Efficient Privacy Preserving Content Based Publish Subscribe Systems

Mohamed Nabeel, Ning Shang, Elisa Bertino
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
{nabeel, nshang, bertino}@cs.purdue.edu

ABSTRACT

The ability to seamlessly scale on demand has made Content-Based Publish-Subscribe (CBPS) systems the choice of distributing *messages/documents* produced by *Content Publishers* to many *Subscribers* through *Content Brokers*. Most of the current systems assume that *Content Brokers* are trusted for the confidentiality of the data published by *Content Publishers* and the privacy of the subscriptions, which specify their interests, made by *Subscribers*. However, with the increased use of technologies, such as service oriented architectures and cloud computing, essentially outsourcing the broker functionality to third-party providers, one can no longer assume the trust relationship to hold. The problem of providing privacy/confidentiality in CBPS systems is challenging, since the solution to the problem should allow *Content Brokers* to make routing decisions based on the content without revealing the content to them. The previous work attempted to solve this problem was not fully successful. The problem may appear unsolvable since it involves conflicting goals, but in this paper, we propose a novel approach to preserve the privacy of the subscriptions made by *Subscribers* and confidentiality of the data published by *Content Publishers* using cryptographic techniques when third-party *Content Brokers* are utilized to make routing decisions based on the content. Our protocols are expressive to support any type of subscriptions and designed to work efficiently. We distribute the work such that the load on *Content Brokers*, where the bottleneck is in a CBPS system, is minimized. We extend a popular CBPS system using our protocols to implement a privacy preserving CBPS system.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

SACMAT'12, June 20–22, 2012, Newark, New Jersey, USA.

General Terms

Design, Protocols, System, Implementation

Keywords

Publish subscribe, Privacy, Confidentiality

1. INTRODUCTION

Many systems, including online news delivery, stock quote report dissemination and weather channels, have been or can be modeled as Content-Based Publish-Subscribe (CBPS) systems. Full decoupling of the involved parties, that is, *Content Publishers* (Pubs), *Content Brokers* (Brokers) and *Subscribers* (Subs), in time, space, and synchronization has been the key [13] to seamlessly scale these systems on demand. Hence, CBPS systems have the huge potential to be enabled over cloud computing infrastructures. In a CBPS system, each Sub selectively subscribes to some Brokers to receive different messages. In the most common setting, when Pubs publish messages to some Brokers, these Brokers, in turn, selectively distribute these messages to other Brokers and finally to Subs based on their *subscriptions*, that is, what they subscribed to. These systems, in general, follow a *push based* dissemination approach, that is, whenever new messages arrive, Brokers selectively distribute the messages to Subs. Figure 1 shows an example CBPS system.

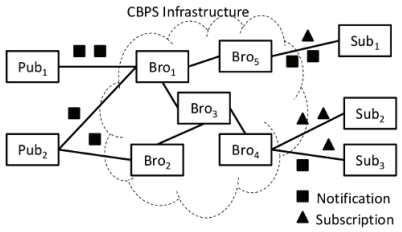

Figure 1: An example CBPS system

It is not feasible to have a private Broker network for each CBPS system and most CBPS systems utilize third-party Broker networks which may not be trusted for the confidentiality of the content flowing through them. Because content represents the critical resource in many CBPS systems, its confidentiality from third-party Brokers is important. Consider the popular example of publishing stock market quotes where Subs pay Pub, that is the stock exchange, either for

the types of quotes they wish to receive or per usage basis. In such a domain, whenever a new stock quote, referred to in general as a *notification*, is published, Brokers selectively send such a notification only to authorized Subs. Confidentiality is important here because Pubs want to make sure that only paying customers have access to the quotes. We say that a CBPS system provides *publication confidentiality* if Brokers can neither identify the content of the messages published by Pubs nor infer the distribution of *attribute values* of the message [1]. For the stock quote example, in the absence of *publication confidentiality*, Brokers may collect stock quotes, re-sell to others, and/or sell derived market data without any economic incentive to Pubs.

At the same time, the privacy of subscribers is also crucial for many reasons, like business confidentiality or personal privacy. We say that a CBPS system provides *subscription privacy* if Brokers can neither identify what subscriptions Subs made nor relate a set of subscriptions to a specific Sub. Consider again the stock quote example. Suppose for example that Sub subscribes to some Brokers for receiving stock quotes characterized by certain attribute values (e.g. bid price < 2438, $1000 <$ bid size < 2000, symbol = "MSFT", etc.). In the absence of *subscription privacy*, such a subscription can reveal the business strategy of Sub. Further, Brokers may profile *subscriptions* of each Sub and sell them to third parties.

Current trends in cloud computing technologies are further pushing brokering functions for content distribution to third-party providers. While such a strategy provides economies of scale, it increases the risk of breaches in publication confidentiality and subscription privacy. Breaches may result from malicious insiders or from platforms that are poorly configured and managed, and that do not have in place proper security techniques. It is thus essential that effective and efficient techniques for publication confidentiality and subscription privacy be devised to allow parties involved in the production and distribution of contents to take full advantages from those emerging computing infrastructures.

Privacy and confidentiality issues in CBPS have long been identified [26], but little progress has been made to address these issues in a holistic manner. Most of prior work on data confidentiality techniques in the context of CBPS systems is based on the assumption that Brokers are trusted with respect to the privacy of the subscriptions by Subs [3, 24, 18]. However, when such an assumption does not hold, both publication confidentiality and subscription privacy are at risk; in the absence of subscription privacy, subscriptions are available in clear text to Brokers. Brokers can infer the content of the notifications by comparing and matching notifications with subscriptions since CBPS systems must allow them to make such decisions to route notifications. As more subscriptions become available to Brokers, the inference is likely to be more accurate. It should also be noted that the above approaches restrict Brokers' ability to make routing decisions based on the content of the messages and thus fail to provide a CBPS system as expressive as a CBPS system that do not address security or privacy issues. Approaches have also been proposed to assure confidentiality/privacy in the presence of untrusted third-party Brokers. These approaches however suffer from one or two major limitations [21, 25, 17, 9]: inaccurate content delivery, because of the limited ability of Brokers to make routing decisions based on content; weak security protocols; lack of privacy guarantees. For example, some of these approaches are prone to false positives, that is, sending irrelevant content to Subs.

In this paper, we propose a novel cryptographic approach that addresses those shortcomings in CBPS systems. To the best of our knowledge, no existing cryptographic solution is able to protect both publication confidentiality and subscription privacy in CBPS systems that address the above shortcomings. A key design goal of our privacy-preserving approach is to design a system which is as expressive as a system that does not consider privacy or security issues. We implement our scheme on top of a popular CBPS system, SIENA [8], and provide several experimental results in order to show our approach is practical.
In summary, our CBPS system exhibits the following properties:

- Notifications and subscriptions are randomized and hidden from Brokers and secure under chosen-ciphertext attacks.
- Both publication confidentiality and subscription privacy are assured as Brokers are able to make routing decisions without decrypting subscriptions and notifications. It is the first system to achieve these properties without sharing keys with Brokers or Subs.
- It supports any type of subscription queries including equality, inequality and range queries at Brokers.
- The computational cost at Brokers are minimized by judiciously distributing the work among Pubs and Subs.

