Efficient Privacy-Preserving Publish Subscribe Middleware

Joud Khoury, Gregory Lauer, Partha Pal, Bishal Thapa, Joseph Loyall, and Nicholas Hoff
Raytheon BBN Technologies
10 Moulton St.
Cambridge MA 02138

Abstract. We address the problem of privacy in publish-subscribe (pub-sub) systems that typically expose some form of published content and subscriber interest, at least to the infrastructure responsible for subscription matching and content delivery, and possibly to other participants depending on the implementation. In our recent work, we proposed P3S, a pub-sub middleware designed to protect the privacy of subscriber interest and confidentiality of published content. P3S combined Ciphertext Policy Attribute Based Encryption (CP-ABE) with Predicate Based Encryption (PBE) in its novel system architecture to achieve the desired level of content (payload and metadata) confidentiality, and subscription privacy. In this work, we build upon P3S to achieve the strongest possible subscription privacy where cleartext subscription is visible only to the subscriber itself. Furthermore, we added support for subscription policy enforcement, improved the expressiveness of predicates by allowing disjunctions of conjunction, and improved the efficiency of the underlying cryptography through enhanced cryptographic construction and optimized implementation of cryptographic primitives. Finally, to the best of our knowledge, this paper presents the first comprehensive and practical implementation of a privacy preserving pub-sub system, demonstrated on a large-scale testbed featuring up to 90 subscribers with a robust, scalable and efficient performance. Our code and testbed specifications are freely available for research and experimentation purpose.

Keywords: publish-subscribe, architecture, security, privacy, experimentation, performance, testbed

1 Introduction

Publish-subscribe (pub-sub) systems provide efficient extensible information exchange between information producers (publishers) and information consumers (subscribers). Pub-sub paradigm decouples information producers from consumers. Publishers can focus on generating information and subscribers on using it, rather than on keeping track of each other. Information dissemination and delivery is based on content rather than on the identity or address of the producers and consumers. Publishers describe the content of messages using metadata,
e.g., attribute-value pairs expressed in XML. Similarly, subscribers register interest in content using predicates described using XPath, XQuery, SQL, or other predicate language. A broker matches subscriber interest with published content. If a subscriber’s predicates executed over the metadata for a published item of content match, the content is delivered to the subscriber. Pub-sub systems have a number of advantages over point-to-point, address- or reference-based systems. First, discovery of information is automatic, which simplifies programming for information-centric distributed systems since information consumers do not need to know (or discover) where information is and do not need to establish connections to possible sources. Second, sharing of information is efficient, only those that expressed interest can access it. Third, configuration is simplified and dynamic conditions are supported. Publishers and subscribers can come and go without any change to each other or the infrastructure. New subscriptions and new types of information can appear dynamically without needing any configuration changes. Finally, the pub-sub system is extensible - published content are self-describing and the schema for describing content are easily updatable.

Because of these reasons, the pub-sub paradigm is widely used in distributed information systems ranging from news and financial services, to civilian and military control systems and intelligence analysis services. However, the benefits of pub-sub systems usually come at the cost of a loss of privacy. In traditional pub-sub systems, the metadata describing published content and the subscription predicates describing consumer interest are visible to others in the distributed system. At best, they are only exposed to the infrastructure responsible for subscription matching and content delivery which is, by definition, outside the protection envelope of the publisher, the subscriber, or both. In a previous paper [21] we argued that this drawback limits the use of pub-sub messaging in a wide range of system and application contexts, and presented an initial implementation and performance analysis of a pub-sub middleware with strong guarantees of (1) content (payload and metadata) confidentiality, and (2) subscription privacy using a combination of innovative architecture and advanced cryptography. Building upon our previous work, we make the following additional contributions in this paper. First, we provide the strongest possible subscription privacy, i.e., a cleartext subscription of a subscriber is not visible anywhere outside of the subscriber. Second, we support subscription policy enforcement to control what a subscriber can subscribe to, while still maintaining subscription privacy. Third, we significantly improve the overall system performance (throughput and latency) using a combination of (1) an efficient construction of the cryptographic algorithms, and (2) an efficient implementation of the underlying cryptographic primitives. Fourth, we improve the expressiveness of subscription predicates by allowing disjunctions of conjunctions. Fifth, we provide an analytical performance model for predicting system performance from the performance and show that the model closely tracks the actual performance. And finally, we present a robust implementation that was deployed outside of the development lab with up to 90 subscribers.
The rest of the paper is organized as follows. Section 2 describes the basic P3S architecture and publish-subscribe operation. Section 3 describes the privacy preserving subscription process and enforcement of subscription policy and Section 4 describes the efficient cryptographic building blocks that are behind P3S’s efficiency. Section 5 describes the P3S implementation over a COTS JMS substrate. Performance evaluation and results are discussed in Section 6. Section 7 discusses related work and Section 8 concludes the paper.

