Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures

Costin Raiciu David S. Rosenblum
Department of Computer Science
University College London
{c.raiciu|d.rosenblum}@cs.ucl.ac.uk

Abstract—**Content-Based Publish/Subscribe (CBPS) is an interaction model where the interests of subscribers are stored in a content-based forwarding infrastructure to guide routing of notifications to interested parties. In this paper, we focus on answering the following question: Can we implement content-based publish/subscribe while keeping subscriptions and notifications confidential from the forwarding brokers? Our contributions include a systematic analysis of the problem, providing a formal security model and showing that the maximum level of attainable security in this setting is restricted. We focus on enabling provable confidentiality for commonly used applications and subscription languages in CBPS and present a series of practical provably secure protocols, some of which are novel and others adapted from existing work. We have implemented these protocols in SIENA, a popular CBPS system. Evaluation results show that confidential content-based publish/subscribe is practical: A single broker serving 1000 subscribers is able to route more than 100 notifications per second with our solutions.**

Index Terms—**confidentiality, content-based publish/subscribe, privacy-preserving range matches**

I. Introduction

CONTENT-BASED publish/subscribe (CBPS) is a convenient interaction model for distributed systems, allowing decoupled messaging through the CBPS infrastructure between two types of actors: (1) *subscribers*, having interests in information they express as *subscriptions*, and (2) *publishers*, producing information of interest as *notifications*. A network of CBPS *brokers* provides a decentralized infrastructure whose role is to disseminate notifications efficiently from the publishers to all the subscribers that have matching interests, optimizing aspects like bandwidth usage or end-to-end latency.

Research in the publish/subscribe area has traditionally focused on the scalability issues of publish/subscribe networks, yielding distributed algorithms for wide-area event notification and matching by using infrastructures comprising a mesh of publish/subscribe brokers [1, 2]. An implicit assumption underlying this research has been that the forwarding brokers must be trusted with subscription and notification information to perform correct content-based matching. This is only acceptable if we are running applications over dedicated infrastructures of trusted brokers. However, we expect CBPS to be enabled on top of existing infrastructures of third party providers (equivalents of content distribution networks, such as Akamai [3]), or even in a peer-to-peer manner, to minimize the costs of deployment or to distribute the burden of maintenance. A small-scale stock quote provider, for instance, might use a third-party CBPS provider to disseminate data to home users.

In this paper, we focus on a particular aspect of security in CBPS, *confidentiality*, that has two facets in this context [4]:

Notification confidentiality: The stock quote provider will be reluctant to disclose stock information to the infrastructure, if it believes that some brokers could re-sell or otherwise exploit this information.

Subscription confidentiality: Subscribers would like to keep their stock quote subscriptions private from the forwarding brokers, as these might leak their business strategy.

Creating solutions to deal with these aspects is paramount if CBPS is to be adopted as a solution for data dissemination. However, the very nature of CBPS—targeting a sweet spot in the space of plain IP-multicast (when subscribers can be clustered into groups based on their topic interests), broadcast (when most subscribers want most of the data) or simple unicast (if few subscribers are interested in even fewer data)—makes solutions quite difficult to obtain, as they must meet strict performance requirements and tight security guarantees.

In this paper, we provide a thorough study of confidentiality in the context of content-based forwarding and present our solution, being the first work to fully address this issue. The contributions of this paper include

1. a formal definition and a systematic analysis of confidentiality in CBPS;
2. several provably secure techniques, either novel or newly adapted from existing work, that enable Confidential Content-Based Publish/Subscribe (C-CBPS) for commonly used types of subscriptions and notifications; and
3. an implementation of our solutions in a popular CBPS system, SIENA [1], making available to the community the first complete implementation of publish/subscribe that supports confidentiality. Evaluation results prove that our solutions are lightweight enough to be suitable for usage in real applications.

The paper is organized as follows: Section II provides background information. Section III presents our definition of the Confidential CBPS problem and also discusses inherent limitations. In Section IV we present our solutions for Confidential CBPS. Section V discusses how these protocols are bundled into a complete solution and implemented in SIENA. Section VI presents experimental results, and Section VII reviews existing work in the area. We conclude in Section VIII with a summary of our contributions and future directions of research.

II. Background

A. Content-Based Publish/Subscribe

One of the distinctive characteristics of the publish/subscribe interaction model is loose coupling between the publishers and subscribers: spatial decoupling, temporal decoupling and flow decoupling [5]. These make publish/subscribe systems ideal

1-4244-0423-1/06/$20.00 ©2006 IEEE

for interactions between a large number of publishers and subscribers, scattered spatially across the entire Internet.

Subscribers have the ability to express their interest in an event by sending a subscription to an infrastructure comprising a decentralized network of publish/subscribe brokers. The infrastructure delivers to the subscribers any published notification that matches their registered interests. In content-based publish/subscribe, the subscription is a predicate containing one or more constraints or *filters* on the attributes of notifications, with each filter applying to a single attribute. An event notification is a set of attributes, where an attribute is a triple: (*name*, *type*, *value*) [1].

The commonly used example is stock quotes dissemination, where the event notification contains attributes such as *subject* (string, the name of the event), *exchange* (string, the name of the stock exchange), *symbol* (string, for example DIS), *price* (float, the current value of the specified stock) and *change* (float, the variation of price with respect to the previous value). An example subscription is (*change*>0) *and* (*symbol*=*"DIS"*).

The main task of the CBPS broker is to match a notification against the subscriptions it stores, determining which subscribers should receive the notification. Subscription S_1 is said to cover S_2 if the set of notifications matched by S_2 is a subset of those matched by S_1 [1]. CBPS systems use the covering relation between subscriptions to ensure a sublinear increase of broker matching time with the number of stored subscriptions.

B. Security Preliminaries

We say that a function f is *negligible* in t if, for any polynomial p there exists t_0 such that for all $t > t_0$, $f(t) < 1/p(t)$. We use PPT as a shorthand for *probabilistic polynomial time*.

We provide the following standard definitions from the literature on provable security [6], which we will use throughout this paper.

Pseudorandom Function. A pseudorandom function is computationally indistinguishible from a random function. Formally, a function family $\{F_K : \{0,1\}^n \rightarrow \{0,1\}^m | K \in \{0,1\}^t\}$ is pseudorandom if for every PPT oracle algorithm A the following value is negligible in t: $|Pr[A^{F_K(\cdot)}(1^t) = 1] - Pr[A^R(1^t) = 1]|$, where R is a random function selected uniformly at random from the set of functions from $\{0,1\}^n \rightarrow \{0,1\}^m$. The probabilities are taken over the choice of K and R, respectively.

Pseudorandom Permutation. A pseudorandom permutation is computationally indistinguishible from a truly random permutation. Formally, a permutation family $\{E_K : \{0,1\}^n \rightarrow \{0,1\}^n | K \in \{0,1\}^t\}$ is pseudorandom if for every PPT oracle algorithm A, the following value is negligible in t: $|Pr[A^{E_K(\cdot)}(1^t) = 1] - Pr[A^\pi(1^t) = 1]|$, where π is a permutation selected uniformly at random from the set of bijections from $\{0,1\}^n \rightarrow \{0,1\}^n$. The probabilities are taken over the choice of K and π, respectively.

III. Problem Statement and Implications

A. Security Assumptions and Scope

We assume that the publishers and subscribers are trustworthy and that they share a common secret used for confidential CBPS. Although the latter reduces the decoupling provided by CBPS (as publishers and subscribers must use an auxiliary distribution channel for the common secret), it is central to preserving confidentiality. The group key distribution problem has been studied intensely in the literature, with several solutions available [7]. Here, we treat the key distribution problem as orthogonal to this work.

