QUIP: A protocol for securing content in peer-to-peer publish/subscribe overlay networks

Amy Beth Corman* **Peter Schachte*** **Vanessa Teague**

*National ICT Australia, Victoria Lab
Department of Computer Science & Software Engineering
The University of Melbourne
Email: {amy,schachte,vteague}@csse.unimelb.edu.au

Abstract

Publish/subscribe networks provide an interface for publishers to perform many-to-many communication to subscribers without the inefficiencies of broadcasting. Each subscriber submits a description of the sort of content they are interested in, then the publish/subscribe system delivers any appropriate messages as they are published. Although publish/subscribe networks offer advantages over traditional web-based content delivery, they also introduce security issues. The two security problems that we solve are: ensuring that subscribers can authenticate the messages they receive from publishers, and ensuring that publishers can control who receives their content. We propose QUIP, a protocol which adds efficient authentication and encryption mechanisms to existing publish/subscribe overlay networks. The idea is to combine an efficient traitor-tracing scheme (by Tzeng and Tzeng (2001)) with a secure key management protocol. This allows publishers to restrict their messages to authorised subscribers and to add and remove subscribers without affecting the keys held by the other subscribers.

Keywords: Peer-to-peer, publish/subscribe, security, network protocol

1 Introduction

There are many variations on content delivery systems used by the Internet today, each with its own advantages and disadvantages. We propose using a peer-to-peer publish/subscribe model to efficiently manage and securely deliver intermittent content. We will use web comics, the dissemination of electronic comic strips, as an illustrative example throughout this paper.

There are many advantages to be gained from using a publish/subscribe system as opposed to the current model of delivering web comics via web pages. One advantage to subscribers is that a push model is more convenient than a pull model because it is not necessary for subscribers to take action in order to have their content delivered. A push model is especially beneficial if publications are made at unpredictable intervals since the subscriber will never waste time looking for something that is not there. A subscriber may use the same interface to access multiple web comics from different publishers. A publish/subscribe system enables the possibility of a single billing interface for these to the subscriber. This has multiple benefits. It is simpler for the subscriber to only enter payment information in one place, and only requires the subscriber to trust the payment authority and not each individual publisher. This allows the subscriber to enjoy content from any publisher without needing to worry about the safety of their payment details. Centralised payment improves efficiency since only one organisation handles payment leaving publishers free to focus on creating new content. This will enable new publishers to gain subscribers without needing a reputation for trustworthiness. Publish/subscribe systems also provide advantages in matching publishers to interested subscribers.

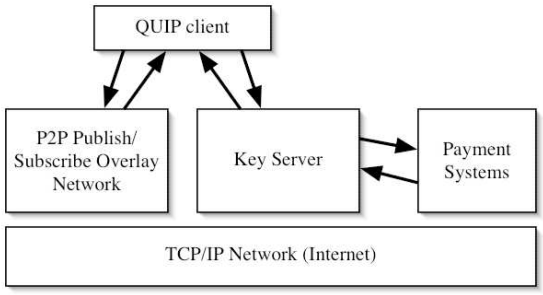

Figure 1: An illustration of the relationship of the network layers.

A publish/subscribe system provides further advantages for publishers. In the current web delivery method, publishers depend on advertisers for income, which means that the subscribers must view ads on the same page as the comic. This system allows the publishers to charge the subscriber directly. The publishers do not need to maintain a constant presence on the Internet and need only connect long enough to transmit the publication into the publish/subscribe network.

We introduce QUIP, a protocol for secure content distribution in peer-to-peer publish/subscribe overlay networks. The goal of QUIP is to securely provide all the advantages of a publish/subscribe distribution system. Our approach is to add flexible and efficient encryption and authentication mechanisms to existing publish/subscribe systems.

