Secret Sharing in Pub/Sub Using Trusted Execution Environments

Javier Munster
Middleware Systems Research Group
University of Toronto
javier.munster@mail.utoronto.ca

Hans-Arno Jacobsen
Middleware Systems Research Group
University of Toronto

ABSTRACT

An essential security concern in the publish/subscribe paradigm is that of guaranteeing the confidentiality of the data being transmitted. Existing solutions require that some initial parameters, keys or secrets be exchanged or otherwise established between communicating entities before secure end-to-end communication can occur. Most existing solutions in the literature either weaken the desirable decoupling properties of pub/sub or rely on a completely trusted out-of-band service to disseminate these values. This problem can be avoided through the use of Shamir's secret sharing scheme, at the cost of a prohibitively large number of messages, scaling exponentially with the path length between publisher and subscriber. Intel's Software Guard Extensions (SGX) offers trusted execution environments to shield application data from untrusted software running at a higher privilege level. Unfortunately, SGX requires the use of Intel's proprietary hardware and architecture.

We mitigate these problems through HyShare, a hybrid broker network used for the purposes of sharing a secret between communicating publishers and subscribers. The broker network is composed of regular brokers that use Shamir's secret sharing scheme and brokers with SGX to reduce the overall number of messages needed to share a secret. By fine tuning the combination of these brokers, it is possible to strike a balance between network resource use and hardware heterogeneity.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; • **Software and its engineering → Publish-subscribe / event-based architectures**;

KEYWORDS

Publish/Subscribe, key management, confidentiality, secret sharing, Intel SGX

ACM Reference Format:
Javier Munster and Hans-Arno Jacobsen. 2018. Secret Sharing in Pub/Sub Using Trusted Execution Environments. In *Proceedings of ACM International Conference on Distributed Event-Based Systems (DEBS '18)*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3210284.3210290

1 INTRODUCTION

Pub/Sub has emerged as an efficient communication paradigm [15]. As a result of this, pub/sub has become prevalent in a wide variety of application domains, including social networking (e.g., Facebook's Wormhole [39]), music streaming (e.g., Spotify [37]), healthcare [4, 29, 41, 42], cyber-physical systems (e.g., location-based services [50], sensor networks [33, 49]), business process management [22], policy management [48] and stock quotes dissemination [47]. The prevalence of pub/sub is further reflected in the large number of open and close source implementations available, including Amazon Simple Notification Service [2], Apache Pulsar [3], Google Cloud Pub/Sub [16], HedWig [17], Hermes [1], MQTT [27], PADRES [20], Redis [35] and Siena [7].

The challenges of guaranteeing confidentiality in pub/sub have been known for some time [47]. That is, publishers and subscribers (collectively referred to as *clients*) may wish to keep sensitive information secret from the underlying pub/sub infrastructure (i.e., the broker network). The need for confidentiality becomes apparent when considering that developers may wish to take advantage of the processing power offered by using external (and not necessarily trusted) cloud computing services to provide the computing resources needed by the pub/sub infrastructure. Applications that would require such confidentiality guarantees are numerous. For example, a common concern among social network users is the privacy of their personal information. Additionally, regulations often dictate that some data (e.g., patient health records) must be encrypted before online transmission [5]. Encryption is also desirable in financial systems like stock quote dissemination services as an added layer of data protection, preventing unauthorized users from having access to the disseminated events and from having brokers disseminate events to unauthorized users [47].

Within the pub/sub literature, some encryption schemes require that communicating entities establish a secret key (e.g., symmetric-key encryption [8, 13, 23]), others require the sharing of a public key (e.g., asymmetric-key encryption [10, 21, 28]) and others require the sharing of a shared secret (e.g., attribute-based encryption [19, 43, 46, 52]) which is later used for key generation. We make no distinctions in this paper as to the kind of preliminary information that must be transmitted between publishers and subscribers, only note that every proposed solution requires the sharing of secrets. All of these techniques assume the presence of a completely and unilaterally trusted key management service that exists independently of the untrusted brokers to share secrets among communicating publishers and subscribers. Although this strong assumption of trust is itself a problem, there are additional requirements of such a service which themselves present problems. Such a service must not be compromised, as it would defeat the security of the entire system. Additionally, the service must always be available to accommodate newly joining publishers, subscribers and brokers. The service must have a mechanism in place to know when a publisher wishes to make alterations to the set of subscribers that have access to its events, and what these alterations are. Additionally, a leaving client might require key changes from the encryption scheme to ensure forward secrecy. To get around this, some solutions change keys periodically at every epoch and send them to the appropriate publishers and subscribers [30, 45]. If the epoch is too large then the system will be unresponsive. If the epoch is too small, then key changes will occur frequently, which can be expensive depending on the implementation details of the trusted key management service.