2 Overview

P3S makes use of advanced cryptographic techniques, namely Predicate Based Encryption (PBE) [22, 25] and Ciphertext Policy Attributed based Encryption (CP-ABE) [26] in an innovative architecture in conjunction with traditional public-key infrastructure-based techniques such as TLS that makes sure that subscriber interests and content (payload and metadata) are not disclosed to unauthorized entities. The core of the P3S system consists of four services (see Figure 1): Attribute-based access control and Registration Authority (ARA), Predicate Based Encryption Token Server (PBE-TS or TS for short), Dissemination Service (DS) and Repository Service (RS). We use the term server to emphasize the administrative distinction from service that the server is run and managed on its own host platform and plays an authoritative role, such as issuing cryptographic credentials. ARA and TS are of this nature and they are run on their own host. In contrast, Dissemination and Repository services are

![Diagram of P3S system with annotated interactions](image-url)
part and parcel of the regular information dissemination operation of a pub-sub system. DS and RS could run on a single broker host, but for efficiency they may run on their own hosts or could be replicated on multiple hosts.

Clients (i.e., users of P3S, both publishers and subscribers) register with the ARA by presenting their PKI credential. If a client’s credential is valid for the P3S installation, the registration process returns the following information to the client. First, reference (contact information) and public keys for the DS, RS and the TS. Second, the schema representing the PBE metadata format, i.e., information about the fields and values over which published metadata will be specified, used for specifying subscription interests. And finally, a CP-ABE key pair, where the private key depends on the attributes associated to the client.

A subscriber subscribes in the P3S system by obtaining a PBE token from the TS corresponding to the subscription that can be used to decrypt encrypted messages whose metadata match the predicate. P3S uses several techniques to protect the confidentiality of the subscriber interest and to eliminate the possibility of leakage and compromise. First, the TS that provides PBE tokens is a separate component from the DS and RS responsible for handling published messages. Second, the interaction with the TS uses an oblivious transfer protocol to prevent leakage of the subscriber interest to the TS, and per-subscriber predicate policies to control what a subscriber can subscribe to. TLS is used for subscriber-TS interaction to prevent eavesdropping.

The Publication process involves using CP-ABE to encrypt the payload, and using PBE to create a metadata encrypted ciphertext. The publisher needs to obtain a PBE public key from the TS before it publishes anything. This step can be done once during initialization or anytime prior to its first publication. The publisher generates a unique GUID from a large space and then uses PBE encryption to encrypt that GUID with the metadata description using the PBE public key. The publisher sends the resulting metadata encrypted ciphertext to the DS which then forwards it to all the subscribers. The publisher then CP-ABE encrypts a tuple containing the GUID and payload, using the CP-ABE key it obtained during its registration with the ARA. The CP-ABE encryption specifies a policy that defines the attributes that a subscriber must have in order to decrypt the payload. The publisher then sends a tuple containing the GUID and the CP-ABE-encrypted tuple (containing the GUID and payload) to the DS. The DS then forwards it to the RS, which stores the CP-ABE-encrypted tuple (<GUID, payload>) indexed by the GUID for later retrieval by subscribers. As clients receive the metadata encrypted cipher text disseminated by the DS, they attempt to decrypt it using the PBE token(s) that represent their interest(s). If there is a match, the GUID referencing the payload is revealed; if not nothing is revealed. In the case of a match, the subscribing client fetches the encrypted payload referenced by the GUID from the RS. It can only see the plaintext payload if it has the right attributes to decrypt the CP-ABE encrypted content. All communication from the publisher to the DS, from the DS to the RS, and from the DS to the subscribers is over TLS.

The basic interaction pattern of registering, subscribing, and serving sub-
subscriptions by published content was described in [21] and has not changed. However, the cryptographic primitives used to realize PBE and CP-ABE in the previous version, and the subscription generation process has been completely revamped. The PBE [13] system used in the prior P3S work supported only binary alphabets. In the current version, we have developed and implemented a new PBE variant that is much more efficient (Section 4). Previously, subscriber interest was revealed to the TS, which required that it be a trusted entity. In the current version we implemented a blind PBE token computation based on Oblivious Transfer (OT) [18] that prevents the TS from learning subscriber interests while still allowing it to compute the PBE token for the subscriber and to implement per-subscriber subscription policies. The analytic model that we use to predict performance was updated to use measurements of cryptographic primitive execution times to estimate throughput/latency as a function of system parameters. In addition, the new version of the P3S system was subjected to thorough test and evaluation both internally and externally. Results from the external evaluation, performed by the MIT Lincoln Laboratory, are not available publicly at this time, but observations from the experiments indicate that the P3S system performed as predicted by our analytic models. The next few sections describes these enhancements and achievements in detail.

### 3 Subscription Privacy and Policy

Availability of cleartext subscriptions at the TS can be a serious problem in certain deployment scenarios. We explored a couple of “systems” approaches to address this leakage. First, the TS can be run by a trusted agent. However the cleartext subscription information has to be protected from disclosure, attacks and is certainly a vulnerability. Second, the TS functionality could be embedded in each subscriber. This protects the cleartext subscription from disclosure, but requires replication of PBE master key information and provides no opportunities for implementing per-subscriber subscription policies. That is, there is no mechanism for limiting the subscriptions that a subscriber can generate.

A “cryptographic” approach we developed offers a better solution, and also supports per-subscriber subscription policies. In this approach:

1. Subscribers (clients) send their subscriber IDs to the TS
2. The TS uses a per-subscriber predicate policy to generate a database of all tokens that the subscriber is allowed to request
3. The TS and subscriber engage in an oblivious transfer protocol to allow the subscriber to obtain *one* token without the TS learning which token has been obtained.

