Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures

Costin Raiciu and David S. Rosenblum
London Software Systems
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom
{c.raiciu|d.rosenblum}@cs.ucl.ac.uk

Abstract

Content-based publish/subscribe 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. Much attention has been given to finding scalable distributed algorithms for this interaction model, while security has received considerably less scrutiny from the research community. In this paper, we focus on answering the following question: Can we implement content-based forwarding while keeping subscriptions and notifications confidential from the forwarding routers? Our contributions include an analysis of the problem, showing that the maximum level of attainable security in our setting is quite restricted, and that even this level is expensive to obtain for generic subscription functions. We focus on enabling confidentiality for commonly used applications and subscription languages in content-based publish/subscribe and propose a series of practical solutions that we have embedded in the implementation of Siena [3], a popular content-based pub/sub system. Evaluation results show that confidential content-based forwarding is practical, having acceptable overhead when compared to its plaintext counterpart.

Keywords: confidentiality, content-based forwarding, publish/subscribe, security

1 Introduction

Content-based publish/subscribe is a convenient interaction model for distributed systems, allowing decoupled messaging 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 content-based publish/subscribe *routers* 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 delay.

Research in the pub/sub area has traditionally focused on the scalability issues of publish/subscribe networks, yielding distributed algorithms that enable wide-area event notification and matching by using infrastructures comprising a mesh of pub/sub routers [3, 1].

An implicit assumption underlying this research has been that the forwarding routers 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 routers. However, we expect content-based publish/subscribe to be enabled on top of existing infrastructures of third party providers (content-based equivalents of Akamai), 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 resort to this technology and use an existing provider to disseminate data to home users.

So far, security appears to be an Achilles' heel in these systems, delaying their wide-spread adoption. Recently, we are witnessing an increasing interest in pub/sub security from the research community [20, 18, 14, 15, 2, 9]. We share the belief that the interaction model proposed by content-based publish/subscribe routing will not be widely accepted in practice unless strict security properties can be guaranteed, and we think that, although major steps have been achieved, security in publish/subscribe is still far from being a tangible goal.

In this paper, we therefore focus on a particular aspect of security in content-based publish/subscribe, *confidentiality*, complementing existing work in the field. Confidentiality comes in two flavors in content-based forwarding [20]:

- *Notification confidentiality:* The stock quote provider will be reluctant to disclose any stock information to the infrastructure, if it believes that some routers could re-sell or otherwise exploit this information (possibly with the purpose of making a profit).
- *Subscription confidentiality:* Paying subscribers would like to keep their subscriptions private from the forwarding routers, as these interests might leak their business strategy.

Creating solutions to deal with these aspects is paramount if content-based pub/sub is to be adopted as a solution for data dissemination. However, the very nature of pub/sub—targeting a sweet spot in the space of plain IP-multicast (when subscribers can be approximately clustered into a small number of groups based on their 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 be characterized by strict performance requirements combined with tight security guarantees.

This paper provides a study of confidentiality in the context of content-based forwarding and presents our solution, being the first work to fully address this issue. The contributions of this paper include:

1. a precise definition of confidentiality for publish/subscribe, along with a brief discussion of the limitations of the concept, from both a theoretical and technical point of view;
2. several techniques that enable confidential content-based forwarding for commonly used types of subscriptions and notifications, including a novel protocol based on Yao's garbled circuit construction [21] that allows the use of *arbitrary* functions (represented as boolean circuits) as a subscription, has low communication overhead, and achieves security equivalent to the one-time pad for multiple messages (see [11] for details);
3. an implementation of our solutions in a popular content-based pub/sub solution, SIENA [3], making available to the community the first complete and practical implementation of pub/sub that supports confidentiality; and
4. evaluation results demonstrating that our solutions are lightweight enough to be suitable for usage in real applications.

This paper is organized as follows: Section 2 provides background information on content-based pub/sub. Section 3 presents a definition of the content-based pub/sub problem and discussions of its limitations and connections to existing protocols. In Section 4 we describe several protocols that enable confidential content-based forwarding. Section 5 discusses how these protocols are bundled into a complete solution and implemented in SIENA, with a security analysis of the resulting solution presented in Section 6. Section 7 presents experimental results, and Section 8 reviews existing work in the area. We conclude in Section 9 with a summary of our contributions and future directions of research.

2 Background

One of the distinctive characteristics of the pub/sub interaction model is loose coupling: spatial decoupling, temporal decoupling and flow decoupling [6]. These make pub/sub systems ideal for interactions between a potentially unlimited 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 pub/sub routers. The infrastructure delivers to the subscribers any published notification that matches their registered interests. In content-based pub/sub, 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*) [3].

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" and exchange= "NYSE"*.

3 Problem Statement and Implications

In the ideal scenario where routers are trustworthy, publish/subscribe networking offers many advantages over traditional broker-based event dissemination middleware, mostly because of its wide-area scalability [3, 1]. However, if the routers in the network are not trusted, the dissemination model becomes unsuitable for security-sensitive applications. This can arise in cases when publishers and subscribers wish to exchange data over a public, un-trusted, network, but at the same time benefit from the advantages of content-based routing.

3.1 Security Assumptions

We assume that the publishers and subscribers are trustworthy and that the communication links between participants are secure, ensuring message secrecy, authenticity, and replay protection. This requirement can easily be achieved by using established techniques such as IPSec [13] for each hop of the communication: publisher–router, router–router, router–subscriber.

We assume that the publishers and subscribers share a secret key they will use for confidential content-based forwarding of messages. Clearly, it is difficult to prevent a malicious subscriber from disseminating all the information it receives to other parties. This is outside the scope of this work.

3.2 Adversarial Model

Our main assumption is that routers are semi-honest, following the protocol as prescribed but trying to infer additional information just by recording and processing the messages they route. Also, routers are assumed to be computationally bounded. If routers were allowed to deviate from the content-based forwarding protocol, denial of service attacks could be mounted easily, affecting the correct operation of the infrastructure. Byzantine agreement protocols could be combined with multipath routing to detect and exclude misbehaving routers from the infrastructure. Here, we do not consider the effects of this type of behavior, leaving it as an issue for future research.

In this work, we do not consider external adversaries. The assumption that publishers and subscribers are trusted and that they share a secret key creates the basis for authenticated communication between them, ensuring that malicious adversaries are kept outside.

3.3 Problem Statement

Informally, subscription confidentiality prevents the router from finding out the contents of a subscription, while notification confidentiality prevents disclosure of notification information. A close inspection of the above two properties yields the following result.

