used in each solution, we also present the kind of routing supported (topic-based or content-based) and whether the pub/sub is brokerless. We also present whether the solution supports equality (=), range and string prefix filtering by the brokers. For space reasons, we omit other forms of filtering like string suffix filtering though a few of the solutions presented do support them. The last column presents whether a solution supports subscription covering or not. Although this technique is technically optional in terms of confidentiality, it is often necessary in terms of performance.

4.1 Symmetric-key Encryption

Overview

Symmetric-key encryption is a common technique for encryption where a single key is used for both encryption and decryption. Symmetric-key encryption has been used in both topic-based and content-based pub/sub systems by encrypting events and subscriptions to prevent brokers from learning the content of events or any subscriber's interests when performing routing operations.

Symmetric-key encryption operations tend to be faster than asymmetric-key ones due to the fact that only one key needs to be generated and managed. If the key is leaked or otherwise compromised then all data encrypted with that key is compromised as well. One technique for limiting the damage caused by the compromise of a key is to only use a key for some period of time and then discard it. A new key must be generated and distributed to all entities that need it after each time period elapses. The literature presented considers the problems of key generation, management and dissemination to

Table 4.1: Summary of the confidentiality issues addressed by each paper

Technique	**Paper**	**Routing**	**Brokerless**	**SC**	**IC**	**=**	**Range**	**Prefix**	**Covering**
Symmetric-key Encryption	Crescenzo et al. [24]	Topic	✗	✓	✓	✓	-	-	✓
	Chen et al. [16]	Content	✗	✓	✓	✓	✓	✗	✗
	Raiciu and Rosenblum [89]	Content	✗	✓	✓	✓	✓	✓	✓
	Li et al. [59]	Content	✗	✓	✓	✓	✓	✓	✗
Homomorphic Cryptosystems	Nabeel et al. [71]	Content	✗	✓	✓	✓	✓	✗	✗
	Klonowski and Rózanski [55]	Topic	✓	✗	✓	✓	-	-	✓
ASPE	Onica et al. [75]	Content	✗	✓	✓	✓	✓	✗	✓
	Choi et al. [18]	Content	✗	✓	✓	✓	✓	✗	✓
Functional Encryption	Tariq et al. [109]	Content	✓	✗	✓	✓	✓	✓	✓
	Ion et al. [49, 48]	Content	✗	✗	✓	✓	✓	✓	✗
	Pal et al. [81]	Content	✗	✓	✓	✓	✓	✗	✗
	Srivatsa et al. [107]	Content	✗	✗	✗	✗	✗	✗	✗
	Yuen et al. [124]	Content	✗	✗	✗	✗	✗	✗	✗
Multiple Layer Commutative Encryption	Shikfa et al. [98]	Content	✗	✓	✓	✓	✗	✗	✓
Oblivious Transfer	Crescenzo et al. [21]	Topic	✓	✗	✓	✓	-	-	✗
Secret Sharing	Yoon and Kim [123]	Content	✗	✗	✓	✗	✗	✗	✗

be orthogonal to the solutions presented.

There are three solutions that use symmetric-key encryption for securing events and subscriptions. Crescenzo et al. [24] support equality matching and use this to implement a topic-based filtering. Chen et al. [16] use simple and efficient encryption techniques to implement content-based routing. Unfortunately, this efficiency comes at some security cost since the authors' scheme is vulnerable to known plaintext attacks. Li et al. [59] also implement content-based routing by constructing a scheme that supports prefix filtering. Raiciu and Rosenblum [89] adapt a protocol proposed by Goh [36] to perform keyword matching using Bloom filters and propose additional protocols for supporting range matching using dictionaries. These solutions require the sharing of secrets and the establishment of security parameters between publishers and subscriber. Although often there is no mention of how this is achieved while preserving space decoupling, similar solutions use a trusted third party entity to aid during this initial set-up phase without violating space decoupling.

Filtering

Filtering in topic-based pub/sub is fairly trivial to implement since the equality relationship between two plaintext messages is preserved between their respective ciphertexts. This is sufficient for the implementation of basic topic-based filtering, where routers check for a match between an event's topic and the topic specified in the subscription [24]. Chen et al. [16] mask the topic by first calculating the XOR of the unique topic identifier and a shared key and then applying a one-way hashing function. The key is used in order to prevent the brute force attack of hashing every possible topic.

Content-based filtering requires the support of granular filtering based on the contents of the event, beyond the matching provided by topic-based filtering. In the literature, symmetric-key encryption is used to support range filtering [16] and prefix filtering [59].

Given that $x > y \iff x + k > y + k$, a randomly generated secret k which is shared by clients can be generated and added to the event's attribute and the subscriber's filter without altering the ordering of the attributes. Range filtering is thus possible through a shared key k [16]. This scheme is vulnerable to known plaintext attacks.

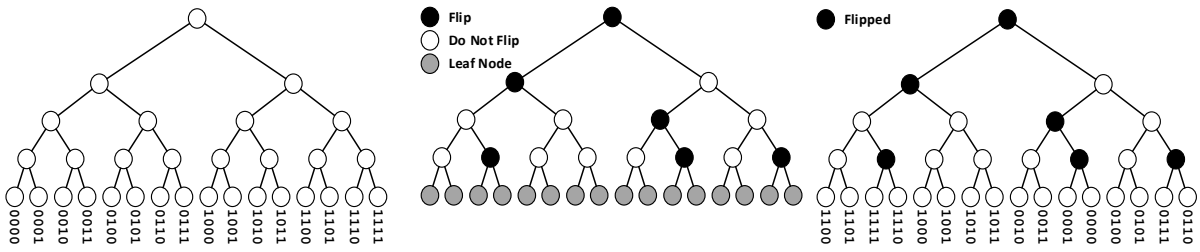
Li et al. [59] have adapted a prefix-preserving encryption scheme for use with pub/sub. Through this scheme, it is possible to perform equality, range and prefix filtering. Some of the key insights of this paper are that the problem of range filtering can be reduced to prefix filtering and that the prefix-preserving scheme introduced by Xu et al. [121] can be adapted for use with pub/sub.

Algorithm 1 Decomposition of a range $[a, b]$ into a set of prefixes. Note that $a \leq b$ for the original input variables and inputs are always of the same length.

1: **procedure** RANGETOPREFIX(a, b)
2: $\Delta \leftarrow min_{i \in \mathcal{N},\ i>0}(a_i < b_i)$
3: **if** $\nexists\ \Delta$ **then**
4: **return** a
5: **end if**
6: **if** $a_\Delta ... a_n = 00...0\ \wedge\ b_\Delta ... b_n = 11...1$ **then**
7: **return** $a_1 ... a_{\Delta-1}*$ ▷ Returns $*$ if $\Delta = 1$
8: **end if**
9: $A \leftarrow$ RANGETOPREFIX($a_{\Delta+1} ... a_n, 11...1$)
10: Prepend $a_1 ... a_{\Delta-1}0$ to all prefixes in A
11: $B \leftarrow$ RANGETOPREFIX($00...0, b_{\Delta+1} ... b_n$)
12: Prepend $a_1 ... a_{\Delta-1}1$ to all prefixes in B
13: **return** $A \cup B$
14: **end procedure**

Range filtering can be reduced to prefix filtering. For example, the interval $[00100000, 01101111]$ (the 8-bit binary representation of $[32, 111]$) can be represented as a set of prefixes $\{001*, 010*, 0110*\}$. The $*$ is used to denote an arbitrary suffix. It is possible to decompose an arbitrary range $[a, b]$ where a and b are two binary numbers with the same number of digits n through the use of Algorithm 1. We use the notation a_i to denote the i^{th} digit of a and likewise b_i to refer to the i^{th} digit of b (note that $a = a_1 a_2 ... a_n$ and $b = b_1 b_2 ... b_n$). Furthermore, we assume that $[a, b]$ is a valid range (i.e., $a \leq b$).

Algorithm 1 recursively decomposes the input range $[a, b]$ into a set of prefixes. When



(a) Plaintext tree (b) Encryption function (c) Ciphertext tree

Figure 4.2: An example of prefix-preserving encryption.

evaluated, these prefixes effectively match any number within the original input range. If the range is valid then a set of equivalent prefixes also exists. The algorithm finds the most significant digit where a and b differ (denoted as Δ) in line 2. Δ exists if and only if $a \neq b$. There are two cases where a prefix is immediately apparent. The first is that Δ does not exist (i.e., $a = b$), in which case the prefix is the full input a (lines 3 – 5). The second case is when the full range of values is covered by the digits following Δ, in which case $a_1...a_{\Delta-1}*$ forms a prefix (lines 6 – 8). Otherwise, the problem is split into two and solved recursively. By definition, $a_\Delta = 0$ and $b_\Delta = 1$. Therefore, the range $[a_1...a_{\Delta-1}0a_{\Delta+1}...a_n, a_1...a_{\Delta-1}011...1]$ must be within the original input range $[a, b]$ since the lower bounds are equal to a and the upper bounds are both less than or equal to b. Similarly, $[a_1...a_{\Delta-1}100...0, b]$ is within the original input range $[a, b]$ since the lower bounds are less than or equal to b while the upper bound is exactly b. The union of these two ranges is equal to the full input range $[a, b]$ (i.e., $[a_1...a_{\Delta-1}0a_{\Delta+1}...a_n, a_1...a_{\Delta-1}011...1] \cup [a_1...a_{\Delta-1}100...0, b] = [a, b]$). Splitting the ranges in this manner puts it into a convenient form for the algorithm to recursively find all prefixes (lines 9 – 13).

It is interesting to note that the number of prefixes in the returned set from Algorithm 1 scales with the number of digits in the input. The size of the prefix set will be less than or equal to $2(n-1)$ [59]. This is a tight bound as interval $[1, 2^n - 1]$ has exactly $2(n-1)$ prefixes in the set.

The prefix-preserving encryption proposed by Li et al. [59] for use with the aforementioned prefix matching uses a symmetric prefix-preserving encryption function shared by publishers and subscribers. We say that two n-bit numbers $a = a_1a_2...a_n$ and $b = b_1b_2...b_n$

share a k-bit prefix ($0 \leq k \leq k$), if $a_1a_2...a_k = b_1b_2...b_k$ and $a_{k+1} \neq b_{k+1}$ when $k < n$. An encryption function E is said to be prefix-preserving, if, given two numbers a and b that share a k-bit prefix, $E(a)$ and $E(b)$ also share a k-bit prefix.

It is useful to consider a geometric interpretation of the prefix-preserving encryption function. If a plaintext can take any value of an n-bit number, the entire set of plaintexts can be represented by a complete binary tree of height n, where each node in the tree (excluding the root node) corresponds to a digit value. This is called the plaintext tree, an example of which is shown in Figure 4.2(a) for $n = 4$.

Algorithm 2 Encryption algorithm from Li et al. [59].

1: **for** $i \leftarrow 1, n$ **do**
2: $\quad a'_i \leftarrow a_i \oplus f_{i-1}(a_1a_2...a_n)$
3: **end for**
4: **return** $a'_1a'_2...a'_n$

Algorithm 3 Decryption algorithm from Li et al. [59].

1: **for** $i \leftarrow 1, n$ **do**
2: $\quad a_i \leftarrow a'_i \oplus f_{i-1}(a_1a_2...a_n)$
3: **end for**
4: **return** $a_1a_2...a_n$

A prefix-preserving encryption function can be viewed as specifying a binary variable for each non-leaf node (including the root node) of the plaintext tree. This variable specifies whether the encryption function flips that bit or not. Applying the encryption function results in the rearrangement of the plaintext tree into a ciphertext tree. Figure 4.2(c) shows the ciphertext tree resulting from the encryption function shown in Figure 4.2(b). Note that an encryption function will, therefore, consist of $2^n - 1$ binary variables. In general, the prefix-preserving encryption function f_i maps $\{0,1\}^i$ to $\{0,1\}$, for $i = 1, 2, ..., n-1$ and f_0 is a constant function. Given plaintext $a = a_1a_2...a_n$, the ciphertext $a'_1a'_2...a'_n$ is computed using Algorithm 2. Conversely, Algorithm 3 shows the reverse operation, decrypting the plaintext from the ciphertext.

Raiciu and Rosenblum [89] perform keyword matching by having publishers and subscribers share security parameters. These parameters are used in the hashing of strings. To subscribe, keywords are divided into fixed sized strings called *words* which are then hashed. This collection of hashed *words* becomes the subscription that gets sent to brokers. When publishing an event, brokers perform the same steps and insert the hashed

words into a Bloom filter with the desired false positive rate. This is what is sent to the brokers. When filtering events, a broker simply has to match the presence of hashed *words* in the Bloom filter. This solution supports subscription covering by checking if two subscriptions are equal to one another.

Alternatively, Raiciu and Rosenblum [89] also propose the use of dictionaries instead of Bloom filters for keyword matching. Subscribers send the indices of where the *words* they are interested in hash to as part of their subscription. Publishers construct a dictionary with the value of an event's *word* set to 1. The broker can then filter events by checking the dictionary indices. Although this approach does not generate false positive matches, depending on the number of possible keywords, the size of the dictionary can be very large. The size of the encrypted notification is 32kB for the English language [89].

There are two proposed methods by Raiciu and Rosenblum [89] for supporting range filtering of numeric attributes using dictionaries. In the first method, l points $p_1, p_2, ..., p_l$ are chosen to split the domain of an event attribute. The rest of the protocol is similar to keyword matching, except the *words* used are from the set: $\{\text{“} > p_1\text{”}, \text{“} > p_2\text{”}, ..., \text{“} > p_l\text{”}, \text{“} < p_1\text{”}, \text{“} < p_2\text{”}, ..., \text{“} < p_l\text{”}\}$. Event attributes and subscriptions are approximated with these constraints. All *words* that match the attribute's value are inserted into the dictionary. Subscriptions send the *words* that most accurately approximate their actual interest. Only two of these attributes are sufficient to upper and lower bound a range of interests. This method requires that publishers and subscribers agree on the points and has a trade-off between the number of points chosen (which directly affects the size of the published dictionary), and the accuracy of the range filtering. The domain does not need to be partitioned by the chosen points, with exponentially spaced partitioning schemes suggested by Raiciu and Rosenblum [89] if the range of values is very large.

The second method for range filtering proposed by Raiciu and Rosenblum [89] for supporting range filtering of numeric attributes using dictionaries encodes various partitions of the attribute space with differing starting indices, which are used to approximate subscription filters. This method is a trade-off between the size of the subscriptions and matching time on one hand and the number of false positives and security attained (i.e.,

information leaked due to imprecise subscriptions), on the other.

4.2 Homomorphic Cryptosystems

Overview

Homomorphic cryptosystems allow for the encryption of data while still allowing for some computations to be performed on the cyphertext that are analogous to performing these computations on the plaintext itself. These computations do not require knowledge of the plaintext, and are carried out while the data is still encrypted. There are two cryptosystems that have been used in pub/sub: Paillier and ElGamal.

The Paillier homomorphic cryptosystem [80] has been adapted for use with pub/sub by Nabeel et al. [71]. The authors use this cryptosystem for encrypting the attributes of an event so that useful computation can be done on encrypted values, achieving useful filtering without trusting the honest-but-curious brokers. This technique is used strictly to encrypt attributes. The event payload is encrypted using attribute-based encryption techniques further discussed in Section 4.4.

The Paillier cryptosystem has the homomorphic property of allowing for the computation of the sum of two plaintext messages by computing solely on the corresponding ciphertexts. Specifically, for two attributes m_1 and m_2 and the Paillier homomorphic encryption function E, $E(m_1 + m_2) = E(m_1) \cdot E(m_2)$. This property is leveraged to ascertain which value is larger, which allows brokers to perform range filtering. Range filtering directly supports equality filtering. For example, if we want to check that $m = c$ where c is a constant value, it is sufficient to check that $m > c - 1 \wedge m < c + 1$ (assuming 1 is the lowest possible increment for c).

Nabeel et al. [71] propose several enhancements to the Paillier cryptosystem. Part of the private key is made public without sacrificing the integrity of the scheme. The authors introduce a shift in computational complexity from decryption to encryption in order to perform faster filtering at each broker. Additionally, event attributes are blinded to minimize how much information is leaked to the brokers when preventing matching.

These techniques are presented in detail.

The ElGamal universal re-encryption scheme of Golle et al. [38] has been adapted for use with brokerless pub/sub by Klonowski and Rózanski [55]. Despite the homomorphic properties of ElGamal, these techniques are not leveraged to enhance the filtering performed at routers. The only filtering supported by this scheme is equality matching. Instead, the re-encryption scheme is used to ensure that specific nodes are involved in the transfer of a message through onion routing techniques explained in greater detail in Chapter 6.

Filtering

In general, the Paillier encryption function $E(m, r)$ takes a message m and random value r and produces a corresponding ciphertext. We omit parameter r and use the shorthand $E(m)$ wherever r is not relevant to the context. The homomorphic properties of the system allow for the computation of the sum of two plaintext messages by computing solely on the corresponding ciphertexts. Specifically, for two messages m_1 and m_2, $E(m_1 + m_2) = E(m_1) \cdot E(m_2)$.

In the solution presented by Nabeel et al. [71], publishers encrypt each attribute in an event separately using the Paillier encryption function. Subscribers have the additional step of multiplying each value in their interests by -1 before encryption. When a broker receives an event, it can do range matching by multiplying together the corresponding attributes in the event and subscription. Using the Paillier homomorphic property, $E(m_1) \cdot E(-m_2) = E(m_1 - m_2)$. By decrypting the result of the multiplication, the broker can learn the difference of the two values and route the event accordingly. Nabeel et al. [71] introduce various modifications to the Paillier homomorphic cryptosystem to ensure that brokers cannot learn the value of m_1 or m_2 by decrypting either $E(m_1)$ or $E(m_2)$, to blind the attribute so that brokers do not learn the real difference between m_1 and m_2 and to increase the performance of the cryptosystem in pub/sub systems.