One-out-of-two Oblivious Transfer (1-2 OT) [18] allows a receiver to obtain one of two messages that a sender has without the sender knowing which message was obtained while ensuring the sender that only one message was obtained. It operates roughly as follows:
The Sender generates two random strings $x_0$ and $x_1$ and sends them to the Receiver.

The Receiver selects the string $x_b$ corresponding to the message $b$ it wants and uses it to blind a random key it generates.

The Receiver sends the blinded random key to the Sender who attempts to unblind it using each of the random strings $x_0$ and $x_1$. It obtains $y_0$ and $y_1$, one of which is the key and the other of which is garbage, however the Sender cannot determine which.

The Sender encrypts each message with the corresponding $y_b$ and sends them to the Receiver.

The Receiver can only decode the message corresponding to the $x_b$ it selected - the other message decodes to garbage.

Of course, the TS has more than two tokens and thus requires a One-out-of-K Oblivious Transfer (1-K OT) protocol. 1-K OT is built from 1-2 OT as follows:

Each of the K messages is encrypted with a unique combination of keys. Imagine that each message has a unique binary label $\log_2(k)$ bits long, where $b_j$ is the value of the $j^{th}$ bit. The set of encryption keys is a choice from a pair of keys for each bit position and thus the key set is $\{K^b_j, j = 1... \log_2(K)\}$.

The K encrypted messages are sent to the Receiver.

The Receiver uses $\log_2(K)$ invocations of 1-2 OT to obtain the keys required to decode the one message.

While only $\log_2(K)$ invocations of 1-2 OT are required to obtain a token, this approach requires all the possible (permitted by the policy) encrypted tokens to be sent to the subscriber. If there are $M$ fields in the subscription and each field can take on one of $L$ values, then there are potentially (i.e., policy does not constrain the subscriber) $K = L^M$ tokens for the TS to compute and send whenever a new subscription is processed. To put it in perspective, for 10 fields with only 10 values per field, this amounts to computing and transferring 10 billion tokens per subscription request. The time required to compute a token is approximately 0.5 milliseconds per field on a high-end computer, and thus it would take over a day to compute all these tokens using a 500-core compute cluster. A 10-field token is approximately 1250 bytes long and thus the full database of tokens would require 12.5 terabytes of storage and would require over a day to send using a gigabit link. It should be clear that the approach described so far will only work for very small numbers of fields and values per field, or where the per-subscriber policy very aggressively filters what subscriptions are allowed.

We developed a way to address this problem by exploiting the internal structure of the PBE token. The PBE token has multiple components. Some of the components exist to introduce randomness so that tokens from different subscriptions cannot be combined. But there is only one component that depends on the subscription and it can be written can be written as:

$$K_0 = \prod_{i=1}^{I} K_0^{(i)}$$
where $l$ is the number of fields in the subscription.

The structure of $K_0$ allows us to use $M$ 1-K OTs to transfer the $M$ sub-tokens $K^{(1)}_0$ that, when multiplied together, constitute the PBE-token. The TS then only has to generate $ML$ sub-tokens rather than $L^M$ tokens. The implication is that, even for a much more aggressive scenario than before - say, 100 fields each of which can take on 1000 values - the TS needs to create only 100,000 subtokens, rather than $10^{300}$ tokens! A sub-token only takes 0.5 milliseconds to generate, so this larger scenario only takes 50 seconds to generate using a single core (or only a little over a second using a 32 core cluster). The size of the sub-token database is also drastically reduced - instead of terabytes for a 10 field, 10 value scenario, the database is 13 megabytes for a 100 field, 1000 value scenario.

The above sub-token OT enhancement requires a small change in how the TS enforces per-subscriber subscription policies. Upon verifying the ID of the subscriber, the TS now produces a database of all allowed sub-tokens for that subscriber. The subscriber and TS then engage in $M$ OT exchanges where $M$ is the number of fields in the subscription. The $i^{th}$ exchange is a 1-of-$L'_i$ OT where $L'$ is the number of values the subscriber is allowed to pick from in the $i^{th}$ field.

The sub-token OT enhancement does impose a slight restriction on the types of per-subscriber subscription policies we can support: the policies can only independently restrict the values that each sub-token can take on, it cannot impose restrictions that involve multiple fields. For example, if the fields were names or addresses, then the per-subscriber policy could restrict the choice of names or addresses (including the wildcard ‘*’, considered a separate value) that the subscriber could specify. But it will not be possible to enforce policies linking multiple fields, for instance, for a user A, if $State = MA$, then $LegalAge \geq 19$.

4 Cryptographic Efficiency

Recall that in P3S matching occurs at the subscribers: each subscriber tries to decrypt all metadata encrypted ciphertexts it receives using its PBE tokens. The throughput of the system is defined as the maximum publication rate for which the system can keep up. When the system is not network bound (i.e., bandwidth at the DS is abundant), throughput is equivalent to the maximum rate at which the slowest subscriber may consume messages (match metadata). Accordingly, we focused on enhancing the efficiency of the subscriber matching function which is inherently CPU intensive and ends up being the bottleneck for most configurations of interest. Note that the case where the system is network bound instead is less interesting because P3S performs roughly the same as a baseline AMQ system (i.e., the number of publications coming through is not enough to generate compute load in either the baseline or the P3S system). After a brief recap of P3S subscription matching, we will describe the cryptography enhancements.