Proposition 1 *Subscription and notification confidentiality can only be enforced concurrently.*

If notifications are sent in plaintext, the router can create arbitrary notifications and match them against the encrypted subscription. Combined with regression techniques, random sampling of the notification space could reveal an approximation of the subscription function with minimal effort. Conversely, if notifications are encrypted, a malicious router can implement a form of binary search through carefully chosen subscriptions to discover the value of the notification in logarithmic time.

Therefore, the major security requirement is *simultaneous notification and subscription confidentiality* such that a publish/subscribe router is able to determine if a notification matches a subscription without being able to infer other information about either the notification or the subscription.

Definition 1 (Confidential Content-Based Publish/Subscribe) *Consider two types of parties P (publishers) and S (subscribers) having private inputs. Each P_i has as sequence of notifications, and each S_j has some information interests (filters), a subset of which is active at any point in time. All S_j periodically send their active filters to a semi-honest third party, the router R. Each P_i independently sends its sequence of notifications to R. R must be able to determine which active filters match each notification, without learning anything more about the filter(s) and notification(s) than is implied by the outcome of the match and the history of previous matches in the system. The outcome of a match is binary, and upon a true match R forwards the notification towards the subscribers associated with the matching filter.*

The definition above generalizes naturally to the multi-router case where the number of forwarding routers is arbitrarily large.

While the notifications are essentially data, filters are selection criteria applied to the data values, such as range matches for integers or regular expressions for strings. The expressiveness of filters varies somewhat with different pub/sub systems, and it is often possible to combine filters into more complex subscriptions using logical conjunction, disjunction and negation. The definition above imposes no constraints on the expressivity of the filters, which in the general case can be any polynomial time function.

3.4 Limitations of Security

The maximum level of attainable security in content-based forwarding is quite limited. Here we present a brief overview of these limitations, deferring detailed analysis and proofs to an extended version of this paper, due to space limitations.

At a high level, achieving confidentiality for both subscriptions and notifications requires that both subscribers and publishers share a common secret. Simply put, the parties have to create a group key before content-based publish/subscribe interaction commences. Thus, the complete decoupling of content-based publish/subscribe is reduced by the necessity for the publishers and subscribers to use a key distribution channel.

A solution achieving confidentiality in content-based publish/subscribe according to Definition 1 must ensure that the router is not able to infer more information about the subscription(s) and notification(s) than is implied by the outcome of the match. 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 router. One way to mitigate this is to use anonimizing, trusted proxies acting as intermediaries between the routers and the publishers and subscribers, respectively.

Theorem 1 *The encryption schemes used for notifications and subscriptions cannot have "indistinguishability" for multiple messages.*

The proof, omitted due to lack of space, is based on the fact that in non-trivial cases different subscriptions/notifications can be distinguished when enough notifications/subscriptions are available. For instance, routers can infer the relation between the subscriptions they store by analyzing the outcome of the match for all incoming notifications. Furthermore, if the router knows some characteristics of the notifications (such as their approximate distribution), it can infer what the relation between two subscriptions is with increasing statistical precision as the number of matched notifications increases. The same considerations hold for notifications, when the number of subscriptions increases.

Corrolary 1 *The most secure encryption notification scheme for confidential content-based forwarding is a 1-to-1 encryption of the plaintext (i.e. every plaintext can be encrypted to a single ciphertext and viceversa, and different plaintexts have associated different ciphertexts).*

The proof of the corollary follows directly from the fact that messages can be distinguished.

Theorem 2 *Every function computable in polynomial time has a protocol as secure as Corrolary 1. The size of the subscription may, however, be exponential in the size of the notification.*

It is simple to create such a protocol: select a random permutation of a truth table representation of the function, and encrypt every possible input by using a symmetric encryption scheme (or a one-way function). Matching at the router is performed by looking for the proper entry (given the notification) in the encrypted and permuted truth table.

Theorem 3 *Protocols that implement confidential content-based forwarding for arbitrary functions and have the size of the subscriptions bounded by a polynomial in n (where n is the size of the notification) must allow the router to distinguish the encryptions of k sequences of bits in the notification, where k is a constant.*

Here is a sketch of the proof. Assume a subscription accepts all notifications that have the m'th bit equal to 0, where $1 \leq m \leq n$. If only n-bit sequences can be distinguished, the subscription must contain all the 2^{n-1} accepted inputs. If, in general, encryptions of k-bit substrings can be distinguished, then the subscription will contain 2^{k-1} entries. Therefore, to have subscriptions with sizes polynomially bounded, the number of distinguishable bits must be independent of n.

This is a strong negative result: scalably supporting generic functions (with respect to the size of the input, n) only allows a reduced level of security.

3.5 Candidate Solutions

Research in cryptography has produced many important results in the broad area of secure function evaluation [8, 12, 5]. Several protocols in this space have resemblance and appear applicable to the content-based publish/subscribe problem; a more detailed description of these is omitted due to lack of space. However, none of them is truly applicable to our setting, because they all share two major shortcomings: first, they are designed to work for single invocations, and second, they have high cost.

These protocols have been designed for single invocations and are vulnerable when the same key is used to send multiple messages (which is a necessity in pub/sub). We could theoretically apply the single message protocol in the context of pub/sub, 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. For instance, the information-theoretically secure protocol described by Ishai [12] can be easily broken when used for multiple messages, while the semantically secure protocol described by Feige [8] becomes as secure as the one time pad in the same context.

Moreover, the cheapest instances of these protocols still have prohibitively high costs. One protocol we have investigated—proposed by Ishai [12]—has $O(n^2)$ communication complexity, where n is the size of the attribute. The proportionality constant hidden behind the big-O notation is quite big, being the square of the size of the alphabet for string matching, with communication complexity as high as $900 \cdot \mathit{length}(\mathit{string})^2$, which is impractical even for small strings (e.g., a 50-byte string becomes 2MB encrypted). The other protocol we have considered, by Feige et al. [8], sends $k \cdot n$ bits for a notification represented on n bits. The value k is a security parameter, with a usual value of 80. In this case, a 4-byte integer will be encoded as 320 bytes, which is very expensive if messages are frequent.

