Efficient Key Updates through Subscription Re-encryption for Privacy-Preserving Publish/Subscribe

Emanuel Onica
Alexandru Ioan Cuza University of Iaşi, Romania*
eonica@info.uaic.ro

Pascal Felber, Hugues Mercier and Etienne Rivière
University of Neuchâtel, Switzerland
first.last@unine.ch

ABSTRACT

Content-based publish/subscribe (pub/sub) is an appealing information dissemination paradigm for distributed systems. Consumers of data subscribe to a pub/sub service, typically offered through a distributed broker overlay, and indicate their interests as constraints over the information content. Publishers generate the information flow, which the brokers filter and route to the interested subscribers. Protecting the information confidentiality, and in particular the interests of subscribers, is an important concern when brokers are located in untrusted domains such as public clouds. Encrypted matching techniques allow untrusted brokers to store encrypted subscriptions and match them against encrypted publications. Updates of encryption keys regularly happen in such contexts due to changes in trust relations. These key updates cause the invalidation of stored encrypted subscriptions and force subscribers to re-encrypt and re-submit them. This long and costly operation impacts the pub/sub service continuity and performance. In this paper, we propose a novel technique that allows updating encrypted subscriptions directly at the brokers while maintaining privacy. We present an implementation of the technique for the ASPE encrypted matching scheme and prove the security of our extension. We evaluate its practical effectiveness through a prototype implementation including a dependable key distribution protocol. Our experiments show the ability to handle key updates while preserving service continuity and performance.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; D.4.6 [**Security and Protection**]: Information flow controls

General Terms

Algorithms, Security, Performance

Keywords

publish/subscribe, confidentiality, key management

*Work done while author was with University of Neuchâtel

Middleware '15, December 07–11, 2015, Vancouver, BC, Canada

DOI: http://dx.doi.org/10.1145/2814576.2814805

1. INTRODUCTION

Content-based publish/subscribe (pub/sub) [13] is a distributed information dissemination model. It allows decoupled communication between sources of data (*publishers*) and its consumers (*subscribers*). Subscribers register their *subscriptions* to a pub/sub service, expressing their interest in data through a set of constraints on the data content itself. A high-performance pub/sub service is generally provided by a set of servers dedicated to that task. These servers can either be organized as a network of brokers [6, 19, 27] or as a set of processing operators organized as a stream processing application [2]. We will commonly refer to these dedicated machines as *brokers* in the rest of this paper. Publishers disseminate information by sending *publications* to any of the brokers. A publication is composed of a header including a set of attributes that characterize the published information, and an optional payload. Brokers filter the incoming publication flow by matching headers of publications against the constraints in the subscriptions they store. Matching publications are routed by the brokers and delivered to the interested subscribers.

A seminal pub/sub example is that of a stock exchange service [23]. Subscribers are companies or individuals interested in the fluctuation of different stocks, who submit appropriate subscriptions (e.g. $\{symbol =$"`GOOG`" and $value \leq 600\}$). Publishers are stock market operators sending publication flows of trade quotes for each stock. A typical publication could be $\{symbol =$ "`GOOG`", $value = 600.70$, $variation = -5.55$, $date = 2015.09.04\}$. Another application of interest for pub/sub is e-health systems [14, 17, 18]. Figure 1 illustrates the information flow in a pub/sub system.

Deploying a pub/sub service in a public cloud is an appealing option. It brings significant advantages in terms of cost-effectiveness due to on-demand provisioning and elasticity mechanisms [4]. It also allows providing a publicly reachable service with little upfront investment. Nevertheless, the confidentiality of subscriptions and publications is a major hurdle for the deployment of pub/sub services on public infrastructures [38]. In our stock exchange example, an investment company may want to keep private the interests expressed in the registered subscriptions. The subscription given as an example above, $\{symbol =$"`GOOG`" and $value \geq 1200\}$, might for instance indicate that the corresponding subscriber is interested in selling the corresponding stock. It might also reveal part of her portfolio or divulge her investment strategy. In a e-health system, headers of publications and constraints in subscriptions reveal private information about the patients and their ongoing treatments. Providing confidentiality guarantees is difficult when the

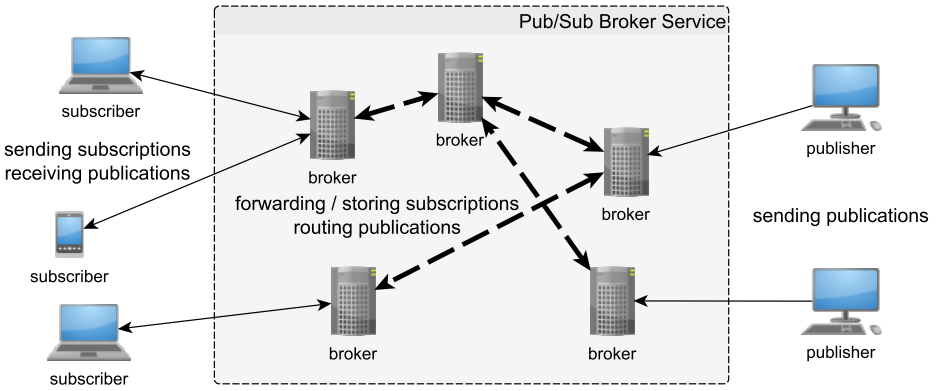

Figure 1: A generic broker-based publish/subscribe system.

brokers implementing the pub/sub service are located in a public infrastructure. Documented exploits [22, 29, 34] indeed show that attackers were able to obtain unauthorized information about the state of virtual machines deployed on public clouds, by exploiting weaknesses in the hypervisor [29,34] or even by exploiting side-channel attacks against the L1 cache in shared multi-core CPUs [22]. Moreover, in the wake of recent global surveillance disclosures, there are increasing concerns related to undesired accesses to private data. Non-uniform regulation on stored data access [21] contributes to such situations, because external entities (e.g., law enforcement agencies) can have different or variable access rights based on where the data is located. As data location is not always transparent to the customers due to migrations performed by cloud providers, privacy guarantees from these providers are not necessarily reliable. Due to these liabilities, the typical assumption is to consider brokers located in public clouds as *honest-but-curious*, and to encrypt all the pub/sub data they manipulate. Unfortunately, normal encryption techniques prevent the brokers from matching messages based on their content. As a result, encryption techniques have been proposed that allow the brokers to match and route *encrypted pub/sub data* without disclosing plaintext content [9, 18, 20, 24, 28]. We call such an operation *encrypted matching*.

Encrypted matching schemes are based on secret key encryption algorithms [9, 18, 20, 28]. Publishers and subscribers must receive a common or correlated secret key. This key is used to encrypt the headers of publications and constraints in subscriptions. Most schemes use symmetric encryption [9, 20, 28], and schemes that adapt public key (asymmetric) encryption algorithms still require some common secret information to be sent to both publishers and subscribers [18]. For all schemes, the secret key information must be updated regularly. Key updates are needed in particular when there are changes in trust relationships between parties communicating through the pub/sub service. For instance, a key change is required when a host is detected as corrupted and must be evicted from the system; the remaining publishers and subscribers using the current key to encrypt the pub/sub data must then receive and use a new key to keep the communication secure. Another case for key updates is the periodic key refreshing required by various schemes for increasing their resilience against brute-force attacks.

The frequency of required key updates varies depending on the application. For instance, in our stock exchange scenario, we can expect frequent key updates as clients unsubscribe or are evicted from the system for various reasons. Furthermore, large quantities of data traffic such as data generated by frequent changes in stock information typically lead to more frequent periodic key refreshes.

When a new key is distributed to publishers and subscribers, all subscriptions stored by the brokers remain encrypted with the previous, invalidated key. These subscriptions can no longer be matched against publications encrypted with the new key. A naive solution requires that all subscribers provided with the new key re-encrypt their subscriptions and resubmit them to the brokers. This solution presents several major drawbacks, described below, that make it unrealistic for most potential applications.