1 This is the case for example in an enterprise deployment of the system where gigabit ethernet is available to the broker/DS.

2 At steady state, a publisher can only publish as fast as the slowest subscriber.
we implemented, specifically, a larger HVE alphabet and cryptographic primitive optimizations that together with added multithreaded subscriber operation result in more than an order of magnitude increase in overall system throughput compared to the earlier version of P3S.

Predicate Based Encryption enables the matching of subscription interest to published metadata without exposing either the interest or the metadata. PBE supports decrypting a PBE-encrypted message with a PBE token representing the subscriber interest, i.e., a predicate. We constructed a predicate-based encryption system for a general class of equality predicates which we call Hidden Vector Systems or HVEs for short \[2\]. Formally, let \(\Sigma\), the alphabet, be a finite set and let \(\ast\) be a special symbol not in \(\Sigma\). Define \(\Sigma^* = \Sigma \cup \{\ast\}\). The “\(\ast\)” symbol plays the role of a wildcard or “don’t care” value. For \(\sigma = (\sigma_1, \ldots, \sigma_\ell) \in \Sigma^\ell\) define a predicate \(P_{\sigma}^{HVE}\) over \(\Sigma^\ell\) as follows. For \(x = (x_1, \ldots, x_\ell) \in \Sigma^\ell\) set:

\[
P_{\sigma}^{HVE}(x) = \begin{cases} 1 & \text{if for all } i = 1, \ldots, \ell : (\sigma_i = x_i \text{ or } \sigma_i = \ast), \\ 0 & \text{otherwise} \end{cases}
\]

In other words, the vector \(x\) (publication metadata) matches vector \(\sigma\) (subscriber interest) in all the coordinates where \(\sigma\) is not \(\ast\). We refer to \(\ell\) as the width of the HVE. Notice that the size of the alphabet, \(|\Sigma|\), determines the total number of values each field \(x_i\) may take.

### 4.1 Large HVE Alphabet

In the previous system \[21\], we used the HVE construction of Iovino et al. \[13\] since it was the only efficient construction (uses prime order groups, and an asymmetric bilinear map). A major limitation of their construction however was its restriction to the binary alphabet i.e. \(\Sigma = \{0, 1\}\). This reduced the efficiency of the system and its practicality in supporting a richer attribute space. Specifically, in order to support a metadata space of \(l\) attributes/fields, each of which may take one of \(N\) values, we had to construct the \(l \log(N)\)-bit vector \(x\) where the first \(\log(N)\) bits are used to encode the 1st attribute, and so on. We did the same for \(\sigma\) assuming a wildcard spans all bits that represent the attribute. This led to expansion of the HVE width \(l\) by a factor of \(\log(N)\). Consequently, this increases both HVE encryption and decryption times, and reduces the system throughput, by roughly the same factor. The main reason for this throughput reduction, is because the HVE decryption cost (number of cryptographic operations needed) is linear in the number of non-wildcard fields in the vector \(\sigma\) (and similarly the encryption cost is linear in \(l\)) \[13\].

Our new HVE construction extends the Boneh-Waters large alphabet construction \[2\] to support prime order groups and asymmetric bilinear maps. To the best of our knowledge, this is the most efficient HVE construction in the literature. The HVE alphabet is \(\Sigma = \mathbb{Z}_p\) for some very large prime integer \(p\), where \(p\) is the order of the groups in which we work. In practice, one may even hash arbitrary length strings into \(\mathbb{Z}_p\) as described in \[2\]. This eliminates the \(\log(N)\)
blowup in cost discussed earlier, enables a much richer attribute space, and accordingly a more practical system. For example, when the attribute space is as large as \( N = 1024 \) different values, the new large alphabet scheme results in an order of magnitude speedup in overall system performance relative to the binary alphabet scheme. As discussed later in section 5, we make our implementation public as part of P3S distribution.

4.2 Cryptographic Primitive Optimizations

The second major cryptographic performance enhancement is leveraging the Multi-precision Integer and Rational Arithmetic C/C++ Library (MIRACL) [3]. MIRACL provides cryptographic primitives that P3S uses to implement HVE and CPABE. The two key primitives for encryption and decryption are exponentiations and pairings. MIRACL provides optimization for these primitives that significantly speed up the operation in certain cases.

**Exponentiation** computes \( g^x \mod n \). Exponentiation with precomputation: If \( g \) and \( n \) are known in advance, we can speed up the exponentiation using a precomputation. The latter precomputes and stores a table of values \( g^i \mod n \) for different exponents \( i \) and replaces exponentiation with lookups and multiplication [15]. This is directly applicable to HVE encryption for example, since a known constant group element \( V \) is raised to a secret exponents \( s \) that changes for each payload [2].