In the classic Paillier cryptosystem, the public encryption key is a pair (n, g_p). The private decryption key is the pair (λ, μ). To generate these values, set $n = pq$ where p and q are two large prime numbers. Set $\lambda = lcm(p-1, q-1)$ where lcm is the lowest common

multiple of the given parameters. A randomly selected base $g \in \mathcal{Z}^*_{n^2}$ is chosen such that the order of g_p is a multiple of n. Such a g_p can be efficiently found by randomly choosing $g_p \in \mathcal{Z}^*_{n^2}$, then verifying that $gcd(L(g_p^\lambda \pmod{n^2}), n)) = 1$, where $L(u) = (u-1)/n$ for $u \in S_n = \{u < n^2 \mid u = 1 \pmod{n}\}$.

Nabeel et al. [71] introduce a number of modifications to the classic Paillier cryptosystem for use in pub/sub networks. These modifications are summarized as follows:

1. Private key component μ is made public.
2. Computation complexity is shifted away from decryption towards encryption.
3. Event attributes are blinded to prevent information leakage to brokers and unauthorized subscribers.

In the original Paillier cryptosystem, both λ and μ are required to decrypt an encrypted message. However, μ does not need to be private since it is hard to decrypt an encrypted message knowing only μ. It can be shown that if a probabilistic polynomial time adversary can obtain λ from μ, it can solve the discrete logarithm problem. Since this problem is known to be hard, so is obtaining λ from μ [71]. Making μ public is the first modification applied to the original cryptosystem by Nabeel et al. [71].

Computational complexity is shifted away from decryption and towards encryption by raising the encryption by λ. Effectively, the new encryption function $E'(m) = E(m)^\lambda = g_p^{m\lambda} \cdot r^{n\lambda}$. This simplifies the decryption function to $D(c) = L(c \pmod{n^2}) \cdot \mu \pmod{n}$. This optimizes routing as decryption is performed at each broker for filtering purposes.

Due to the computational shift, any broker with access to the private key can decrypt event attributes to learn what they are. To fix this, Nabeel et al. [71] modify the encryption functions used by the publisher and subscriber to blind the attributes through the use of secret blinding parameters g^t and g^{-t}. A trusted third party entity responsible for disseminating keys also disseminates these parameters. Publishers use g^t while subscribers use g^{-t}. The modified encryption functions for the publisher and subscriber are $E''(m) = g^t \cdot E'(m) \pmod{n^2}$ and $E''(m) = g^{-t} \cdot E'(m) \pmod{n^2}$, respectively. Notice that while decryption of m from $E''(m)$ is hard without knowledge of the blinding parameter used, it is still possible to decrypt the difference since

$E''(m_1) \cdot E''(-m_2) = E''(m_1 - m_2) = E'(m_1 - m_2)$. In other words, since the blinding parameters cancel out, the homomorphic properties of the Paillier homomorphic cryptosystem are maintained when using the difference of two attributes blinded by a publisher and a subscriber.

Although the blinding operations introduced by Nabeel et al. [71] prevent individual attribute values from leaking to the broker, it does not prevent them from learning the difference between the publisher and subscriber attributes. The authors mitigate this by randomizing the difference between the two values. First, the domain size (l bits) is limited by setting it to a much smaller value when compared to the plaintext space of the Paillier cryptosystem, n (i.e., $0 \leq m \leq 2^l, l << n$). Therefore, if the difference $\Delta = E'(m_1 - m_2)$ (the difference between the event and subscription attributes) $0 \leq \Delta \leq 2^l \iff m_1 \geq m_2$ and $n - 2^l < \Delta \leq n \iff m_1 < m_2$. Nabeel et al. [71] randomize Δ such that $0 \leq \Delta \leq n/2 \iff m_1 \geq m_2$ and $n/2 < \Delta \leq n \iff m_1 < m_2$. To do this, the encryption functions become $E_p(m) = g^t \cdot E'(m)^{r_p} E'(r_q) \pmod{n^2}$ and $E_s(m) = g^{-t} \cdot E'(-m)^{r_p} \pmod{n^2}$ for publishers and subscribers, respectively. r_p and r_q are random values chosen to randomize the difference Δ and chosen to fall between the specified ranges. Decrypting $E_p(m_1) \cdot E_s(m_2)$ gives $r_p(m_1 - m_2) + r_q$.

The ElGamal-based universal re-encryption scheme of Golle et al. [38] has been adapted for use with brokerless pub/sub by Klonowski and Rózanski [55]. However, the homomorphic properties of ElGamal are not leveraged in the work to provide any additional filtering capabilities beyond basic equality filtering. In the work presented, the scheme is used to guarantee that messages are routed through the nodes specified by the publisher as each node must contribute in the decryption process of the message. This is leveraged to implement onion routing techniques further discussed in Chapter 6. Since in the scheme presented, the ciphertext depends on the brokers involved in the routing, one message may encrypt to different ciphertexts. In this paper, the authors have the publishers perform all the routing before sending the message.

4.3 Asymmetric Scalar-product Preserving Encryption

Overview

Asymmetric scalar-product preserving encryption (ASPE) is an encrypted matching scheme that preserves the distance between two points p_1 and p_2 after encryption but is resilient to attempts to discover what this distance is from the encrypted values themselves [116]. The actual distance information between two encrypted points remains concealed. Instead, this technique allows for the comparison of each of these points to a third point q.

Let P_1 and P_2 be the ASPE encrypted values of any points p_1 and p_2, respectively. Similarly, let Q be the ASPE encrypted value for a third point of reference q. Under the formal definition of ASPE, for $i \in 1, 2$, $p_i \cdot q = P_i \cdot q$ and $p_1 \cdot p_2 \neq P_1 \cdot P_2$. This holds even if $p_i = q$. This is achieved by encrypting points p_1 and p_2 differently than the reference point q [116].

If subscribers are responsible for the ASPE encryption of their own subscription attributes then the system can become vulnerable to a collusion attack along with brokers. A workaround for this is to not have subscribers encrypt attributes. Instead, a trusted security manager is used to encrypt these attributes and prevent such a vulnerability [18]. The work by Onica et al. [75] further hardens ASPE against known plaintext attacks. Onica et al. [75] also go on to enhance the original scheme with in-broker subscription re-encryption when updating the encryption keys used. Performing a key update operation in pub/sub will usually entail a spike in network usage as subscribers simultaneously need to re-send their newly re-encrypted subscriptions interfering with the responsiveness of the system and requires that subscribers remember old keys as events encrypted with older keys may have been delayed during transmission. In-broker subscription re-encryption allows for brokers to change the encryption key of an existing message, without the need for the entire message to be re-transmitted.

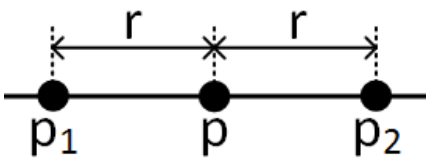


Figure 4.3: An ASPE subscription example.

Filtering

ASPE can be incorporated into pub/sub to provide content-based routing. This supports equality, inequality and range filtering [18]. Subscription covering is supported as well, an often required optimization for scalability purposes that reduces the amount of filtering operations required at each broker.

The basic idea behind using ASPE for content-based filtering is as follows. If a subscriber wishes to create a subscription, they must first generate p_1 and p_2 from their actual interest of p. As shown in Figure 4.3, this is done by first selecting a random value $r > 0$ and setting $p_1 = p - r$ and $p_2 = p + r$. The respective ASPE encrypted values of P_1 and P_2 then form a part of the subscription, which must also include an operator for equality, inequality and range filtering. The event's attribute q must be ASPE encrypted by the publisher, denoted as Q.

Brokers that receive a subscription that includes P_1 and P_2 can then match events with attribute Q by doing some distance operations. Let $Dist(x, y)$ denote the distance between points x and y (irrespective of x and y being plaintext or ASPE encrypted values). Then the following properties hold:

- $q = p$ if and only if $Dist(P_1, Q) - Dist(P_2, Q) = 0$.
- $q > p$ if and only if $Dist(P_1, Q) - Dist(P_2, Q) > 0$.
- $q < p$ if and only if $Dist(P_1, Q) - Dist(P_2, Q) < 0$.

It is through these properties that ASPE can support equality, inequality and range filtering on an attribute. To support conjunctions or otherwise more complicated filtering on multiple attributes, it is a matter of having the subscriber generate and send more of these points as part of its subscription.

4.4 Functional Encryption

Overview

Functional Encryption (FE) refers to a family of related encryption techniques. The techniques applied in the literature use Identity-Based Encryption (IBE), Attribute-Based Encryption (ABE) and Predicate-Based Encryption (PBE). In general, much like asymmetric encryption techniques, FE is used only for the encryption of the key used to decrypt the message payload.

In traditional public-key cryptography, a message is encrypted for a specific receiver using the receiver's public key. IBE is an alternative to this where the public key can be any arbitrary string which uniquely identifies the receiver [13, 19]. In ABE, when the sender encrypts a message, it includes the attributes that are required in order to decrypt it at the receiver. ABE can be considered an extension of IBE [13, 19, 54, 45, 32].

As mentioned, in IBE any valid string which uniquely identifies a user can be the public key of the user. For example, when Alice sends mail to Bob at *bob@company.com* she simply encrypts her message using the public key string *"bob@company.com"* [13]. A master public key acquired from a trusted entity is required by the sender to encrypt and send the message to a user with any arbitrary identity. To decrypt the message, a receiver needs to obtain a private key for its identity from the trusted entity [109]. A sender only needs to know a single master public key in order to send messages to anyone in the system. Receivers obtain private keys for their own identities. Figure 4.4 shows an example of this process.

The concept of ABE was first introduced by Sahai and Waters [92]. They describe the labelling of both ciphertext and keys with sets of attributes. Receivers can decrypt a message if only if the intersect of the attribute sets between their key and the ciphertext is of sufficient size. This construction was extended into Key-Policy ABE (KP-ABE) by Goyal et al. [41], where ciphertexts are labelled with sets of attributes and private keys are associated with access structures. Ostrovsky et al. [79] present a scheme that allows for negation in these access structures. This is in contrast to Ciphertext-Policy ABE (CP-ABE), introduced by Bethencourt et al. [12]; Waters [115], where the keys are labelled

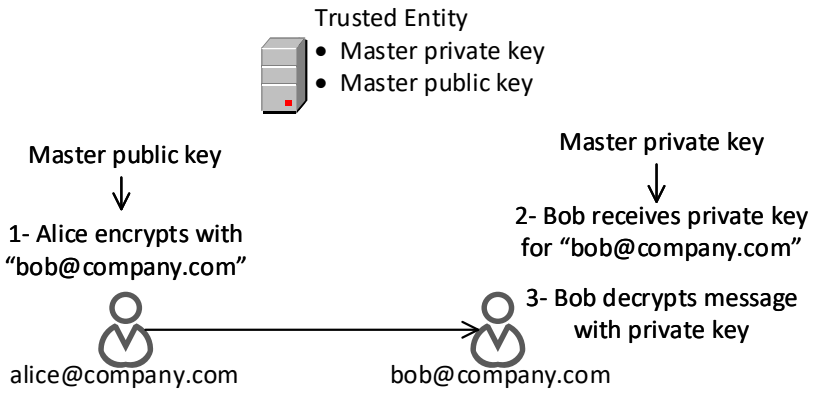


Figure 4.4: Identity-Based Encryption.

with sets of attributes and the ciphertexts are associated with access structures. The literature uses these two forms of ABE to provide confidentiality in pub/sub. The access structures are described as monotonic access trees, where inner nodes are composed of threshold gates which can be used to construct **AND** and **OR** logical gates and leaves describe attributes. More complex access controls like numeric ranges can be handled by converting them to access trees [12].

PBE is very similar to CP-ABE in that messages are associated with policies that must be met by the receiver in order to successfully decrypt a message. Both PBE and ABE are collusion-resistant since the combination of keys can decrypt a message if and only if at least one of those keys can decrypt the message on its own. Where PBE and ABE differ is that ABE does not hide the policies associated with a message [51]. Often, these policies are transmitted in plaintext which could reveal information to the routing brokers [81]. PBE only reveals the policy associated with a message if the receiver successfully decrypts the message.

Ion et al. [49] use the threshold gates of CP-ABE together with multi-user encrypted search [9, 29] to support confidentiality preserving content-based routing. Pal et al. [81] use both CP-ABE and PBE to ensure the confidentiality of events but perform no filter matching at the routers. Instead, all matching is done locally by the subscribers. This ensures the confidentiality of subscriptions since only a trusted third party entity that manages PBE and CP-ABE parameters and the subscriber itself ever know the unencrypted contents of a subscription. This comes at the cost of eliminating the

performance gains of filtering events as they stream through the broker network in a distributed manner that is typical of the pub/sub paradigm. Tariq et al. [109] use ABE and Decomposition Binary Trees to provide confidentiality in brokerless pub/sub. This technique supports subscription covering for more efficient routing of events at the cost of a weaker confidentiality model. Routing clients with access to multiple subscriptions can learn whether a subscription is coarser, finer or not in a containment relation relative to another subscription despite the encryption scheme used.

Yuen et al. [124] and Srivatsa et al. [107] divide event attributes into those that are sensitive and those that are not. Sensitive attributes are encrypted using ABE to preserve their confidentiality, but no routing decisions are made on these values. Instead, brokers route events based on the rest of the attributes which are left in their plaintext form. A trusted third party entity disseminates the security parameters to all clients, sensitive attributes are encrypted and filtering is done normally on the unencrypted attributes. Srivatsa et al. [107] further reduce the number of keys necessary in the system by mapping keys to a range of values which together cover the attribute space. Those keys can be used to decrypt any event with attributes that fall within the key's corresponding attribute ranges.

Filtering

Ion et al. [49] represent subscription filters as access trees. Any filter representing conjunctions and disjunctions of attributes can be represented as a tree in which leaf nodes are attributes and non-leaf nodes are CP-ABE threshold gates. A threshold gate is described by a threshold value and its children. Let x be a non-leaf node with threshold value k_x and having a number of children equal to num_x. The threshold value k_x represents the number of children of x that need to be satisfied in order for the node to be satisfied. When $k_x = 1$, it means that only one child needs to be satisfied, making the threshold gate an OR. When $k_x = num_x$, all children need to be satisfied, making the threshold gate an AND. Since $1 \leq k_x \leq num_x$, more general conditions such as 2 out of 3 attributes should be satisfied. Each leaf node x is described by an attribute and has a threshold value $k_x = 1$, meaning that the leaf node is satisfied when the attribute is

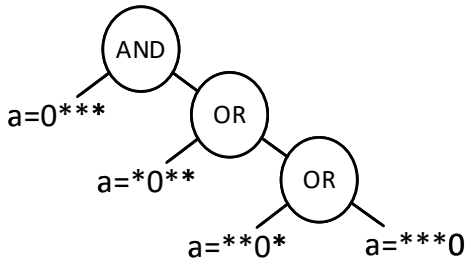


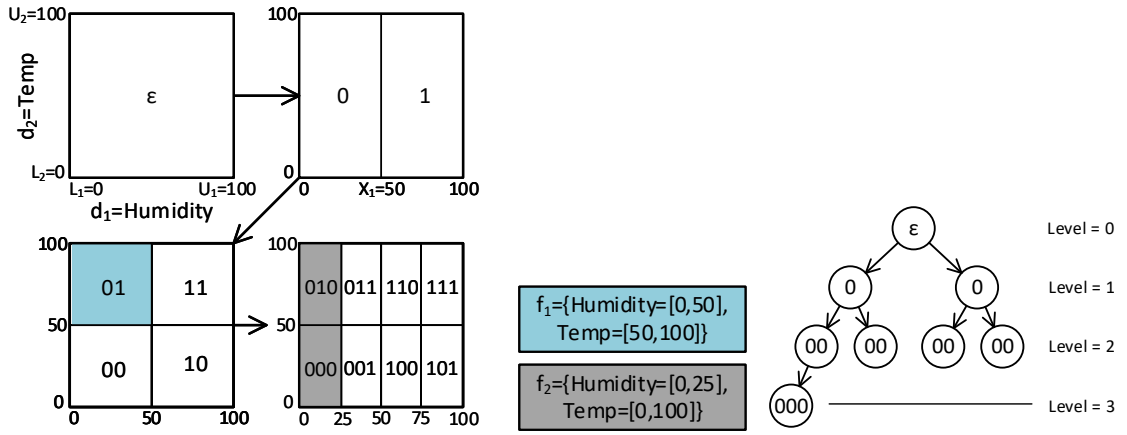
Figure 4.5: Access tree representation for $a < 5$ on 4 bits.

present, and not satisfied otherwise. Numeric ranges can be represented as well. Figure 4.5 shows an example for $a < 5$.

Once subscription filters are represented using these access trees, Ion et al. [49] leverage encrypted search techniques to provide content-based routing. In multi-user *searchable data encryption* (*SDE*), users are able to encrypt, decrypt and make encrypted queries on encrypted data stored on servers. Servers are able to perform computations on the encrypted data, without learning the content of the data or the queries. The paper cites Bao et al. [9] and Dong et al. [29] as example implementations of SDE that would work with their proposed scheme. Both implementations support keyword matching.