The paper is organized as follows. Section 2 overviews the CBPS model and the protocols supported by our system. Section 3 provides some background knowledge about the main cryptographic primitives used. Section 4 provides a detailed description of the proposed protocols. Section 5 reports experimental results for the main protocols as well as the system developed on top of SIENA using the main protocols. Section 6 discusses related work. Section 7 concludes the paper and outlines future work.

2. OVERVIEW

In this section we give an overview of our proposed scheme by showing the interactions between Pubs, Subs and Brokers, and the trust model. Unless otherwise stated, we describe our approach for one Pub, mainly for brevity. However, our approach can be trivially applied to a system with any number of Pubs. In practice, all the parties in a CBPS system are software programs that act on behalf of real entities like actual organizations or end users, and therefore many of the operations of the protocols we propose are performed transparently to real entities.

Each *notification* is characterized by a set of Attribute-Value Pairs (AVPs). It consists of two parts: the actual message in the encrypted form, which we call the *payload message*, and a set of *blinded AVPs* derived from the payload message. As mentioned in Section 1, a payload message also consists of a set of AVPs. In a blinded AVP, the value is blinded, but the attribute name remains in clear text. The blinding encrypts the value in a special way such that it is computationally infeasible to obtain the value from the

[1]We assume that a message consists of a set of attribute-value pairs.

blinded values, and that the blinded values are secure under chosen-ciphertext attacks. The blinded AVPs are placed in the header and the payload message is in the body of the notification. There is a one-to-one mapping between the AVPs in the payload message and the blinded AVPs. Depending on the representation, each attribute name and its corresponding value may be interpreted differently.

In an XML-like syntax, a notification has the following format:

```
<notification>
    <header> -- blinded AVPs -- </header>
    <body> -- enc. payload message -- </body>
</notification>
```

Depending on the representation, each attribute name and its corresponding value may be interpreted differently. For example, the payload could be in a simple property-value format or a complex XML format. If the payload is in XML, attribute names could be the XPaths and values could be the immediate child nodes of XPaths. We use the latter for the examples.

A *subscription* specifies a condition on one of the attributes [2] of the AVPs associated with the notifications. It is an expression of the form ($attr$, $bval_1$, $bval_2$, $bval_3$, op) where $attr$ is the name of the attribute, $bval_1, bval_2, bval_3$ are the blinded values derived from the actual content v and its additive inverse,[3] and op is a comparison operator, either $\geq$ or $<$. All the other comparison operators are derived from op. Note that our approach supports a wide array of conditions including range queries for numerical attributes and keyword queries for numerical and string attributes.

EXAMPLE 1. *In the stock market quote dissemination system, a payload message, that is, a quote, looks like:*

```
<q>
  <symbol>MSFT</symbol>
    <bid>
      <price>2328</price>
      <size>10000</size>
     ...
   </bid>
   <offer>
     <price>2355</price>
     <size>5000</size>
     ...
   </offer>
</q>
```

The set of AVPs, as a collection of pairs,

$$\left\{ \begin{array}{ll} (\text{"/q/symbol"}, \text{"MSFT"}), & (\text{"/q/bid/price"}, 2328), \\ (\text{"/q/bid/size"}, 10000), & (\text{"/q/offer/price"}, 2355), \\ (\text{"/q/offer/size"}, 5000) & \end{array} \right\}$$

from the payload message is blinded and placed in the header of the notification. The notification for the above quote includes these blinded values and the encrypted quote.

2.1 Interactions

We now present an overview of the protocols proposed in our CBPS system. The motivation behind constructing a set of protocols is that they can easily be implemented on top an existing CBPS infrastructure in order to satisfy privacy and security requirements. In summary, `Initialize` protocol initializes the system parameters. `Register` protocol registers Subs with Pubs. `Subscribe` protocol subscribes Subs to Brokers. `Publish` protocol publishes notifications from Pubs to Brokers. `Match` protocol matches notifications with subscriptions at Brokers. `Cover` protocol finds relationships among subscriptions at Brokers. An important property of the two most frequently used protocols, `Match` and `Cover`, is that they are non-interactive. The following gives more details of each protocol.

`Initialize`:
There is a set of system defined public parameters that all Pubs, Brokers and Subs use. In addition to these parameters, Pubs also generate some public and private parameters that are used for subsequent protocols and publish the public parameters. If there are several Pubs, each Pub generates its own public and private parameters.

`Register`:
Subs register themselves with the Pub to obtain a *private key* and *access tokens*. An *access token* includes Sub's *identity* (id) and allows a Sub to subsequently authenticate itself to the Broker from which it intends to request notifications. An *identity* is a pseudonym that uniquely identifies a Sub in the system. A *private key* allows a Sub to decrypt the payload of notifications.

`Subscribe`:
In order to assure confidentiality and privacy, unlike in a typical CBPS system, Subs need to perform an additional communication step with Pub to get the subscription blinded before submitting the subscription to Broker [4].

After authenticating themselves using access tokens to Pubs, Subs receive the content in their subscriptions blinded by the corresponding Pubs. In this step, Subs perform as much computation as it can before sending the subscriptions to Pub so that the overhead on Pubs is minimized. Further, this overhead on Pubs is negligible as subscriptions are fairly stable and the rate of subscriptions is usually way less than that of notifications in a typical CBPS system. Once this step is done, Subs authenticate themselves to Brokers without revealing their identities and present these blinded subscriptions to Brokers. These subscriptions are blinded in such a way that Brokers do not learn the actual subscription criteria, that is, Brokers cannot decrypt the blinded values. However, they can perform `Match` (or `Filter`),[5] and `Cover` protocols based on the blinded subscriptions. Furthermore, no two subscriptions for the same value are distinguishable by Brokers. In order to prevent Brokers from linking different subscriptions from the same Sub, Subs may request for multiple access tokens such that all these access tokens have the same identity but are indistinguishable. For each subscription, Subs may present these different valid access tokens so that Subs' identities are further protected from Brokers.

`Publish`:
Using the counterparts of the secret values used to blind subscriptions, Pubs blind the notifications and publish them to some Brokers. A blinded notification has a set of blinded AVPs and an encrypted payload message. These notifications are blinded in such a way that Brokers do not learn

[2]Note that our approach can easily be extended to subscriptions having multiple attributes.

[3]The additive inverse of a number $v \in \mathbb{Z}_m$ can be represented by the number $m - v$.

[4]Instead of Pub, a trusted third party may be utilized to blind subscriptions in order to reduce the load on Pub.

[5]We use the terms `Match` and `Filter` interchangeably.

actual values in the messages, but can perform `Match` and `Cover` protocols based on the subscriptions. Further, no two notifications for the same content are distinguishable by Brokers.