Preventing a malicious subscriber from disseminating all the information it receives to other parties is equivalent to solving the digital copyright problem. This is also outside the scope of this work.

We assume that brokers are computationally bounded and do not deviate from the content-based forwarding protocol—they correctly forward notifications to interested subscribers. Otherwise, denial of service attacks could be mounted easily (since brokers could simply drop messages), affecting the correct operation of the infrastructure. We do not consider the effects of this type of behavior, leaving it as an issue for future research.

B. Problem Definition

Definition 1: **Confidential Content-Based Publish/Subscribe (C-CBPS).** Consider two types of parties P (publishers) and S (subscribers) having private inputs. Each P_i has a sequence of notifications, and each S_j has some information interests (filters or subscriptions), a subset of which is active at any point in time. C-CBPS is a multi-round protocol between P,S and a third party R, the broker. In each round one of the following can take place: a) S_j (for some j) sends its active filters to R; or b) P_i (for some i) sends a notification to R. A correct implementation of C-CBPS with security parameter t must satisfy the following:

Correctness R must be able to determine in PPT the subset of active filters that matches each notification.

Security For $k \in \mathbb{N}$, define $View_k$ as all the communications R has received from P and S before round k. Define $Plaintext_k = \{N_1, \ldots, N_i, S_1, \ldots, S_j\}$ as the set of subscriptions and notifications sent by P and S before round k. Let O_k be an oracle that has access to $Plaintext_k$ and exports the two following functions: *match*(idx_S,idx_N) and *cover*($idx_{S,1}$,$idx_{S,2}$), defined iff $idx_S, idx_{S,1}, idx_{S,2} \in \{1, \ldots, j\}$ and $idx_N \in \{1, \ldots, i\}$. Finally, define $View^*_k = \{i, j, O_k\}$.

A C-CBPS scheme is secure if, for $k \in \mathbb{N}$, for any PPT algorithm A, any function h, there exists a PPT algorithm A^* such that the following value is negligible in t: $|Pr[A(View_k, 1^t) = h(Plaintext_k)] - Pr[A^*(View^*_k, 1^t) = h(Plaintext_k)]|$

In other words, we require that information leaked to the broker is the same as in an ideal protocol where the broker performs its functionality by submitting the indexes of the subscriptions and notifications it wishes to match (idx_S and idx_N) to an oracle (O_k) with access to the plaintext versions. The above definition implies the following:

Notification Security Notifications possess semantic security for multiple messages (as defined by Goldreich [6]) in the absence of subscriptions. When subscriptions are available, the only thing that is leaked is whether a notification matches the subscription or not. The notifications that are

not matched by the available subscriptions are computationally indistinguishable from random bits.

Subscription Security Subscriptions can be distinguished using the covering relation, and therefore their encryption scheme is not semantically secure. A stronger security model could require that the subscription encryption scheme is also semantically secure. In this paper we discard this stronger model for practical purposes: efficient CBPS solutions rely on the coverage relation between subscriptions, which mandates that a broker should know if two subscriptions are related [8].

Notification Unforgeability It is infeasible for an adversary to create valid encrypted notifications. This is important, since an adversary able to craft arbitrary notifications can use regression techniques to infer an approximation of the subscription function.

Subscription Unforgeability It is infeasible for an adversary to create valid encrypted subscriptions. Otherwise, the adversary can use binary search to discover the value of the notification in logarithmic time. An important consequence of subscription and notification unforgeability is that plaintext subscriptions or notifications cannot be used in the matching process (since these are easy to create by adversaries).

Match Isolation It is infeasible to compute anything from the messages seen at the broker that cannot be computed by applying *match* and *cover* (using an oracle) to the indexes of subscriptions and notifications.

The definition above can be generalized naturally to the multi-broker case where the number of forwarding brokers is arbitrarily large.

Any solution for confidential CBPS consists of the following five algorithms (the first 4 are compulsory for correct C-CBPS, the last one may be provided for efficiency):

`Keygen`(t): run jointly by the publishers and subscribers, outputs the shared private key K when given the security parameter t as input

`IndexSub`(K,S): run by the subscriber, outputs the encrypted subscription S_e when given the plaintext subscription S and the private key K

`IndexNot`(K,N): run by the publisher, outputs the encrypted notification N_e when given the notification S and the private key K

`Match`(N_e,S_e): run by the broker, receives as parameters an encrypted notification N_e and an encrypted subscription S_e and outputs 1 if S_e matches N_e or 0 otherwise

`Cover`(S_{e_1},S_{e_2}): (Optional) run by the broker, accepts as parameters two encrypted subscriptions S_{e_1} and S_{e_2}, and outputs 1 if S_{e_1} covers S_{e_2} or 0 otherwise

For simplicity of exposition, we use the term "encrypt" to denote a secure encoding of subscriptions and notifications that allows C-CBPS. However, we point out that the schemes presented here are not traditional symmetric encryption schemes, since decryption is not usually possible.

C. Limitations of Confidentiality

Regardless of the protocols used, the maximum level of attainable confidentiality in CBPS is quite limited. These limitations arise from the functionality the broker is required to perform (i.e., to decide if an encrypted subscription matches an encrypted notification) and are inherent to the C-CBPS problem. Here, we present a brief overview of these limitations.

C.1 The "Attack at Dawn" Problem

The mere fact that a subscription matches a notification can leak crucial information, if the identities of the publishers and subscribers are known to the broker. An (admitedly contrived) example is this: The army has several operational units that register subscriptions to receive specific commands, such as "attack at dawn". When such a message is matched by such a subscription, the broker knows that something important is going on, even if it does not know the contents of the notification or subscription. Single-broker CBPS infrastructures are most susceptible to this problem; in multi-broker deployments, brokers in the core of the network are unable to discover the identities of the publishers and subscribers without colluding with the edge brokers.

This problem can be mitigated by using sender and receiver anonymizing techniques [9, 10] for communication between the end-users and the brokers.

C.2 Limited Indistinguishability

Subscriptions stored by a broker can be used to distinguish certain notifications (e.g., to tell if they are equal) by matching the subscriptions against the notification: this uses the fact that the broker must be able to match subscriptions against notifications, and is independent of the encryption scheme used for notifications. The more subscriptions that are available, the more likely the broker is to accurately distinguish notifications. In the case where the broker has a complete basis of subscriptions, it can distinguish all notifications with zero probability of error.

C.3 Confidentiality-Generality Tradeoff

We define the "complexity" of a subscription type as $\frac{1}{min_S}$, where min_S is the minimal number of subscriptions needed to recognize all notifications. There is a direct correlation between the complexity of a subscription and the information it leaks about notifications. For instance, the simplest subscription function is equality testing: one such subscription will allow a broker to distinguish notifications that are equal to the specified value. To distinguish all possible notifications without error (i.e., to have a basis), the broker needs $O(2^n)$ distinct subscriptions, where n is the size of the notification in bits. The more complex subscriptions are, the more information is leaked about notifications. For instance, a subscription that accepts all notifications with the k^{th} bit set to a specific value, will allow the broker to distinguish information about the k^{th} bit of all notifications. In this case, only $O(n)$ subscriptions are needed to distinguish notifications with zero probability of error.

C.4 Trust

The assumption that any individual in the potentially large group of publishers and subscribers is trusted is strong: If a single subscriber leaks the secret key to the brokers, then the security of the protocol is compromised.

In the single key setting, detecting the source of information leakage is very difficult, and so is excluding malicious subscribers from the network. Broadcast Encryption [11] is used to trade communication efficiency for accountability and traceability of malicious subscribers in broadcast networks. Unfortunately broadcast encryption cannot be used directly in C-CBPS; devising communication-efficient broadcast encryption techniques that can be used for content-based matching is a research agenda in itself, not explored in this paper.