**Pairing** computes the bilinear map \( e(g_1, g_2) \) where \( g_1 \in G_1 \), \( g_2 \in G_2 \) and \( e(g_1, g_2) \in G_T \). Pairing with precomputation. If \( g_2 \) is a constant that is reused in multiple pairings, then it is possible to speed up the pairing using precomputation of various parameters used in the Miller loop [5]. For example, the HVE token test computes \( e(C_0, K_0) \) where \( K_0 \) is constant (for a given subscription predicate) and \( C_0 \) is different for each publication [2]. These pairings can be sped up using precomputation. **Multi-pairing** (with precomputation) optimizes the computation of products of pairings [10]. These optimization can be combined with pairing precomputation if elements in \( G_2 \) are constant. For example, the PBE token test computes \( \prod_{i \in S} e(C_{i,1}, K_{i,1}) e(C_{i,2}, K_{i,2}) \) where the \( K_{i,1} \) and \( K_{i,2} \) are constant (for a given subscription predicate). These computations can be sped up using multipairing with precomputation.

Table 1 highlights the achieved speedup enabled by these optimizations by showing the actual performance of HVE encryption and decryption algorithms on the Core i7 (MPB OSX 10.7.4) and the Xeon E3-1230v2 (Dell R210 Linux) 64-bit processors. Notice the 4-5x speedup in HVE decryption time which increases the system throughput by the same factor when the system is CPU bound.

5 Prototype Implementation

The current version of the P3S prototype is built on top of the open-source Apache ActiveMQ (AMQ) pub-sub middleware [8] with cryptographic support
Table 1. HVE Performance for width $l = 11$ and no wildcards in the token, and security level AES 128 bits (BN curve); Fixed word size 64 bits for big integer arithmetic.

<table>
<thead>
<tr>
<th>Operation</th>
<th>core i7</th>
<th>Xeon E3-1230v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVE Encrypt Time (ms): no optimization</td>
<td>7.80</td>
<td>5.26</td>
</tr>
<tr>
<td>HVE Encrypt Time (ms): with optimization</td>
<td>2.95</td>
<td>2.00</td>
</tr>
<tr>
<td>Speedup</td>
<td>2.64x</td>
<td>2.63x</td>
</tr>
<tr>
<td>HVE Decrypt Time (ms): no optimization</td>
<td>80.92</td>
<td>51.35</td>
</tr>
<tr>
<td>HVE Decrypt Time (ms): with optimization</td>
<td>15.17</td>
<td>12.79</td>
</tr>
<tr>
<td>Speedup</td>
<td>5.33x</td>
<td>4x</td>
</tr>
</tbody>
</table>

from the Miracl Cryptographic Software Development Kit (SDK) [3]. P3S extends AMQ functionality, but for the most part, retains the Java Message Service (JMS) API messaging standard. We implement a java wrapper of the Miracl SDK which we use for implementing all of P3S's cryptographic functionality (mainly HVE and CP-ABE schemes). Algorithmic details and enhancement to P3S cryptographic operations were described earlier in Sections 3 and 4. In order for our multi-threaded Java components to use Miracl, which is C-based and single-threaded, we developed a Java wrapper that we call jMiracl. The jMiracl java wrapper abstracts the lower level details of elliptic curves from the pairing-based cryptographic algorithms (bilinear maps). The jMiracl library supports multi-threading, pre-computation, and all other optimizations available through Miracl. In the following subsections, we briefly describe the implementation of our P3S prototype, starting with the P3S client libraries that publishers and subscribers use in order to work with P3S. We then describe the implementation of the P3S's service components (i.e., DS, RS, TS and ARA).

5.1 P3S Client-Side Libraries

Figure 2 shows how the P3S client-side libraries are layered on top of AMQ libraries. P3S extends several of the main AMQ implementations of the JMS API, notably the P3SConnection, P3SSession, and P3SConnectionConsumer. P3S additionally implements the JMS API classes P3SServerSession and P3SServerSessionPool, as well as the new class P3SConnectionProducer. The polymorphic extension to the AMQ stack leverages the JMS messaging infrastructure provided by AMQ and enables an existing application to switch their JMS layer to P3S’s privacy-protecting layer mainly by switching the connection factory.

As discussed earlier in section 4, one of the main efficiency enhancements we implemented in P3S was multi-threading of clients (publishers and subscribers). This has the potential to linearly increase the throughput of the system when it is CPU-bound. The JMS API by design associates a client with a single Session. The latter is defined to be a single-threaded context in order to guarantee ordering of messages, i.e., so that a client that belongs to a session always receives (or sends) messages in order. To allow application servers to scale better, JMS additionally provides a way to implement multi-threaded consumers using the
Fig. 2. Layering of P3S client (publisher and subscriber) stack on top of AMQ stack (producer and consumer)

ConnectionConsumer interface. However, no support for multi-threaded producers exists. We extend the AMQ ConnectionConsumer to implement our multi-threaded P3SConnectionConsumer, and we additionally introduce a new multi-threaded producer interface, P3SConnectionProducer.

In our multi-threaded operation, the consumer and producer objects are not created as part of the session anymore. They are directly bound to the connection itself. Using the P3SConnectionConsumer and P3SConnectionProducer instead of their single-threaded counterparts ActiveMQConsumer or ActiveMQProducer enables messages to be dispatched within a configurable pool of sessions, instead of being bound to a single session/thread. An important implication of this design decision however is that P3S applications should not assume any ordering on messages when the pool size (number of threads) is > 1.

5.2 P3S Servers

We briefly describe the implementation each of the P3S servers next. Note that all client-server connections support TLS with server certificates and client password-based authentication.