Access trees are formulated as encrypted queries by subscribers and are sent to brokers as subscriptions. Brokers filter events by executing these subscriptions on encrypted events. More specifically, when a new event reaches a broker, it runs a recursive algorithm on each subscription tree starting with the root node to check if it is satisfied by the attributes of the event. A non-leaf node is satisfied if the number of satisfied children is equal to or greater than its threshold value. If the tree is satisfied, then the filter is satisfied and the event can be forwarded.

Pal et al. [81] use both CP-ABE and PBE in order to ensure event confidentiality. Publishers encrypt an identifier for the event using PBE with the publisher's desired access policy. Event payloads are encrypted using CP-ABE with a similar access policy that excludes confidential attributes since the policy is not protected from the honest-but-curious brokers. Figure 4.7 shows a high level overview of information flow in the proposed system. Publishers send the CP-ABE encrypted event payload and PBE encrypted event identifier to the dissemination server (or broker network) (1). Event payloads are routed



(a) Hierarchical decomposition of Event Space (b) Subscriptions (c) Decomposition Binary Tree

Figure 4.6: Access control enforcement models at a given broker.

to a repository server for storage (2). Event attributes are routed to subscribers (3), which then attempt to decrypt the locally match based on their PBE attributes assigned by a trusted third party entity. If unsuccessful, PBE reveals neither the event identifier nor the access policy to the subscriber. If successful, the decrypted event identifier is used by the subscriber to request the event payload from the repository server (4 and 5). The subscriber must then use CP-ABE to decrypt the payload in order to gain access to the payload.

Tariq et al. [109] achieve weak confidentiality through a combination of decomposing events into Decomposition Binary Trees and ABE to ensure only authorized subscribers can decrypt events. The implementation described by the authors uses ABE, but it could use IBE and possibly other authorization techniques. ABE is very convenient in that it reduces the number of keys that need to be managed by the system relative to traditional public-key encryption techniques.

An event space composed of d distinct numeric attributes can be geometrically modeled as a d-dimensional space such that each attribute represents a dimension in the space. The event space is hierarchically decomposed into regular subspaces, which serve as enclosing approximation for the subscriptions, advertisements, and events. The decomposition procedure divides the domain of one dimension after the other and recursively starts over in the created subspaces. An example of this is shown in Figure 4.6(a) with

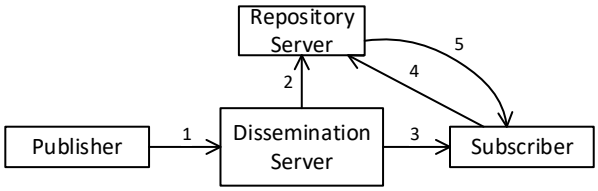


Figure 4.7: High level information flow diagram [81].

$d = 2$.

Subspaces are identified by a bit string of "0" and "1"s. A subspace represented by dz_1 is covered by the subspace represented by dz_2, if dz_2 is a prefix of dz_1. Subscription or advertisement of a client can be composed of several subspaces. A credential is assigned for each of the mapped subspace. For instance, in Figure 4.6(a), f_2 is mapped to two subspaces and therefore possesses two credentials $\{000, 010\}$. An event can be approximated by the smallest (finest granularity) subspace that encloses the point represented by it. To deliver the encrypted event, a ciphertext must be generated for each subspace that encloses the event so that the client whose subscription mapped to any of these subspaces should be able to successfully decrypt the event. For example, an event 010 is enclosed by five subspaces: $0010, 001, 00, 0$, and ϵ. In the case where an event space has a large set of numeric attributes, a decomposition tree can be formed for each attribute to improve scalability. For strings, the event space is decomposed into a trie rather than a binary tree to support more expressive string operations in subscriptions.

This decomposition method creates a trade-off between the coarseness of the decomposition and the number of credentials that must be maintained by the system. While a finer granularity is desirable in order to better serve arbitrary subscriptions, this means that a greater amount of credentials will be required. Conversely, coarser granularity reduces the amount of credentials but will also increase the number of false positive events being delivered to subsribers that can decrypt the events.

When publishing or subscribing, the client must first authenticate with the trusted third party entity by sending all relevant credentials. The trusted third party entity will then authenticate the client and generate the private keys needed for publishing and reply to both kind of requests with all the private keys associated with the credentials of the requested attributes. When publishing, for each attribute of the event, a cipher-

text should be created for every credential that matches the value associated with that attribute, so that a subscriber with any of these credentials can decrypt the event. For example, in case of a numeric attribute with value mapped to 0000, a ciphertext should be disseminated for the credentials $0000, 000, 00$, and 0. These ciphertexts are ordered so that receivers know which ciphertext corresponds to which credential. Since subscribers can't tell whether they have the authorization to decrypt a message or not, a known string is inserted into the plaintext so that a successful decryption can be differentiated from an unsuccessful one.

Subscribers know the position of the ciphertext that corresponds to their credential and attempt decrypting it. They can verify that the decryption was successful through the known plaintext. An unsuccessful decryption denotes that the subscriber is not authorized to decrypt the event since only authorized subscribers can generate the decryption key by the properties of ABE. Since the cost of asymmetric decryption generally increases with the size of the plaintext, the authors suggest that only a symmetric key be encrypted in this way, with the symmetric key decrypting the rest of the event.

Under this scheme, the pub/sub overlay is a virtual forest of logical trees, where each tree is associated with an attribute. A subscriber joins the trees corresponding to the attributes in its subscription. Similarly, a publisher sends an event on all the trees associated with the attributes in the event. In the tree, subscribers are always connected according to the containment relationship between their credentials. To join a tree, a new subscriber must contact a random node within the tree to learn its position within the tree. It does so through a connection request, which the receiving client forwards to its parent or children as needed to find the correct spot for the new node.

There are two event dissemination strategies presented by Tariq et al. [110]. In one-hop flooding, a parent assumes that the children have the same credentials as its own and forwards each successfully decrypted event to all of them. In turn, the children do the same and so on. In this strategy, each subscriber maintains an overlay connection for each credential in the worst case. Moreover, this form of routing leads to false positive events being forwarded since children may have finer subscriptions. Multicredential routing reduces the number of false positives by enabling parents to forward only those events

on each attribute tree that match the credentials of their children. Although the parent does not know the subscription of its children, it can decrypt all messages it forwards and over time could deduce it. This violates weak confidentiality. To preserve weak subscription confidentiality, subscribers divide their original credential(s) for each attribute of their subscription into a number of fine granular credentials. For example, credential 1 for a numeric attribute can be divided into three credentials $10, 110$, and 111. A separate parent connection is maintained for each credential, ensuring that a subscription is not compromised unless multiple parents collude with each other. Subscribers must ensure they are connected to different parents. This can be done through techniques like broadcast revocation [57].

4.5 Multiple Layer Commutative Encryption

Overview

Multiple Layer Commutative Encryption (MLCE) allows intermediate nodes in charge of routing secure traffic to perform transformations on the encrypted data without having access to the plaintext data that is being transferred. Shikfa et al. [98] ensure the confidentiality of events and subscriptions in a content-based pub/sub by leveraging the properties of MLCE to perform keyword matching operations on encrypted data.

In multiple layer encryption, data is encrypted several times with different keys. Multiple layer encryption has been used in the context of multicast security and data aggregation [37, 74, 83]. In Chapter 6.1.1, we present onion routing, a multiple layer encryption scheme used to achieve user anonymity. An encryption mechanism E is commutative if and only if for any data d and any keys k_1 and k_2, $E_{k_1}(E_{k_2}(d)) = E_{k_2}(E_{k_1}(d))$. In the case where the multiple encryption layers all use the same cryptosystem and the cryptosystem is commutative, then the layers can be added and removed in any order. This gives the basic properties of MLCE.

The solution presented by Shikfa et al. [98] is based on the Pohlig-Hellman cryptosystem [86], where subscribers do not need to share a unique and common key with the

publisher (although under some conditions discussed later, a publisher and a subscriber might share keys). Although under this system encryption and decryption keys differ, they are both secret as one can be used to deduce the other. Since this cryptosystem does not differentiate brokers from subscribers, it can be adapted for use with brokerless pub/sub by having brokers subscribe to events and sending their own subscriptions while performing routing operations.

As mentioned previously, the scheme presented by Shikfa et al. [98] only supports keyword matching. This severely limits the ability of the system to do content-based routing. The benefit of using this system is that it allows for the system to perform routing optimizations through subscription covering by using the $SECURE\_TABLE\_BUILDING$ primitive which we present in the next section. Furthermore, multiple encryption layers offer some protection against colluding entities. Messages are compromised if and only if these entities have access to the decryption key of every layer in the message. The number of encryption layers used by the scheme is given by parameter r. This parameter can be increased to secure against more colluding entities at the cost of the required amount of key storage per node and increasing the complexity of the key dissemination required. Furthermore, subscription covering becomes less effective as r increases.

Shikfa et al. [98] consider the problem of key dissemination to be necessary but orthogonal to their proposed confidentiality scheme. Existing solutions tend to use a trusted third party to do this. We discuss key management in more detail in Chapter 5.

Filtering

Shikfa et al. [98] model the pub/sub system as a tree with a single publisher as the root node, subscribers as the leaf nodes and brokers as inner nodes. This model is not meant to represent the entirety of the pub/sub network but the subset of nodes that are active when a single event is published. We refer to this tree as the *dissemination tree*. As part of the proposed solution, four security primitives are implemented:

1. *ENCRYPT_SUBSCRIPTION:* used by subscribers to generate encrypted subscription filters. As input, it takes a subscription filter and some keying material and it

outputs an encrypted version of the subscription filter.

2. *ENCRYPT_EVENT:* used by the publisher to encrypt its notifications. As input, it takes an event's attributes and some keying material and outputs an encrypted version of the event's attributes.

3. *SECURE_LOOK_UP:* allows a broker to decide whether the encrypted event attributes match one of the encrypted subscriptions of its routing table. This primitive should only return the boolean result of the matching operation.

4. *SECURE_TABLE_BUILDING:* allows the broker to build a routing table and to compare two encrypted subscriptions. If two subscriptions match the same content there is no need to forward both of them to the broker's parent. The broker needs to store both of them with their corresponding child in its routing table and only forward one to its parent. As with $SECURE\_LOOK\_UP$, this primitive should only return the boolean result of the matching operation, but it should not leak any additional information about the subscriptions. The aggregation is not necessary for privacy, but vital in order to implement common content-based routing optimizations.

The number of encryption layers in the system proposed by Shikfa et al. [98] is given by the parameter $r \geq 2$. Since key dissemination is considered outside the scope of the paper, the system is assumed to have an initial state consisting of each node sharing a Pohlig-Hellman key pair with each of its one and r-hop neighbours in the dissemination tree. If there are less than r hops between publisher and subscriber, then under this scheme they must share a key pair weakening space decoupling. For simplicity, we set $r = 2$ for the remainder of the section.

In the dissemination tree, published events disseminate downwards from the publisher to the subscriber(s). Likewise, subscriptions disseminate upwards in the dissemination tree through the brokers involved in routing events. We summarize the broker operations in Table 4.2. The broker is denoted by B, its grandparent by G, its grandchild by C, the encryption algorithm is E and the decryption one is D. In the left column, B receives an

Table 4.2: Message processing at a broker

Upwards: Filter Propagation	**Downwards: Event Dissemination**
Remove an encryption layer: $D_{k_{BC}}(SF)$	Remove an encryption layer: $D_{k_{BG}}(EN)$
Update the routing table RT_B: $SECURE\_TABLE\_BUILDING(RT_B, D_{k_{BC}}(SF))$	Secure look-up: $SECURE\_LOOK\_UP(RT_B, D_{k_{BG}}(EN))$
Forward the message upwards	Forward the message downwards

encrypted subscription filter SF and in the right column B receives an encrypted event EN.

$ENCRYPT\_SUBSCRIPTION$ is used by subscriber S_i and requires the subscription filter and two encryption keys $k_{S_iB_j}$ and $k_{S_iB_l}$ where B_j and B_l are S_i's parent node and grandparent node, respectively. It outputs an encrypted filter SF_{S_i} computed as $ENCRYPT\_SUBSCRIPTION(f, k_{S_iB_j}, k_{S_iB_l}) = E_{k_{S_iB_j}}(E_{k_{S_iB_l}}(f))$. S_i then sends the message $[SF_{S_i}, S_i]$ to its parent node B_j. Symmetrically, the publisher P first uses the $ENCRYPT\_EVENT$ to encrypt the event's attributes with the corresponding keys and forwards the packet to the next broker.

Since only keyword matching is supported by this scheme, $SECURE\_LOOK\_UP$ consists of comparing two encrypted strings and checking if they are equal. This is possible since the following property holds: $E(a) = E(b) \iff a = b$. The broker first removes an encryption layer before doing the comparison.

$SECURE\_TABLE\_BUILDING$ is used by brokers to construct the routing tables. Whenever a broker receives a subscription, it first removes an encryption layer and compares it to the entries in its routing table. If there is a row that matches the recently received subscription then there is no need to forward that subscription as it will already receive events matching the same attributes. Otherwise, a new entry in the table is created and the subscription is re-encrypted and forwarded as normal.

4.6 Oblivious Transfer

Overview

Oblivious Transfer was originally introduced by Rabin [87] who informally described it as a game between Alice and Bob. Alice wants to send a message to Bob in such a way that with probability 1/2 Bob will receive the same message Alice wanted to send, and with probability 1/2 Bob will receive nothing. Moreover, Alice does not know which of the two events really happened [25].

Conditional Oblivious Transfer is a variant introduced by Crescenzo et al. [25] in which Alice and Bob have private inputs and share a public predicate that is evaluated over the private inputs. If the predicate holds, then Bob successfully receives the message Alice wanted to send him and if the predicate does not hold, then no matter how Bob plays, he will have no information about the message Alice wanted to send him. Furthermore, no efficient strategy can help Alice during the protocol in computing the actual value of the predicate.

Equality Conditional Oblivious Transfer (eq-COT) is a variation of Oblivious Transfer where Alice wants to privately transfer a message to Bob in a way that the only leaked information is Alice's message when the equality predicate evaluates to 1 with private inputs from Alice and Bob. Alice's (respectively, Bob's) private input is a string x_a (resp., x_b). Alice's message is denoted by m_a, while m_b denotes Bob's output at the end of the protocol. The following requirements hold [21]:

- If $x_a = x_b$ then the probability that $m_a \neq m_b$ is negligible.
- If $x_a \neq x_b$ and m_a is uniformly distributed, then the distribution of m_b is uniform and independent from m_a.
- If $x_a \neq x_b$ then for any adversary corrupting Bob, the protocol's communication transcript reveals no information to the adversary about m_a.
- The protocol's communication transcript reveals no information to an adversary corrupting Alice about whether $x_a = x_b$ or not.

Hybrid Equality Conditional Oblivious Transfer (h-eq-COT) was first introduced by Crescenzo et al. [21] and is a 2-phase protocol that allows Alice to perform an eq-COT of an arbitrary number of messages to Bob. In the first phase, called *asymmetric phase*, Alice and Bob execute a single preliminary eq-COT of a random symmetric key k_a of size k, based on asymmetric cryptography techniques, where k_a denotes Alice's input and k_b denotes the received key by Bob at the end of this phase. In the second phase, called *symmetric phase*, Alice and Bob execute an eq-COT of a message m_a, based on symmetric cryptography techniques, where Alice takes as input k_a, m_a and Bob takes as input k_b and receives m_b at the end of this phase. That is, in all symmetric phase executions of the eq-COT protocol, Alice and Bob take as input the symmetric key returned at the end of the asymmetric phase of the protocol (i.e., the same key if $x_a = x_b$ or random and independent keys otherwise). The advantage of h-eq-COT lies in that once the first phase is done, it can use the faster operations from symmetric key encryption techniques rather than the relatively slow operations from asymmetric key encryption techniques.

During the asymmetric phase, Bob posts an asymmetric encryption of string x_b, where the encryption scheme used allows Alice to later manipulate this encryption and transform it, without knowing x_b, into an encryption of $k_b = k_a(x_b/x_a)^r mod\ p$, for some random value r, where k_a is Alice's input secret key. In this way, if $x_b = x_a$, Bob receives an encryption of $k_b = k_a$, which he can decrypt, while if $x_b \neq x_a$, Bob receives an encryption of a random key k_b independently distributed from k_a.

At any later time during the symmetric phase, to perform an eq-COT transfer of any message m, Alice uses key k_a and Bob uses key k_b, and both use an arbitrary symmetric encryption scheme. Alice can just perform a symmetric encryption of data item m based on k_a and Bob would be able to decrypt the right item whenever $k_b = k_a$, which holds whenever Alice's private input x_a is equal to Bob's private input x_b.

In the pub/sub system described by Crescenzo et al. [21], the asymmetric phase of h-eq-COT is used in order to establish a session key for use with the symmetric phase. The session key then encrypts all published events that are sent to that particular subscriber. A publisher must establish a session key with all subscribers in the system. There is no further event filtering in this system beyond subscribers expressing their interests of

topics to publishers. Events are encrypted with each subscriber's session key and sent to all subscribers. The subscribers are able to decrypt the message if and only if their subscription filters match the attributes of the event. To subscribe, subscribers only have to post their subscription on its public file, without interacting with any other party. This public file is used by the publisher to perform the asymmetric phase of h-eq-COT.