3.6 Requirements

The theoretical limitations of confidentiality combined with technical difficulties in achieving them steers us to look for solutions that can be used in situations where we can relax the security and generality requirements:

- *Subscription language:* Our schemes should efficiently support commonly used subscriptions, along with (some) support for more general subscriptions.
- *Flexibility:* Since a one-size-fits-all protocol with maximum security and small overhead does not exist (according to Theorem 3), a variety of encryption schemes should be supported with different security-to-complexity ratios, to allow applications to choose the most suitable scheme for their requirements.
- *Acceptable level of security:* Common application security requirements should be met. Clearly, it is not recommended to deploy security critical applications (such as military communication) over content-based pub/sub solutions: even the maximum achievable security in this setting does not suffice for such applications.

- *Efficiency:* Supporting confidentiality increases the costs associated with content-based routing on two counts: *local content-based matching time* and *network transmission time.* For a system to be usable, it must minimize these costs and place them at a point where the obtained performance is above some minimal, previously known threshold.

4 Solutions

There are different high level approaches to our problem. We can consider that notifications are encrypted as a whole and fed into an encrypted subscription function. Alternatively, we could consider that notifications are composed of *name=value* pairs where only the values are encrypted, with subscriptions decomposed into encrypted filters on individual attributes; the semantics of such a subscription is the conjunction of its filters.

We consider the latter approach, as it is likely to generate simpler, more efficient constructions; the fact that some structure is leaked to the router is acceptable in most cases. In this way, we can support rather complex subscriptions by using basic filtering mechanisms as building blocks. In the upcoming sections, a notification denotes a single attribute value, and a subscription is a single constraint on the notification, unless stated otherwise.

We present several mechanisms aimed at dealing with most common subscription types (equality tests, range matches and keyword matching), and we introduce a more general protocol, LSCP, that allows arbitrary filtering, but has slightly higher overhead and offers less security (a direct consequence of Theorem 3). The conjunction of these mechanisms enables practical confidentiality for the majority of applications using content-based forwarding.

4.1 LSCP: A Low-Security, Cheap Protocol

To partially achieve the requirements specified in Section 3.6, we present the Low Security Cheap Protocol (LSCP). Its main features are support for arbitrary selection functions (represented as boolean circuits), and zero notification overhead. The overhead of the protocol depends on the complexity of the boolean circuit (the number of gates) and is reflected in increased subscription propagation costs and increased time required to match notifications against subscriptions. Evaluation results presented in Section 7 show that the protocol has acceptable overhead for most subscriptions used in practice.

The LSCP protocol is simple. Using the shared secret, the subscriber will "garble" the circuit representing its subscription and send it to the router, while the publisher will encrypt the notification in a way compatible with the subscriber's circuit. At the router, the encrypted notification is input to the subscription circuit, and the boolean result of the output wire is used to decide whether the notification should be forwarded toward the subscriber.

4.1.1 Encrypting Notifications

In the setup phase, the publisher(s) will generate a random string K_P (with length equal to the length of the notification) by feeding the secret key as seed into a pseudo-random number generator.

When a notification N is ready for transmission, the publisher(s) will compute the $E_N = N \oplus K_P$, producing the encrypted notification, E_N. Next, E_N is sent to the forwarding router.

4.1.2 Encrypting Subscriptions

Each subscriber has a boolean circuit representation of each subscription function.

Each subscriber associates a random masking bit with each wire in the circuit, using a pseudo-random number generator with the secret key as seed. The random masking bits assigned to the input wires must be the same as the bits in the random string generated by the publisher, K_P. The unique output wire is always unmasked so that the router can determine the outcome of a matching operation.

The subscriber proceeds to "garble" the gates of the circuit such that the subscription function is not leaked to the router. The process consists of sequentially garbling all the gates in the circuit. Garbling a single gate has two parts:

1. Flip - the values in the truth table of the gate are negated if the random bit associated to the output wire has value 1, or left unchanged otherwise, obtaining the *randomized truth table*.
2. Permute - the positions of values in the truth table are permuted according to the values of the random bits associated to the gate's input wires. More exactly, assume we have k input wires for a gate G, and assume that the random bit associated with input wire i is b_i. We iterate all the 2^k entries in G's randomized truth table, storing the entry corresponding to $a_k a_{k-1} \cdots a_0$ (i.e. situated at index $a_0 + a_1 \cdot 2 + \cdots + a_k \cdot 2^{k-1}$ in the truth table) at position $(a_0 \oplus b_0) + (a_1 \oplus b_1) \cdot 2 + \cdots + (a_k \oplus b_k) \cdot 2^{k-1}$.

Here is a simple example. Assume we have a gate G with arity 2 and the following truth table, and the random bits assigned to the input wires are $w_1 \leftarrow 0$, $w_2 \leftarrow 1$ and the output wire $w_3 \leftarrow 1$.

$$\begin{array}{cc|ccc|ccc|c}
w_1 & w_2 & w_3 & & w_1 & w_2 & w_3 & & w_1 & w_2 & w_3 \\
\hline
0 & 0 & 0 & \longrightarrow & 0 & 0 & 1 & \longrightarrow & 0 & 1 & 0 \\
0 & 1 & 1 & \textit{flip} & 0 & 1 & 0 & \textit{permute} & 0 & 0 & 1 \\
1 & 0 & 1 & & 1 & 0 & 0 & & 1 & 1 & 0 \\
1 & 1 & 1 & & 1 & 1 & 0 & & 1 & 0 & 0
\end{array}$$

The initial output table is xored with the bit associated to the output wire, w_3 (denoted as *flip*). Then, it is permuted according to the bits associated to the input wires w_1 and w_2 (*permute*). The description of the table sent to the router only includes the enumeration of values of the output wire. Thus, given input 0 1 (i.e. input 0 0 in cleartext) the router will select the second value in the table, 1 (corresponding to plaintext output 0). It is easy to check that for any combination of encrypted input, the encrypted output will be correct.

4.1.3 Complexity

Computational complexity: Encrypting a notification is cheap: a simple xor operation. Encrypting a subscription is an iterative garbling of all the gates in the system, while garbling a gate is a single pass over its output table. The computational complexity of matching is the same as the complexity of evaluating a regular circuit, iterating all the gates in topological order and evaluating them. However, subscriptions are usually expressed in higher level forms (such as relational operators, etc.) that are more efficiently executed. Measurements show that this overhead is acceptable.

Communication complexity: Communication overhead is zero for notifications. The overhead for subscriptions is due to the circuit representation; subscriptions represented in higher-level forms occupy significantly less space. Although polynomial, this overhead is considerable for some simple functions, such as multiplication. The evaluation section presents typical sizes of subscriptions used in current systems, concluding that this overhead is acceptable in most cases.