- First, it can result in a spike of network usage, as all subscribers simultaneously need to re-send their newly re-encrypted subscriptions. The completion of the key update phase is also tied to the network layer capabilities, and in particular to the ones of the subscriber with the worst connection. When a large volume of subscriptions requires resubmission, the key update phase can become prohibitively long. The quality of service is also affected: handling registration of subscriptions at the brokers results in higher load and network usage, which ultimately impacts the matching efficiency and increases notification delays.

- Second, resubmission of subscriptions can interfere with the proper operation of the pub/sub service itself. The implementation of this service typically relies on complex algorithms for storing sets of subscriptions at the brokers and for optimizing the matching performance. Subscriptions can be partitioned internally between multiple matching operators to allow parallel processing [2], complex routing tables can be formed to allow efficient communication between the brokers [6, 19, 27], and subscriptions can even migrate between brokers to reduce inter-broker communication [8]. Handling a massive load of updated subscriptions might require a costly re-organization of the data and routing structures, or require the re-optimization of the subscription storage. This can again have a serious impact on the efficiency and responsiveness of the pub/sub service.

- Finally, and as simple as it might seem, key updates handled through subscription resubmission force subscribers to actually keep their previously sent subscriptions at hand. Typically, a pub/sub service offers dependability guarantees (e.g., brokers storing subscriptions can have replicas, or even run more complex mechanisms handling recovery after failures [7]). If subscribers are required to redundantly store subscriptions, they might need to develop or pay for another reliable storage service to handle their own failures.

Pub/sub systems follow a decoupled communication model, where publishers are not aware of the exact subscribers that will receive a publication. Finding a way of distributing common secret key information to publishers and subscribers makes key management cumbersome in this context. Key management was generally considered as orthogonal by prior work on encrypted matching schemes. Consequently, the additional but important issue of subscription invalidation was also left aside. We are, to the best of our knowledge, the first to provide a solution to this problem, in addition to a simple but functional key management scheme adapted to the specificities of privacy-preserving pub/sub based on encrypted matching.

1.1 Our contributions

We present a key update mechanism that offers *in-broker subscription re-encryption*. Untrusted brokers storing subscriptions are given a token K_R related to the encryption key that allows them to directly update encrypted subscriptions *in place*. We develop an extension for a state-of-the-art encrypted matching scheme, Asymmetric Scalar-product Preserving Encryption (ASPE) [9]. We first modify and extend the original scheme to increase its security. Afterwards, we describe the addition of in-broker subscription re-encryption and prove that it maintains the same level of security.

We implement our solution in a prototype featuring a simple key management architecture adapted to the specific nature of pub/sub communication. First, we maintain a partial decoupling between the individual publisher and subscriber nodes. This is one of the major concerns of key management in secure pub/sub systems. Second, we take advantage of the ZooKeeper coordination service [16] for the key distribution and synchronization, inheriting its dependability guarantees. Third, we prove the effectiveness of our solution by evaluating the filtering delays when subject to key updates under a constant flow of incoming publications filtered against large sets of stored subscriptions.

The paper is organized as follows. We present the related work in Section 2. In Section 3, we discuss the original ASPE scheme and our additions. We introduce our novel in-broker subscription re-encryption mechanism for ASPE in Section 4 and analyze its security in Section 5. In Section 6, we present our key management architecture. We present our implementation and its evaluation in Section 7. Finally, we conclude in Section 8.

2. RELATED WORK

We are not aware of any encrypted matching scheme or key management solution that addresses the problem of key updates for encrypted subscriptions stored in untrusted domains. Key management aspects have actually received limited attention for privacy-preserving pub/sub using encrypted matching in general. Published schemes [9, 20, 28] typically defer the provisioning of keys to some third party mechanism. We review below the work related to key management for content-based pub/sub.

Solutions such as [18] and [35] adapt Attribute Based Encryption (ABE) [5, 15], or use similar techniques for publication payload encryption. The message headers are still secured with encrypted matching schemes. *Multi-user searchable data encryption* [12] is used in [18]. This work does not address the key update problem but shares similarities with ours, as brokers apply a proxy re-encryption scheme for both subscriptions and publications. This re-encryption is done for matching purposes and applied to every publication and subscription when they reach a broker. The technique does not consider key updates but still permits revoking a client by removing the corresponding key part from the brokers. This does not cover cases when a periodical key refresh is needed to increase resilience against brute-force attacks or situations when keys are leaked and a key update is mandatory without client eviction. A conceptually similar key management is proposed in [35] for the simpler topic-based pub/sub model. This restricts the encrypted matching to equalities on a single attribute. Both schemes [18, 35] require a central authority for the key distribution and consider a partial coupling model similar to the one used in our implementation.

In [24], the authors present an encrypted matching scheme that adapts the public key Paillier cryptosystem [25]. This scheme does not require distributing a common key to system clients. However, it requires that subscribers initially submit their subscriptions to the publishers from whom they wish to potentially receive publications. These publishers encrypt the subscriptions they receive and return them to subscribers, who in turn submit them to the brokers. This direct interaction between publishers and subscribers adds a strong level of coupling, which goes against one of the primary objectives of pub/sub communication.

The authors of [1] use One-way Function Trees (OFT) [31] for key management. Common key information is distributed to users grouped in various domains. This is similar to our key dissemination architecture. However, their security model does not consider the use of encrypted matching. Instead, brokers are restricted to matching over a limited set of attributes accessible in plaintext form. Therefore, the issue of subscriptions' invalidation is not considered.

In [32], the authors propose a solution for broker-based private matching. Brokers do not store the effective encrypted subscriptions, and therefore the issue of subscriptions' invalidation does not exist. Queries (subscriptions) are broadcast to all system subjects (publishers), who perform a partial matching with their information and return a response. This architecture is fundamentally different from the pub/sub model of existing content-based pub/sub service platforms (e.g., [6, 7, 19, 27]), which is our target.

There exist several peer-to-peer secure pub/sub systems where our work does not apply due to fundamental differences in the functional architecture. In a peer-to-peer pub/sub system there are no brokers: every peer acts as a publisher and a subscriber and performs matching. The assumptions about trust and potential attacks are different and therefore require specific security solutions (e.g., by building on top of communication layers that provide source and destination obfuscation through onion routing [30]). Security frameworks for peer-to-peer pub/sub are described in [11, 37].

3. BACKGROUND: ASPE

The encrypted matching scheme on which we base our work is Asymmetric Scalar-product Preserving Encryption (ASPE). ASPE was initially introduced by [39] for secure kNN (k nearest neighbors) queries on encrypted databases. It was adapted for pub/sub systems in [9]. The security assumptions consider brokers as honest-but-curious entities that are allowed to obtain the matching results but not the information contained in the messages. The attributes in publications' headers and in subscriptions' constraints are represented as coordinates of multidimensional points. Matching publications with subscriptions requires evaluating *Euclidean distance differences* between these points.

3.1 Matching principle

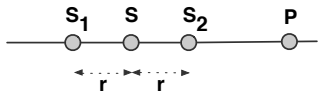

Figure 2: ASPE single dimension example.

For ease of understanding, we first present the matching principle for a single dimension. Consider a subscription S, and a publication P that must be compared to S (assume without loss of generality that $S < P$). Two reference subscription points S_1 and S_2 are chosen, placed symmetrically at a secret random equal distance r from S. This is illustrated in Figure 2. The distance difference, $D_1 \triangleq dist(S_1,P)-dist(S_2,P)$, can be compared to zero to determine the relation between S and P. For instance, we have

$$\begin{aligned} &dist(S_1,P)-dist(S_2,P) > 0 \Leftrightarrow \\ &\quad dist(S_1,P) > dist(S_2,P) \Leftrightarrow S < P. \end{aligned} \tag{1}$$

Note that the distance r and the resulting reference subscription points S_1 and S_2 are chosen by the subscriber independently of the publication P (which naturally the subscriber does not know). Relation (1) above holds no matter if $P > S_2$ or $P < S_2$.

The principle of ASPE is to preserve the ability to perform distance evaluations over ciphertexts, allowing to match S and P. However, the scheme does not preserve simple distance evaluations (e.g., finding the value of $dist(S,P)$) because this would introduce vulnerabilities to known-plaintext attacks [39].