In general, we seek to solve the problem of efficiently sharing a secret among communicating clients when the underlying pub/sub infrastructure is not trusted and no external trusted service exists to share secrets. To do this, we introduce *HyShare*, a hybrid broker network that leverages the *Iterative Secret Share Propagation Scheme* (*ISSPS*) [51] and *Intel's Secure Guard Extensions* (*SGX*) [24].

To the best of our knowledge, ISSPS is the only scheme in the literature to share secrets between publishers and subscribers under the presence of a number of arbitrarily behaving brokers (Byzantine brokers) and requires no additional resources to those already provided by the classic pub/sub paradigm. Since we enhance ISSPS, our system shares this strength. Unfortunately, ISSPS is inefficient, with the number of messages required to share a secret scaling exponentially with the path length between the publishers and subscribers. This is a problem for ISSPS even when the path length is low. We show in this paper that even with a path length of two, hundreds of messages are needed to share a secret even on relatively simple topologies. At a path length of three, the number of messages needed to share a secret often exceeds a thousand. This is a problem as the literature often considers systems with path lengths of two or more [32, 36, 37]. HyShare mitigates this weakness by enhancing ISSPS with SGX.

SGX is a set of instructions and memory access changes added to the Intel architecture that allow an application to instantiate a secure container (*enclave*) [24]. An enclave is a protected area of execution in memory which provides confidentiality and integrity even in the presence of privileged malware. Attempted access to the enclave memory area from software not resident in the enclave are prevented even from privileged software such as virtual machine monitors, BIOS or operating systems. Unfortunately, solely using SGX to share a secret would restrict users to using Intel's architecture and hardware. This inflexibility carries the risk of a vulnerability targeting specific hardware compromising the entire network. In addition, since SGX is a fairly recent development and is proprietary technology from Intel, the hardware for it is expensive and has not yet seen adoption by cloud service providers.

To address these problems, we propose using a hybrid broker network using both ISSPS and enclaves. The hybrid network is composed of two types of brokers: *ISSPS brokers* and *SGX-enabled brokers*. ISSPS brokers run a modified ISSPS scheme. SGX-enabled brokers reduce the number of messages generated by ISSPS brokers. From our experiments, we observe that even a single SGX-enabled broker is sufficient to drastically reduce the overall number of messages needed to share a secret. With a sufficient number of SGX-enabled brokers in the network, the number of messages needed to share a secret grows linearly rather than exponentially with the path length between communicating clients. In general, our contributions are four-fold:

1. We analyze, verify and strengthen the original claims made of ISSPS.
2. We enhance ISSPS with SGX enclaves, making the number of messages needed to share a secret scale linearly rather than exponentially with path length.
3. We implement both ISSPS and our proposed enhancement using PADRES, a content-based pub/sub system.
4. We thoroughly evaluate our implementation using real, publicly available stock quote dissemination data.

In Section 2, we present our system model which presents pub/sub, SGX and the assumptions we make about the threat model under consideration. In Section 3, we introduce Shamir's secret sharing scheme and ISSPS. In Section 4, we discuss the state-of-the-art solutions in greater detail. Section 5 goes into further detail of ISSPS and its collusion tolerance properties. Section 6 presents HyShare in detail. Section 7 presents the evaluation of our schemes and Section 8 concludes.

2 SYSTEM MODEL

In this section, we introduce our system model for pub/sub and SGX. We describe the specific realization of a trusted execution environment, namely SGX, used in this paper. In addition, we detail any assumptions made about the pub/sub system, SGX and enclaves.

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 flooded 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 in subscriptions to identify a particular event stream. In content-based pub/sub, events contain attributes. Subscribers can further filter the events they receive by specifying the subset of attributes they are interested in within subscriptions.

We assume that any subscriber that can subscribe to events is authorized to receive them. Likewise, any publisher that can advertise an event is authorized to publish said event. Issues of access control are considered orthogonal to our work. 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 commonly considers the honest-but-curious threat model where all brokers do not deviate from the system specifications and protocols [31]. We consider a stronger threat model composed of 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 and 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. [53] and Srivatsa et al. [44]. 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. [34] have implemented secure content-based routing using SGX on purely encrypted attributes.

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 tursted hardware in the remote computer. The trusted hardware estalishes 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 is being performed on it [11].

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 [11]. 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 the paper. Similarly, we do not in general 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 [11]. 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*) [18].

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*). Although we do not use SSPS directly in our work, it is necessary for understanding ISSPS.