`Match`:
For each notification from Pubs, Brokers compare it with Subs' subscriptions. If there is a match, that is, the subscription satisfies the notification, Brokers forward the notification to the correct Subs. The outcome of the `Match` protocol allows Brokers to learn neither the notification nor the publication values. It also prevents Brokers from learning the distribution of the values.

`Cover`:
For each subscription received from Subs, Brokers check if *covering* relationship holds with the existing subscriptions. A subscription S_1 covers another subscription S_2 if all notifications that match S_2 also match S_1. Finding covering relationships among subscriptions allows to reduce the size of the subscription tables maintained by each Broker, and hence improves the efficiency of matching. Like the `Match` protocol, the outcome of the `Cover` protocol does not allow the Brokers to learn the subscription values nor their distribution.

2.2 Trust model

In the system design, we consider threats and assumptions from the point of view of Pubs and Subs with respect to third-party Brokers. We assume that Brokers are honest but curious; they perform PS protocols correctly, but curious to know what Pubs publish and Subs consume. In other words, they are trusted for these PS protocols but not for the content in the notifications and subscriptions nor for the privacy of Subs if they make one or more subscription requests. Further, Brokers may collude. Pubs are trusted to maintain the privacy of Subs. However, our approach can be easily modified to relax this trust assumption. Pubs are also trusted to correctly perform PS protocols and not to collude with any other parties.

3. BACKGROUND

Some of the mathematical notions and the cryptographic building blocks which inspired our approach are described below.

3.1 Pedersen commitment

A cryptographic "commitment" is a piece of information that allows one to commit to a value while keeping it hidden, and preserving the ability to reveal the value at a later time. The *Pedersen commitment* [20] is an unconditionally hiding and computationally binding commitment scheme which is based on the intractability of the discrete logarithm problem.

Pedersen Commitment

Setup A trusted third party T chooses a multiplicatively written finite cyclic group G of large prime order $\mathfrak{p}$ so that the computational Diffie-Hellman problem is hard in G.[6] T chooses two generators g and h of G such that it is hard to find the discrete logarithm of h with respect to g, i.e., an integer x such that $h = g^x$. It is not required that T know the secret number x. T publishes $(G, \mathfrak{p}, g, h)$ as the system parameters.

Commit The domain of committed values is the finite field $\mathbb{F}_\mathfrak{p}$ of $\mathfrak{p}$ elements, which can be represented as the set of integers $\mathbb{F}_\mathfrak{p} = \{0, 1, \ldots, \mathfrak{p}-1\}$. For a party U to commit a value $\alpha \in \mathbb{F}_\mathfrak{p}$, U chooses $\beta \in \mathbb{F}_\mathfrak{p}$ at random, and computes the commitment $c = g^\alpha h^\beta \in G$.

Open U shows the values α and β to open a commitment c. The verifier checks whether $c = g^\alpha h^\beta$.

[6]For a multiplicatively written cyclic group G of order q, with a generator $g \in G$, the *Computational Diffie-Hellman problem (CDH)* is the following problem: Given g^a and g^b for randomly-chosen secret $a, b \in \{0, \ldots, q-1\}$, compute g^{ab}.

3.2 Paillier homomorphic cryptosystem

The *Paillier homomorphic cryptosystem* is a public key cryptosystem by Paillier [19] based on the "Composite Residuosity assumption (CRA)." The Paillier cryptosystem is homomorphic in that, by using public key, the encryption of the sum $m_1 + m_2$ of two messages m_1 and m_2 can be computed from the encryption of m_1 and m_2. Our approach and protocols are inspired by how the Paillier cryptosystem works. Hence, we provide some internal details of the cryptosystem below so that readers can follow the rest of the paper.

Key generation

Set $n = pq$, where p and q are two large prime numbers. Set $\lambda = \mathrm{lcm}(p-1, q-1)$, i.e., the least common multiple of $p-1$ and $q-1$. Randomly select a base $g \in \mathbb{Z}/(n^2)^\times$ such that the order of g_p is a multiple of n. Such a g_p can be efficiently found by randomly choosing $g_p \in \mathbb{Z}/(n^2)^\times$, then verifying that

$$\gcd(L(g_p^\lambda \pmod{n^2}), n)) = 1, \text{ where } L(u) = (u-1)/n \tag{1}$$

for $u \in S_n = \{u < n^2 | u = 1 \pmod{n}\}$. In this case, set $\mu = \left(L(g_p^\lambda \pmod{n^2})\right)^{-1} \pmod{n}$. The public encryption key is a pair (n, g_p). The private decryption key is (λ, μ), or equivalently (p, q, μ).

Encryption $E(m, r)$

Given plaintext $m \in \{0, 1, \ldots, n-1\}$, select a random $r \in \{1, 2, \ldots, n-1\}$, and encrypt m as $E(m, r) = g_p^m \cdot r^n \pmod{n^2}$. When the value of r is not important to the context, we sometimes simply write a short-hand $E(m)$ instead of $E(m, r)$ for the Paillier ciphertext of m.

Decryption $D(c)$

Given ciphertext $c \in \mathbb{Z}/(n^2)^\times$, decrypt c as

$$D(c) = L(c^\lambda \pmod{n^2}) \cdot \mu \pmod{n}. \tag{2}$$

More specifically, the homomorphic properties of Paillier cryptosystem are:

$$D(E(m_1, r_1)E(m_2, r_2) \pmod{n^2}) = m_1 + m_2 \pmod{n},$$
$$D(g^{m_2}E(m_1, r_1) \pmod{n^2}) = m_1 + m_2 \pmod{n},$$
$$D(E(m_1, r_1)^k \pmod{n^2}) = km_1 \pmod{n}.$$

Also note that the Paillier cryptosystem described above is semantically secure against chosen-plaintext attacks (IND-CPA).

In the construction of our CBPS system, the Paillier homomorphic cryptosystem is used in a way that public and private keys are judiciously distributed among Pubs, Subs, and Brokers such that the confidentiality and privacy are assured based on homomorphic encryption. A detailed description of the construction is presented in Section 4.

4. PROPOSED SCHEME

In this section, we provide a detailed description of the privacy preserving CBPS system we propose. As introduced in Section 2, the system consists of 6 protocols: 1) `Initialize`, 2) `Register`, 3) `Subscribe`, 4) `Publish`, 5) `Match`, and 6) `Cover`.