3.2 Multidimensional ASPE

We now describe the encryption scheme when points S and P have multiple dimensions, followed by the encryption and the matching principle themselves. We note that while multidimensional kNN queries are covered in the original scheme for databases [39], it is only briefly mentioned as an extension without proof in the paper adapting it to pub/sub [9]. We provide a more comprehensive description of the multidimensional adaptation of ASPE for pub/sub, as well as a proof. In particular, we show that the adaptation of ASPE to pub/sub requires stricter restrictions on the selection of points representing subscriptions than what is mentioned in the brief description in [9].

Let d be the number of attributes in publications. A subscription contains up to d constraints on these attributes. Each value v_i associated to a constraint for dimension i in the subscription is first encapsulated in an intermediary multidimensional subscription point S_i where dimension i is set to v_i and the other dimensions $k \neq i$ are set randomly. Two reference subscription points $S_{i1} = (q_1,q_2,...,q_{i-1},v_i-r_i,q_{i+1},...,q_d)$ and $S_{i2} = (q_1,q_2,...,q_{i-1},v_i+r_i,q_{i+1},...,q_d)$ are chosen symmetrically at a randomly selected distance r_i from S_i, as in the unidimensional case. The r_i value must be kept private to preserve the security of the scheme. The adaptation in [9] only states that S_{i1} and S_{i2} must be at equal distance from S_i (e.g., in two dimensions, S_{i1} and S_{i2} would be selected on a circle centered on S_i). This is actually insufficient: not only S_{i1} and S_{i2} must be equidistant from S_i, but they also need to have all their values q_k for dimensions $k \neq i$ equal for the matching relation to hold.[1] This constraint is essential in the following theorem, in which we demonstrate the matching result validity.

Matching Theorem. Let $P(x_1,x_2,...,x_{i-1},m,x_{i+1},...,x_d)$, $S_{i1} = (q_1,q_2,...,q_{i-1},v_i-r_i,q_{i+1},...,q_d)$ and $S_{i2} = (q_1,q_2,...,q_{i-1}, v_i+r_i,q_{i+1},...,q_d)$ be three points in a d-dimensional space, such that $y = v_i-r_i$ and $z = v_i+r_i$ ($v_i = (y+z)/2$). Then, $dist(S_{i1},P) \diamond dist(S_{i2},P) \Leftrightarrow m \diamond v_i$, where $\diamond \in \{>,<,=\}$.

Proof. Without loss of generality we consider $\diamond$ as $>$. We have

$$\begin{aligned} &dist(S_{i1},P) > dist(S_{i2},P) \Leftrightarrow \\ &(q_1-x_1)^2+\cdots+(y-m)^2+\cdots+(q_d-x_n)^2 > \\ &\quad (q_1-x_1)^2+\cdots+(z-m)^2+\cdots+(q_d-x_n)^2 \Leftrightarrow \\ &\quad y^2-2ym+m^2 > z^2-2zm+m^2 \Leftrightarrow \\ &\quad 2m > (z^2-y^2)/(z-y) \Leftrightarrow 2m > z+y \Leftrightarrow m > v_i. \end{aligned}$$

■

From the matching theorem, we get that matching between each constraint value v_i and a publication P can be obtained from the sign of the expression $dist(S_{i1},P)-dist(S_{i2},P)$, as seen in (1). The sign of this distance difference is the same as the sign of the expression $D_i \triangleq dist(S_{i1},P)^2-dist(S_{i2},P)^2$. Interestingly, D_i can be expressed as the following sum of scalar products:

$$D_i = \|S_{i1}\|^2-\|S_{i2}\|^2+2(S_{i2}-S_{i1})P. \tag{2}$$

The key principle of ASPE is to encrypt points S_{i1}, S_{i2} and P such that the ability to compute the scalar product between subscription and publication points is preserved, but the scalar product between two subscription points is not preserved. Preserving this scalar product allows to calculate the sign of D_i over the encrypted data. ASPE is a secret key encryption scheme. The information shared by subscribers and publishers is M, an invertible square matrix of dimension $d+1$. The key on the subscriber side is M^T, whereas the key on the publisher side is M^{-1}. As specified in [39], the matrix

[1] We can also prove that the condition in [9] is not sufficient with the following counterexample. Consider a publication P in two dimensions x and y, with $P_x = P_y = 10$. It is represented by the point $P = (P_x,P_y) = (10,10)$. Consider a subscription S with the constraint $x > 5$. It is represented as $S_i = (5,2)$, with its second dimension selected at random. S_{i1} and S_{i2} are selected on the circle of radius $\sqrt{2}$ centered on S_i, as $S_{i1} = (4,3)$ and $S_{i2} = (6,1)$. While $P_x > S_x$, we have $d(S_1,P) = 9.21$ and $d(S_2,P) = 9.84$ so the condition $d(S_1,P) > d(S_2,P)$ is false, yielding a false negative.

key and the plaintext points can be augmented with artificial dimensions for increased resilience against brute force attacks. We also ensure that the matrix M is not orthogonal, which would result in the same encryption keys for both subscribers and publishers. Such a case might lead to known-plaintext attacks, based on finding the exact distance between two publication points as discussed in [39]. The plaintexts S_{i1}, S_{i2} and P are multiplied with the matrix keys to obtain the corresponding encrypted values $S'_{M^T i1}$, $S'_{M^T i2}$ and $P'_{M^{-1}}$:

$$\begin{aligned} S'_{M^T i1} &\triangleq M^T(S_{i1},-0.5\|S_{i1}\|^2)^T \\ S'_{M^T i2} &\triangleq M^T(S_{i2},-0.5\|S_{i2}\|^2)^T \\ P'_{M^{-1}} &\triangleq M^{-1}q(P,1)^T \end{aligned} \tag{3}$$

where q is a positive random scaling factor added for security purposes (denying an attacker the possibility to restrict the solution space when trying to find a key). The additional dimension $-0.5\|S_{i\{1,2\}}\|^2$ is added to the plaintexts to preserve the sign of D_i during encrypted filtering. The matching phase relies on the $MM^{-1} = I_{d+1}$ key reduction that occurs when ciphertexts are multiplied. An untrusted broker that receives $S'_{M^T i1}$, $S'_{M^T i2}$ and $P'_{M^{-1}}$ can compare S with P by evaluating

$$(S'_{M^T i2} - S'_{M^T i1})P'_{M^{-1}} = 0.5qD_i. \tag{4}$$

The result of equation (4) is D_i from (2) scaled by a positive factor. The broker can decide on the matching outcome for the constraint according to the sign of this result.

4. THE ASPE KEY UPDATE

We now present our main contribution: a key update that solves the problem of invalidation of stored subscriptions. We do so by adding support for in-broker subscriptions' re-encryption to ASPE. At each key update, we provide untrusted brokers with a secure token K_R that can be used in a transformation τ to securely re-encrypt the subscriptions they store. This eliminates the need for subscribers to redundantly store their subscriptions and to resubmit them at each key update, and thus relieves the key update from restrictions imposed by the network layer.

Hardened ASPE

We first change the original scheme to increase its resilience to known plaintext attacks (KPAs). The subscriber computes on its own the difference $S'_{M^T i} = S'_{M^T i2} - S'_{M^T i1}$ from (4). $S'_{M^T i}$ is then submitted to the brokers as subscription ciphertext, instead of submitting $S'_{M^T i2}$ and $S'_{M^T i1}$ separately as proposed in the original scheme. We discuss the security implications of this ASPE modification in Section 5.

ASPE with in-broker subscription re-encryption

Our second and main contribution is the support for in-broker subscription re-encryption. Consider a first matrix key M^T and a second key N^T. We need a key update token K_R and a transformation τ such that K_R can re-encrypt a subscription encrypted with M^T into the same subscription encrypted with N^T.