A content-based publish/subscribe system works as follows. A publisher must advertise the details of their expected publications. In the context of web comics, this could include the name of the comic, a category of the type of comic and any other descriptive data the publisher would like to provide. A subscriber then decides what comics they are interested in and sets up a filter that will match events as described by the publisher (in the simplest case, this would be the title of the comic). The publisher then publishes events (such as a single comic) which are routed through the publish/subscribe infrastructure

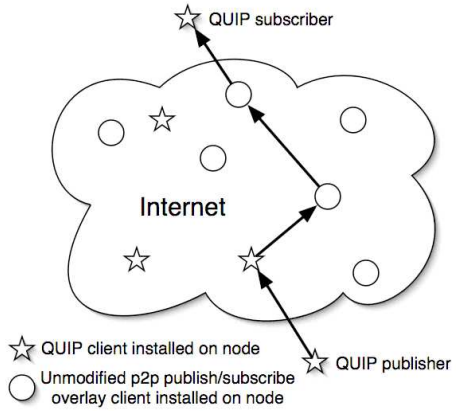

Figure 2: An illustration of p2p publish/subscribe overlay layer of the network.

to the interested subscribers.

The relationship between our protocol and the existing network systems is shown in Figure 1. We create a *key server* which is separate from the publish/subscribe overlay network. We also add a QUIP *client* which interacts with both the publish/subscribe network and the key server. The key server does not interact with the publish/subscribe overlay directly. The QUIP key server will also interface with existing *payment systems*. It is not necessary for all nodes in the publish/subscribe network to run our client, as illustrated in Figure 2.

QUIP is designed for applications with a large number of subscribers and a smaller number of publishers. A single entity may be both a subscriber and publisher. A subscriber may subscribe to multiple publications and a publisher may publish multiple publications.

QUIP uses a public key traitor tracing scheme proposed by Tzeng and Tzeng (2001) (described in Section 4.6) which provides the ability to add and remove subscribers by updating only the publication key. Each subscriber is given a unique key in this scheme which makes it possible to determine which user has leaked their key if a key is posted publicly.

The rest of this paper is organised as follows: Section 2 discusses related work, Section 3 discusses our threat model and requirements, Section 4 describes the QUIP protocol in detail, Section 5 comments on performance and Section 6 concludes and suggests some further work.

2 Related Work

We have similar but not identical goals to Eventguard (Srivatsa & Liu 2005). Both Eventguard and QUIP are systems which sit on top of existing publish/subscribe overlay networks. Eventguard has six main components which are designed to protect each of the five major publish/subscribe actions (subscribe, unsubscribe, publish, advertise, unadvertise) plus routing. Eventguard uses ElGamal for encryption, signatures and the creation of tokens. The major differences between our two solutions are that we do not assume private channels and we do not put so much emphasis on the privacy of subscriptions. Eventguard needs to update subscriber topic keys whenever one subscriber leaves, while this is not necessary with QUIP.

Although Srivatsa and Liu claim Eventguard does not depend on the underlying IP network to provide confidentiality, integrity or authenticity, it is hard to see how these properties are provided based on the examples given. If we look at the subscribe guard as an example (shown in Equation 1) we can see that the key $K(w)$ is being transmitted to the subscriber from the *Replicated Trusted Meta-service* (referred to as the MS) and there is no provision for encrypting it or protecting it from eavesdropping.

$$sb(w) = \langle K(w), T(w), sig^{S}_{MS}(T(w)), UST^{S}(w) \rangle \quad (1)$$

It is clear that a secure channel is needed between the subscriber and the MS to maintain the secrecy of the key. We assume that Srivatsa and Liu intended to use SSL to secure communication between the MS and the clients, which would fix the problem of eavesdropping. It is still not clear, however, how re-keying is to be handled securely without client certificates. In Eventguard, each subscriber and topic pair has a unique identifier, but it is not stated how this would be securely mapped to the correct user. Whenever a subscriber unsubscribes, a new topic key must be generated and communicated securely to the remaining subscribers and the publisher. It is also important to note that the traffic generated from re-keying grows quadratically with the group size.