4.1 Initialize

A trusted party, which could be one of the Pubs, runs a Pedersen commitment setup algorithm [20] to generate system wide parameters $(G, \mathfrak{p}, g, h)$. These parameters have the same meaning and purpose as mentioned in Section 3. The same party also runs a key generation algorithm similar to Paillier [19] to generate the parameters $(n, p, q, g_p, \lambda, \mu)$. Only Pubs know the parameters (p, q, λ). The parameters (n, g_p, μ) are public. Note that unlike in Paillier, μ is public in our scheme. The system parameter l is the upper bound on the number of bits required to represent any data values published, and we refer to it as *domain size*. For example, if an attribute can take values from 0 up to 500 $(< 2^9)$, l should be at least 9 bits long. For reasons that will soon become clear in this section we choose l such that $2^{2l} \ll n$.[7] In addition to these parameters, each Pub has a key pair (K_{pub}, K_{pri}) where K_{pri} is the private key used to sign access tokens of Subs and K_{pub} is the public key used by Brokers to verify authenticity and integrity of them. Each Pub also has a symmetric key K, which it shares only with Subs and is used to encrypt the payload messages. Each Pub computes two pairs of secret values (e_m, d_m) and (e_c, d_c) such that $e_m + d_m \equiv 0 \pmod{\phi(n^2)}$, and $e_c + d_c \equiv 0 \pmod{\phi(n^2)}$, where $\phi(\cdot)$ is Euler's totient function and $e_m \neq e_c$. Note that we have $g^{e_m} g^{d_m} \equiv g^{e_c} g^{d_c} \equiv 1 \pmod{n^2}$. Pub uses e_m to blind Paillier encrypted notifications and d_m, d_c, e_c to blind Paillier encrypted subscriptions.[8] Let s be the largest number $\in \mathbb{Z}$ such that $2^s < n$ and $u \in \mathbb{Z}$ such that $l < u < s - 1$. Finally, each Pub chooses two secret random values $r_m, r_c \in \mathbb{Z}$ such that $1 < r_m, r_c < 2^{u-l}$ and $r_m \neq r_c$. These values are used to prevent Brokers from learning the distribution of the difference of the values that are being matched. In summary, $(G, \mathfrak{p}, g, h, n, g_p, \mu, K_{pub})$ are the public parameters that all the parties know, $(p, q, \lambda, K_{pri}, r_m, r_c, (e_m, d_m), (e_c, d_c))$ are private parameters of Pubs. Note that in a practical implementation, most of these parameters can be auto-generated by a computer program which usually only requires Pub to pre-determine l depending on the domain of the content of notifications.

4.2 Register

As shown in Figure 2, each Sub registers itself with Pub by presenting an id (identity), a pseudonym uniquely identifying Sub. In a real-world system, registration may involve Subs presenting other credentials and/or making payment. Upon successful registration, Pub sends K, the symmetric key, to Sub.[9] During this protocol, each Sub also obtains its initial access token, a Pedersen commitment signed by Pub.

An access token allows Sub to authenticate itself to Broker from which it intends to request notifications as well as to create additional access tokens in consultation with Pub. To create the first access token, Sub encodes its id as an element $\langle \mathsf{id} \rangle \in \mathbb{F}_{\mathfrak{p}}$, chooses a random $a \in \mathbb{F}_{\mathfrak{p}}$, and sends the commitment $com(\langle \mathsf{id} \rangle) = g^{\langle \mathsf{id} \rangle} h^a$ and the values $(\langle \mathsf{id} \rangle, a)$. The Pub signs $com(\langle \mathsf{id} \rangle)$ and sends the digital signature $K_{pri}(com(\langle \mathsf{id} \rangle))$ back to the Sub.

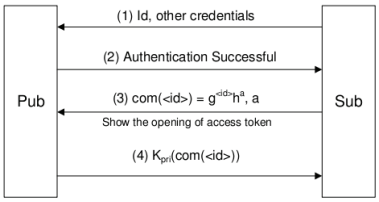

Figure 2: Sub registering with Pub

4.3 Subscribe

During this protocol, Subs inform their interests to Brokers as subscriptions. Before subscribing to messages, as Figure 3 illustrates, Subs must authenticate themselves to Brokers. Sub gives a zero-knowledge proof of knowledge (ZKPK) of the ability to open the commitment $com(\langle \mathsf{id} \rangle)$ signed by Pub:

$$\text{ZKPK}\{(\langle \mathsf{id} \rangle, a) : com(\langle \mathsf{id} \rangle) = g^{\langle \mathsf{id} \rangle} h^a\}$$

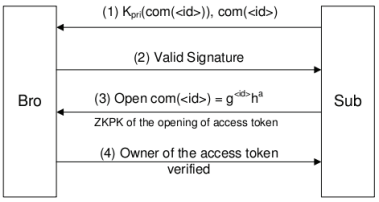

Figure 3: Sub authenticating itself to Broker

Notice that the ZKPK of the commitment opening does not reveal the identity of Sub. Further, Sub may use different access tokens by having different random a values for different subscriptions to prevent Brokers from linking its subscriptions to one access token [10] [11].

[7] We use notation $a \ll b$ to denote that "a is sufficiently smaller than b."

[8] The "blind" operation will be introduced in Section 4.3.

[9] We use a symmetric encryption algorithm in the presentation. In practice, Pubs and Subs can choose any encryption scheme, symmetric or not, to hide the payload messages in transmission. In our extended version, we use a fine-grained encryption technique based on broadcast group key management in order to selectively and efficiently encrypt payload messages [22]. Attribute based encryption or proxy re-encryption, as mentioned in Section 6, could be a possible choice as well.

[10] One may use a randomized signature scheme on a committed value [7] to achieve the same objective at the expense of additional computation cost.

[11] Our scheme only provides application level privacy, but

If the ZKPK is successful, Sub may submit one or more subscriptions. Recall that subscriptions are blinded by Pub before sending to Broker. The subscription "blinding" functions, $bval_m$, $bval_{c_1}$, $bval_{c_2}$ are defined as follows:
Let v be the original subscription.

$$E(v) = g_p^v \cdot r_1^n \pmod{n^2}$$
$$bval_m(E(-v)) = g^{d_m} \cdot (E(-v))^{r_m \lambda} \pmod{n^2} \quad (3)$$
$$bval_{c_1}(E(-v)) = g^{d_c} \cdot (E(-v))^{r_c \lambda} \pmod{n^2} \quad (4)$$
$$bval_{c_2}(E(v)) = g^{e_c} \cdot (E(v))^{r_c \lambda} \cdot (E(r))^{\lambda} \pmod{n^2} \quad (5)$$

where $d_m, e_m, r_m, d_c, e_c, r_c$ are generated during `Initialize`, r in Formula 5 is a random number such that $r \leq min\{r_c, 2^{(s-1-u)}\}$.

Sub sends $E(v)$ and $E(-v)$, where v is the original subscription for the attribute $attr$, to Pub. Pub sends back the blinded subscription to Sub and Sub sends the tuple $\langle attr, bval_{c_1}(E(-v)), bval_{c_2}(E(v)), bval_m(E(-v)), op\rangle$ to Broker. The first two blinded values in the subscription are used by Broker for `Cover` protocol and the third one for `Match` protocol. Note that Sub performs these encryptions to reduced the load on Pubs. It should also be noted that equality filters in our protocols are treated as range filters preventing Brokers from distinguishing equality filters from range filters. For example, in order to subscribe for $v = 5$, Sub subscriber for a range filter where $v \leq 5$ and $v > 4$. Except for range filters, each subscription from the same Sub are treated as disjunctive conditions.