4.1.4 Discussion

This protocol is, in fact, a simplified version of the Yao's garbled circuit construction [21], used in the manner described by Feige [8] and adapted to work for multiple messages. The simplification itself does not significantly weaken the obtained security [1], but it produces important efficiency gains: compared to the initial construction, local content-based matching speed for our protocol is 5 times faster in our tests, and communication complexity is reduced by a factor of k, where k is the security parameter (with a typical value of 80).

The LSCP protocol permits the encryption of any subscription function, although with potentially high communication overhead, due to the blowup in the number of gates. The security it obtains might not be satisfactory in some cases. Here, we present complementary mechanisms efficiently supporting restricted types of subscriptions and ensuring better privacy at the same time. These mechanisms are therefore preferable for cases where their expressivity suffices for the space-efficient description of subscribers' interests.

[1]The proof is omitted due to space limitations.

4.2 Supporting Equality Filtering

In cases where the number of values an attribute takes is small or when the subscriptions are equality checks, we use a straightforward solution to enable content-based forwarding: symmetric encryption.

The publisher uses a deterministic symmetric encryption algorithm, parameterized with the shared secret key, to encrypt the notification it wishes to send. The subscriber will similarly encrypt the values it wishes to test for equality, and send them to the router. The latter will simply match the encrypted notifications it receives against the encrypted subscriptions it has stored, forwarding when the ciphertexts match. If notifications are expected to be reasonably large in size, we can speed up content-based matching by trading off some communication efficiency: Each encrypted notification will include a keyed hash (typically 180 or 256 bits), and the subscriptions will consist of the keyed hash of the expected input.

4.3 Supporting Range Matches

Range matches are a common level of expressiveness provided in many implementations of pub/sub for content-based filtering of numeric values. LSCP can also support range matches, but with slightly higher overhead.

A protocol proposed by Lu et al. transforms range matches into sets of prefix matches and uses prefix-preserving encryption to test subscription matching [15]. Range matches can be transformed into a set of at most $2 \cdot n$ prefix matches, where n is the size of the interval being matched [15]. In simple cases, the number of prefixes is quite small, making this protocol efficient on average. The size of the encrypted notifications is equal to the plaintext size, as in LSCP.

In most cases, this protocol is more efficient than LSCP for range filtering. Also, as discussed in Section 6, the protocol is more secure.

4.4 Supporting Keyword Searches

Substring matching is the most expressive operation currently implemented on strings in common content-based publish/subscribe architectures. It can be supported with the LSCP protocol, but with high costs: The binary representation of a subscription for a substring in a 50-character string is as large as 40KB, requiring thousands of gates for its circuit representation.

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 [10]. The idea is quite simple: We break the string into words and construct a secure Bloom filter to signal existence of a word in the string; the string itself is encrypted by using regular symmetric encryption and is not used in the matching process.

The construction proceeds as follows: The publisher creates a list of words for the string and creates a corresponding Bloom filter. By using the secret key, it selects an appropriate number of cryptographic hash functions and applies them sequentially to all the words, to obtain the "trapdoor" for each word. The trapdoor is then cryptographically hashed again using a unique string identifier (which can be a random nonce), and the resulting values are stored in the Bloom filter. The filter and the string identifier are sent to the router.

The subscriber's interest consists of a single word or a collection of words it wants to match against the string. It will independently create the trapdoor (using the secret key) and send it to the router.

Presented with a string identifier, a Bloom filter (from the publisher) and a trapdoor (from the subscriber), the router will cryptographically hash the trapdoor using the unique string identifier and check the Bloom filter to see if the word is present.

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.

5 Implementing confidential content-based pub/sub

The mechanisms described above have been implemented in SIENA [3], a wide-area event-notification service. SIENA is a content-based pub/sub infrastructure where brokers are vertices in a connected overlay acyclic graph. Notifications are sets of $(attribute_{name} = attribute_{value})$ pairs and subscriptions are a conjunction of their filters, each applied to individual attributes. Each filter is a tuple specifying the $attribute_{name}$ and the desired constraint: this can be any of the usual relational operators for numeric values and substring matching for string values.

We choose Siena due to its popularity and widespread adoption within the research community. Our proposals can be embedded easily in other content-based pub/sub solutions (such as Gryphon[1], Elvin[19], etc.).

5.1 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 constraint, that contains, besides the attribute name, a serialization of the encrypted subscription, encoded either using LSCP, prefix-preserving encryption or keyword matching. The encrypted subscription is embedded as a byte array (already supported by Siena) to avoid modifying marshalling code in the original implementation. We created a encrypted matcher manager object that 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. Modifications to the basic Siena code are minimal: altogether, 15 lines of code were added to the original implementation.

To create the circuits corresponding to the subscription function, we use a compiler developed by Malkhi et al.[16]. The subscriptions are expressed in a high level language called SFDL and compiled into a binary circuit that is then used as input for our LSCP.

We used the SHA-256 cryptographic hash [17] function throughout our implementation. In some points it is used as a pseudo-random number generator using as seed the secret key (i.e. for generating the bits corresponding to the wires in the circuit) while in others is used as a keyed hash function (keyword encryption), and also as a basis for a symmetric encryption scheme.

5.2 Encoding attribute names and types

It might be desirable to hide the names of the attributes from the routers in some applications. Also, some of the mechanisms require that the size of the attribute values are pre-set to function correctly. To these ends, the name of the attribute is concatenated with its type and size and cryptographically hashed using the secret key as additional input. The Siena implementation only attempts to match two attributes if their names are equal; thus, the existing code will work correctly with encrypted attribute names.

5.3 Subscription covering

Subscriptions have a natural partial ordering relation based on the set notifications they match. If subscription S_1 accepts a superset of the notifications S_2 accepts, S_1 is said to *cover* S_2. Exploiting this relation, Siena inhibits the propagation of subscriptions towards neighbors where more general subscriptions have already been propagated. This reduces the broker-side memory and processor overhead (as each broker has to maintain a smaller list of subscriptions locally), as well as communication overhead.

Implementing subscription covering brings important efficiency gains, increasing the overall scalability of the solution without reducing the security (see Theorem 1).