3.1 Shamir's Secret Sharing Scheme

The (k, n) secret sharing scheme introduced by Shamir [38] 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 as likely as the other 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 $q(x) = a_0 + a_1x + \ldots + a_{k-1}x^{k-1}$.
(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:

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

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 paper, 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 $[D_0, D_1, \ldots, D_i, \ldots, D_n]$ 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 (introduced in Sections 3.2 and 3.3) depending on which one is being used in context, and despite the fact that the latter requires an additional number of fragments.

3.2 Secret Share Propagation Scheme

The *Secret Share Propagation Scheme* (*SSPS*) [51] 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 receive at least k of these fragments and 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. 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.

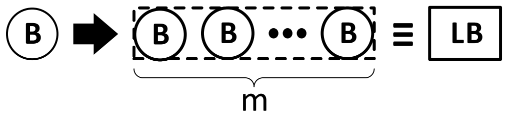


Figure 1: Transforming a single physical broker into a logical broker composed of m logically coupled physical brokers.

Any starting topology can be transformed into one with an arbitrary number of 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, $m \leq n$. 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 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. Figure 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.

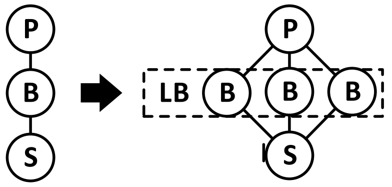


Figure 2: A simple topology transformation example for generating 3 parallel paths from a single path.

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

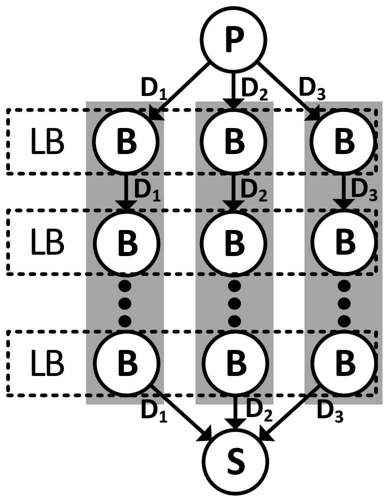


Figure 3: The propagation of fragments in SSPS for $n = m = 3$, with the parallel paths highlighted in gray.

single colluding broker in each of the k distinct parallel paths (i.e., k colluding brokers) is enough to compromise D.

3.3 Iterative Secret Share Propagation Scheme

The *Iterative Secret Share Propagation Scheme* (*ISSPS*) [51] 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 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 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 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

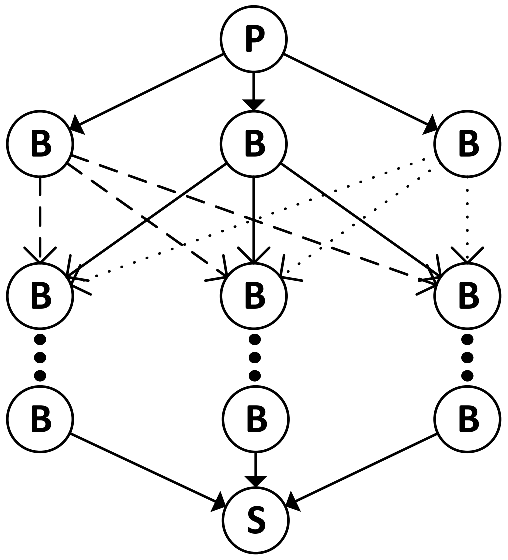


Figure 4: The propagation of fragments generated in ISSPS with $n = m = 3$.

requires n^h messages in the last hop alone of the transmission of a secret.

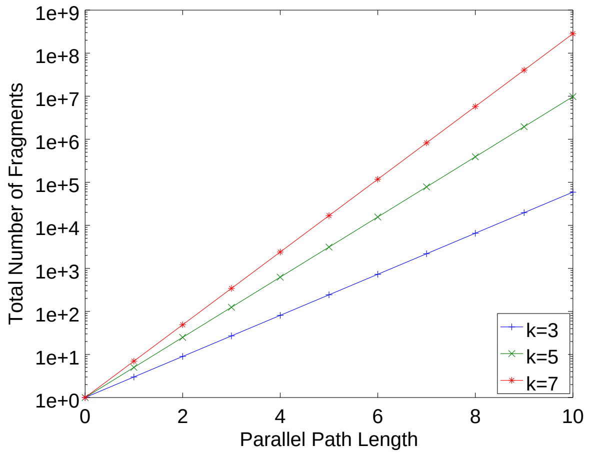


Figure 5: Number of fragments received by a subscriber on a log scale for the sharing of a single secret as the path length increases.