In QUIP, we have chosen to relax the requirement for privacy of subscriptions because we feel that for many applications including web comics only a moderate amount of privacy is necessary. We feel that the mechanisms provided in Eventguard are unnecessary in our context, and are insufficient where privacy is essential. Specifically, with Eventguard the name of topics is obscured but only from random observers, since every subscriber to a particular topic knows the name of that topic, they can identify the name of that topic in any traffic they see and may then determine other subscribers to the same topic. We also feel that since we are considering the case of web comics as an example and this sort of privacy does not exist in the current situation of viewing web comics via HTTP, this is not expected by likely subscribers.

Other work in this area has had different priorities from ours. Opychral and Prakash (2001) focus on ensuring that only authorised subscribers know the contents of a particular event. They propose an interesting group key caching scheme which is used to improve efficiency. They only encrypt the content during the last network hop from the publish/subscribe router to the subscriber. They do not endeavour to secure the content while it is being transferred from router to router and therefore they must trust all routers.

Raiciu and Rosenblum (2005) work on the problem of keeping the details of subscription filters and event notifications confidential (a secret from the routing nodes along the path) while maintaining the routers' ability to route the notifications to the correct subscribers. They assume that subscribers and publishers are honest and that secure channels are established. They also assume that the routers themselves are semi-honest and follow the protocol (for example, that the routers will forward an event if it matches and not just drop it).

We have chosen a different threat model and requirements, as discussed in the following section.

3 Threat model and requirements

We will consider two kinds of requirements for our publish/subscribe system: security requirements and performance requirements. Of these two, we prioritise security requirements but are still concerned with performance. We consider the additional work required for QUIP in our analysis of performance, as each publish/subscribe overlay network will have a different

baseline performance. We consider performance to be adequate provided that all types of transaction (subscription, unsubscription, publication of content, advertisement of new publications and unadvertisement of publications) occur within a few minutes. We do not require these to be instantaneous, but would like them to be as quick as possible. This level of performance is sufficient for many applications including a publish/subscribe network focused on web comics.

We propose a conservative threat model with as few assumptions as possible. An actual adversary may not have all the capabilities we propose, but we prefer to consider security against a worst case adversary. We consider an adversary to have the ability to add, delete and modify network traffic at both the underlying network layer and the publish/subscribe overlay layer. We also assume an adversary who may be a valid participant in the protocol or may collude with some other number of valid participants. We assume that the key server is trustworthy.

A summary of the security goals for the system are:

- To protect the content such that only authorised subscribers may read it
- To protect the payment information such that only the key server learns it
- To authenticate the source of messages from subscribers, publishers and the key server to one another
- To protect the integrity of messages in transit

Wang *et al.* (2002) provide a survey of security issues for publish/subscribe systems. They mention some security issues we have chosen not to address, such as subscription confidentiality and network availability. Although we have not addressed these issues directly, there is other work (Raiciu & Rosenblum 2005, Castro, Druschel, Ganesh, Rowstron & Wallach 2002) which does address these issues by making changes to the publish/subscribe overlay itself and could therefore still be used in conjunction with QUIP. Wang *et al.* also discuss content authentication and encryption, but assume that the only ways to achieve them involve either a *Public Key Infrastructure* or a set of keys shared between each publisher and subscriber. Using a PKI solution for content distribution is inefficient for a large number of subscribers since the content needs to be encrypted with each subscriber's individual key. Likewise, if we consider a system with a large number of shared keys we see that this would remove many of the benefits of the publish/subscribe system such as scalability and the separation of publishers from subscribers. Our traitor tracing approach using a key server avoids these problems.

In any restricted content system, it is possible for authorised subscribers to copy or release the content once they have validly decrypted it. It is difficult to see how this issue can be overcome and we do not attempt to resolve it in this paper. Our solution improves the situation by requiring the traitor to actively resend the decrypted content in an ongoing fashion or risk being caught for sharing the secret key. This increases the amount of work necessary for unauthorised copying and introduces some delay in the availability of unauthorised content.