We have implemented subscription coverage for all our proposals. Keyword matching is simple to test: S_1 is covered by S_2 if all S_1's encrypted keywords are contained in S_2's list of encrypted words. Covering for exact matches is performed by checking whether items in S_1's subscription are contained in S_2's subscription. Implementing covering for prefix matching checks whether all S_1's the prefixes S_1 are covered by at least one prefix in S_2's list of prefixes. Prefix f_1 covers prefix f_2 if $f_1 = f_2$ or if f_1 is a prefix of f_2.

Supporting subscription covering for LSCP is a bit trickier. Given two arbitrary circuits, it is NP-hard to infer their relation solely by looking at their structure on non trivial cases. To enable subscription covering, we insert *hints* and *covering tests* in each subscription. The hints describe the subscription in a succint way (such as '< 5'). The covering tests are in fact an encrypted circuit taking as input the hint and answering the question "is this subscription covered by the subscription described by this hint?". The price paid is increased communication overhead for subscription, proven to be manageable for normal subscriptions. The scheme, however, is not practical for all possible subscriptions, since in some cases the covering tests might be prohibitively large. Our implementation is conservative: when hints and covering tests are available they are used to decide the covering relation between subscriptions; otherwise, when hints, covering tests or both are absent, we assume that there is no covering relation between subscriptions.

5.4 Key Management

Multiple mechanisms with different security guarantees will be enabled concurrently in our system. Using the secret key directly in all these systems is a great security hazard if the key can be recovered by an attacker, as the security of the system will be reduced to the security of its weakest component.

However, this need not be the case. Given the session key, an attribute name and type, and the name of a mechanism, we create a new key by feeding it into a pseudo random number generator. Each combination of attribute name, type and mechanism will thus have its own key which will be used for encryption/decryption.

If any of these keys is leaked, the information available to an attacker is minimal. Since the function we used is collision resistant, it is infeasible for an adversary to retrieve the "master" key, even if the name of the attribute, the mechanism or type are known. We also obtain the effect that breaking a single attribute will not cascade into breaking other attributes.

5.5 Supporting new encryption mechanisms

Our implementation allows easy integration of new encrypted matching algorithms. A new protocol must only implement the *EncryptedMatcher* interface and register itself with the encrypted matcher factory, by adding a single line—the operation identifier and the name of the class implementing the interface—in the manager's configuration file.

6 Security Analysis

Our solution for confidential content-based forwarding is a combination of several protocols used to allow filtering for different types of attributes, together with mechanisms to allow hiding attribute names and types from the forwarding routers. The security of the system relies on the security of all its components.

In general, the best security one can hope for in a single mechanism is exact matching of deterministically encrypted notifications and subscriptions (see Theorem 1), if the number of notifications and subscriptions is expected to be large. If, however, the number of subscriptions is small, we might use a notification encryption scheme that has indistinguishable encryptions in the absence of subscriptions (or viceversa).

Using different encryption schemes for different attributes means that some attributes might be easier to "break" by an adversary. The fact that one attribute's values can be retrieved does not mean that more "secure" attributes will also be broken. In fact, if the attributes are independent, breaking one attribute does not provide any advantage in breaking another attribute, due to our key management solution.

6.1 Complexity-security tradeoff

As theorems 2 and 3 indicate, achieving maximum security for arbitrary functions is prohibitively expensive. Therefore, given an attribute and a required subscription expressivity, the application must select the protocol that obtains the proper tradeoff between security and communication overhead. If such a protocol does not exist, content-based publish/subscribe is a poor fit for the application and alternative mechanisms, like secure multicast, should be used instead.

6.2 LSCP

Notification security is equivalent to the security of the one-time pad used for multiple invocations. More exactly, the router can compute the XOR between plaintext notifications simply by XORing the corresponding ciphertexts. Furthermore, if the router obtains a plaintext-ciphertext pair, then all notifications can be completely broken. Subscriptions are susceptible to be broken with some extra work if notifications are broken, by using inference methods, etc.

The information adversary may also gain knowledge without breaking the notifications. First, the adversary can infer the structure of each gate (i.e. it can count the number of 1's and 0's in the garbled output table and tell if it is an and/or type of gate, an xor type of gate, etc.). However, the adversary is completely in the dark as to the actual values of the input and output wires. This information could be used to infer probable relationships between subscriptions, but these relationships could be inferred anyways even with a perfect protocol in time, so they are acceptable. The adversary can decrypt a gate if it performs a brute force search of the space of values assigned to the wires, of size $2^{|w|}$ for a set w of wires. If it possesses partial information about the subscription (such as the bits associated with some of the wires), then the search space can be reduced.

LSCP has polynomial communication overhead for subscriptions and allows the router to distinguish every bit in the notification. According to Theorem 3, one cannot do better than this, except perhaps allowing the router to distinguish sequences of k bits instead, where k is constant — which will in turn make the subscription larger. We choose $k = 1$ consistent to our desire to achieve low overhead.

6.3 Equality Matching

The security of this protocol relies on the security of the symmetric encryption scheme used; if the underlying scheme is secure, then this protocol has maximum security in the context of content-based matching. Either encrypted notifications or encrypted subscriptions can be made indistinguishable, if it is expected that their counterparts are less frequent.

Statistical attacks might be applicable in this case (and therefore for all protocols enabling confidential content-based matching). For instance, for a subscription that filters encrypted single English words, the protocol might be vulnerable to frequency analysis that will expose the encryptions of common words ("the", "and", etc.).

Disclosing a plaintext-ciphertext pair will reveal all the encryptions of the same plaintext, but will not affect the other ciphertexts.

6.4 Range Matching

The security of the range-matching algorithm is better than LSCP, although far from ideal. Every possible prefix is encrypted in a unique way. Therefore, obtaining a plaintext-ciphertext pair will reveal the plaintext $k+1$-bit prefixes of all messages that share k encrypted bits with the known ciphertext, as opposed to breaking the protocol completely in LSCP's case. For a more detailed analysis of the security of this protocol, please refer to Fan et al. [7].

6.5 Keyword Matching

The keyword-matching protocol has best achievable security, with indistinguishable notifications (due to unique string IDs) and distinguishable keywords. A detailed security analysis can be found in the original paper by Goh et al. [10].

7 Evaluation

This section assesses the performance of our protocols. We begin by evaluating the performance of confidential forwarding for typical subscriptions with the purpose of singling out the characteristics of each protocol. We conclude with a brief evaluation of a hypothetical financial data dissemination application that uses our solution.

7.1 Evaluation Methodology

All the data we used for testing is synthetic, being generated uniformly at random or by using the Zipf distribution. The choice of parameters was partially based on the study of previous experimental evaluations in the same context [4].