4 RELATED WORK

The issues of confidentiality and privacy in pub/sub are clearly identified by Wang et al. [47]. Unfortunately, the standard encryption and cryptographic techniques used to guarantee confidentiality in other contexts is not directly applicable to pub/sub which has the seemingly paradoxical task of routing events based on their confidential contents. This is especially challenging in content-based pub/sub, where brokers must match events based on confidential event attributes and subscription filters.

There has been a growing amount of work in the literature to achieve confidentiality in pub/sub through the use of various cryptographic techniques [31]. Some of the mechanisms used include symmetric-key encryption [8, 13, 23], homomorphic cryptosystems [21, 28], asymmetric scalar-product preserving encryption [10], multiple layer commutative encryption [40], oblivious transfer [12] and attribute-based encryption [19, 43, 46, 52]. A common thread to every solution we've observed in the literature is the need for a 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. To the best of our knowledge, Yoon and Kim [51] (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 commonly used honest-but-curios threat model. Unfortunately, the number of messages needed to share a secret grows exponentially with the number of broker hops between clients. This is the primary problem that we seek to solve in this paper. 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 [34]. 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. [25], 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 [6, 14, 26].

5 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 [51] 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 5.1. *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 5.2. *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 and 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.

6 HyShare COMPONENTS

In this section, we present the various aspects which compose HyShare. First, we introduce *ISSPS brokers*, a type of broker that runs a modified version of ISSPS. Next, we introduce *SGX-enabled brokers*, which 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 section 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. With HyShare, we propose using ISSPS brokers to share secrets and SGX-enabled brokers to reduce the overall number of messages used to share a secret.

6.1 ISSPS Brokers

ISSPS brokers run a modified version of ISSPS. Figure 6 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.

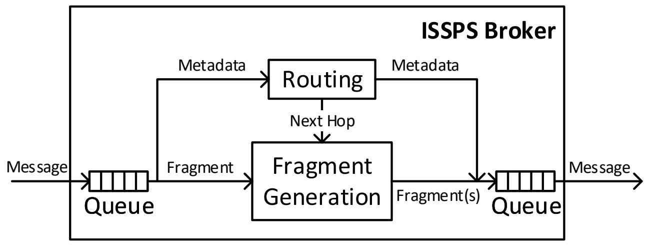


Figure 6: High level ISSPS Broker Overview.

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 attestation, as presented in Section 6.2. 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.

6.2 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 [11]. 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 twofold. 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 7 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 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.

6.3 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 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 $4k$,

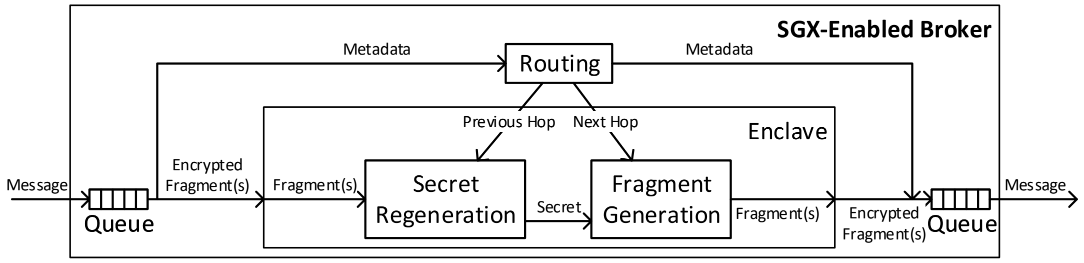


Figure 7: High level SGX-enabled broker overview.

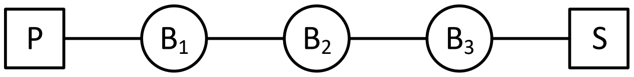


Figure 8: A simple topology pub/sub topology composed of a publisher, a subscriber and three brokers.

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, Cheung and Jacobsen [9] can be used with HyShare with some modifications. In Cheung and Jacobsen [9], the authors 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.

7 EVALUATION

We have implemented our solution in PADRES [20], a content-based pub/sub system. PADRES supports the direct connection between clients and multiple brokers. Additionally, PADRES can be set to support 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.

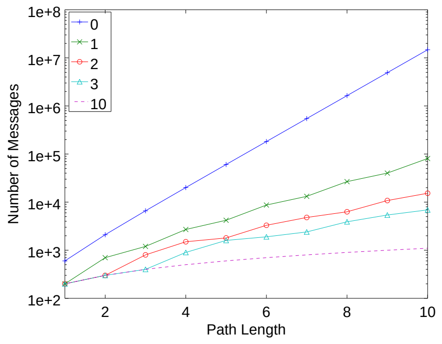
7.1 Experimental Setup

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.

