UNIVERSITY OF TRENTO - ITALY

ICT DOCTORAL SCHOOL
DEPARMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE

Security of Publish/Subscribe Systems

A dissertation submitted by
Mihaela Ion

for the degree of
Doctor of Sciences

accepted on the recommendation of
Dr. Bruno Crispo, co-advisor, University of Trento, Italy
Dr. Giovanni Russello, co-advisor, University of Auckland, New Zealand
Prof. Dr. Ernesto Damiani, examiner, University of Milan, Italy
Dr. Brian LaMacchia, examiner, Microsoft Research, Redmond and University of Washington, US
Dr. Massimiliano Sala, examiner, University of Trento, Italy
Dr. Eve Schooler, examiner, Intel Labs, Santa Clara, US

MIHAELA ION
University of Trento

Abstract:
The increasing demand for content-centric applications has motivated researchers to rethink and redesign the way information is stored and delivered on the Internet. Increasingly, network traffic consists of content dissemination to multiple recipients. However, the host-centric architecture of the Internet was designed for point-to-point communication between two fixed endpoints. As a result, there is a mismatch between the current Internet architecture and current data or content-centric applications, where users demand data, regardless of the source of the information, which in many cases is unknown to them.

Content-based networking has been proposed to address such demands with the advantage of increased efficiency, network load reduction, low latency, and energy efficiency. The publish/subscribe (pub/sub) communication paradigm is the most complex and mature example of such a network. Another example is Information Centric Networking (ICN), a global-scale version of pub/sub systems that aims at evolving the Internet from its host-based packet delivery to directly retrieving information by name. Both approaches completely decouple senders (or publishers) and receivers (or subscribers) being very suitable for content-distribution applications or event-driven applications such as instant news delivery, stock quote dissemination, and pervasive computing. To enable this capability, at the core of pub/sub systems are distributed routers or brokers that forward information based on its content. The basic operation that brokers need to perform is to match incoming messages or publications against registered interests or subscriptions.

Though a lot of research has focused on increasing the networking efficiency, security has been only marginally addressed. We believe there are several reasons for this. First of all, security solutions designed for point-to-point communication such as symmetric-key encryption do not scale up to pub/sub systems or ICN applications, mainly because publishers and subscribers are decoupled and it is infeasible for them to establish or to maintain contact and therefore to exchange keying material. In this thesis we analyse several such emerging applications like Smart Energy Systems, Smart Cities and eHealth applications that require greater decoupling of publishers and subscribers, and possible full decoupling.

Second, in large applications that run over public networks and span several administrative domains, brokers cannot be trusted with the content of exchanged messages. Therefore, what pub/sub systems need are solutions that allow brokers to match the content of publications against subscriptions without learning anything about their content. This task is made even more difficult when subscriptions are complex, representing conjunctions and disjunctions of both numeric and non-numeric inequalities. The solutions we surveyed were unable to provide publication and subscription confidentiality, while at the same time supporting complex subscription filters and keeping key management scalable.

Another challenge for publish/subscribe systems is enforcing fine-grained access control policies on the content of publications. Access control policies are usually enforced by a trusted third party or by the owner holding the data. However, such solutions are not possible for pub/sub systems. When brokers are not trusted, even the policies themselves should remain private as they can reveal sensitive information about the data.

In this thesis we address these challenges and design a novel security solution for pub/sub systems when brokers are not trusted such that: (i) it provides confidentiality of publications and subscriptions, (ii) it does not require publishers and subscribers to share keys, (iii) it allows subscribers to express complex subscription filters in the form of general Boolean expressions of predicates, and (iv) it allows enforcing fine-grained access control policies on the data. We provide a security analysis of the scheme.

Furthermore, to secure data caching and replication in the network, a key requirement for ICN systems and recently also of pub/sub systems that extended brokers with database functionality, we show how our solution can be transformed in an encrypted search solution able to index publications at the broker side and allow subscribers to make encrypted queries. This is the first full-fledged multi-user encrypted search scheme that allows complex queries. We analyse the inference exposure of our index using different threat models.

To allow our encrypted routing solution to scale up to large applications or performance constrained applications that require real-time delivery of messages, we also discuss subscription indexing and the inference exposure of the index.

Finally, we implement our solution as a set of middleware-agnostic libraries and deploy them on two popular content-based networking implementations: a pub/sub system called PADRES, and an ICN called CCNx. Performance analysis shows that our solution is scalable.

Keywords: Security, Access Control, Publish/Subscribe, Attribute-based Encryption, Multi-User Encrypted Search

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Content-based networking

The increasing demand for content-distribution applications is motivating researchers to rethink and redesign the way information is stored and delivered on the Internet. The IP model no longer matches the way users and applications request and share information today. Users are more interested in sharing and retrieving information, and care less about which specific end point is holding the information. Furthermore, network traffic increasingly consists of content dissemination to multiple recipients. Content producers are both large organizations such as news agencies, movie studios, but also users who share photos or videos with friends on social networks. Thus, the host-centric architecture of the Internet designed for point-to-point communication between two fixed endpoints is not scalable or relevant for the current data-centric applications.

Content-based networking has been proposed to address such demands, motivated by both application-level and network-level considerations. At the application-level, consumers are more interested in expressing *what* content they are interested in retrieving, and less about *where* that content can be found. At the network level, identifying content rather than location allows more efficient networking by duplicating and caching content in the network. The shift from host-centric to content-centric networking has several advantages, such as network load reduction, low dissemination latency, and energy efficiency. Van Jacobson described content-based networking as the third revolution in telecommunication networks, as we move from connecting wires (public switched telephone network [PSTN]) to connecting nodes (all-IP networks) to connecting information [Pentikousis 2012].

Two kinds of content-based communication models have been proposed independently: publish/subscribe [Carzaniga 2001, Eugster 2003] and Information-Centric

Networking [Jacobson 2007]. The publish/subscribe (pub/sub) paradigm has been around for over 25 years and is now used in many applications such as instant news delivery, stock quotes dissemination, and pervasive computing. Information-Centric Networking (ICN) is a relatively new research field, described as a global-scale version of the publish/subscribe paradigm [Ghodsi 2011]. ICN aims at evolving the Internet from its host-based packet delivery to directly retrieving information by name [Ahlgren 2012]. Both pub/sub and ICN focus on finding and delivering information to users instead of connecting end hosts that exchange information. Implementations of these two paradigms are generally designed to run alongside or independent of TCP/IP, and do not disrupt existing networks. However, more ambitious designs aim at replacing TCP/IP [Fotiou 2012].

Apart from current content-dissemination applications, the shift to content-based networking is also motivated by new emerging applications such as Smart Grids and Smart Cities, mobile and pervasive applications. Moreover, as the number of mobile devices and sensors connected to the Internet is increasing, content-centric networks are becoming even more important. First of all, they allow asynchronous communication between devices which do not need to be online at the same time to communicate. This is especially important for mobile devices which are turned off or in sleep mode to save energy when not used or may not be online all the time due to mobility. Second, ICNs are build around the idea of in-network content storage, thus making data available a greater percentage of the time and allowing mobile devices the freedom to be turned off more often to save power. This feature is also provided by more recent publish/subscribe systems (e.g., PADRES [Jacobsen 2010]) which cache publications in the network and allow users to query them later.

1.1.1 Publish/Subscribe vs. Information Centric Networking

Though both pub/sub systems and ICN are designed for forwarding data based on its name or content, there are some important differences between them which come from the fact that they were designed with different applications in mind. Pub/sub systems were intended for event-driven applications such as instant news delivery or workflow management. For that reason, (i) publications are usually valid for a short period of time and are deleted once they reach the intended subscribers, (ii) the communication is initiated by the publishers, and (iii) subscriber interests are valid until unsubscribed. ICN were designed for retrieving data such as documents or media files and because of that, (i) ICN focus on caching data in the network to increase availability, (ii) the communication is initiated by the receiver, and (iii) interests are cancelled once the data is delivered. Because many applications need both kinds of communication models, unifying both approaches has been proposed [Carzaniga 2011] and systems that combine both pub/sub and ICN already exist. For example, the publish-subscribe Internet (PSI) architecture [Xylomenos 2012] has been proposed as an ICN approach to the future Internet with higher support for mobility and in-network caching. [Zhang] builds a simple pub/sub system on top of an ICN for a secure Home Energy Management System (HEMS). Another

example is the PADRES pub/sub system [Jacobsen 2010] which enhanced brokers with databases in order to enable historic data queries.

Last but not least, another important difference between the two models is the expressiveness of the subscription or interest. Pub/sub systems use more expressive subscription filters that range from names to general Boolean expressions of predicates, while ICN use only names to request data.

In this thesis we target a security solution that can work with any content-based networking solution and implement our libraries to be middleware-agnostic. We design our solution to meet all the needs of the more complex pub/sub systems and show that it can also be applied to ICN with a concrete implementation. In fact, we can even integrate our encrypted filtering algorithm with ICN, thus enabling more expressive interests on top of the naming provided by ICN.

1.2 Security challenges

Content-based networking requires new security mechanisms to address its specific communication model. Traditionally, security mechanisms and access control policies are enforced point-to-point or through the use of trusted third parties. Such mechanisms were designed for the host-centric point-to-point Internet and are not suitable in content-based networks where data creators and consumers are decoupled and not aware of each other. Moreover, because the network can be public, it cannot be trusted to enforce access control mechanisms. Once the data is published on the network, publishers lose all control over who gets access to their data. What content-based networking needs is to explicitly secure the content itself, as opposed to securing the end-to-end communication channel or end-points. Because the data is sometimes cached or stored in the network, we cannot rely on trusted end-points to control access to the data and enforce access control policies.

Most of the work on pub/sub and ICN is focusing on networking mechanisms and efficient routing, while leaving security mechanisms to be added in the future. ICN usually guarantees data integrity, binding names to content through signatures, but does not provide a specific key exchange and management mechanism or other security mechanisms such as encryption. However, there are many scenarios that require control over who can access the information. For example, a stock quote service could provide to paying customers information on stock prices by using a pub/sub system. In this case, only paying subscribers should be able to access messages. At the same time, subscribers may wish to keep the details of their interests private from anybody spying on the network. Unauthorised parties that are able to eavesdrop on messages or subscriptions should not be able to access their content. Another application scenario that can benefit from the use of pub/sub systems is in the medical sector where physicians are notified when certain events happen such as changes in the condition of a patient who is monitored by different medical devices or sensors. Such information should be available to the authorised personnel only to protect the patient's privacy.

To enable content-based networking, at the core of pub/sub systems are distributed routers or brokers that forward information based on its content. If the brokers are trusted, for example if they are under the direct control of the organization using the pub/sub system, the confidentiality of the events and filters can be ensured by securing the communication between brokers, between publishers and brokers, and between brokers and subscribers. However, in many scenarios brokers cannot be considered trusted, either because a malicious employee could get access to the data and misuse it, or because the pub/sub system has been outsourced to another company. Outsourcing the IT infrastructure is a business model adopted more and more by companies because it reduces costs and improves the quality of services and operations. In fact, even sectors such as healthcare, initially reluctant to adopt this model, are slowly employing it [Ondo 2006]. Because of that, there is a need for confidentiality and access control solutions that can be applied when brokers are untrusted and therefore could compromise the confidentiality of publications and subscriptions.

Another issue to consider is the privacy of the data transmitted over the pub/sub network, because it could contain sensitive information about individuals or organizations (e.g., personal and medical data). When such sensitive information is being sent, it should be possible to control how the data is disclosed and to whom. The disclosure of sensitive data is usually protected through the enforcement of *access control policies*. A policy specifies who can access the data and under which conditions. In pub/sub systems, publications or events contain multiple attributes and different access control policies could apply to each. For example, in an e-health application, the names of the patients should be disclosed to their doctors, but not to researchers that aggregate and analyse data from various patients. Access control solutions for pub/sub systems [Miklos 2002, Bacon 2008] require brokers to have access both to the policies and the content of events in order to enforce fine-grained policies on the attributes of the event (e.g., name, address). However, these solutions are not suitable for outsourced environments because they reveal the content of events to the untrusted brokers. On the other hand, encrypting only the content of events and leaving the access control policies unencrypted so that they can be enforced by brokers may not be sufficient because the policies could reveal sensitive information about the data. For example, if an event sent by the Gateway installed at a patient's home has attached a policy granting access to the data to a cardiologist, an attacker could infer that the patient suffers from a heart condition even if the actual event is encrypted.

1.3 Thesis contributions

This thesis makes the following contributions:

1. We describe several emerging applications requiring security mechanisms such as Smart Energy Systems, Smart Cities, and eHealth that rely on a pub/sub

system for communication. In these applications, a point-to-point communication model would not scale up.

2. We propose a novel solution for confidentiality and access control in pub/sub systems. Compared to existing solutions, our scheme is able to provide at the same time confidentiality of publications (or events) and subscriptions (or filters), fine-grained access control policies, and complex encrypted filtering of events, while not requiring publishers and subscribers to share keys.

3. We provide a novel solution for indexing and querying in-network cached encrypted publications. Most encrypted search solutions for databases provide keyword search or conjunctions of keywords. We provide a survey of existing schemes and show that none of them is able to support both multi-users that can read and write to the database with their own unique key, and complex queries. Our solution is the first one to have such features.

4. We provide a novel solution for efficient encrypted routing. Though efficient filtering algorithms have been researched intensively for non-encrypted pub/sub systems, confidentiality preserving filtering algorithms do not scale well when the number of subscriptions increases. Moreover, they have other limitations such as less expressive filters (e.g., only keyword match) and key sharing. We propose a solution for an efficient and scalable filtering algorithm, while maintaining the properties of our confidentiality and access control scheme.

5. We implement our solutions as middleware-agnostic libraries and integrate and test them with popular versions of both a pub/sub and an ICN system.

1.4 Thesis outline

The organization of the thesis chapter by chapter is as follows:

Chapter 2 describes the properties of publish/subscribe systems and several research applications such as Smart Energy Systems, Smart Cities, and remote patient monitoring in healthcare applications where a classic point-to-point model would not scale up. We then illustrate the need for security requirements in such systems with a possible attack over an unsecured system.

Chapter 3 proposes a basic confidentiality solution that protects against the honest-but-curious threat model, the most referenced threat model for pub/sub systems in literature. We define a set of security properties needed to protect against it, while at the same time preserving the decoupling and expressive filtering properties of pub/sub systems. We review existing solutions addressing confidentiality and show that none of them provides the full range of security properties needed. We describe a novel solution that provides all of these properties and prove it is secure under the chosen-plaintext attack.

Chapter 4 enhances the proposed solution to allow enforcing fine-grained access control policies on the data without relying on a trusted third party. This solution addresses a more powerful threat model and extends the previous solution to defend against it.

Chapter 5 provides a novel solution that allows indexing and querying in-network cached publications. Current encrypted search solutions cannot support at the same time multi-users with read and write capabilities and complex encrypted queries. Our solution allows performing encrypted complex queries on the data, and multi-users, each user having a unique key that can be revoked without affecting the other users.

Chapter 6 enhances the scheme with an efficient encrypted matching algorithm that increases the performance of the scheme while still providing the same security properties.

Chapter 7 discusses the implementation of our middleware-agnostic libraries both with a classic pub/sub system enhanced with databases for publication caching, and with a popular information-centric network. We provide an extensive performance evaluation and comparison of the various schemes.

Finally, in Chapter 8 we provide a summary of our results and contributions, and discuss future directions.

Chapter 2

Publish/Subscribe Systems

Contents

2.1 The publish/subscribe communication paradigm

The publish/subscribe model is an asynchronous communication paradigm where senders, known as *publishers*, and receivers, known as *subscribers*, exchange messages in a loosely coupled manner, i.e., without establishing direct contact. The messages that publishers generate are called *events* or *publications*. Publishers do not send events directly to subscribers, instead a network of interconnected brokers is responsible for delivering the events to the interested subscribers. In fact, publishers do not know who receives their events and subscribers are not aware of the source of information. In order to receive events, subscribers need to register interest with a broker through a *filter* or *subscription*. When a new event is published, brokers forward it to all subscribers that expressed a filter matched by the event. Figure 2.1 shows a simple pub/sub network that forwards messages from publishers to interested subscribers.

The pub/sub communication paradigm has the advantage of allowing the full decoupling of the communicating entities [Eugster 2003] which enables dynamic and flexible information exchange between a large number of entities. The communicating parties do not need to know each other or establish contact in order to exchange content. Moreover, if durable subscription is enabled, publishers and subscribers do not need to actively participate in the interaction at the same time. If a subscriber is offline when a publisher creates an event, the broker will store the event until the subscriber becomes online and the event can be delivered.

These characteristics make the pub/sub communication model well suited for a wide range of information-driven and event-driven applications. For example, pub/sub has been proposed for information dissemination applications

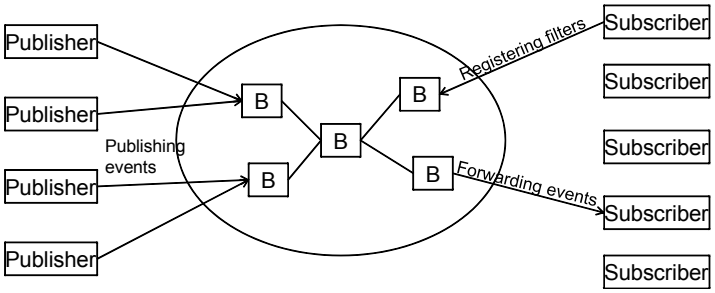


Figure 2.1: The pub/sub infrastructure connects publishers and subscribers via a network of interconnected brokers.

such as instant news delivery, stock market quotes distribution, auction bids [Bornhovd 2002], and air traffic control. Other applications of pub/sub are mobile systems [Cugola 2002b], ubiquitous computing [Langheinrich 2000], distributed workflow management systems [Cugola 2002a], and peer-to-peer systems [Heimbigner 2001].

2.2 Topic vs. content-based publish/subscribe

Several pub/sub implementations that differ in the granularity used in the definition of the filters have been proposed in the literature. The most simple one is *topic-based*, in which subscribers subscribe to a topic identified by a *keyword* [Zhuang 2001]. A topic-based scheme is similar to the notion of group communication. When subscribing to a topic T, a subscriber becomes a member of group T. When an event for topic T is published, the event is broadcasted to all the members of that group. Organizing topics in hierarchies allows a better management of subscriptions [Singhera 2008]. For example, by registering to a topic, a subscriber is also registered to all its subtopics. Figure 2.2 shows an example of hierarchical topics for an application that monitors ambient conditions such as temperatures, humidity and air quality in different towns.

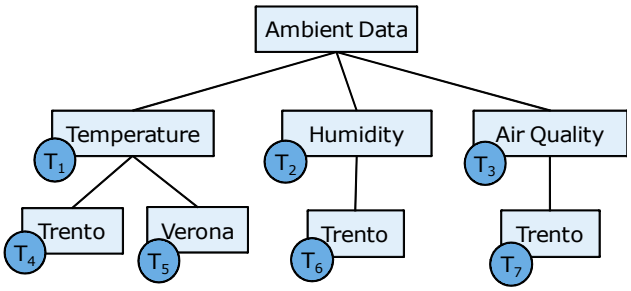


Figure 2.2: Topic-based subscriptions.

Topic-based schemes are easy to implement but they offer limited expressiveness. *Content-based* schemes are more flexible and allow specifying subscriptions based on the actual content of the event. To express a filter on the content of an event, subscribers need a query language and understanding of the data formats. For example, in Gryphon [Banavar 1999] and Siena [Carzaniga 2001] events consist of sets of ($attribute\_name = attribute\_value$) pairs and filters are specified as SQL WHERE clauses. Java Message Service (JMS) [Hapner 2002] does not allow filtering on the content of the event, but instead events carry properties in their headers and subscribers can define filters on them. Filters that apply to the composition of simple events have also been proposed (such as in [Bacon 2000]). When expressing such a filter, subscribers are notified upon the occurrence of the composite event.

Content-based schemes are more expressive and fit more naturally the way users search for information on the Internet. Table 2.1 compares the expressiveness of topic and content-based subscriptions for the scenario in Figure 2.2. For the first subscription, a user would need to subscribe to several topics to get all the events related to Trento. In the other examples, subscribing to a topic returns a super-set of the data, requiring the user to filter out undesired results. This causes overhead for the network and the users, and could be a real problem for users with limited resources using mobile devices.

Table 2.1: Topic-based vs. content-based filters.

Subscription	**Topic-based**	**Content-based**
All Trento data	T_4, T_6, T_7	Trento
All Trento temperature since 2008	T_4 (super-set)	temperature and Trento and year>2008
All Trento air quality with Air Quality Index>50	T_7 (super-set)	airquality and Trento and aqi>50

Because of its generality and expressiveness, we will focus on content-based filtering. Topic-based filtering can be considered a sub-case of content-based where filters contain only one attribute (or keyword). We assume that filters define constraints in the form of *attr_name-op-attr_value* where *op* can be one of the comparison operators such as $=, \leq, <, \geq$, and $>$. Constraints can be logically combined using *AND*, *OR* and *NOT* to form complex subscription patterns.

In the following we describe several applications that use a publish/subscribe system to enable many-to-many communication between a large number of loosely-coupled entities. For such applications, the classic point-to-point communication model, or even group communication would not scale up.

2.3 Application scenarios

The most common application scenario for pub/sub systems is on demand content delivery. Users subscribe to particular media content or news and usually pay a

subscription fee. Brokers deliver to them content matching their subscription every time new content is published. In the following we propose more complex scenarios inspired from emerging applications such as Smart Energy Systems, Smart Cities and eHealth. We use these scenarios to illustrate how information is published and routed in a pub/sub system. The entities, message types and scenarios we describe emerged from various research projects and represent our vision on how such novel systems could run using a pub/sub system. We will further use these examples to explain our different threat models and requirements for a secure pub/sub scheme.

2.3.1 Smart Energy Systems

Smart Energy Systems are large distributed systems that connect energy suppliers, consumers and their devices, and provide dedicated services that monitor and control energy consumption with the goal of reducing costs, optimizing energy usage and increasing the reliability of the Energy Grid. The classic point-to-point communication model does not scale up to a large Smart Energy System because data generated by one entity, e.g., a Smart Meter, is likely of interest to many entities such as an Energy Management and Control System (EMCS), a neighbourhood aggregator, a Utility Company, a Real-Time Pricing System, a Load Control System, many of which are unknown to the data publishing device. The pub/sub communication model was designed to deliver data asynchronously and reliably between a large number of loosely coupled entities in a many-to-many manner and fits well the communication requirements of a Smart Energy System, as shown in Figure 2.3.

A Smart Energy System provides dedicated services such as scheduling, actuators and real-time pricing to reduce energy consumption, lower peak time demand, reduce costs, and increase the reliability of the Energy Grid through the smoothing out or reduction of peak loads. A Smart Energy System could have the following entities:

- **Energy Monitor**: measures the real-time energy consumption of each device inside a house.
- **Energy Management & Control System (EMCS)**: installed in each house, a local EMCS schedules and turns on/off different devices in the house such as dish washer, Electric Vehicle (EV) charging, and air conditioning.
- **Customer Gateway**: acts as an interface between the Home Area Network (HAN) and neighborhood or city area network. All messages between the HAN and the external network are sent and received through the Gateway.
- **Home Area Network (HAN)**: provides connectivity among all the devices inside a house such as appliances, medical devices, Smart Meter, EV, solar panels, wind turbine, battery, the EMCS and Gateway.

- **Real-Time Pricing (RTP)**: calculates energy prices based on current demand and offer. The pricing is calculated in such a way that it incentivizes consumers to reduce peek consumption and overall fluctuations in demand.
- **EV Scheduler**: a service that receives status information and energy needs of the EVs in a neighborhood, and pricing information, and schedules how the EVs should be charged such that the network in not overloaded.
- **Neighborhood Area Network (NAN) Aggregator**: aggregates the information about the energy consumption in a neighborhood.
- **Energy Storage**: either inside a house or in a neighborhood, it is used to store energy in low demand periods and use it during high demand. If medical devices are in use, such storages become very important.
- **Load Control**: controls that the network is not overloaded, analyses peak/low demand periods, predicts future demand, detects problems and risks of blackouts.

A robust pub/sub system connects all these entities and reliably delivers messages between them as shown in Figure 2.3.

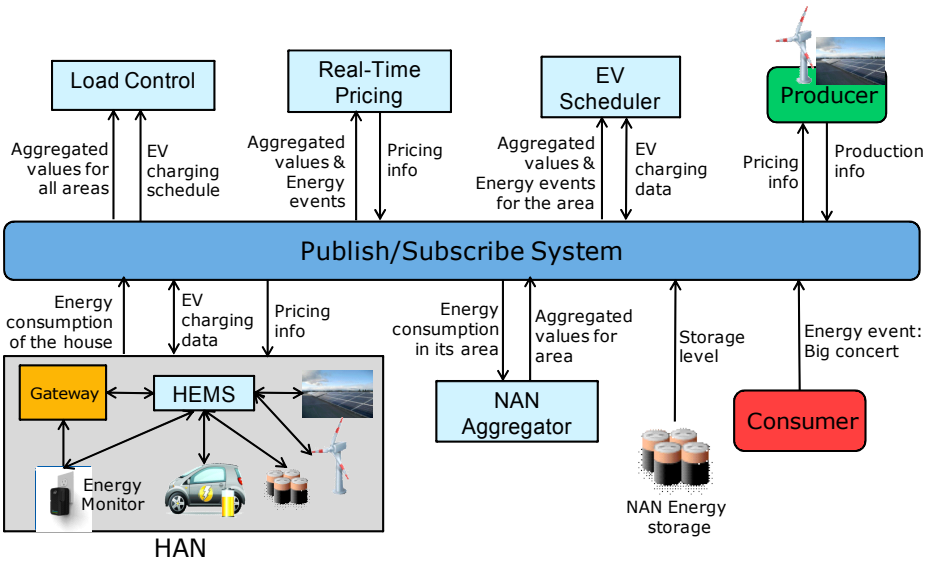


Figure 2.3: A Smart Energy System and data flows between main entities.

2.3.1.1 The EV Scheduling Use Case

In the following we take a particular use case and look into more detail at the messages exchanged and how they are routed by the publish/subscribe system. We developed this application together with the Energy and Sustainability Lab (ESL)

at Intel Labs, Santa Clara, US. Before charging their cars, users send a scheduling request to a neighbourhood EV scheduler that sends back the times and power at which the car should be charged such that it does not overload the network and lowers the cost for the consumer. In particular, the main goal of the scheduler is not to overload the local transformer which was not provisioned to support an EV for every household. In the future, when more people will own EVs, this could become a real problem. A study [pec a] conducted by the Pecan Street Project [pec b] in a test bed of 10 households owning EVs in the Mueller neighbourhood in Austin, Texas, in the US, over a period of two months, showed that people turn everything on at the same time, even during weekends. When people come home at the end of a day they turn on television, computer, air conditioning, other appliances, along with plugging in their EV for recharging. An EV charging at maximum power can use as much energy as an entire household, thus doubling the overload on the neighbourhood transformer. In this particular installation, a transformer can serve up to 12 houses and is very expensive to replace. Using an EV scheduling service would reduce the risk of overloading and damaging the transformer by coordinating across homes.

In Figure 2.4, we show potential subscriptions and messages for an EV scheduling use case. There are four publishers and four types of messages or publications. The message types are advertised by each publisher in the form of an *advertisement* which is sent before publishing any messages.