4 Technical description of QUIP

We propose adding valuable security properties to existing publish/subscribe overlay services without restricting the system to a particular overlay network. We do not assume that the overlay service provides any security properties. Eugster *et al.* (2003) discuss the main types of publish/subscribe system, *topic-based*, *type-based* and *content-based*. We believe QUIP could be used in conjunction with any of these types, however we have given examples for a *content-based* publish/subscribe system such as Siena (Carzaniga, Rosenblum & Wolf 1998, Carzaniga, Rosenblum & Wolf 2000).

$S_A(x)$	x signed by participant A
$\{x\}_{K_A}$	Encryption of x with the key K_A which was chosen by participant A
x, y	Concatenation of x and y
$H(x)$	Cryptographic hash of x
n	A nonce (fresh unpredictable value)
ID_A	A unique identifier for participant A

Table 1: Explanation of notation used in equations

There is a single trusted authority which will handle key management and payment called the key server. The key server separates subscribers from publishers. Each publisher sets the prices of their publications individually. If the publisher chooses to make their publication freely available, no keys are necessary so the key server is not involved in the transactions at all. If the publisher chooses to secure a publication, the authority makes sure that payment has occurred and provides the appropriate keys to the subscriber and publisher. QUIP uses a key management system described in Section 4.6. Unlike previous work such as Eventguard, when subscribers are added or removed in QUIP, only the publisher's key must be changed. Existing subscribers are not affected.

There are five major publish/subscribe operations which we want to secure. These are advertisement, unadvertisement, subscription, unsubscription and publication. We will discuss the protocol used for each of these operations in the sections below. We will also discuss an initialisation phase which happens only once as each new client joins the system.

We will describe the protocol we propose throughout Section 4 using the notation given in Table 1. Our protocol uses some basic cryptographic primitives including digital signatures, symmetric and asymmetric encryption, and cryptographic hash functions.

4.1 Initialisation

When an individual joins the service, they download a software client and are given a randomly generated identification number (ID) by the key server. The software client includes the public key of the key server, which is used to set up an initial secure tunnel. The key server will also generate a certificate linking this ID with a public key for each client. The certificates are used to authenticate signatures throughout the protocol. This enables us to require that messages be signed by both publishers and subscribers. We can then determine the ID of the participant who has originated a particular signed message. We have chosen to use a randomly generated ID to preserve some privacy related to the identity of clients. It is important to note that this privacy is only from other subscribers, publishers and attackers but not the key server. If the subscriber uses any paid service, the key server will have an entry in its database containing the subscriber's payment details linked with their ID.

In the initialisation phase, the client (designated as A in the equations) requests an ID and corresponding certificate from the key server (KS) as shown in

Equation 2. The initial request, a session key chosen by A and the public key of A are encrypted with the key server's public key which was downloaded with the QUIP client.

The key server generates a random ID and a certificate linking the public key provided by A to the new ID, encrypts them with the session key sent by A in the request and sends them back to A as shown in Equation 3.

$$A \rightarrow KS : \{\mathsf{ID\_request}, K_A, K_{A\_pub}\}_{KS\_pub} \quad (2)$$

$$KS \rightarrow A : \{ID_A, Cert_{ID_A}\}_{K_A} \quad (3)$$

4.2 Advertisement

Publishers must advertise their publications so that subscribers know what is available for subscription. This is done in two phases, the publisher must first inform the publish/subscribe overlay about the advertisement. If the publisher wishes to secure the publication, then the key server must also be informed. The publisher must send the key server the topic or title of the publication ($PubTitle$), the cost and the billing period (which is the same as the key change period). This message also includes the ID of the publisher and a fresh nonce as shown in Equation 4. These values are included so that these requests cannot be replayed by an attacker. The key server will verify that the signature on the message matches the ID contained in the message, that the nonce has not been used (by this ID) previously and that the $PubTitle$ is unique. If all these checks pass, the key server will respond as shown in Equation 5 confirming the request.

$$A_{publisher} \rightarrow KS : S_A(\mathsf{Advertise}, PubTitle, Cost, BillingPeriod, ID_A, n) \quad (4)$$

$$KS \rightarrow A_{publisher} : S_{KS}(\mathsf{AdvertiseAccept}, PubTitle, Cost, BillingPeriod, ID_A, n) \quad (5)$$

4.2.1 Unadvertisement

A publisher may choose to stop offering a publication. The revocation of an advertisement must be done both within the publish/subscribe system and the key server. The key server will then stop accepting subscriptions for the publication. The messages sent are identical to the ones above in Section 4.2 for advertisement except that the initial value changes from Advertise to Unadvertise.