A constraint value v_i in a subscription S encrypted with M^T has the form

$$\begin{aligned} S'_{M^T i} = S'_{M^T i2} - S'_{M^T i1} = \\ M^T((S_{i2},-0.5\|S_{i2}\|^2)-(S_{i1},-0.5\|S_{i1}\|^2))^T. \end{aligned} \tag{5}$$

The same encryption using N^T results in

$$\begin{aligned} S'_{N^T i} = S'_{N^T i2} - S'_{N^T i1} = \\ N^T((S_{i2},-0.5\|S_{i2}\|^2)-(S_{i1},-0.5\|S_{i1}\|^2))^T. \end{aligned} \tag{6}$$

From (5) and (6), we observe that only the multiplication with M^T and N^T changes between the two ciphertexts, whereas the rest of the expression remains the same. Thus, we can define the update token $K_R \triangleq N^T M^{T^{-1}}$ and the transformation τ as

$$S'_{N^T i} \triangleq K_R S'_{M^T i}. \tag{7}$$

For extra obfuscation and for rescaling the matrix elements within acceptable bounds, the new key matrix N^T can be multiplied with a random value r' before distributing K_R to the brokers. This results in $K_R \triangleq (N^T r')M^{T^{-1}}$. The scaling does not impact the matching correctness, and subsequent key updates are completely independent of the previous random factors. We discuss the security implications of the scaling in Section 5.2.

5. SECURITY ANALYSIS

Encrypted matching schemes used in publish/subscribe differ significantly from typical encryption algorithms. Unlike private or public-key cryptography, encrypted matching schemes do not necessarily feature a decryption algorithm, and there is no specific model in the literature for evaluating the security of such schemes. The authors of [28] rightfully argue that the indistinguishability of encrypted messages is limited by the matching possibility itself. This directly restricts the use of standard indistinguishability game proofs considered for attacks where the attacker has the power to actively choose either the plaintext or the ciphertext. The security analysis on the original ASPE scheme for kNN queries [39] covers three attack levels:

- Level 1: the attacker can only access the set C of ciphertexts (ciphertext only attack — COA);
- Level 2: the attacker knows C and a subset of plaintext messages P, but not the exact correspondence between P and the corresponding subset in C;
- Level 3: the attacker knows the exact correspondence between the plaintext subset P and a subset I of the ciphertexts set C (known plaintext attack — KPA).

The security model assumes brokers as honest-but-curious entities, and publishers and subscribers as trusted users of the system. An attacker is considered to be an external malicious entity that manages to access the encrypted information stored or communicated by the brokers. The analysis in [39] concludes that the basic scheme version of ASPE is secure up to Level 2; for Level 3 the attacker is able to form and solve a system of equations that allows obtaining the key. The ASPE adaptation for pub/sub in [9] mostly relies on the same evaluation, with a little extra analysis centered on a particular Level 1 attack.

The authors of [9] also briefly present the possible use of an intermediary trusted security manager for addressing collusion between malicious subscribers and untrusted brokers. The approach forbids disseminating encryption keys to

subscribers. The keys are given to the security manager, to whom subscribers must send each subscription for encryption in a preliminary step. The encrypted subscriptions are then forwarded to the brokers. Our key management architecture could forbid collusion in this manner, as we describe in Section 6.3. However, handling collusions through such a delegation of encryption, or not considering collusion at all, are equivalent problems for the above attack levels (with encryption delegation the subscribers do not possess key information, so they cannot collude). Therefore, we do not consider collusions in our analysis.

5.1 Hardened ASPE security

We start by analyzing the security of our Hardened ASPE extension. In an informal way, we follow the approach from [39] and prove that Hardened ASPE is resilient to Level 3 attacks. We consider as encryption key the matrix M^T with elements $m_{i,j}$. r_i is the random distance used in the encryption of each constraint as described in Section 4, and d is the number of attributes in the messages. Let us assume that the attacker has Level 3 knowledge of the mapping between each plaintext value v_i in a constraint and its ciphertext $S'_{M^T i} = (e_{1i}, e_{2i}, ..., e_{(d+1)i})$. From this correspondence and from $S'_{M^T i} = S'_{M^T i2} - S'_{M^T i1}$, for each v_i the attacker can form the system

$$\begin{aligned} 2r_i(m_{1,i} - v_i m_{1,d+1}) &= e_{1i}; \\ 2r_i(m_{2,i} - v_i m_{2,d+1}) &= e_{2i}; \\ &\vdots \\ 2r_i(m_{d+1,i} - v_i m_{d+1,d+1}) &= e_{(d+1)i}. \end{aligned} \tag{8}$$

The system has d+1 equations, and its 2d+3 unknowns are r_i and the i-th and d+1-th columns of the matrix key M^T. The attacker can augment this system with similar equation sets obtained from other constraints, possibly chosen from other subscriptions. For any added constraint on attribute i the number of equations is increased by d+1, and the number of unknowns by 1 (a different r_i, the matrix key remaining the same). Therefore, the attacker can build an overdetermined system, which he can attempt to solve.

Security Theorem 1. Any system SYS composed of multiple subsystems of type (8), having as unknowns a random value r_i per component subsystem and the elements of the matrix key M^T (where $M^T \neq [0]_{d+1}$[2]), has an infinity of solutions.

Proof. Let us consider one component subsystem of type (8). By solving for r_i in each equation, the subsystem can be rewritten as

$$\begin{aligned} \frac{e_{1i}}{m_{1,i} - v_i m_{1,d+1}} = \frac{e_{2i}}{m_{2,i} - v_i m_{2,d+1}} = \\ \cdots = \frac{e_{(d+1)i}}{m_{d+1,i} - v_i m_{d+1,d+1}}. \end{aligned} \tag{9}$$

The terms in (9) can be paired in equations of the form

$$e_{ji} m_{k,i} - e_{ji} v_i m_{k,d+1} - e_{ki} m_{j,i} + e_{ki} v_i m_{j,d+1} = 0. \tag{10}$$

The subsystem is composed exclusively of equations without a free term. The same reduction can be applied to any other component subsystem of SYS. Therefore, any system SYS is linear and homogenous. Consequently, the system has at least one solution for the unknown matrix elements, i.e., the all-zero matrix. We also know that this is not the only solution, since the key matrix cannot be null. It follows that the solution for M^T is the nullspace of the system's SYS coefficient matrix. A nullspace determines an infinity of individual solutions (although having a specific structure) [36]. ■

From Security Theorem 1, a Level 3 attacker cannot determine the key by solving the system SYS no matter how many subscriptions are in his possession. The authors of [39] followed a similar analysis for the original ASPE scheme by forming a system of equations that can be solved, the basic scheme being secure only to a Level 2 attack. The system of equations can also be solved for the pub/sub adapted ASPE presented in [9]. In both cases, the system can be solved because the brokers are given two separate vectors S'_{Ki1} and S'_{Ki2} instead of only $S'_{Ki} = S'_{Ki1} - S'_{Ki2}$ as we propose. It should be noted that if a random value used for encryption is leaked (r_i in (8)), then a Level 3 attacker can solve our system like it can solve the scheme from [9]. However, guessing a random value used for encryption is also critical in other schemes (e.g., in [24] the encryption mechanism uses two random values that must be kept secret).

Our hardened ASPE thus provides a greater level of security than the original scheme, adding resilience against Level 3 attacks. An apparent drawback of the increased security is that the broker can no longer determine coverage between encrypted subscriptions (e.g., whether a subscription like $value > 100$ is more general than $value > 300$), which denies possible matching optimizations. The possibility to determine coverage relations is, however, not recommended from a security point-of-view as it may allow inferring the nature of subscriptions based on the knowledge of their definition domain [28]. Recent techniques have been specifically developed for improving matching performance when the coverage support is disallowed in scenarios demanding security [3].