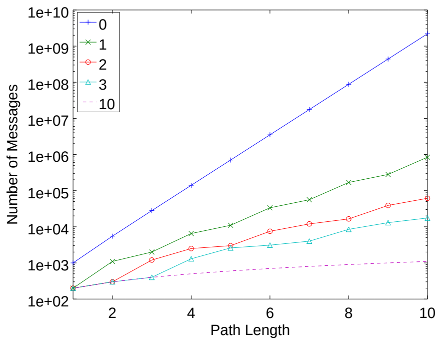
We ran HyShare on a cluster of 23 nodes. Each node has a 1.86 GHz Intel Xeon 5120 processor with 10 GB of memory. The nodes are physically interconnected using a switched 1 Gbps Ethernet network. HyShare is implemented using mostly Java. The only exception is the code executing inside enclaves which is implemented using C++ as Java is not supported by SGX. Enclaves were run using the SGX simulation mode provided by the SGX SDK. We used the default launch configuration for PADRES, which sets the maximum memory allocation 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.

7.2 Line Topology

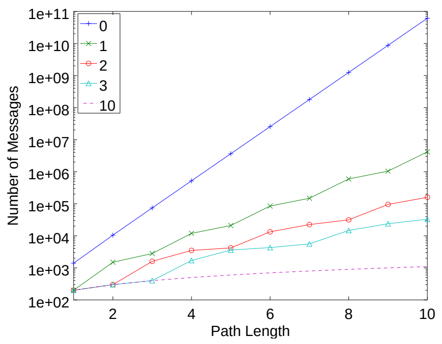
We begin our evaluation by comparing the total number of messages required to share 400 secrets in HyShare for various numbers



(a) $k = 3$

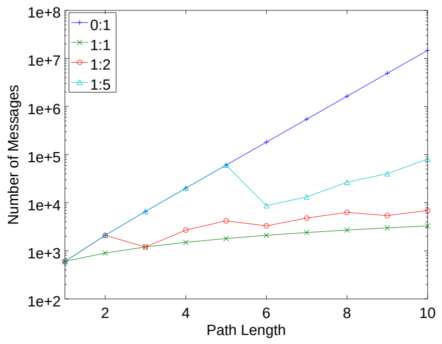


(b) $k = 5$

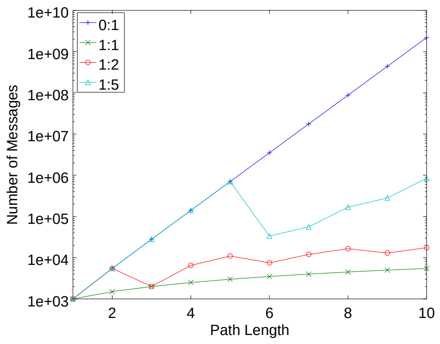


(c) $k = 7$

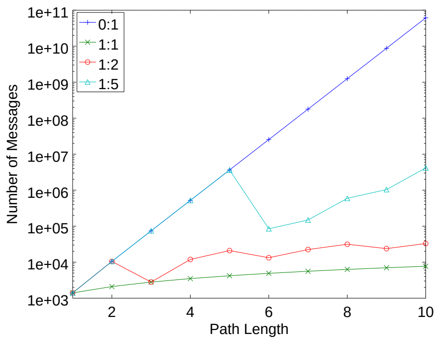
Figure 9: Effect on the number of messages needed to share secrets with varying path length and constant number of SGX-enabled brokers.



(a) $k = 3$



(b) $k = 5$



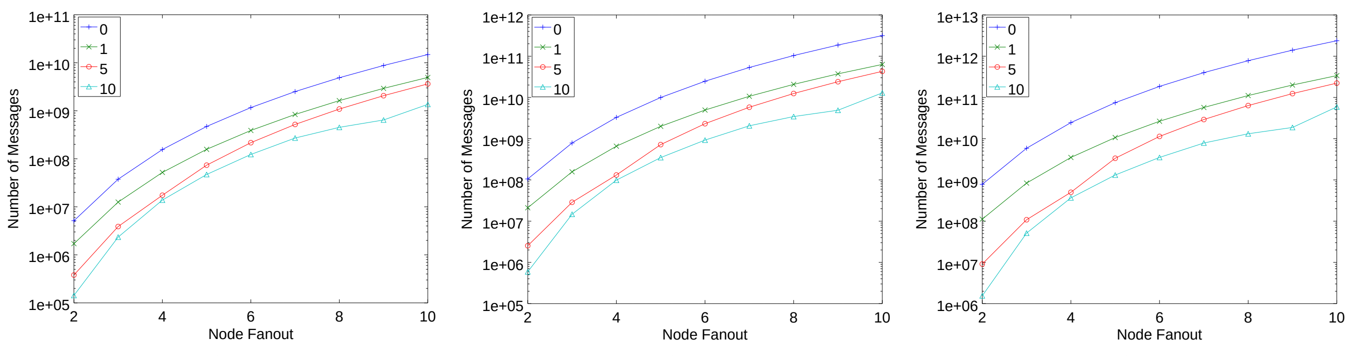
(c) $k = 7$

Figure 10: 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 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.