4.3 Subscription

When a client wishes to subscribe to a new publication this is also done in two phases. First the subscription process within the publish/subscribe overlay is followed so that the client receives the content. Then the client will check if the content is encrypted or not. If the content is not encrypted, the client displays it. If the content is encrypted, the client contacts the key server to find out the subscription costs, as shown in Equation 6. A nonce is again included to ensure that the cost request and response are current. The key server responds giving the cost, the billing period and the start time of the next billing period, as shown in Equation 7. If the subscriber wishes to subscribe, they will respond as shown in Equation 8. The key server will again check the signature and that all the details match before processing the request. The key server will request payment information (as described in Section 4.5) if it does not already have the details, or the details are rejected by the payment system. The key server will then check that the payment information is valid before providing the subscriber with the content decryption key which will begin to work at the start of the next billing cycle. This is shown in Equation 9.

$$A_{subs} \rightarrow KS : S_A(PubTitle, \mathsf{CostQuery}, ID_A, n) \quad (6)$$

$$KS \rightarrow A_{subs} : S_{KS}(PubTitle, Cost, BillingPeriod, BillingStart, ID_A, n) \quad (7)$$

These cost query messages may be encrypted for extra privacy but we feel that in general it would not be worth the extra computation and complexity given that it would only hinder an attacker who is monitoring the conversation between the client and the key server but not the publish/subscribe overlay. An attacker who is monitoring the publish/subscribe overlay will know which publications a subscriber is receiving. It is possible for the subscriber to confound this sort of monitoring by subscribing to many publications without actually reading the content, but again, we feel these gains are probably not worth the effort.

$$A_{subs} \rightarrow KS : S_A(\{PubTitle, \mathsf{Subscribe}, Cost, BillingPeriod, BillingStart, ID_A, n\}_{KS\_pub}) \quad (8)$$

$$KS \rightarrow A_{subs} : S_{KS}(\{PubTitle, BillingStart, K_{PubTitle}, ID_A, n\}_{K_{ID_A\_pub}}) \quad (9)$$

4.3.1 Unsubscription

When a subscriber no longer wishes to receive a publication, they must notify both the publish/subscribe overlay and the key server of their intentions. This is very straightforward. The subscriber uses the same message format as for subscribing as given in Equation 8 except that the keyword Subscribe is replaced with Unsubscribe.

4.4 Publication

Publication is the publish/subscribe operation which is performed most frequently. A publisher will encrypt and sign their publication with the current content key provided to them by the key server as shown in Equation 10. Also included is an *enabling block*, discussed in Section 4.6. The encrypted publication is then encapsulated inside the normal publish/subscribe publication format. The key server will generate a new content key for the publisher at each billing interval if the set of subscribers changes. It is important to note that the subscriber's content keys do not change.

$$A_{publisher} \rightarrow Publish/subscribe\_overlay : S_A(\{K_{Content}\}_{Enabling\_block}, Enabling\_block, \{Publication\}_{K_{Content}}) \quad (10)$$

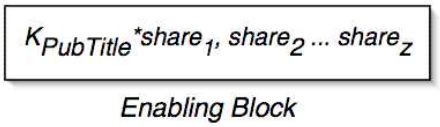

Figure 3: An illustration of the format of an enabling block.