IV. Solutions

There are two high level approaches to solving C-CBPS: one way is to encrypt notifications as a whole and match them against the encrypted subscription. This approach requires support for complex subscriptions appearing in practice, which are difficult to support efficiently and securely in the same time, as we show in Section IV.D.

The alternative approach, used in this paper, relies on the structure of notifications (which are collections of attribute-value pairs). This allows us to support overall complex subscriptions that are composed from several simpler building blocks. The security of this compound protocol is weaker than Definition 1, even though the basic building blocks are secure. However, through careful selection of the attributes that the broker is allowed to "match", the amount of leaked information can be controlled and should be acceptable in practical cases. Obviously, in the case where a subscription consists of a single filter and the notification contains a single attribute, the compound protocol is secure according to Definition 1.

In this section, we present C-CBPS solutions for types of subscriptions supported by the wide majority of CBPS solutions nowadays: We adapt a scheme from Song et al. [12] to support *equality tests* (string and numeric-valued attributes), we define two novel schemes for *range matches* (numeric-valued attributes) and we use a scheme from Goh et al. [13] for *keyword matching* (string-valued attributes). We also discuss techniques to support more general subscriptions.

In the descriptions of the basic C-CBPS schemes, we assume each notification is a single value, and each subscription is a single filter.

A. Equality Filtering

One of the prominent filtering functions used in practice is equality matching. To support equality matches, we use the first step of the solution proposed by Song et al. for searches on encrypted data [12]. The idea is to compute the "hidden" value of an attribute by passing its plaintext value as argument to a pseudorandom function, keyed with the secret key. The encrypted subscription is the hidden value of the plaintext. Encrypted notifications are composed of two parts: a random nonce r, generated by the publisher, and the result of feeding r to a pseudorandom function, keyed with the hidden value of the notification's plaintext.

Let F be a pseudorandom function. The algorithms for *Equal* C-CBPS are:

`Keygen`(t): select K from $\{0,1\}^t$ uniformly at random
`IndexSub`(K,S): return $F_K(S)$
`IndexNot`(K,N): select rnd uniformly at random. Let $h = F_K(N)$. Return $(rnd, F_h(rnd))$.
`Match`(N_e,S_e): Let $N_e = (rnd, two)$. Return 1 if $F_{S_e}(rnd) = two$, 0 otherwise
`Cover`(S_1,S_2): Return 1 if $S_1 = S_2$, 0 otherwise

Theorem 1: Equal is a correct implementation of C-CBPS.

The proof is presented in the Appendix.

This scheme is cheap from both the computation and communication points of view. Computation-wise, the scheme adds a few cheap operations to creating subscriptions/notifications and a single function application for matching.

B. Keyword Filtering

Substring matching is the most expressive operation currently implemented on strings in common CBPS architectures. We choose to support a simpler operation—keyword matching—based on the observation that this suffices for many applications. The protocol we use has been proposed by Goh [13]. The idea is to break the string into words and construct a Bloom filter [14] to signal existence of a word in the string. The subscription is a single keyword.

Let F be a pseudorandom function. Let BF be a Bloom filter. The algorithms for Keyword C-CBPS (Goh [13]) are:

`Keygen`(t): select r as the number of hash functions in the Bloom filter BF with the desired false positive rate. Select $K = (k_1, k_2, \ldots, k_r)$ uniformly at random from $\{0,1\}^{rt}$.
`IndexSub`(K,S): return $(F_{k_1}(S), \ldots, F_{k_r}(S))$
`IndexNot`(K,N): extract keywords $w_1, \ldots, w_n$ from N. Select a random nonce rnd. For $i = 1 \ldots n$, compute $(x_{i,1}, \ldots, x_{i,r})$ = `IndexSub`(K, w_i), compute the codeword ($y_1 = F_{rnd}(x_{i,1})$, $y_2 = F_{rnd}(x_{i,2})$, $\ldots$, $y_r = F_{rnd}(x_{i,r})$) and set $BF[y_j] = 1$ for $j = 1 \ldots r$. Return (rnd, BF)
`Match`(N_e,S_e): Let $S_e = (x_1, x_2, \ldots, x_r)$. Let $N_e = (rnd, BF)$. Compute codewords $y_i = F_{rnd}(x_i)$ for i=1 $\ldots r$ and check if the bit corresponding to y_i is set in BF. If there exists i such that $BF[y_i] = 0$ return 0, otherwise return 1
`Cover`(S_1,S_2): Return 1 if $S_1 = S_2$, 0 otherwise.

We make the assumption that all strings have a predefined length and that they have the same number of words. This prevents an attacker from distinguishing two notifications by counting the number of bits set in the BF. When the latter assumption does not hold, we can add random bits to the BF to simulate the proper number of words [13].

Theorem 2: Keyword is a correct implementation of C-CBPS.

A proof sketch is presented in the Appendix.

The overhead of the protocol mainly lies in transmitting the Bloom filter, which can be as large as the size of the string, if all the words are included as possible keywords.

Alternative Schemes. Other solutions are available for keyword filtering [12, 15, 16]. The scheme proposed by Chang et al. [15] is based on creating a dictionary that has one bit for every possible word in the string. The dictionary is shuffled using a pseudorandom permutation and blinded using pseudorandom functions and a random nonce. The notification includes the blinded dictionary, along with the random nonce.The sub-

scription contains the shuffled index of the word plus a "hidden" version of the index.

Let F, G be two pseudorandom functions and E be a pseudorandom permutation. The *Dictionary* scheme is:

`Keygen`(t): select $K = \{K_1, K_2\}$ uniformly at random from $\{0,1\}^t \times \{0,1\}^t$.

`IndexSub`(K,S): find index λ of S in the dictionary D. Return $\{index = E_{K_1}(\lambda), F_{K_2}(index)\}$

`IndexNot`(K,N): let J and I be two bit index strings of size $|D|$, initialized to 0. For all words $w_1, \ldots, w_n$ in N, find λ_i (the index of w_i in the dictionary) and set $I[E_{K_1}(\lambda_i)] = 1$. Select a random nonce rnd. For $i = 1 \ldots |D|$, compute $r_i = F_{K_2}(i)$ and set $J[i] = I[i] \oplus G_{r_i}(rnd)$. Return ($rnd$, J)

`Match`(N_e,S_e): Let S_e = ($index$, r_{index}). Let N_e = (rnd, J). If $J[index] \oplus G_{r_{index}}(rnd) = 1$ return 1, otherwise, return 0

`Cover`(S_1,S_2): Return 1 if $S_1 = S_2$, 0 otherwise.

Theorem 3: Dictionary is a correct implementation of C-CBPS.

A proof sketch is presented in the Appendix.

Compared to *Keyword*, *Dictionary* does not generate false positive matches and does not impose any restrictions on the number of words in the document. However, the size of the encrypted notification is 32kB for the English language [15], being very expensive for small documents. The expected size of string attributes in CBPS is quite small (hundreds of bytes usually) favoring the first scheme. *Dictionary* can be used when the size of the string is larger or comparable to 32kB or in cases where the dictionary is smaller.

C. Numeric Filtering

Filtering numeric attributes is frequent in practice and is consequently supported by most implementations of content-based publish/subscribe. Let $D \subset \mathbb{R}$ be the notification space. Given a notification $N \in D$, the subscription can have two forms: a) inequality tests ($N > l_b$, $N < u_b$) or b) range tests ($l_b < N < u_b$), for $l_b, u_b \in D$. We define two novel C-CBPS schemes for the two cases.