Example 2. *Sub wants to get all the notifications with bid price less than 22. The subscription has the format ("/quote/bid/price", 346213, 152311, 453280, <) where the second and third parameters are the blind values of* 22 *and* -22*, respectively, for* `Cover` *protocol to use, and the fourth is the blinded value of* -22 *for* `Match` *protocol to use.*

4.4 Publish

Using e_m, the counterpart of d_m which is used to blind subscriptions for `Match` protocol, and other private parameters, Pubs blind the notifications using the function $bval_n$ as defined below.
Let x be one value in the notification.

$$bval_n(x) = g^{e_m} \cdot (E(x))^{r_m \lambda} \cdot E(r)^{\lambda} \pmod{n^2}$$
$$= g^{e_m} \cdot E((r_m x + r)\lambda) \pmod{n^2},$$

where e_m and r_m are generated during `Initialize`, r is selected uniformly at random such that $r \leq min\{r_m, 2^{(s-1-u)}\}$.

Pubs publish the blinded notifications to Brokers. A notification has a set of blinded AVPs and an encrypted payload message. For an illustration purpose, let us assume these AVPs are numbered from 1 to t, where t is the number of attributes of the payload message M being considered. The blinded notification looks like $(\langle attr_1, bval_n(x_1)\rangle, \ldots, \langle attr_t, bval_n(x_t)\rangle)$, where $attr_i$ and x_i are the i^{th} attribute name and value respectively.

not network level privacy. For example, it does not hide IP addresses. In order to provide network level privacy/anonymity, one needs to utilize other orthogonal techniques such as Tor [12]

Table 1: Matching Decision

diff	Decision
$< n/2$	$x \geq v$
$> n/2$	$x < v$

4.5 Match

For each notification from Pub, Broker compares it with Subs' subscriptions to make routing decisions. We explain the `Match` operation for one attribute in the message, but it can be naturally extended to perform on multiple attributes. If at least one of the attributes in the message matches, we say that the subscription matches the notification, and in this case Broker forwards the notification to the corresponding Subs. For range filters, the conjunction of two corresponding `Match` operations is taken.

Let the blinded values be $bval_n(x)$ and $bval_m(E(-v))$ that Broker has received from Pub and Sub, respectively, for an attribute $attr$ with subscription value being v and notification value being x. Broker computes the following value *diff* and then makes the matching decision based on Table 1.

$$diff = L(bval_n(x) \cdot bval_m(E(-v)) \pmod{n^2}) \cdot \mu \pmod{n},$$

,

where L, μ are public parameters derived from Paillier.

Before we show that the above computation gives a *diff* equal to $r_m \cdot (x - v) + r$, we describe how `Match` protocol gives the correct matching decision while outputting a (controlled) random *diff* value to Broker. Recall that in `Initialize`, the domain of the input values is set to $0 \sim 2^l$. Therefore, $0 \leq x, v \leq 2^l$. Notice that the difference of any two values x and v is either between $0 \sim 2^l$ if the difference is positive, or between $(n - 2^l) \sim n$ if the difference is negative. Also, notice that the range $2^l \sim (n - 2^l)$ is not utilized. In order to randomize the difference, we take advantage of this unused range and multiply the actual difference with a random secret value r_m and add another random value r both selected by Pub. The idea behind r_m and r are to first expand $0 \sim 2^l$ range to $0 \sim 2^u$ and $(n - 2^l) \sim n$ to $n - 2^s \sim n - n_m$, and then expand them to $0 \sim n/2$ and $n/2 \sim n$ respectively. Thus the difference is randomized, yet it allows Broker to make correct matching decisions without resulting in false positives or negatives.

During `Match` protocol, Broker does not learn the content under comparison. This is achieved due to the fact that without knowing λ, Broker cannot perform decryption freely, but is forced to engage into the protocol described below. Not knowing the values r_m and r, Broker does not learn the exact difference of the two values under comparison as well.

The following shows the correctness of *diff*. Let

$$y = bval_n(x) \cdot bval_m(E(-v)) \pmod{n^2}.$$

$$
\begin{aligned}
y &= g^{e_m} \cdot (E((r_m x + r)\lambda) \cdot g^{d_m} \cdot (E(-v))^{r_m \lambda} \\
&\quad (\text{mod } n^2) \\
&= g^{e_m + d_m} \cdot \{E(r_m x + r)) \cdot E(-r_m v)\}^{\lambda} \quad (\text{mod } n^2) \\
&= (E(r_m(x - v) + r))^{\lambda} \quad (\text{mod } n^2) \\
\mathit{diff} &= L(y) \cdot \mu \quad (\text{mod } n) = r_m(x - v) + r. \qquad (6)
\end{aligned}
$$

4.6 Cover

Subscriptions are categorized into groups based on the covering relationships so that Brokers can perform `Match` protocol efficiently. For each subscription received from Subs, Brokers check if covering relationship holds within the existing subscriptions. If it exists, they add the new subscription to the group with the covering subscription, otherwise a new group is created for the new subscription.

Notice that we have not used the blinded values $bval_{c_1}(E(-v))$ and $bval_{c_2}(E(v))$ in subscriptions yet. These two values are used in the `Cover` protocol. In what follows, we explain how the `Cover` protocol works.

Let S_1 and S_2 be two subscriptions for the same *attr* and compatible *op*. Two *op*'s are compatible if either both of them are of the same type. $bval_{c_1}(E(v_1))$ and $bval_{c_2}(E(-v_1))$ refer to the so far unused blinded values of v_1 and of its additive inverse, respectively, of the subscription S_1. The blinded values $bval_{c_1}(E(v_2))$ and $bval_{c_2}(E(-v_2))$ have similar interpretations.

Broker computes one of the following two values in order to decide the covering relationship.

$$
\begin{aligned}
\mathit{diff}_1 &= L(bval_{c_2}(E(v_1)) \cdot bval_{c_1}(E(-v_2)) \\
&\quad (\text{mod } n^2)) \cdot \mu \quad (\text{mod } n) \\
\mathit{diff}_2 &= L(bval_{c_2}(E(v_2)) \cdot bval_{c_1}(E(-v_1)) \\
&\quad (\text{mod } n^2)) \cdot \mu \quad (\text{mod } n) \qquad (7)
\end{aligned}
$$

diff_1 and diff_2 give results $r_c \cdot (v_1 - v_2) + r$ and $r_c \cdot (v_2 - v_1) + r'$ respectively, where r, r' are random numbers. Broker uses the same matching Table 1 that is used for making matching decision to make the covering decision. The covering decision for range filters is performed in a similar way, but we omit the details due to lack of space. Similar to `Match`, Brokers do not learn the actual subscription values.

