Sharing a Secret Using Pub/Sub

Anonymous Author(s)

ABSTRACT

Publish/subscribe (pub/sub) is a communication paradigm that allows loosely-coupled clients to communicate in an asynchronous fashion. This paradigm supports the flexible development of heterogeneous, large-scale and event-driven systems. A basic security concern in pub/sub is that of guaranteeing the confidentiality of the data being communicated. Though this problem can be addressed through the use of various encryption techniques, they still require the establishment of an initial shared secret between the communicating entities. To do this, existing solutions rely on a out-of-band service to disseminate secrets or keys. The problem with this approach is that it requires additional resources to build this external infrastructure that works independently of the existing pub/sub resources. Another problem is that it requires that clients (publishers and/or subscribers) trust this service. We propose two novel schemes to disseminate shared secrets that utilize only the resources provided by the pub/sub network. These solutions tolerate serveral colluding brokers without the need for an external trusted service. To the best of our knowledge, no such solution exists. Furthermore, we prove the privacy claims of our solution and show how it is resilient to a large ratio of colluding brokers to total brokers. We evaluate our solutions by comparing our schemes and measuring the amount of overhead added by their implementation.

KEYWORDS

publish/subscribe, key management, confidentiality

ACM Reference format:

Anonymous Author(s). 2017. Sharing a Secret Using Pub/Sub. In *Proceedings of Distributed and Event-Based Systems, Barcelona, Spain, June 19–23 (DEBS),* 5 pages.

DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Publish/subscribe (pub/sub) is a communication paradigm that allows loosely-coupled clients to communicate in an asynchronous fashion. This paradigm supports the flexible development of heterogeneous, large-scale and event-driven systems. Pub/sub is prevalent in many application domains such as social networking, distributed business processes, cyber-physical systems and even internet architectures. Many pub/sub applications are sensitive to privacy violations. For example, social networks are built upon pub/sub paradigm. A common concern among users is that the social network service may accidentally deliver personal contents to users who are not entitled to them. Another example would be when transmitting health records through pub/sub. Regulations often dictate that such data is to be encrypted before any communication.

Confidentiality is a basic security issue identified for pub/sub systems [6]. That is, clients (publishers and/or subscribers) may wish to keep sensitive information secret from the underlying pub/sub infrastructure. The infrastructure itself be partially or completely untrusted to read data. To guarantee data confidentiality, existing solutions leverage various encryption techniques [4], most of which require a shared secret (or a key) be established among communicating clients.

The objective of this paper is to address the problem of sharing a secret among communicating clients when the underlying pub/sub infrastructure is untrusted. To do this, Shamir's secret sharing scheme [5] is adapted for use with pub/sub. We present two implementations in this paper, one of which builds on the other. They allow for the dissemination of a secret without the need for dedicated secret sharing resources and without sacrificing any of the desirable decoupling properties of pub/sub. Furthermore, the approach is resilient to a number of colluding brokers within the pub/sub infrastructure. The solutions are implemented in a content-based pub/sub system [1] though it is easily adaptable for any topic-based or broker-less pub/sub that supports cyclic topologies.

In the literature, solutions to this problem often take the same general approach: a trusted out-of-band service is used to store and disseminate secrets (and/or keys) as needed [4]. After some interchange of messages to establish some initial parameters, various encryption techniques are then used to ensure the privacy of events. The primary difference between the existing work and the solutions we present is that our solutions utilize the resources provided by the pub/sub infrastructure without the need for a trusted service with dedicated resources.

Our proposed schemes require that the underlying pub/sub infrastructure provide multiple redundant paths from each publisher to each subscriber. We address this by providing a method for constructing a topology with the required properties from any initial topology. Another drawback to our schemes is that they increase the number of messages that must be routed by the brokers. One of the solutions in particular can significantly increase the amount of processing required of each broker in order to successfully transmit a secret. Due these costs, we suggest using our schemes to establish a shared secret and then using any of the existing encryption techniques.

The main contributions of this paper are as follows:

- Design and implementation of two secret sharing schemes that use only the resources available in the classic pub/sub paradigm. To the best of our knowledge, no such solution exists.
- Proof of the privacy guarantees for each of our schemes. The schemes are resilient to a number of colluding brokers

[1] System name blinded for peer review.

DEBS, Barcelona, Spain

DOI: 10.1145/nnnnnnn.nnnnnnn

expressed as a ratio of colluding brokers to total brokers. This ratio is determined directly by one of the input parameters of our scheme.

In Section 2 we briefly introduce pub/sub and Shamir's secret sharing scheme. In Section 3 we discuss the state-of-the-art solutions in greater detail. Section 4 presents our solution. Section 5 presents the evaluation of our schemes and Section 6 concludes.