- A *pricing* message, published by the RTP, contains information about prices per time period and consumption class. The EVS and the HAN both registered subscriptions to receive such messages, i.e., *S3: class=pricing.*

- An *ev_charging* message is a charging request sent by a HAN. The EVS of a neighborhood subscribes to this type of messages, i.e., *S1:class=ev_charging AND ZIP=92000.*

- An *ev_schedule* message sent by the EV Scheduler in response to a charging request. The HAN subscribes to it with *S4: class=ev_schedule AND ID=xxx.*

- A *power_event* message published by a consumer when a special event is happening that will require a lot of power such as a concert. The EVS subscribes to this event with *S2: class=power_event and ZIP=92690.*

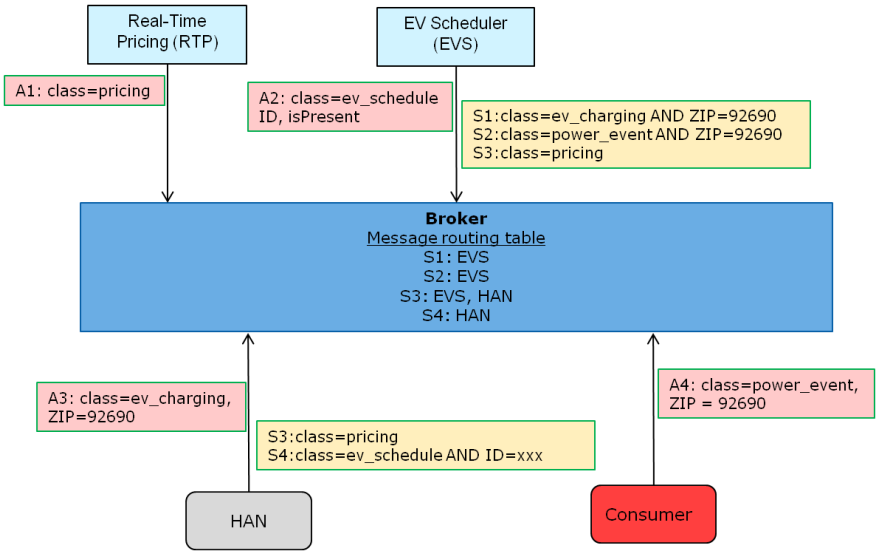


Figure 2.4: Subscriptions and message types for the EV charging use case.

Figure 2.4 shows a simple case in which only one broker is used to deliver messages from publishers to interested subscribers. The broker maintains a message routing table that has entries of the form: *subscription: destinations*. A message that matches the subscription will be forwarded to all the destinations paired with the subscription. Figure 2.4 shows the routing table for this example.

To increase the scalability and reliability of the system, pub/sub systems are usually distributed and messages are forwarded through multiple hops. Figure 2.5 shows the same example, but with several distributed brokers. We note that in this case each broker becomes either a publisher or a subscriber or both to its neighbouring brokers. Brokers only know their immediate neighbours and are unaware of the source of publications or subscriptions.

Many pub/sub systems [Fidler 2005] require publishers to send advertisements that describe the type of message they will publish. Advertisements are used to create subscription routing tables in the following way: Instead of broadcasting a subscription to all its neighbours, a broker only sends a subscription to a neighbour broker that sent an advertisement that matches the subscription. For example, in Figure 2.5, Broker1 only forwards subscription *S3: class=pricing* to Broker4 because that is its only neighbour that sent an advertisement matching *S3*.

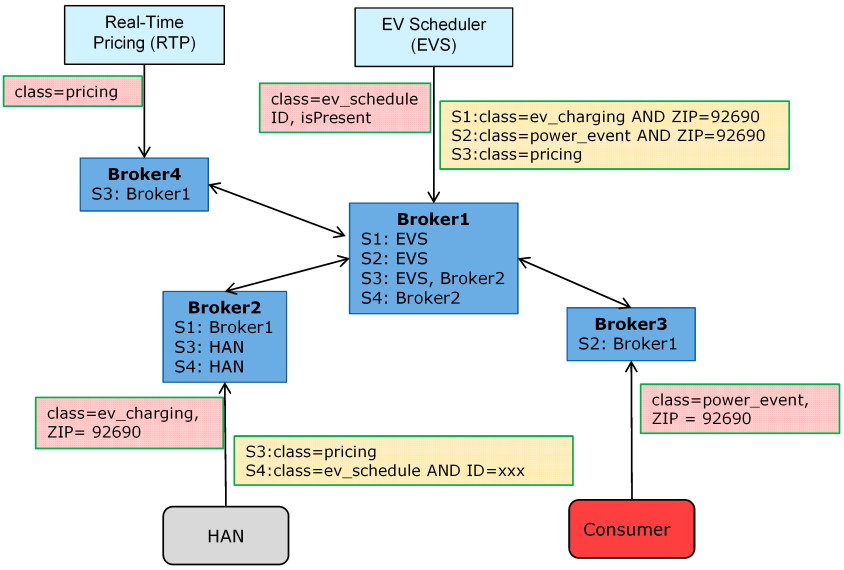


Figure 2.5: Subscription routing tables in a distributed scenario.

2.3.2 Smart Cities

A similar application is a Smart City in which large amounts of data are generated for example by sensors that monitor the ambient environment (e.g., pollution level, air quality, noise levels, flooding), the traffic (e.g., traffic congestion, accidents, closed roads, closed subway stations), and by people that report events etc. A pub/sub system would deliver messages from publishers only to interested subscribers without requiring publishers to know the addresses or identities of the subscribers. In the following we describe two applications that we developed together with the Energy and Sustainability Lab (ESL) at Intel Labs, Santa Clara, US.

2.3.2.1 Sensing the Smart City

Figure 2.6 shows an application for sensing, monitoring and informing users of environmental conditions. Several sensors measure the levels of humidity, temperature, pollution, and chemical substances in the environment and publish them on the pub/sub network. Neighbourhood aggregators subscribe to readings from their areas and discard outliers and average the readings, detect misbehaving sensors and inconsistencies between readings. They publish aggregated values on the network. Users in the neighbourhood and Air Quality Control centers subscribe and receive these aggregated values. The Air Quality Control center analyses the values and issues recommendations and warnings that are published on the network and are delivered to interested authorities, researchers or citizens in the area.

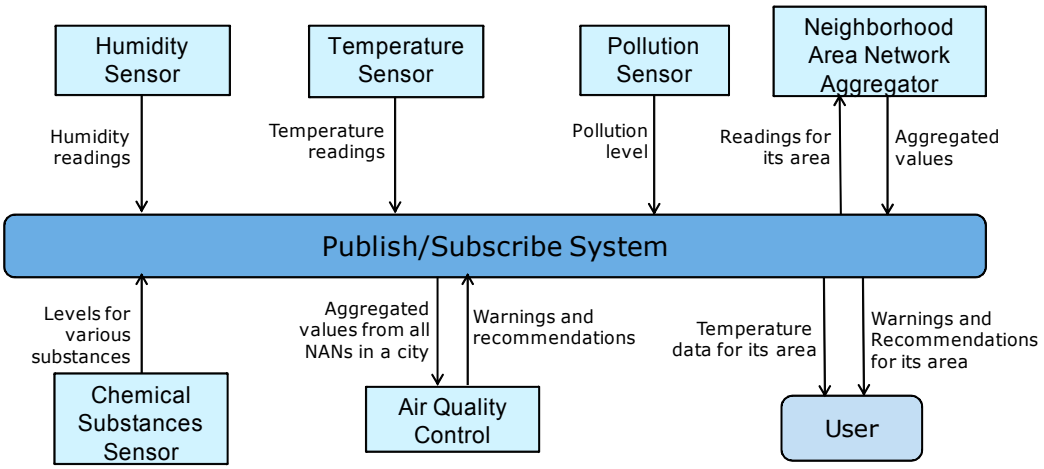


Figure 2.6: Publishing and subscribing to sensor information.

2.3.2.2 Mobility in the Smart City

Figure 2.7 shows a scenario in which subscriptions are location dependent. Let us consider the case of an electric car driving on a road that needs to charge its battery. The car may subscribe to messages about EV charging stations near its location or trajectory that have a specific time, power, availability and price. EV charging stations publish their location, availability and price on the network. It is the job of the pub/sub system to match the message published by different charging stations to the subscription conditions and then to deliver to subscribers only desired information. If the car driver sees an incident on the road, it may report it to the police and the message will be delivered to the closest police officer that subscribed to traffic incidents in the area.

2.3.3 Healthcare

In the following, we present the details of an e-health application that we developed together with the San Raffaele Hospital in Milan, Italy. The e-health application is designed for remotely monitoring patients with a chronic disease that do not require hospitalisation, such as heart disease or diabetes. While the patient is at home, it is necessary to continuously monitor specific vital sign parameters. Moreover, for these kinds of patients a continuous and correct lifestyle is fundamental in order to improve the quality of their lives. In particular, in patients with heart diseases it is important to monitor both some physiological parameters and the patients' habits, including the diet and the physical activities. This information needs to be distributed to the interested parties such as professional caregivers to provide required feedback, prescribe medicines, and schedule appointments in case of unexpected conditions.

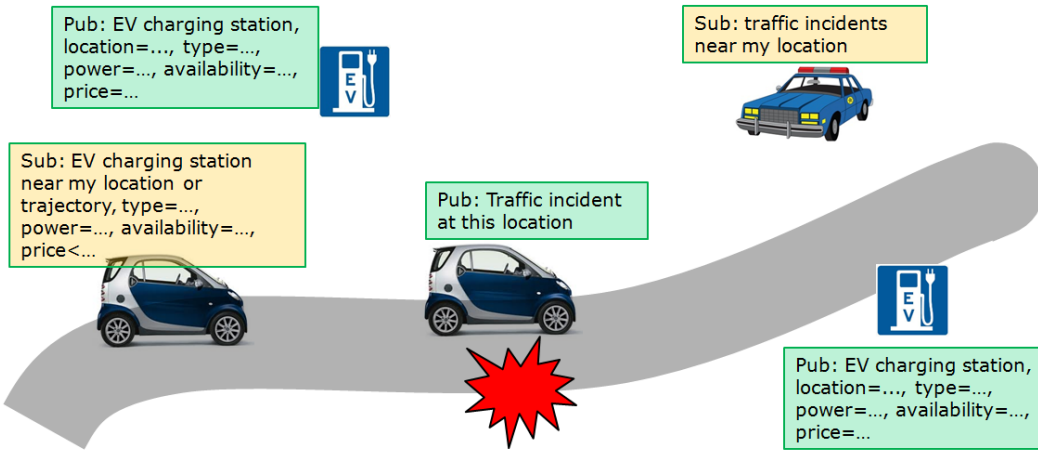


Figure 2.7: Mobility examples.

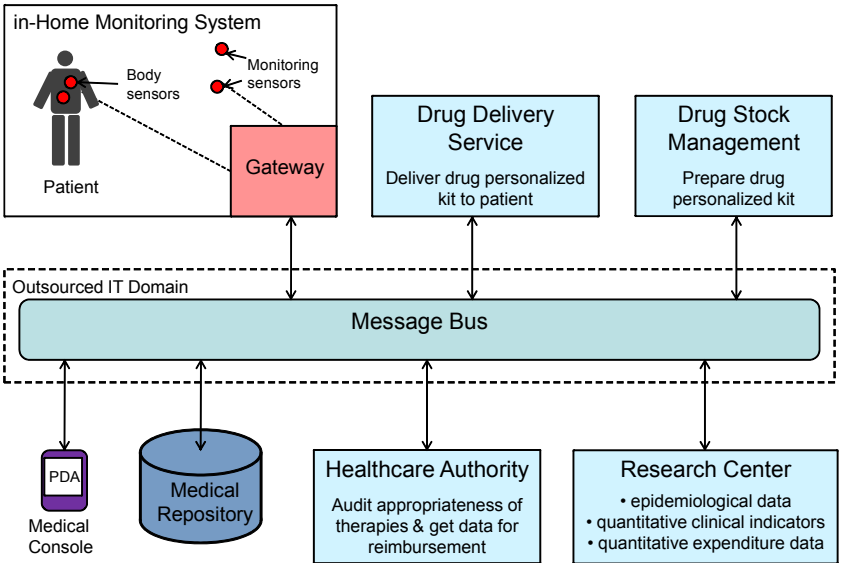


Figure 2.8: An e-health application scenario for monitoring chronic diseases.

The e-health application is composed of several distributed components that are shown in Figure 2.8. Each component is described in details as follows.

The **in-Home Monitoring System** (iHMS) is a component that performs the gathering of patient's data through the use of electronic devices self-managed by the patient. These devices have wireless means to connect to a central gateway where the data is gathered. Regarding the physiological parameters it could be useful to monitor the blood pressure, the heart rate and the ECG at regular time intervals. Regarding the life style, the relevant data to be monitored is the weight and the amount of physical exercises (e.g., walking and/or running). In our application, we employ the following devices:

- a wrist integrated device able to measure the blood pressure and the blood oxygen levels;
- a portable device for ECG and heart rate monitoring;
- a smart scale that automatically sends periodic weight measures;
- a device to be worn during physical exercises that can monitor some relevant parameters, such as the number of steps during a walking activity, the total energy consumption (kcal), the duration of the physical activity, etc.

In the hospital, the **Medical Console** (MC) retrieves the data collected by the iHMS. The patient's doctor can use the MC for accessing the vital signs and lifestyle data stored in the **Medical Repository** (MR). The MC can also alert the doctor if the values exceed some personalised thresholds. The doctor can inform the patient that a visit is required and some more specific medical tests have to be performed. The doctor can also decide to prescribe a therapy, composed of a list of drugs to be regularly assumed by the patient, and to give advice in order to improve the patient's lifestyle.

The drug therapy, depending on the specific disease, can include drugs that can be directly provided by the hospital and/or drugs to be purchased through the territorial pharmacies. In the former case, the drugs provided by the hospital are reimbursed by the **Healthcare Authority** (HA). In order to assure the appropriateness of the therapy and the exact amount of costs to be reimbursed, the hospital has to provide to the HA data related to the patient's conditions and drug costs.

The **Drug Stock Management** (DSM) is in charge of preparing a personalised drug kit, composed of the drugs prescribed by the doctor for a specific patient. To prepare the therapy kit, the doctor has to provide to the DSM personnel the therapeutic data. The drugs can be delivered at home thanks to a **Drug Delivery Service** (DDS) that performs the transportation directly to the patient's home.

Finally, there is a **Research Centre** that performs data processing on the received information from various hospitals and healthcare authorities. Examples of possible data processing are the analysis of epidemiological data (e.g., number of patients with heart diseases treated with a specific drug) and the calculation

of quantitative clinical and expenditure indicators (e.g., number of patients with repeated outpatient visits or total average cost for each patient).

2.4 Security requirements

We motivate the need for security mechanisms with an example of a possible attack that could be carried out on an unprotected pub/sub system. We use as reference the last application we introduced.

Let us assume that the San Raffaele hospital has decided to outsource the pub/sub system to an IT company such as a cloud provider that provides and maintains the servers where the service is deployed, and that no security mechanisms are in place. The pub/sub system connects all publishers and subscribers and allows them to send and receive events asynchronously through a shared set of interfaces. Subscribers can specify their interest in particular messages through filters. Filters are strings which express constraints on the attributes of the events.

If messages are sent in the clear, a malicious employee of the company managing the pub/sub system can easily get access to the events that are exchanged through the brokers. The attacker could also be another user of the same cloud provider that can mount a collocation attack [Ristenpart 2009] and capture the memory of the pub/sub broker application. Therefore, in this setting the patients' privacy is at risk. We illustrate the risk of violating the patients' privacy through an example.

Let us suppose that a patient named "John Smith" suffering from a heart condition is monitored remotely by a doctor. The medical devices carried by the patient monitor the blood pressure, heart rate and ECG. At regular time intervals, these values are published by the gateway.

The doctor monitoring the patient registered the following subscription filter with a broker to be notified immediately of any significant change in the patient's condition. The doctor expressed the following filter:

name="John Smith" AND (heart_rate>120 OR systolic_pressure>150 OR diastolic_pressure>100).

The doctor will be notified when values exceed the specified thresholds. When values are normal, they are logged in the patient's history and the doctor can retrieve them when necessary. If an employee is able to capture the filter, he will be able to infer that John Smith is suffering from a heart condition.

Now let us suppose that the doctor received an event which indicates a change in the patient's condition and needs to write the following prescription:

name="John Smith", age=70, address="via Tartini 12, Padova", symptom ="high blood pressure", disease="primary hypertension", medication="Catapres".

The doctor publishes this message through a broker. An employee of the company providing the pub/sub system could access the messages that come in and out of the broker. If the event is sent in cleartext, the employee would learn the personal data and medical condition of this patient, and even infer that the prescribed drugs will be delivered to the patient the next day at the specified address.

The employee could also access the filter registered by the doctor and infer that the patient John Smith has a heart problem. That is why, in order to protect sensitive data contained by events and filters transmitted over an untrusted outsourced pub/sub system, cryptographic mechanisms are needed to provide *event and filter confidentiality*. Moreover, because different policies apply to different attributes of the event, without a proper access control mechanism that can enforce *fine-grained access control policies*, legitimate subscribers would learn information that they are not supposed to. For example, if no access control mechanisms are in place, researchers would learn the names and addresses of patients, though only anonymized data should be available to them.

The next chapters will discuss in detail each of the these requirements. Chapter 3 provides a basic confidentiality solution, and Chapter 4 enhances the scheme to additionally support fine-grained access control policies.

Chapter 3

A Basic Confidentiality Scheme

Contents

3.1 Threat model

We start by assuming an honest-but-curious model for publishers, brokers and subscribers, as in most papers [Srivatsa 2007, Shikfa 2009]. This model assumes that although the entities in the system follow the protocol, they may be curious to learn information by analysing the messages (events or filters) that are exchanged on the message bus. For example, a broker may try to read the content of an event or try to learn the filtering constraints of subscribers. Subscribers may want to read the events delivered to other subscribers.

We assume there is at least one Trusted Authority which generates encryption and decryption keys used to protect data from unauthorised access. The authority does not misbehave and is trusted by all the entities of the system.

3.2 Required security properties

In the following we enumerate the properties that a confidentiality scheme for pub/sub systems should provide, assuming an honest-but-curious threat model. As previously discussed, because publishers and subscribers are decoupled and cannot always share secret keys, a crucial property that needs to be ensured by any encryption scheme for pub/sub systems is scalable key management that does not require establishing and maintaining shared (group) keys. We define this property as follows and we require it for our scheme:

Definition 1 (P1: Scalable key management)**.** *A simplified and scalable key management eliminates the need for publishers and subscribers to share keys and supports the loosely-coupled model of the pub/sub paradigm. Key sharing would require redistribution of new keys to all participants and re-encryption of all filters when a filter is unsubscribed, thus affecting the scalability of the system.*

Moreover, we require that the content of publications and subscriptions is protected and that the broker does not learn anything about them during the matching process. this leads to the following properties.

Definition 2 (P2: Publication confidentiality)**.** *The publication confidentiality property ensures that the content of the event is hidden from the brokers and only intended subscribers are able to decrypt the event. By intended subscribers we mean subscribers that registered a filter matched by the event. In some cases, subscribers might need to obtain an authorization from the Trusted Authority in order to register a filter.*

Definition 3 (P3: Subscription confidentiality)**.** *The subscription confidentiality property ensures that the details of the filters are hidden from the brokers. The broker should be able only to tell if an event matches a filter but gain no other information about the event or the filter.*

Definition 4 (P4: Complex encrypted matching)**.** *Brokers should be able to match complex encrypted filters against encrypted events without learning anything about the content of events or filters. By complex encrypted filters, we mean filters that can express conjunctions and disjunctions of equalities, inequalities and negations in an encrypted form.*

3.3 Related work

In the following we show that current solutions for ensuring confidentiality in pub/sub systems provide only some of the properties defined above, but not all

of them at the same time. We review the most significant schemes and show for each which of the desired properties are satisfied.

Khurana [Khurana 2005] proposes a scheme that targets confidentiality of events but not of filters. Events are encoded in XML format and only specific fields (e.g., price) are encrypted with a symmetric key k. The publisher then encrypts k with its public key and attaches it to the message. The brokers forward the events based on the fields left unencrypted and a proxy service changes the encryption of k to an encryption with the public key of the subscriber. This solution achieves partially *P2* encrypting only specific fields but not the entire event. Properties *P3* and *P4* are not addressed because events are forwarded based on unencrypted event fields and filters. Key management is scalable and does not require publishers and subscribers to share a key, hence achieving *P1*.

Raiciu et al. [Raiciu 2006] target simultaneous event and filter confidentiality. The method primarily encrypts only the attribute value. The name of the attribute can be hidden by concatenating it with the attribute type and size and then hashing it (*P2* and *P3* can be achieved). Publishers and subscribers are required to share a group key (*P1* is not achieved) which is used to encrypt events and filters. The subscriber uses the shared key to "garble" the circuit representation of the subscription function. The publisher encrypts the event in a way compatible with the subscriber's circuit. The broker inputs the encrypted event to the subscription circuit in order to check if there is match. The method can support equality filtering, range matching and keyword matching (*P4* is partially achieved).

Srivatsa et al. [Srivatsa 2007] propose a specific hierarchical key management scheme that achieves confidentiality of events (*P2*) and filters (*P3*). A trusted centralized authority distributes encryption keys to publishers and authorization keys to subscribers. To support range matching, keys are organized in a hierarchical structure, each key corresponding to an interval. An authorization key corresponds to a filter and is able to derive the encryption key for an event that matches the filter. Because all publishers and subscribers obtain the same keys, unsubscription requires rekeying. At specific time intervals, keys are regenerated and subscriptions need to be reconfirmed, which is the main disadvantage of this method. Property *P1* is thus not achieved. Each event has a routable topic attribute which is encrypted using an encrypted search technique. To prevent dictionary attacks on the events, the routable attributes are tokenized and transformed in pseudo-random chains. The approach is vulnerable to inference attacks which use information about the frequency at which events are published to learn information about an event. To prevent these attacks, a probabilistic multi-path event routing scheme is proposed at the cost of extra overhead. This method supports routing based on only one keyword (the topic), hence not achieving *P4*. It is possible to express inequality conditions but they can only be checked at the subscriber side and not by the brokers. When the subscriber receives an event matching the expressed topic, the authorization key of the subscriber will allow deriving a correct decryption key only if the numerical value of the attribute is in the range specified by the subscriber.

Shikfa et al. [Shikfa 2009] propose a solution based on multiple layer commuta-

tive encryption that achieves content and filter confidentiality (*P2* and *P3*). The method uses a local key management in which each node needs to share a secret key with the immediate r neighbours. This has the advantage that if a subscriber leaves the system, only local keys need to be revoked. However, this solution requires managing a large number of keys and does not adapt well to network changes. For example, in a pub/sub system with a single broker, each publisher would need to share a key with each subscriber and an event will need to be encrypted for each possible subscriber. Thus key management is not scalable and *P1* is not achieved. To avoid collusion attacks, r can be set as big as necessary. If r consecutive nodes collude, they can decrypt their children's subscriptions, but not the subscriptions of other nodes. Events and filters contain only one keyword. Encrypted routing tables are created for a single keyword and the matching operation is basically an equality test (*P4* is not achieved).

Chen et al. [Chen 2010] target information (*P2*) and subscription (*P3*) confidentiality and use symmetric encryption to achieve this goal. Their scheme requires the publisher to distribute a secret key k and a random number r to all subscribers. The random r is added to numerical values by both publisher and subscribers in order to hide the real values from the brokers. This method has the drawbacks that it requires a publisher to establish contact with all subscribers and to redistribute keys and re-encrypt subscriptions every time a subscriber leaves the system (*P1* is not achieved). The scheme can express equalities of numerical and non-numerical attributes, and numerical comparisons. The paper targets only filters with single constraint (*P4* is partially achieved).

Nabeel et al. [Nabeel 2009] propose a scheme based on Pedersen commitment and Pailier homomorphic encryption to achieve blinding of attribute values in notifications and filters. The scheme supports only equality of strings and numerical attributes and inequalities of numerical attributes, hence *P4* is only partially achieved. Because the attribute names are left unencrypted, *P2* and *P3* are only partially achieved. In order to register a filter, subscribers need to register themselves first with the publisher to obtain a private key k for decrypting the message content (encrypted with symmetric encryption) and secret values used to blind attribute values. Publishers use the counterparts secrets to encrypt the events. Because the scheme requires publishers and subscribers to share keys, *P1* is not achieved.

Choi et al. [Choi 2010] propose a scheme based on Asymmetric Scalar-product Preserving Encryption which allows comparing the distance between a data point and a query point with the distance between the same query point and another data point. This allows brokers to check equality and inequality conditions without learning the values of the attributes (*P2* and *P3* are achieved). The scheme supports equality, inequality, range and conjunction filtering (*P1* is partially achieved). The scheme requires that publishers and subscribers share a secret which is used for encrypting numerical values, hence *P4* is not achieved.

Maji & Bagchi [Maji 2012] propose v-CAPS, a confidentiality preserving routing protocol. The main drawback of this solution is that it requires subscribers to contact publishers and send them their subscription filters. Hence, property *P1* is

not achieved, and *P3* is only partially achieved as publishers learn the interests of subscribers. Furthermore, publishers are required to compute the filters matched by their events and also compute covering relations between filters, usually the task of the brokers. For each event, publishers compute a receiver vector (RV) containing the IDs of the filters that match the event. The RVs are encrypted using the encrypted search technique from [Song 2000b]. The solution achieves confidentiality of events (*P1*) and complex filters (*P4*) but is not scalable nor generally applicable.

Table 3.1: Properties achieved by current confidentiality schemes.

	P1: Scalable Key Management	**P2: Publication Confidentiality**	**P3: Subscription Confidentiality**	**P4: Complex Encrypted Filtering**
[Khurana 2005]	Yes	Partially	No	No
[Raiciu 2006]	No	Yes	Yes	Partially
[Srivatsa 2007]	No	Yes	Yes	No
[Shikfa 2009]	No	Yes	Yes	No
[Chen 2010]	No	Yes	Yes	No
[Nabeel 2009]	No	Partially	Partially	Partially
[Choi 2010]	No	Yes	Yes	Yes
[Maji 2012]	No	Yes	Partially	Yes

Table 3.1 summarizes which of the *P1-P4* properties are satisfied by each of the surveyed schemes. None of the solutions provides all the properties at the same time. We observe that in order to provide confidentiality of events and filters, current solutions limit the expressiveness of the filter. If more complex filters are allowed, confidentiality is provided only for specific attributes. Moreover, most solutions require publishers and subscribers to share a group key which hampers the loose coupling and scalability of the pub/sub model. Our goal is to propose a solution that can achieve both confidentiality of event and filters and complex filters while keeping key management and event routing scalable.

3.4 Background on security mechanisms

This section provides background information on the techniques used in our solution. We are particularly interested in encryption schemes that do not require publishers and subscribers to share keys such as proxy encryption, and in encrypted search techniques that possess the same property.

3.4.1 Proxy encryption

Proxy encryption techniques [Canetti 2007] rely on a proxy server to transform a ciphertext encrypted under A's key into a ciphertext of the same message that can be decrypted by B's key. While performing the transformation, the proxy server does not learn the content of the message. Such schemes do not require A and B to share keys.

In the following we give the details of a concrete proxy encryption construction from [Dong 2008a]. The scheme consists of five algorithms: an initialization

algorithm PE–Init, a key generation algorithm PE–KeyGen, a user side encryption algorithm PE–Enc–U, a proxy side re-encryption algorithm PE–Enc–S, a proxy pre-decryption algorithm PE–Dec–S, and a user side decryption algorithm PE–Dec–U.