4.7 The Distribution of Load

We now briefly explain the rationale behind the distribution of work load among Pubs, Subs and Brokers. If there are $O(N)$ notifications and $O(S)$ subscriptions, in the worst case, Broker needs to perform $O(NS)$ `Match` protocols. Thus, Brokers have to perform significantly more work compared to Pubs and Subs in a typical CBPS system. This is one of the key reasons why the performance of Brokers degrades as the number of notifications and/or subscriptions in the system increases. By optimizing for the frequent case, one can achieve a significant overall system improvement. We followed this well-known design principle to redistribute the load on Brokers partly to Pubs and Subs. Notice that there are no exponentiation operations in both `Match` and `Cover` protocols. Hence, these protocols can be performed very efficiently. This is made possible at the cost of extra work at Pubs and Subs. Since the protocols at Pubs and Subs are executed less frequently compared to those at Brokers, our distribution leads to a better overall system performance. The experimental results show that the protocols at Brokers are very efficient and those at Pubs and Subs also run fast.

5. EXPERIMENTAL RESULTS

In this section, we present experimental results for various operations and the two main protocols, `Match` and `Cover`, in our system as well as our privacy preserving CBPS (PP-CBPS) system itself which extends an enhanced SIENA system by implementing privacy preserving matching and covering using our protocols. For the protocol experiments, we have built a prototype system in Java that incorporates our techniques for privacy preserving `Match` and `Cover` protocols as described in Section 4.

The experiments are performed on an Intel® Core™ 2 Duo CPU T9300 2.50GHz machine running GNU/Linux kernel version 2.6.27 with 4 Gbytes memory. We utilize only one processor for computation. The code is built with Java version 1.6.0. along with Bouncy Castle lightweight APIs [6] for most cryptographic operations including the symmetric-key encryption. The Paillier cryptosystem is implemented as in the paper [19], except that we modified the algorithms to fit our scheme. We first look at the experiments mainly on the two important protocols, `Match` and `Cover`, and then describe the system experiments performed on PP-CBPS system.

5.1 Protocol experiments

In our experiments we vary values of n in Paillier cryptosystem and the domain size l, and fix the parameters for Pedersen commitment generation, digital signature generation/verification, zero-knowledge proof of knowledge protocol, and symmetric key encryption/decryption. In all our experiments we only measure computational cost, and assume the communication cost to be negligible. All data obtained by our experiments correspond to the average time taken over 1000 executions of the protocols with varying values for the bit length of n in the Paillier cryptosystem and the domain size l. Appendix B shows the computation time for the general operations.

In the experiment shown in Figure 4, we vary the bit length of n in the Paillier cryptosystem. Figure 4 shows the time to generate blinded subscriptions and notifications whose values are less than 2^l where l, the domain size, is fixed at 100, a reasonably large value. The time to generate blinded values increases as the bit length of n increases, but even for large bit lengths, it takes only a few milliseconds. The time required to blind subscription is split into two tasks with the Sub performing the encryption and the Pub performing the blinding, but to blind notifications, the Pub performs both operations as one task. We remark that the overall computational cost can be reduced by employing well-known caching techniques.

We measure in our experiment the performance impact on blinding when l, the domain size, is changed. We fix n to be of length 1024 bits and measure the time to blind subscriptions and notifications for $l = 10, 20, \cdots, 100$. As shown in Figure 5, the domain size does not significantly affect the performance of the blinding operations. Further, as indicated by both Figure 4 and Figure 5, the time for either component of the subscription blinding is less than that for notification blinding. Since for each subscription, the overhead at the Pub is less compared to the time required to

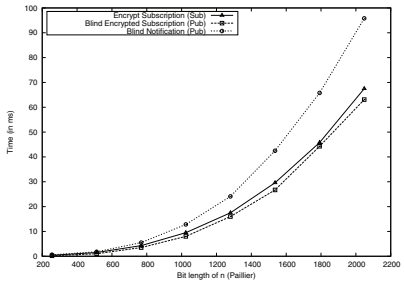

Figure 4: Time to blind subscriptions and notifications for different bit lengths of n

blind a notification, our decision to blind part of the subscription at the Pub is comparable to blinding additional notifications.

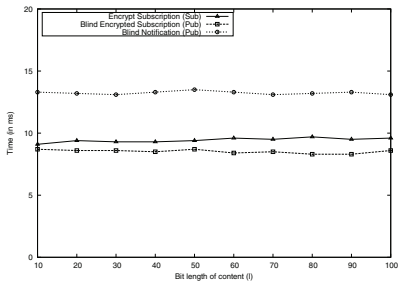

Figure 5: Time to blind subscriptions and notifications for different l

In a CBPS, `Match` is the most executed protocol. Hence, it should be very efficient so as not to overload Brokers. For each `Subscribe` protocol, Brokers may need to invoke the `Cover` protocol and, therefore, we want to have a very efficient `Cover` protocol as well. In the following two experiments, we observe the time to perform these protocols.

Figure 6 shows the execution time of `Match` and `Cover` protocols as the bit length of n in the Paillier cryptosystem is changed while the domain size l is fixed at 100 bits. The time for both protocols increases approximately linearly with the bit length of n. Note that they take only a fraction of a millisecond (less than 100 microseconds) even for large bit lengths of n. This indicates that our `Match` and `Cover` protocols are very efficient for large bit lengths of n.

Figure 7 shows the time to execute `Match` and `Cover` protocols as the domain size l is changed while the bit length of n is fixed at 1024. Similar to the blind computations, computational times remain largely unchanged for different l values.

An observation made through all our protocol experiments is that the domain size l does not significantly affect the computational time of the key protocols `Publish`, `Subscribe`, `Match` and `Cover`, but the bit length n of the Paillier cryptosystem does. However, even for large bit lengths of n, our protocols take only a few microseconds or milliseconds and thus they are very efficient and practical.

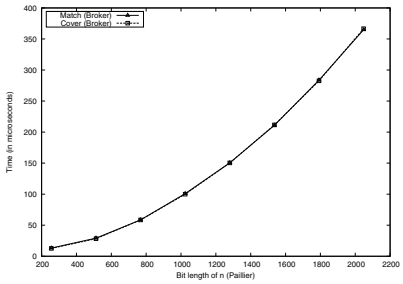

Figure 6: Time to perform match and cover for different bit lengths of n

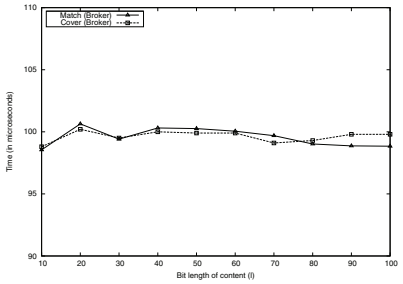

Figure 7: Time to perform match and cover for different l

5.2 System experiments

In this section, we provide the experiments performed on our PP-CBPS system. PP-CBPS is constructed by a freely available popular wide-area event notification implementation SIENA. SIENA provides a pluggable-architecture that allows to incorporate our protocols to provide `Match` and `Cover` operations. All the testing data are generated uniformly at random. In all the experiments, the average time to match a notification with a subscription is measured where 1000 notifications are generated each time and the system groups the subscriptions according to the covering relationships at the time of subscription. It should be noted that the matching time does not include the time to create notifications and subscriptions which is measured in our protocol experiments in Section 5.1.