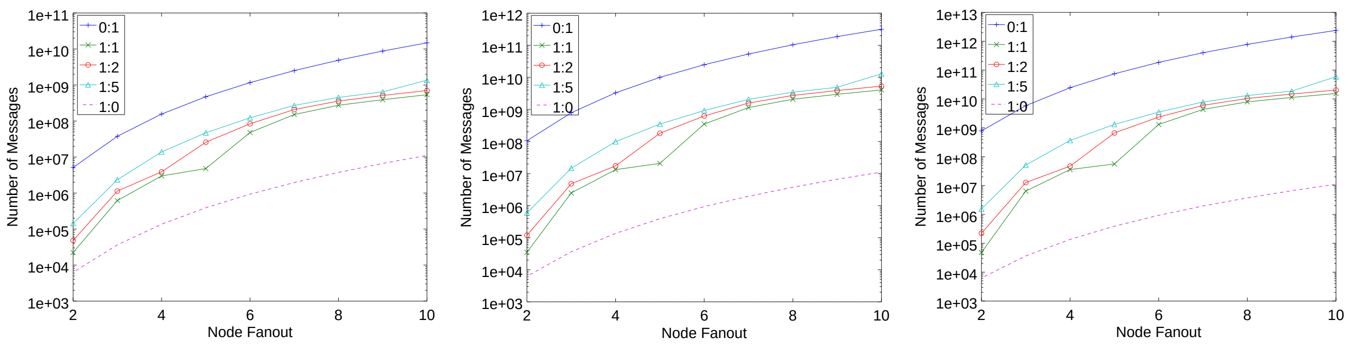
Hybrid broker networks are designed to reduce (or even halt in its entirety) the exponential growth of messages that is observed at each broker hop in ISSPS. In Figure 9 and Figure 10, we show the effects on the total number of messages needed to share secrets as the path length increases with varying number of SGX-enabled brokers. The three graphs in these figures show the same results with different input parameter k. More specifically, Figure 9 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 10, we show the effects of maintaining a ratio of SGX-enabled brokers to 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,



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

Figure 11: Effect on the number of messages needed to share secrets with varying node fanout and constant number of SGX-enabled brokers.



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

Figure 12: Effect on the number of messages needed to share secrets with varying node fanout and maximum ratio of SGX-enabled brokers to ISSPS brokers.

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.

7.3 Stock Quote Dissemination

A classic motivating example for the need of secret sharing in pub/sub is that of a stock quote notification system [31, 47]. 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 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 11 and Figure 12. As before, the subfigures **??**, **??** and **??** represent the experiment executed using various values for the input parameter k. The legend of Figure 11 shows the number of SGX-enabled brokers within the topology, and its effects as the node fanout is increased. In Figure 12, 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 13, 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

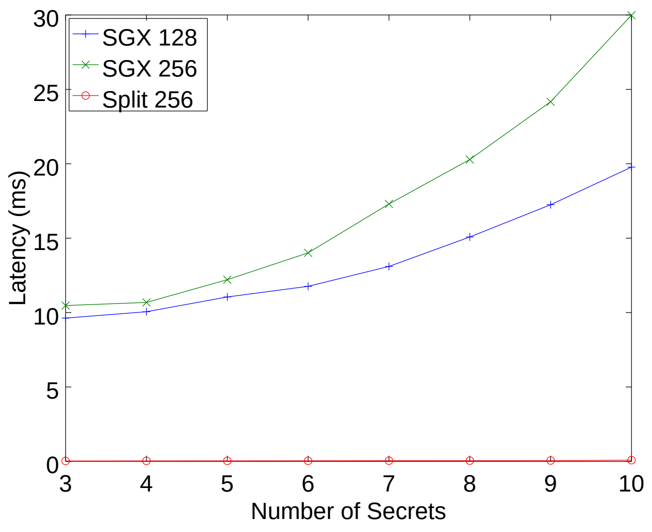


Figure 13: Latencies comparing SGX-enabled broker operations to ISSPS broker operations for various key sizes.

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 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.

8 CONCLUSIONS

Pub/Sub has seen wide implementation and adoption. Also ubiquitous are the various networked systems where privacy is important. Recent developments in SGX and trusted execution environments call for the revisiting of the problem of confidentiality in pub/sub. The focus of our work is on secret sharing, which can be used either to hide sensitive data from unauthorized entities or as a precursor to more complex encryption techniques.