The initialization algorithm PE–Init is run by a trusted Key Authority (KA) and generates the public and master secret keys as shown in Algorithm 1. The public parameters pk are distributed to the users and proxy server, while the master secret key mk is stored securely by the KA.

Algorithm 1 PE-Init

Input: A security parameter 1^k.
Output: The public parameters pk and the master secret key mk.

1: Generate two prime numbers p and q such that $q = (p-1)/2$ and $|q| = k$.
2: Generate a cyclic group $\mathbb{G}$ with generator g such that $\mathbb{G}$ is the unique order q subgroup of $\mathbb{Z}_p^*$.
3: Choose x uniformly at random from $\mathbb{Z}_q^*$ and compute $h = g^x$.
4: $pk \leftarrow (\mathbb{G}, g, q, h)$
5: $mk \leftarrow x$
6: **return** (pk, mk).

For each new user, the KA runs PE–KeyGen(x, i) as shown in Algorithm 2, where i is the identity of the user. The KA securely distributes the user side key x_{i1} to the user and (i, x_{i2}) to the proxy server.

Algorithm 2 PE-KeyGen: Key generation for a new user.

Input: The public key pk, the master secret key $mk = x$ and the user identity i.
Output: The client side key x_{i1} and the broker side key set (i, x_{i2}).

1: Choose a random x_{i1} from $\mathbb{Z}_p$.
2: $x_{i2} \leftarrow x - x_{i1}$
3: **return** x_{i1} and (i, x_{i2}).

Figure 3.1 provides an overview of how a message encrypted with the key of user i is transformed by the proxy server and decrypted with the key of user j.

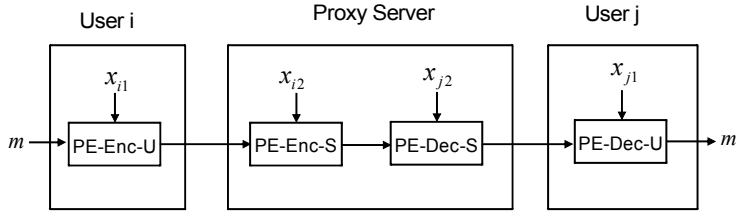


Figure 3.1: Proxy encryption, transformation and decryption.

In the following we give the details of these algorithms.

First, the user encrypts a message m using its unique key as shown in Algorithm 3.

Algorithm 3 PE-Enc-U: The user side proxy encryption.

Input: An element m, the public parameters $pk = (\mathbb{G}, g, q, h)$, and the user side key x_{i1}.
Output: The ciphertext $PE_i(k)$.

1: Choose r randomly from $\mathbb{Z}_q$.
2: $PE_i(m) \leftarrow (g^r, g^{rx_{i1}}m)$
3: **return** $PE_i(m)$.

The server re-encrypts the ciphertext computed by the user using the other side of the key.

Algorithm 4 PE-Enc-S: Server re-encryption.

Input: The ciphertext $PE_i(m)$, the public parameters $PK_{SE} = (\mathbb{G}, g, q, h)$, and the broker side key (i, x_{i2}) for user i.
Output: The ciphertext $PE(m)$.

1: Compute $(g^r)^{x_{i2}} \cdot g^{rx_{i1}}m = g^{r(x_{i1}+x_{i2})}m = g^{rx}m$.
2: $PE(m) \leftarrow (g^r, g^{rx}m)$
3: **return** $PE(m)$.

The server can pre-decrypt any message encrypted by any user such that it can be decrypted only by user i as shown in Algorithm 5.

Algorithm 5 PE-Dec-S: Server pre-decryption.

Input: The encrypted element $PE(m) = (g^r, g^{rx}m)$ and the server side key set (i, x_{x2}) corresponding to user i.
Output: The server pre-decrypted element $d_i(m)$ that only can be decrypted by user i.

1: Compute $g^{rx}m \cdot (g^r)^{-x_{i2}} = g^{r(x-x_{i2})}m = g^{rx_{i1}}m$.
2: $d_i(m) \leftarrow (g^r, g^{rx_{i1}}m)$
3: **return** $d_i(m)$.

User i finalises the decryption using its unique key.

Algorithm 6 PE-Dec-U: User decryption.

Input: The pre-encrypted element $d_i(m) = (g^r, g^{rx_{i1}}m)$ and the user key x_{i1}.
Output: The plaintext m.

1: $m \leftarrow g^{rx_{i1}}m \cdot (g^r)^{-x_{i1}}$
2: **return** m.

3.4.2 Multi-user encrypted search

To preserve the decoupling of publishers and subscribers, we require an encrypted search technique that allows multiple users to encrypt data and perform queries

on the data without sharing keys. The main idea is that users are able to encrypt and decrypt messages, and make encrypted queries, while the server performs computations on the encrypted data, without learning the content of the messages or the queries. Many single-user encrypted search techniques have been proposed [Song 2000b, Golle 2004a, Katz 2008] which can support keyword search or conjunction of keywords. The disadvantage is that in order to support multiple users, the users would need to share keys.

More recently, multi-user encrypted search methods were introduced [Bao 2008, Dong 2008a] that allow single keyword searches. With these schemes, each user has its own pair of secret keys which can be revoked when the user leaves the system. The solution of [Dong 2008a] is more efficient and has a practical implementation. In our scheme, we will extend this solution to support more complex queries such as conjunctions and disjunctions of equalities and inequalities. The searchable data encryption (SDE) scheme from [Dong 2008a] allows an untrusted server to perform keyword searches on data without revealing the data or the keywords to the server. To allow encrypted searches, users define a set of keywords for each document and encrypt them using a PE-based keyword encryption scheme. To search for documents containing a particular keyword, a user computes a trapdoor for the keyword. The trapdoor is used by the server to test the encrypted keywords of the stored document. In this way, the server can identify a match without learning the keyword.

We give in the following the details of the main algorithms of SDE: SDE–Init, SDE–KeyGen, KE–Enc–U, KE–Enc–S, Trap–U, Trap–S and SDE–Match. Figure 3.2 gives an overview of how these algorithms are used to match a keyword kw_i computed by user i against a trapdoor for word kw_j computed by user j.

The initialization method SDE–Init is run by the KA once at setup.

Algorithm 7 SDE-Init

Input: A security parameter 1^k.
Output: The public parameters PK_{SE} and the master secret key MK_{SE}.

1: Run PE–Init to generate the public parameters $(\mathbb{G}, g, q, h)$ and master secret key x.
2: Choose a collision-resistant hash function H.
3: Choose a pseudorandom function f.
4: Choose a random key s for f.
5: $PK_{SE} \leftarrow (\mathbb{G}, g, q, h, H, f)$
6: $MK_{SE} \leftarrow (x, s)$
7: **return** (PK_{SE}, MK_{SE}).

For each new user i, the KA computes a unique key set K_{ui} and a corresponding server-side key set K_{si} as shown in Algorithm 8.

Algorithm 8 SDE-KeyGen: Key generation for each new user.

Input: The master secret key MK_{SE} and the user identity i.
Output: The client side key set K_{ui} and the broker side key set K_{si}.

1: Choose a random x_{i1} from $\mathbb{Z}_p$.
2: $x_{i2} \leftarrow x - x_{i1}$
3: $K_{ui} \leftarrow (s, x_{i1})$
4: $K_{si} \leftarrow (i, x_{i2})$
5: **return** (K_{ui}, K_{si}).

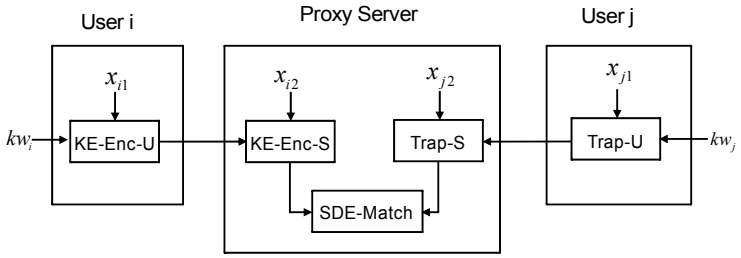


Figure 3.2: Encrypted keyword match by an untrusted server.

After encrypting the content of a document using PE–Enc–U, the user encrypts the keywords of the document using KE–Enc–U.

Algorithm 9 KE-Enc-U: The user side keyword encryption

Input: Keyword w, the user side key set $K_{ui} = (s, x_{i1})$ of user i, and the public parameters $PK_{SE} = (\mathbb{G}, g, q, h, H, f)$.
Output: The client encrypted element $KE_i(w)$.

1: Choose a random r_w from $\mathbb{Z}_q^*$.
2: $\sigma_w \leftarrow f_s(w)$
3: $\hat{c}_1 \leftarrow g^{r_w + \sigma_w}$
4: $\hat{c}_2 \leftarrow \hat{c}_1^{x_{i1}}$
5: $\hat{c}_3 \leftarrow H(h^{r_w})$
6: $KE_i(w) \leftarrow (\hat{c}_1, \hat{c}_2, \hat{c}_3)$
7: **return** $KE_i(w)$.

The server re-encrypts the document using PE–Enc–S and the keywords using KE–Enc–S as shown in Algorithm 10.

Algorithm 10 KE-Enc-S: Server side keyword re-encryption.

Input: The client encrypted keyword $KE_i(w) = (\hat{c}_1, \hat{c}_2, \hat{c}_3)$ and the server side key set $K_{si} = (i, x_{i2})$ corresponding to user i.
Output: The server re-encrypted keyword $KE(w)$.

1: $c_1 \leftarrow (\hat{c}_1)^{x_{i2}} \cdot \hat{c}_2 = \hat{c}_1^{x_{i1}+x_{i2}} = (g^{r_w+\sigma_w})^x = h^{r_w+\sigma_w}$
2: $c_2 \leftarrow \hat{c}_3$
3: $KE(w) \leftarrow (c_1, c_2)$
4: **return** $KE(w)$.

To perform a search on documents encrypted by any user in the system, a user j encrypts a keyword as a trapdoor using Trap–U.

Algorithm 11 Trap-U: The user side trapdoor encryption.

Input: A keyword w, the public parameters $PK_{SE} = (\mathbb{G}, g, q, h, H, f)$, and the user side key $K_{uj} = (s, x_{j1})$.
Output: The trapdoor for the keyword $TD_j(w)$.

1: Choose a random r_w from $\mathbb{Z}_q$.
2: $\sigma_w \leftarrow f_s(w)$
3: $td_1 \leftarrow g^{-r_w} g^{\sigma_w}$
4: $td_2 \leftarrow h^{r_w} g^{-x_{j1} r_w} g^{x_{j1}\sigma_w} = g^{(x-x_{j1})r_w} g^{x_{j1}\sigma_w} = g^{x_{i2} r_a} g^{x_{i1}\sigma_w}$
5: **return** $TD_j(w) = (td_1, td_2)$.

The server re-encrypts the trapdoor of the user using Trap–S.

Algorithm 12 Trap-S: The server side trapdoor re-encryption.

Input: The trapdoor $TD_j(w) = (td_1, td_2)$ and the server side key $K_{sj} = (j, x_{i2})$ for user j.
Output: The re-encrypted trapdoor $TD(w)$.

1: Compute $td_2^{x_{js}} \cdot td_2 = (g^{-r_w} g^{\sigma_w})^{x_{j2}} \cdot g^{x_{j2} r_w} g^{x_{j1}\sigma_w} = g^{(x_{j1}+x_{j2})\sigma_w} = g^{x\sigma_w} = h^{\sigma_w}$.
2: $TD(w) \leftarrow h^{\sigma_w}$
3: **return** $TD(w)$.

The server can now perform the match between a re-encrypted keyword and trapdoor.

Algorithm 13 Match: Single keyword match.

Input: A server re-encrypted keyword $KE(a) = (c_1, c_2)$, a server re-encrypted trapdoor $TD(b)$ and public parameters PK_{SE}.
Output: *true* or *false*.

1: **if** $c_2 \stackrel{?}{=} H(c_1 \cdot TD(b)^{-1})$ **then**
2: **return** *true*
3: **else**
4: **return** *false*
5: **end if**

3.5 Proposed solution

In the following, we discuss in details our scheme for providing confidentiality in pub/sub systems.

3.5.1 Assumptions

We assume that an event E consists of: (i) the message M that represents the content of the event and (ii) a set of attributes a_i that characterise M and are used for event filtering by the brokers. An attribute can be a string (e.g., "financial news"), or have the form *attr_name=attr_value*, where *attr_value* can be either a string or a number. Filters represent conjunctions and disjunctions of attributes, equalities of the form *attr_name=attr_value* and numerical inequalities such as *attr_name op attr_value*, where *op* can be one of $\leq, <, \geq$, and $>$. Current content-based pub/sub systems use a similar model. For example, in JMS [Hapner 2002] events consist of a message body, and a header defining properties in the form *prop_name=value*. Subscribers can define constraints on the values of the properties. In Siena [Carzaniga 2001], an event consists of $(attribute, value)$ pairs. To encrypt such an event with our method, we assume the content M consists of the $(attribute, value)$ pairs, so M simply contains all attributes a_i.

To provide confidentiality of the event, both M and the attributes a_i need to encrypted. The content M needs to be encrypted by the publisher and decrypted by all the authorised subscribers, without requiring publishers and subscribers to share keys (*P1*). Brokers forwarding the event from publishers to subscribers should not be able to access M. The attributes a_i describe the event and are used by brokers to match the event against registered filters. Hence, attributes and filters need to be encrypted by publishers and subscribers in such a way that brokers are able to evaluate encrypted filters using the encrypted attributes and without learning what they are. In the following we give the details and discuss a basic solution for confidentiality of events and filters.

3.5.2 Solution overview

In the following we propose a solution that addresses threat model 1: honest-but-curious and provides properties *P1* (Scalable key management), *P2* (Publication confidentiality), *P3* (Subscription confidentiality), and *P4* (Complex encrypted matching).

Proxy encryption (PE) has the property that it allows full decoupling of the communicating parties. In PE, each user (publisher or subscriber in our case) has a private key that allows it to encrypt and decrypt messages. To publish a message, a publisher needs to encrypt it just once with its private key. Before the message is delivered to a particular subscriber, the broker performs a re-encryption of the message so that the message can be decrypted by the subscriber with its private key. Hence, if an event matches n filters, the publisher needs to encrypt it just once, and the brokers will need to perform n re-encryptions, one for each

subscriber. The advantage of this method is that the publishers and subscribers do not need to share keys. To encrypt the message M, we use the El Gamal-based proxy encryption scheme from [Dong 2008a] and described in algorithms PE–Enc–U and PE–Enc–S.

Providing encrypted filtering can be seen as a problem of encrypted search. The broker needs to verify if a list of attributes attached to the event match a complex encrypted filter. Current solutions for pub/sub systems [Chen 2010, Nabeel 2009, Choi 2010] which achieve filters more complex than keyword search, require publishers and subscribers to share secret keys. On the other hand, multi-user encrypted search techniques [Dong 2008a, Bao 2008] which do not require users to share keys, only provide keyword search.

In order to support complex filters without requiring publishers and subscribers to share keys, we represent filters as tree access structures [Bethencourt 2007] capable of expressing conjunctions and disjunctions of equalities and inequalities, and encrypt the leaf nodes of the tree with SDE. In the following, we show the steps that are performed in our scheme.

3.5.3 Initialization

The initialization algorithm SDE–Init is run by a trusted **Key Authority** (KA) once at setup and defines the public and private security parameters for SDE as shown in Algorithm 7. The KA publishes the public parameters PK_{SE}, and keeps securely the master secret key MK_{SE}.

For every new user (publisher or subscriber), the KA runs SDE–KeyGen(MK_{SE}, i) as shown in Algorithm 8, where i is the identity of the user. The algorithm generates the user side secret key K_{ui} and the corresponding server side key K_{si}. The KA securely distributes K_{ui} to the user and K_{si} to the local broker of the user.

3.5.4 Event encryption

Figure 3.3 shows the event encryption steps, run by a publisher p. The publisher first defines a set of attributes $\gamma = \{a_1, ..., a_n\}$ and the content M of the event to be encrypted. It then encrypts the event following these steps:

1. The publisher p encrypts the message content M:
   - Generate a random AES encryption key k.
   - Encrypt M under k using AES as $c_{AES}(M) \leftarrow$ AES–Enc(M, k).
   - Encrypt k using proxy encryption as
   $$PE_p(k) \leftarrow \text{PE–Enc–U}(k, PK_{SE}, K_{up})$$
   - $c_p(M) \leftarrow (PE_p(k), c_{AES}(M))$.

We note that the actual message M is encrypted with AES, and the key is encrypted using proxy encryption. Encrypting the whole message with PE would be too inefficient.

2. The publisher encrypts the attributes. For every attribute $a \in \gamma$, the publisher computes a trapdoor $TD_p(a) \leftarrow \mathsf{Trap\text{–}U}(K_{up}, a)$. Trapdoors do not allow recovering the attribute through decryption, but instead they are only used to check keyword equality without gaining any information about the matched keywords.

3. The publisher sends the encrypted message together with the attribute trapdoors to the broker: $E_p = (c_p(M), \{TD_p(a)\}_{a\in\gamma})$.

4. The broker locates the key $K_{sp} = (p, x_{p2})$ corresponding to the publisher and re-encrypts the message:

   - $PE(k) \leftarrow \mathsf{PE\text{–}Enc\text{–}S}(PE_p(k), K_{sp})$
   - $c(M) \leftarrow (PE(k), c_{AES}(M))$

5. The broker re-encrypts the trapdoors. For each trapdoor $TD_p(a)$, the broker computes $TD(a) \leftarrow \mathsf{Trap\text{–}S}(TD_p(a), K_{sp})$.

   The re-encrypted event becomes: $E = (c(M), \{TD(a)\}_{a\in\gamma}))$.

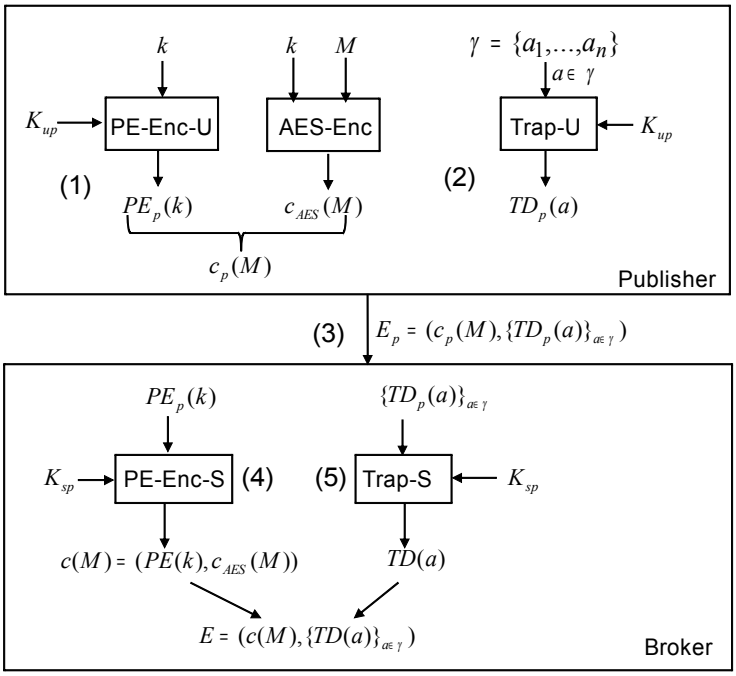


Figure 3.3: Event encryption with Proxy Encryption.

3.5.5 Filter encryption

The filter encryption algorithm makes use of the keyword encryption scheme of SDE. KE–Enc–U (see Algorithm 9) encrypts the keyword on the user side, and KE–Enc–S (see Algorithm 10) re-encrypts the keyword on the server side. Later, during event matching, the broker will check if keywords encrypted with these algorithms match the attributes previously encrypted as trapdoors.

Figure 3.4 shows the main steps for generating and encrypting the filter.

1. The subscriber defines the filter as an *access tree* F where numeric inequalities are expanded using the bit representation.

2. To provide confidentiality of the filter, the subscriber encrypts each leaf node x as $KE_s(x) \leftarrow \mathsf{KE–Enc–U}(K_{us}, PK_{SE})$.

3. The subscriber sends the encrypted filter F_s to the broker. The broker locates the key K_{ss} corresponding to the subscriber and re-encrypts the leaf-node attributes of F_s. For each leaf node $KE_s(a)$, the broker computes $KE(a) \leftarrow \mathsf{KE–Enc–S}(K_{ss}, KE_s(a))$. We call the re-encrypted filter F^*.

The above operations provide confidentiality of the filter, thus achieving property *P2*. At the same time, the filter is able to express conjunctions and disjunctions of equalities and inequalities, thus achieving property *P3*.

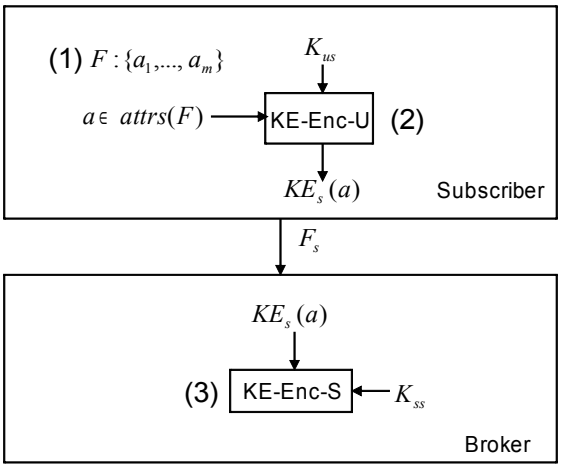


Figure 3.4: Filter generation and encryption.

3.5.6 Encrypted matching

The encrypted matching algorithm relies on the keyword matching algorithm SDE–Match of SDE which is only able to match single keywords as shown in Algorithm 13.

Algorithm 14 TreeEval: Access Tree Evaluation

Input: A node x of the re-encrypted tree F^*, and the trapdoors $\{TD(a)\}_{a\in\gamma}$ of the re-encrypted event E.
Output: *true* or $false$.

```
1: if x is a leaf node then
2:   for all trapdoors TD(a) do
3:     if SDE–Match(TD(a), attr(x)) then
4:       return true
5:     end if
6:   end for
7: else
8:   l = 0
9:   while l < threshold(x) do
10:    for all children c of x do
11:      if TreeEval(c, {TD(a)}_{a∈γ}) then
12:        l++
13:      end if
14:    end for
15:  end while
16:  if l = threshold(x) then
17:    return true
18:  end if
19: end if
20: return false
```

When a new event E is published, for every filter F^* the broker runs a recursive algorithm TreeEval (see Algorithm 14) on the tree F^* starting with the root node to check if it is satisfied by the attributes of the event. A non-leaf node x is satisfied if the number or satisfied children is equal or greater than k_x, the threshold value of the node. A leaf node containing an attribute b encrypted as $KE(b) = (c_{b1}, c_{b2})$ is satisfied if b is among the attributes of the event, encrypted as $\{TD(a)\}_{a\in\gamma}$. To check if a leaf node attribute b matches an event attribute a, the broker needs to verify if SDE–Match$(KE(b), TD(a))$ returns *true*.

3.5.7 Event decryption

If the filter F^* is satisfied, before forwarding the event E to the subscriber s that registered the filter, the broker pre-decrypts the event such that only s can decrypt it using its secret key.

The event decryption makes use of the following algorithms from SDE: PE–Dec–S (see Algorihtm 5) run on the broker side to pre-decrypt an element encrypted with PE such that it can be decrypted only by the key of a user i, and PE–Dec–S (see Algorithm 6) run by user i to retrieve the plaintext. The server on its own cannot decrypt the message.

The event decryption algorithm proceeds as follows and as shown in Figure 3.5:

1. Before sending the event to the subscriber s, the broker pre-decrypts

$PE(k)$ such that it can only be decrypted by subscriber s: $d_s(k) \leftarrow \mathsf{PE\text{–}Dec\text{–}S}(PE(k), K_{ss})$. It then sets $c_s(M) \leftarrow (PE_s(k), c_{AES}(M))$.

2. The broker forwards the ciphertext $c_s(M)$ to subscriber s.

3. After receiving the ciphertext, the subscriber decrypts it:

   - Decrypt the key: $k \leftarrow \mathsf{PE\text{–}Dec\text{–}U}(d_s(k), K_{us})$
   - Decrypt the message: $M \leftarrow \mathsf{AES\text{–}Dec}(c_{AES}(M), k)$

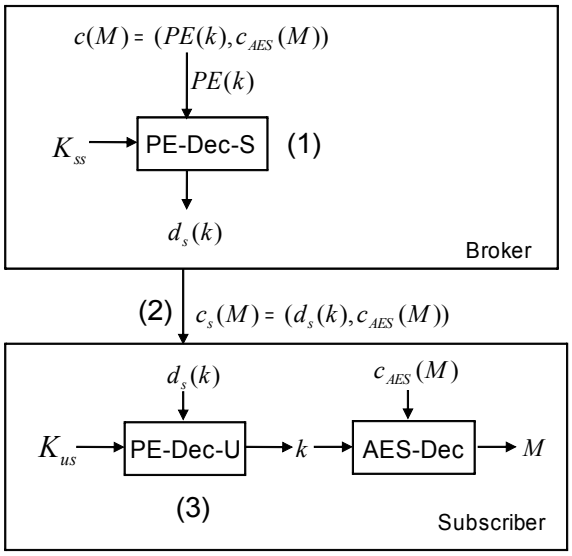


Figure 3.5: Event decryption.

3.6 Security analysis

This section evaluates the security of the scheme. We start by introducing some concepts needed to understand the analysis and then discuss the security of the filter encryption and event encryption.

3.6.1 Preliminaries

We start by defining some concepts that are useful to understand the proof. In general a scheme is considered secure if no adversary can break the scheme with probability significantly greater than random guessing. The adversary's advantage in breaking the scheme should be a negligible function of the security parameter.

Definition 5 (Negligible Function)**.** *A function f is negligible if for each polynomial $p()$ there exists N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.*

We consider a realistic adversary that is computationally bounded and show that our scheme is secure against such an adversary. We model the adversary as a randomized algorithm that runs in polynomial amount of time and show that the success probability of any such adversary is negligible. An algorithm that is randomized and runs in polynomial amount of time is called a probabilistic polynomial time (PPT) algorithm.

Our scheme relies on the existence of a pseudorandom function f. Intuitively, the output a pseudorandom function cannot be distinguished by a realistic adversary from that of a truly random function. Formally, a pseudorandom function is defined as:

Definition 6 (Pseudorandom Function)**.** *A function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is pseudorandom if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:*

$$|Pr[\mathcal{A}^{f_k(\cdot)} = 1] - Pr[\mathcal{A}^{F(\cdot)} = 1]| < negl(n)$$

where $k \rightarrow \{0,1\}^n$ is chosen uniformly randomly and F is a function chosen uniformly randomly from the set of functions mapping n-bit strings to n-bit strings.

Our proof relies on the assumption that the Decisional Diffie-Hellman (DDH) is hard in a group $\mathbb{G}$, i.e., it is hard for an adversary to distinguish between group elements $g^{\alpha\beta}$ and g^{γ} given g^{α} and g^{β}.

Definition 7 (DDH Assumption)**.** *The Decisional Diffie-Hellman (DDH) problem is hard regarding a group $\mathbb{G}$ if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that $|Pr[\mathcal{A}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1] - Pr[\mathcal{A}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1]| < negl(k)$ where $\mathbb{G}$ is a cyclic group of order q ($|q| = k$) and g is a generator of $\mathbb{G}$, and $\alpha, \beta, \gamma \in \mathbb{Z}_q$ are uniformly randomly chosen.*

The schemes we are using in our solution (i.e., PE, KE) have been proven to be indistinguishable under chosen plaintext attack (*IND-CPA*) and we will prove that our scheme is also IND-CPA secure. A cryptosystem is considered IND-CPA secure if no PPT adversary, given an encryption of a message randomly chosen from two plaintext messages chosen by the adversary, can identify which message was encrypted with non-negligible probability.