4.5 Payment details

Occasionally it will be necessary for both subscribers and publishers to provide payment information to the key server. Since this information will change infrequently, it makes sense to send it only when necessary. The key server may request the payment details (as shown in Equation 11) or the publishers and subscribers may volunteer them at any time (as shown in Equation 12).

$$KS \rightarrow A_{client} : S_{KS}(\mathsf{PaymentdetailsRequest}, ID_A, n) \quad (11)$$

$$A_{client} \rightarrow KS : S_A(\{PaymentDetails, ID_A, n\}K_{KS\_pub}) \quad (12)$$

4.6 Key management and traitor tracing

The cryptography used has a significant impact on the performance of any secure system. We propose using a public key traitor tracing scheme designed by Tzeng and Tzeng (2001) which provides many advantages over symmetric and traditional asymmetric encryption schemes in this setting. There are two main advantages. First is the ability to revoke the keys of some subscribers without affecting the keys of the other subscribers. The second is the fact that each subscriber has a unique key which makes it easier to tell who has leaked a key if one is found in public or used by an unauthorised person. Further it is possible to discover who has leaked their keys even if they have combined their shares to create a new key. Once the traitor has been discovered the traitor's share can be revoked which will also disable the new key.

If the total number of subscribers is given by s, it is possible to revoke the keys of some subset of the total number of subscribers given by z. We must estimate s and generate that many shares of the key in advance. Encryption is done by creating an *enabling block* which contains z shares as shown in Figure 3. The enabling block is then used to encrypt the symmetric content key $K_{Content}$. This value is concatenated with the content encrypted with $K_{Content}$ as shown in Equation 10. The enabling block contains z shares (each subscriber key is a unique share), the shares contained in the enabling block are **not** able to decrypt the content key (and therefore not able to decrypt the content). When we wish to revoke a share, we simply include it in the enabling block when we encrypt. We fill in the rest of the z shares in the enabling block with unused shares. When the subscriber receives the publication they will first use their share of the publication key referred to as $K_{PubTitle}$ in Equation 9 to decrypt the content key ($K_{Content}$) and then use the content key to decrypt and read the publication.

By using a content key we can limit the extra computation necessary to use the public key traitor tracing scheme. The small (128 bit) content key is the only value we encrypt with the slower scheme. The computation time is also dependent on the number of shares we can revoke (z) instead of the total number of subscribers s. By encrypting a constant length value (the content key) we know exactly how long it will take for each new publication. If the subscriber base has not changed, we can reuse the same *enabling block* for multiple publications. The actual content which may be of varying lengths is encrypted using the content key and a symmetric algorithm which will be significantly faster and may also be pre-computed once the content has been determined (before the billing period might have finished and the subscriber set is known).

The key server handles all key management for the system. This authority provides a single point of failure so we have kept the involvement of the authority to a minimum. The key server is involved only when the publisher chooses to require security for their publication. It may be necessary to issue a new key if the estimate of the number of subscribers s is too small and all the available shares have been used, or if the number of shares we wish to revoke is greater than z.

5 Performance analysis

Preliminary performance calculations based on the average computation time needed for each type of cryptographic operation show that a typical desktop PC is sufficient for use as the key server. We have shown the additional calculations required by QUIP from each of the participants for each publish/subscribe operation in Table 2 (additional relative to an unsecured publish/subscribe overlay). This is an improvement on the amount of computation required as compared to other secure additions to publish/subscribe systems mainly because the publisher only needs to encrypt each publication once. It is also an improvement on the number of messages that must be sent because any number of subscriber changes results in only one message from the key server to the publisher each billing period (a typical symmetric encryption based system would require a message to each subscriber and the publisher each billing period).

There is also a small amount of additional memory needed. In order to avoid replay attacks, the key server must keep a list of all the nonces used by a particular ID in subscription and advertisement requests.

6 Conclusion

QUIP addresses a number of important security issues in publish/subscribe systems. It allows subscribers to authenticate the messages they receive from publishers, and it permits publishers to restrict their messages to authorised subscribers and to add and remove subscribers efficiently. It represents a significant improvement over previous work in securing publish/subscribe networks because it properly addresses the issue of key management and therefore does not need to make strong assumptions about the security of the underlying infrastructure. The traitor-tracing approach makes addition and revocation of subscribers much more efficient than in previous work. Publishers who are happy for their content to be freely distributed do not need to do any extra work even if the publish/subscribe system offers QUIP.