In this paper, we introduced HyShare, a solution that uses hybrid broker networks for secret sharing. The broker networks are composed of two types of brokers: ISSPS brokers and SGX-enabled brokers. 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 in Section 7, we believe that we have only scratched the surface of what is possible. We believe that these considerations will become important if and when SGX technology becomes more affordable and adopted by cloud service providers. Additionally, we want to further expand on the problem of dynamic broker placement within the broker network based on observed traffic patterns.

ACKNOWLEDGMENTS

We thank Bogdan Scaunasu for his help with the experiments. This work was supported in part by ORF and NSERC.

REFERENCES

[1] Allegro. 2015. Hermes: Reliable and easy to use message broker built on top of Kafka. (2015). Retrieved January 31, 2018 from http://hermes.allegro.tech/
[2] Amazon Web Services. 2018. What is Pub/Sub Messaging? (2018). Retrieved January 31, 2018 from https://aws.amazon.com/pub-sub-messaging/
[3] Apache Software Foundation. 2018. Apache Pulsar. (2018). Retrieved January 31, 2018 from https://pulsar.apache.org/
[4] Jean Bacon, David M. Eyers, and Jatinder Singh. 2010. Securing Event-Based Systems. In *Principles and Applications of Distributed Event-Based Systems*. IGI Global, 119–139.
[5] Jean Bacon, David M. Eyers, Jatinder Singh, and Peter R. Pietzuch. 2008. Access Control in Publish/Subscribe Systems. In *Proceedings of the Second International Conference on Distributed Event-based Systems (DEBS '08)*. ACM, 23–34.
[6] Luciano Barreto, Leomar Scheunemann, Joni Fraga, and Frank Siqueira. 2017. Secure Storage of User Credentials and Attributes in Federation of Clouds. In *Proceedings of the Symposium on Applied Computing (SAC '17)*. ACM, New York, NY, USA, 364–369.
[7] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. 2001. Design and Evaluation of a Wide-area Event Notification Service. *ACM Trans. Comput. Syst.* 19, 3 (Aug. 2001), 332–383.
[8] Weifeng Chen, Jianchun Jiang, and N. Skocik. 2010. On the privacy protection in publish/subscribe systems. In *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on.* 597–601.
[9] Alex King Yeung Cheung and Hans-Arno Jacobsen. 2010. Load Balancing Content-Based Publish/Subscribe Systems. *ACM Trans. Comput. Syst.* 28, 4, Article 9 (Dec. 2010), 55 pages.
[10] Sunoh Choi, Gabriel Ghinita, and Elisa Bertino. 2010. *A Privacy-Enhancing Content-Based Publish/Subscribe System Using Scalar Product Preserving Transformations.* Springer Berlin Heidelberg, 368–384.
[11] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.
[12] Giovanni Crescenzo, Brian Coan, John Schultz, Simon Tsang, and Rebecca N. Wright. 2014. 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*. Springer-Verlag New York, Inc., 114–132.
[13] Giovanni Di Crescenzo, Jim Burns, Brian Coan, John Schultz, Jonathan Stanton, Simon Tsang, and Rebecca N. Wright. 2013. *Efficient and Private Three-Party Publish/Subscribe.* Springer Berlin Heidelberg, 278–292.
[14] Shlomi Dolev, Juan A. Garay, Niv Gilboa, Vladimir Kolesnikov, and Yelena Yuditsky. 2012. Efficient Private Distributed Computation on Unbounded Input Streams. *CoRR* abs/1208.4909 (2012). arXiv:1208.4909
[15] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)* 35, 2 (2003), 114–131.
[16] Google Cloud Platform. 2017. What is Google Cloud Pub/Sub? (Sept. 2017). Retrieved January 31, 2018 from https://cloud.google.com/pubsub/docs/overview
[17] Hadoop Wiki. 2011. HedWig. (Jan. 2011). Retrieved January 31, 2018 from https://wiki.apache.org/hadoop/HedWig
[18] Intel. 2016. Intel Software Guard Extensions Remote Attestation End-to-End Example. (July 2016). Retrieved April 24, 2018 from https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-example
[19] Mihaela Ion, Giovanni Russello, and Bruno Crispo. 2012. Design and implementation of a confidentiality and access control solution for publish/subscribe systems. *Computer Networks* 56, 7 (2012), 2014 – 2037.
[20] Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh. 2010. *The PADRES Publish/Subscribe System.* IGI Global, 164–205.
[21] Marek Klonowski and Bartek Rózanski. 2005. Privacy protection for p2p publish-subscribe networks. (2005).
[22] Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsen. 2010. A Distributed Service Oriented Architecture for Business Process Execution. *ACM Transactions on the Web* 4, 1 (January 2010), 2:1–2:33.
[23] Jun Li, Chenghuai Lu, and Weidong Shi. 2004. An Efficient Scheme for Preserving Confidentiality in Content-Based Publish-Subscribe Systems. (2004).