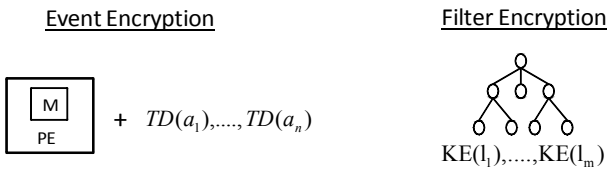


Figure 3.6: Event and filter encryption.

3.6.2 Scheme overview

Figure 3.6 shows the different encryption schemes that are used to provide confidentiality of events and filters.

To ensure confidentiality of events, our scheme encrypts the message content M with proxy encryption (PE) and filter attributes using the keyword encryption algorithm of SDE. [Dong 2011] proves that the concrete PE construction and the single keyword encryption scheme KE built upon El Gamal are IND-CPA under the assumption the DDH problem is hard relative to the group on which El Gamal is defined. In the following we show that the filter encryption and event encryption schemes are also IND-CPA secure. We show that breaking our scheme reduces to breaking the above cryptosystems (i.e., PE, KE) that have been proven to be IND-CPA secure.

3.6.3 Security of filter encryption

Our filter encryption scheme FE uses the single keyword encryption scheme KE of SDE to encrypt the leaf nodes of the tree. We recap bellow the operations needed to encrypt a filter:

- SDE–Init(k) The KA generates the public key $PK_{SE} = (\mathbb{G}, g, q, h, H, f)$, and the master secret key $MK_{SE} = (x, s)$.
- SDE–KeyGen(MK, i) The KA gives to user i the key $K_{ui} = (x_{i1}, s)$ and to the broker $K_{si}(i, x_{i2})$.
- FE–U(F) On every leaf node l of the filter, the user calls KE–Enc–U(l, K_{ui}) and computes $KE_i(l)$.
- FE–B(F) The broker re-encrypts every leaf node by calling KE–Enc–S($KE_i(l), K_{si}$) to compute $KE(l)$.

In the following we prove that the filter encryption scheme is secure in the sense that the broker learns nothing about the encrypted leaf nodes in a chosen plaintext attack.

[Dong 2011] showed that the single keyword encryption scheme KE is IND-CPA secure against the broker and proved that the following holds:

Theorem 1. *If the DDH problem is hard relative to $\mathbb{G}$, then the keyword encryption* KE *scheme is IND-CPA secure against the broker. That is, for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that:*

$$Succ^{\mathcal{A}}_{KE,B}(k) = Pr\left[b' = b \;\middle|\; \begin{array}{c} (PK_{SE}, MK_{SE}) \leftarrow \mathsf{SDE\text{–}Init}(1^k) \\ (K_u, K_s) \leftarrow \mathsf{SDE\text{–}KeyGen}(MK_{SE}, U) \\ kw_0, kw_1 \leftarrow \mathcal{A}^{\mathsf{KE\text{–}Enc\text{–}U}(K_u,\cdot)}(K_s) \\ b \xleftarrow{R} \{0,1\} \\ KE_i(kw_b) = \mathsf{KE\text{–}Enc\text{–}U}(K_u, kw_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{KE\text{–}Enc\text{–}U}(K_u,\cdot)}(K_s, KE_i(kw_b)) \end{array} \right] \tag{3.1}$$

$$< \tfrac{1}{2} + negl(k)$$

where U is a set of user IDs, K_u are the user side key sets, K_s are the broker side keys.

Using this result, we now show that our filter encryption scheme FE using KE to encrypt the leaf nodes of the tree is also IND-CPA against the broker.

Theorem 2. *If the single keyword encryption* KE *scheme is IND-CPA secure against the broker, then the filter encryption scheme* FE *is also IND-CPA. That is, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that $Succ^{\mathcal{A}}_{FE,B}(k) < \frac{1}{2} + negl(k)$.*

Proof. To prove FE is secure, we define the following game in which the adversary $\mathcal{A}$ challenges the game with two filters F_0 and F_1 having the same structure and the same number of leaf nodes t. We construct the following vector containing the encryption of leaf nodes from both filters: $\vec{C}^{(i)} = C(l_0^1), ..., C(l_0^i), C(l_1^{i+1}), ..., C(l_1^t)$. The success probability of the adversary in distinguishing the encryption of the two filters is defined as:

$$Succ_{\mathcal{A}}(k) = \frac{1}{2}Pr[A(\vec{C}^0) = 0] + \frac{1}{2}Pr[A(\vec{C}^t) = 1] \tag{3.2}$$

In the following we show that breaking the FE scheme reduces to breaking the KE game. In the KE game from [Dong 2011], the adversary challenges the game with two keywords kw_0 and kw_1 and tries to distinguish between their encryptions. Let us consider a PPT adversary $\mathcal{A}'$ who attempts to challenge the single keyword encryption scheme KE using the corresponding FE adversary $\mathcal{A}$ as a sub-routine The game is the following:

- $\mathcal{A}'$ is given the parameters $(\mathbb{G}, q, g, h, H, f)$ as input and for each user i is given $K_{si} = (i, x_{i2})$.
- $\mathcal{A}'$ passes these parameters to $\mathcal{A}$.
- $\mathcal{A}$ generates two filters F_0 and F_1 having the same non-leaf nodes and the same number of leaf-nodes t and gives them to $\mathcal{A}'$.
- $\mathcal{A}'$ chooses $i \xleftarrow{r} [1, t]$. It then uses l_0^i, l_1^i to challenge the single keyword encryption KE game. The adversary gets back c_b^i as the result, where c_b^i is the encryption of either l_0^i or l_1^i. $\mathcal{A}'$ uses this result to construct a hybrid vector $(c_0^1, ..., c_0^{i-1}, c_b^i, c_1^{i+1}, ..., c_1^t)$ and sends it to $\mathcal{A}$.
- $\mathcal{A}'$ outputs b', the bit output by $\mathcal{A}$.

$\mathcal{A}$ is required to distinguish $\vec{C}^{(i)}$ and $\vec{C}^{(i-1)}$ and the probability of $\mathcal{A}$'s success in distinguishing correctly is:

$$Succ^i_{\mathcal{A}}(k) = \frac{1}{2}Pr[A(\vec{C}^{(i)}) = 0] + \frac{1}{2}Pr[A(\vec{C}^{(i-1)}) = 1] \tag{3.3}$$

Because i is randomly chosen, it holds that:

$$\begin{aligned} Succ_{\mathcal{A}'}(k) &= \sum_{i=1}^{t} Succ_{\mathcal{A}}^{i}(t) \cdot \tfrac{1}{t} \\ &= \tfrac{1}{2t} Pr[A(\vec{C}^0) = 0] + \sum_{i=1}^{t-1} (Pr[A(\vec{C}^i) = 0] \\ &\quad + Pr[A(\vec{C}^i) = 1]) + \tfrac{1}{2} Pr[A(\vec{C}^t) = 1] \\ &= \tfrac{1}{t}(\tfrac{1}{2} Pr[A(\vec{C}^0) = 0] + \tfrac{1}{2} Pr[A(\vec{C}^t) = 1]) + \tfrac{t-1}{2t} \\ &= \tfrac{1}{t} Succ_{\mathcal{A}}(k) + \tfrac{t-1}{2t} \end{aligned} \tag{3.4}$$

Because the success probability of $\mathcal{A}'$ to break the single keyword encryption scheme is $Succ_{\mathcal{A}'}(k) < \frac{1}{2} + negl(k)$, it follows that $Succ_{\mathcal{A}}(k) < \frac{1}{2} + negl(k)$.

3.6.4 Security of event encryption

The main task of a broker in a pub/sub system is to match incoming events against stored filters. Though both events and filters are encrypted, the protocol leaks to the broker the outcome of the matching operation, which events match which filters, and the pattern of the sequence of events and filters arriving at the broker. In the following we show that our basic scheme does not leak anything beyond this.

We adapt the definition of non-adaptive indistinguishability security introduced for encrypted databases by [Curtmola 2006a] and adapted by [Dong 2011] in a multi-user setting. As opposed to encrypted databases where the server stores the data and executes queries as they come, a broker in a pub/sub system stores the filters (i.e., the queries) and evaluates all of them on a given event (i.e., a data item). For our proof we use the same idea as in [Curtmola 2006a, Dong 2011] and show that given two non-adaptively generated histories with the same length and outcome, no PPT adversary can distinguish one from another based on what it can observe from the interaction. Non-adaptive history means that the adversary cannot choose sequences of events based on previous events and matching outcomes.

In our protocol, a *history* is represented by the interactions between a broker B and all publishers and subscribers connected to the broker, i.e., a history is a sequence of events and filters arriving at the broker. We will refer in the following to these as *requests*.

Definition 8 (History)**.** *A history $\mathcal{H}_i$ is an interaction between a broker and all publishers and subscribers connected to it, over i requests (event publication or filter subscription), on a set of filters $\mathcal{F}$ stored by the broker. $\mathcal{H}_i = (\mathcal{F}, r_1^{u_1}, ..., r_i^{u_i})$, where u_i are the identifiers of the users (publishers or subscribers) making the requests (publications or subscriptions).*

We formalize the information leaked to a broker as a *trace*. We define two kinds of traces: the trace of a single request and the trace of a history. The trace of an event publication request leaks to the broker (i) the list of filters that match a given event, and (ii) which event attribute matches which leaf node in the filters. The outcome of the event matching operation can be represented as a set of filter IDs that match the event, a list of event attribute trapdoors, and for

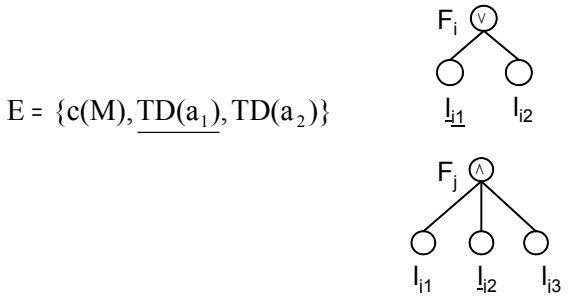


Figure 3.7: Event matching against two filters. $TD(a_1)$ matches l_{i1} and l_{j2}.

each trapdoor, a list of indexes pointing to filter leaf nodes matching the trapdoor. The encryption of filter attributes is non-deterministic, meaning that equal attributes encrypt to different ciphertexts due to the fact that a unique random number is selected every time an attribute is encrypted. Because of this, all leaf nodes in filters are different and the broker cannot tell if two leaf node ciphertexts encrypt the same attribute or not. However, when the broker matches an event against the filters, it can identify which filter attributes match the same trapdoor of an event. Formally, a trace of a publication request can be written as: $pt(E) = \{id(F_1), ..., id(F_m), |c(M)|, (TD(a_1), index_k, ..., index_j), ...,$ $(TD(a_n), index_l, ..., index_v)\}$, where $|c(M)|$ is the size of the ciphertext.

Figure 3.7 shows an event containing two attributes with their trapdoors $TD(a_1)$ and $TD(a_2)$ and two filters registered on the broker F_i and F_j. Let us assume that $TD(a_1)$ matches leaf nodes l_{i1} of F_i and l_{j2} of F_j, and that $TD(a_2)$ does not match any leaf nodes. In this case only filter F_i is satisfied. The trace of the event is $pt(E) = \{F_i, |c(M)|, (TD(a_1), l_{i1}, l_{j2}), TD(a_2)\}$.

A subscription request leaks to the server the tree structure of the filter, i.e., the internal nodes of the filter containing threshold gates such as *AND* and *OR* conditions. We refer to the structure of F without any leaf node information as $struct(F)$.

Definition 9 (Trace of a Request)**.** *We define the trace of a request r as:*

$$Tr(r) = \begin{cases} u, pt(E) & \text{if publication} \\ u, id(F), struct(F) & \text{if subscription} \end{cases} \tag{3.5}$$

where u is the id of the user.

We define the event filtering pattern $\mathcal{P}$ over a history $\mathcal{H}_i$ to be a binary matrix with columns corresponding to events and rows corresponding to trapdoors. $\mathcal{P}[j,k] = 1$ if trapdoor j was present in event k and $\mathcal{P}[j,k] = 0$ otherwise.

The trace of a history includes the encrypted filter set $\mathcal{F}$ stored by the broker and which can change as filters are registered and deregistered, the trace of each request (i.e., filter subscription or event publication), and the event publication pattern $\mathcal{P}_i$.

Definition 10 (Trace of a History)**.** *We define the trace of a history* $\mathcal{H}_i = (\mathcal{F}, r_1^{u_1}, ..., r_i^{u_i})$ *as:*

$$Tr(\mathcal{H}_i) = (\mathcal{F}, Tr(r_1^{u_1}), ..., Tr(r_i^{u_i}), \mathcal{P}_i) \tag{3.6}$$

During an interaction, the adversary cannot see directly the plaintext of the event, instead it sees the ciphertext. The view of a request is defined as:

Definition 11 (View of a Request)**.** *We define the view of a request* $t_1^{u_1}$ *under a key set* K_{ui} *as:*

$$V_{K_{ui}}(t^{u_i}) = \begin{cases} c(M), pt(t^{u_i}) \text{ if publication} \\ struct(F) \text{ and leaf nodes } \{KE_{u_i}(a_1), ..., KE_{u_i}(a_n)\} \\ \text{ if subscription} \end{cases} \tag{3.7}$$

Definition 12 (View of a History)**.** *We define the view of a history* $\mathcal{H}_i = (\mathcal{F}, r_1^{u_1}, ..., r_i^{u_i})$ *as:*

$$V_{K_u}(H_i) = (\mathcal{F}, V_{K_{u1}}(r^{u_1}), ..., V_{K_{ui}}(r^{u_i})) \tag{3.8}$$

The security definition is based on the idea that the scheme is secure if nothing is leaked to the adversary beyond what the adversary can learn from traces.

We define the following game in which an adversary $\mathcal{A}$ generates two histories $\mathcal{H}_{i0}$ and $\mathcal{H}_{i1}$ with the same trace over i requests. Then the adversary is challenged to distinguish the views of the two histories. If the adversary succeeds with negligible probability, the scheme is secure.

Definition 13 (Non-adaptive indistinguishability against a curious broker)**.** *The event encryption scheme is secure in the sense of non-adaptive indistinguishability against a curious broker if for all* $i \in \mathbb{N}$ *and for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function negl such that:*

$$Pr\left[b' = b \;\middle|\; \begin{array}{l} (PK_{SE}, MK_{SE}) \leftarrow Init(1^k) \\ (K_u, K_b) \leftarrow KeyGen(MK_{SE}, U) \\ \mathcal{H}_{io}, \mathcal{H}_{i1} \leftarrow \mathcal{A}(K_b) \\ b \xleftarrow{R} \{0,1\} \\ b' \leftarrow \mathcal{A}(K_b, V_{K_u}(\mathcal{H}_{ib})) \end{array} \right] < \frac{1}{2} + negl(k) \tag{3.9}$$

where U *is a set of user IDs,* K_u *are the user side key sets,* K_s *are the broker side keys,* $\mathcal{H}_{i1}$ *and* $\mathcal{H}_{i0}$ *are two histories over* i *requests such that* $Tr(\mathcal{H}_{i0}) = Tr(\mathcal{H}_{i1})$*.*

Theorem 3. *If the Decisional Diffie-Hellman (DDH) problem in hard relative to* $\mathbb{G}$*, then the basic confidentiality scheme is a non-adaptive indistinguishable secure scheme. The success probability of a PPT adversary* $\mathcal{A}$ *in breaking the basic scheme is defined as:*

$$\begin{aligned} Succ^{\mathcal{A}}(k) = {} & \tfrac{1}{2} Pr[\mathcal{A}((PE(\vec{M}_0), FE(\vec{F}_0), TD(\vec{a}_0))) = 0] + \\ & \tfrac{1}{2} Pr[\mathcal{A}((PE(\vec{M}_1), FE(\vec{F}_1), TD(\vec{a}_1))) = 1] \\ & < \tfrac{1}{2} + negl(k) \end{aligned} \tag{3.10}$$

Proof. We consider an adversary $\mathcal{A}'$ that challenges the PE IND-CPA game using $\mathcal{A}$ as a sub-routine. $\mathcal{A}'$ does the following:

- $\mathcal{A}'$ receives public parameters $\mathbb{G}, q, g$ and the server side (i, x_{i2}) keys. It then picks a random user ID u and queries the oracle with $m = 1$ to obtain the ciphertext $(g^r, g^{rx_{u1}})$. $\mathcal{A}'$ then computes $g^{rx_{u1}} g^{rx_{u2}} = g^{rx}$. It can then compute for every user i, $g^{rx_{i1}} = g^{rx} g^{-rx_{i2}}$ because it knows $x_{i2} = x - x_{x1}$.

- $\mathcal{A}'$ computes $g = g^r$ and $h = g^{rx} = g^x$ and chooses H, f, s. $\mathcal{A}'$ sends $(\mathbb{G}, q, g, h, H, f)$ to $\mathcal{A}$ together with the server side keys.

- To generate a view of a history $\mathcal{H}_i = (\mathcal{F}, q_1^{u_1}, ..., q_i^{u_i})$. $\mathcal{A}'$ performs the following steps:

  - For each filter $F \in \mathcal{F}$, do the following: for each attribute b in F, choose a random number z and compute $\hat{c}_1 = g^{z+\sigma}, \hat{c}_2 = (g^{rx_{i1}})^{z+\sigma} = (g^{z+\sigma})^{x_{i1}}$, $\hat{c}_3 = H(h^z)$ where $\sigma = f_s(b)$. Then compute (c_1, c_2) as $c_1 = \hat{c}_{i2}^x \cdot \hat{c}_2$, $c_2 = \hat{c}_3$. (c_1, c_2) is the ciphertext for attribute b.
  - For each publication request $r^{ui} = (M, \{a_i\})$, compute $c(M) = (g^{\tilde{r}}, g^{\tilde{r}x} M)$, where $\tilde{r}$ is random and for each attribute a compute $TD(a) = (td_1, td_2)$ such that $td_1 = g^{-r_a} g^{\sigma_a}$ and $td_2 = g^{x_{i2} r_a} g^{x_{i1} \sigma_a}$, where $\sigma_a = f_s(a)$. Then partially decrypt $c'(M) = (g^{\tilde{r}}, g^{\tilde{r}x_{ui1}} M)$.
  - For each subscription request $qr^{ui} = F$, for each attribute in F compute $(\hat{c}_1, \hat{c}_2, \hat{c}_3)$.

- $\mathcal{A}$ outputs $\mathcal{H}_{i0}, \mathcal{H}_{i1}$. $\mathcal{A}'$ encrypts every keyword and trapdoor in $\mathcal{H}_{i1}$ by itself and challenges the PE IND-CPA game with $\vec{M}_0$ and $\vec{M}_1$, the vectors of all event messages in the two histories. It gets the result $PE(\vec{M}_b)$ where $b \xleftarrow{R} \{0, 1\}$ and forms a view of a history $(PE(\vec{M}_b), \mathsf{FE}(\vec{F}_1), TD(\vec{a_1}))$, where $\mathsf{FE}(\vec{F}_1)$ is the encryption of the leaf nodes of the filter trees and $TD(\vec{a_1}))$ are the trapdoors computed for all attributes of each event in the trace. It sends the view to $\mathcal{A}$.

- $\mathcal{A}$ tries to determine which vector was encrypted and outputs $b' \in \{0, 1\}$.

- $\mathcal{A}'$ outputs b'.

[Dong 2011] proved that the El Gamal based PE scheme is IND-CPA. We proved that the filter encryption scheme FE is also IND-CPA secure. From these two results it follows that:

$$\begin{aligned} \tfrac{1}{2} + negl(k) &> Succ_{PE}^{\mathcal{A}'}(k) \\ &= \tfrac{1}{2} Pr[\mathcal{A}((PE(\vec{M}_0), FE(\vec{F}_1), TD(\vec{a}_1))) = 0] + \\ &\quad \tfrac{1}{2} Pr[\mathcal{A}((PE(\vec{M}_1), FE(\vec{F}_1), TD(\vec{a}_1))) = 1] \end{aligned} \tag{3.11}$$

Now let us consider another adversary $\mathcal{A}''$ who wants to distinguish the pseudorandom function f using $\mathcal{A}$ as a sub-routine. The adversary does the following:

- It generates $(\mathbb{G}, q, g, h, H)$ as public parameters, and sends them to $\mathcal{A}$ along with f. For each user i, it chooses randomly x_{i1}, x_{i2} such that $x_{i1} + x_{i2} = x$. It sends all (i, x_{i2}) to $\mathcal{A}$ and keeps all (i, x_{i1}, x_{i2}).
- $\mathcal{A}$ outputs $\mathcal{H}_{i0}, \mathcal{H}_{i1}$. $\mathcal{A}''$ encrypts all the event messages in $\mathcal{H}_{i0}$ as $PE(\vec{M}_0)$. It chooses $b \xleftarrow{R} \{0,1\}$ and asks the oracle to encrypt all keywords and trapdoors in $\mathcal{H}_{ib}$. It combines the results to form a view $(PE(\vec{M}_0), FE(\vec{F}_b), TD(\vec{a}_b))$ and returns it to $\mathcal{A}$.
- $\mathcal{A}$ outputs b'. $\mathcal{A}''$ outputs 1 if $b' = b$ and 0 otherwise.

There are two cases to consider: Case 1: the oracle in $\mathcal{A}''$s game is the pseudorandom function f, then:

$$\begin{aligned} Pr[\mathcal{A}''^{f_s(.)}(1^k) = 1] = \\ \tfrac{1}{2}Pr[\mathcal{A}((PE(\vec{M}_0), FE(\vec{F}_0), TD(\vec{a}_0))) = 0] + \\ \tfrac{1}{2}Pr[\mathcal{A}((PE(\vec{M}_0), FE(\vec{F}_1), TD(\vec{a}_1))) = 1] \end{aligned} \tag{3.12}$$

Case 2: the oracle in $\mathcal{A}''$s game is a random function F, then for each distinct attribute a, σ_a is completely random to $\mathcal{A}$. Moreover we know the traces are identical, so $KE(\vec{F}_b)$ and $TD(\vec{a}_b)$ are completely random to $\mathcal{A}$. In this case:

$$Pr[\mathcal{A}''^{f_s(.)}(1^k) = 1] = \frac{1}{2} \tag{3.13}$$

Because f is a pseudorandom function, by definition it holds that:

$$\begin{aligned} |Pr[\mathcal{A}''^{f_s(.)}(1^k) = 1] - Pr[\mathcal{A}'^{f_s(.)}(1^k) = 1]| < negl(k) \\ Pr[\mathcal{A}''^{f_s(.)}(1^k) = 1] < \tfrac{1}{2} + negl(k) \end{aligned} \tag{3.14}$$

Sum up $Succ_{PE}^{\mathcal{A}'}(k)$ and $Pr[\mathcal{A}''^{f_s(.)}(1^k) = 1]$:

$$\begin{aligned} 1 + negl(k) > \tfrac{1}{2}Pr[\mathcal{A}((PE(\vec{M}_0), FE(\vec{F}_0), TD(\vec{a}_0))) = 0] + \\ \tfrac{1}{2}Pr[\mathcal{A}((PE(\vec{M}_0), FE(\vec{F}_1), TD(\vec{a}_1))) = 1] + \\ \tfrac{1}{2}Pr[\mathcal{A}((PE(\vec{M}_0), FE(\vec{F}_1), TD(\vec{a}_1))) = 0] + \\ \tfrac{1}{2}Pr[\mathcal{A}((PE(\vec{M}_1), FE(\vec{F}_1), TD(\vec{a}_1))) = 1] \\ = \tfrac{1}{2}Pr[\mathcal{A}((PE(\vec{M}_0), FE(\vec{F}_0), TD(\vec{a}_0))) = 0] + \\ \tfrac{1}{2} + \\ \tfrac{1}{2}Pr[\mathcal{A}((PE(\vec{M}_1), FE(\vec{F}_1), TD(\vec{a}_1))) = 1] + \\ = \tfrac{1}{2} + Succ^{\mathcal{A}}(k) \end{aligned} \tag{3.15}$$

Therefore $Succ^{\mathcal{A}}(k) < \frac{1}{2} + negl(k)$.

3.7 Implementation and performance analysis

We implemented our encryption scheme in Java based on the SDE implementation from [Dong 2011]. We implemented the functions for event and filter encryption

at the client side, re-encryption at the broker side, encrypted filtering and event decryption described above. In the following, we measure and compare the performance of each function. We tested the implementation on an Intel Core2 Duo 2.8 GHz with 3.48 GB of RAM.

Figure 3.8 shows the performance of the basic encryption blocks from SDE. We used a 1024-bit prime as p and SHA-1 as the hash function. We note that SDE-Match, the most critical operation that needs to be performed event thousands of times during the matching process, is the most efficient one.

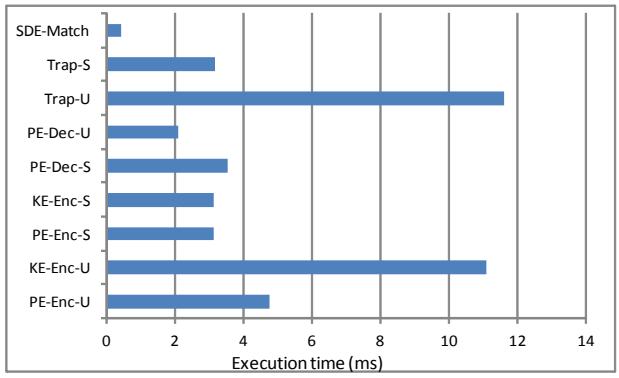


Figure 3.8: SDE basic operations performance time.

Using the basic blocks of SDE, we implemented the main functions of our scheme. Figure 3.9 compares event encryption and decryption operations both at publisher and broker side. In the figure, EV-Enc-U, the most costly operation, represents event encryption at publisher side, EV-Enc-B represesnts event re-encryption at the broker side, EV-Dec-B represents event pre-decryption at the broker side, and EV-Dec-U represents event decryption at the user side. We note that event encryption times grow linearly with the number of attributes, which was to expect because together with the content, each attribute is encrypted individually. On the other hand, event decryption times are constant because they only decrypt the event content, and not the attributes.

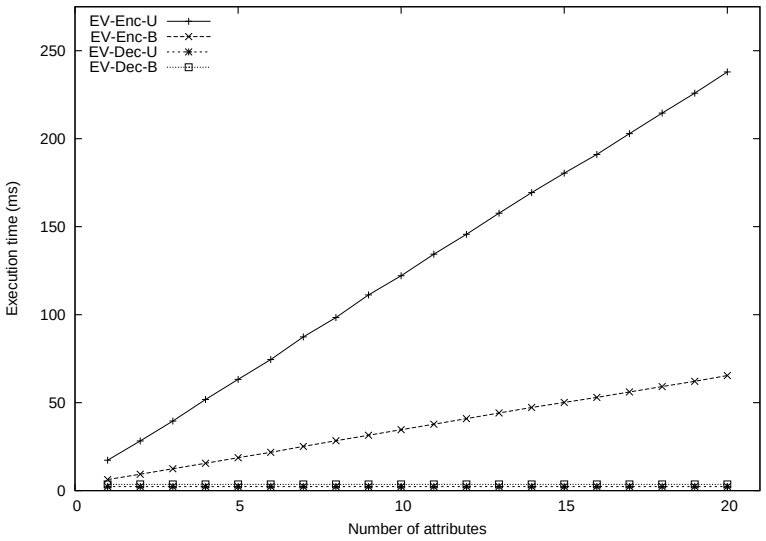


Figure 3.9: Event encryption and decryption times.