P3S Dissemination Service (DS). The DS is a restricted configuration of the AMQ Broker with a few custom extensions embodied in a P3S Broker plugin. The most notable modification involves anonymizing the producerId which by default gets sent in the header of each JMS message. The P3S Broker plugin clears the producerId field before forwarding messages to matching subscribers. This prevents subscribers from learning anything about the producer’s identity. The DS/Broker is restricted to manage only two JMS topics PRIVACY.MDATA and PRIVACY.CONTENT, all P3S subscribers are automatically subscribed to the former, and the RS is automatically subscribed to the latter. We implemented a plugin to support password-based authentication at connection time for controlling group/user read/write access to the JMS topics—this level of authentication is on top of the PKI based authentication (administered by ARA)
that all participants in P3S are subjected to. In the default configuration, only the authenticated RS is allowed to subscribe to the PRIVACY.CONTENT topic.

**Repository Service.** The Repository Service performs two main tasks: (1) store published payloads, and (2) serve stored payloads upon subscriber retrieval requests. The RS functionality is provided by three independent but collaborating services components. First, the Message Persistence Service listens to content/payload messages on the PRIVACY.CONTENT topic and stores them into an underlying message store. Our prototype uses a relational database (embedded Apache Derby) to store content payloads and implements multiple JDBC adapters to support different RDBMS vendors. The embedded Derby IO times can be too slow for our target data rates, so we also implemented an in-memory cache (default configuration). The RS can be configured with either one of these data stores or with both for layered storage/retrieval. Second, the Content Retrieval Web Service is used by subscribers to request the content payload corresponding to a valid GUIDs. The RS utilizes an embedded Apache Jetty web server to host this service. Finally, the Message Cleanup Service is a daemon that continuously checks for expired messages and removes them.

**PBE Token Server.** The PBE-TS implements the efficient, blind, OT-based PBE token generation scheme described in 3. It serves subscriber requests for cryptographic tokens (and public parameters) without learning their plaintext interests while still enforcing per-subscriber policies. We defined a new Client-Server wire format which the PBE TS and the subscribers use underneath for messaging. All higher level client-server protocols including the Blind HVE and Oblivious Transfer use the P3S wire format. The PBE-TS is multi-threaded and can execute multiple Blind HVE sessions in parallel.

**ABE Registration Authority.** The ARA serves the publishers and subscribers the system public parameters. A subscriber may also request a private key corresponding to its access attributes. The ARA utilizes an embedded Apache Jetty web server to host the web service.

### 6 Performance Evaluation

We evaluate P3S performance and scalability via extensive analysis and real-world experimentation using carefully designed hardware/software testbed. First, we used analytic models to characterize P3S performance with small-scale prototype feeding the parametrization; the results were presented in [21]. This section presents the comprehensive experimental evaluation of improved P3S that validates the performance model using a full-fledged deployment on a large-scale high-performance machine cluster. Here, we present the details on the evaluation criterias, the testbed and the performance results from the experimentation.

#### 6.1 Criteria

We use correctness, end-to-end latency and throughput as the performance metrics; we consider number of subscribers, CPU core assignment per subscriber,
payload size, PBE token width, security level and the subscription match rate as the configurable parameters, and employ a multi-threaded test harness protocol to ensure realistic but controlled evaluation of P3S with reproducible results.

**Metrics.** The three measures chosen to gauge robustness, scalability and efficiency of a practical P3S system are:

- **Correctness:** We evaluate the robustness of P3S by ensuring that
  (a) Subscribers receive all the items that match their subscription, and
  (b) Subscribers receive no items that do not match their subscription
  We calculate (i) False Positives, which are publications that do not match a given client’s subscription but are delivered to the client, and (ii) False Negatives, which are publications that match a given client’s subscription, but are not delivered to that client. It is straight-forward to measure (b) by examining the System Under Test (SUT) output logs but challenging to measure (a) given the asynchronous subscriber matching, multi-threading and asynchronous SHUTDOWN of the P3S clients. We implemented all our tests to wait enough time between subscription updates and sending of SHUTDOWN messages in order for the messages in the queue to be cleared. This allowed us to accurately measure (a).

- **Throughput:** This is the maximum rate at which publications can be injected into the P3S system, such that all publications are properly matched and delivered. The major throughput bottlenecks are the publisher CPU (time to do PBE and ABE encryption), publisher bandwidth (time to send payload and metadata), subscriber bandwidth (time to receive payload and metadata), and subscriber CPU (time to do PBE matching).

- **End-to-End Latency:** This is the time it takes for a single publication to reach all matching subscribers when the system is unloaded. Latency is the time required to encrypt the payload and metadata, send the metadata to the subscriber, have the subscriber perform the PBE match, retrieve the payload (this may require waiting while the payload is delivered to the RS) and decode the payload.