5.2 In-broker subs. re-encryption: security

We now analyze the security impact of the ASPE in-broker subscriptions re-encryption; we formally prove that the secure tokens do not help the attacker to derive any of the secret keys. We recall that a key update implies sending to the brokers the matrix product $K_R = (N^T r')M^{T^{-1}}$. To simplify the discussion, and without loss of generality, we consider $r' = 1$ in the following. Each element of $M^{T^{-1}}$ can be expressed using the terms in the original key: $M^{T^{-1}} = \frac{adj(M^T)}{|M^T|}$, where $adj(M^T)$ is the adjugate of M^T and $|M^T|$ its determinant. Let $N^T(i)$ and $K_R^T(i)$ be the ith column of the matrices N^T and K_R^T, respectively. From the definition of K_R^T, it follows that

$$\begin{bmatrix} adj(M^T)^T & [0]_{d+1} & \dots & [0]_{d+1} \\ [0]_{d+1} & adj(M^T)^T & \dots & [0]_{d+1} \\ \dots & & & \\ [0]_{d+1} & [0]_{d+1} & \dots & adj(M^T)^T \end{bmatrix} \begin{bmatrix} N^T(1) \\ N^T(2) \\ \dots \\ N^T(d+1) \end{bmatrix} = \begin{bmatrix} K_R^T(1)|M^T| \\ K_R^T(2)|M^T| \\ \dots \\ K_R^T(d+1)|M^T| \end{bmatrix}. \tag{11}$$

This expression corresponds to the system of equations that can be formed by an attacker. If $a_{j,k}$ and $t_{j,k}$ are respectively

[2] $[0]_{d+1}$ denotes the null square matrix of size d+1.

the elements of $adj(M^T)$ and K_R, the system of equations can be written as

$$\begin{aligned} a_{1,1}n_{1,1}+a_{2,1}n_{1,2}+ \ ... \ +a_{d+1,1}n_{1,d+1} &= t_{1,1}|M^T|; \\ a_{1,2}n_{1,1}+a_{2,2}n_{1,2}+ \ ... \ +a_{d+1,2}n_{1,d+1} &= t_{1,2}|M^T|; \\ ... & \\ a_{1,d+1}n_{d+1,1}+a_{2,d+1}n_{d+1,2}+ \ ... \ +a_{d+1,d+1}n_{d+1,d+1} & \\ = t_{d+1,d+1}|M^T|. & \end{aligned} \tag{12}$$

Security Theorem 2. Any system SYS composed of multiple subsystems of type (8), augmented with a subsystem of type (12), which adds as unknowns the elements of the matrix key N^T (where $N^T \neq [0]_{d+1}$), has an equivalent space of infinite solutions as the non-augmented system SYS.

Proof. We know from Security Theorem 1 that the non-augmented system SYS has an infinity of solutions. Let us consider one of these solutions for the matrix key M^T. Keeping the elements of the new key N^T as unknowns, we can observe that (12) becomes a linear system. This means given a solution for M^T in system SYS, a specific dependent solution for N^T *may* exist in the augmented system.

We want to prove that the space of infinite solutions for the augmented system is not smaller than for the non-augmented SYS. For this to hold, given a solution for M^T a specific dependent nonzero solution for N^T *must* exist, otherwise the exploration space for an attacker might be reduced by the augmentation. The coefficient matrix of system (12) is the first factor in the left-hand side of (11) and is a diagonal block matrix. Its determinant is the product of the determinants of the diagonal blocks $adj(M^T)^T$. Since M^T is invertible, we know that the determinant of $adj(M^T)^T$ is nonzero. It follows that the determinant of the coefficient matrix in system (12) is nonzero, which proves that there is a nonzero solution for N^T for any nonzero solution for M^T. ■

As proved by Security Theorem 1, the set of solutions for M^T in case of a Level 3 attack is expressed through a nullspace form and is of infinite size. From system (12), without a solution for M^T, an attacker cannot determine the dependent solution for N^T (and vice versa). Therefore, the security of in-broker subscription re-encryption for ASPE depends on the security of the original ASPE scheme. Moreover, Security Theorem 2 proves that the security of the original scheme against a Level 3 attack is not impacted by the introduction of the token K_R.

We are aware that the proposed ASPE in-broker subscription re-encryption presents forward and backward secrecy issues. More precisely, if one key M^T is leaked, then an attacker having access to an untrusted broker can try to obtain the key N^T from M^T and $K_R = (N^T r')M^{T^{-1}}$. The random scaling factor r' is the only obstacle that prevents the attacker from learning the new key. Note that the original ASPE scheme [9, 39] uses a similar scaling in the publication encryption (the q factor in Eq. (3)). For forward and backward secrecy purposes, we consider that a key update should be periodically performed without sending the secure token. This will require subscription resubmission, but for such cases this can be performed as a background operation, done at a slower pace and without strict constraints on the key update duration. Moreover, a subscriber could partition his subscriptions based on different sensitivity levels or subscription duration, and execute the subscription resubmission only for a smaller set of highly critical or persistent subscriptions. The transitional state design we propose in Section 6 as part of the key distribution protocol reduces the processing delays of publications in such situations.

6. KEY MANAGEMENT

We now describe a key management architecture adapted for privacy-preserving pub/sub through encrypted matching. The solution follows a simple design: its purpose is to exemplify the use of in-broker subscription re-encryption. It also takes into account two specific aspects of key management for pub/sub: the need for service continuity while a key update is in progress and the need for reliability guarantees.

6.1 Preliminaries

We consider security domains groupings for subscribers, publishers and brokers. Domains have specific rights for accessing information. Setting such rights based on knowing which domains interact is a reasonable assumption for key management applied in a practical scenario [1], and provides the ability to disseminate the proper key information to interacting domains. For instance, in the stock exchange example introduced in Section 1, an investment company knows from which stock exchange it wants to obtain the publication flow (e.g., New York, London, etc). This implies that the stock exchange and the investment company should use a compatible key in their data encryption. Pub/sub decoupling is still maintained at the level of individual hosts: a host belonging to the investment company that registers a subscription does not know the exact host that emits publications. They however receive compatible keys for encryption, knowing that the domains to which they belong should be able to communicate.

Without lack of generality, for simplicity of description we consider the case of three domains: one for subscribers, one for publishers and one for brokers. We define the following participating entities:

- KDC: a central *key distribution coordinator* (trusted authority);
- KDM_s, KDM_p, KDM_b: three *key domain managers* (dedicated hosts in each of the subscriber, publisher and broker domains, respectively);
- the *individual hosts*: publishers, subscribers and brokers in each domain.

The KDC maintains a set of *cryptographic contexts*. Each context is a set of compatible key information: a publication encryption key for publishers, a subscription encryption key for subscribers and a key update token for brokers. The key information between contexts does not need to be compatible, for instance the size of the keys can be different.

We define a *secure pub/sub chain* as a set of publisher-broker-subscriber security domains used for pub/sub data encrypted under the same cryptographic context. For example, when an investment company registers encrypted subscriptions to a pub/sub broker service for encrypted publications emitted by a stock exchange, the three domains form a secure pub/sub chain. The KDC keeps the mapping from cryptographic contexts to secure pub/sub chains where these are used. A domain can be part of multiple secure pub/sub chains, and therefore use multiple cryptographic contexts.

6.2 Service reliability

Our key distribution implementation relies on the ZooKeeper [16] coordination service. ZooKeeper simplifies the coordination and dependability aspects of distributed applications. It offers access to a shared namespace organized hierarchically for the application nodes, similarly to a file system. ZooKeeper uses an access control list (ACL) system to control the access to the shared configuration data. One of the features of ZooKeeper is the support for *watches*, which allow a client to be notified when there is an update (including creation and deletion) to a particular datum in the namespace. The KDC and the KDMs use watches for synchronization of the key distribution.

The main reason for using ZooKeeper is making the key management system highly-available. ZooKeeper provides dependability and consistency guarantees by replicating the state over multiple servers and ensuring update linearizability. The state in the key distribution protocol is stored in the ZooKeeper namespace. If the KDC or one of the KDMs crashes while executing the protocol, it can be restarted, read its latest configuration from ZooKeeper and seamlessly resume its execution.