Figure 3.10 compares the execution times for filter encryption at subscriber and broker side. FE-U represents the encryption time at subscriber side and FE-B represents the filter re-encryption time at the broker side. The operations are performed for filters with different numbers of leaf nodes (i.e., attributes). We observe that the times grow linearly with the number of attributes and that the re-encryption time at the broker side is smaller than the encryption at the subscriber side, which is desirable if the broker needs to handle a large number of subscriptions and publications.

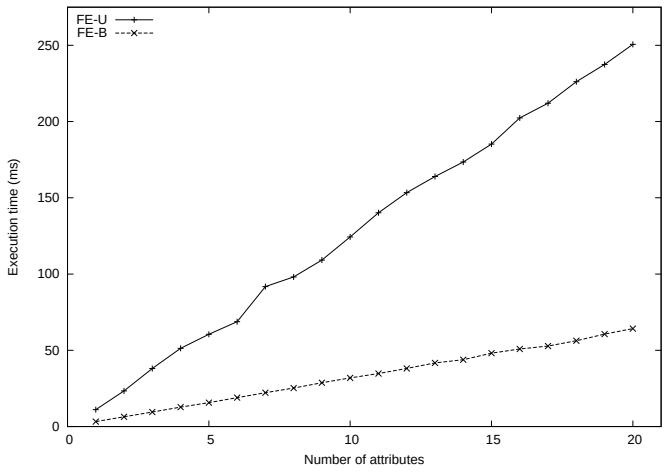


Figure 3.10: Filter encryption and re-encryption times.

The matching time of an event against a filter at the broker side also depends on the number of attributes. Matching a filter containing only one attribute against

an event with one attribute takes about 0.42 ms. The matching time increases linearly with the number of performed equality checks between leaf nodes and event attributes. Figure 3.11 shows the encrypted matching times for different numbers of attributes. The figure shows the matching time of one event against one filter. Because this solution is not indexed, matching one filter against 1000 events, would multiply the average matching time by 1000, putting the matching time of all 1000 events in the range of a few seconds. In some applications, this overhead may not be acceptable. In Chapter 6 we will discuss indexing strategies that could speed up the matching process.

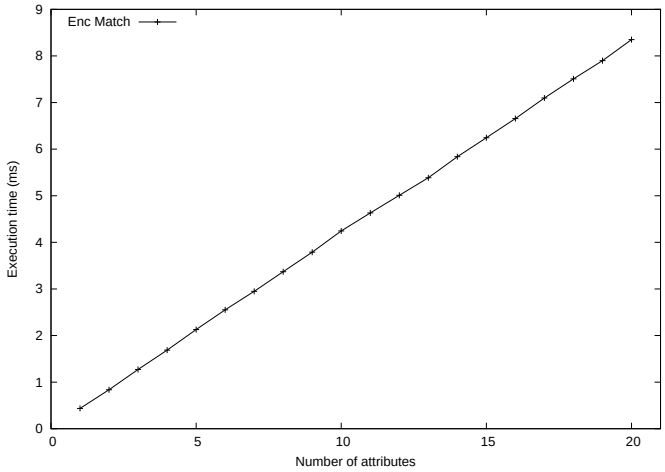


Figure 3.11: Encrypted matching times.

Chapter 4

Enforcing Fine-Grained Access Control Policies

Contents

4.1 Threat model

We previously assumed an honest-but-curious threat model for brokers, publishers and subscribers which means that they follow the protocol correctly, but are curious to learn as much as possible about the exchanged messages. Under this assumption,

brokers forward events according to the protocol and do not disclose encrypted event content or keys to other entities in the system. In the following, we additionally assume that brokers may try to *collude* with publishers or subscribers, or that publishers and subscribers might collude between them in order to get access to the content of events and filters for which they are not authorised. This means that brokers could make encrypted events available to publishers or subscribers that did not express a valid filter for them. Brokers could also try to combine any keys they might have that are required to run the protocol with the keys of publishers and subscribers.

For example, let us assume an employee of the IT company providing the pub/sub system and who has access to the keys stored at the broker and the ciphertext of publications, colludes with an employee of a pharmaceutical company that supplies some of the medicines to the hospital. This company subscribed to receiving new orders for drugs. The company would like to know the names and addresses of patients suffering from specific conditions in order to target advertisements to drugs and colludes with the malicious employee in order to learn this information.

4.2 Security properties

In addition to the properties *P1-P4* already identified, we require that an access control scheme also ensures the following properties.

Definition 14 (P5: Fine-grained access control policies)**.** *Providing mechanisms for enforcing fine-grained access control policies ensures that all access to data is legitimate, as granted by a Trusted Authority or by publishers themselves that can specify constraints about who can access the content of their events under specific conditions. The enforcement of access control policies should not reveal any information about the events or filters to brokers. Policies could be enforced for the whole event or for specific attributes of the event.*

Definition 15 (P6: Collusion resistance)**.** *Providing the collusion resistance property ensures that publishers, brokers and subscribers are not able to combine their keys in order to gain unauthorised access to the content of events or filters.*

4.3 Related work

None of the confidentiality solutions surveyed in Section 3.3 can enforce fine-grained access control policies on events (*P5*). The solutions from [Miklos 2002, Bacon 2008] address this problem, but they make the assumption that the brokers are trusted to read and even write events and enforce the policies. Bacon et al. [Bacon 2008] proposed a role-based access control mechanism for multiple administrative domains sharing a pub/sub network. They assume that each publisher and subscriber is connected to a local broker which is trusted to perform the encryption and decryption of the events or filters. An event is an instance of an event

type. Furthermore, an event type has an owner, a type name, and a list of event attributes. Each event attribute is associated with its own independent secret key to which trusted brokers have access. In order to perform content-based routing, brokers need to be authorised to access the decryption keys. Intermediate brokers are assumed to be untrusted, and hence, because complex encrypted filtering mechanisms are not provided, they can only forward events based on a single topic. Access control over the event content is enforced by controlling access to the decryption keys. Local brokers check the client's credentials against access control policies. To perform fine-grained access control over the content of an event, brokers can transform an event instance either after publication or before notification to a particular subscriber. Transformations may alter the values of an event or transform the event into another type. A broker can degrade, enrich or produce related event instances. This model allows fine-grained enforcement of access control policies over the attributes of events, but it needs to trust the brokers to enforce the policies and transform events. In our threat model, brokers are not trusted, so these solutions cannot be applied.

In order to enforce access control policies, current solutions require brokers to have access to the content of events, which is contrary to the event confidentiality requirements. Our goal is to propose a solution that can achieve both confidentiality of events and filters, support complex filters and enforce access control policies, while keeping key management scalable.

4.4 Background on security mechanisms

Attribute-based encryption (ABE) has several advantages over symmetric key encryption. First of all, senders and receivers do not need to share secret keys, thus simplifying key management for large scale dynamic applications. With ABE, a receiver can decrypt a ciphertext only if a decryption policy is satisfied. Thus, ABE also has the advantage that access control mechanisms are embedded in the data and decryption keys, and does not require a third party to enforce policies. Third, because messages and receivers can be described using any attributes and values of these attributes, ABE allows defining flexible and fine-grained access control policies.

In the following we give the details of the main types of ABE: Key-Policy ABE and Ciphertext-Policy ABE.

4.4.1 Key-Policy Attribute-based Encryption

Goyal et al. [Goyal 2006a] introduced Key-Policy ABE (KP-ABE) in which ciphertexts are labelled with sets of attributes and private keys are associated with access structures. A key is able to decrypt a ciphertext if its associated access structure is satisfied by the attributes of the ciphertext. The access structure, represented as a tree, allows expressing any monotone access formula consisting of *AND*, *OR*, or threshold gates.

KP-ABE consists of the following algorithms: KP–ABE–Init, KP–ABE–Enc, KP–ABE–KeyGen and KP–ABE–Dec.

The initialization algorithm is run by a Key Authority and computes the public parameters PK_{KP} that are sent to all encrypters and decrypters, and the master secret key MK_{PK} that is stored securely by the authority.

Algorithm 15 KP-ABE Init

Input: The security parameter 1^k and a number n, the maximum number of attributes under which a message can be encrypted.

Output: The public parameters PK_{KP} and the master secret key MK_{PK}.

1: Choose $\mathbb{G}_1$ a bilinear group of prime order p and size k, and let g be a generator of $\mathbb{G}_1$.
2: Choose $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ a bilinear map. The bilinear map e has the following properties:

   1. Bilinearity: for all $u, v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
   2. Non-degeneracy: $e(g, g) \neq 1$.

3: Define the Lagrange coefficient $\Delta_{i,S}$ for $i \in \mathbb{Z}_p$ and a set S of elements in $\mathbb{Z}_p$:
$\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$.
4: Choose a collision resistant function $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$. This function will be used to map each attribute to a number in $\mathbb{Z}_p^*$ which allows using arbitrary strings as attributes and adding them to a user's private key.
5: Choose a random y from $\mathbb{Z}_p$ and compute $g_1 = g^y$.
6: Choose a random element g_2 from $\mathbb{G}_1$.
7: Choose $t_1, ..., t_{n+1}$ uniformly at random from $\mathbb{G}_1$.
8: Let N be the set $\{1, 2, ..., n+1\}$. Define a function T as:
$T(X) = g_2^{X^n} \prod_{i=1}^{n+1} t_i^{\Delta_{i,N}(X)}$.
8: $PK_{KP} \leftarrow (g_1, g_2, t_1, ..., t_{n+1})$
8: $MK_{KP} \leftarrow y$
9: **return** PK_{KP}, MK_{KP}

Anybody can encrypt an element m of group $\mathbb{G}_2$ under a set of attributes using just the public parameters as shown in Algorithm 16. We note that the set of attributes γ is obtained by mapping each string attribute to a $\mathbb{Z}_p^*$ element using the collision resistant function H_1.

In order to decrypt a message, a user needs a secret key generated by the key authority using the master secret key. Each decryption key is computed for an access tree structure that represents the access rights of the user. For example, a key could grant access to a user to messages encrypted under attributes that satisfy a policy like "SYM=GOOG or SYM=IBM". The main idea is that any filter representing conjunctions and disjunctions of attributes can be represented as a tree in which leaf nodes are attributes and non-leaf nodes are threshold gates. A threshold gate is described by a threshold value and its children. Let x be a non-leaf node with threshold value k_x and having a number of children equal to

Algorithm 16 KP-ABE-Enc

Input: An element m, a set of attributes γ, and the public parameters PK_{KP}.
Output: The ciphertext $KP(m)$.

1: Choose a random s from $\mathbb{Z}_p$.
2: $E^{'} \leftarrow m \cdot e(g_1, g_2)^s$
3: $E^{''} \leftarrow g^s$
4: **for all** a in γ **do**
5: compute $E_a \leftarrow T(a)^s$
6: **end for**
7: $KP(m) \leftarrow (\gamma, E^{'}, E^{''}, \{E_a\}_{a \in \gamma})$.
8: **return** $KP(m)$.

num_x. The threshold value k_x represents the number of children of the non-leaf node that need to be satisfied in order for the node to be satisfied. When $k_x = 1$ it means that only one child needs to be satisfied, making the threshold gate an *OR*. When $k_x = num_x$, all children need to be satisfied making the threshold gate an *AND*. By allowing threshold values between 1 and num_x (i.e., $1 \leq k_x \leq num_x$), one can express more general conditions such as 2 out of 3 attributes should be satisfied. Each leaf node x is described by an attribute and has a threshold value $k_x = 1$, meaning that the leaf node is satisfied when the attribute is present, and not satisfied otherwise. Additionally, define the following functions on the tree: $threshold(x)$ returns the threshold value of the node, $attr(x)$ is defined only for a leaf node and returns the attribute associated with x, and $parent(x)$ returns the parent of a node x. Furthermore, define an ordering between the children of every node x and give each child an index from 1 to num_x. The function $index(x)$ returns the index associated to node x. Figure 4.1 shows the tree generated for the simple access policy "SYM=GOOG or SYM=IBM". The threshold value of the root note is 1.

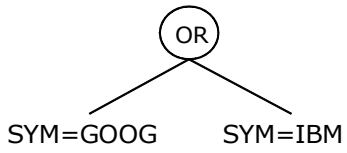


Figure 4.1: A simple access policy tree.

The KA generates a decryption key for a specific access policy. The details of the key generation algorithm are given in Algorithm 17.

A user is able to decrypt a ciphertext, only if the attributes of the ciphertext satisfy the access structure of its key. The decryption algorithm makes use of a recursive function DecryptNode shown in Algorithm 18 that is called on the root node of the access tree embedded in the decryption key. We note that in KP-ABE, the attributes under which the message is encrypted are attached to the ciphertext, and also the access policy is attached to the decryption key in the clear,

Algorithm 17 KP-ABE KeyGen

Input: An access tree F, public parameters PK_{KP} and master secret key $MK_{PK} = y$.
Output: A decryption key D_F for the access structure.

1: **for all** nodes x in tree F **do**
2: create a polynomial q_x with degree $d_x = threshold(x) - 1$
3: **end for**
4: For the root node r, set $q_r(0) = y$ and choose d_r points at random to completely define the polynomial q_r.
5: **for all** other nodes x in tree F **do**
6: set $q_x(0) = q_{parent(x)}(index(x))$
7: choose d_x other points randomly to completely define q_x.
8: **end for**
9: **for all** leaf nodes x in F **do**
10: $D_x \leftarrow g_2^{q_x(0) \cdot T(b)^{r_x}}$, where $b = attr(x)$
11: $R_x \leftarrow g^{r_x}$ where r_x is chosen uniformly at random from $\mathbb{Z}_p$ for each node x.
12: **end for**
13: $D_F \leftarrow \{(D_x, \{R_x)\}_x$, where x is leaf node in F.
14: **return** D_F.

which could leak sensitive information. Our solution uses ABE, but it encrypts the attributes in the ciphertext and access policy. If the root node r is satisfied, $\mathsf{DecryptNode}(KP(m), D_F, r)$ returns $e(g, g_2)^{ys} = e(g_1, g_2)^s$, because $q_r(0)$ was set to be y in KP–ABE–KeyGen. The user obtains m by dividing $E' = m \cdot e(g_1, g_2)^s$ with $e(g_1, g_2)^s$.

4.4.2 Ciphertext-Policy Attribute-based Encryption

Bethencour et al. [Bethencourt 2007] proposed a concrete construction for ciphertext policy ABE (CP-ABE) in which policies (access structures) are associated with data and attributes are associated with keys. This is similar to the capability model in access control. A key can decrypt some data if its associated attributes satisfy the policy associated with the data.

The initialization algorithm CP–ABE–Init is run by the Key Authority and generates the public parameters PK_{CP} which are sent to all users and the master secret key MK_{CP} which is kept securely by the authority. Algorithm 19 shows the details of the algorithm.

Algorithm 18 KP-ABE DecryptNode

Input: A ciphertext $KP(m) = (\gamma, E^{'}, E^{''}, \{E_a\}_{a\in\gamma})$, a decryption key D_F, a node x in the access tree of the key.

Output: A group element of $\mathbb{G}_2$ or $\perp$ if the node cannot be satisfied by the attributes of the ciphertext.

if x is a leaf node **then**
  if $attr(x) \in \gamma$ **then**
    return $\frac{e(D_x,E'')}{e(R_x,E_b)} = \frac{e(g_2^{q_x(0)}\cdot T(b)^{r_x},g^s)}{e(g^{r_x},T(b)^s)} = \frac{e(g_2^{q_x(0)},g^s)\cdot e(T(b)^{r_x},g^s)}{e(g^{r_x},T(b)^s)} = e(g,g_2)^{sq_x(0)}$
  else
    return $\perp$
  end if
else
  for each child z of x **do**
    $F_z \leftarrow \mathsf{DecryptNode}(KP(m), D_F, z)$
  end for
  Let S_x be an arbitrary set of $threshold(x)$ children nodes z such that $F_z \neq \perp$.
  if no such S_x set exists **then**
    return $\perp$.
  else
    return

$$
\begin{aligned}
F_x &= \prod_{z\in S_x} F_z^{\Delta_{i,S'_x(0)}}, \text{ where } i = index(z), S'_x = \{index(z) : z \in S_x\} \\
&= \prod_{z\in S_x} (e(g,g_2)^{s\cdot q_z(0)})^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z\in S_x} (e(g,g_2)^{s\cdot q_{parent(z)}(index(z))})^{\Delta_{i,S'_x}(0)} \text{ (by construction)} \\
&= \prod_{z\in S_x} (e(g,g_2)^{s\cdot q_x(0)})^{\Delta_{i,S'_x}(0)} \\
&= e(g,g_2)^{sq_x(0)} \text{ (using polynomial interpolation)}
\end{aligned} \tag{4.1}
$$

  end if
end if

Algorithm 19 CP-ABE Init

Input: The security parameter 1^k.

Output: The public parameters PK_{CP} and the master secret key MK_{CP}.

Choose $\mathbb{G}_0$ and $\mathbb{G}_1$ two bilinear groups of prime order p and size k, and let g be a generator of $\mathbb{G}_0$.
Choose $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$ a bilinear map.
Define the Lagrange coefficient $\Delta_{i,S}$ for $i \in \mathbb{Z}_p$ and a set S of elements in $\mathbb{Z}_p$:
$\Delta_{i,S}(x) = \prod_{j\in S, j\neq i} \frac{x-j}{i-j}$.
Choose two random exponents $\alpha, \beta \in \mathbb{Z}_p$.
$PK_{CP} \leftarrow (\mathbb{G}_0, g, h = g^\beta, f = g^{1/\beta}, e(g,g)^\alpha)$
$MK_{CP} \leftarrow (\beta, g^\alpha)$
return PK_{KP} and MK_{KP}.

The Key Authority issues to each user a secret decryption key for the attributes or credentials of the user. This key is generated using the master secret key MK_{CP} as shown in Algorithm 20.

Algorithm 20 CP-ABE KeyGen

Input: A set of attributes S, the public parameters PK_{CP}, and master secret key MK_{CK}.
Output: A decryption key SK for the attributes.

1: Choose a random $r \in \mathbb{Z}_p$, and then a random $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$.
2: Compute the key as:
$SK = \left(D = g^{(\alpha+r)/\beta}, \forall j \in S : D_j = g^r \cdot H(j)^{r_j}, D_j^{'} = g^{r_j}\right)$
3: **return** SK.

To encrypt a message, a user needs to define first an access control policy describing the attributes of users that can decrypt the message. For example, one decryption policy might be: "nurse and level>3 and hospital=San Raffaele". To allow expressing conditions such as "level>3", [Bethencourt 2007] show how to extend the access tree to represent numeric inequalities by using a "bag of bits" representation of the numeric value. For example, $a < 7$ can be represented as shown in Figure 4.2, assuming that the value of a is represented on 4 bits.

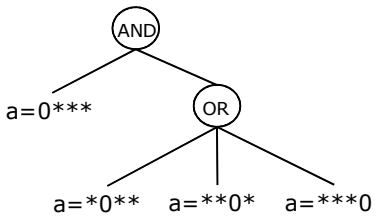


Figure 4.2: Tree representation for $a < 7$ on 4 bits.

The algorithm for representing numeric inequalities as access trees is the following.

Algorithm 21 Inequality Policy Generation

Input: A char gt which is '0' for "less than" and '1' for "greater than" inequalities, the attribute name $attr$, the comparison value v, and the number of bits b on which to represent v.

Output: The TreePolicy p.

1: Find the position i of the first bit in v that does not equal gt starting from the least significant bit.
2: Create a TreePolicy p with only a leaf node that has gt at the ith position, e.g., $attr : **gt*$.
3: **for** $i = i + 1; i < b; i++$ **do**
4: **if** gt equals '1' **then**
5: **if** the bit at position i is '1' **then**
6: update p to a AND policy with one child the old p and the other a leaf node that has gt at the ith position.
7: **else**
8: update p to an OR policy with one child the old p and the other a leaf node that has gt at the ith position.
9: **end if**
10: **end if**
11: **if** gt equals '0' **then**
12: **if** the bit at position i is '1' **then**
13: update p to an OR policy with one child the old p and the other a leaf node that has gt at the ith position.
14: **else**
15: update p to an AND policy with one child the old p and the other a leaf node that has gt at the ith position.
16: **end if**
17: **end if**
18: **end for**
19: **return** p

In a second step, the policy is simplified by merging AND subtrees with AND parents and OR subtrees with OR parents.

To encrypt a message $m \in \mathbb{G}_1$ under a policy T, a user only needs the public parameters PK_{CP} from the key authority and then proceeds as shown in Algorithm 22.

Algorithm 22 CP-ABE-Enc

Input: An element m, an access tree structure T, and the public parameters PK_{CP}.
Output: The ciphertext $CP(m)$.

1: **for all** nodes x in tree T **do**
2: create a polynomial q_x with degree $d_x = threshold(x) - 1$
3: **end for**
4: for the root node R, choose a random s and set $q_R(0) = s$.
5: choose d_R points at random to completely define the polynomial q_r.
6: **for all** other nodes x in tree T **do**
7: set $q_x(0) = q_{parent(x)}(index(x))$
8: choose d_x other points randomly to completely define q_x.
9: **end for**
10: Let Y be the set of leaf nodes in T. Compute the ciphertext as:
$CP(m) = \Big(F, \widetilde{C} = me(g,g)^{\alpha s}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(attr(y))^{q_y(0)}\Big)$.
11: **return** $CP(m)$.

To decrypt a ciphertext, the attributes of the user need to satisfy the access policy under which the ciphertext was encrypted. The decryption algorithm makes use of a recursive DecryptNode function which fails and returns $\perp$ if the policy cannot be satisfied. The decryption algorithm calls DecryptNode on the root node of the tree, which proceeds as shown in Algorithm 23. If the tree is satisfied, the algorithm returns $e(g,g)^{rq_r(0)} = e(g,g)^{rs}$. The message m can be computed as: $\widetilde{C}/(e(C,D)/A) = \widetilde{C}/(e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{rs}) = m$.

Algorithm 23 CP-ABE: DecryptNode

Input: A ciphertext $CP(m) = \left(F, \widetilde{C} = me(g,g)^{\alpha s}, C = h^s, \forall y \in Y : C_y, C_y^{'}\right)$, a decryption key $SK = (D, \forall j \in S : D_j, D_j^{'})$, and a node of T.

Output: A group element of $\mathbb{G}_1$ or $\perp$ if the node cannot be satisfied by the attributes of the key.

if x is a leaf node **then**
  if $attr(x) \in S$ **then**
    return $\frac{e(D_i,C_x)}{e(D_x',C_x')} = \frac{e(g^r \cdot H(i)^{r_i}, h^{q_x(0)})}{e(g^{r_i}, H(i)^{q_x(0)})} = e(g,g)^{rq_x(0)}$
  else
    return $\perp$
  end if
else
  for each child z of x **do**
    $F_z \leftarrow \mathsf{DecryptNode}(CP(m), SK, z)$
  end for
  Let S_x be an arbitrary set of $threshold(x)$ children nodes z such that $F_z \neq \perp$.
  if no such S_x set exists **then**
    return $\perp$.
  else
    return

$$
\begin{aligned}
F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S_x'(0)}}, \text{ where } i = index(z), S_x' = \{index(z) : z \in S_x\} \\
&= \prod_{z \in S_x} (e(g,g)^{r \cdot q_z(0)})^{\Delta_{i,S_x'}(0)} \\
&= \prod_{z \in S_x} (e(g,g)^{r \cdot q_{parent(z)}(index(z))})^{\Delta_{i,S_x'}(0)} \text{ (by construction)} \\
&= \prod_{z \in S_x} (e(g,g)^{r \cdot q_x(0)})^{\Delta_{i,S_x'}(0)} \\
&= e(g,g)^{rq_x(0)} \text{ (using polynomial interpolation)}
\end{aligned}
\tag{4.2}
$$

  end if
end if

4.5 Solution details

Our basic confidentiality solution described in Chapter 3 provides confidentiality of events (*P2*) and filters (*P3*), complex filters (*P4*), while not requiring publishers and subscribers to share keys (*P1*). The scheme is also secure against collusion between publishers and subscribers. If any number of publishers and subscribers share their keys, they are still not able to determine the master key and decrypt other messages because the broker holds the other side of all their keys. However, if a publisher or subscriber collude with the broker, they can compute the master key for proxy encryption $MK_{PE} = x$ and then are able to decrypt all the events. In order to circumvent this weakness, we need an encryption scheme that is collusion resistant and assigns to each subscriber a key that decrypts only events that satisfy

the subscriber's filter. If a subscriber colludes with a broker, other subscribers or publishers, they should not be able to decrypt events for which they did not register a valid filter. In the following we introduce a new scheme that can additionally provide the collusion resistance property.

To achieve this, we use KP-ABE encryption. In KP-ABE, messages are encrypted under a set of attributes. In our scenario, publishers already attach a set of attributes to each event. These attributes describe the events and are used by brokers to match events against filters. We propose to encrypt the message content M under the same set of attributes γ used for routing. Decryption keys in KP-ABE are defined for a tree access policy such as the one we used to express filters, and can decrypt only ciphertexts that have attached attributes that satisfy the policy. Hence, if the Trusted Authority gives to each subscriber a KP-ABE decryption key corresponding to the subscriber's filter, the subscriber will be able to decrypt only events that satisfy the filter and not others. In this way, each subscriber has a set of keys that decrypt only events to which it subscribed. KP-ABE [Goyal 2006a] is collusion resistant. Brokers do not get any KP-ABE keys while publishers are only given public keys. Subscribers are given a secret key for each filter policy, but because the KA uses a unique random number to blind each key policy, subscribers cannot combine their keys to get access to events for which they did not subscribe. For example, if subscriber s_1 has a key for *SYM=GOOG AND PRICE>100* and subscriber s_2 has a key for *MSFT AND PRICE>20*, they cannot combine their keys to decrypt an event with attributes *SYM=GOOG, PRICE=90*. KP-ABE encryption and decryption are more computationally expensive than PE, but do not require the broker to perform re-encryption or pre-decryption of events. In the following we give the details of the enhanced scheme supporting fine-grained access control using KP-ABE.

4.5.1 Initialization

The Trusted Authority runs the initialization algorithm for KP-ABE KP–ABE–Init (Algorithm 15) to generate the public parameters PK_{KP} and the master secret key MK_{KP}. The KA also runs the initialization algorithm of SDE SDE–Init to generate the public parameters PK_{SE} and the master secret key $MK_{SE} = (x, s)$. Additionally, the KA computes and gives to every user i (publisher or subscriber) the secret key $K_{ui} = (s, x_{i1})$ and gives the corresponding key $K_{si} = (i, x_{i2})$ to its local broker.