Under the presented schemes, there is a very strong coupling of publishers and subscribers. Crescenzo et al. [21] propose a modification to their schemes that use a repository server that is reminiscent of a broker. It allows publishers and subscribers to share information using a publicly accessible file. Subscribers are responsible for pulling events from this file which they are interested in. This alternative does not solve the space decoupling problem.

4.7 Secret Sharing

Yoon and Kim [123] present the only truly distributed solution to the problem of disseminating secrets in a pub/sub network in the literature. Whereas previous solutions require the use of an out-of-band trusted third party to disseminate information in order to use the various encryption techniques, this solution uses the existing broker network to transmit secrets between clients. This solution has a stronger threat model, with curious brokers that may arbitrarily delay or drop messages in their entirety. It also tolerates a number of Byzantine brokers. This functionality is provided at the cost of increased processing time and strict topology requirements. Yoon and Kim [123] make use of Shamir's secret sharing scheme [96] to split the secret value into multiple pieces before sending them to the appropriate subscribers through the broker network. Only entities with a sufficient number of pieces can reconstruct the original secret value. As long as the colluding brokers do not collectively have a sufficient number of pieces, the original secret value remains secure. We will omit reintroducing the details of this solution as we have already presented this work in Section 2.3.2 and Section 2.3.3.

On its own, this solution does not allow for content-based routing on sensitive values by the brokers. Furthermore, it requires the transmission of a large number of messages

in order to transmit an event end-to-end. The severity of this latter problem can be mitigated by limiting the use of this solution only to establish the security parameters and/or keys required to implement any of the other solutions presented in the previous sections. After this initial step, the other encryption-based solutions could be used to avoid these drawbacks. This provides an alternative to the more common method of relying on a trusted third party entity. Although this would mitigate the problem, it does not solve it. Figure 2.5 shows the minimum number of fragments needed to share a single secret. This number grows exponentially as the number of brokers between clients increases.

4.8 Key Management

Key management within pub/sub systems has more depth than those used in standard public-key communication. Scalable key management techniques are required in order to ensure the use of the encryption techniques can be effectively used in pub/sub. Within the context of key management, there are two problems that must be addressed. First, keys must be disseminated to clients while maintaining the decoupling properties of pub/sub. Second, key usage must scale in a manageable way as the number of clients and events grows.

In content-based systems, every event can potentially have a different set of interested subscribers. It is desirable to encrypt messages so that only interested subscribers can read the message. A naïve solution to this problem is to split subscribers into groups, and assign a key to each group. Events intended for that subgroup of subscribers can then be encrypted using this key. The problem with the naïve implementation is that if the system has n clients, then there are 2^n possible subgroups. Since each event can go to a different subgroup in the worst case scenario, 2^n keys would need to be generated, disseminated and managed by either the publisher and/or some trusted third party [77]. With thousands of subscribers, it is infeasible to setup static security groups for every possible subgroup [106]. Another solution might be for publishers to encrypt each event using individual subscriber keys. This is difficult to implement without simultaneously

weakening the decoupling properties of pub/sub. If the publisher has to replicate, encrypt and transmit an event for each subscriber, then it would not be leveraging the efficient multicasting usually performed by brokers. Additional group key management protocols like broadcast encryption [33], secure multicast [43, 44] and logical key hierarchies [88] have the same weaknesses [106].

Several of the techniques presented in this section address the problem of key management by using keys generated based on the attribute space rather than the individual subscribers. This makes the number of keys that must be managed linear in the number of subscriptions and independent from the number of subscribers [106]. In the case of topic-based pub/sub, this approach is trivial to implement since a key can be generated and disseminated for each possible topic. Implementation in content-based pub/sub is not trivial, as subscribers might only be authorized to access a subset of events. Chapter 4.4 presents multiple approaches to this problem. The event space decomposition of Tariq et al. [109] shown in Figure 4.6 generates buckets from the event space. An event's attributes all fall within one or more of these buckets, each of which is assigned a corresponding key. A similar approach is presented by Srivatsa and Liu [105] using numeric attribute key trees. This approach only decomposes a single attribute by having the full range represented by sub-ranges at the leaf nodes. Parent nodes represent the union of its childrens' ranges. Each node is assigned a corresponding key that allows for the decryption of events with an attribute encompassed by the node. The key of a node can be used to derive keys assigned to child nodes, but not vice versa.

The literature has made use of epoch-based keys, where the encryption key changes after some time period defined by the system. In order to continue publishing events and receiving events, clients must generate and/or be issued the new keys. This allows for the system to eliminate unadvertisement and unsubscription messages as advertisements and subscriptions are only valid for during that epoch [106].

Various key caching techniques are presented by Opyrchal and Prakash [77] and Srivatsa and Liu [106]. These allow for some performance enhancements as keys can be remembered rather than having to regenerate them. We do not go into further detail on these techniques as they are fairly typical of non-pub/sub systems.

Chapter 5

Authorization

For the purposes of this thesis, authorization is the property of being authorized to perform various actions within the pub/sub system. This chapter presents the techniques used for arbitrarily controlling the flow of messages through pub/sub beyond the regular routing operations performed by brokers. These techniques are used to control which entities within the system can send, receive and decrypt specific messages. By necessity, the latter requires some form of encryption using the techniques discussed in Chapter 4 and some form of access control (AC) achieved by managing which entities have access to the decryption keys. Although there is a large amount of work in the literature dedicated to this subject, we present only those techniques that are specifically tailored for use with pub/sub.

Whereas Chapter 4 focuses on encryption in pub/sub, this chapter emphasizes techniques for exerting control over which entities in the system have access to plaintext messages. In general, encryption requires some form of control over which entities in the system have access to plaintext messages, otherwise it is trivially defeated. The same is not true in the reverse. In other words, while confidentiality requires authorization, authorization does not necessarily require confidentiality.

In some applications, authorization is critical to the adoption and use of pub/sub systems. Authorization concerns should be considered wherever the information being disseminated is sensitive or private. Healthcare is a common example of this sort of application [50, 49, 103, 102]. In this application, the events are generated containing

private patient information. Control over which entities have access to the events is not only desirable, but often is a requirement dictated by government regulations of how such data is to be handled.

Other applications considered within the context of authorization in the literature include payment systems [114], geolocation-based services [78] and financial markets [114, 125]. Payment systems are those where subscribers pay for events generated by publishers. Subscribers receive and pay for those events that match their subscription. Some form of authorization is required in order to only deliver events to paying subscribers. Geolocation services advertise goods and services to users that are within specific geographic locations. In these systems, users generate events containing their current location, which are distributed to the subscribers that are within some range of the subscriber. Users tend to be reticent in broadcasting their location indiscriminately to the public, and so they have an interest in which entities have access to exactly what information. Financial systems involve stock quote dissemination systems where improper or insufficient AC might leak a potential customer's investment strategy which could result in a loss of return in investments.

The control of which entities can send data mitigates some forms of denial of service attacks that flood the system with messages, as presented in Chapter 7. By blacklisting these entities and blocking their messages from being forwarded through the broker network, the effects of such an attack are localized to edge brokers. If publishers charge for their events, AC is used to ensure only the paying subscribers can receive and/or view only the messages they are entitled to.

We categorize the pub/sub authorization literature into two categories: client and broker AC. The two forms are largely independent of one another and not mutually exclusive. Client AC details the techniques used for dictating what actions clients may take within the system. Similarly, broker AC details the rules governing broker behaviour within the system.

Table 5.1: Message types controlled by the various solutions presented.

Paper	Routing	Advertise	Publish	Subscribe	Receive
Fongen and Mancini [34]	Topic	✓	✓	✓	✓
Pallickara et al. [82]	Topic	✗	✓	✓	✗
Ion et al. [49]	Content	✗	✗	✗	✓
Srivatsa et al. [107]	Content	✓	✓	✓	✗
Bacon et al. [7]	Content	✓	✓	✓	✗
Tariq et al. [110]	Content	✓	✓	✗	✗
Opyrchal et al. [78]	Content	✗	✗	✓	✓
Tarkoma [111]	Content	✓	✗	✓	✗
Zhao and Sturman [125]	Content	✗	✓	✓	✓
Miklos [66]	Content	✓	✓	✓	✗

5.1 Client Access Control

Client AC is the enforcement of arbitrary AC rules that control the messages that a client may transmit through a pub/sub network. All of the actions that a client might take are subject to control, including (un)advertising, publishing, (un)subscribing and the receiving of events. For example, a given client may be authorized to publish events belonging to specific topics but not others. Alternatively, the client may not be allowed to subscribe to particular events or is to receive events with attributes within certain attribute ranges. Clients may also be limited to a maximum event publishing/receiving rate. Table 5.1 summarizes the kinds of AC rules supported by the academic literature.

In the literature, the AC rules are centrally administrated by an AC Manager which disseminates the rules to brokers within the system. Changes to the AC rules must be disseminated again to the appropriate brokers within the system [66]. All messages include a token generated when clients authenticate with the AC Manager which have privileges associated with them and are valid for a limited amount of time [34]. These tokens are checked by brokers for their validity before the event is forwarded. Brokers are trusted to enforce these rules and discard messages which do not conform to them. Clients may renew expired tokens by re-authenticating with the AC Manager.

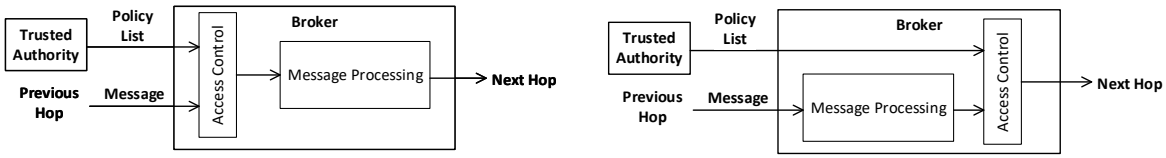
Ideally, AC rules are verified as early as possible in the transmission of a message so that resources are not wasted in the transmission of unauthorized messages. In the case of AC rules regarding publishing, this may be as simple as having the edge broker

enforce the rules. In the case of subscriptions and receiving AC rules, these need to be forwarded to other brokers in order to minimize the amount of unnecessary work that is done in routing. If the routing algorithms used could route events through multiple paths, this becomes more difficult. It is often simpler to have edge brokers perform all AC checks [66, 125, 78]. In the presence of malicious brokers which may inject their own messages into the system (or may not properly enforce the AC rules) then every broker should verify the AC rules to discard unauthorized messages [111].

This form of AC is essential for the mitigation of denial of service attacks by clients [82, 117]. This is where a malicious entity will flood the system with messages. With sufficient messages, the effective routing of messages can grind to a halt. Through client AC, a maximum message rate rule would prevent a flood of messages from being forwarded into the system. Additionally, an AC Manager may prevent such clients from receiving or renewing their tokens when the client authenticates, effectively blacklisting the entity from using the pub/sub resources.

The generation of AC rules is often application-specific. One option is to have an *event owner* which can define the attributes that any given subscriber must have in order to subscribe [34, 78]. Although publishers are usually the event owners for the events that they publish, decoupling the two allows for more granular AC rules. For example, a publisher event owner cannot directly specify the identities of subscribers with the authority to subscribe to events without simultaneously violating the space decoupling property of pub/sub. Publisher event owners are strictly limited to defining the general attributes that potential subscribers may have. Another option is to have the AC Manager dictate all the AC rules. Since it is a trusted third-party, there is no risk of violating space decoupling. Finally, it is also possible to have a hybrid system where event owners may negotiate with the AC Manager regarding the AC rules [7, 82]. Event owners may delegate their power to influence the AC rules to other entities [78].

Bacon et al. [7] use Role-Based Access Control (RBAC) in order to get around the space decoupling issue. In general, RBAC simplifies the administration of access control for large systems [93]. Roles provide an indirection layer between clients and access control privileges. Privileges are assigned to roles which are separately associated with



(a) Pre-matching (b) Post-matching

Figure 5.1: Access control enforcement models at a given broker.

clients. For example, an auditor role might have permission to subscribe to events but not to publish. A client associated with the auditor role would not be able to publish events since their role does not grant them the authority to do so. This effectively decouples access control management from the administration of clients [7]. Even if an authorized publisher were to directly modify the access control privileges of a role, they would not be violating the space decoupling property of pub/sub as the policies are independent of the identity of the members within that role.

The AC rules supported by topic-based pub/sub systems shown in Table 5.1 are coarse-grained, generally allowing supporting the whitelisting and blacklisting of various messages. The content-based pub/sub systems can define granular rules based on the event's attributes by reusing the same mechanisms that perform the routing of events. Some systems check the AC rules before doing any routing (pre-matching) and some do so in reverse (post-matching)[1] [119]. Both design models are illustrated in Figure 5.1.

Zhao and Sturman [125] specifically deal with the issue of dynamically changing AC rules during the regular operation of the pub/sub system. By including a version number in the policies, it is possible to account for changing privileges between clients. All access control policy versions are persisted by a trusted entity, and brokers maintain a cache of the latest rules. This trusted entity disseminates new versions of the access control list to the pub/sub network and responds to queries from brokers for specific policy versions. When a new access control list is received by a broker, edge brokers choose an arbitrary point after which they start applying the new policies to incoming messages. The version number used is added to the message by the edge brokers so that the other brokers know

[1]Operations performed at the pre-matching and/or post-matching step do not have to involve AC rules. Mercier et al. [62] and Barazzutti et al. [10] have used Bloom filters during pre-matching to quickly eliminate non-matching events before doing more expensive matching operations

which set of rules to apply to it.

5.2 Broker Access Control

Broker AC refers to the enforcement of arbitrary AC rules that control the messages that a broker may transmit through a pub/sub network. Any message that is to be sent or received by a broker may be subject to these rules. The motivation behind broker AC is similar to client AC. That is, brokers may inject messages that may disrupt the pub/sub system. These injected messages could be used as part of a denial of service attack or as part of a more sophisticated attack to the pub/sub system.

In general, broker AC takes two forms: hop-level and domain-based AC. Hop-level controls are used to control messages at a hop-by-hop level. Domain-based AC controls the flow of messages between clusters of brokers referred to as domains.

5.2.1 Hop-Level Access Control

Hop-level AC refers to the hop-by-hop interaction between brokers. Just as clients may send unauthorized messages, so too may brokers. A broker may erroneously forward an unauthorized message, or they may maliciously inject their own. This attack vector can be used as part of a denial of service attack or as part of a more sophisticated attack.

Tian et al. [112] add a privacy and priority attribute to subscriptions, which are checked and enforced by brokers. The privacy attribute is used for AC purposes. The privacy specified by the event is matched to the known privacy level that the subscriber has access to. This privacy level is managed by a trusted service. The priority attribute determines the order in which subscriptions are processed by the brokers.

Interaction Control is a policy model that adds granular hop level controls to pub/sub. These policies are maintained and enforced by each broker within the pub/sub network [104]. These policies can affect both the connections formed by brokers (and by extension the construction of the broker network) as well as the flow of information within them. The policies take the form of point-to-point (hop-level) controls that directly specify what information is allowed to be sent to each entity directly connected to the broker. This

effectively bounds the flow of information within the network. This is motivated by large pub/sub systems spanning multiple administrative domains, each with their own access control policies [104, 99, 100] as discussed in Chapter 5.2.

Information Flow Control shares similarities with Interaction Control. Information Flow Control uses data labelling within messages to enforce where data may go within the system. Data is tagged with security labels akin to what is already done with data types at the programming language level. These tags can be checked by processes to ensure that data can only flow to processes with compatible labels [6]. DEFCon (Decentralized Event Flow Control) is a framework that implements Information Flow Control that is used in the broader context of middleware [65, 64]. PrivateFlow is a prototype implementation of pub/sub that makes use of the DEFCon framework [84].

5.2.2 Domain-Based Access Control

Domains refer to a collection of brokers which all trust one another but may not necessarily trust brokers belonging to other domains [7]. Large pub/sub systems may span several domains. In such systems, a Domain Manager is responsible for administrating AC policies. Intra-domain AC mechanisms are akin to those mentioned in the previous section, specifying which clients are allowed to send what messages. Inter-domain AC (referred to simply as Domain AC for the rest of this thesis) is similar, specifying the AC rules to be followed for messages between events.

Bacon et al. [7] describe a large pub/sub system spanning multiple administrative domains. Domain AC is negotiated between the respective Domain Managers (or their authorized delegates). Edge brokers encrypt events from their clients. The only exception are the topic identifiers which are left in plaintext. If a broker is in a trusted domain to that topic, then it can retrieve the decryption key, decrypt the event and perform content-based routing normally. Otherwise, the broker cannot decrypt the event's attributes and can only do limited routing based on the topic identifier (akin to topic-based routing). It is also possible for the domain manager to specify that an event cannot be routed through specific domains. Finally, as events reach their destination, they are decrypted at the subscriber's edge brokers. Public-key encryption is used in this work, administrated by

the domain managers. Edge brokers are assumed to be within the same domain as their clients and thus trusted.

Chapter 6

Anonymization

At its core, anonymity seeks to obfuscate the identities of actors within the pub/sub system. Anonymity is similar to confidentiality in that it seeks to keep sensitive information from an adversary. Whereas confidentiality deals with the sensitive data being transmitted, anonymity is concerned with the sensitive identities of the various entities within the system. Each message sent from one entity to another could compromise the identity of the sender and/or receiver to an adversary within the context of both hop-by-hop and end-to-end communications. In this chapter, we present the academic literature on the subject of pub/sub anonymity.

The space decoupling is already a property of pub/sub systems. This means that anonymity can be achieved without significant alterations to the basic pub/sub paradigm. This means that anonymity can be guaranteed through the use of various enforcement mechanisms. We organize the approaches in the literature into two broad categories: communication and data anonymity. The former addresses the anonymous transmission of messages while the latter addresses the problem of the data itself identifying entities within the system.