2 BACKGROUND

Some encryption schemes require that communicating entities establish a secret key (e.g., symmetric-key encryption), others require the sharing of a public key (e.g., asymmetric-key encryption) and others only require the sharing of a shared secret (e.g., attribute-based encryption) which is later used for key generation. Since this paper is agnostic of any single encryption scheme, we use the terms key and secret are used interchangeably.

2.1 Publish/Subscribe

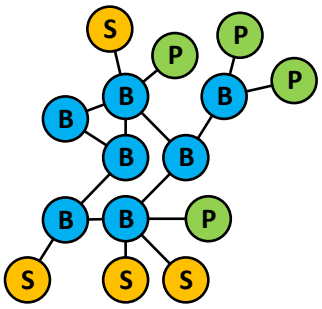

Figure 1: A sample instance of a pub/sub network. P denotes a publisher, B denotes a broker and S denotes a subscriber.

Publish/subscribe is a communication paradigm where publishers (data sources) and subscribers (data sinks) communicate through events (messages). Brokers route all events from publishers to a subset of subscribers. Broker federation refers to the set of all brokers. A client refers to an entity that is a publisher and/or a subscriber. Figure 1 shows an example of a pub/sub network.

The pub/sub paradigm is appealing due to loose-decoupling between publishers and subscribers. Events are usually routed through the broker network. In broker-less pub/sub, the routing is done entirely by clients.

Publishers must first send an advertisement before they can publish events to the broker federation. The advertisement defines an event space for events to be published in the future. Subscribers can express their interest in a subset of all events through a subscription. Brokers use the information from both advertisements and subscriptions to route events through the broker federation. In topic-based pub/sub, subscribers may only subscribe to event topics. In content-based pub/sub, subscriptions may further refine their interest based on the attributes of the event.

2.2 Shamir's Secret Sharing Scheme

Shamir's secret sharing scheme was originally introduced by Shamir [5] and referred to as the (k, n) threshold scheme. In this paper, we refer to it as the (k, n) secret sharing scheme. The scheme splits a secret value into n fragments, k of which are required to recover the original secret value. It is based on polynomial interpolation.

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 all coefficients a_i to random values.
(3) Set a_0 to the 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 $q(0)$. Finding $q(x)$ itself is a matter of solving for k unknowns with k equations.

2.3 Threat Model

Our threat model follows the commonly used honest-but-curious model [4]. All entities (publishers, subscribers and brokers) do not deviate from the system specifications and protocols. Entities are curious to find out any additional information by analyzing any data available to them. Furthermore, we consider the threat posed by colluding brokers to our schemes.

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. We do not consider the issue of subscribers leaking secrets as that would be akin to solving the digital copyright problem.

3 RELATED WORK

As mentioned previously, the issues of confidentiality and privacy in pub/sub are explored thoroughly by Wang et al. [6]. To the best of our knowledge, none of the literature has solved the problem of key dissemination by strictly utilizing only the resources offered by pub/sub network containing honest-but-curious entities rather than relying on a trusted out-of-band service.

One related line of work for protecting event privacy is using cryptographic techniques so that events are encrypted before being sent out by the publisher. In this way, even if brokers or unintended subscribers get the message, they cannot read the content without the decryption key. The problem with this method is that the encryption hinders the of brokers to do content-based routing. Although we highlight a few of these approaches, a more comprehensive list can be found in the survey written by Onica et al. [4].

In **?**], the authors' protocol derives a set of attributes from the content of the publication and run the matching algorithm over this set of attributes instead of the encrypted payload. The work uses homomorphic encryption techniques to execute matching operations over the encrypted data without involving decryption or revealing private event data to the brokers. This approach uses a trusted key management service to generate and disseminate secrets to publishers and subscribers.

In Onica et al. [3], the authors use a trusted coordinator and various dedicated hosts for managing key dissemination. A ZooKeeper coordination service is used to simplify the coordination and dependability aspects of the application. The hosts communicate directly with the clients and brokers to disseminate keys. As before, the trusted coordinators and various hosts act as a trusted key dissemination service.

In **?**], the authors identify the problem of sacrificing the desirable decoupling properties of pub/sub in order to have publisher and subscribers directly share a secret. The authors leverage attribute-based encryption to perform matching on encrypted event attributes, but still rely on a trusted authority to disseminate the required secrets.