In all the tests, a single instance of the enhanced SIENA matching engine was evaluated. All experiments were run on an 1.7Ghz Intel Centrino Processor with 1Gb of RAM running Windows XP and Sun's JDK 1.5. Computation time is measured by using the function *System.nanoTime()* available in Java 1.5, averaged over a number measurements to filter out inherent measurement variations (due to task scheduling, OS background activity, etc.). All the measurements were repeated to obtain a standard error of at most 1% of the measured value.

7.2 Protocol Evaluation

We consider two types of subscriptions we believe are representative for most applications using content-based forwarding: filtering numeric attributes with relational operators (i.e. $<, >, \geq, \leq, \neq, =$) and filtering strings with keywords. We also consider cases where more elaborate types of filtering are required and provide experimental results showing that confidential filtering is indeed feasible.

7.2.1 Relational Matches on Numeric Attributes

We created several subscriptions containing a single constraint, "$< x$", where x is generated randomly in the interval [0,100]. Notifications contain a single integer attribute that is chosen uniformly and randomly from the same interval. We measure two things: the local computation time at the router needed to match all subscriptions against a notification, averaged for 250 notifications, and the size of the subscriptions (measuring the subscription overhead). The size of the notifications is the same for all the three mechanisms.

Figure 1.a shows the matching time for various numbers of subscriptions stored at the router. As expected, the computation time increases for all mechanisms with the number of stored subscriptions, though with a steeper slope for LSCP. The prefix matching mechanism is quite efficient as compared to the plaintext alternative, so it is a good fit for normal filtering on integers. LSCP has a higher overhead, with notification matching time reaching 30ms at 1000 stored subscriptions. Although this is substantially less efficient than prefix-matching (5ms), LSCP does reasonably well if we consider the fact that it supports any subscription function.

We also test the effect subscription covering has on local matching performance, by evaluating LSCP-C (LSCP enhanced with subscription coverage) and Prefix-C (prefix matching with subscription coverage). LSCP-C trades subscription propagation overhead for smaller local computation time (order of magnitude better than plain LSCP in this case). Prefix matching with subscription covering is almost as good as plaintext filtering.

The communication size due to subscriptions is presented in Table 1. As we can see, the circuit representation of the subscription (around 60 gates) makes the communication overhead of LSCP quite high. The overhead of LSCP-C is approximately double, due to the addition of the *covering* circuit and of the *hint*. Prefix subscriptions are quite large as well (600 bytes), but can be reduced to 100 bytes on average if we optimize the serialization routine (since currently 1 bit is represented as 1 byte for simplicity).

Based on this small example, we can extrapolate that using confidential number filtering has acceptable overhead for typical filters, for both local computation and communication.

Table 1 : Subscription Sizes for Numeric Filtering

	LSCP	*LSCP-C*	*Prefix*	*Prefix-C*	*Plaintext*
Subscription size (bytes)	973	2255	600	600	20

7.2.2 Keyword Matching

The keyword matching algorithm we have implemented is efficient and meets most application requirements. LSCP can also be used for this and even to create arbitrary subscriptions, but these can become prohibitively large, making its applicability restricted.

We generated strings comprising 50 words extracted randomly from a predefined collection containing 10000 words. We added these words into a searchable Bloom filter, with a false positive rate of 0.1. Subscriptions specify a single word, selected using a Zipf distribution with parameter 1.1 from the same collection.

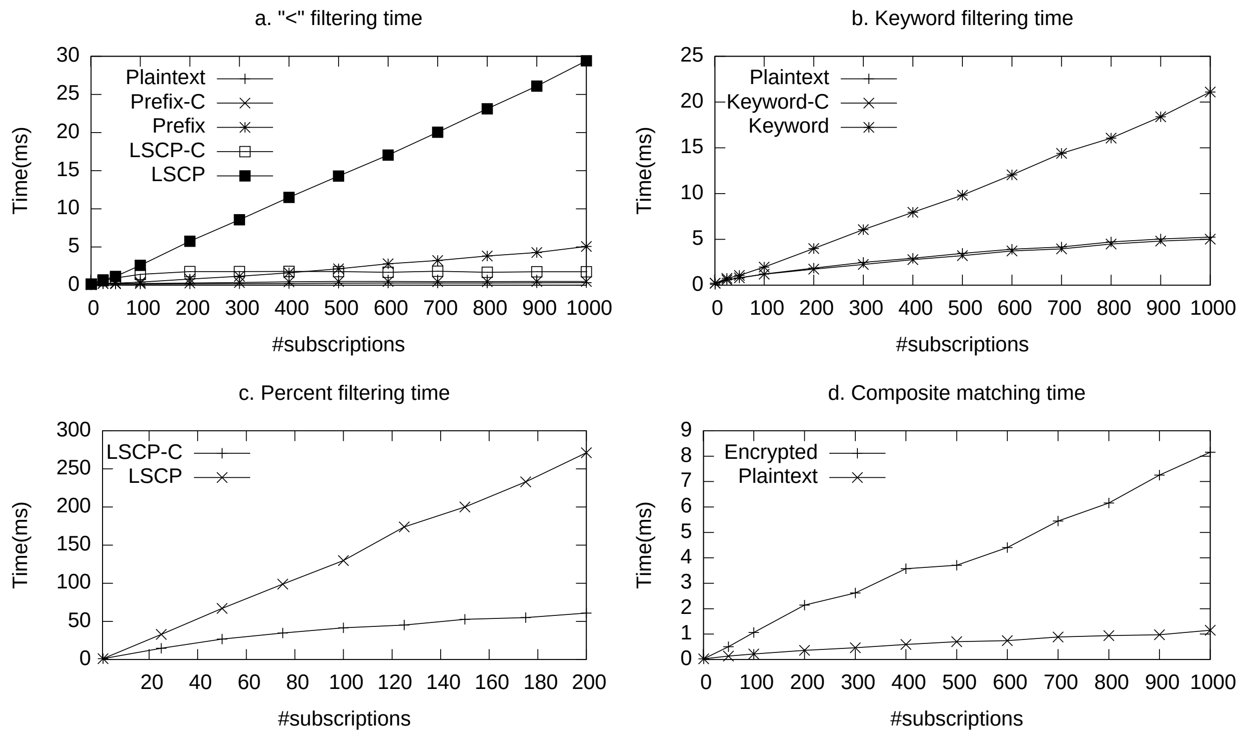

Figure 1: Analysis of matching time for confidential content-based forwarding

We tested the keyword algorithm and its plaintext counterpart. Although the comparison is a bit forced (since arbitrary plaintext substring matching is a lot more expressive than simple keyword matching), it is instructive to look at the performance of the encrypted protocol, presented in Figure 1.b.