However, any security protocol makes some assumptions about the power of the adversary. Perhaps the most serious in our paper is the assumption that the key server cannot be subverted. This represents a single point of failure. An interesting open question is whether Tzeng and Tzeng's traitor-tracing scheme

Participant	Operation	Extra computation
Client	Initialise	Asymmetric encrypt, Generate public/private key pair
Key Server	Initialise	Generate random ID, Create certificate, Symmetric encrypt
Subscriber	Subscribe	Signature
Key Server	Subscribe	Signature check, Update subscriber set
Subscriber	Unsubscribe	Signature
Key Server	Unsubscribe	Signature check, Update subscriber set
Publisher	Advertise	Signature
Key Server	Advertise	Create new public traitor tracing key shares
Publisher	Unadvertise	Signature
Key Server	Unadvertise	Signature check
Publisher	Publish	Signature, Public traitor tracing encrypt content key, Symmetric encrypt content
Subscriber	Publish	Signature check, Public traitor tracing decrypt content key, Symmetric decrypt content

Table 2: Additional computation required by QUIP

can be extended to allow the key server to be distributed. Ideally, we could both improve security and reduce the load on each server by having several key servers, of which each subscriber needed to contact a subset. Then the adversary would have to subvert several servers.

A related but more difficult issue is the auditability of the key server, as mentioned by Wang *et al* (2002). A publisher who is suspicious of being underpaid has no way of testing the key server's honesty. The server can simply claim that there are fewer paying subscribers than there really are. Unfortunately it is difficult to see how the publisher could test this except in some out-of-band way. It seems unreasonable to assume that a publisher can observe communications to or from the key server. Another issue equally difficult to resolve is the problem of cheating publishers, who collect their money but fail to deliver appropriate content when promised. For example, it seems impossible for the system to detect when a cartoon has been published before, or is too dark or blurry to read. Perhaps a reputation system would mitigate this problem.

Depending on the content being delivered, different applications of publish/subscribe systems have different security requirements. We have addressed one such set of requirements in this paper, and we expect our solution to be consistent with other protocols that provide subscription confidentiality or guarantee message delivery.

References

Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. (1998), Design of a scalable event notification service: Interface and architecture, Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado.

Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. (2000), Achieving scalability and expressiveness in an internet-scale event notification service, *in* 'Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing', Portland, Oregon, pp. 219–227.

Castro, M., Druschel, P., Ganesh, A., Rowstron, A. & Wallach, D. S. (2002), Secure routing for structured peer-to peer overlay networks, *in* 'Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation (OSDI 2002)'.

Eugster, P. T., Felber, P. A., Guerraoui, R. & Kermarrec, A.-M. (2003), 'The many faces of publish/subscribe', *ACM Computing Surveys* **35**(2), 114–131.

Opyrchal, L. & Prakash, A. (2001), Secure distribution of events in content-based publish subscribe systems, *in* 'Proceedings of the 10th USENIX Security Symposium'.

Raiciu, C. & Rosenblum, D. S. (2005), Enabling confidentiality in content-based publish/subscribe infrastructures, Technical report, University College London:w Technical Report RN/05/30.

Srivatsa, M. & Liu, L. (2005), Securing publish-subscribe overlay services with eventguard, *in* 'Proceedings of the 12th ACM conference on Computer and communications security'.

Tzeng, W.-G. & Tzeng, Z.-J. (2001), A public-key traitor tracing scheme with revocation using dynamic shares, *in* 'Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography', Vol. 1992 of *Lecture Notes In Computer Science*, Springer-Verlag, London, UK, pp. 207–224.

Wang, C., Carzaniga, A., Evans, D. & Wolf, A. L. (2002), Security issues and requirements for Internet-scale publish-subscribe systems, *in* 'Proceedings of the Thirty-Fifth Annual Hawaii International Conference on System Sciences', Big Island, Hawaii.