4 SECRET SHARING SCHEMES

In this section we introduce two solutions that make use of Shamir's secret sharing scheme in pub/sub. To do so, we first show how to generate multiple paths from publisher to subscriber given any broker federation - a prerequisite to both solutions. Then we discuss each solution. We refer to the first solution as the Line-based Scheme (LS) and the second as the Tree-based Scheme (TS). TS builds on LS. For simplicity and without loss of generality, we consider single publisher and subscriber pub/sub networks.

4.1 Broker Replication

We replicate brokers in order to generate multiple paths between all communicating clients. These paths must meet one requirement: the intersect between the set of brokers in each path must be empty. We denote paths that meet this property as *parallel paths*. Both TS and LS require that k parallel paths exist in the broker topology

To convert any pub/sub topology into one with at least k parallel paths, we first replace each broker in the topology with a Virtual Broker (VB). Each VB is composed of k physical brokers. In LS, each physical broker within the VB connects to a different physical broker within the neighboring VB. In TS, each physical broker has the same neighbors as the VB (i.e. if a given VB is a neighbor to another VB then so are all its physical brokers).

Figure 2 shows an example of this broker replication for $k = 3$. After replication, we immediately generate 3 parallel paths from one client to another. In Figure 3, each parallel path is denoted by a gray region.

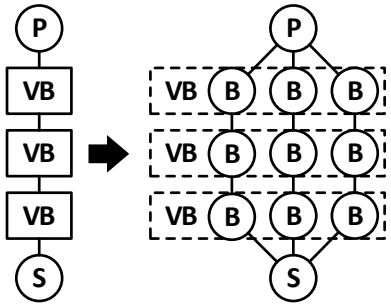

Figure 2: A broker replication example for generating 3 parallel paths from a single broker path.

The advantages of this form of broker replication lies in its simplicity. Any existing topology can be immediately replicated in order to create parallel paths. This comes at the cost of increasing the size of the broker network. An alternative to this would be to try and detect the existing parallel paths within the topology, but this does not guarantee that k parallel paths will be found. The broker redundancy introduced by this replication could be further leveraged to guarantee the delivery of any event, even in the presence of faulty (or byzantine) brokers. We consider this extension to be beyond the scope of this paper.

4.2 Line-based Scheme

As mentioned in Section 2, the (k, n) secret sharing scheme produces n fragments from a secret D, k of which are needed to regenerate the original secret. For the rest of the paper, we set $k = n$. Setting differing values for k and n is possible but would unnecessarily increase message redundancy.

Once the parallel paths are established, the LS implementation is trivial to implement. The publisher must generate the fragments from D and send each through a different parallel path. The subscriber can regenerate D once it has the k fragments. Figure 3 shows an example of how fragments would flow through the broker federation for $k = 3$.

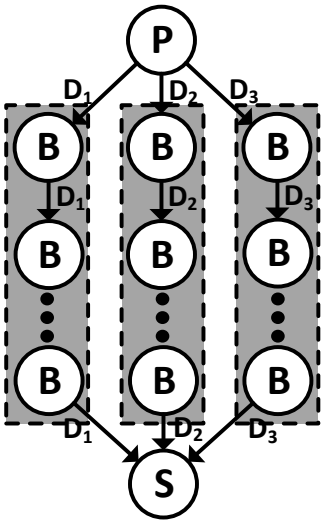

Figure 3: The propagation of fragments generated from a (3, 3) secret sharing scheme in LS.

4.3 Tree-based Scheme

TS builds on top of LS. As before, it requires broker replication in order to get k parallel paths. It also requires that the publisher generate k fragments from D and send each fragment to one of the neighboring replicated brokers. The difference between TS and LS is that the secret sharing scheme is applied at each broker hop on all fragments received. Each received fragment is treated as the secret, and the newly generated fragments are each sent to a different physical broker within the next hop VB. Figure 4 shows

an example of this for $k = 3$. Eventually, all fragments converge at the subscriber, which can then regenerate the original secret.

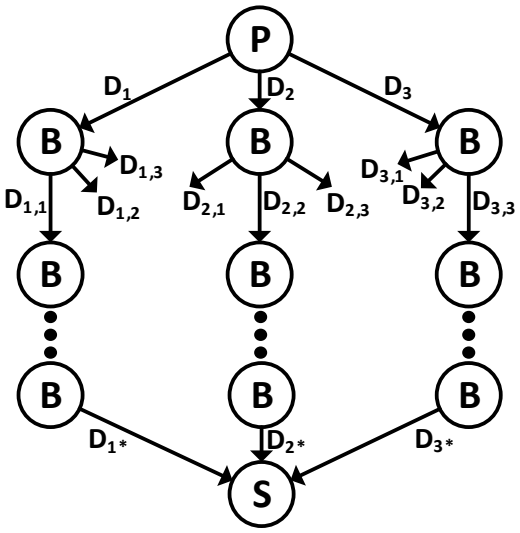

Figure 4: The propagation of fragments generated from a (3, 3) secret sharing scheme in TS.