Keyword matching (denoted as *Keyword*) has a slightly steeper slope than substring matching (*Plaintext*), being on average 4 times as expensive. Keyword matching enhanced with subscription coverage (*Keyword-C*) is slightly faster than the plaintext substring matching (where subscription coverage is implemented) and can be implemented on top of *Keyword* without additional communication overhead.

Communication overhead for the two protocols is shown in Table 2. Subscriptions are more expensive because we send a trapdoor (depending on the size of the Bloom filter, the number of keywords indexed, etc.) as opposed to a single word. However, the overhead is completely manageable. Notifications are twice as large as their plaintext counterparts, as they include both the encrypted text and the Bloom filter. The latter is approximately as large as the text if we consider all words to be possible keywords and allow a rate of false positives of 0.1. However, if we only select k most significant words in a text (by ranking words based on their appearance count and penalizing common words) both the communication and local matching overhead can be significantly reduced.

Table 2 : Subscription Sizes for Keyword Filtering

	Keyword	*Keyword-C*	*Plaintext*
Subscription size (bytes)	630	630	22
Notification size (bytes)	830	830	460

7.2.3 General Filtering

In some applications, more elaborate filtering might be needed. In such cases, prefix matching and keyword matching can still be used, but with high overhead. For instance, testing whether a number is a multiple of 4 will generate a huge number of prefixes, dependent on the size of the data type. It is in such cases that

LSCP should be used. Its flexibility allows the representation of arbitrary subscription functions, with less overhead than the simpler subscriptions we have considered so far.

For the purposes of this evaluation, we select a few different kinds of subscriptions and measure their local matching time and their size. The results are all presented in Table 3. The first measurement is given as a reference: testing an integer against a constant.

The first measurement simulates filtering in a stock quote dissemination application, where the subscriber is interested in stocks with anomalous price variations. We use LSCP to represent the desired subscription ($\frac{change}{price} \geq threshold$), and place as inputs to this subscription the two attributes.

Matching a single notification against the simple subscription $\frac{change}{price} \geq 25\%$—implemented using repeated additions—takes 0.048ms, so we can perform around 20000 matches per second at a router holding a single subscription. If we use arbitrary thresholds, the size of the circuit dramatically increases (due to the use of multiplication in the subscription) from 50 gates to 1500 gates, having a subscription as large as 88KB. The variation of matching time against the number of subscriptions is presented in Figure 1.c. The matching time is quite large as compared to more simple filtering (like range matches), mostly because of the explosion in gates in the subscription. The matching time for 200 subscriptions is 250ms for LSCP. LSCP-C (LSCP with covering) does significantly better, taking 55ms to match 200 subscriptions.

Our second example is inspired by online games using content-based pub/sub as a communication substrate. Players should only receive those events happening their field of vision, to minimize communication overhead and to avoid looking around the corner. Assume that the user is situated at coordinate (0,0) and there is an obstacle at (100,100). The user should not receive events from coordinates that are masked by the obstacle; that is, the user should subscribe for $x \neq y \ or \ x \leq 100$). In this case, matching a notification takes 0.075ms. To allow arbitrarily positioned obstacles of varying sizes, we resort to multiplication for event filtering. Again, we witness a dramatic increase in the number of gates in the circuit and consequently in the size of the subscription (48kB), with an increase of the matching time to around 0.6ms.

Our third and final example is geared towards filtering arrays of values. Assume that a temperature sensor reads the temperature every minute, but only publishes the bundled results every 10 minutes, to maximize its battery life. A fire-station subscriber is interested in being notified if the temperature has increased for more than 5 consecutive readings. The subscription is a circuit that counts the number of values in the notification (an array of temperature readings) for which $temp[i] \leq temp[i+1]$, where $i = 1 \cdots 9$. Filtering a notification against a single subscription takes 0.275ms, and subscription overhead is 16KB.

Table 3 : Subscription Sizes for General Filtering

Subscription type	*Time (ms)*	*Size (bytes)*
Integer filtering "<"	0.021	973
Anomaly detection for stocks	0.048	2892
	1.1	88000
Obstacle-masked events	0.075	7401
	1.67	130000
Temperature monitoring	0.275	16800

As a general rule, LSCP flexibility comes with a cost. The more complicated the subscription function is, the larger the subscription will be and the larger the matching overhead. In most cases, subscription functions should be reducible to a minimum complexity that is supported efficiently by LSCP. For instance, it is obvious that using multiplication in the subscription function yields significant communication overhead that might be unacceptable for most applications, unless subscriptions are very rare. Particular mechanisms (like using addition or representing functions otherwise) should be investigated for such occurrences.

7.3 Financial Data Dissemination Application

In reality, each subscription contains several constraints on the attributes of a notification. In the previous sections we evaluated the performance of subscription filtering containing single attribute constraints. Here we provide a brief evaluation of a hypothetical financial data dissemination application, to give an insight into the performance of confidential content-based forwarding in more realistic cases.

In the financial data application, there are two types of notifications: real-time stock quotes (containing *exchange*, *symbol*, *price*, *change*) and news regarding the stock market (*exchange*, *symbol*, *text*, *evolution*). Subscribers are interested in one of the two types of notifications.

Subscriptions for stock quotes include equality constraints on *exchange* (chosen uniformly at random), *symbol* (zipf), range constraints on *price* and *change* (uniformly at random) and possibly constraints characterising anomalous behavior ($\frac{change}{price} \geq threshold$, where *threshold* is chosen uniformly at random). Subscriptions for news include equality constraints on *exchange* and *symbol*, keyword matching on *text*, and possibly matching for *evolution* (to determine, for instance, if stocks have had a monotonically increasing trend).

We measured local matching time for the following configuration of parameters: 90% of notifications are stock quotes (with the remaining 10% news), 50% of the subscribers are interested in stock quotes and the rest in news. 30% of news subscribers are also interested in the evolution field, while 10% of stock subscribers are also interested in anomalies. The values we have chosen are intended to be rough estimates of distributions in the real world. These measurements give us insight into the overhead of confidential content-based matching for real applications; however, exact performance measurement is not possible in the absence of real data traces.