4.5.2 Event encryption

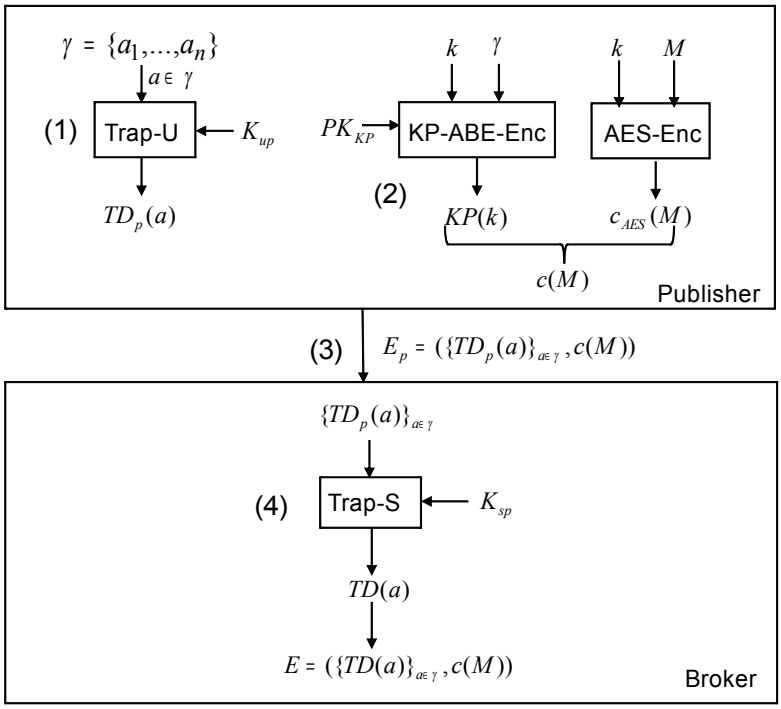


Figure 4.3: Event encryption with KP-ABE.

Figure 4.3 shows the steps needed to encrypt an event. The publisher specifies a set of attributes γ and a content M and encrypts them as follows:

1. The publisher p encrypts the attributes as trapdoors. For every attribute $a \in \gamma$, the publisher computes a trapdoor as $TD_p(a) \leftarrow \mathsf{Trap\text{–}U}((x_{p1}, s), a)$ as in Algorithm 11.

2. The publisher p encrypts the message content M:

   - Generate a random AES encryption key k.
   - Encrypt M under k using AES as $c_{AES}(M) \leftarrow \mathsf{AES\text{–}Enc}(M, k)$.
   - Encrypt k using KP-ABE as $KP(k) \leftarrow \mathsf{KP\text{–}ABE\text{–}Enc}(k, \gamma, PK_{KP})$.
   - Because $KP(k) = (\gamma, E', E'', \{E_a\}_{a\in\gamma})$ contains the set of attributes γ unencrypted, we replace them with the trapdoors of the attributes and create the event:
     $E_p = (\{TD_p(a)\}_{a\in\gamma}, E', E'', \{E_a\}_{a\in\gamma}, c_{AES}(M)) = (\{TD_p(a)\}_{a\in\gamma}, c(M))$.

3. The publisher sends the encrypted event E_p to the broker.

4. The broker locates the key $K_{sp} = (p, x_{p2})$ corresponding to the publisher and re-encrypts each trapdoor $\{TD_p(a)\}_{a\in\gamma}$ as $TD(a) \leftarrow \mathsf{Trap\text{–}S}(TD_p(a), K_{sp})$. The final encrypted event is:
$E = (\{TD(a)\}_{a\in\gamma}, E', E'', \{E_a\}_{a\in\gamma}, c_{AES}(M)) = (\{TD(a)\}_{a\in\gamma}, c(M))$.

The above operations provide confidentiality of the message and attributes of the event, thus achieving property *P1*.

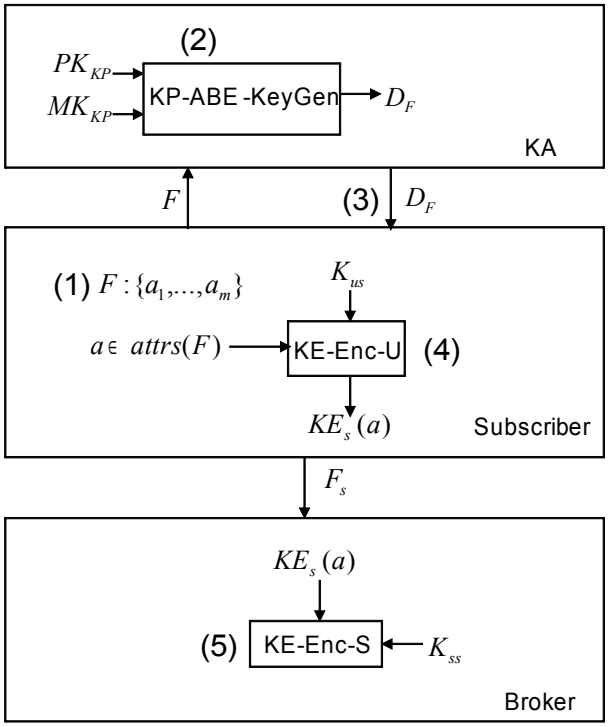


Figure 4.4: Filter generation and encryption.

4.5.3 Filter encryption

Figure 4.4 shows the main steps for generating and encrypting the filter. Filter construction as an access tree and encryption with SDE is performed as in Section 3.5.5. Additionally, we require that the KA generates a decryption key corresponding to the filter policy that decrypts only events that match the filter. Algorithm 17 shows the decryption key generation algorithm.

The steps that need to be performed to generate and encrypt a filter are the following.

1. The subscriber defines the filter as an access tree F. The access tree is constructed as in Section 3.5.5.

2. The subscriber sends the filter F to the KA and requests a decryption key D_F. The KA runs $\mathsf{KP\text{–}ABE\text{–}KeyGen}(F, PK_{KP}, MK_{KP})$ to generate the decryption key D_F.

3. The KA sends the decryption key D_F to the subscriber on a secure channel. The subscriber stores the key securely.

4. To provide confidentiality of the filter, the subscriber encrypts each leaf node x in F by running $KE_s(a) = \mathsf{KE\text{–}Enc\text{–}U}(K_{us}, a)$, where $a = attr(x)$. The subscriber sends the filter F_s encrypted in this way to the broker.

5. The broker re-encrypts the filter as in the previous scheme. The broker locates the key $K_{ss} = (s, x_{s2})$ corresponding to the subscriber and re-encrypts the leaf-node attributes of F_s. For each attribute $KE_s(a)$ run $KE(a) \leftarrow \mathsf{KE\text{–}Enc\text{–}S}(K_{ss}, KE_s(a))$.

4.5.4 Encrypted matching

The encrypted matching operation is the same as in the basic confidentiality scheme from Chapter 3.

4.5.5 Event decryption

The subscriber performs the KP-ABE decryption of the event using the key D_F. If the key policy of the subscriber is satisfied by the attributes of the event, the KP-ABE decryption returns the key k used to encrypt the content M. The subscriber then decrypts the message content as $M \leftarrow \mathsf{AES\text{–}Dec}(k, c_{AES}(M))$.

4.6 User revocation and subscription expiration

Users can be prevented from publishing new events and registering new filters by revoking their SDE broker side of the key $K_{si} = (i, x_{i2})$. The Trusted Authority revoking the user's rights needs to send a request to the broker to delete the user's key from its keystore. In the basic scheme, subscribers have one key for decrypting all events. So if a subscriber has registered several filters and the authority wants to revoke a subset of the filters, the only way to prevent the subscriber from receiving events matching the revoked subscriptions, is for the KA to contact the broker and request to unsubscribe the expired filters.

In the enhanced scheme, subscribers get a unique decryption key from the KA for each filter. The KA can make decryption keys valid until a specific date or time. After that, the subscriber cannot use the key any more to decrypt events and needs to contact the KA to renew the key. The subscriber does not need to contact the broker to renew the filter every time the key gets renewed. We present in the following the modified enhanced scheme with subscription expiration dates.

4.6.1 Initialization

The Initialization algorithm is run by the KA as in the previous scheme.

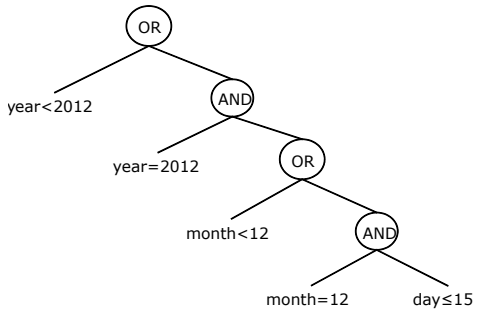


Figure 4.5: Example of a policy for expiration date 15/12/2012.

4.6.2 Event encryption

To prevent subscribers from using expired decryption keys, publishers add to the list of attributes under which M is encrypted date information. For example, the publisher adds to the list of attributes $\{a_i\}$ that describe the event the following attributes: "day=17, month=9, year=2011". These attributes do not need to be encrypted, but only used in the KP-ABE encryption of M. So for every date attribute a, the publisher needs to additionally compute $E_a = T(a)^s$ and add it to the encrypted event E_p (see Section 4.5.2).

4.6.3 Filter encryption

The subscriber generates the filter and sends it to the KA to obtain a decryption key as before. The KA extends the access tree structure of the filter F with the condition that date is prior to a specific expiration date and creates a KP-ABE decryption key for the extended access structure. The extended access structure becomes F AND *date policy*. Figure 4.5 shows an example of a date policy. The leaf nodes representing numeric qualities and inequalities need to further be expanded as sub-trees using "bag of bits" representations of the attributes as shown in Figure 4.2.

4.6.4 Encrypted matching

This operation is performed by the broker as in the previous scheme. The date attributes added by the publisher to the event are simply ignored by the broker.

4.6.5 Event decryption

The subscriber uses the KP-ABE decryption key to decrypt the event. The decryption succeeds only if the key is unexpired.

4.7 Enforcing publisher-defined access control policies

To allow publishers to express constraints on who can access the message content M (or parts of the content), we use CP-ABE encryption. CP-ABE allows a publisher to encrypt a message under an access policy, similar to the policy we used to represent a filter. Only subscribers possessing attributes that satisfy the policy can decrypt the message. For example, a publisher might specify that only employees of a particular organization holding a specific position should have access to the content of an event. The drawback of CP-ABE is that it sends the encryption policy in the clear together with the ciphertext. To hide the policy from the broker, we encrypt the policy using PE which preserves the decoupling of publishers and subscribers. When the subscriber receives the policy, it can decrypt the event only if it has the required credentials. However, this approach does not enable the broker to check the policies, and as a result, subscribers would receive events that they cannot decrypt, resulting in network overhead. In the following we describe a solution that enables brokers to check encrypted policies by using the multi-user SDE scheme to encrypt policies and subscribers' attributes. In this way, it is possible to verify if a subscriber satisfies the access policy expressed by the publisher, while not revealing the policy to the broker, and without requiring publishers and subscribers to share keys.

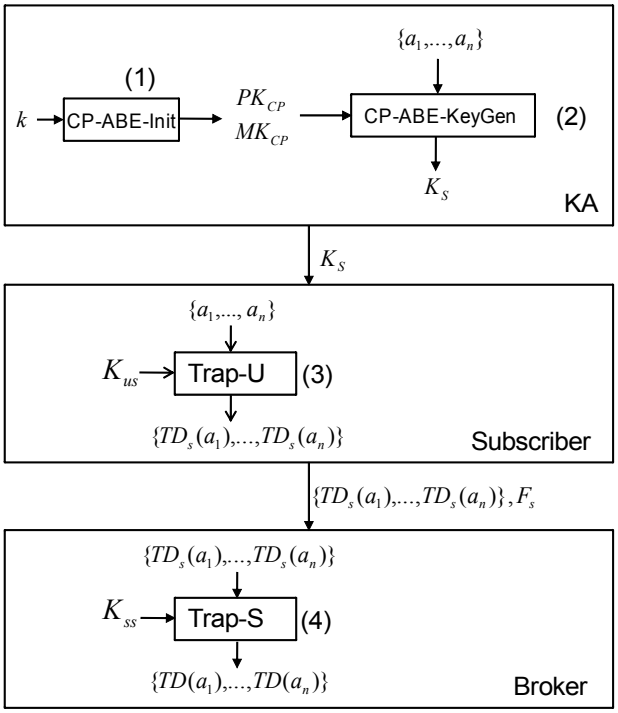


Figure 4.6: Decryption key generation and attribute encryption.

Figure 4.6 shows the steps needed to generate a CP-ABE decryption key for the subscriber and to encrypt the subscriber's attributes using SDE.

1. The KA runs the CP–ABE–Init(k) algorithm to generate the public key PK_{CP} and the master secret key MK_{CP} for CP-ABE.

2. The KA receives a request from a user to certify its attributes. The KA runs CP–ABE–KeyGen to generate a decryption key K_s for the subscriber's attributes. To make the decryption key valid until a specific expiration date, the KA includes in the key attributes for the expiration date such as "day=1", "month=12", "year=2013".

3. The subscriber encrypts its attributes using multi-user SDE by invoking the method KE–Enc–U described in Algorithm 9.

4. The broker re-encrypts the subscriber's attributes by calling the method KE–Enc–S described in Algorithm 10.

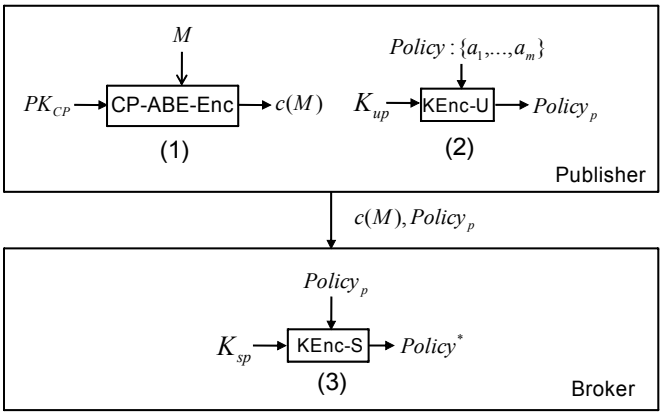


Figure 4.7: Policy encryption.

Figure 4.7 shows the steps for encrypting an event under a policy and encrypting the policy.

1. The publisher generates the policy *Policy* as a tree access structure. The publisher encrypts the message using CP-ABE under the policy.

2. The publisher encrypts the policy *Policy* with SDE by calling the function KE–Enc–U described in Section 6.4 on the leaf nodes of the policy. The subscriber then forwards the ciphertext together with the encrypted policy to the broker. If needed, the subscriber could re-encrypt the message using the enhanced method. The CP-ABE encryption ensures that only subscribers possessing the required attributes are able to decrypt the message, thus enforcing the access control policy. The enhanced method ensures that only subscribers who registered a filter matching the event can decrypt it.

3. The broker re-encrypts the policy *Policy** by calling KE–Enc–S on the leaf nodes of the access tree. The broker can now match the policy *Policy* against

the attributes $TD(a_1), ..., TD(a_n)$ of the publisher and will forward the event to the subscriber, if (i) the event matches the subscriber's interest, and (ii) the subscriber has the required attributes.

4.8 The e-health application revisited

In the following we show how by applying our solution to the example in Section 2.1 we are able to provide confidentiality of the data and enforce that only authorised parties are able to access it.

First we show how filters are generated and encrypted. Filters are expressed as access trees following the construction from [Bethencourt 2007], which is able to express inequalities of numerical attributes. The idea is to create an attribute for each bit of the number and use AND and OR gates to express the inequality. We illustrate how to do this for heart rate >120 in Figure 4.8. 120 = 1111000 in binary.

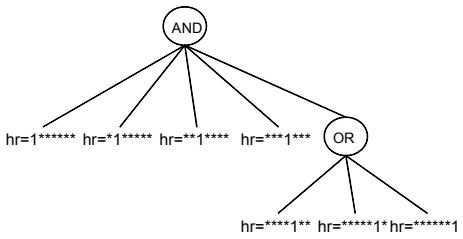


Figure 4.8: Access tree implementing heart_rate>120

The leaf attributes of the tree are encrypted using KE–Enc–U. The filter becomes:

$KE_d(name = John\_Smith)$ AND
$(KE_d(heart\_rate = 1******)$ AND $KE_d(heart\_rate = *1*****)$ AND $KE_d(heart\_rate = **1****)$ AND $KE_d(heart\_rate = ***1***)$ AND $(KE_d(heart\_rate = ****1**)$ OR $KE_d(heart\_rate = *****1*)$ OR $KE_d(heart\_rate = ******1))$
OR $KE_d(systolic\_pressure > 150)$ OR $KE_d(diastolic\_pressure > 100))$, where d is the ID of the doctor. We omit details for $KE_d(systolic\_pressure > 150)$ and $KE_d(diastolic\_pressure > 100))$ which are similar to the heart rate representation.

To encrypt a prescription, a doctor proceeds as follows. For every attribute like patient name, age, address, medication, etc. there is a hospital policy in place which specifies who can view the attribute and under which conditions. The policy is expressed using a CP-ABE access tree. Each attribute will be encrypted under the corresponding CP-ABE access tree. For example, the doctor would need to encrypt the prescription under the following policies.

P_{name}: (nurse AND level>3 AND San_Raffaele) OR (doctor AND San_Raffaele) OR DMS
P_{age}: senior_researcher OR (auditor AND HA1) OR (doctor AND San_Raffaele) OR DMS
$P_{address}$: DSM OR (nurse AND level>3 AND San_Raffaele) OR (doctor AND San_Raffaele)
$P_{symptom}$: senior_researcher OR auditor OR (nurse AND level>3 AND San_Raffaele) OR

(doctor AND San_Raffaele)
$P_{disease}$: DSM OR senior_researcher OR (auditor AND HA1) OR (nurse AND level>3 AND San_Raffaele) OR (doctor AND San Raffaele)
$P_{medication}$: DSM OR senior_researcher OR (auditor AND HA1) OR (nurse AND level>3 AND San_Raffaele) OR (doctor AND San_Raffaele)

The prescription becomes:
$\{name = JohnSmith\}_{P_{name}}$,
$\{age = 70\}_{P_{age}}$,
$\{address = via\ Tartini\ 12,\ Padova\}_{P_{address}}$,
$\{symptom = high\ blood\ pressure\}_{P_{symptom}}$,
$\{disease = primary\ hypertension\}_{P_{disease}}$,
$\{medication = Catapres\}_{P_{medication}}$.

Each attribute of the prescription will be encrypted under the corresponding policy. Though different parts of the information should be made available to different consumers such as the patient, Healthcare Authority, Research Center, DMS, other doctors or nurses working in the hospital, with our solution the message is encrypted and published only once. Without a proper encryption technique in place, a different message would need to be created for each different consumer type, as has been proposed in [Bacon 2008] and [Miklos 2002], where the local broker takes care of creating the different events.

To additionally enforce policies embedded in decryption keys issued by the Trusted Authority, the event is encrypted using KP-ABE under the attributes it contains, i.e. name=John Smith, age=70, etc. This ensures that only subscribers that registered a filter matched by the attribute of the event will receive the event. Brokers will be able to perform the check without learning the attributes of the event or the details of the filter. To hide the attributes from the broker, the publisher will replace them with trapdoors computed as in SDE.

The published message contains the encrypted prescription (using CP-ABE and KP-ABE) and the trapdoors of the prescription attributes computed as shown in step (3) in Figure 2. The trapdoors will be: TD(name=John Smith), TD(age=70), TD(symptom=primary hypertension) etc. The encrypted prescription will be forwarded to consumers by checking if these attributes satisfy the filters they registered. When they receive the message, only those who satisfy the CP-ABE policies will be able to recover the content of the prescription (or specific parts of it).

4.9 Security analysis

Figure 4.9 shows the different encryption schemes that are used to provide confidentiality of events and filters and fine-grained access control.

The filter and attribute encryptions are the same for both the basic and enhanced schemes. The two schemes differ in the way they encrypt the messages content. To prove the security of the enhanced scheme, we only need to prove that the

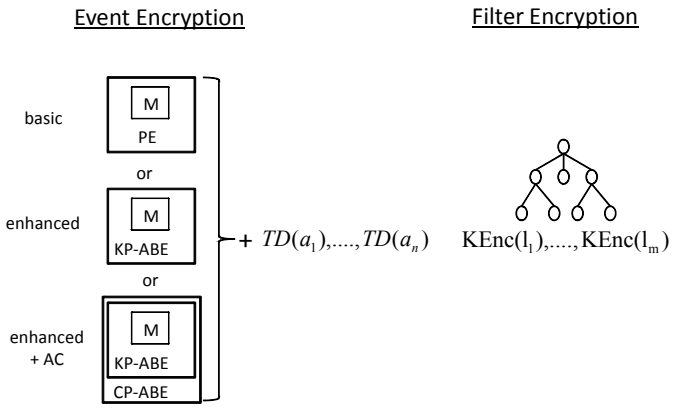


Figure 4.9: Event and filter encryption with access control.

message content encryption using KP-ABE is IND-CPA secure. About KP-ABE, [Ostrovsky 2007] proves that the KP-ABE with non-monotonic access structures in the attribute-based selective-set model reduces to the hardness of the Decisional Bilinear Diffie-Hellmann (DBDH) assumption, generally considered a hard problem. To prove the security of message encryption using KP-ABE, the proof is similar as for the basic scheme. The difference is that we consider an adversary $\mathcal{A}'$ that challenges the KP-ABE IND-CPA game using $\mathcal{A}$ as a subroutine and show that breaking the scheme reduces to breaking the KP-ABE IND-CPA game.

Theorem 4. *If the Decisional Bilinear Diffie-Hellmann (DBDH) problem is hard relative to $\mathbb{G}$, then the enhanced scheme is a non-adaptive indistinguishable secure scheme. The success probability of a PPT adversary $\mathcal{A}$ in breaking the enhanced scheme is defined as:*

$$\begin{aligned} Succ^{\mathcal{A}}(k) &= \tfrac{1}{2}Pr[\mathcal{A}((KP(\vec{M}_0), FE(\vec{F}_0), TD(\vec{a}_0))) = 0] + \\ &\quad \tfrac{1}{2}Pr[\mathcal{A}((KP(\vec{M}_1), FE(\vec{F}_1), TD(\vec{a}_1))) = 1] \\ &< \tfrac{1}{2} + negl(k) \end{aligned} \tag{4.3}$$

To enforce publisher specific access control policies, we use two layers of encryption: KP-ABE and CP-ABE. [Cheung 2007] proves CP-ABE to be IND-CPA under the DBDH assumption. [Bellare 2003] shows that if a cryptosystem is secure in the sense of indistinguishability, then the cryptosystem in the multi-user setting, where related messages are encrypted using different keys, is also secure. In our case each encryption layer uses an independent key so the combination is at least as secure as any individual encryption. Thus, encryption using KP-ABE and CP-ABE is at least IND-CPA secure. To prove that message encryption using the access control scheme is secure, we need to prove the following theorem.

Theorem 5. *If the Decisional Bilinear Diffie-Hellmann (DBDH) problem in hard relative to $\mathbb{G}_0$, then the enhanced scheme with access control is a non-adaptive indistinguishable secure scheme. The success probability of a PPT adversary $\mathcal{A}$ in*

breaking the scheme is defined as:

$$\begin{aligned} Succ^{\mathcal{A}}(k) = & \tfrac{1}{2} Pr[\mathcal{A}((CP(KP(\vec{M}_0)), FE(\vec{F}_0), TD(\vec{a}_0))) = 0] + \\ & \tfrac{1}{2} Pr[\mathcal{A}((CP(KP(\vec{M}_1)), FE(\vec{F}_1), TD(\vec{a}_1))) = 1] \\ & < \tfrac{1}{2} + negl(k) \end{aligned} \tag{4.4}$$

4.10 Implementation and performance analysis

For the enhanced scheme supporting access control policies, we used the SDE implementation described in the last chapter and implemented the KP-ABE and CP-ABE schemes as described in [Goyal 2006a] and [Bethencourt 2007] respectively, based on the Java Pairing Based Cryptography Library (jPBC)[1].

In our implementation we used for both KP-ABE and CP-ABE the symmetric "type A" pairings provided by jPBC, which are constructed on the curve $y^2 = x^3 + x$ over the field F_q, for some prime $q = 3 \bmod 4$. $\mathbb{G}_1$ is the group of points $E(F_q)$ with order r, some prime factor of $q + 1$. As parameters, we used 160 bits for r and 512 bits for q. We tested the implementation on an Intel Core2 Duo 2.8 GHz with 3.48 GB of RAM as previously.

We first compare the times for event encryption and decryption under the scheme introduced in this chapter, with the execution times of the same functions under the previous scheme. Filter generation and matching is the same in both schemes. In Figure 4.10 we compare the times needed for event encryption at the publisher side and re-encryption at the broker side, for the two methods. The figure shows the event encryption times at the publisher side, EV-Enc-U for the basic method and EV-Enc-U-2 for the enhanced method. We notice that event encryption in the enhanced scheme which uses KP-ABE to encrypt the message content M is only a little more expensive than the encryption in the basic scheme that uses PE. The opposite is true for event re-encryption at the broker side which is slightly slower when PE is used. That is because additionally to re-encrypting the attributes, the basic method also needs to re-encrypt the message content. Such re-encryption is not necessary when KP-ABE is used. In Figure 4.10, EV-Enc-B represents the times for event re-encryption using the basic method and EV-Enc-B-2 represents re-encryption times using the enhanced method. The encryption times grow linearly with the number of non-numerical attributes. When numerical attributes are used, a non-numerical attribute is created for each bit on which the attribute value is represented, and the encryption times grow faster. The encryption times shown in the figure are expressed in milliseconds and represent an average over 1000 executions.

[1] http://gas.dia.unisa.it/projects/jpbc/

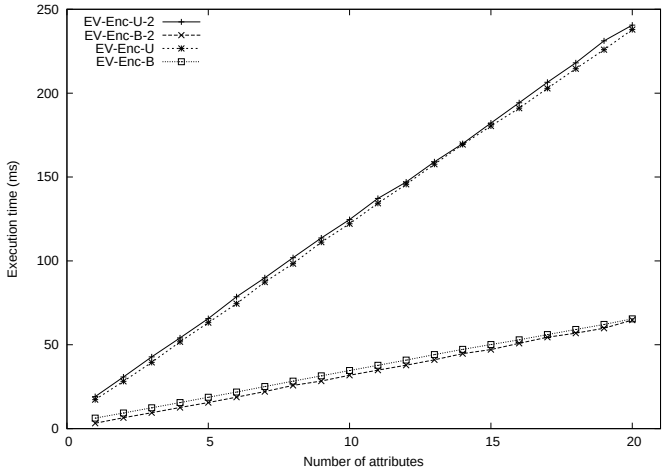


Figure 4.10: Event encryption times - comparison of the basic and enhanced schemes.

Figure 4.11 compares the event decryption times when using the two methods. Event decryption with PE at broker side (EV-Dec-B) and subscriber side (EV-Dec-U) are constant, while decryption with KP-ABE (EV-Dec-2) grows linearly with the number of attributes. In order to reduce the event decryption time at the subscriber side, we use an optimised method in which the subscriber does not repeat the encrypted matching operations between the attributes of event and filter, already performed by the broker. The broker attaches to the filter the information about which leaf nodes are satisfied by which attributes of the event. This reduces the KP-ABE decryption time for which we obtain better results than with PE when the number of attributes is smaller than 11. For values greater then 11, decryption times are bigger for the enhanced method.

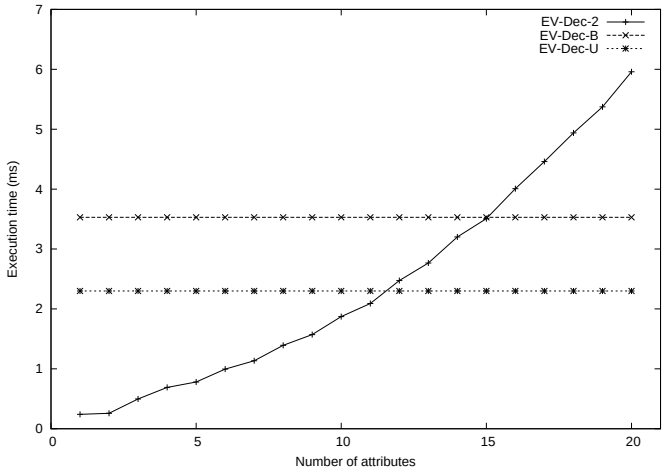


Figure 4.11: Event decryption times - comparison of the basic and enhanced schemes.