6.1 Communication Anonymity

This dimension of anonymity strictly concerns itself with the sending of a message without revealing the sender. It is independent of the message content and payload. Along this

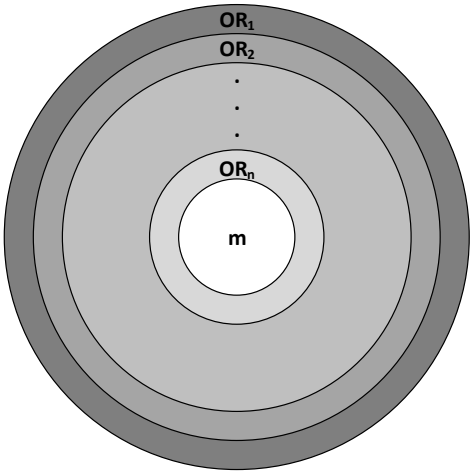


Figure 6.1: The anatomy of a packet with message m to be sent through the specified Onion Routers (OR) in the network.

dimension, the adversary is considered to be any entity (or collection of entities) other than the sender of the message. This model holds for all forms of messages transmitted using the pub/sub system, including events, advertisements and subscriptions.

We present two mechanisms that have been used in the pub/sub literature to ensure communication anonymity: Onion Routing and the Logical Link Orientation Scheme. The key idea behind both is that of relaying a message several times on to its destination. While other forms of communication anonymity (i.e., garlic routing [26], broadcast anonymous routing [56], etc.) could probably be adapted for use with pub/sub with relatively low effort, these techniques have not been applied to pub/sub in the literature.

6.1.1 Onion Routing

Onion Routing [27] is a mechanism for anonymous communication. Using a distributed overlay network, messages are routed through a series of relay nodes. These relays obfuscate the identity of the message source. Onion routing offers a simple and flexible design that is easy to use and deploy, running as a user-level process without the need of any special privileges from each relaying node.

Users send a specially constructed message to a relay node (also known as an onion router). Upon receipt of a message, an onion router peels away a layer of encryption by decrypting the message with its session key. This reveals the identity of the next

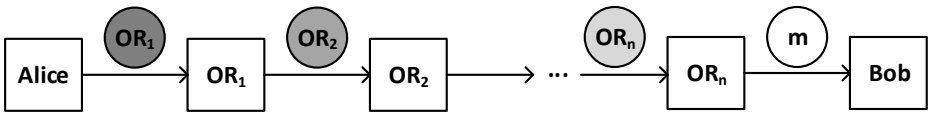


Figure 6.2: A message m as it goes through a network of Onion Routers (OR).

hop (which itself is either another relay or the message destination) in the routing path. Figure 6.1 shows these layers (denoted by concentric circles) in a high level view of the anatomy of a packet. This process continues until the last hop in the onion network decrypts the message and transmits it to its intended destination. This entire process is shown in Figure 6.2. Another feature of Onion Routing is that of proxy addresses that can be announced to others. Messages can be sent to these addresses without revealing their true identities and sacrificing their anonymity.

Onion Routing has been used for both regular [113] and brokerless [55] pub/sub. Vo and Bellovin [113] apply Onion Routing without modification acting as an anonymous proxy between clients and the broker federation. The strength of this approach is that the anonymous communication mechanism could be easily replaced by another one. The problem is that it requires the additional resources of an additional dedicated Onion Routing network.

Klonowski and Rózanski [55] use a modified version of Onion Routing that incorporates ElGamal-based Universal Re-encryption [39] rather than the traditional AES encryption. Their solution provides communication anonymity to subscribers in a brokerless pub/sub network. This allows for the Onion Routing operations to be done by the pub/sub network itself rather than by a distinct proxy network. The authors do not present or evaluate an implementation of their proposed solution.

There are several weaknesses to using Onion Routing as an anonymous communication network [27]. Onion Routing is not steganographic, meaning that it does not conceal the fact that a particular user is connected to the network. Rather, it hides the identity of message sources. Furthermore, Onion Routing is vulnerable to colluding nodes at the end points of a routing path. In general, if an adversary controls $m > 1$ of N nodes, they can correlate at most $(m/N)^2$ of the traffic.

6.1.2 Logical Layer Scheme

The scheme presented by Anceaume et al. [3] provides a mechanism for ensuring communication anonymity for publishers in brokerless pub/sub systems. This work is further refined by Datta et al. [23], who generalizes the scheme into regular pub/sub with brokers and provides anonymity to subscribers as well. Anonymity is achieved because each hop does not know if the previous hop is the origin of a message or merely a forwarding node. The scheme works under a crash failure model.

Algorithm 4 Initial values for a node p joining a layer.

$id_p \leftarrow max_{i \in \mathcal{N}(p)}(id_i) + 1$
if $\forall\ (j,k) \in \mathcal{N}(p),\ val_j = val_k$ **then**
    $val_p \leftarrow (val_j + 1)\ mod\ 3$
else if $\cup_{j \in \mathcal{N}(p)} val_j = \{0,1,2\}$ **then**
    $val_p \leftarrow min_{i \in \mathcal{N}(p)}(val_i)$
else
    $val_p \leftarrow 0$
end if

A key concept to the work is that of a layer, which is a logical directed acyclic graph. A layer is created for each topic in the pub/sub network, with each node representing a client in the network. Layers dictate which nodes (called sink nodes) are allowed to send and forward messages at a given time. Sink nodes can tell they are sink nodes by using data available locally and on the logically neighbouring nodes in the layer. Sink nodes change over time. The scheme guarantees starvation freedom since every node will eventually get a chance to send a message.

Every node within a layer has an integer identifier id and a value $val \in \{0,1,2\}$. A layer is constructed by incrementally adding nodes in a way that the acyclic property is preserved. Edge orientation for the edge between two neighbouring nodes p and q is determined by the following property:

$$p \rightarrow q \Leftrightarrow (val_p = (val_q + 1)\ mod\ 3) \vee (val_q = val_p \wedge id_q < id_p)$$

When a new node joins a layer, it will choose any n neighbours, where n is chosen by the number of crash failures that are to be supported by the system. Up to $n-1$ nodes may fail without disconnecting the network. The new node's initial values for id and val are given by Algorithm 4, where $\mathcal{N}(p)$ is the set of neighbouring nodes of p. The identifier is set to an ever-increasing integer. A sink node is a node with all adjacent edges in the layer pointed towards it. At least one sink node is guaranteed to exist within a layer.

After message transmission, sink nodes execute an update step that has the effect of re-orienting adjacent edges. The value val is updated according to Algorithm 5. Once that is done, the adjacent edges are re-oriented according to the edge orientation property mentioned previously. Every node is guaranteed to eventually become a sink node under this scheme.

Unfortunately, the authors do not provide an implementation to evaluate their work so the performance and scalability of this approach are unknown.

6.2 Data Anonymity

We define data anonymity as the aspect of anonymity that deals with the identification of actors based on evaluation of the data itself. Within the context of pub/sub, this encompasses the identification of actors based on the data contained in advertisements, subscriptions and events. In general, these techniques involve partial information hiding rather than the full encryption of data.

Algorithm 5 Value update step for sink node p.

if $\forall\ j \in \mathcal{N}(p),\ val_j = val_p$ **then**
  $val_p \leftarrow (val_p + 1)\ mod\ 3$
else if $\exists\ j \in \mathcal{N}(p),\ val_j = (val_p + 1)\ mod\ 3$ **then**
  $val_p \leftarrow max_{i \in \mathcal{N}(p)}(val_i)$
end if

The literature has used k-anonymity [108] and ℓ-diversity [60] in the anonymization of events [126] and subscriptions [90]. k-anonymity is a collection of techniques for the

anonymization of data that was originally introduced within the context of databases [108]. Among these techniques are generalization and suppression. Generalization is the replacement or otherwise recoding of a value with a less specific but semantically consistent value. Suppression involves not releasing a particular value at all. A k-anonymized dataset is one in which any given record is indistinguishable from $k-1$ other records. ℓ-diversity further expands on k-anonymity by ensuring the diversity of values in the original dataset (also called microdata).

Zhou et al. [126] provide privacy-preserving stream dissemination through partial information hiding by leveraging k-anonymity in pub/sub. The work also details how it can be extended to adopt ℓ-diversity, though the presented implementation does not do so. The motivation behind the work is a system like healthcare where the various data sinks might be authorized to access different anonymity levels of some dataset. The threat model used is one of subscribers with varying levels of trust and fully trusted brokers. Brokers apply *version derivation* to anonymize events efficiently in transit based on the subsribers' trust. Version derivation is the process of generating a k′-anonymous dataset from a k-anonymous dataset where $k' \geq k$. The proposed solution can achieve user specified latency bounds in data with the trade-off that users might receive over-anonymized events (i.e., recipients receive a k′-anonymous event instead of the k-anonymous dataset they have access to where $k' > k$).

Rao et al. [90] adapt k-anonymity and ℓ-diversity for use with pub/sub by anonymizing subscriptions. Matched subscriptions will lead to events being delivered to some subscribers that are not interested (false positives), masking the identity of the subscriber that is interested in those events. The authors motivate the work by introducing a collusion attack between untrusted brokers and compromised subscribers that is unmitigated by traditional encryption techniques. Colluding entities work together to identify the interests of subscribers by having untrusted brokers join their knowledge of event routing with compromised subscribers that have full access to the event by subscribing to it. This attack is mitigated through data anonymity by having subscribers first send their subscriptions to a trusted anonymization engine which cloaks the subscription by anonymizing and generalizing it along with other subscriptions in the system. When

there is a match with a cloaked subscription, brokers must forward the event to at least k subscribers. In this way, the identity of the interested subscriber is protected since colluding entities do not know which of the subscribers is truly interested in the event. Subscribers must do an additional matching phase with their uncloaked subscriptions in order to filter out any false positive events they may have received. This technique could easily be coupled with the existing confidentiality techniques mentioned in Chapter 4 for additional privacy guarantees. This solution strictly covers the implementation of ℓ-diversity based on the different values that an attribute in a subscription may take. It does not cover other forms of ℓ-diversity which are not easy to implement without knowledge of the events being generated. For example, it does not cover how to implement probabilistic, entropy or recursive ℓ-diversity.

Chapter 7

Integrity

In this chapter, we outline a range of attacks that specifically target pub/sub systems as well some techniques for mitigating these attacks. These attacks threaten the integrity of the pub/sub system directly by threatening its availability or by gathering data about the system and probing for weaknesses that could be exploited in further attacks. The attacks found in the academic literature are not numerous, likely due to a lack of research in this area as well as the lack of a standard approach to addressing security issues rather than the overall coverage of this subject by the existing literature. Both Wun et al. [117] and Aniello et al. [4] support this conclusion and argue that this could slow the adoption of pub/sub into enterprise applications.

There are network attacks that target general communication systems. Replay attacks, message injection and message dropping are examples of this. These general attacks tend to have mitigation strategies that can be applied to pub/sub systems. In this chapter, we do not focus on these attacks. We limit ourselves to the literature that has been specifically developed for pub/sub systems. To the best of our knowledge, there are three attacks developed or analyzed within the context of pub/sub systems integrity in the literature. DoS attacks as classified by Wun et al. [117], the overlay scan attack for inferring the topology of a broker network by Aniello et al. [4] and bogus broker attacks from Tarkoma [111].

Wun et al. [117] introduce a taxonomy of Denial of Service (DoS) attacks in pub/sub. We borrow from this work the elements that can be universally applied to any pub/sub

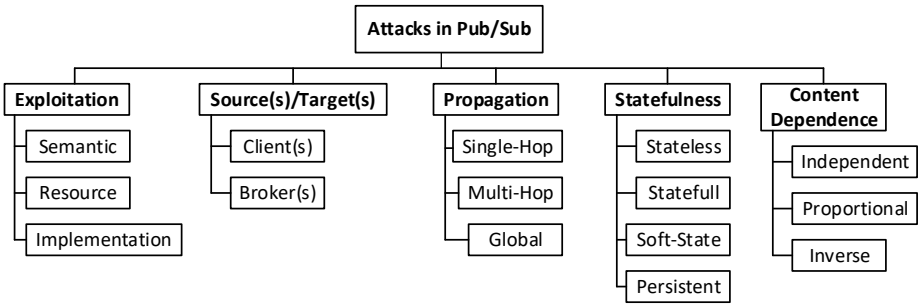


Figure 7.1: A taxonomy of attacks in pub/sub [117].

attack scenario and present it in Figure 7.1. A pub/sub attack exploits various weaknesses in a pub/sub system (e.g., resource limitations, semantic weaknesses or implementation flaws). The source(s) and target(s) of an attack can be the various entities within pub/sub. Propagation refers to the message routing properties of the attack within the pub/sub network, including single-hop between two brokers, multi-hop through an arbitrary number of brokers and global which propagates to all brokers. Statefulness refers to the lasting effects of the attack after the adversary has stopped all direct activity and the system's ability to recover (both automatically and through direct intervention). The content of a message determines how it is processed and delivered within a pub/sub system. Content dependence refers to a property that reflects how the attack depends on the content of the message.

7.1 Denial of Service

Denial of Service (DoS) is a form of attack where the perpetrator seeks to make a resource within the pub/sub system unavailable. These attacks have remained prominent in the Internet despite continuous defensive efforts [117]. Wun et al. [117] note that this form of attack displays two characteristics that are unique to the pub/sub domain:

1. Localization: Heavy message loads cause bottlenecks in brokers, which prevent the propagation of DoS attacks to other regions of the network.

2. Transmission: The ability of an attack to induce effects at distant brokers without affecting brokers along the path of propagation.

Localization is a benefit to pub/sub systems since DoS counter-measures can take advantage of localization to achieve high availability. The second characteristic is something that systems must carefully guard against since transmission enables dangerous remote attacks from potentially arbitrary points in the network. As a direct consequence, early detection of DoS attacks can be much more difficult [117].

The taxonomy presented in Figure 7.1 is a subset of the taxonomy presented by Wun et al. [117] for classifying DoS attacks on pub/sub systems. We omitted the component techniques category, which are reusable techniques which DoS attacks may make use of to mount an attack. The reason for this is that the examples provided where specific only to DoS attacks.

7.2 Overlay Scan Attack

The Overlay Scan Attack as presented by Aniello et al. [4] is an attack that a malicious user could use to infer the internal topology of a system. On its own, the attack is not designed to bring down a network, but rather for an adversary to probe the network in an attempt to find potential weaknesses and plan a future attack. The topology inference is performed by only using the standard primitives provided by the pub/sub middleware and assuming minimal knowledge on the target system.

In this attack, malicious subscribers subscribe to events generated by malicious publishers. The publishers then generate shaped traffic into the broker network by publishing events (i.e., multi-hop propagation). The malicious subscribers observe event latency changes along with time. Through these observations, it is possible to infer bottlenecks or otherwise overworked internal brokers within the network. Furthermore, it is possible to infer the broker topology. The event content is largely independent as long as it matches the subscriptions of colluding subscribers.

This form of attack requires pushing the pub/sub system to the saturation point, which is defined as being close to the point where at least one broker is working slightly above its maximum computational rate. As various delays are measured and analyzed, it is possible to pinpoint the position of the saturated broker in the network. If saturation

cannot be reached at a particular broker, then the attack will not be able to pinpoint that broker's position. These undetected brokers are referred to as invisible brokers. This attack works despite the presence of application traffic generating noise and network links of varying latencies.

The attack works in three phases: the setup, path discovery and path merging phases. The setup phase involves initializing several malicious clients, connecting to various brokers across the target network and publishing/subscribing to each others' events on a portion of the event space that is generally unused by honest clients. The path discovery phase involves the clients publishing events to saturate the network and measuring event latency. By iteratively altering the clients that publish events, the attackers are able to determine a path of brokers involved in the message dissemination. The path merging phase eliminates the brokers present in multiple paths gathered from the path discovery phase and infers the topology of the broker network.

7.3 Bogus Broker Attacks

Tarkoma [111] outlines a series of attacks that are possible when brokers are compromised. These *bogus brokers* may collude with other compromised brokers to spam clients, monitor traffic, inject and/or drop messages. Depending on the broker network topology, the effects of this attack could cripple the capabilities of the network to disseminate legitimate messages. In the extreme, spam sent to clients could be considered a form of DoS attack. In general, the percentage of bogus brokers within the network is a key factor in mitigating this form of attack.

Prevention of selective message dropping attacks is difficult and requires redundancy in the routing topology [111]. For example, if a client is connected to the pub/sub network through a single broker which happens to be compromised, then the bogus broker could choose to arbitrarily drop select messages to and from this client. Since this broker is the single point of access to the broker network, the client does not know that this has happened. By connecting to multiple edge brokers, there is a greater probability that one of these brokers is not compromised and will not selectively drop messages. This

form of attack can generate partitions in the network if the bogus broker is the only node connecting one partition to another.

The confidentiality techniques presented in Chapter 4 can mitigate the effects of bogus brokers monitoring of traffic, but they do not conceal the fact that a match has occurred. This is described as the "attack at dawn" problem [89]. The problem can be mitigated by using filtering techniques that have a false positive matching rate. Oblivious Transfer, presented in Chapter 4.6, is a promising technique for addressing the "attack at dawn" problem. Although Crescenzo et al. [21] do implement Oblivious Transfer techniques into pub/sub, they do not explicitly attempt to solve this problem. Their solutions also has the additional cost of weakening the space decoupling property of pub/sub.