6.3 Key distribution

The KDC distributes the key information contained in a cryptographic context to the KDMs in the secure pub/sub chain using that context. The particular key given to each KDM depends on the domain: a specific *pub/sub encryption key* to KDM_p and KDM_s and the key update token to KDM_b. The KDC encrypts the distributed key with the KDM's public key.[3] In a practical scenario, the number of domains in a secure pub/sub chain is limited. This allows using individual public key encryption in the KDC to KDM distribution.

Further, the KDMs distribute the key information to the hosts in their security domain. We consider for our simple demonstration implementation that intra-domain communication is secure (e.g., the hosts are isolated in a network that is private to that domain). In a production deployment, the intra-domain distribution could take different forms depending on the use case, optimization needs and security requirements (e.g., by adapting other secure group communication protocols [26, 31]).

A key update is triggered by a KDM joining or leaving a secure pub/sub chain, by an explicit request (e.g., when a individual host in the KDM's domain gets corrupted and has to be evicted), or simply as a periodical key refresh.

We now describe the complete key update protocol phase. It is illustrated in Figure 3. The implementation of the protocol steps is based on interactions through the ZooKeeper coordination service. Reaction to new data (e.g., when a new key is available) or joining mechanisms (e.g., when the KDC waits upon KDMs to complete a distribution) are implemented with ZooKeeper watches on the corresponding data in the shared namespace.

Using ZooKeeper permits to resume the protocol based on the configuration state. This state is saved by the coordination service in case of failure during any of the steps. The protocol steps are: **1)** The KDC stores the new key information in the ZooKeeper namespace (the key itself or the key update token), encrypted with the corresponding KDMs's public keys. The KDC sets an *updated* flag. Through previously set ZooKeeper watches, all KDM_s and KDM_b are notified of the availability of a new key. **2)** All KDM_s and KDM_b retrieve and decrypt the new key information. They disseminate that key information in their domain. **3)** Subscribers obtain the new key. They start using it for encryption and notify the KDM_s. Brokers obtain the key update token. They start re-encrypting the stored subscriptions. They keep both the new and the old subscriptions sets in a *transitional state*. When the re-encryption is finished, they notify the KDM_s. **4)** When they have been notified by all hosts in their domains, KDM_s and KDM_b notify the KDC through setting a *retrieved* flag. **5)** The KDC waits for the setting of *retrieved* flags by all KDM_s and KDM_b. Then, it sets an *updated* flag to notify all KDM_p about the new key. **6)** Each KDM_p retrieves and decrypts the new key information. It notifies the publishers in its domain. **7)** Publishers obtain the new key. They start using it in encryption, and notify their KDM_p. **8)** Each KDM_p waits for notification from all publishers in its domain. It then sets a *retrieved* flag. **9)** The KDC waits for *retrieved* flag notifications from all KDM_p. In turn, it notifies the KDM_bs by a *validated* flag. **10)** The KDM_bs are notified of the *validated* flag. They notify the brokers in their domains. This triggers the end of the *transitional state* and the deletion of the old subscriptions, which ends the process.

As described in step 3, the brokers maintain a transitional state and keep the subscriptions encrypted with the old key along with the ones encrypted with the new key. The transitional state ensures that the pub/sub service is continuously functional, which is essential for critical applications.

Since step 3 is executed in parallel at both subscriber and broker hosts, it might happen that a broker receives a subscription encrypted with an old key after the stored subscriptions have already been re-encrypted. We handle this case by allowing the broker to re-encrypt *on-the-fly* such subscriptions and storing them along the others.

The protocol guarantees that publications encrypted with the new key are sent only after all stored subscriptions are in the new re-encrypted form. Also, subscriptions encrypted with the old key are removed only after the publisher hosts start using the new key. However, due to network asynchrony, it might happen that some publications encrypted with an older key are delayed and arrive at the broker after the subscriptions encrypted with the old key have been removed. To avoid this problem, we make an assumption on a bound of such asynchrony and maintain the transitional state for a short additional time (typically, less than a second).

As mentioned in Section 5 the ASPE adaptation for pub/sub in [9] assumes the existence of an intermediate trusted security manager for encrypting subscriptions and eliminating collusions between potentially malicious subscribers and brokers. Although we believe that delegating subscription encryption would introduce additional overhead, which is not experimentally evaluated in [9], our key management architecture can be easily adapted to accommodate such a solution. The KDM_s can play the role of the security manager in [9] and handle the subscriptions encryption. In such case steps 2 and 3 in the key update protocol are eliminated, the final point for the key dissemination being KDM_s. The subscribers would send their subscription for encryption to KDM_s, receive them back and submit them to brokers, as suggested in [9]. Note that following such an approach, the

[3] In our implementation we use the Elliptic Curve Integrated Encryption Scheme (ECIES), ISO 18033-2 [33].

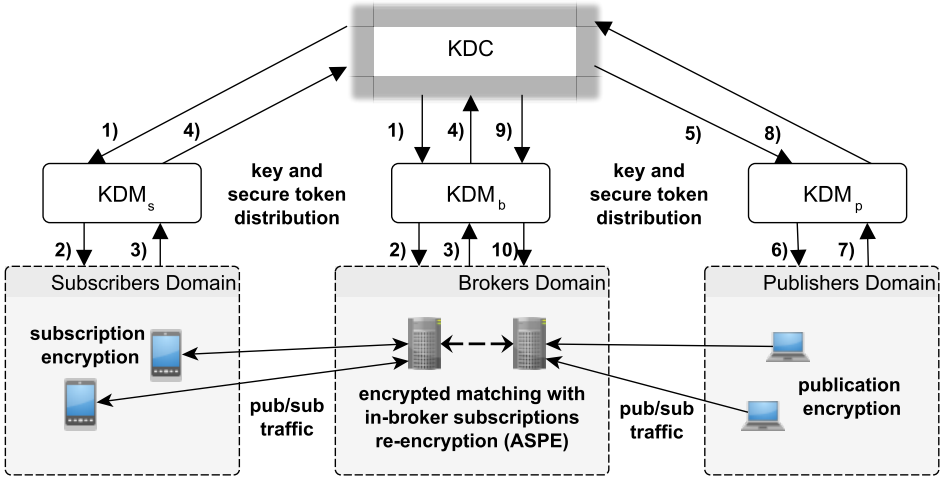

Figure 3: Key update protocol phases.

effectiveness of our in-broker subscription re-encryption solution would be even higher. Without in-broker subscription re-encryption each key update will inflict a massive overhead on the KDM_s caused by re-encrypting all subscriptions for all subscribers.

7. EVALUATION

In this section, we present the evaluation of our solution. We start by describing the test platform setup, and we follow with relevant evaluations demonstrating the efficiency of our approach.

7.1 Experimental settings

We implement our solution on top of an existing high-performance pub/sub system, StreamHub [2]. This system splits the pub/sub service functionality into three successive operations: partitioning, matching and notification. These operations are implemented as a chain of operators. Each operator is deployed over several machines and its load is partitioned. The central operation, matching stored subscriptions with an incoming flow of publications, corresponds to the *Matcher* operator. Each instance of the Matcher operator corresponds to a broker in the pub/sub model. The stored subscriptions are replicated on all Matchers and we split the incoming flow of publications (note that, as described in [2], the opposite is also possible but yields higher notification delays). Our implementation is done in C++ and we use the Crypto++ and ZooKeeper libraries for the key exchange protocol.[4]

We deploy the system on 12 nodes, each with 8 Xeon cores at 2.4 GHz and 8 GB of memory. Six nodes are dedicated to the key management infrastructure: KDC, KDMs, and ZooKeeper server replicas. Six nodes support the operators, among which three of them are used to deploy 15 instances of the Matcher that perform the filtering in parallel. The remaining nodes are used for workload injection. We use the implementation of the complete key update protocol including all steps presented in Section 6.3. In our first tests (Sections 7.2 and 7.3) we emulate the set of subscribers and one publisher on a single host. This simplification impacts the overall system key dissemination time by abstracting the group dissemination time for the subscribers set. In a real large-scale deployment, this time will depend on the number of subscribers and their transport links characteristics.[5] To investigate this case as well, in the final test of our evaluation (Section 7.4) we emulate up to 30 subscriber hosts, obtaining acceptable increases in the key dissemination time. This comes, however, as a secondary result. The main purpose of our evaluation is to show the critical advantage brought by the in-broker subscription re-encryption and its impact on the service quality of the pub/sub engine itself.