These two protocols, despite being presented in the context of C-CBPS, have wider applicability. In particular, they can be used for privacy preserving range matches: let the CBPS broker be a public filestore and assume one user (publisher) stores some files, including metadata such as file size. The same user can then send a query (i.e. a subscription) to the server requesting files that have the size bounded by some constraints; using this special instance of C-CBPS, the fileserver can figure out which files he should return without finding ouy anything more than necessary.

C.1 Supporting Inequality Subscriptions

Choose l points, $p_1, \ldots, p_l \in D$ as reference points. We consider the following dictionary: {"$> p_1$", "$> p_2$", ..., "$> p_l$" "$< p_1$", "$< p_2$", ..., "$< p_l$"}. Subscriptions will be approximated with one of these constraints. Each notification N is considered to be a document containing the words in the dictionary that it matches. These are encrypted using the *Dictionary* scheme we have previously described. The *Inequality* scheme is:

`Keygen`(r): K = *Dictionary*.`Keygen`(r). Agree on a set of l reference points $p_1, \ldots, p_l \in D$.

`IndexSub`(K,S): Let $S = (type, value)$, where $type$ can be "$<$" or "$>$". Find i such that $|value - p_i| = \min_{j=1}^{l} |value - p_j|$. Return *Dictionary*.`IndexSub`(K, $type|p_i$)

`IndexNot`(K,N): Let $N_w = \{t_i|p_i$, where t_i="$>$" if $N > p_i$ and t_i="$<$" if $N < p_i$, for $i = 1 \ldots l\}$. Return *Dictionary*.`IndexNot`(K, N_w)

`Match`(N_e,S_e): return *Dictionary*.`Match`(N_e,S_e)

`Cover`(S_1,S_2): we check whether the subscriptions are the same by using *Dictionary*.`Cover`. Full subscription covering cannot be checked without additional information in this case. We present an efficient solution in the Implementation section, which leaks some additional information

Theorem 4: Suppose all subscriptions can be expressed exactly using the mechanisms above. Then, *Inequality* is a correct implementation of C-CBPS.

A proof sketch is presented in the Appendix.

The overhead of this scheme is due to the size of the dictionary, equal to $2 \cdot l$. There is a direct tradeoff between this overhead and the precision it allows for subscriptions.

If we want perfect subscriptions (0 false positive and negative matches), we set $l = |D|$. This can be expensive in reality (e.g., for 4 byte integers we have $\sim 10^9$ points). We describe an exponentially spaced partitioning scheme that is useful in many practical scenarios. Approximating the 4 byte positive integers with $[1 \ldots 10^9]$, we select as reference points: $1, 2, 3, \ldots, 10, 20, 30, \ldots, 100, 200, 300, \ldots, 1000, \ldots, 10^8$ $, 2 \cdot 10^8, 3 \cdot 10^8, \ldots, 10^9$. Although the number of reference points is only 100 (the notification has only 12 bytes), the precision is acceptable if we consider that subscriber sensitivity decreases as notification values increase.

C.2 Supporting Range Subscriptions

To support $l_b < N < u_b$ subscriptions, our initial idea was to have the publishers and subscribers a-priori agree on a partitioning $P = \{p_1, \ldots, p_l\}$ of D. The publisher encrypts the index of the subset N belongs to by using *Equal*. The subscribers include as subscriptions encrypted versions of the indexes of the subsets in the partition they are interested in (i.e., all $p_i \in P$ such that $p_i \cap (l_b, u_b) \neq \emptyset$). However, sending multiple subsets leaks more information than necessary. Therefore, we would have to approximate the subscription with a single subset in the partition. This is not very precise since subscription sizes $(u_b - l_b)$ and l_b vary among subscribers.

The initial idea can be refined as follows. Create several partitions of D, $P_1, \ldots, P_m$, with different subset sizes and different starting offsets. Create a dictionary containing as words the index of the partition concatenated with the subset index, for all m partitions. A notification can be expressed as a document with this dictionary by listing the subsets it is included in. The subscription is approximated with one of the subsets in these partitions. The *Range* scheme is:

`Keygen`(r): Generate K using *Dictionary*.`Keygen`. Agree on m partitions of D, $P_1, P_2, \ldots, P_m$, where $P_i = p_{i,1} \cup$

$p_{i,2} \ldots \cup p_{i,l_i}$. Let $p_{i,j} = [a_{i,j}, b_{i,j}]$

`IndexSub`(K,S): Let $S = (l_b, u_b)$. Find the best approximation of S in $P_1, \ldots, P_m$. In particular, find x and y such that $|l_b - a_{x,y}| + |u_b - b_{x,y}| = \min_{i=1}^{m} \min_{j=1}^{l_i}(|l_b - a_{i,j}| + |u_b - b_{i,j}|)$. Return *Dictionary*.`IndexSub` ("x, y")

`IndexNot`(K,N): Let N_w = {"x, y"| where $x \in \{1, \ldots, m\}$ and $y \in \{1, \ldots, l_x\}$ such that $N \in p_{x,y}$. Return *Dictionary*.`IndexNot`(K, N_w)

`Match`(N_e,S_e): return *Dictionary*.`Match`(N_e,S_e)

`Cover`(S_1,S_2): we can easily check to see if two subscriptions are the same by using *Dictionary*.`Cover`. However, we cannot properly check full covering without additional information. In Section V we describe an efficient coverage solution that can be used instead, but leaks more information than necessary

Theorem 5: Suppose all subscriptions can be expressed exactly (i.e., without generating false positives or negatives) using the above algorithm. Then, *Range* is a correct implementation of C-CBPS.

A proof sketch is presented in the Appendix.

The scheme creates an explicit tradeoff between the size of the subscriptions and matching time on one hand, and the number of false positives and the security attained (i.e., information leaked due to imprecise subscriptions), on the other. A partitioning scheme with zero false matches for any range subscription has $|D|^2$ points, being quite expensive. A better scheme can be obtained if we focus on subscription sizes likely to be used in practice. Keeping in mind that CBPS targets distributed applications where subscribers are highly selective (otherwise multicast or broadcast can be used), it is reasonable to assume that any single single subscription accepts between 5% and 10% of the notifications. In this case, given a (non-zero) desired false match rate, the size of the proper partitioning does not depend on $|D|$. For instance, in the Evaluation section, we use a partitioning scheme that yields for a uniform distribution of notifications 5% false matches (out of the 5%-10%) accepted by the subscription) and contains only 890 words in the dictionary.

In general, given a desired cost, choosing the proper partitioning is application specific and should take into consideration the distributions of subscriptions and notifications. An algorithm that determines the optimal partitioning strategy for a specificied cost is presented by Hore et al. [17] and could be used for this task.

D. Supporting Generic Subscriptions

Supporting arbitrary functions as subscriptions is not a goal in itself, as the maximum achievable security is not satisfactory: Only $O(|N|)$ carefully chosen subscriptions are enough to distinguish every notification. This, combined with the knowledge of a plaintext-ciphertext pair, completely breaks the notification encryption scheme. However, it is interesting to discuss approaches for generic subscription functions as a possible starting point to support other subscription functions of practical interest.

There is a tradeoff between the amount of information leaked to the brokers and the communication overhead. Therefore, to support generic subscriptions we can trade confidentiality for communication efficiency.

At one end of the solution space, the minimum amount of information is revealed and communication size is very expensive. Consider an enumeration of all functions from $D \rightarrow \{0,1\}$. The dictionary will contain the indexes of all these functions. We use *Dictionary* to encode arbitrary subscriptions by encrypting the proper index. Notifications will include as words all the indexes of functions that accept them. This scheme is secure for all possible subscriptions as it does not leak more information than what is needed. The communication size is huge: Every notification has $2^{|D|}$ bits.