**Parameters.** We evaluate the P3S throughput and end-to-end latency as a function of (i) the number of subscribers, (ii) the payload size, (iii) the complexity of the subscriptions, (iv) security parameters such as AES size, and (v) the subscription match rate. Here, the subscription complexity is the number of non-wildcards in the subscription, the HVE width and the number of subscriptions per subscriber. The match rate is the fraction of subscribers that will match any given publication. As part of our controlled tests, we design the publications and subscriptions to achieve a desired match rate. This helps us perform statistical analysis of the P3S performance. In terms of choosing above parameters for configuration, we used the modeling results from [21] and simple P3S tests to pick system parameters with the most impact on P3S performance. For example, P3S works by throttling the publication rate to ensure that the subscribers receive them all. This ensures that the publication rate need not be varied while analyzing throughput. Similarly, the number of subscribers have very minimal impact on the latency given that the
subscribers can match tokens in parallel. Therefore, the number of subscribers need not be varied while analyzing latency. However, varying the number of wildcards in a subscription have significant impact on the matching time of the subscription. Because of the way PBE works, the wildcard field does not add to the matching time, so a subscription with 5 non-wildcards and 6 wildcards is expected to take roughly half as long to match a subscription of 11 non-wildcards. Finally, the network bandwidth setting is less interesting since P3S performs similar to the baseline. We focus on CPU bandwidth setting instead, to assess the P3S specific overhead cost.

**Methodology.** Figure 3 illustrates how the P3S components were distributed across the testbed and the maximum number of cores assigned to each entity. First, the test-harness is implemented as a tightly-coupled software suite that follows a client-server model with two main components: (i) the master harness that executes on a server and injects packets into the system and (ii) the slave harness that runs on the client machines and captures and timestamps delivered payloads. Second, the harness allows specifying number and the IDs of the virtual cores to pin components to making way for multi-threaded parallelization of P3S components. And finally, the test harness runs on an independent network from the P3S system keeping it from impacting the performance. Postprocessing of the harness logs determine the throughput, latency and correctness of the system under test.

![Fig. 3. Test-Harness framework supports multi-threaded clients (with multiple cores)](image)

### 6.2 Experimentation Setup

In this section, we present the details of our experimentation framework: the testbed components, hardware/software specification, test configurations and experiment design used to collect data for the performance graphs.
Testbed Layout. Our testbed comprises a large cluster of high-performance Hyper-Threading (HT) enabled multi-core machines as SUTs to host various P3S components all connected via gigabit Ethernet links. It runs DS, RS and publisher (along with the master harness) on individual Intel Xeon X5650 Quad Core HT @2.66GHz SUTs with 48GB RAM while ABE ARA and PBE TS share a single SUT as seen in Figure 3. Since, ABE ARA is the lightest of the four P3S components, we assign 1 core to ABE ARA, while rest of the components get 3-4 cores each. The subscribers are distributed among the 30 Intel Xeon E3-1230v2 Quad Core/8T HT @3.3GHz SUTs with 16GB RAM such that each subscriber gets at least 1 core reserved for itself while the slave harness running on the host gets one as well. To minimize test artifacts, we (i) ran the DS and RS on separate machines communicating over the network, so that their communication was not unrealistically fast, (ii) ensured that no subscriber was on the same machine as the DS, RS, or PBE TS and (iii) ensured that all subscribers were on the same type of machine since pub-sub throughput is constrained by the slowest subscriber.

The SUT hosts run Ubuntu 12.04 as their OSes, and the P3S core runs ActiveMQ v5.5.1 as its baseline. Java openjdk1.6 is used for jvm.

Test Configuration. Table 2 depicts the parameter configurations we use to run experiments and collect data sets for performance analysis. We run an experiment per combination of the parameter setting, and repeat each experiment 10 times before averaging the results to achieve performance plots depicted in the next section. Due to space limitation, we only present representative viewgraphs in this paper.

6.3 Experimentation Results

We evaluate P3S performance in terms of correctness, throughput and latency.

Correctness. Our experimentation results show zero false positives and zero false negatives. Each P3S subscriber receives all items that match their subscriptions and no item that does not match the subscriptions.

Throughput. We run a comprehensive suite of throughput tests, varying number of subscribers, the message size, and the number of wildcards in the subscription. The match rate was fixed at 10%. Our experimentation results closely
Fig. 4. Throughput vs number of subscribers for (a) 1 and (b) 5 non-wildcards. The lines show numbers from analysis.

tracks the analysis results as seen in Figure 4. Figure 4 shows P3S throughput when number of subscribers and the payload size are varied for subscriptions with 1 and 5 non-wildcards respectively. As our models predicted, we see that the number of subscribers does not have an impact on the throughput, however, the payload size and the number of wildcards in the subscription do [See Figure 5(a)].

Fig. 5. (a) Throughput and (b) Latency vs payload size for varying numbers of non-wildcards. The lines show numbers from analysis.

End-to-End Latency. Figure 5(b) shows the P3S latency as a function of payload size and number of wildcards in the subscription. We observed that for small payloads, the limiting step in subscription matching is the PBE match operation, as shown in Table 3. However, for larger payloads, the bottleneck shifts to the RS bandwidth where a matched subscriber is waiting on the RS to deliver the payload.