Unwanted messages (spam) can be prevented through the access control techniques presented in Chapter 5. Tarkoma [111] propose that nodes within the pub/sub network generate their own public/private key pair and place their public key into a trusted lookup service. This allows every entity to authenticate received messages, add their own authentication and forward them to the next hop. If the network determines that a particular node is sending spam, then it could be blacklisted from the pub/sub network. Every node would be responsible for authenticating the received messages. All messages that fail the authentication are dropped.

Chapter 8

HyShare Overview

In this chapter, we present the various aspects which form HyShare. First, we present the threat model for HyShare. Next, we introduce *ISSPS brokers*, a type of broker that runs a modified version of ISSPS. This is followed by our introduction of *SGX-enabled brokers*, which are used to greatly reduce the number of messages needed to share a secret. Together, these two types of brokers are the core foundation of HyShare. We conclude the chapter by discussing broker placement within the broker network topology.

ISSPS is appealing for its security guarantees but comes at the cost of a large number of messages required to share a secret. SGX is appealing due to the enclave's ability to provide a trusted execution environment but comes at the cost of hardware hegemony and a need for Intel's attestation services. With HyShare, we propose a compromise that uses ISSPS brokers to share secrets and SGX-enabled brokers to reduce the overall number of messages used to share a secret.

8.1 Threat Model

In our secret sharing work, we consider a stronger threat model than the honest-but-curious threat model commonly used by the existing literature. This threat model considers a broker network composed of mostly honest-but-curious brokers and some number of brokers that behave arbitrarily by altering or dropping messages. We refer to the latter kind of broker as a *Byzantine broker*. Additionally, brokers may collude by sharing data

to try to compromise the secret being shared. We assume the problem of achieving a live broker network (i.e., that a path from any client to any other client in the system that does not go through a Byzantine broker exists) to be solved through sufficient redundancy in the number of brokers.

In order to route secret sharing messages in content-based pub/sub, we use a model similar to the one used by Yuen et al. [124] and Srivatsa et al. [107]. Under this model, although there are sensitive event attributes to be protected by encryption, these values are not used to make routing decisions. Routing decisions are made by the rest of the event attributes which are left in plaintext. This model is used to simplify our work and is not a requirement. Pires et al. [85] have implemented secure content-based routing using SGX on purely encrypted attributes.

8.2 ISSPS Collusion Tolerance

In this section, we revisit the issue of collusion tolerance in ISSPS. We specifically show the ratio of colluding to non-colluding brokers that must exist in order to share a secret. Informed by this ratio, a system administrator can make a rational decision of how to provision the resources in the pub/sub system in order to be able to tolerate a given number of colluding brokers without compromising the secrets exchanged, at the cost of increased redundancy within the system and the required processing power. Colluding brokers are weaker than Byzantine brokers in that they do not deviate from the protocols but may reveal or otherwise leak information that they process. Collusion is a subset of the operations which a Byzantine broker may perform. The claims made by Yoon and Kim [123] regarding Byzantine brokers still hold, but we revisit the protection against collusion within the system which was largely omitted in the previous work.

We first show that in the worst case scenario, ISSPS can tolerate up to $k-1$ colluding brokers. We then show that the worst case scenario arises from the scenario in which all PBs within an LB are compromised. Since this scenario is somewhat contrived, we go on to show that a relatively large number of PBs may be compromised (well in excess of k), without necessarily compromising the original secret.

Theorem 1. *ISSPS can tolerate up to $k-1$ colluding PBs in the worst case scenario without compromising the original secret value D.*

Proof. Assume by way of contradiction that any $k-1$ PBs collude to learn D. From ISSPS, each PB receives at most k fragments, each corresponding to a different application of the secret sharing scheme. This implies that $k-1$ colluding brokers can only learn of at most $k-1$ fragments for any individual secret. Therefore, by the starting assumption, the colluding brokers must have regenerated D from $k-1$ fragments, which contradicts the secret sharing scheme's claim that at least k fragments are required to regenerate a secret. □

In general, SSPS is weaker than ISSPS by imposing more of the broker network topology. In ISSPS, due to the repeated application of Shamir's secret sharing scheme, k PBs within a single LB must be compromised in order for the original secret to be compromised. Fragments from different LBs cannot be combined to regenerate the original secret. This is not true in SSPS, where the PBs may be from different LBs and could collude to compromise D.

Theorem 2. *If all PBs in ISSPS contain at least one non-colluding broker then D is secure.*

Proof. If there exists one non-colluding broker at every broker, and the rest are assumed to be colluding to try to learn D, then there must exist a communication path from the publisher to the subscriber composed of only non-colluding brokers. Let us denote this path by H.

Each vertex in H represents an entity in the path (publisher, broker or subscriber). During the execution of ISSPS, each edge would represent the transmission of a different fragment. Each edge links two vertices, one representing the source entity which generated the fragment and the other the sink entity which receives it. The edge value is the value of the fragment being transmitted at that stage in the communication channel.

Since communication links are secure, each edge in H represents a fragment that is known only by the source and sink entities (i.e. it is only known by non-colluding

entities). Since only k fragments exists, by Shamir's secret sharing scheme, the colluding PBs cannot learn of the value of the previous edge that the source entity received and used to generate this new fragment. This argument can be applied backwards through H until the source is reached, with the conclusion that the colluding brokers are missing one of the required fragments needed to regenerate D. □

By the previous theorems, it follows that ISSPS can tolerate the following ratio of colluding PBs to total PBs:

$$\frac{|ColludingBrokers|}{|TotalBrokers|} \leq \frac{k-1}{k}$$

This ratio only holds in the best case scenario. The worst case scenario for ISSPS only tolerates up to $k-1$ colluding brokers. If k physical brokers within a single ISSPS broker collude, they would gain access to sufficient fragments to regenerate D.

8.3 ISSPS Brokers

ISSPS brokers run a modified version of ISSPS. Figure 8.1 shows the high-level overview of the relevant parts of an ISSPS broker. In ISSPS, when a message is received, it goes into an input queue. When a message is ready to be processed, it is dequeued. The metadata is used to make routing decisions. The broker must determine whether the next hop is another broker or a subscriber. If the next hop is a broker, then the input fragment must undergo another iteration of Shamir's secret sharing scheme, splitting the fragment into n new fragments to be forwarded to the various brokers within the next hop's logical broker. If the next hop is the subscriber, then no further fragmentation is necessary and the broker can just forward the received fragments directly. Once processed, these messages are placed in an output queue for transmission.

In HyShare, the next hop has one additional possible value. The next hop could be either an ISSPS broker, a subscriber or an SGX-enabled broker. The first two cases do not deviate from ISSPS. Note that a broker will always know if the next-hop broker is SGX-enabled due to enclaves running their attestation protocol when connecting to other

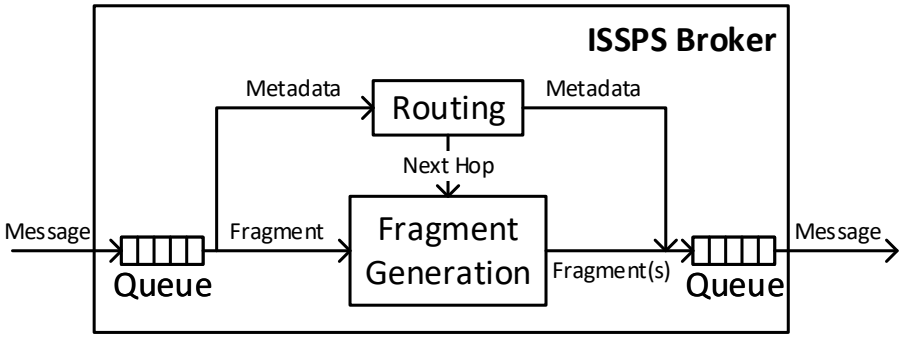


Figure 8.1: High level ISSPS Broker Overview.

entities. As part of attestation, a session key is established for the secure transmission of messages to and from an enclave. If the next hop is indeed an SGX-enabled broker then the current broker does not need to generate new fragments from the received message. Instead, fragments must be encrypted using the session key before transmission. An additional difference is that incoming messages may have an encrypted fragment if it was sent by an SGX-enabled broker. The session key established during attestation when first connecting to the SGX-enabled broker must be used to decrypt fragments received from an SGX-enabled broker.

8.4 SGX-Enabled Brokers

SGX-enabled brokers are restricted to using relatively new and proprietary Intel hardware when compared to an ISSPS broker, which has minimal hardware requirements. The trade-off to such restrictions is that it has the capability to execute code in a trusted executing environment. Within an enclave, secrets can be safely regenerated from fragments and then split into fragments again without sacrificing their integrity. In addition, there is no need for system calls or a significant amount of memory to do this. This point is significant as SGX disables system calls within enclaves and the memory space is limited to 128 MB pages which have a significant overhead when evicted from the processor as they must be encrypted [20]. Since fragments are bounded in size and the code for implementing Shamir's secret sharing scheme is 29.4 kB, it is reasonable to assume that all fragments necessary to regenerate a secret will fit within a single enclave page.

When bringing up the broker network for the first time or when adding an SGX-enabled broker to an existing broker network, it must attest to all entities it connects to. The reason for this is two-fold. First, it proves to the other entities that the broker is indeed executing its code from within an enclave, which is necessary as those entities behave differently if they are sending or receiving a message from an SGX-enabled broker. Second, this establishes a session key for encrypting messages sent to and from the SGX-enabled broker. The cost of attestation is incurred once when an SGX-enabled broker joins the broker network, for each entity it connects to, and once for each other entity that connects to it as part of regular execution.

Entities that send a secret or a fragment to an SGX-enabled broker do not need to generate fragments by running Shamir's secret sharing scheme. Instead, they only need to encrypt the secret or fragment they wish to transmit and send it to the SGX-enabled broker. Brokers receiving a message from an SGX-enabled broker need to execute the additional step of decrypting their received secret or fragment using their session key. Although an SGX-enabled broker receives all fragments required to regenerate a secret (and generates all n fragments, at least k of which are needed to regenerate a secret), the fragments cannot be read by the broker as they are always encrypted by a session key outside of the enclave.

Figure 8.2 shows a high level architecture of an SGX-enabled broker. We assume that the previous and next hop is an ISSPS broker and thus show the secret regeneration and fragment generation steps. The secret regeneration block is not necessary if the previous hop is another SGX-enabled broker or a client and the fragment generation block is not needed if the next hop is likewise an SGX-enabled broker or a client. Note that the input queue is not exactly a queue as we do not dequeue messages out of it until all fragments required to regenerate the original secret are received. We choose to wait for all fragments before entering the enclave in order to avoid excessive calls to enter and exit the enclave and their associated overheads. Although one could begin the decryption process and store the fragment within the enclave until k fragments are received, we advice against this as it would result in k enclave entries and exits.

To protect against adversaries with full physical access to the hardware and host

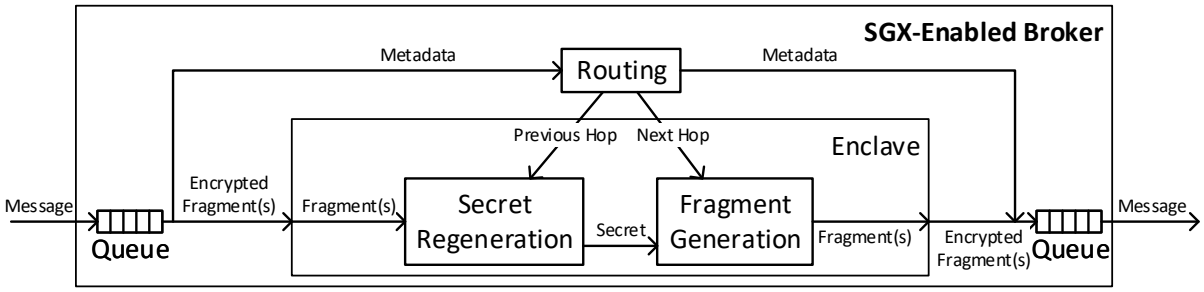


Figure 8.2: High level SGX-enabled broker overview.

operating system of the SGX-enabled brokers, the system administrator can replicate SGX-enabled brokers akin to what is done in ISSPS by turning the SGX-enabled broker into a logical broker composed of multiple physical brokers redundantly processing messages.

8.5 Broker Placement

The placement of SGX-enabled brokers within the broker network affects the overall reduction in the number of messages by HyShare. In ISSPS, the number of messages needed to share a secret grows exponentially in $\Theta(n^h)$, where h is the number of broker hops between communicating clients. We propose using enclaves to periodically regenerate the original secret as the fragments are being sent through the broker network, breaking up long chains of consecutive ISSPS brokers and reducing the ever growing number of fragments and messages. The more enclaves are used, the greater the reduction in the number of messages needed to share a secret among communicating entities. This forms the basis of our hybrid broker network composed of ISSPS brokers and SGX-enabled brokers.

In ISSPS, each hop in the process of secret sharing exponentially increases the total number of messages needed. Inserting an SGX-enabled broker in between the publisher and subscriber(s) effectively partitions the network. Secret sharing from publisher to subscriber is split into secret sharing from publisher to SGX-enabled broker and from SGX-enabled broker to subscriber(s). Since the number of hops in each individual secret sharing process is reduced, so too are the total number of messages needed to share

a secret. Asymptotically, the overall reduction is dominated by the *longest chain of unbroken ISSPS-broker-to-ISSPS-broker* (*LIBIB*) hops involved in the communication within the partitions made by SGX-enabled brokers. We illustrate this using an example.

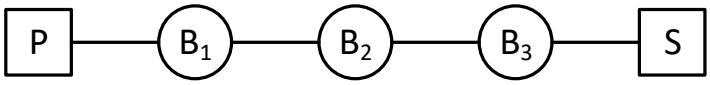


Figure 8.3: A simple topology pub/sub topology composed of a publisher, a subscriber and three brokers.

Figure 8.3 shows a simple pub/sub network. Assuming the brokers in this network are LBs then, using ISSPS, the minimum number of messages needed to share a single secret is k at the first hop between publisher and B_1. B_1 then sends k^2 messages to B_2, which itself sends k^3 messages to B_3. At this point B_3 simply forwards all received messages to the subscriber. In total, this requires $2k^3 + k^2 + k$ messages, or $\Theta(k^3)$. We now show the improvement achieved by HyShare in such a system when using a single SGX-enabled broker. Using the same pub/sub network, but making B_1 an SGX-enabled broker, then the minimum number of messages needed to share a secret is $2k^2 + 2k$, or $\Theta(k^2)$. Note that in this case, making B_1 the SGX-enabled broker is symmetrically equivalent to making B_3 the SGX-enabled broker. This is not necessarily the case in more complex network topologies. While a reduction in messages has occurred, it is not the optimum case for a single SGX-enabled broker system in this specific example. The lowest number of messages is achieved when B_2 is the SGX-enabled broker. In such a case, the minimum number of messages needed to share a secret is $4k$, or $\Theta(k)$. The improvement is due to the reduction in the LIBIB hop count. Whereas when B_1 is the SGX-enabled broker, the LIBIB hop is two, when B_2 is the SGX-enabled broker then LIBIB is zero. When LIBIB is zero, the number of messages needed to share a secret scales linearly.

We note that, if the resources are available, it is possible to construct a broker network composed of only SGX-enabled brokers. In this case, there is no need for ever generating fragments as the session key generated from attestation is sufficient in transmitting a secret end-to-end. In such a case, the number of messages needed scales linearly with the number of secrets to be shared. The disadvantage of this approach is in the inflexibility

of the hardware used to construct an individual broker.

More generally, to maximize the reduction of messages, SGX-enabled brokers should be placed at highly trafficked locations within the topology. Optimal SGX-enabled broker placement is thus dependent on traffic patterns that are application dependent. If traffic distributions are known ahead of time and the broker network topology does not change during operation, then it is possible to statically place SGX-enabled brokers within the topology offline. For example, assuming uniform traffic distributions and a tree-based broker topology with publishers connected to the root and subscribers at leaf nodes, the most effective placement of SGX-enabled brokers is as close to the root node as possible. A system administrator looking to provision multiple SGX-enabled brokers into this broker system could simply replace ISSPS brokers with SGX-enabled brokers in the order given by a breadth first traversal of the broker network. In sparsely clustered topologies, brokers at the edges of clusters would make ideal SGX-enabled broker placement.

For more realistic pub/sub systems, the problem of SGX-enabled broker placement within the network becomes more complex. This problem is very similar to the well explored problem of load balancing, where additional resources are provisioned based on observed traffic patterns and bottlenecks during execution. Due to the modular nature of its components, existing work on load balancing in pub/sub can be used with HyShare with some modifications. For example, Cheung and Jacobsen [17] use a distributed load detector to find bottlenecks in the network based on some performance metrics and provision additional brokers to offload an overloaded broker. To use such a system with HyShare, the performance metrics would have to be adjusted based on the number of secrets being shared by any given broker and the number of fragments observed for any individual secret. This metric can be used by the load balancing system to insert SGX-enabled brokers to reduce the overall number of messages observed by a broker.