At the other end of the solution space, we have examined and implemented a protocol based on Yao's garbled circuit construction to support generic subscriptions, expressed as boolean circuits [18]. The size of the communication is small (subscription size is directly proportional to the number of gates in the circuit, while notification size is the same as the plaintext version). However, this scheme allows the broker to distinguish every bit of the notification, and therefore a single plaintext-ciphertext pair is needed to completely break notifications (without needing |N| "good" subscriptions as a basis).

V. Implementing C-CBPS

The mechanisms described above have been implemented in Siena [1], a wide-area event-notification service. Siena is a content-based publish/subscribe infrastructure where brokers are vertices in a connected overlay acyclic graph. Siena supports as filters relational operators for numeric values and substring matching for string values. The implementation can be downloaded at *http://www.cs.ucl.ac.uk/staff/c.raiciu/securepubsub/*

We chose Siena due to its popularity and widespread adoption within the research community. Our algorithms can be embedded easily in other CBPS solutions (such as Gryphon [2] or Elvin [19]).

A. The High-Level Matching Algorithm

Siena notifications are attribute-value pairs. We treat attribute values as confidential data that will be encrypted using one of the C-CBPS schemes described previously. The publishers control which attributes can be "matched" and the schemes that can be used for matching. Subscriptions comprise one or multiple filters, each of which refers to a single attribute. A subscription matches a notification if all its filters match the corresponding encrypted values of the attributes.

We describe the operation of the composite Confidential CBPS algorithm by revisiting the stock quote dissemination example (presented in Section A). Assume the publisher wishes to allow clients to filter stock quotes based on the *symbol* and *change* attributes in the notification. Accordingly, it will use *Equal* to allow equality filtering for *symbol* and *Inequality* to allow threshold checks for *change*.

The publisher and the subscribers will jointly run the `Keygen` phase of *Equal* and *Inequality* to create the keys K_e and K_i, respectively.

Assume one subscriber wishes to receive all stock quotes with (*change* > 0) *and* (*symbol*="*DIS*"). The subscriber will generate the following encrypted version: (*name*="*change*", *scheme*="*Inequality*", *filter* = *Inequality*.`IndexSub` (K_i,>

0)) *and* (*name* = *"symbol"*, *scheme*= *"Equal"*, *filter*= *Equal*.`IndexSub` (K_e,"DIS")).

Consider the notification containing (*change*=10, *symbol*="DIS", *exchange*="NYSE"). Its encrypted version has two parts:

Matchable Part is (*name*="change", *scheme* = *"Inequality"*,*value=Inequality*.`IndexNot`(K_i,10)), (*name*="symbol",*scheme*=*"Equal"*,*value=Equal*.`IndexNot`(K_e,"DIS"))

Payload is an encryption of the whole notification using a symmetric encryption scheme.

When the broker receives a notification, it iterates all the subscriptions it stores and performs the following steps:

1. For each attribute in the subscription it looks for an attribute with the same name in the matchable part of the notification, that has been encrypted using the same encryption scheme. If there exists an attribute in the subscription not present in the notification, the result of the match is 0.
2. For each attribute-filter pair with matching names and schemes it calls the corresponding `Match` algorithm, passing the *value* and *filter* as parameters.
3. If all filters in the subscription match the corresponding attributes in the notification, the result is 1. Otherwise, the result is 0.
4. Upon a true match, the broker sends the *payload* to the subscriber(s), if the latter is directly connected to the broker. Otherwise, it forwards the entire notification to the broker(s) closer to the subscriber(s).

B. Implementation Notes

Generally, we tried to keep modifications to the existing SIENA code minimal to allow backward compatibility and easy integration with deployed solutions. We created *encrypted filters*, a new type of filter, that contains, besides the attribute name, a serialization of the encrypted subscription, encoded either using one of the schemes described in the previous section. An encrypted matcher manager keeps track of supported encryption formats and makes sure that notifications are matched against subscriptions only if they have been encrypted in the same way.

We used the SHA-256 cryptographic hash function [20] throughout our implementation as a pseudorandom function. We used 128-bit AES [21] for the symmetric encryption scheme and as a pseudorandom permutation.

C. Subscription Covering

Exploiting the covering relation between subscriptions increases matching performance. We now discuss how to enable full subscription covering for *Inequality* and *Range* and what information this leaks.

Inequality. Add a *hint* to every subscription, which has two parts: a) the result of applying *Inequality*.`IndexNot` to the threshold value and b) a deterministic encryption of the type (i.e., "<" or ">").

The covering algorithm is to return 0 if $S_1.hint.type \neq S_2.hint.type$. Otherwise, return *Inequality*.`Match` (S_1, $S_2.hint.threshold$). Clearly, this scheme has some overhead due to larger subscriptions. The scheme allows a broker to match the hint of one subscription against any other subscriptions even in the case when the subscriptions have different types (and therefore cannot cover each other). We explore the performance benefits of covering in the Evaluation section.

Range. The solution is to add encrypted versions of the approximate bounds (i.e., $a_{x,y}$ and $b_{x,y}$) to the subscription.

The covering algorithm is: return *Range*.`Match` (S_1,$S_2.a$) *and Range*.`Match` (S_1,$S_2.b$). This scheme has higher subscription overhead. The scheme allows a broker to independently check whether the margins (a and b) of one subscription are contained by the other, which is more than knowing the coverage relation. We evaluate the performance benefit of this scheme in the Evaluation section.

D. Key Management

Each attribute has one or a few supported subscription types. The schemes we have presented assume that a secret key will be generated for each of these. Clearly, this does not scale well with the number of searchable attributes.

To circumvent this, we use a master key and a pseudorandom function to generate a key for a given attribute name, type (int, string), and encryption scheme (*Range*). This is achieved by keying the pseudorandom function with the master key and applying it to the string obtained by concatenating the attribute name, type and the name of C-CBPS scheme. Each combination of attribute name, type and C-CBPS scheme will thus have its own key which will be used for C-CBPS. The security of any single C-CBPS scheme holds under computational assumptions. Furthermore, if any of these derived keys is leaked, the information available to an attacker is minimal: it is infeasible for an adversary to retrieve the "master" key, even if the name of the attribute, the type and C-CBPS scheme are known.

VI. EVALUATION

This section compares the performance of our solutions against plaintext filtering. The results show that the overhead imposed on the brokers and the network for confidential content-based forwarding is acceptable, making the solutions practical.

A. Evaluation Methodology

All the data used for testing is synthetic, being generated uniformly at random or using the power law distribution. In all the tests, a single instance of the enhanced SIENA matching engine was evaluated. All experiments were run on a 1.7Ghz Intel Centrino Processor with 1Gb of RAM running Windows XP and Sun's JDK 1.5. Time is measured by using the function *System.nanoTime()* available in Java 1.5. All the measurements were repeated to obtain a standard error of at most 1% of the measured value.

Matching time is measured as the time the broker spends to identify the set of matching subscribers, when presented with a notification. We define the reference matching time as the average matching time required to match a notification against 1000 subscriptions. Subscription and notification sizes are measured in terms of total network bytes sent, including SIENA's protocol overhead.