7 Related Work

An approach outlined in [14] uses re-encryption and onion routing type indirection to dissociate the location of predicate matching from both the publisher
Table 3. Latency Test Results

<table>
<thead>
<tr>
<th>component</th>
<th>model duration (ms)</th>
<th>actual duration (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>publisher – PBE encryption</td>
<td>4.4</td>
<td>7</td>
</tr>
<tr>
<td>publisher – ABE encryption</td>
<td>2.7</td>
<td>3</td>
</tr>
<tr>
<td>subscriber – PBE match</td>
<td>28</td>
<td>24</td>
</tr>
<tr>
<td>subscriber – RS retrieve</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>subscriber – ABE decryption</td>
<td>7.4</td>
<td>6</td>
</tr>
<tr>
<td>sum of components</td>
<td>45.5</td>
<td>44</td>
</tr>
<tr>
<td>actual end-to-end latency</td>
<td></td>
<td>48</td>
</tr>
</tbody>
</table>

and the subscriber. This scheme appears to be specialized for a P2P content sharing network. In [20], a policy-based approach is presented where data owners can specify who can access their publications and under what condition. But the broker and the policy enforcement mechanism can see both the published content and subscriber interest. A content-based pub-sub scheme where content decryption keys are shared using Pedersen commitment and matching is performed on blinded attribute-value pairs is presented in [10] (and followed up in [17]). In this scheme, subscribers need to register a-priori with the publishers, and brokering is limited to equality of strings and comparison of numerical values. Identity-based encryption is used in [24] to achieve a level of content confidentiality and subscription privacy in peer-to-peer event-based broker-less pub-sub system. Unlike P3S, both subscriber interest and content description (in the form of credentials, which are simply a binary encoding of the metadata space) is visible to the key server (which is analogous to the TS in P3S). In addition, maintenance of the overlay topology requires each node to know the subscription of its parents as well as children, limiting the level of subscription privacy. Contrail [23] presents a novel form of pub-sub for smart phone platforms using sender-side content filters for privacy. This scheme has a strong association between the publisher and subscriber since they perform a handshake to install the sender-side filter.

Ion et al. [12] presents a scheme that protects subscription privacy as well as confidentiality of the publications (and metadata) assuming a trusted authority. The system uses proxy re-encryption and Key Policy Attribute-Based Encryption (KP-ABE) to express interests and perform blind matching. Unlike P3S, plaintext interests of all subscribers are visible to the trusted authority and the broker can learn the complexity of the subscription access tree. Each publication is re-encrypted by the broker and matched against all available subscriptions, making the broker a likely performance bottleneck. Barazutti et al. [11] present an efficiency enhancement for privacy-preserving pub-sub by using a bloom filter based pre-processing step that saves expensive cryptographic operation upstream on content that are not likely to match. This scheme requires that publishers and subscribers share a set of secret keys (used in creating the bloom filters) that is not known to the broker. The bloom filter representation of private subscriptions and publications leaks information since the MAC scheme used is deterministic. The authors suggested adding noise to the filters as a potential remedy.
Similar to the homomorphic encryption flavor of [17], the work by Choi et al. [4] leverages a distance preserving encryption scheme whereby the broker computes on encrypted elements (publication and subscription), and deduces whether there is a match based on the encrypted result (which is a distance measure between the plaintexts). The system protects against an HBC broker only and requires publishers and subscribers to exchange secrets apriori. In [6] Di Crescenzo et al. describes a system that protects subscriber interest and content description using a conditional oblivious transfer [7] scheme where private extraction of a value is successful only if a public predicate shared by the sender and receiver is satisfied on the private inputs. In their system, subscriptions and publications are based on symmetric cryptography for efficiency, and subscribers share keys with the publishers. Matching occurs at the clients and a third party service stores and transfers the data only to the matching clients.

8 Conclusion

We described a robust, scalable and efficient privacy-preserving pub-sub middleware that provides a high level of privacy and confidentiality. The high level of privacy is achieved by a combination of advanced cryptography and innovative system architecture. No element in the system, even a network eavesdropper, has visibility of cleartext content and subscription interest. The efficiency is achieved by an efficient large alphabet HVE implementation, multiple levels of optimizations in the cryptographic library as well as the parallel and decentralized subscription matching. The robustness is achieved by layering the privacy-preserving capabilities on top of the robust content-based pub-sub engine Apache MQ, and using Apache MQ features for disseminating metadata-encrypted GUIDs and sending CP-ABE encrypted content to the RS. The current prototype has been successfully transitioned out of the development laboratory for independent evaluation and tested with close to 100 clients under various combination of subscription matching probability, payload size and publication rate. Evaluation results are not yet publicly available, but preliminary observations indicate that the P3S handled the load robustly and delivered matching content to subscribers without throwing exceptions of failures.

While P3S is a significant step forward in privacy-preserving pub-sub, there are a number of issues that need additional research. For example, the OT scheme currently used in P3S scales linearly with the number of actual values each field can take i.e., PBE token generation will become more expensive for larger alphabets. Although not a problem for long standing subscriptions, scenarios where subscriptions change frequently will be adversely affected when the alphabet size is large. Replacing the current scheme by the more efficient OT scheme of [19] is an option. Other possibilities include finding ways to amortize the cost of OT or developing a blind PBE scheme with a built in (efficient) key extraction protocol similar to [11]. Traffic analysis attack is another issue that needs further research. One potential approach here will be to leverage techniques like jamming-resistant anonymous broadcast protocol [9].
9 Acknowledgement

The authors acknowledge the collaboration and guidance of Brent Waters and Vitaly Shmatikov of University of Texas, Austin in integrating CP-ABE and PBE in P3S. This work is supported by the Intelligence Advanced Research Project Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) Contract No. DIIPC20195. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation there-on. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI/NBC, or the U.S. Government.

References

25. Waters, B.: Prime order lwe with large universe, bBN P3S Internal Report