Chapter 9

HyShare Evaluation

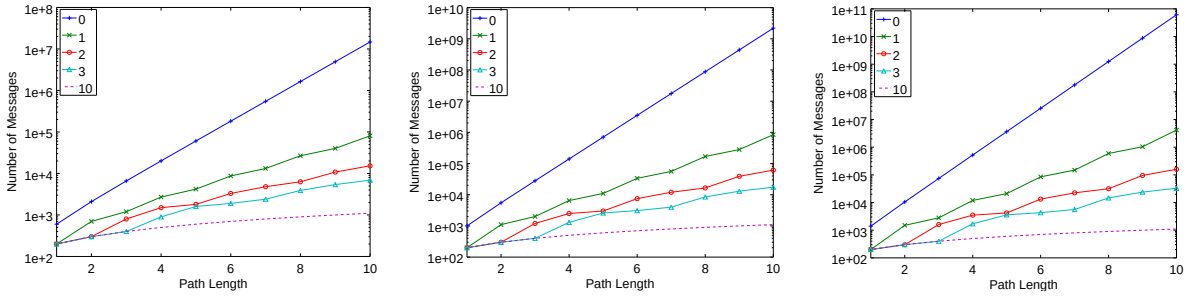
In Chapter 8, we proposed HyShare, a novel secret sharing scheme for pub/sub. In this chapter, we present our evaluations of HyShare. Since our work sets out to improve on ISSPS, we compare HyShare's performance to ISSPS's.

We have implemented HyShare in PADRES [53], a content-based pub/sub system. PADRES supports the direct connection between clients and multiple brokers. Additionally, PADRES supports cyclic broker topologies. Both of these features are requirements for our work since clients connect to multiple physical brokers when connecting to a logical broker and this creates cycles in the network graph.

9.1 Experimental Setup

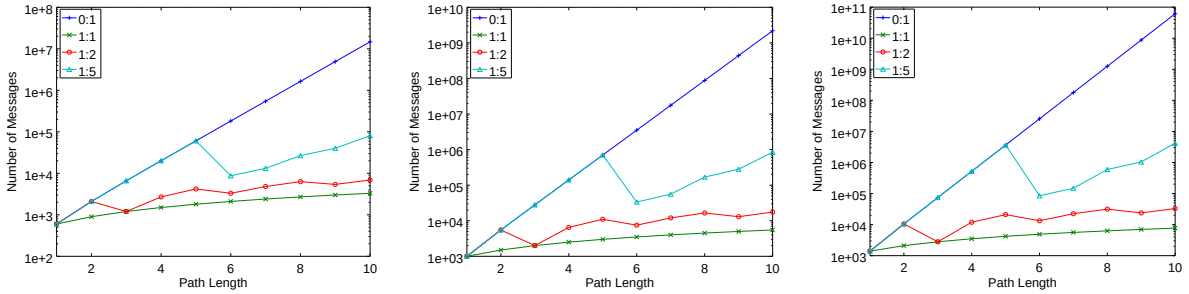
To simplify our evaluation, we set $k = n$. This means that all generated fragments by the publisher and intermediary brokers are required by an entity in order to regenerate the original secret. Although it is desirable for $n > k$ so that the system can tolerate dropped messages and Byzantine brokers, this is outside the scope of our performance evaluation. Each experiment was executed five times and results were averaged.

We ran HyShare on a cluster of 23 nodes. Each node has a 1.86 GHz Intel Xeon 5120 processor with 10 GB of memory. The nodes are physically interconnected using a switched 1 Gbps Ethernet network. PADRES is implemented in Java. We used the default launch configuration for PADRES, which sets the maximum memory allocation



(a) $k = 3$ (b) $k = 5$ (c) $k = 7$

Figure 9.1: Effect on the number of messages needed to share secrets with varying path length and constant number of SGX-enabled brokers.



(a) $k = 3$ (b) $k = 5$ (c) $k = 7$

Figure 9.2: Effect on the number of messages needed to share secrets with varying path length and maximum ratio of SGX-enabled brokers to ISSPS brokers.

of the JVM to 64 MB for clients and 512 MB for brokers. In the cases where more than 23 brokers are needed, each machine in the cluster runs multiple brokers. We set up our broker network so that no two adjacent or replicated brokers ever ran on the same physical node.

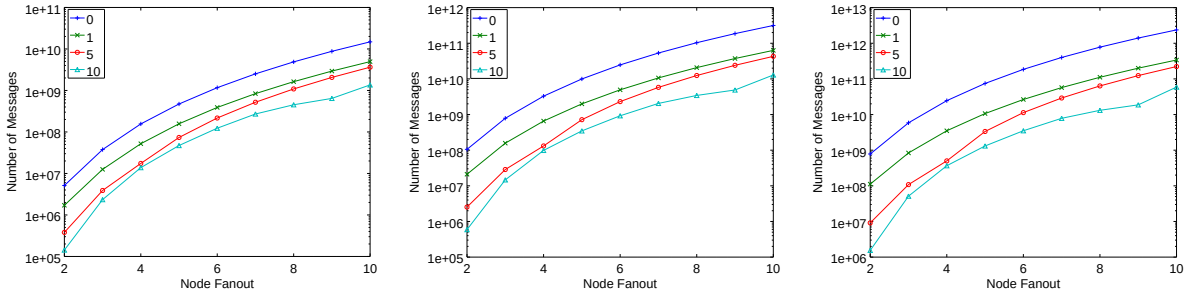
HyShare is almost entirely implemented in Java. The only exception is the code executing inside enclaves which is implemented using C++, as Java is not supported by SGX. This mixed implementation is significantly easier to implement than getting the JVM to run within an enclave, which would be a research problem on its own and well beyond the scope of this thesis. Enclaves were run using the SGX simulation mode provided by the SGX SDK.

9.2 Line Topology

We begin our evaluation by comparing the total number of messages required to share 400 secrets in HyShare for various numbers of SGX-enabled brokers in the hybrid broker network. For this experiment we use the simplest possible topology, consisting of a line of brokers with a publisher and subscriber at opposing ends. This illustrates the dramatic effects of path length on the total number of messages needed in order to share a secret among communicating entities. Since more complex topologies that have a comparable number of hops between clients will result in a greater number of messages needed to share a secret, this comparison forms a baseline for the number of messages saved by HyShare.

Hybrid broker networks are designed to reduce (or even halt in its entirety) the exponential growth of messages that is observed at each broker hop in ISSPS. In Figure 9.1 and Figure 9.2, we show the effects on the total number of messages needed to share secrets as the path length increases with varying number of SGX-enabled brokers. The three graphs in these figures show the same results with different input parameter k. More specifically, Figure 9.1 shows the total number of messages needed to share a secret when the maximum number of SGX-enabled brokers is held constant. The maximum number of SGX-enabled brokers of each series is identified in the legend. When there are no SGX-enabled brokers, the number of messages grows exponentially. A single SGX-enabled broker both dramatically and immediately slows the growth in messages as the path length increases. When the path length between clients is ten, the reduction in messages from replacing even a single ISSPS broker with one SGX-enabled broker in the topology is over two orders of magnitude. In the case where k is set to seven, the reduction in the number of messages needed to share a secret approaches four orders of magnitude. As the number of SGX-enabled brokers increases, the total number of messages needed to share a secret decreases. The dashed line series shows the maximum reduction in messages that occurs when the entire topology is composed of SGX-enabled brokers.

In Figure 9.2, we show the effects of maintaining a ratio of SGX-enabled brokers to



(a) $k = 3$ (b) $k = 5$ (c) $k = 7$

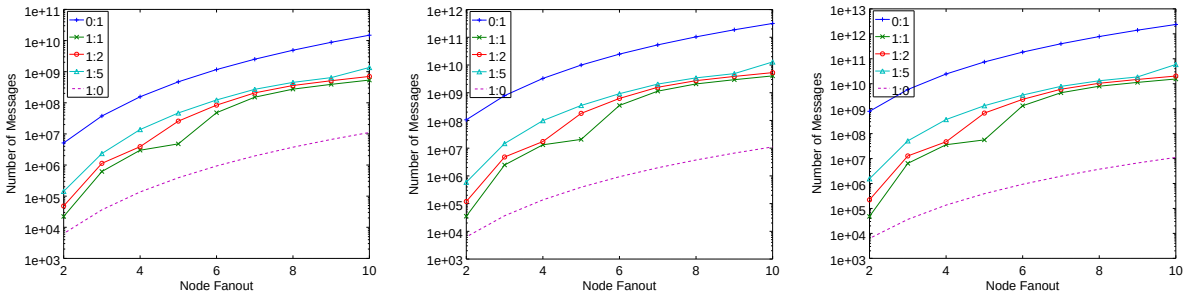
Figure 9.3: Effect on the number of messages needed to share secrets with varying node fanout and constant number of SGX-enabled brokers.

ISSPS brokers less than or equal to the specified ratio in the legend. Since the ratios are strictly enforced, the benefits of using HyShare are not observed until there are enough brokers in the network to add an SGX-enabled broker without exceeding the ratio. This is why the various series show no improvement until a certain number of hops is reached. A linear growth in the number of messages needed is achievable when the ratio of SGX-enabled brokers to ISSPS brokers reaches $1 : 1$. Higher ratios of SGX-enabled brokers result in a $\Theta(1)$ message reduction. As before, we observe that a single SGX broker has a significant impact on the number of messages needed to share secrets. This effect can be seen most dramatically in the $1 : 5$ series, in which the first SGX-enabled broker is added to the broker network when the path length reaches six.

9.3 Stock Quote Dissemination

A classic motivating example for the need of secret sharing in pub/sub is that of a stock quote notification system [114, 76]. Publishers are stock markets issuing quotes, while subscribers are investors interested in receiving these quotes. A content-based pub/sub system can be used for filtering quotes according to the subscriber's interests, but it requires that subscribers reveal sensitive information about their interests and investment strategy to an untrusted broker network. This is the driving motivation behind this experiment, which uses real-world stock quotes to evaluate HyShare.

The broker topology resembles a tree topology, with a number of well-connected



(a) $k = 3$ (b) $k = 5$ (c) $k = 7$

Figure 9.4: Effect on the number of messages needed to share secrets with varying node fanout and maximum ratio of SGX-enabled brokers to ISSPS brokers.

brokers at the root of the tree and subscribers at the leaf nodes. The publishers connect to these root brokers. We keep the average path length fixed to five brokers and vary the average *node fanout* (i.e., the number of outgoing connections to each broker).

The results of this experiment are shown in Figure 9.3 and Figure 9.4. As before, the subfigures (a), (b) and (c) represent the experiment executed using various values for the input parameter k. The legend of Figure 9.3 shows the number of SGX-enabled brokers within the topology, and its effects as the node fanout is increased. In Figure 9.4, the legend shows the maximum number of SGX-enabled brokers to ISSPS brokers as a ratio. The $0:1$ ratio denotes a broker network containing no SGX-enabled brokers, while the ratio $1:0$ denotes a broker network exclusively composed of SGX-enabled brokers. From these graphs, it is possible to see how even a few SGX-enabled brokers within the broker network have the ability to greatly reduce the overall number of messages needed to share secrets.

In Figure 9.5, we show the overhead incurred by various broker operations. ISSPS brokers only have to split received fragments, so we show the time taken to generate fragments from an original secret with a size of 256 bits. Splitting a secret is a very fast operation, requiring only the generation of random values and evaluating the generated function. Since enclaves do not allow system calls, we measure the entire time elapsed within an enclave with secret sizes of 128 and 256 bits. While the enclaves are significantly slower than the splitting operations performed at ISSPS brokers, the splitting operation needs to be executed for every fragment received. Thus, the latencies introduced from

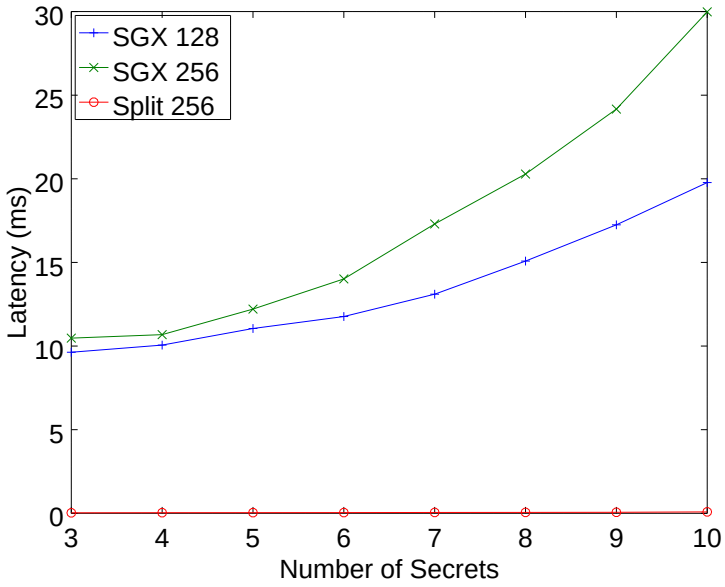


Figure 9.5: Latencies comparing SGX-enabled broker operations to ISSPS broker operations for various key sizes.

the use of SGX-enabled brokers should consider the overall reduction in the amount of messages in the system and the decreased number of splitting operations that must be performed.

Chapter 10

Conclusions

In this thesis, we have presented the security literature available within the domain of pub/sub. We have categorized, summarized and presented the weaknesses of the academic literature. In addition, we have presented HyShare, our own novel work that addresses the problem of sharing a secret amongst communicating clients without relying on an unilaterally trusted, universally available, out-of-band service to do this.

In Chapter 4, we presented the confidentiality literature. This includes a variety of techniques for preventing information from being leaked to trusted-but-honest brokers or eavesdroppers while still retaining some ability to route events to the subset of interested subscribers. Common to most of the solutions presented is the assumption that there exists a mechanism for the initial sharing of security parameters, shared secrets and/or keys. The literature often assumes the presence of a service that operates similar to a Certificate Authority. In other words, that an unilaterally trusted, universally available, out-of-band service exists to share some prerequisite values amongst communicating clients. The existence of such a service is an assumption that is addressed, in part, by our own work on HyShare. Although there are several techniques developed for ensuring confidentiality in pub/sub, we note that with the exclusion of ISSPS (the work we extend), all the techniques operate under the honest-but-curious threat model. As such, the honest-but-curious threat model with its relatively weak adversary has become the limits of the current research. We believe that future research should tackle stronger threat models. Supporting stronger threat models would increase the trust that users

and developers have in the system and would encourage the use of cloud services with pub/sub.

Chapter 5, we summarize the authorization literature that deals with permissions within pub/sub. This includes the arbitrary control over which entities within the system are allowed to send or receive information. Broadly speaking, the literature falls into two categories: client and broker access control. As we show in Table 5.1, none of the literature presented allowed for the full arbitrary control of all messages within content-based pub/sub.

Chapter 6 presents the anonymization literature, detailing how to hide the identity of entities within the pub/sub system. The literature falls into two categories: communication and data anonymity. In communication anonymity, there are no existing solutions for the anonymization of brokers. This would be necessary for a fully anonymous pub/sub system that operates akin to Freenet [46]. In addition, the existing work on adapting onion routing for use with pub/sub begs the question of how garlic routing could be adapted as well. In data anonymity, there are extensions to ℓ-diversity for which further development is needed before they can be used with pub/sub. Typically, applying ℓ-diversity to data requires full knowledge of the data. This is not possible for events, which are generated dynamically and are not known ahead of time. Adaptation of these techniques for use with pub/sub will require additional research.

Chapter 7 presents the integrity literature. It outlines various attacks and mitigation strategies for pub/sub. There is a lack of papers on this subject, likely due to the lack of research on the subject rather than the inherent integrity of pub/sub. Additional research is required into vulnerabilities that are exposed when using pub/sub.

In Chapter 8 and Chapter 9, we introduced and evaluated HyShare, our novel secret sharing solution. HyShare uses a hybrid broker network composed of ISSPS brokers and SGX-enabled brokers used in the sharing of a secret. ISSPS brokers can be implemented relatively inexpensively using existing cloud services, but on their own require a large number of messages to share a secret. SGX-enabled brokers are restricted to Intel's hardware and architecture but can dramatically reduce the overall number of messages needed to share a secret. In addition, we expand on the existing work by strengthening

the security claims of secret sharing in pub/sub. As future work, we want to consider the implications of having the entire broker network composed of SGX-enabled brokers. Although we briefly present the reduction in the total number of messages needed to share secrets, we believe that there are more benefits to be gained from using trusted execution environments in pub/sub. Additionally, we want to further expand on the problem of dynamic broker placement within the broker network based on observed traffic patterns.

Bibliography

[1] Allegro. Hermes: Reliable and easy to use message broker built on top of kafka, 2015.

[2] Amazon Web Services. What is pub/sub messaging?, 2018.

[3] Emmanuelle Anceaume, Ajoy K. Datta, Maria Gradinariu, and Gwendal Simon. Publish/subscribe scheme for mobile networks. In *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing*, POMC '02, pages 74–81, New York, NY, USA, 2002. ACM.

[4] Leonardo Aniello, Roberto Baldoni, Claudio Ciccotelli, Giuseppe Antonio Di Luna, Francesco Frontali, and Leonardo Querzoni. The overlay scan attack: Inferring topologies of distributed pub/sub systems through broker saturation. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 107–117, New York, NY, USA, 2014. ACM.

[5] Apache Software Foundation. Apache pulsar, 2018.

[6] Jean Bacon, David Evans, David M. Eyers, Matteo Migliavacca, Peter Pietzuch, and Brian Shand. *Enforcing End-to-End Application Security in the Cloud*, pages 293–312. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[7] Jean Bacon, David Eyers, and Jatinder Singh. Securing event-based systems. In *Principles and Applications of Distributed Event-Based Systems*, pages 119–139. IGI Global, 2010.

[8] Jean Bacon, David M. Eyers, Jatinder Singh, and Peter R. Pietzuch. Access control in publish/subscribe systems. In *Proceedings of the Second International Conference on Distributed Event-based Systems*, DEBS '08, pages 23–34, New York, NY, USA, 2008. ACM.