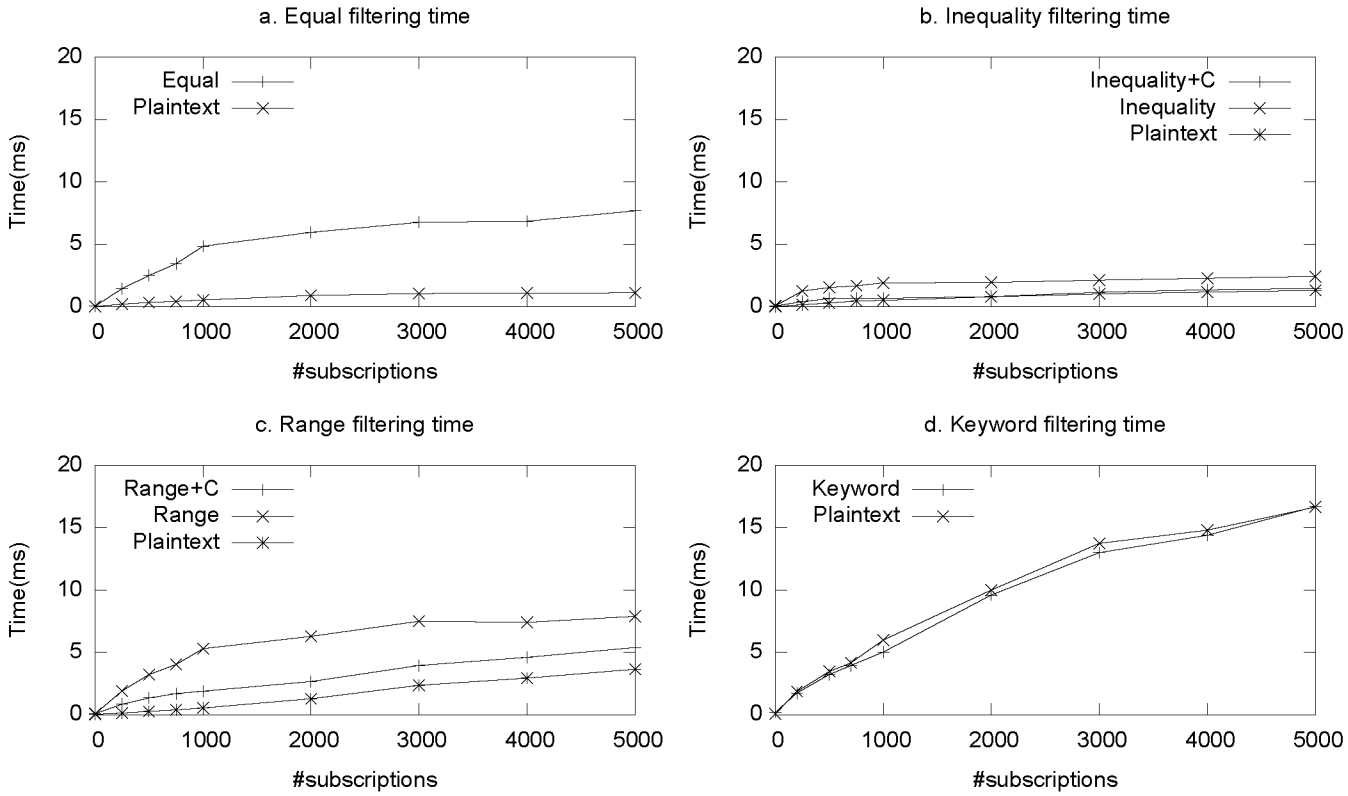

Fig. 1. Matching Time Comparison

We consider two major types of subscriptions that are representative for most applications using content-based publish/subscribe: filtering numeric attributes using relational operators (i.e., $<, >, \geq, \leq, \neq, =$) and filtering strings using keywords. We use the schemes with subscription covering enabled, except for the cases of *Inequality* and *Range* which are tested both with and without full covering.

B. Matching Time Measurements

Filtering Numeric Attributes. Notifications are integers drawn uniformly at random from $[0, 1000]$. The results are presented in Figure 1.

To test *Equal*, we select subscriptions uniformly at random from $[0, 1000]$. *Equal* is, on average, 6 times as expensive as its plaintext counterpart, and has a reference matching time of 4.5ms. When the number of subscriptions exceeds 1000, the coverage relation kicks in, dampening the increase in matching time (Figure 1.a).

Inequality filtering (Figure 1.b) uses 200 uniformly selected reference values $(0, 5, \ldots, 1000)$, and has a false positive match rate of 2% (i.e. the subscriber receives 2% notifications it shold not receive). We tested the scheme using subscription covering (*Inequality*+C) and without it (*Inequality*). Subscriptions are randomly selected to be of type $N > l_b$ or $N < u_b$ where l_b and u_b are uniformly distributed in $[500, 1000]$ and $[0, 500]$ respectively. *Inequality*'s reference matching time is 1.5ms, being 3 times as expensive as plaintext. Implementing covering brings important benefits: *Inequality*+C has a reference matching time of 0.8ms and is only 1.7 times as expensive as plaintext. The careful reader may have noticed that *Inequality* has better performance than *Equal*: this is due to the higher clustering of subscriptions for *Inequality*, which is due to approximation of subscriptions. However, when comparing the time to match notifications against a single subscription (i.e., the base performance), *Equal* is faster by 20%.

Range filtering (Figure 1.c) partitions $[0, 1000]$ for subscriptions accepting 5% to 10% of the notifications and has a false matching rate of 5%. The resulting partitions have a total of 890 subsets. The size of the subscriptions matched is selected uniformly from $\{50, 60, \ldots, 100\}$, with the beginning of the matched interval selected uniformly at random. *Range*'s reference matching time is 5ms, being 6 times as expensive as plaintext. Implementing full subscription covering reduces matching time by a factor of two: *Range*+C is 3 times as slow as plaintext and has a reference matching time of 2ms.

Filtering Strings with Keywords. For these experiments, notifications are strings comprising 50 words extracted randomly from a predefined collection containing 10000 words. We select subscriptions to be power-law distributed on the same collection. We tested *Keyword* and its plaintext counterpart for filtering on these strings. Although the comparison is a bit forced (arbitrary plaintext substring matching is more expressive than simple keyword matching), it is instructive to look at the performance of *Keyword*, presented in Figure 1.d. Remarkably,

TABLE I
COMMUNICATION OVERHEAD

	Subscription (bytes)	*Notification (bytes)*
Equal Plaintext	19	16
Equal	63	231
Range Plaintext	30	16
Range+C	847	378
Range	63	378
Inequality Plaintext	23	16
Inequality+C	264	213
Inequality	67	213
Keyword Plaintext	23	443
Keyword	364	704

Keyword is slightly faster than plaintext substring matching.

We notice that matching time of *Keyword* is quite good when compared to the simpler *Equal*. Although *Keyword* has worse base performance (5 times as as slow as *Equal*), higher subscription clustering significantly improves performance for large numbers of subscriptions.

Filtering with Composite Subscriptions. We also tested filtering using subscriptions that contain multiple constraints. Each subscription contains all the types of constraints we have discussed. The reference matching time is 9ms in this case, being 4 times as expensive as plaintext; this allows a broker to test 110 notifications against 1000 subscriptions in 1s. We conclude that the time overhead due to confidential CBPS is acceptable in practice.

C. Communication Overhead

Average subscription and notification sizes are presented in Table 1. Clearly, C-CBPS is not cheap: notifications are on average 15 times as large, and subscriptions are 10 times as large, when compared to their plaintext counterparts. If we consider the most expensive scheme, *Range*, we see that although the output of *Range*.`IndexNot` is only 100 bytes in size, the encrypted notification is 378 bytes; this is due to a particularity of the SIENA protocol which escapes a large number of characters in the byte arrays it serializes.

When a large number of attributes are "matchable", this overhead can become significant. In practice, only a subset of attributes in a notification must be matchable, and therefore the rest of the attributes can be encrypted symmetrically without adding (much) overhead.

Parameters. We explored the tradeoffs *Inequality* and *Range* allow. There is an inverse proportionality relation between the false match rate and the communication overhead due to "matchable" attributes. The slopes fall a bruptly from large false positive rates to acceptable ones if we increase communication overhead to as little as 200 bytes. There is a similar tradeoff between the false match rate and the matching time. These tradeoffs provide an intuition on how to select partitioning schemes for specific applications, based on the desired false match rate.