In the enhanced scheme, to enforce access control policies, for each filter registered by a subscriber, the KA generates a KP-ABE decryption key that enables the subscriber to decrypt only events matching this filter. The decryption key generation is more expensive than filter encryption, but this can be acceptable if we assume that the KA has more computational resources than a subscriber. Figure 4.12 shows the times for key generation for a filter, computed by the KA, Keygen(KA), for different number of attributes. We observe again that the times grow linearly with the number of attributes.

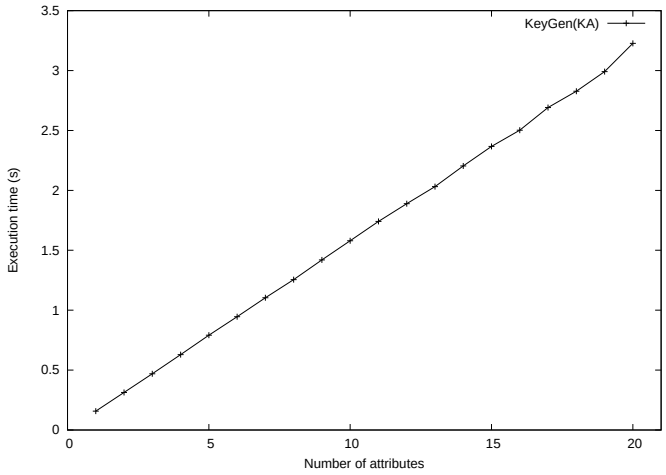


Figure 4.12: Decryption key generation times.

If the publisher wants to express additional constraints on who can read the content of an event, the event needs to be encrypted with CP-ABE. Figure 4.13 shows the decryption key generation for the attributes of the subscriber performed by the KA (KeyGen), the encryption policy generation performed by the publisher (PolicyGen), the message encryption (Enc), and the message decryption (Dec) times. To measure the performance of the CP-ABE encryption, we encrypted a simple message string under different policies with different numbers of attributes. We notice that the times increase linearly with the number of attributes in the policy. The decryption time depends both on the number of attributes in policy and on the number of attributes in the key. We used the same attributes for both and we considered the worst case scenario in which all the attributes in the policy need to be checked.

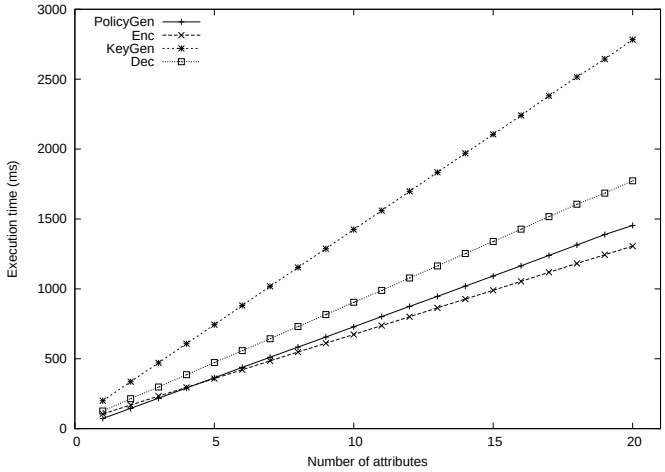


Figure 4.13: CP-ABE encryption time.

Chapter 5

Querying In-Network Cached Publications

Contents

Caching data in the network enhances support for user and device mobility, making data available all the time and reducing power consumption for mobile devices. Moreover, querying historic publications can be useful in many applications. For example, a researcher from a hospital could need to query data about patients with a particular disease over a number of years. An Energy Management & Control System might need past data in order to predict and better schedule the usage of devices inside a house. A Utility company might also need past data in order to better predict demand and failures. Though many pub/sub systems discard publications once they are delivered to interested subscribers, some of them enhance brokers with databases where they can store historic publications. One such example is the PADRES middleware that we used to integrate and test our scheme as it will be explained in Chapter 7. Moreover, most ICN implementations cache and replicate data in the network.

In this chapter we are concerned with how to index and query cached publications in a database stored at a broker. The problem with searches over encrypted databases and building secure indexes has long been studied in the literature. We start by giving an overview of such techniques and highlight their shortcomings.

An encrypted search scheme for cached publications should maintain the same security properties we identified previously for publications and subscriptions, i.e., *P1*: Scalable key management, *P2*: Publication confidentiality , *P3*: Subscription (or query) confidentiality, and *P4*: Complex encrypted matching. Additionally, we may want to support *P5*: Fine-grained access control policies and *P6*: Collusion resistance. In the following we show that none of the solutions we surveyed provides all these properties. We then describe how we can adapt our event and publication schemes to provide all of these properties for search on encrypted databases as well.

Indexes allow faster query matching and data retrieval. However, an index also reveals statistical information about the data such as keyword frequency information. This information, correlated with knowledge about the plaintext data, could enable a curious broker to infer with certain probability the encrypted keywords. We further discuss the problem of building secure indexes and measures for assessing the inference exposure of an index. We adopt two attack models, one in which the attacker knows the keyword frequency, and a more powerful one in which the attacker knows both the encrypted and plaintext databases. Our solution has the particularity that it represents numeric values using attributes derived from their binary representation. This representation allows us to solve the complex problem of supporting numeric inequalities. We use an existing inference exposure metric to assess whether this kind of representation increases or decreases the inference exposure of a database as compared to directly indexing the numeric value.

5.1 Encrypted search approaches and their shortcomings

In this section, we present an extensive review of current schemes addressing search on encrypted data. We categorise the existing approaches using two key aspects. The first aspect is related to the key and user management supported by each scheme. We have found the following three categories:

- *single-user* schemes in which only one key is able to write and read (i.e., perform search queries) on the database;
- *semi-fledged multi-user* schemes in which one user is able to write and several users are able to read, or several users are able to write and only one to read;
- *full-fledged multi-user* schemes in which each user has its own key and is able to read and write.

The second aspect is related to the expressiveness of the queries supported by a scheme. We identify three categories: *simple keyword* supporting one equality match performed over a keyword; *conjunction of keywords* that is similar to the previous one but where multiple keywords can be expressed in one single query; and *complex queries* where it is possible to express range conditions, subset operations, conjunctive normal forms (CNF), or disjunctions.

Table 5.1 summarises our categorisation. Most solutions provide keyword search or conjunctions of keywords. None of the surveyed schemes is able to support both multi-users that can read and write to the database, and perform complex queries. Our goal is to fill such a gap.

Table 5.1: Comparison of search on encrypted data schemes.

	Keyword	**Conjunction of keywords**	**Complex queries**
Single user	[Song 2000a] [Goh 2003] [Chang 2005] [Hacigümüş 2002] [Kamara 2012]	[Golle 2004b] [Bösch 2011]	[Wang 2006] [Hore 2004] [Popa 2011] [Hore 2011]
Semi-fledged multi-user	[Boneh 2004] [Curtmola 2006b] [Zhu 2011]	[Baek 2008] [Rhee 2010] [Cao 2011]	[Boneh 2007] [Katz 2008] [Yang 2011] [Li 2011] [Lu 2011]
Full-fledged multi-user	[Bao 2008] [Dong 2008b] [Shao 2010]	[Hwang 2007]	**No solution yet**

In the following, we survey the existing approaches based on the key and user management they support.

5.1.1 Single-user schemes

Song et al. [Song 2000a] are the first to address practical keyword search over encrypted data using symmetric encryption. The user encrypts a document word by word and stores the ciphertext in the cloud. To search for a keyword, the user computes a capability using the secret key and sends it to the server which then tests each word in every document. This scheme is not secure in practice because it reveals statistical information such as the frequency of each word.

To overcome this weakness, Goh [Goh 2003] proposes an efficient secure index construction built using pseudo-random functions and Bloom filters. For each document, a Bloom filter is created from the keywords of the document. To prevent statistical analysis attacks, each Bloom filter is randomised using a unique document identifier. Bosch et al. [Bösch 2011] extend [Goh 2003] with wildcard searches such as *flower, flower*, *lower by inserting wildcardified versions of the words in the index. Conjunctions can be represented as a union of keywords with the disadvantage that the server learns which documents contain the individual words, and not just the result of the query. Moreover, Bloom filters introduce false positives which creates computational overhead for the user.

Chang and Mitzenmacher [Chang 2005] propose a similar solution which builds an index for each document in the form of a vector with an entry for each word in the dictionary. Their solution has better security than [Goh 2003] because it does not reveal the number of words in a document. However, it is less efficient and does not support arbitrary updates with new words, rendering it unsuitable for databases that need to be updated frequently.

Golle et al. [Golle 2004b] propose a scheme allowing multi-keyword search with one encrypted query. The capability is a vector with an entry for each possible keyword. As compared to [Goh 2003, Bösch 2011] this scheme has a better security model because the server learns which documents match the conjunctive query, but does not learn which documents contain the individual keywords. However, like [Chang 2005] this scheme is not practical for large databases with many keywords and arbitrary updates.

Curtmola et al. [Curtmola 2006b] introduce the first symmetric searchable encryption (SSE) scheme that achieves sub-linear search time, which is optimal. The search time is linear in the number of documents that contain a word. They achieve this by creating an index which maps each keyword to the list of documents that contain the word, instead of computing a per document index as [Goh 2003, Chang 2005, Golle 2004b]. They introduce formal security models for SSE and prove their scheme to be secure against adaptive chosen-keyword attacks, a stronger security model. Kamara et al. [Kamara 2012] extend this scheme and introduce a dynamic symmetric searchable encryption (DSSE) scheme that allows adding and deleting documents from the index.

The schemes mentioned above are limited to keyword searches. Range queries such as age>50 are much harder to evaluate in encrypted form than simple keyword searches. Bucketization [Hore 2004, Wang 2006] has been proposed for reducing range queries to equality searches. The main idea is that by splitting the domain of values in several buckets, each range query is transformed in a list of bucket identifiers. The server sends back to the user all the documents contained in the buckets and then the user needs to decrypt the data and discard false positives. Bucketization has several shortcomings. First of all, most of the computations are performed by the user. The user needs to pre-compute the buckets before encrypting the data, and filter out false positives after the query is executed. Computing the optimal bucketization that minimizes the number of false positives is an NP-hard problem and requires knowledge about both the data and query distribution. However, the optimal distribution reveals to the server statistical information about the buckets. To prevent such attacks, buckets should be selected with the same probability, which leads to high false positives rates. Finally, bucketization does not work well when new data needs to be inserted or updated.

Hacigumus et al. [Hacigümüş 2002] propose a solution that achieves confidential SQL queries by using bucketization and a specific protocol for each kind of query which requires several interactions between user and server. They are able to support queries such as "select all employees with salary greater than the average" at the cost of additional computations on the client side and several interactions between user and server.

Hore et al. [Hore 2011] extend bucketization to multi-dimensional data which enables conjunctions of range queries over several numeric attributes. The user builds and stores an index on its side and range queries are transformed in bucket IDs. The bucketization algorithm is very complex and does not scale well when the number of dimensions increases, being more suitable for databases with a small

number of fields. Moreover, the solution is not suitable for dynamic updates and the paper does not discuss re-bucketization.

Another popular method for providing range queries is Order Preserving Encryption (OPE) [Boldyreva 2009]. The main idea is that if $x < y$, then $E(x) < E(y)$, where $E()$ is the encryption function. OPE does not introduce false positives and is very efficient. However, Boldyreva et al. [Boldyreva 2011] showed that because OPE reveals the order relation between ciphertext values, it is not secure for small domains with well-known values and distributions.

Popa et al. [Popa 2011] propose CryptDB, a practical system that provides SQL database confidentiality. The system relies on a trusted proxy server that intercepts user queries to the protected database. The proxy holds a secret master key and encrypts and decrypts data and queries. For matching keywords, CryptDB uses an efficient implementation of [Song 2000a] and range queries are provided using OPE [Boldyreva 2009]. Simple computations on numeric data such as computing the mean are achieved using an implementation of homomorphic encryption based on the Paillier cryptosystem [Paillier 1999]. However, maintaining and securing the proxy server may not be feasible for many companies who choose cloud computing as a way of simplifying operations and reducing costs. Moreover, the methods used for providing keyword search and range queries have been shown not to provide sufficient security.

5.1.2 Semi-fledged multi-user schemes

All the above schemes are based on symmetric encryption. The first public-key encryption scheme with keyword search (PEKS) was proposed by Boneh et al. [Boneh 2004]. Any user possessing the public key can encrypt a message and only the owner of the private key can generate keyword search capabilities or trapdoors. However, the trapdoor encryption scheme is vulnerable to inference attacks. Subsequently, [Baek 2008, Rhee 2010, Zhu 2011] improved the security of the scheme and [Baek 2008] also introduced conjunctions of keywords. Public key encryption is very computationally expensive which makes these schemes too inefficient for large databases. Moreover, because only one user can read, these schemes do not fit the one-to-many pub/sub model, being more suitable for applications such as selective email forwarding in which only the receiver of the messages can generate search trapdoors, as [Boneh 2004] proposed.

Yang et al. [Yang 2011] propose a solution in which the database owner (DO) encrypts the data and assigns to each user a unique key for searching and reading the data. The main idea is that the DO splits the master secret uniquely between each user and the server. So for each user, the server holds a corresponding secret key which is used to re-encrypt the user's search queries. To revoke a user, the DO instructs the sever to delete this key. A major drawback of this method is that the search operation is inefficient because it requires an expensive pairing computation on elliptic curves. The authors also propose a rudimentary access control mechanism which does not allow revocation and requires users to share a secret per authorised

keyword, making the scheme single-user. This secret is then used as trapdoor. A scheme for conjunctions is also proposed but it is highly inefficient because an index needs to be created for every conjunction of words per document which increases exponentially with the number of words.

A number of papers use predicate encryption to support encrypted search because it can achieve more complex queries. In a predicate encryption scheme, the secret key corresponds to a predicate and the ciphertext is associated with a list of attributes. A key decrypts a ciphertext if the associated attributes satisfy the key predicate. Boneh and Waters [Boneh 2007] propose a public key-based predicate encryption scheme that supports conjunctions, range queries (such as less-than and greater than), and subset queries on encrypted data. Katz et al. [Katz 2008] extend the scheme from [Boneh 2007] and propose a predicate encryption for inner products scheme which supports conjunctions, disjunctions, conjunctive normal forms (CNF) and disjunctive normal forms (DNF). The above solutions are based on public key encryption and do not provide predicate (or query) privacy because the server can encrypt any plaintext with the public key and test the query. Shen et al. [Shen 2009] propose a symmetric key based predicated encryption scheme which achieves predicate privacy. Though predicate encryption schemes can support more complex queries, they are inefficient because they require expensive evaluations of pairing operations on elliptic curves. Such schemes are much less inefficient than searchable symmetric encryption and no concrete implementation exists.

Li et al. [Li 2011] propose a solution for searching on encrypted health care records which supports a special type of CNF formula, where conjunctions are across multiple attributes while disjunctions refer to the same attribute. Queries over predefined numeric attribute ranges can be supported through a hierarchical bucketization of the domain. The solution is based on hidden vector encryption (HVE) and uses multiple trusted authorities to distribute search capabilities to users, thus achieving fine-grained access control over the stored data. User revocation is handled by adding time information to the index and to the capability of each user such that a user cannot search after its capability expired. Because HVE requires a large number of exponentiations and pairings, all the operations are very slow and the scheme is only suitable for searching on a small personal record, and not on a large database.

Cao et al. [Cao 2011] propose multi-keyword ranked search over encrypted data. The user creates a trapdoor for a set of keywords and is given back documents ranked by the number of keywords they match. The scheme ensures data and query privacy. The database owner (DO) creates an initial index using a secret key. The index encrypts an $n \times m$ matrix where n is the number of documents and m is the number of words in the dictionary. Building the index is very expensive and the index cannot be changed dynamically.

Lu and Tsudik [Lu 2011] propose a solution similar to ours which relies on attribute-based encryption (ABE) [Goyal 2006b] and blind Boneh-Boyen weak signature scheme [Belenkiy 2009]. However, in their solution, only the DO is able to encrypt the data and it needs to be online to help authorised users extract search

tokens and decryption keys. In our solution, each user can encrypt and query the data with its own key, without the need of maintaining an online DO. Another disadvantage is that their solution only supports conjunctions and disjunctions of equalities as in the scheme of [Goyal 2006b]. Though [Bethencourt 2007] showed how to extend the access structure of [Goyal 2006b] to additionally support inequalities, the authors did not implement this feature, while our scheme supports it.

5.1.3 Full-fledged multi-user schemes

Hwang et al. [Hwang 2007] extend PEKS to multi-user settings and conjunctive keyword search. To encrypt a message that can be read by n users, the sender needs the public keys of all the users. The paper introduces a new multi-receiver encryption scheme based on ElGamal and pairings which allows the user to encrypt the plaintext only once and include in the ciphertext n trapdoors obtained from the public keys of each receiver. Each of the n users can generate a trapdoor for a conjunctive keyword query using its private key. Both the encryption and the test algorithm performed by the server are inefficient because they use pairing operations. Moreover, ciphertext size is large and grows linearly with the number of users. Adding new users to the system requires re-encrypting all the data.

Bao et al. [Bao 2008] propose a multi-user solution for keyword searches on encrypted databases in which each user has its own key for writing and reading. The solution is based on proxy encryption and bilinear maps. Index generation is an interactive algorithm run between user and server. The user of bilinear maps and the interactive encryption algorithm make this scheme inefficient. Dong et al. [Dong 2008b] propose searchable data encryption (SDE), a similar multi-user scheme which supports keyword search. SDE is based on proxy encryption and does not require interactive protocols or pairing. As a result, SDE encryption and search operations are much more efficient.

Shao et al. [Shao 2010] introduce Proxy Re-Encryption with keyword Search (PRES), a combination of proxy re-encryption and PEKS. The multi-user property is given by the fact that the server is able to repeatedly transform a ciphertext encrypted by a user's key into a ciphertext that can be decrypted by another user's key. This scheme only supports keyword match and the use of public key encryption for computing the ciphertext and of pairing computations for testing keywords makes it inefficient. In fact, from the full-fledged multi-user schemes, the only one that has been implemented and proven to be efficient in practice is the solution of Dong et al. [Dong 2008b]. That is why we chose to extend it in our scheme.

To conclude the related work, we notice that there is a gap in literature because no solution can provide both full-fledged multi-user support and complex queries. Our solution is the first one to provide such properties. Moreover, by leveraging ABE our solution can also support fine-grain access control policies, and by combining ABE policies with SDE, it can also provide policy confidentiality.

5.2 Solution description

In the following we assume that one or several designated brokers enhanced with database functionality, store events published over a longer period of time. Such brokers could store all the events, or just certain event types with particular attributes by becoming a subscriber and registering specific filters. These brokers do not need to know the details of the filter, instead they could receive an encrypted filter from a trusted authority and register it. For example, in the e-Health application scenario, the hospital or the Healthcare authority, or even the patient, can decide what kind of information they want to store and send an encrypted filter to a broker.

5.2.1 Event encryption and indexing

Events encrypted using either the basic confidentiality scheme from Chapter 3 or the enhanced scheme from Chapter 4 have the form $E = (\{TD(a)\}_{a\in\gamma}, c(M))$, where $TD(a) = (g^x)^{f_s(a)}$ with f a pseudorandom function, s a secret number, and x the master secret key stored by the trusted authority, and $c(M)$ is the ciphertext encrypted either with PE or ABE. An attribute a has the form $attr\_name = value$. The broker who does not have the keys for decrypting the message content, stores the events in a database. Table 5.2 shows an example of encrypted events stored by a broker.

Table 5.2: Events index.

Event ID	Trapdoors	Encrypted content
E_1	$TD(a_1)$, $TD(a_2)$	$c(M_1)$
E_2	$TD(a_3)$, $TD(a_2)$	$c(M_2)$
E_3	$TD(a_4)$, $TD(a_1)$	$c(M_3)$
...	...	...

To retrieve the events, users make queries in the form of an encrypted access tree policy as explained in the Filter Encryption algorithm in Section 3.5.5. To speed up how long it takes to identify the events that contain certain attributes, the broker creates an index that maps each trapdoor to the events that contain it as shown in Table 5.3.

Table 5.3: Trapdoor index.

Trapdoor	Event ID
$TD(a_1)$	E_1, E_3
$TD(a_2)$	E_1, E_2
$TD(a_3)$	E_2
$TD(a_4)$	E_3
...	...

5.2.2 Query encryption

The query has the structure of a subscription filter with internal nodes representing AND and OR relations, and leaf nodes representing attributes. Previously, we encrypted the leaf nodes of the tree with the keyword encryption (KE) algorithm of SDE, a proxy-based probabilistic algorithm that requires re-encryption by the local broker. Using a probabilistic algorithm to encrypt the leaf nodes of the tree ensured that the server could not distinguish between leaves encrypting the same attribute. This prevented the server from learning statistical information from the encrypted filters it stored, such as the number of distinct attributes and the number of occurrences of each. In the following we take a different approach which allows evaluating faster encrypted queries on the events stored in the database. We encrypt the leaf nodes of the tree as trapdoors using the Trap–U algorithm (Algorithm 11) on the user side and Trap–S (Algorithm 12) on the local broker's side. Figure 5.1 shows the modified query encryption algorithm which in steps (2) and (3) uses the trapdoor algorithms. Figure 5.2 shows an example of a tree with leaf nodes encrypted as trapdoors.

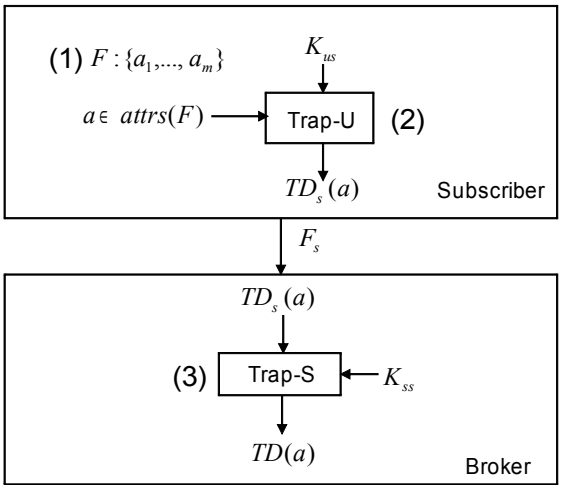


Figure 5.1: Query encryption as an access tree using the trapdoor algorithm.

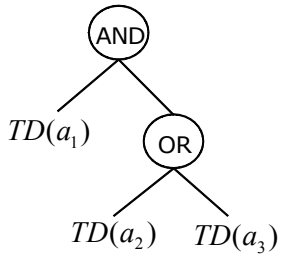


Figure 5.2: Query encrypted using the trapdoor algorithm.

5.2.3 Event matching

When a new query comes in the form of an access tree, the broker identifies and retrieves all the events that match the query. The broker uses two main algorithms for that. First, it identifies the events that could potentially match the query, thus discarding all the events that cannot be matched, as shown in Algorithm 28. The broker retrieves all the events that contain in their set of trapdoors $\{TD(a)\}_{a\in\gamma}$ one of the trapdoors in the filter, where γ is the set of attributes under which the event was encrypted. The broker then tries to match the filter against each of the events using Algorithm 25.

Algorithm 24 Encrypted Event Filtering

Input: A trapdoor index $TrapIdx$, an event index $EvIdx$, a re-encrypted query represented as an access tree F^*.

Output: a list E of event IDs that match the query F^*.

```
1: initialize a list of event IDs E
2: for all re-encrypted leaf nodes l ∈ F* do
3:    if l belongs to TrapIdx then
4:       for all event IDs e in TrapIdx(l) do
5:          add e to E
6:       end for
7:    end if
8: end for
9: for all event IDs e ∈ E do
10:   retrieve the set A of trapdoors of e
11:   if iTreeEval(F*.root, A) is false then
12:      remove e from E
13:   end if
14: end for
15: return E.
```

Algorithm 25 iTreeEval: Access Tree Evaluation with Index

Input: A node x of a re-encrypted tree F^*, and the set of trapdoors $A = \{TD(a)\}_{a\in\gamma}$ of the re-encrypted event E.
Output: *true* or *false*.

```
1: if x is a leaf node then
2:   if attr(x) belongs to {TD(a)}_{a∈γ} then
3:     return true
4:   end if
5: else
6:   l = 0
7:   while l < threshold(x) do
8:     for all children c of x do
9:       if iTreeEval(c, {TD(a)}_{a∈γ}) then
10:        l++
11:      end if
12:    end for
13:  end while
14:  if l = threshold(x) then
15:    return true
16:  end if
17: end if
18: return false
```

We note that the iTreeEval algorithm is more efficient than the TreeEval from Chapter 3, described in Algorithm 14 because it does not make any calls to the SDE–Match algorithm of SDE shown in Algorithm 13.

5.3 Inference exposure

Building an index over encrypted data allows faster query evaluation, increasing the efficiency of the system. For example, when a user searches for documents or records containing a particular keyword and the data is not indexed, the server needs to test each document one by one. However, if documents are indexed under the keywords they contain, the server can easily locate and retrieve only the documents containing the word without processing the other documents or records.

The main concern when building an index is that it can reveal sensitive information about the data like the number of documents containing each word. If the server knows the kind of data that is being indexed and the frequency of plaintext values, it might be able to infer with a certain non-negligible probability, the correspondence between known plaintext and encrypted values. Several techniques can be employed to flatten the work frequency like using hash functions with collisions that map specific words to the same index value. Another solution could be to insert fake data in the database. Such methods have performance and computation costs, as they create false positives. In order to decide if such a method is needed, we must first be able to assess the inference exposure of the encrypted dataset.

Figure 5.3: Plaintext data and indexed data using direct encryption.

Account	Customer	Balance
Acc1	Alice	100
Acc2	Alice	200
Acc3	Bob	300
Acc4	Chris	200
Acc5	Donna	400
Acc6	Elvis	200
Acc7	Fred	300

Enc_tuple	I_A	I_C	I_B
ri4uUIeuhje4	π	α	μ
J3oiu4y3j0h8	ϖ	α	κ
45hFjm/woier	ξ	β	η
Y43u89jkre4u	ϱ	γ	κ
KJTi34u928rf	ς	δ	θ
KJFp9ieu34ju	Γ	ε	κ
IK39ru209ukj	τ	ϕ	η

The exposure is specific to each dataset and depends on the data distribution. The acceptable inference exposure degree is specific to each application or even to each attribute. In the following we will model exposure as the probability for the server to correlate ciphertext values with plaintext values. Our goal is not to propose another protection method, but instead to evaluate the exposure of data encrypted with our scheme in order to determine when protection mechanisms are needed. In particular, we represent numeric values on bits and create a different attribute for each bit position, thus we want to analyse what is leaked by such a representation, called a "bag of bits" in [Bethencourt 2007]. For example, the "bag of bits" representation of a=7 is "a=0***, a=*1**, a=**1*, a=***1". To reduce the exposure to the desired level, we can apply the collision method from [Ceselli 2005] and choose the pseudorandom function f used in our scheme to be a hashing function with collision. No other modifications are required to our scheme, except for choosing the right f function depending on the distribution of the data that will be encrypted.

We start by explaining current inference exposure metrics for encrypted databases, and then assess the inference of our scheme for different threat models and indexing approaches.