The results are presented in Figure 1.d and Table 4. The local computation time for confidential forwarding is around 8ms at 1000 subscriptions, being 8 times as large as its plaintext counterpart. On average, encrypted notifications are nearly 5 times as large as the plaintext ones, while subscriptions are nearly 20 times as large. However, subscriptions are typically less frequent than notifications, so this overhead should be acceptable.

We believe these results are a good indicator that confidential forwarding is indeed practical, and has the potential of providing applications with security guarantees at acceptable overhead.

Table 4 : Communication size for Financial Data Dissemination

	Notification size (bytes)	*Subscription size (bytes)*
Plaintext	125	52
Encrypted	595	9000

8 Related Work

To the best of our knowledge, this is the first fully featured solution for preserving confidentiality in content-based publish/subscribe that has been presented in the literature.

Security in pub/sub was first analyzed by Wang et al., acknowledging the new difficulties posed by this interaction model[20]. Security requirements are identified as integrity, confidentiality and availability at application level, and integrity and availability at infrastructure level. This work focuses on application level confidentiality, addressing both notification and subscription confidentiality. Our observation that subscription and notification confidentiality are only meaningful when used in tandem corrects the misconception in [20] that subscription and notification confidentiality can be achieved independently.

Li et al. [15] address the same issue of achieving subscription and notification confidentiality in content-based pub/sub systems. They support range matching as selection criteria (i.e. a subscription can test if an integer is in a given interval) and encode them by transforming ranges into several prefix matching problems. The notifications are encrypted by using prefix-preserving encryption. Matching is reduced to checking whether an encrypted notification shares a common prefix with one of the desired prefixes. Our approach adopts their work for range matches among other schemas, in the effort to provide applications with a wide range of options in terms of achieved security - cost ratio. The security of the scheme also has some shortcomings: prefix-preserving encryption yields identical encryptions for the same plaintext, allowing the router to identify encryptions of the same plaintext by looking at the encrypted text; moreover, the information leaked by the known-plaintext attack can be significantly higher than what is hinted at in the paper, if we consider that possible messages are not uniformly distributed (which is usually the case).

Research by Khurana [14] focuses on a different adversarial model where the routers themselves are trusted but the adversary can "tap" communication both inside and outside the router network. The solution achieves confidentiality under these assumptions by encrypting the sensitive parts of the XMLbased notifications. The encryption scheme does not use end-to-end shared keys, relying on a security proxy that encrypts notifications with the subscriber's public key when notifications leave the event system.

A different problem, confidentiality on the final hop from the event system to the subscribers, is approached by Opyrchal et al. by using a group key management protocol [18]. This approach circumvents group key creation for every delivered event by maintaining a key cache at the border routers and the subscribers.

Another topic explored in current research is trust management. Fiege et al. [9] present a way to create trust groups in event systems, while Belokosztolszki et al. present a role-based access security implementation [2].

9 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 and opening the road for real deployment.

We have discussed the general problem of ensuring confidentiality for any type of subscription and have discussed technical and theoretical limitations affecting the amount of confidentiality we can achieve in practice.

We have described techniques that are applicable to most problems that content-based pub/sub promises to support. These include two existing techniques aimed at enabling keyword searches in strings [10] and prefix-matching for all data types [7], as well as a novel protocol called LSCP that supports arbitrary subscription functions. LSCP is meant to complement the two other mechanisms in cases where their applicability is limited. We have implemented these mechanisms in a popular content-based publish/subscribe infrastructure in a modular way, allowing the seamless addition of new mechanisms if such need should arise.

Our end goal was to create a solution that allows applications to select the most suitable level of security at an appropriate cost. The implementation in Siena is easily extensible, providing a good starting point for supporting confidentiality in real applications.

To exemplify our techniques and validate our claims, we have evaluated them against their plaintext counterparts. Results show that achieving confidentiality is not only possible but also practical, with acceptable overhead compared to the plaintext solution.

The current implementation can be extended in several ways. First, we could use optimized data structures (such as tries) to organize data that is matched using prefix or exact matches, with the potential of increasing the content-based matching speed. Additional security mechanisms could be researched and added to the current implementation, to offer support for other as yet unforeseen types of subscriptions that we might encounter in practice.

A complementary problem worthy of future attention is to mitigate denial-of-service attacks mounted by malicious routers. Such routers could selectively (or completely) drop messages, depriving some of the subscribers of their information and thus impairing the functionality of the pub/sub network.

References

[1] 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*, page 262, Washington, DC, USA, 1999. IEEE Computer Society.

[2] Andras Belokosztolszki, David M. Eyers, Peter R. Pietzuch, Jean Bacon, and Ken Moody. Role-based access control for publish/subscribe middleware architectures. In *Proceedings of the 2nd international workshop on Distributed event-based systems, DEBS*, pages 1–8, New York, NY, USA, 2003. ACM Press.

[3] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[4] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM*, pages 163–174, Karlsruhe, Germany, August 2003.

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

[6] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[7] Jinliang Fan, Jun Xu, Mostafa H. Ammar, and Sue B. Moon. Prefix-preserving ip address anonymization: measurement-based security evaluation and a new cryptography-based scheme. *Computer Networks*, 46(2):253–272, 2004.

[8] 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*, pages 554–563, New York, NY, USA, 1994. ACM Press.

[9] Ludger Fiege, Andreas Zeidler, Alejandro P. Buchmann, Roger Kilian-Kehr, and Gero Mühl. Security aspects in publish/subscribe systems. In *Third Intl. Workshop on Distributed Event-based Systems, DEBS*, Edinburgh, Scotland, UK, May 2004. IEE The Institution of Electrical Engineers.

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

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

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

[13] S. Kent and R. Atkinson. Security architecture for the internet protocol. In *RFC 2401*, 1998.

[14] Himanshu Khurana. Scalable security and accounting services for content-based publish/subscribe systems. In *Proceedings of the 2005 ACM symposium on Applied computing, SAC*, pages 801–807, New York, NY, USA, 2005. ACM Press.

[15] Jun Li, Chengluai Lu, and Weidong Shi. An efficient scheme for preserving confidentiality in content-based publish/subscribe systems, 2004.

[16] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay — a secure two-party computation system. In *Proceedings of Usenix Security Symposium*, 2004.

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

[18] Lukasz Opyrchal and Atul Prakash. Secure distribution of events in content-based publish subscribe systems. In *Proceedings of 10th USENIX Security Symposium*, August 2001.

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

[20] 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.

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