VII. RELATED WORK

To the best of our knowledge, this is the first complete and secure solution for C-CBPS that has been presented in the literature. We split the related work section in three parts: work in the broad area of secure function evaluation, work on privacy preserving keyword searches, and work on security in CBPS.

Secure Function Evaluation. Research in cryptography has produced many important results in the broad area of secure function evaluation [22–24]. Several protocols in this space resemble and appear applicable to the CBPS problem. However, none of these is of practical importance for C-CBPS. First, the protocols have been designed for single invocations and are vulnerable when the same key is used to send multiple notifications (which is a necessity in publish/subscribe). For instance, the information-theoretically secure protocol described by Ishai [23] can be broken easily when used for multiple messages, while the semantically secure protocol described by Feige [22] becomes as secure as the one time pad in the same context. In theory, we can use such single message protocols in the context of publish/subscribe, but with tremendous overhead: For *every* published notification, the publisher and all subscribers would generate a new key, the subscribers would then register their subscriptions, and finally the publisher would send the notification. Secondly, even the cheapest instances of these protocols has high costs for single invocations.

Privacy Preserving Keyword Searches. Motivated by public file servers and email servers, a more practical approach was taken by the security community to solve the problem of searching encrypted files using keywords.

The pioneering work in this direction is due to Song et al. [12], who propose a scheme that encrypts each word in the document in a way that allows a user to search using an encrypted keyword. To test whether a given keyword is in an encrypted file, a sequential scan of the file is needed; this approach does not scale well for large documents. Schemes were proposed by Goh [13] and Chang et al. [15] that use indexes to address this issue and propose stronger security models. For practical reasons, we used the first scheme for keyword search and the second as a basis for supporting range matches. Our work employs a security model that is similar to the one from Chang et al. [15], extended to deal with arbitrary subscriptions and to allow subscription covering (that was implicit in the initial model). Our mechanisms can be used to provide privacy preserving range matches for numeric values.

Security in CBPS. Security in publish/subscribe was first analyzed by Wang et al., acknowledging the new difficulties posed by this interaction model [4]. Security requirements are identified as integrity, confidentiality and availability at application level, and integrity and availability at infrastructure level. Our work presents solutions for application level confidentiality, addressing both notification and subscription confidentiality.

Li et al. [25] address the same issue of achieving subscription and notification confidentiality in CBPS systems. They support range matching as selection criteria and encode ranges by transforming them into several prefix matching problems. The notifications are encrypted by using prefix-preserving encryption. Matching is reduced to checking whether an encrypted notification contains one of the desired prefixes. This scheme has distinguishable notifications and is not secure according to our model. Although it can be modified to have computationally indistinguishable notifications, the modified scheme still leaks

more information than necessary (due to prefix matching) and is still not secure.

A solution that supports equality matches using Bloom filters is implemented in the Siena Fast Forwarding Module [26]. This solution has distinguishable notifications, being insecure according to our model.

A comprehensive solution to security in publish/subscribe is proposed by Srivatsa et al. [27]. Confidential CBPS is only supported for equality matches through the direct use of pseudorandom functions and is not secure according to our model.

VIII. Summary and Future Work

This paper presents a study of confidentiality in content-based publish/subscribe, addressing some of the security concerns particular to this interaction model. We have presented a formal security model and analyzed the general C-CBPS problem, pointing out its inherent limitations.

We have described provably secure techniques that allow content-based routing for the large majority of applications occurring in practice. We have described two novel protocols that support range matches in C-CBPS but can also be applied in other areas, such as privacy preserving range matching.

We have implemented these mechanisms in a popular CBPS infrastructure, making available the first implementation of C-CBPS. To assess the performance impact of C-CBPS, we have evaluated our solutions against their plaintext counterparts. Results show that achieving confidentiality is practical, a broker being able to match 100 notifications per second when it has 1000 subscribers.

In future work, we intend to develop additional security mechanisms that offer support for other as yet unforeseen types of subscriptions that might be encountered in practice.

Acknowledgements

We thank Yvo Desmedt for scrutinizing the security protocols and proofs, and for discussions and thoughtful reviews on previous versions of this paper; Antonio Carzaniga and Alex Wolf for fruitful discussions of these ideas; and the anonymous referees for their detailed feedback. Costin Raiciu is supported by a UCL Departmental Studentship. David Rosenblum holds a Wolfson Research Merit Award from the Royal Society. This work was partially supported by the European IST FET programme in project SENSORIA (IST-2005-016004).

References

[1] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf, "Design and evaluation of a wide-area event notification service", *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, 2001.

[2] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman, "An efficient multicast protocol for content-based publish-subscribe systems", in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, ICDCS*, Washington, DC, USA, 1999, p. 262, IEEE Computer Society.

[3] Akamai, "http://www.akamai.com".

[4] Chenxi Wang, Antonio Carzaniga, David Evans, and Alexander Wolf, "Security issues and requirements in internet-scale publish-subscribe systems", in *Proceedings of Hawaii International Conference on System Sciences*, 2002.

[5] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, "The many faces of publish/subscribe", *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[6] Oded Goldreich, *Foundations of Cryptography*, vol. Basic Tools, Cambridge University Press, 2001.

[7] Sandro Rafaeli and David Hutchison, "A survey of key management for secure group communication", *ACM Comput. Surv.*, vol. 35, no. 3, pp. 309–329, 2003.

[8] Antonio Carzaniga and Alexander L. Wolf, "Forwarding in a content-based network", in *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003, pp. 163–174.

[9] D. Chaum, "The dining cryptographers problem: unconditional sender and recipient untraceability", *Journal of Cryptology*, vol. 1, no. 1, 1988.

[10] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router", in *Proceedings of the 13th USENIX Security Symposium*, 2004.

[11] Amos Fiat and Moni Naor, "Broadcast encryption", in *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, New York, NY, USA, 1994, pp. 480–491, Springer-Verlag New York, Inc.

[12] Dawn Song, David Wagner, and Adrian Perrig, "Practical techniques for searches on encrypted data", in *Proceedings of the IEEE Symposium on Security and Privacy*, 2000.

[13] Eu-Jin Goh, "Secure indexes", Cryptology ePrint Archive, Report 2003/216, 2003.

[14] Burton H. Bloom, "Space/time trade-offs in hash coding with allowable errors", *Communications of the ACM*, vol. 13, no. 7, 1970.

[15] Yan-Cheng Chang and Michael Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data.", in *ACNS*, 2005.

[16] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano, "Public key encryption with keyword search", in *EUROCRYPT*, 2004.

[17] Bijit Hore, Sharad Mehrotra, and Gene Tsudik, "A privacy-preserving index for range queries", in *Proceedings of VLDB - Conference on Very Large Databases*, 2004.

[18] Andrew C. Yao, "How to generate and exchange secrets", in *Proceedings of the IEEE Symposium of Foundations of Computer Science, FOCS*, 1986.

[19] Bill Segall, David Arnold, Michael Henderson Julian Boot, and Ted Phelps, "Content based routing with elvin4", in *Proceedings of AUUG2K*, 2000.

[20] National Institute of Standards and Technology, "Secure hash standard", 2002.

[21] Joan Daemen and Vincent Rijmen, *The design of Rijndael: AES — the Advanced Encryption Standard*, 2002.

[22] Uri Feige, Joe Killian, and Moni Naor, "A minimal model for secure computation (extended abstract)", in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, STOC*, New York, NY, USA, 1994, pp. 554–563, ACM Press.

[23] Yuval Ishai and Eyal Kushilevitz, "Private simultaneous messages protocols with applications", in *Proceedings of Israel Symposium on Theory of Computing Systems*, 1997, pp. 174–184.