[9] Feng Bao, Robert H. Deng, Xuhua Ding, and Yanjiang Yang. Private query on encrypted data in multi-user settings. In *Proceedings of the 4th International Conference on Information Security Practice and Experience*, ISPEC'08, pages 71–85, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] R. Barazzutti, P. Felber, H. Mercier, E. Onica, and E. Riviere. Efficient and confidentiality-preserving content-based publish/subscribe with prefiltering. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1, 2015.

[11] Luciano Barreto, Leomar Scheunemann, Joni Fraga, and Frank Siqueira. Secure storage of user credentials and attributes in federation of clouds. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 364–369, New York, NY, USA, 2017. ACM.

[12] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

[13] 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, pages 213–229, London, UK, UK, 2001. Springer-Verlag.

[14] Dell Cameron. The great data breach disasters of 2017, December 2017.

[15] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, August 2001.

[16] Weifeng Chen, Jianchun Jiang, and N. Skocik. On the privacy protection in publish/subscribe systems. In *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on*, pages 597–601, June 2010.

[17] Alex King Yeung Cheung and Hans-Arno Jacobsen. Load balancing content-based publish/subscribe systems. *ACM Trans. Comput. Syst.*, 28(4):9:1–9:55, December 2010.

[18] Sunoh Choi, Gabriel Ghinita, and Elisa Bertino. *A Privacy-Enhancing Content-Based Publish/Subscribe System Using Scalar Product Preserving Transformations*, pages 368–384. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[19] Clifford Cocks. *An Identity Based Encryption Scheme Based on Quadratic Residues*, pages 360–363. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[20] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[21] Giovanni Crescenzo, Brian Coan, John Schultz, Simon Tsang, and Rebecca N. Wright. Privacy-preserving publish/subscribe: Efficient protocols in a distributed model. In *Revised Selected Papers of the 8th International Workshop on Data Privacy Management and Autonomous Spontaneous Security - Volume 8247*, pages 114–132, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[22] Heidi Daitch. 2017 data breaches – the worst so far, December 2017.

[23] Ajoy Kumar Datta, Maria Gradinariu, Michel Raynal, and Gwendal Simon. Anonymous publish/subscribe in p2p networks. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 22–26, 2003.

[24] Giovanni Di Crescenzo, Jim Burns, Brian Coan, John Schultz, Jonathan Stanton, Simon Tsang, and Rebecca N. Wright. *Efficient and Private Three-Party Publish/Subscribe*, pages 278–292. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[25] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, pages 74–89, Berlin, Heidelberg, 1999. Springer-Verlag.

[26] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 67–95, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[27] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[28] Shlomi Dolev, Juan A. Garay, Niv Gilboa, Vladimir Kolesnikov, and Yelena Yuditsky. Efficient private distributed computation on unbounded input streams. *CoRR*, abs/1208.4909, 2012.

[29] Changyu Dong, Giovanni Russello, and Naranker Dulay. Shared and searchable encrypted data for untrusted servers. In *Proceeedings of the 22Nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 127–143, Berlin, Heidelberg, 2008. Springer-Verlag.

[30] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[31] European Comission. 2018 reform of eu data protection rules, 2018.

[32] Chun-I Fan, Yi-Fan Tseng, and Chih-Wen Lin. Attribute-based encryption from identity-based encryption. *IACR Cryptology ePrint Archive*, 2017:219, 2017.

[33] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor,

Advances in Cryptology — CRYPTO' 93, pages 480–491, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[34] Anders Fongen and Federico Mancini. *Identity Management and Integrity Protection in Publish-Subscribe Systems*, pages 68–82. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[35] Jonathan Garber. Facebook's value drops almost $50 billion in 2 days, March 2018.

[36] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.

[37] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. *Hiding Routing information*, pages 137–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[38] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. *Universal Reencryption for Mixnets*, pages 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[39] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. *Universal Reencryption for Mixnets*, pages 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[40] Google Cloud Platform. What is google cloud pub/sub?, September 2017.

[41] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[42] Hadoop Wiki. Hedwig, January 2011.

[43] H. Harney and C. Muckenhirn. Group key management protocol (gkmp) architecture, 1997.

[44] H. Harney and C. Muckenhirn. Group key management protocol (gkmp) specification, 1997.

[45] Javier Herranz. Attribute-based encryption implies identity-based encryption. *IACR Cryptology ePrint Archive*, 2017:54, 2017.

[46] The Freenet Project Inc. Freenet, April 2017.

[47] Intel. Intel software guard extensions remote attestation end-to-end example, July 2016.

[48] Mihaela Ion, Giovanni Russello, and Bruno Crispo. *Supporting Publication and Subscription Confidentiality in Pub/Sub Networks*, pages 272–289. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[49] Mihaela Ion, Giovanni Russello, and Bruno Crispo. Design and implementation of a confidentiality and access control solution for publish/subscribe systems. *Computer Networks*, 56(7):2014 – 2037, 2012.

[50] Mihalea Ion, Giovanni Russello, and Bruno Crispo. An implementation of event and filter confidentiality in pub/sub systems and its application to e-health. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 696–698, New York, NY, USA, 2010. ACM.

[51] Vincenzo Iovino and Giuseppe Persiano. *Hidden-Vector Encryption with Groups of Prime Order*, pages 75–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[52] Information technology – Security techniques – Information security management systems – Overview and vocabulary. Standard, International Organization for Standardization, Geneva, CH, February 2016.

[53] Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh. *The PADRES Publish/Subscribe System*, pages 164–205. IGI Global, 2010.

[54] Ari Juels and Michael Szydlo. Attribute-based encryption: Using identity-based encryption for access control. 2004.

[55] Marek Klonowski and Bartek Rózanski. Privacy protection for p2p publish-subscribe networks. 2005.

[56] Panayiotis Kotzanikolaou, George Chatzisofroniou, and Mike Burmester. Broadcast anonymous routing (bar): scalable real-time anonymous communication. *International Journal of Information Security*, 16(3):313–326, Jun 2017.

[57] A. Lewko, A. Sahai, and B. Waters. Revocation systems with very small private keys. In *2010 IEEE Symposium on Security and Privacy*, pages 273–285, May 2010.

[58] Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsen. A Distributed Service Oriented Architecture for Business Process Execution. *ACM Transactions on the Web*, 4(1):2:1–2:33, January 2010.

[59] Jun Li, Chenghuai Lu, and Weidong Shi. An efficient scheme for preserving confidentiality in content-based publish-subscribe systems, 2004.

[60] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007.

[61] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 10:1–10:1. ACM, 2013.

[62] Hugues Mercier, Emanuel Onica, Etienne Rivière, and Pascal Felber. *Performance/Security Tradeoffs for Content-Based Routing Supported by Bloom Filters*, pages 129–140. Springer International Publishing, Cham, 2013.

[63] Robinson Meyer. The cambridge analytica scandal, in 3 paragraphs, March 2018.

[64] Matteo Migliavacca, Ioannis Papagiannis, David M. Eyers, Brian Shand, Jean Bacon, and Peter Pietzuch. Defcon: High-performance event processing with information security. In *Proceedings of the 2010 USENIX Conference on USENIX*

Annual Technical Conference, USENIXATC'10, pages 1–1, Berkeley, CA, USA, 2010. USENIX Association.

[65] Matteo Migliavacca, Ioannis Papagiannis, David M. Eyers, Brian Shand, Jean Bacon, and Peter Pietzuch. *Distributed Middleware Enforcement of Event Flow Security Policy*, pages 334–354. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[66] Z. Miklos. Towards an access control mechanism for wide-area publish/subscribe systems. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 516–521, 2002.

[67] Kazuhiro Minami, Adam J. Lee, Marianne Winslett, and Nikita Borisov. Secure aggregation in a publish-subscribe system. In *Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society*, WPES '08, pages 95–104, New York, NY, USA, 2008. ACM.

[68] Toshiyuki Miyamoto and Sadatoshi Kumagai. An optimal share transfer problem on secret sharing storage systems. In *Proceedings of the 5th International Conference on Parallel and Distributed Processing and Applications*, ISPA'07, pages 371–382, Berlin, Heidelberg, 2007. Springer-Verlag.

[69] MQTT. Mqtt, November 2014.

[70] 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, pages 28–39, New York, NY, USA, 2018. ACM.

[71] Mohamed Nabeel, Stefan Appel, Elisa Bertino, and Alejandro Buchmann. *Privacy Preserving Context Aware Publish Subscribe Systems*, pages 465–478. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[72] Christoph P. Neumann, Florian Rampp, Richard Lenz, and Michael Daum. A mediated publish-subscribe system for inter-institutional process support in health-

care. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 14:1–14:4. ACM, 2009.

[73] U.s. Department of Health and Human Services. Breach portal: Notice to the secretary of hhs breach of unsecured protected health information, May 2018.

[74] Melek Önen and Refik Molva. *Secure Data Aggregation with Multiple Encryption*, pages 117–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[75] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. Efficient key updates through subscription re-encryption for privacy-preserving publish/subscribe. In *Proceedings of the 16th Annual Middleware Conference*, Middleware '15, pages 25–36, New York, NY, USA, 2015. ACM.

[76] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. Confidentiality-preserving publish/subscribe: A survey. *ACM Comput. Surv.*, 49(2):27:1–27:43, June 2016.

[77] Lukasz Opyrchal and Atul Prakash. Secure distribution of events in content-based publish subscribe systems. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, Berkeley, CA, USA, 2001. USENIX Association.

[78] Lukasz Opyrchal, Atul Prakash, and Amit Agrawal. Supporting privacy policies in a publish-subscribe substrate for pervasive environments. *Journal of Networks*, 2(1):17–26, 2007.

[79] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 195–203, New York, NY, USA, 2007. ACM.

[80] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*, pages 223–238. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[81] Partha Pal, Greg Lauer, Joud Khoury, Nick Hoff, and Joe Loyall. *P3S: A Privacy Preserving Publish-Subscribe Middleware*, pages 476–495. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[82] Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil, Geoffrey Fox, Yan Yan, and Yi Huang. A framework for secure end-to-end delivery of messages in publish/subscribe systems. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID '06, pages 215–222, Washington, DC, USA, 2006. IEEE Computer Society.

[83] Alain Pannetrat and Refik Molva. Multiple layer encryption for multicast groups. In *CMS 2002, Communication and Multimedia Security Conference, September 26-27, 2002, Portoroz, Slovenia / Also published in the book : Advanced Communications and Multimedia Security /Borka Jerman-Blazic & Tomaz Klobucar, editors, Kluwer Academic Publishers, ISBN 1-4020-7206-6, August 2002*, Portoroz, SLOVENIA, 08 2002.

[84] I. Papagiannis, M. Migliavacca, P. Pietzuch, B. Shand, D. Eyers, and J. Bacon. Privateflow: Decentralised information flow control in event based middleware. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 38:1–38:2, New York, NY, USA, 2009. ACM.

[85] Rafael Pires, Marcelo Pasin, Pascal Felber, and Christof Fetzer. Secure content-based routing using intel software guard extensions. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, pages 10:1–10:10, New York, NY, USA, 2016. ACM.

[86] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, January 1978.

[87] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 1981. `https://eprint.iacr.org/2005/187`.

[88] Sandro Rafaeli and David Hutchison. A survey of key management for secure group communication. *ACM Comput. Surv.*, 35(3):309–329, September 2003.

[89] Costin Raiciu and David S Rosenblum. Enabling confidentiality in content-based publish/subscribe infrastructures. In *Securecomm and Workshops, 2006*, pages 1–11. IEEE, 2006.

[90] W. Rao, L. Chen, and S. Tarkoma. Toward efficient filter privacy-aware content-based pub/sub systems. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2644–2657, Nov 2013.

[91] RedisLabs. Redis, 2018.

[92] Amit Sahai and Brent Waters. *Fuzzy Identity-Based Encryption*, pages 457–473. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[93] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, Feb 1996.

[94] Vinay Setty, Gunnar Kreitz, Roman Vitenberg, Maarten van Steen, Guido Urdaneta, and Staffan Gimåker. The hidden pub/sub of spotify: (industry article). In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 231–240. ACM, 2013.

[95] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[96] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[97] Yogeshwer Sharma, Philippe Ajoux, Petchean Ang, David Callies, Abhishek Choudhary, Laurent Demailly, Thomas Fersch, Liat Atsmon Guz, Andrzej Kotulski, Sachin Kulkarni, Sanjeev Kumar, Harry Li, Jun Li, Evgeniy Makeev, Kowshik Prakasam, Robbert Van Renesse, Sabyasachi Roy, Pratyush Seth, Yee Jiun Song, Kaushik Veeraraghavan, Benjamin Wester, and Peter Xie. Wormhole: Reliable

pub-sub to support geo-replicated internet services. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 351–366. USENIX Association, 2015.

[98] Abdullatif Shikfa, Melek Önen, and Refik Molva. *Privacy-Preserving Content-Based Publish/Subscribe Networks*, pages 270–282. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[99] J. Singh, L. Vargas, and J. Bacon. A model for controlling data flow in distributed healthcare environments. In *2008 Second International Conference on Pervasive Computing Technologies for Healthcare*, pages 188–191, Jan 2008.

[100] J. Singh, L. Vargas, J. Bacon, and K. Moody. Policy-based information sharing in publish/subscribe middleware. In *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*, pages 137–144, June 2008.

[101] Jatinder Singh and Jean Bacon. Event-based data control in healthcare. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, pages 84–86, New York, NY, USA, 2008. ACM.

[102] Jatinder Singh and Jean Bacon. *Event-Based Data Dissemination Control in Healthcare*, pages 167–174. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[103] Jatinder Singh, David M. Eyers, and Jean Bacon. Credential management in event-driven healthcare systems. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, pages 48–53, New York, NY, USA, 2008. ACM.

[104] Jatinder Singh, David M. Eyers, and Jean Bacon. Disclosure control in multi-domain publish/subscribe systems. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 159–170, New York, NY, USA, 2011. ACM.

[105] M. Srivatsa and L. Liu. Secure event dissemination in publish-subscribe networks.

In *27th International Conference on Distributed Computing Systems (ICDCS '07)*, pages 22–22, June 2007.

[106] Mudhakar Srivatsa and Ling Liu. Scalable access control in content-based publish-subscribe systems. 2006.

[107] Mudhakar Srivatsa, Ling Liu, and Arun Iyengar. Eventguard: A system architecture for securing publish-subscribe networks. *ACM Trans. Comput. Syst.*, 29(4):10:1–10:40, dec 2011.

[108] Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):571–588, October 2002.

[109] M. A. Tariq, B. Koldehofe, and K. Rothermel. Securing broker-less publish/subscribe systems using identity-based encryption. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):518–528, Feb 2014.

[110] Muhammad Adnan Tariq, Boris Koldehofe, Ala' Altaweel, and Kurt Rothermel. Providing basic security mechanisms in broker-less publish/subscribe systems. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 38–49, New York, NY, USA, 2010. ACM.

[111] S. Tarkoma. Preventing spam in publish/subscribe. In *26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'06)*, pages 21–21, July 2006.

[112] Yuan Tian, Biao Song, Mohammad Mehedi Hassan, and Eui-nam Huh. An efficient privacy preserving pub-sub system for ubiquitous computing. *Int. J. Ad Hoc Ubiquitous Comput.*, 12(1):23–33, January 2013.

[113] Binh Vo and Steven Bellovin. *Anonymous Publish-Subscribe Systems*, pages 195–211. Springer International Publishing, Cham, 2015.

[114] Chenxi Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3940–3947, Jan 2002.

[115] Brent Waters. *Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization*, pages 53–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[116] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 139–152, New York, NY, USA, 2009. ACM.

[117] Alex Wun, Alex Cheung, and Hans-Arno Jacobsen. A taxonomy for denial of service attacks in content-based publish/subscribe systems. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems*, DEBS '07, pages 116–127, New York, NY, USA, 2007. ACM.

[118] Alex Wun and Hans-Arno Jacobsen. A Policy Management Framework for Content-based Publish/Subscribe Middleware. In *ACM Middleware*, pages 368–388, Newport Beach, CA, U.S., November 2007.

[119] Alex Wun and Hans-Arno Jacobsen. *A Policy Management Framework for Content-Based Publish/Subscribe Middleware*, pages 368–388. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[120] Alex Wun, Milenko Petrovic, and Hans-Arno Jacobsen. A System for Semantic Data Fusion in Sensor Networks. In *DEBS 2007*, pages 75–79, Toronto, Canada, June 2007.

[121] Jun Xu, Jinliang Fan, M. H. Ammar, and S. B. Moon. Prefix-preserving ip address anonymization: measurement-based security evaluation and a new cryptography-

based scheme. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 280–289, Nov 2002.

[122] Zhengdao Xu and Arno Jacobsen. Adaptive location constraint processing. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 581–592, New York, NY, USA, 2007. ACM.

[123] Young Yoon and Beom Heyn Kim. Secret forwarding of events over distributed publish/subscribe overlay network. *PLOS ONE*, 11(7):1–23, 07 2016.

[124] Tsz Hon Yuen, Willy Susilo, and Yi Mu. *Towards a Cryptographic Treatment of Publish/Subscribe Systems*, pages 201–220. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[125] Yuanyuan Zhao and D. C. Sturman. Dynamic access control in a content-based publish/subscribe system with delivery guarantees. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pages 60–60, 2006.

[126] Yongluan Zhou, Lidan Shou, Xuan Shang, and Ke Chen. Dissemination of anonymized streaming data. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, DEBS '15, pages 104–115, New York, NY, USA, 2015. ACM.