7.2 Key update with in-broker subscription re-encryption

We first test the impact of a key update on the pub/sub engine performance when we use the in-broker subscription re-encryption. We use a constant flow of 500 publications per second. The system stores 5,000 subscriptions replicated on the 15 Matcher instances. We use a synthetic workload of subscriptions and publications, each with 4 constraints or attributes. Each incoming publication matches 0.2% of the stored subscriptions, triggering 10 notifications. We trigger a key update when sending the 138^{th} publication. Figure 4 shows the delay between the emission of a publication and the reception of the first notification. The total key update time is measured at the central KDC for the duration of the entire key update phase across all domains. The key update time of brokers is measured at the broker domain's KDM, starting with a key update notification trigger and ending with the final confirmation that the previous-key encrypted subscriptions have been deleted. In this experiment the total time for the key update is 184 ms, with 120 ms specifically for the protocol steps involving the broker domain. The impact of the in-broker subscription re-encryption on the experienced delay in normal conditions is present, but minimal. The 60 publications processed during the key update phase involving the brokers have a median notification delay of 7.1 ms (average 8.1 ms, standard deviation 2.6 ms) compared

[4] `http://www.cryptopp.com/` and `http://zookeeper.apache.org/`

[5] External protocols [26, 31] can be adapted to optimize the distribution.

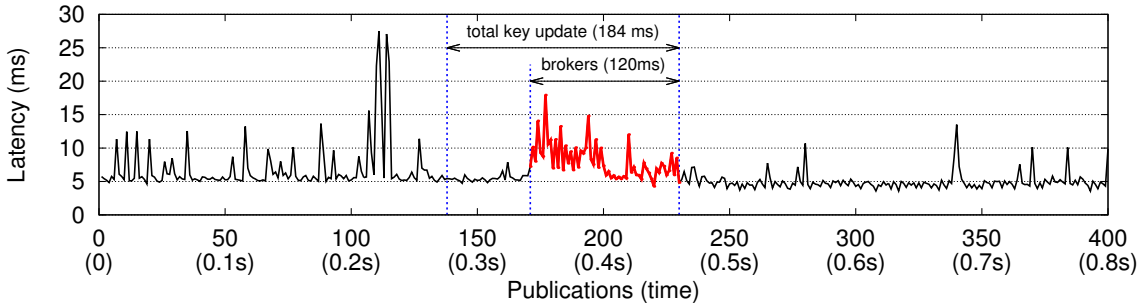

Figure 4: Impact of a key update on the notification delay.

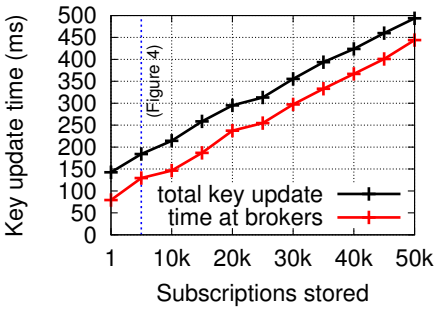

Figure 5: Impact of subscriptions set size on key update time.

to the median notification delay of 4.7 ms (average 4.7 ms, standard deviation 1.2 ms) for the 160 publications that follow the key update.

In Figure 5, we present the impact of the number of subscriptions stored at the brokers on the key update time, from 1 to 50,000 subscriptions. As expected, the total time of in-broker subscription re-encryptions increases linearly with the number of stored subscriptions, but is able to re-encrypt a massive number of subscriptions in less than half a second. This is orders of magnitude faster than the time it would take for all subscribers to re-encrypt and resubmit the same set of subscriptions.

We note that a broker must maintain the stored encrypted subscriptions in memory while re-encrypting these. The space complexity of ASPE encrypted subscriptions is in $O(d^2)$ where d is the number of constraints in a subscription. More precisely, as described in Section 3.2, for each constraint value, the encrypted equivalent requires two vectors of floating point values, each of size $d+1$. Despite the quadratic space complexity, in practice we observe that the memory overhead inflicted at the brokers level is negligible. For our test settings, using subscriptions with 4 constraints, the ASPE encrypted subscription state size was approximately 45 MB for 50,000 subscriptions.

7.3 Key update without in-broker subscription re-encryption

We emulate the case with 5,000 subscriptions when in-broker subscription re-encryption is disabled. In this situation, the subscribers must re-encrypt and resubmit their subscriptions. We emulate the limitation at the network layer by capping the bandwidth at a maximal number of subscriptions sent per second. When the key update is triggered, the workload injection machine emulates the resubmission by re-encrypting and sending 100, 250 or 500 subscriptions per second. We present the measured notification delays and the total key update time in Figure 6. Due to the large number of publications processed during the key update, we present the median of the notification delays rather than all actual times as we did for Figure 4. The key update time is now completely dominated by the time required for subscribers to re-send their subscriptions. If the key update is triggered to revoke a previously authorized subscriber, the revocation may only be effective after several seconds. This leaves a much longer window of opportunity for an attacker.

Another important aspect is the impact on the measured notification delays, which depends on the rate of received re-subscriptions. The increase in notification delay results from the extra load imposed by the re-subscriptions, which have to be processed by the Matchers. It illustrates the tradeoff between the total key update time and its impact on the service performance. Trying to minimize the key update time by increasing the re-subscription rate has a clear negative impact on notification delivery time. Using in-broker subscription re-encryption is clearly a better approach in this respect.

7.4 Key update protocol scalability

The primary purpose of our key update protocol proposed in Section 6.3 was to exemplify the use of in-broker subscription re-encryption. However, we also evaluate the key update scalability outside the in-broker subscription re-encryption step to highlight the costs of the protocol itself. We consider the case where each broker only stores one subscription, thus the in-broker subscription re-encryption takes negligible time. The results of this additional experiment are presented in Figure 7.

The longer key update resulting from an increase in the number of hosts is limited: only up to 6 ms per added subscriber host. We observe in our evaluation that the key distribution time is dominated by the largest domain group. This is the reason why the variations obtained for 30 subscribers with 5, 10, and 15 brokers are not significant (in the range of 20 ms). We believe that splitting the domains into smaller groups, when possible, might provide interesting performance gains but leave this investigation for future work.

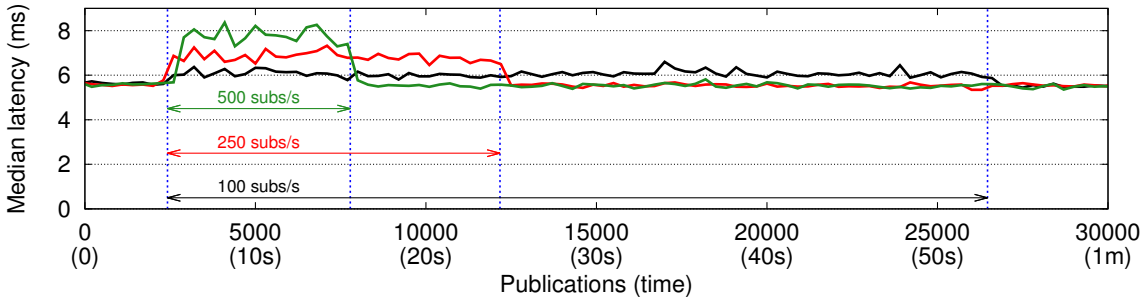

Figure 6: Key update impact without in-broker subscription re-encryption.

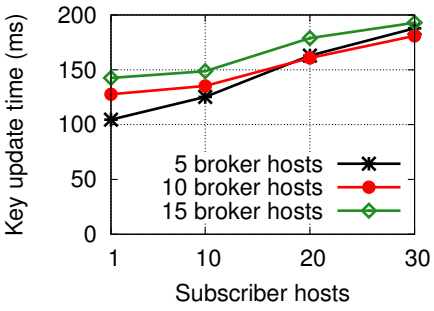

Figure 7: Key distribution time.