[24] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan, "Private information retrieval", in *Proceedings of IEEE Symposium on Foundations of Computer Science, FOCS*, 1995, pp. 41–50.

[25] Jun Li, Chengluai Lu, and Weidong Shi, "An efficient scheme for preserving confidentiality in content-based publish/subscribe systems", Tech. Rep. GIT-CC-04-01, Georgia Institute of Technology, 2004.

[26] "Siena documentation", http://www.cs.colorado.edu/ serl/cbn/forwarding/.

[27] Mudhakar Srivatsa and Ling Liu, "Securing publish-subscribe overlay services with eventguard", in *Proceedings of the 12th ACM conference on Computer and communications security*, 2005.

Appendix

It is easy to see from the descriptions that all the schemes we propose correctly match subscriptions against notifications and conservatively solve subscription coverage (i.e., they can give false negatives, but not false positives). We have also experimentally tested the correctness of our schemes. Henceforth, the proofs only analyze the security of the proposed schemes.

Theorem 1: Equal is a correct implementation of C-CBPS

Proof. We want to show that for any k, any function h and any algorithm A (i.e., running at the broker), there is an algorithm A^* (i.e., running with access to the oracle) such that the following value is negligible in t: $\delta = |Pr[A(View_k, 1^t) = h(Plaintext_k)] - Pr[A^*(View_k^*, 1^t) = h(Plaintext_k)]|$.

The idea, borrowed from Chang et al. [15], is to prove that A^* can use $View_k^*$ to construct a view $View'_k$ that is computationally indistinguishable from $View_k$. If this is the case, A^* can simulate the desired functionality by calling A with parameter $View'_k$, and therefore δ is negligible.

Without loss of generality, assume that the C-CBPS protocol consists of two consecutive phases: *registration* (consecutive rounds in which subscribers send their interests to the broker) and *operational* (consecutive rounds where publishers send notifications to the broker). It is simple to see that if the protocol is secure in this case, it is also secure when subscription registrations and notifications are interleaved. Assume $Plaintext_k = \{N_1, \ldots, N_k, S_1, \ldots, S_n\}$, that is, the k^{th} round in the *operational* phase. Then, $View_k$ is $\{(rnd_1, f_{f_K(N_1)}(rnd_1), \ldots, (rnd_k, f_{f_K(N_k)}(rnd_k), f_K(S_1), \ldots, f_K(S_n)\}$.

Let us consider the special cases first. Assume $n = 0$, that is, there are no subscriptions. A^* selects all entries in $View'_k$ (corresponding to encrypted notifications) uniformly at random. In this case, A^* simulates A properly, otherwise we can use (A, A^*) to distinguish pseudo-random bits from random bits.

Next, assume $k = 0$, meaning that no notifications have been received yet. In this case, A^* proceeds as follows. For each $i = 1 \ldots n$, check to see if there exists $j < i$ such that O_0.*cover*(j,i)=1. If such j does not exist, select subscription S_i in $View'_0$ uniformly at random. Otherwise, set $S_i = S_j$.

A^* feeds this view to A. The only difference between $View_k$ and $View'_k$ is the way the distinct subscriptions are chosen. We claim that whatever A can compute from $View'_k$ can also be computed using $View_k$; otherwise the pair (A, A^*) can be used to distinguish pseudo-random bits from truly random bits.

Now consider the general case. A^* generates n subscriptions as described above and adds them to $View'_k$. Let $S_d = S_1, \ldots, S_m$ be the set of independent subscriptions. Next, A^* generates k notifications as follows.

For all $i = 1, \ldots, k$ A^* checks if there exists $j \in \{1, \ldots, k\}$ such that O_k.*match*(i,j)=1. If so, A^* generates a random nonce rnd and adds $(rnd, f_{S_j}(rnd))$ to $View'_k$; otherwise it adds a value selected uniformly at random.

There are two differences between $View_k$ and $View'_k$: a) distinct subscriptions are pseudo-random as opposed to truly random, and b) notifications that are not matched by the distinct subscriptions are generated truly randomly instead of pseudo-randomly (i.e., using f). Therefore, if A can compute something more from $View_k$ we can use it to distinguish pseudo-random bits from random-bits. This concludes the proof.

Theorem 2: Keyword is a correct implementation of C-CBPS

Proof Sketch. The paper by Goh [13] presents a proof of security under the IND-CKA2 model, which focuses on notification indistinguishability. Here we show that breaking C-CBPS security for *Keyword* can be used to break IND-CKA2 security for *Keyword*, and therefore IND-CKA2 security implies C-CBPS security for keyword matching.

The attacker in the IND-CKA2 game selects uniformly at random n distinct keywords $\{S_1, \ldots, S_n\}$ and finds out their encrypted versions by using the IND-CKA2 challenger. The attacker further selects two plaintext documents uniformly at random, N_0 and N_1, ensuring that the known keywords are contained by both N_0 and N_1 or by neither. N_0 and N_1 are passed to the challenger in the IND-CKA2 game, which replies with an encryption of N_b where b is uniformly random from $\{0, 1\}$.

Let us assume that the attacker can compute a functionality h given $View_1 = \{S_1, \ldots, S_n, N_b\}$, that cannot be computed only using $View_1^*$. If h does not depend on the value N_b, then h can compute something relating to the subscriptions, besides the coverage relation; by using an argument similar to Theorem 1, we can see that this will allow one to distinguish random bits from pseudo-random bits, and is therefore impossible. Therefore, it must be that h depends on N_b, meaning that h will present non-negligible distinct outputs for $b = 0$ and $b = 1$. The attacker uses this output to guess the value of b, therefore winning the IND-CKA2 game. This completes our proof sketch.

Theorem 3: Dictionary is a correct implementation of C-CBPS

Proof Sketch. Definition 1 provides a security model for C-CBPS regardless of the subscription function, by mandating that the information the broker can learn by using the messages received from the publishers and subscribers can also be learnt by accessing an oracle. The security model provided by Chang et al. [15] is an instance of our model, where the subscription function is keyword matching and the oracle is replaced by access to the actual information (i.e., which document contains which keyword). The difference between their model and ours is the treatment for subscriptions (keywords). They assume that all keywords are different (and therefore no information is gained by seeing they are different), while we allow the broker to distinguish whether one subscription covers another subscription. In the case of keyword matching, two subscriptions cover one another only if they are equal. If we only consider the subset of distinct subscriptions, we can directly use the security proof in Chang et al. [15] to prove security in C-CBPS. The redundant subscriptions do not leak any additional information about notifications, and do not leak more information about subscriptions that cannot be discovered by using the oracle. Therefore, *Dictionary* is C-CBPS secure.

Theorem 4: If all subscriptions are expressed exactly, *Inequality* is a correct C-CBPS implementation

Proof Sketch. *Inequality* is an instance of *Dictionary* that contains as words "$> p_1$", "$> p_2$", ..., "$> p_l$" "$< p_1$", "$< p_2$", ..., "$< p_l$". Since the approximation is assumed to be perfect and *Dictionary* is secure (Theorem 3), verifying inequality using the dictionary gives as much information as verifying with the oracle. It follows that *Inequality* is also secure.

Note that the assumption that subscriptions are expressed exactly is important. Without this, the broker can infer additional information. Here is a simple example: assume the notification space is $0, \ldots, 10$ and the reference points are $0, 5, 10$. Subscription $S = x > 7$ will be approximated with $S_a = x > 5$. Given encrypted notifications 4 and 6, the broker cannot distinguish them in the ideal case (when testing against S, none of them is matched), however it can tell they are different in reality (as S_a will match 6 and not match 4).

Theorem 5: If all subscriptions are expressed exactly (i.e., without generating false positives or negatives) , *Range* is a correct C-CBPS implementation

Proof Sketch. The same reasoning applies as before.