[24] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. 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)*. ACM, Article 10, 1 pages.

[25] Kazuhiro Minami, Adam J. Lee, Marianne Winslett, and Nikita Borisov. 2008. Secure Aggregation in a Publish-subscribe System. In *Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society (WPES '08)*. ACM, 95–104.

[26] Toshiyuki Miyamoto and Sadatoshi Kumagai. 2007. 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)*. Springer-Verlag, Berlin, Heidelberg, 371–382.

[27] MQTT. 2014. MQTT. (Nov. 2014). Retrieved January 31, 2018 from http://mqtt.org/

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

[29] Christoph P. Neumann, Florian Rampp, Richard Lenz, and Michael Daum. 2009. A Mediated Publish-subscribe System for Inter-institutional Process Support in Healthcare. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS '09)*. ACM, Article 14, 4 pages.

[30] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. 2015. Efficient Key Updates Through Subscription Re-encryption for Privacy-Preserving Publish/Subscribe. In *Proceedings of the 16th Annual Middleware Conference (Middleware '15)*. ACM, 25–36.

[31] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. 2016. Confidentiality-Preserving Publish/Subscribe: A Survey. *ACM Comput. Surv.* 49, 2, Article 27 (June 2016), 43 pages.

[32] Melih Onus and Andréa W. Richa. 2011. Minimum Maximum-degree Publish-subscribe Overlay Network Design. *IEEE/ACM Trans. Netw.* 19, 5 (Oct. 2011), 1331–1343.

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

[34] Rafael Pires, Marcelo Pasin, Pascal Felber, and Christof Fetzer. 2016. Secure Content-Based Routing Using Intel Software Guard Extensions. In *Proceedings of the 17th International Middleware Conference (Middleware '16)*. ACM, New York, NY, USA, Article 10, 10 pages.

[35] RedisLabs. 2018. Redis. (2018). Retrieved January 31, 2018 from https://redis.io/

[36] J. Reumann. 2009. Pub/Sub at Google. (2009).

[37] Vinay Setty, Gunnar Kreitz, Roman Vitenberg, Maarten van Steen, Guido Urdaneta, and Staffan Gimåker. 2013. The Hidden Pub/Sub of Spotify: (Industry Article). In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems (DEBS '13)*. ACM, 231–240.

[38] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613.

[39] 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. 2015. 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)*. USENIX Association, 351–366.

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

[41] Jatinder Singh and Jean Bacon. 2008. Event-based Data Control in Healthcare. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion (Companion '08)*. ACM, 84–86.

[42] Jatinder Singh and Jean Bacon. 2009. Event-Based Data Dissemination Control in Healthcare. In *Electronic Healthcare*, Dasun Weerasinghe (Ed.). Springer Berlin Heidelberg, 167–174.

[43] Mudhakar Srivatsa, Ling Liu, and Arun Iyengar. 2011. EventGuard: A System Architecture for Securing Publish-Subscribe Networks. *ACM Trans. Comput. Syst.* 29, 4, Article 10 (dec 2011), 40 pages.

[44] Mudhakar Srivatsa, Ling Liu, and Arun Iyengar. 2011. EventGuard: A System Architecture for Securing Publish-Subscribe Networks. *ACM Trans. Comput. Syst.* 29, 4, Article 10 (dec 2011), 40 pages.

[45] Muhammad Adnan Tariq, Boris Koldehofe, Ala' Altaweel, and Kurt Rothermel. 2010. 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)*. ACM, 38–49.

[46] M. A. Tariq, B. Koldehofe, and K. Rothermel. 2014. Securing Broker-Less Publish/Subscribe Systems Using Identity-Based Encryption. *IEEE Transactions on Parallel and Distributed Systems* 25, 2 (Feb 2014), 518–528.

[47] C. Wang, A. Carzaniga, D. Evans, and A. Wolf. 2002. Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9 - Volume 9 (HICSS '02)*. IEEE Computer Society, 3940–3947.

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

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

[50] Zhengdao Xu and Arno Jacobsen. 2007. Adaptive Location Constraint Processing. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD '07)*. ACM, New York, NY, USA, 581–592.

[51] Young Yoon and Beom Heyn Kim. 2016. Secret Forwarding of Events over Distributed Publish/Subscribe Overlay Network. *PLOS ONE* 11, 7 (07 2016), 1–23.

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

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