Figure 8 shows the time to perform equality filtering in PP-CBPS (secure matching) and SIENA (plain matching) for different number of subscriptions in the system. Notifications and subscriptions are drown uniformly from 10 bit random integers. We use a small domain size to demonstrate the effect of covering on the overall system with and without security. As can be seen, PP-CBPS performs the matching within 10x of that of SIENA and is still quite efficient to match thousands of subscriptions within 10 ms. In both cases, the increase in matching time with the number of subscriptions is sub-linear since the covering operation groups the similar subscriptions together, reducing the number of `Match` protocols needs to be executed.

Figure 9 shows the time to perform equality filtering in PP-CBPS for two different domain sizes, 10 and 25 bits, of notifications and subscriptions for different number of subscriptions in the system. It should be noted that SIENA

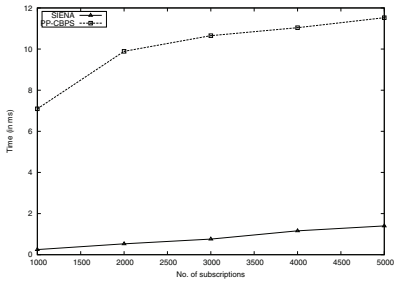

Figure 8: Equality filtering time

currently does not support domain sizes larger than 27 bits, but our protocols can work under much larger domains. As can be seen, the matching is more efficient with smaller domains. This is due to the fact that smaller domains create more covering relationships than larger domains and, hence, less matching protocols need to be executed to match a notification against all the subscriptions. Further, observe that the rate of increase of the overall matching cost decreases as the number of subscriptions increases. This, again, is due to the covering protocol.

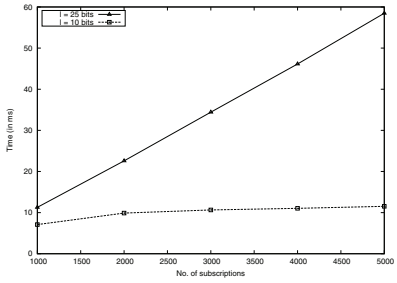

Figure 9: Equality filtering time for different domain sizes

Figure 10 shows the time to perform inequality filtering in PP-CBPS for two different domain sizes, 10 and 25 bits, of notifications and subscriptions for different number of subscriptions in the system. We observe results similar to that of equality filtering in Figure 9. However, notice that the inequality filtering is much more efficient than equality filtering for the same domain size. This is due to the fact that inequality subscriptions create more covering relationships than equality subscriptions requiring much less matching operations.

Even though, according to the protocol experiments in Section 5.1, the time to perform individual `Match` or `Cover` operations remains largely constant for different domain sizes, the overall system performs better with smaller domain sizes. As the domain size is reduced, there is a higher probability of having subscriptions satisfying covering relationships. Hence, the number of matching operations need to be performed reduces considerably leading to a better performance.

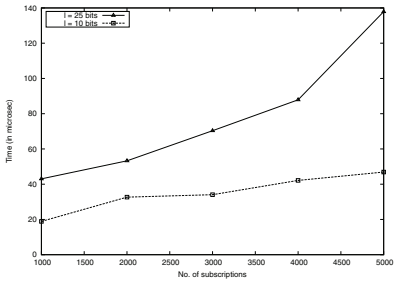

Figure 10: Inequality filtering time for different domain sizes

6. RELATED WORK

In addition to the research work discussed in Section 1, our work is related to research in proxy re-encryption systems [16, 2], searchable encryption [23, 4, 5], secure multiparty computation [14, 11] and private information retrieval [10, 15].

Proxy re-encryption system.

In a proxy re-encryption system one party A delegates its decryption rights to another party B via a third party called a "proxy." More specifically, the proxy transforms a ciphertext computed under party A's public key into a different ciphertext which can be decrypted by party B with B's private key. In such a system neither the proxy nor party B alone can obtain the plaintext. A direct application of the proxy re-encryption system does not solve the problem of CBPS: with the proxy as the Broker, it does not by default have the capability of selectively making content-based routing decisions. However, it might still be possible to use proxy re-encryption as a building block in the construction of a CBPS system for data confidentiality.

Searchable encryption.

Search in encrypted data is a privacy-preserving technique used in the *outsourced storage model* where a user's data are stored on a third-party server and encrypted using the user's public key. The user can use a query in the form of an encrypted token to retrieve relevant data from the server, whereas the server does not learn any more information about the query other than whether the returned data matches the search criteria. There have been efforts to support simple equality queries [23, 4] and more recently complex ones involving conjunctions and disjunctions of range queries [5]. These approaches cannot be applied directly to the CBPS model.

Secure Multiparty Computation (SMC).

SMC allows a set of participants to compute the value of a public function using their private values as input, but without revealing their individual private values to other participants. The problem was initially introduced by Yao.Since then improvements have been proposed to the initial problem [14, 11]. SMC solutions rely on some form of zero-knowledge proof of knowledge (ZKPK) or oblivious transfer protocols which are in general interactive. Interactive protocols are not suitable for the CBPS model. Hence SMC solutions do not work for the CBPS model. Further, these

solutions usually have a higher computational and/or communication cost which may not be acceptable for a CBPS system.

Private Information Retrieval (PIR).

A PIR scheme allows a client to retrieve an item from a database server without revealing which item is retrieved. Approaches of PIR assume either the server is computationally bounded, where the problem reduces to oblivious transfer, or there are multiple non-cooperating servers each having the same copy. Having only two communication parties, PIR schemes are not directly applicable to the Pub-Sub-Broker architecture of the CBPS model. Moreover, similar to SMC solutions, PIR schemes in general have a higher communication complexity which may not be acceptable for a CBPS system.

7. CONCLUSIONS AND FUTURE WORK

We have presented an efficient cryptography-based approach to preserve subscription privacy and publication confidentiality in a CBPS system in which third-party Brokers perform `Match` and `Cover` protocols to make routing decisions for subscriptions without learning the actual content of the notifications published by Pubs and the subscriptions made by Subs. The experimental results on both the protocols and the system, in Section 5 show that our techniques are practical and efficient. We believe that our cryptographic techniques have a broader application of performing privacy-preserving third-party comparisons.

We are currently integrating our privacy preserving protocols to Apahce ActiveMQ [1], a popular open source Java message broker middleware.

Acknowledgments

The work reported in this paper has been partially supported by the MURI award FA9550-08-1-0265 from the Air Force Office of Scientific Research.

8. REFERENCES

[1] Apache. ActiveMQ. `http://activemq.apache.org/`.

[2] G. Ateniese, K. Benson, and S. Hohenberger. Key-private proxy re-encryption. In *RSA '09*, pages 279–294, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] E. Bertino, B. Carminati, E. Ferrari, B. Thuraisingham, and A. Gupta. Selective and authentic third-party distribution of XML documents. *IEEE TKDE*, 16(10):1263–1278, Oct. 2004.