As discussed in the following section, TS allows us to make stronger security claims than LS. Unfortunately, this comes at the cost of an increase in the number of messages routed by the broker federation and more processing required from each broker and subscriber. The number of fragments received at the subscriber is k^h where h is the number of brokers in a parallel path. This is precisely what is shown in Figure 5 for various values of k.

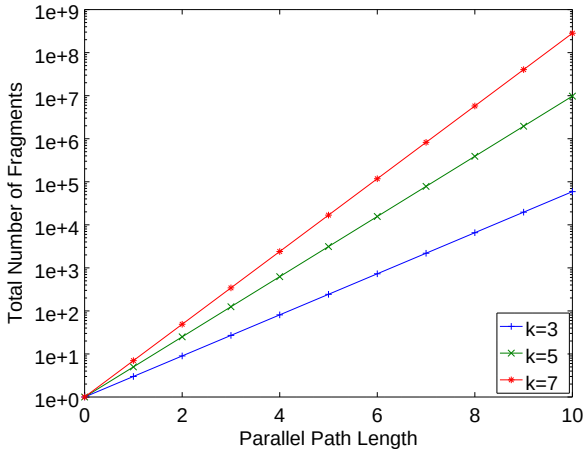

Figure 5: Number of fragments received by a subscriber as the path length increases.

4.4 Security Analysis

Theorem 4.1. *LS can tolerate up to $k-1$ colluding brokers in the worst case scenario without compromising the original secret value D.*

Proof. Assume by way of contradiction that any $k-1$ brokers collude to learn D. Since all brokers receive at most one fragment, the colluding brokers can only learn the values of at most $k-1$ fragments. By the assumption, they must have regenerated D from at most $k-1$ fragments. This contradicts the primary claim from the (k, n) secret sharing scheme - that at least k fragments are required to regenerate D. □

Corollary 4.2. *LS can tolerate collusion from any number of brokers belonging in up to $k-1$ parallel paths.*

Proof. Assume by way of contradiction that all brokers in $k-1$ parallel paths collude to learn D. From LS, all brokers within a single parallel path learn of at most 1 fragment. If all brokers within $k-1$ parallel paths collude, they learn the values of up to $k-1$ fragments. This means by the assumption that the colluding brokers were able to regenerate D from $k-1$ fragments, which is a contradiction. □

We can grasp these claims intuitively from the example shown in Figure 3. If all fragments shared with any two of three parallel paths are leaked to an adversary then the adversary would only learn the values of at most two fragments. Two fragments are insufficient to recover D in a $(3, 3)$ secret sharing scheme.

TS does not improve on the worst case scenario from LS. That being said, this scheme allows us to make a stronger security claim for the best case scenario that is independent of parallel paths.

Theorem 4.3. *TS can tolerate up to $k-1$ colluding brokers in the worst case scenario without compromising the original secret value D.*

Proof. Assume by way of contradiction that any $k-1$ brokers collude to learn D. From TS, each broker 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. □

Theorem 4.4. *If all virtual brokers contain at least one non-colluding broker then D is secure.*

Proof. If there exists one non-colluding broker at every virtual 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 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 TS, 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 brokers 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.

□

Let CB be the set of colluding brokers and B be the set of all brokers, then, by the previous theorems and corollaries, the following ratio of colluding brokers to total brokers holds for both LS and TS:

$$\frac{|CB|}{|B|} = \frac{k-1}{k}$$

That being said, LS has the added requirement that an entire parallel path be composed of non-colluding entities. This requirement is not necessary in TS. For TS, at least one of the brokers within the virtual broker must be non-colluding.

5 EVALUATION

We implement our solution in a PADRES [2], a content-based pub/sub system.

TODO: the follwing graphs:

- Splitting times vs k
- Reconstructing D (time) vs k
- Total time taken to share a secret vs k (also compared to a regular publication)

Each graph will include various sizes for D (the original secret, in bits)

6 CONCLUSION

REFERENCES

[1] 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.

[2] 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.

[3] 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, New York, NY, USA, 25–36. DOI: http://dx.doi.org/10.1145/2814576.2814805

[4] 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. DOI: http://dx.doi.org/10.1145/2940296

[5] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613. DOI: http://dx.doi.org/10.1145/359168.359176

[6] 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, Washington, DC, USA, 303–. http://dl.acm.org/citation.cfm?id=820747.821330