8. CONCLUSION AND FUTURE WORK

We presented a novel mechanism for key update in privacy-preserving pub/sub systems taking into account the problem of invalidation of subscriptions when the key is updated. The use of *in-broker subscription re-encryption* eliminates the need for subscribers to re-encrypt and resubmit their subscriptions when a key update is triggered. We augmented and strengthened the ASPE encrypted matching scheme [9] to support this feature. Our test implementation of the key update protocol maintains a level of partial decoupling between publishers and subscribers, ensures service continuity and has a minimal impact on the notification delays.

We also consider this work as a first step towards extended research on key management in scenarios where key-dependent encrypted queries are stored in untrusted domains and need to be updated in-place upon a key update. For instance, operators of Complex Event Processing (CEP) engines and active databases in general could also benefit from similar techniques, using other encrypted processing techniques (e.g., searchable symmetric encryption [10]).

Our current implementation is well-suited for corporate business software destined to cloud platforms integrated broadband services. As future work, we consider adapting our key update mechanism for event-based applications that follow a producer/consumer design pattern, typically used in systems where key update has an even more critical impact. Possible applications include mobile systems and Internet-of-Things dedicated platforms. Adapting a lightweight solution like ours fits with the severe bandwidth, storage and computational power constraints in such systems.

Finally, it would be interesting to adapt other encrypted matching schemes than ASPE for supporting the key management architecture we proposed. This can be complemented by improvements in the key distribution performance through solutions such as OFT [31], which could decrease the cardinality of groups affected by a key exchange in a domain. We believe such solutions could easily integrate with our hierarchical architecture model. This is a research avenue we intend to pursue.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257843 (SRT-15 project).

9. REFERENCES

[1] J. Bacon, D. M. Eyers, J. Singh, and P. R. Pietzuch. Access control in publish/subscribe systems. In *DEBS 2002: 2nd ACM Int. Conf. on Distributed Event-Based Systems*.

[2] R. P. Barazzutti, P. Felber, C. Fetzer, E. Onica, M. Pasin, J.-F. Pineau, E. Rivière, and S. Weigert. StreamHub: A massively parallel architecture for high-performance content-based publish/subscribe. In *DEBS 2013: 7th ACM Int. Conf. on Distributed Event-Based Systems*.

[3] R. P. Barazzutti, P. Felber, H. Mercier, E. Onica, and E. Rivière. Thrifty privacy: efficient support for privacy-preserving publish/subscribe. In *DEBS 2012: 6th ACM Int. Conf. on Distributed Event-Based Systems*.

[4] R. P. Barazzutti, T. Heinze, A. Martin, E. Onica, P. Felber, C. Fetzer, Z. Jerzak, M. Pasin, and E. Rivière. Elastic scaling of a high-throughput content-based publish/subscribe engine. In *ICDCS 2014: IEEE International Conference on Distributed Computing Systems*.

[5] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *SP 2007: IEEE Symp. on Security and Privacy*.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19, 2001.

[7] R. Chand and P. Felber. XNet: a reliable content based publish subscribe system. In *SRDS 2004, 23rd Symposium on Reliable Distributed Systems*.

[8] A. Cheung and H.-A. Jacobsen. Load balancing content-based publish/subscribe systems. *ACM Trans. Comput. Syst.*, 28(4), 2010.

[9] S. Choi, G. Ghinita, and E. Bertino. A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations. In *DEXA 2010: Database and Expert Systems Applications*.

[10] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *CCS 2006: 13th ACM conference on Computer and comm. security*.

[11] G. Di Crescenzo, B. Coan, J. Schultz, S. Tsang, and R. N. Wright. Privacy-preserving publish/subscribe: Efficient protocols in a distributed model. In *Data Privacy Management and Autonomous Spontaneous Security*, LNCS. Springer, 2014.

[12] C. Dong, G. Russello, and N. Dulay. Shared and searchable encrypted data for untrusted servers. In *2008: Data and App. Security*.

[13] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[14] B. Eze, C. Kuziemsky, L. Peyton, G. Middleton, and A. Mouttham. Policy-based data integration for e-health monitoring processes in a B2B environment: experiences from Canada. *Journal of theoretical and applied electronic commerce research*, 5(1), 2010.

[15] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS 2006: 13th ACM conference on Computer and communications security*.

[16] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free coordination for internet-scale systems. In *ATC 2010: USENIX Annual Technical Conference*.

[17] M. Ion, G. Russello, and B. Crispo. An implementation of event and filter confidentiality in pub/sub systems and its application to e-health. In *CCS 2010: 17th ACM conf. on Computer and comm. security*.

[18] M. Ion, G. Russello, and B. Crispo. Design and implementation of a confidentiality and access control solution for publish/subscribe systems. *Computer Networks*, 56(7), 2012.

[19] H.-A. Jacobsen, A. Cheung, G. Lia, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The PADRES publish/subscribe system. In *Handbook of Research on Adv. Dist. Event-Based Sys., Pub./Sub. and Message Filtering Tech.*, 2009.

[20] J. Li, C. Lu, and W. Shi. An efficient scheme for preserving confidentiality in content-based publish/subscribe systems. Technical Report GIT-CC-04-01, Georgia Institute of Technology, 2004.

[21] S. Liston. The Cloud: data protection and privacy – Whose cloud is it anyway? In *GSR 2012: 12th Global Symposium for Regulators*.

[22] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy*, 2015.

[23] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram. Scalable ranked publish/subscribe. *Proc. VLDB Endow.*, 1(1):451–462, Aug. 2008.

[24] M. Nabeel, N. Shang, and E. Bertino. Efficient privacy preserving content based publish subscribe systems. In *SACMAT 2012: 17th ACM symposium on Access Control Models and Technologies*.

[25] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT 1999: 17th International Conference on Theory and Application of Cryptographic Techniques*.

[26] A. Perrig, D. Song, and J. D. Tygar. ELK, a new protocol for efficient large-group key distribution. In *SP 2001: IEEE Symposium on Security and Privacy*.

[27] P. R. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *ICDCS 2002: 22nd International Conference on Distributed Computing Systems*.

[28] C. Raiciu and D. S. Rosenblum. Enabling confidentiality in content-based publish/subscribe infrastructures. In *Securecomm 2006: 2nd Int. Conf. on Security and Privacy in Comm. Networks*.

[29] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *CCS 2009: 16th ACM conference on Computer and communications security*.

[30] V. Schiavoni, E. Rivière, and P. Felber. Whisper: Middleware for confidential communication in large-scale networks. In *ICDCS 2011: IEEE International Conference on Distributed Computing Systems*.

[31] A. T. Sherman and D. A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering*, 29(5):444–458, May 2003.

[32] A. Shikfa, M. Önen, and R. Molva. Broker-based private matching. In *PETS 2011: 11th Int. Conf. on Privacy Enhancing Technologies*.

[33] V. Shoup. A proposal for an iso standard for public key encryption. Technical report, IBM Zurich, 2001.

[34] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, and L. Lo Iacono. All your clouds are belong to us: security analysis of cloud management interfaces. In *CCSW 2011: 3rd ACM workshop on Cloud computing security*.

[35] M. Srivatsa and L. Liu. Secure event dissemination in publish-subscribe networks. In *ICDCS 2007: IEEE International Conference on Distributed Computing Systems*.

[36] G. Strang. *Introduction to Linear Algebra, 4th edition*. Wellesley-Cambridge Press and SIAM, 2009.

[37] M. A. Tariq, B. Koldehofe, A. Altaweel, and K. Rothermel. Providing basic security mechanisms in broker-less publish/subscribe systems. In *DEBS 2010: 4th ACM Int. Conf. on Distributed Event-Based Systems*.

[38] C. Wang, A. Carzaniga, D. Evans, and A. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *HICSS 2002: 35th Annual Hawaii International Conference on System Sciences*.

[39] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis. Secure kNN computation on encrypted databases. In *ACM SIGMOD 2009: international conference on Management of data*.