[4] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano. Public-key encryption with keyword search. In *EUROCRYPT '04*, 2004.

[5] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. *Theory of Cryptography*, pages 535–554, May 2007.

[6] Bouncycastle. Bouncy Castle Crypto APIs. `http://www.bouncycastle.org/`.

[7] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO '04*, pages 56–72. Springer-Verlag, 2004.

[8] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM TCS*, 19(3):332–383, 2001.

[9] S. Choi, G. Ghinita, and E. Bertino. A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations. In *DEXA '10*, 2010.

[10] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *FOCS '95*, pages 41–50, Oct 1995.

[11] I. Damgård, M. Geisler, and M. Kroigard. Homomorphic encryption and secure comparison. *Int. J. on App. Crypto.*, 1(1):22–31, 2008.

[12] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Usenix Sec. '04*, 2004.

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

[14] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT '04*, 2004.

[15] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: Single database, computationally-private information retrieval. In *FOCS '97*, pages 364–373, 1997.

[16] T. Matsuo. Proxy re-encryption systems for identity-based encryption. In *PAIRING '07*, pages 247–267, 2007.

[17] K. Minami, A. J. Lee, M. Winslett, and N. Borisov. Secure aggregation in a publish-subscribe system. In *WPES '08*, pages 95–104, New York, NY, USA, 2008. ACM.

[18] M. Nabeel and E. Bertino. Secure delta-publishing of XML content. In *ICDE '08*, pages 1361–1363, April 2008.

[19] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99*, pages 223–238, 1999.

[20] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '92*, pages 129–140, London, UK, 1992. Springer-Verlag.

[21] C. Raiciu and D. S. Rosenblum. Enabling confidentiality in content-based publish/subscribe infrastructures. In *SECURECOMM '06*, pages 1–11, 28 2006-Sept. 1 2006.

[22] N. Shang, M. Nabeel, F. Paci, and E. Bertino. A privacy-preserving approach to policy-based content dissemination. In *ICDE '10*, 2010.

[23] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *SP '00*, page 44, Washington, DC, USA, 2000. IEEE Computer Society.

[24] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services with eventguard. In *CCS '05*, pages 289–298, New York, NY, USA, 2005. ACM.

[25] M. Srivatsa and L. Liu. Secure event dissemination in publish-subscribe networks. In *ICDCS '07*, page 22, Washington, DC, USA, 2007. IEEE Computer Society.

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

APPENDIX

A. SECURITY ANALYSIS

In this section, we briefly analyze the security of the proposed CBPS system. The proposed system is built upon provably secure cryptographic primitives: digital signatures, Pedersen commitment, Schnorr's zero-knowledge proof protocol, and a modified Paillier homomorphic encryption.

A.1 Privacy-preserving subscription

The subscription protocol is privacy preserving in that it supports anonymous credential authentication of the Subs to Brokers. When a Sub subscribes to a Broker, it shows an access token containing a Pedersen commitment of Sub's identity attribute value $\langle \mathsf{id} \rangle$ together with a digital signature from a Pub. The Broker verifies the digital signature using the Pub's public key K_{pub} to make sure that the Pedersen commitment is a valid one approved by the Pub. Due to the unconditional hiding property of the Pedersen commitment scheme, the Broker learns nothing about the value $\langle \mathsf{id} \rangle$ from $com(\langle \mathsf{id} \rangle) = g^{\langle \mathsf{id} \rangle} h^a$. By performing a zero-knowledge proof of knowledge protocol, the Sub can convince the Broker that the Sub knows the values $\langle \mathsf{id} \rangle$ and a, thus has the ability to open the commitment, but prevents the Broker from learning the actual values. Without knowing the values $\langle \mathsf{id} \rangle$ and a, anyone without valid ownership to the access token cannot open the commitment. This provides a mechanism to defend identity theft. In such a way, the combined use of digital signatures and the ZKPK technique realizes a privacy-preserving authentication.

A.2 Privacy-preserving matching and covering

`Match` and `Cover` protocols are privacy preserving in that while Brokers are performing matching and covering operations correctly, they do not learn the actual values in Subs' subscriptions or Pubs' notifications.

To see that `Match` preserves Pub's and Sub's privacy, we look at the underlying scheme. When Sub subscribes, Broker gets a subscription specified with blinded values $bval_{c_1}(E(v))$, $bval_{c_2}(E(-v))$, and $bval_m(E(-v))$ from which the actual value v cannot be recovered knowing only the public parameters of Paillier and μ [19]. Note that Broker even may not be able to feed these blinded values into formula (1) in an attempt to recover the unblinded values, because in general the blinded values are not in the domain S_n of function $L(\cdot)$ (see Section 3.2). In this way the Broker is forced to follow the `Match` protocol as specified, obtain $r_m \cdot (x - v) + r$, and make matching decisions using Table 1.

Similarly, in `Cover` protocol, although Broker is able to perform operation as in formula (7) to obtain $r_c \cdot (v_1 - v_2) + r$ or $r_c \cdot (v_2 - v_1) + r'$, then use Table 1 to make covering decisions, it cannot perform decryption to get either v_1 or v_2 from the blinded values. In this way, Subs' subscription privacy is protected.

Note that since r and r' are selected uniformly at random for each execution of $bval_n$, $bval_{c_1}$ and $bval_{c_2}$ functions, the *diff* values obtained from `Cover` and `Match` do not reveal the actual distribution. Even for multiple subscriptions and notifications with the same values, Broker gets different *diff* values due to the randomization. Having said that, however, it should be noted that `Match` and `Cover` inherently leaks certain information about subscriptions and notifications even with such randomization. It is hard, if not impossible, to prevent such leakages.

B. STANDARD PROTOCOL EXPERIMENTS

We compare our protocol results with the well established computations to show that our approach is efficient and practical.

Table 2: Average computation time for general operations

Computation	Time (in ms)
Create access token (Sub)	4.21
Open access token (Pub)	4.17
Sign access token (Pub)	4.10
Verify token signature (Broker)	0.36
ZKP of access token (Sub)	4.18
ZKP of access token (Broker)	6.31
Encrypt payload message (Pub)	34.56
Decrypt payload message (Sub)	0.36

Table 2 shows the average running time for various operations for which we kept the system parameters constant. Access token creation, opening, signing are performed during `Register` protocol and based on Pedersen commitment scheme. Pub signs the access token using SHA-1 and RSA with 1024-bit long private key K_{pri}. Verification of the signature on the access token using the public key K_{pub}, and the ownership proof of the access token via the ZKPK are performed during `Subscribe` protocol. Zero-Knowledge Proof (ZKP) protocols are generally considered time consuming, but in our approach ZKP computation is comparable to other operations in the system, in that it takes merely a few milliseconds. For the experiments, we set the payload size to 4 Kbytes and used AES-128 as the symmetric key algorithm. These performance results demonstrate that the constructs we use and the computations are very efficient.