5.3.1 Background

The problem of inference exposure in encrypted indexes was analysed by [Ceselli 2005]. They assume that the data is organized in a table as shown on the left side of the table of Figure 5.3. Furthermore, their work assumes that for each database entry, the whole row is encrypted using some encryption scheme independent of the index, and that for each indexable field, the value is encrypted using some indexing function. We note that our event encryption scheme uses the same principle. The event content is encrypted using PE or ABE, while the attributes that are used by the broker to match events against filters are encrypted with the trapdoor algorithm from SDE. In our case, the trapdoor algorithm is the indexing function. [Ceselli 2005] consider two cases for the indexing function: direct encryption as shown on the right side table of Figure 5.3 which preserves word frequency, and a hashing function with collision which modifies the work frequency, and thus provides protection from inference attacks as shown in Table 5.4. We note that this solution does not support complex queries with numeric inequalities as it is not possible for the server to compare index values.

Table 5.4: Indexed data using a hash function with collision.

Enc_tuple	$\mathbf{I_A}$	$\mathbf{I_C}$	$\mathbf{I_B}$
ri4uUIeuhje4	π	α	μ
J3oiu4y3j0h8	ϖ	α	κ
45hFjm/woier	ξ	δ	θ
Y43u89jkre4u	ϱ	α	κ
KJTi34u928rf	ς	β	κ
KJFp9ieu34ju	Γ	β	κ
IK39ru209ukj	τ	δ	θ

We call this kind of index that organizes data in a table, a *two-dimensional index* because values are indexed under the attribute name, and then records are indexed under the values they contain. Our event encryption scheme, on the other hand, uses a *one-dimensional index*, as *attr_name* = *value* becomes an index value by itself and is not indexed under the attribute name. The correspondence between attribute name and attribute values is hidden from the broker in our scheme. We will analyse and compare the exposure of using both a one and two-dimensional index. Our scheme can be easily turned into a two-dimensional scheme by attaching to each *attr_name* = *value* pair the encrypted name of the attribute.

In the following we analyse and compare the exposure of our encryption scheme when using a one and two-dimensional index. We are interested in assessing how representing numeric values as a "bag of bits" affects the exposure of the index. We will consider the same two threat models introduced in [Ceselli 2005]. In the first one called $\mathsf{Freq} + \mathsf{DB}^{\mathsf{K}}$, the server knows the frequency of each word and has access to the entire encrypted database. In the second one, called $\mathsf{DB} + \mathsf{DB}^{\mathsf{K}}$, the server knows both the encrypted and non-encrypted database.

5.3.2 Threat model 1: $\mathsf{Freq} + \mathsf{DB}^{\mathsf{K}}$

This threat model assumes the attacker knows the frequency of each attribute value and has access to the entire encrypted database. The frequency information could be approximate or exact. It is probably unlikely that the broker can have the exact distribution, especially because the distribution changes over time. We will assume the worst-case scenario in which the broker has exact knowledge of the plaintext data frequency.

The main idea in modelling inference exposure is that values with the same number of occurrences become indistinguishable to the server. We follow [Ceselli 2005] and group values with the same number of occurrences in *equivalence classes* as shown below. A.1 is the class of values from column A that appear once and so forth. The probability of guessing a single value is 1 over the number of values in the class. So the probability of guessing any value from class A.1 is 1/7.

A.1 = $\{\pi, \varpi, \xi, \varrho, \varsigma, \Gamma, \tau\}$ = {Acc1, Acc2, Acc3, Acc4, Acc5, Acc6, Acc7}
C.1 = $\{\beta, \gamma, \delta, \varepsilon, \phi\}$ = {Bob, Chris, Donna, Elvis, Fred}
C.2 = $\{\alpha\}$ = {Alice}
B.1 = $\{\mu, \theta\}$ = {100, 400}

Figure 5.4: Quotient and IC tables.

$\mathbf{Qt_A}$	$\mathbf{Qt_C}$	$\mathbf{Qt_B}$
A.1	C.2	B.1
A.1	C.2	B.3
A.1	C.1	B.2
A.1	C.1	B.3
A.1	C.1	B.1
A.1	C.1	B.3
A.1	C.1	B.2

$\mathbf{IC_A}$	$\mathbf{IC_C}$	$\mathbf{IC_B}$
1/7	1	1/2
1/7	1	1
1/7	1/5	1
1/7	1/5	1
1/7	1/5	1/2
1/7	1/5	1
1/7	1/5	1

B.2 = $\{\eta\}$ = {300}
B.3 = $\{\kappa\}$ = {200}

From the equivalence classes, we can compute the *quotient table* which is obtained by replacing each value in the index with the class to which it belongs, as shown in Figure 5.4. The *inverse coefficient (IC) table* is obtained by replacing each value with 1 over the size of the class to which it belongs. The IC is the probability of guessing the value if the server only knows frequency information. The probability of guessing an entire row is the product of the IC of each value. For example, the probability of guessing the first row is 1/14. The exposure coefficient ε associated with the entire encrypted table can be computed as the average probability of guessing each row using the formula proposed by [Ceselli 2005]:

$$\varepsilon = \frac{1}{n} \sum_{i=1}^{n} \prod_{j=1}^{k} IC_{i,j} \tag{5.1}$$

The exposure coefficient for the table, computed using Equation 5.1 is $\varepsilon = \frac{1}{7} \cdot \frac{12}{35}$ = 0.049, which means that an attacker can guess the entire table with probability 4.9%.

5.3.2.1 Inference of the 2-dimensional index

We now analyse how representing numeric values as a "bag of bits" impacts the exposure coefficient when the two-dimensional indexing is maintained. In this case, the plaintext database is as shown in Table 5.5.

We assume the server knows exactly which values correspond to each bit position. For the above example, the IC table will be as shown in Table 5.6, where B1 refers to the first bit of the Balance field starting from the most significant bit, B2 to the second bit and so on.

The IC table is show in Table 5.7 and the exposure coefficient computed using Equation 5.1 is $\varepsilon = \frac{1}{7} \cdot \frac{3}{7} = 0.061$, greater than the exposure for Figure 5.4, while for the numeric field Balance the exposure coefficient is 1. Because there are only two possible values for a bit, either 0 or 1, the server can guess the value with probability 1, unless the number of occurrences for 1 and 0 bit values is exactly the same. We conclude from this result that a 2-dimensional index as the one above would not be secure.

Table 5.5: Plaintext database with "bag of bits" representation of numeric values.

Account	Customer	Balance
Acc1	Alice	B=0******** B=*0******* B=**1****** B=***1***** B=****0**** B=*****0*** B=******1** B=*******0* B=********0
Acc2	Alice	B=0******** B=*1******* B=**1****** B=***0***** B=****0**** B=*****1*** B=******0** B=*******0* B=********0
Acc3	Bob	B=1******** B=*0******* B=**0****** B=***1***** B=****0**** B=*****1*** B=******1** B=*******0* B=********0
Acc4	Chris	B=0******** B=*1******* B=**1****** B=***0***** B=****0**** B=*****1*** B=******0** B=*******0* B=********0
Acc5	Donna	B=1******** B=*1******* B=**0****** B=***0***** B=****1**** B=*****0*** B=******0** B=*******0* B=********0
Acc6	Elvis	B=0******** B=*1******* B=**1****** B=***0***** B=****0**** B=*****1*** B=******0** B=*******0* B=********0
Acc7	Fred	B=1******** B=*0******* B=**0****** B=***1***** B=****0**** B=*****1*** B=******1** B=*******0* B=********0

Table 5.6: Quotient table - 2D index.

Qt_A	Qt_C	Qt_{B1}	Qt_{B2}	Qt_{B3}	Qt_{B4}	Qt_{B5}	Qt_{B6}	Qt_{B7}	Qt_{B8}	Qt_{B9}
A.1	C.2	B1.4	B2.3	B3.4	B4.3	B5.6	B6.2	B7.3	B8.7	B9.7
A.1	C.2	B1.4	B2.4	B3.4	B4.4	B5.6	B6.5	B7.4	B8.7	B9.7
A.1	C.1	B1.3	B2.3	B3.3	B4.3	B5.6	B6.5	B7.3	B8.7	B9.7
A.1	C.1	B1.4	B2.4	B3.4	B4.4	B5.6	B6.5	B7.4	B8.7	B9.7
A.1	C.1	B1.3	B2.4	B3.3	B4.4	B5.1	B6.2	B7.4	B8.7	B9.7
A.1	C.1	B1.4	B2.4	B3.4	B4.4	B5.6	B6.5	B7.4	B8.7	B9.7
A.1	C.1	B1.3	B2.3	B3.3	B4.3	B5.6	B6.5	B7.3	B8.7	B9.7

Table 5.7: IC table - 2D index.

IC_A	IC_C	IC_{B1}	IC_{B2}	IC_{B3}	IC_{B4}	IC_{B5}	IC_{B6}	IC_{B7}	IC_{B8}	IC_{B9}
1/7	1	1	1	1	1	1	1	1	1	1
1/7	1	1	1	1	1	1	1	1	1	1
1/7	1/5	1	1	1	1	1	1	1	1	1
1/7	1/5	1	1	1	1	1	1	1	1	1
1/7	1/5	1	1	1	1	1	1	1	1	1
1/7	1/5	1	1	1	1	1	1	1	1	1
1/7	1/5	1	1	1	1	1	1	1	1	1

5.3.2.2 Inference of the 1-dimensional index

We now assume that rows are encrypted as in our event encryption scheme. For the example we considered above, the broker stores 7 events that were encrypted under the attributes *Account=Acc1, Customer=Alice, Balance=100* and so on. The attributes are encrypted using the trapdoor algorithm, with the attribute name and value concatenated and in random order such that the broker cannot identify which values refer to the same attribute. We use the same principle that values with the same number of occurrences are indistinguishable to the broker and compute the following equivalence classes for the whole table as shown in Table 5.8.

Table 5.8: Quotient table - 1D index.

T.1	T.2	T.4	T.3	T.4	T.3	T.6	T.2	T.3	T.7	T.7
T.1	T.2	T.4	T.4	T.4	T.4	T.6	T.5	T.4	T.7	T.7
T.1	T.1	T.3	T.3	T.3	T.3	T.6	T.5	T.3	T.7	T.7
T.1	T.1	T.4	T.4	T.4	T.4	T.6	T.5	T.4	T.7	T.7
T.1	T.1	T.3	T.3	T.3	T.4	T.1	T.2	T.4	T.7	T.7
T.1	T.1	T.4	T.4	T.4	T.4	T.6	T.5	T.4	T.7	T.7
T.1	T.1	T.3	T.3	T.3	T.3	T.6	T.5	T.3	T.7	T.7

Table 5.9: IC table - 1D index.

1/13	1/2	1/5	1/5	1/5	1/5	1	1/2	1/5	1/2	1/2
1/13	1/2	1/5	1/5	1/5	1/5	1	1	1/5	1/2	1/2
1/13	1/13	1/5	1/5	1/5	1/5	1	1	1/5	1/2	1/2
1/13	1/13	1/5	1/5	1/5	1/5	1	1	1/5	1/2	1/2
1/13	1/13	1/5	1/5	1/5	1/5	1/13	1/2	1/5	1/2	1/2
1/13	1/13	1/5	1/5	1/5	1/5	1	1	1/5	1/2	1/2
1/13	1/13	1/5	1/5	1/5	1/5	1	1	1/2	1/2	1/2

We note that in the case of a 1-dimensional index, the equivalence classes become much bigger as they are computed over the entire table instead of just over one row. The equivalence classes become even bigger when the server stores different event types. Table 5.9 shows the IC table in this case. The average exposure coefficient of the table is $\varepsilon = 9.32440341e-7$ much smaller than in the case of the 2 dimensional index. A one-dimensional (1D) index would be secure under the $\mathsf{Freq} + \mathsf{DB}^{\mathsf{K}}$ threat model, more secure than a two-dimensional (2D) index because equivalence classes are equal or bigger than equivalence classes for the 2D index, but never smaller. On the other hand, for large datasets, the "bag of bits" representation reduces the number of distinct values in the database, as each numeric field will be mapped to n "buckets", where n is the number of bits on which the number is represented. Even so, the exposure under this threat model is small as we will show in the following for different datasets.

5.3.2.3 Inference comparison on synthetic datasets

We generate several datasets with various number of fields, following different distributions and compute the inference exposure for each. To generate the data we

used the Commons Math 3.1.1 library[1] which implements various distributions.

Event type 1 contains 1 non-numeric attribute (or field) following a Zipf distribution, and a numeric attribute on 10 bits following a uniform distribution. Figure 5.5 shows the exposure of datasets of various sizes. As we expected, the 2D index with "bag of bits" representation (BR_2D) is not secure because it gives much bigger exposure values than the 1D index with "bag of bits" representation (BR_1D) and the direct encryption methods DE_1D and DE_2D. Because of that we will omit it in the next figures.

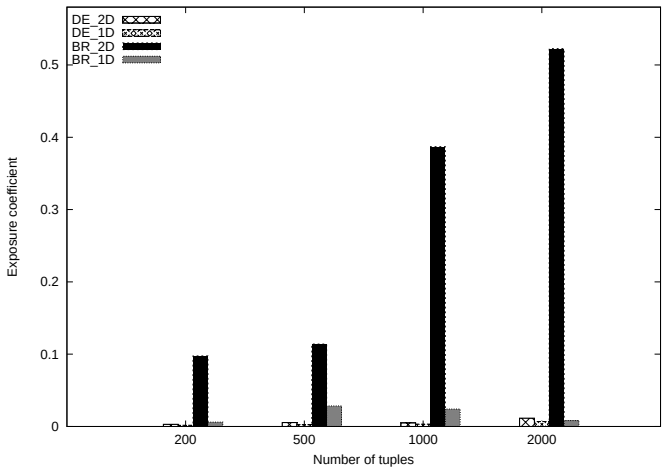


Figure 5.5: Inference exposure of the type 1 event.

Event type 2 additionally contains a numeric field following a Zipf distribution, so it has one non-numeric and 2 numeric attributes. The numeric attributes are represented on 10 bits. The exposure coefficients for different data sizes, for direct encryption using one-dimensional index (DE_1D) and two-dimensional index (DE_2D) and for "bag of bits" representation one-dimensional index (BR_1D) are shown in Figure 5.6. The inference exposure for this event is much smaller than for event type 1, under 0.003 (i.e., 0.3%). Under this threat model, the more attributes an event has, the smaller the probability that an attacker can infer the table.

[1] http://commons.apache.org/proper/commons-math/

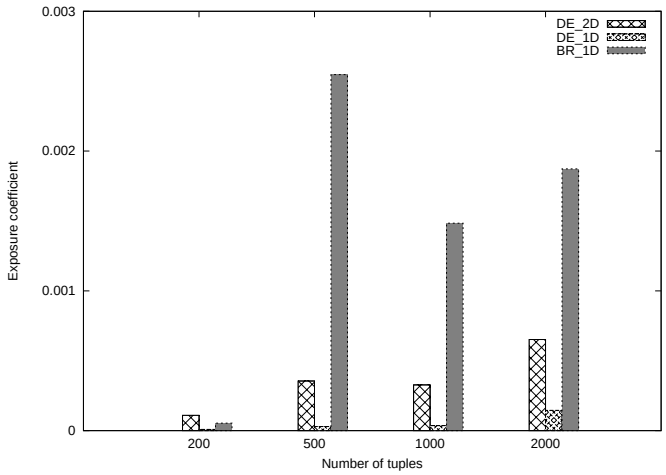


Figure 5.6: Inference exposure of the type 2 event.

In conclusion, representing numeric values as a "bag of bits" slightly increases the inference exposure for large datasets, but the exposure still remains very low. In the following we consider a more powerful threat model.

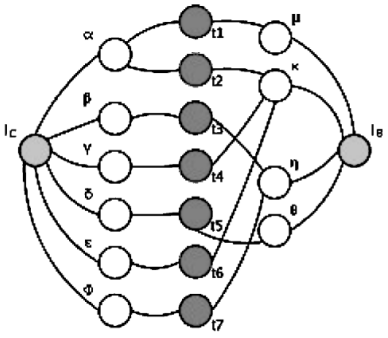
5.3.3 Threat model 2: $\mathsf{DB} + \mathsf{DB}^{\mathsf{K}}$

In the second model, $\mathsf{DB} + \mathsf{DB}^{\mathsf{K}}$, the attacker additionally knows the frequency of correlations between values. We assume that the attacker knows the exact unencrypted DB and has access to the encrypted DB. Using this knowledge, the attacker tries to match plaintext with ciphertext values. [Ceselli 2005] proposed assessing the inference exposure in this case by constructing a row-column-value (RCV) graph from the encrypted table which has a vertex for each (i) attribute name (of color column), (ii) distinct value in a row (of color value), and (iii) row or association (of color row). Figure 5.7 shows the RCV graph for the Accounting example, using only two indices, I_C and I_B.

Enc_Accounts1

Enc_tuple	I_A	I_C	I_B
x4Z3tfX2ShOSM	π	α	μ
mNHg1oC010p8w	ϖ	α	κ
WsIaCvfyF1Dxw	ξ	β	η
JpO8eLTVgwV1E	ϱ	γ	κ
qctG6XnFNDTQc	ς	δ	θ
4QbqC3hxZHklU	Γ	ϵ	κ
3jO9DfoaU78Hs	τ	ϕ	η

(a)



(b)

Figure 5.7: Encrypted table (a) and the corresponding RCV graph (b) from [Ceselli 2005].

Based on the observation that the plaintext and encrypted RCV graph have the same structure, Ceselli et al. propose an algorithm for determining the probability of guessing each vertex by counting the number of ways in which vertices can be associated to create isomorphic graphs. This can be achieved by computing the *equitable partition* of the graph following the algorithm of McKay [McKay 1981]. This algorithm groups the vertices of a graph in sets where each set C_j contains vertices that can be substituted one for the other in an automorphism. For the graph given above, the equitable partition of attribute vertices is $\{(\alpha)(\beta,\varphi)(\gamma,\varepsilon)(\xi)(\mu)(\eta)(\kappa)(\theta)\}$.

The exposure of the whole table can be computed as the probability of guessing each value divided by the total number of values as shown in Equation 5.2, where m is the total number of values and n is the total number of partitions.

$$\varepsilon = \sum_{i=1}^{m} \frac{p_i}{n} = \sum_{j=1}^{n} \sum_{v_i \in C_j} \frac{1}{|C_j| m} = \sum_{j=1}^{n} \frac{1}{m} = \frac{n}{m}. \tag{5.2}$$

We will use this method to analyse the exposure of our indexing algorithm which represents numeric values as "bag of bits". We use the C implementation of the algorithm of McKay from [McKay] to generate the equitable partition. For the example with columns Customer and Balance, the exposure of the 2D direct encryption is 8/10, as there are 8 partitions and 10 distinct values. For 1D direct encryption the graph does not contain the column vertices, having only two kind of colors. The exposure in this case is smaller and is equal to 7/10. For the 1D "bag of bits" representation we use, the exposure is smaller and is equal to 11/22. In this case, representing the values as "bag of bits" reduces the exposure because the dataset is small and the "bag of bits" representation increases the number of vertices.

However, in general, the more fields or attributes the database has, the more constrained the RCV graph becomes, leading to big inference coefficients. Ceselli et al. showed that if for a database with 2 fields and 2,000 tuples, the inference exposure is 18/2006, for 4 tuples the inference becomes 1467/2262. Because we represent numeric values as "bag of bits", we create more fields, and so the exposure increases. Our experiments confirmed this observation. For large datasets containing only numeric attributes we obtained exposure coefficients of 1, consistent with the values reported by Ceselli et al.

For databases with 4 attributes that give exposure coefficients of 1, a hash function with collision factor of 3, meaning that in average three different plaintext values are hashed to the same value, decreased the exposure coefficient from 1 to 0.3-0.4.

Chapter 6

Efficient Encrypted Routing

Contents

6.1 Introduction

Pub/sub systems are suitable for large-scale or time-constrained applications such as stock quote dissemination, security alerts, and location-based services. In such applications, in order to deliver messages in a timely manner, brokers need to efficiently filter events against a large number of registered subscriptions (or filters). Matching an event against each subscription does not scale up when the number of subscriptions increases. Building more efficient filtering solutions is a problem well researched in literature [Carzaniga 2003, Whang 2009, Bittner 2005]. However, in order to support the desired efficiency, most systems [Carzaniga 2003, Mühl 2001, Li 2005] restrict the subscription language to conjunctions of predicates which are faster to evaluate than general Boolean expressions. Other systems transform filters expressed as general Boolean expressions in several filters that represent conjunctive-normal forms (CNF) or disjunctive-normal forms (DNF) [Whang 2009]. This approach, however, has been shown not to be efficient because it significantly increases the number of filters [Bittner 2005]. Only recently, more

efficient algorithms have been proposed for matching filters expressed as general Boolean expressions [Bittner 2005, Fontoura 2010].

Many applications such as stock quote dissemination, online auctions, or eHealth applications require solutions for preserving the confidentiality of both events and subscriptions, but at the same time, they also require an efficient encrypted filtering algorithm that does not significantly impact the scalability of the system. Existing solutions for preserving confidentiality focus mainly on the security aspects and neglect scalability. They usually require matching an incoming event against all subscriptions one by one, because subscriptions are not indexed [Choi 2010, Chen 2010]. If indexes are supported, the expressiveness of the filter is reduced to single keyword matches [Srivatsa 2005, Shikfa 2009]. None of the solutions we surveyed addressing confidentiality is able to efficiently filter encrypted events against complex encrypted filters such as general Boolean expressions. In this chapter we propose a novel solution for an efficient and scalable filtering algorithm, while maintaining the properties of our confidentiality scheme, i.e., confidentiality of events and filters, complex encrypted filters able to express general Boolean expressions, and scalable key management that does not require publishers and subscribers to share keys.

6.2 Background

A pub/sub network consists of a number of *brokers* (or routers) connected in a specific topology. Most research pub/sub prototypes targeting efficiency, such as SIENA [Carzaniga 2001], PADRES [Li 2005], REBECA [Mühl 2001], XRoute [Chand 2003], and GRYPHON [Banavar 1999] assume that brokers form a fixed acyclic graph. The pub/sub network allows publishers to send messages to interested subscribers without having to discover each other or even establish direct contact.

Messages sent by publishers consist of several attribute-value pairs. For example, in a stock quote dissemination application, an event could have the form: "SYMBOL=MSF, PRICE=30, QUANTITY=100".

In order to receive events, subscribers need to register a subscription or filter with a local broker. The most expressive filters are defined as general Boolean expressions containing conjunctions and disjunctions of predicates. A *predicate* defines a constraint on an event attribute. Constraints are defined using operators such as $=$, $<$, $>$, $\leq$, and $\geq$. For example, a subscriber who registered the filter "SYMBOL=MSFT and PRICE>25" would receive the above mentioned event.

For each incoming event, a broker needs to determine all subscriptions whose Boolean expression is matched by the event. This process is called event filtering and can be defined as follows.

Definition 16 (Event filtering)**.** *A broker of the pub/sub system with a registered set of subscriptions S, given an incoming event message E, needs to find every subscription $s \in S$ that is matched by the attributes of E.*

Brokers perform event filtering in order to deliver events from a publisher to all subscribers who registered a filter matched by the event. This process is called event routing and can be defined as follows.

Definition 17 (Event routing)**.** *The pub/sub system consisting of a set of distributed brokers connected in a specific topology, is given an incoming event E and needs to determine all brokers connected to subscribers that registered a filter matched by the event.*

In the following we briefly survey how current pub/sub systems address the problems of efficient event filtering and routing defined above.

6.2.1 Event filtering algorithms

In this section we analyse current event filtering algorithms in pub/sub systems. We are interested in two aspects: (i) the complexity of the subscription language, and (ii) the indexing strategy employed to support fast filtering.

Subscription languages in current solutions can either support: general filters, be limited to some Boolean expressions such as disjunctive normal forms (DNF) and conjunctive normal forms (CNF), or just conjunctions of predicates.

Current solutions use the following indexing approaches: no indexing at all, in which case subscriptions are tested one by one, predicate indexes on a per-attribute basis (all predicates on the same attribute are indexed under the attribute name), subscription indexes which compact subscriptions together, or both predicate and subscription indexes. Table 6.1 classifies the surveyed solutions based on these criteria.

Index \ Sub	General	DNF & CNF	DNF	Conjunctions
No indexing	[Segall 2000]	-	-	-
Predicate	[Bittner 2005]	[Whang 2009] [Fontoura 2010]	[Carzaniga 2003]	[Ashayer 2002]
Subscription	[Campailla 2001]	-	-	-
Sub & Pred	-	-	-	[Li 2005]

Table 6.1: Event Filtering Algorithms

Among current pub/sub systems, Elvin [Segall 2000] supports a general Boolean subscription language and sophisticated predicates including regular expression matching for strings. However, Elvin does not index predicates or subscriptions, instead each event is evaluated against each subscription, making Elvin unsuitable for large applications.

The approach from [Campailla 2001] also supports general Boolean expressions. It represents subscriptions using Ordered Binary Decision Diagrams (OBDDs), a compact way of representing Boolean functions as a rooted, directed acyclic graph, which exploits similarities between functions. Though faster than Elvin, this approach does not scale well either, being suitable only for applications in which sub-

scriptions are highly similar with respect to both predicates and the combination of predicates.

[Li 2005] uses Modified Binary Decision diagrams (MBDs) to index subscriptions and in addition uses a one-dimensional predicate index. However, this approach can only support conjunction subscriptions and requires high predicate redundancy between subscriptions.

The counting algorithm [Ashayer 2002] supports conjunctive subscriptions only and uses a one-dimensional predicate index. The algorithm counts the number of fulfilled predicates per subscription and then checks if it equals the number of overall predicates. This approach is scalable and does not require redundancy among predicates. Moreover, it is easy to register and unregister subscriptions.

[Carzaniga 2003] indexes predicates and uses an extended counting algorithm to evaluate DNF expressions. General Boolean expressions need to be transformed in DNF form leading to an increase of the number of filters.

[Whang 2009] uses inverted list data structures to index Disjunctive or Conjunctive Normal Forms (DNF or CNF). This approach requires transforming general Boolean expressions in DNF or CNF, leading to an increase in the index size. The idea is to index all predicates using a hash table which allows to search for subscriptions containing the predicate. Subscriptions are further ordered by the number of conjunctions/disjunctions they contain.

[Bittner 2005] extends the counting algorithm to support general Boolean expressions represented as trees with inner nodes containing Boolean operators AND, OR, and NOT and leaf nodes containing predicates. Predicates are indexed in order to allow fast identification of all satisfied predicates by an incoming event. In a second step, subscriptions containing satisfied predicates are identified and the Boolean expression of each subscription is evaluated.

[Fontoura 2010] describes and compares the performance of two algorithms able to match general Boolean expressions. The first algorithm, Dewey ID, represents Boolean expressions as trees with alternating AND-OR nodes in every path from the root to the leaves. Leaf nodes are conjunctions of the form $State \in \{CA, NY\}$ or $State \notin \{CA, NY\}$. Conjunctions (which also include simple predicates) are then annotated with a compact description of where the conjunction appears in the tree using Dewey IDs. A Dewey IDs encodes the path from the root node of the tree to the conjunction stored in the leaf node. The Boolean expression tree is not stored, instead relevant parts of it can be reconstructed from the matched Dewey IDs and can be evaluated to true or false. The second algorithm is called Interval IDs and experiments show it generally outperforms the Dewey ID. We give in the following the details of this algorithm.

The Interval ID algorithm maps each leaf node of a Boolean Expression (BE) tree to a sub-interval of the $[1, M]$ interval, where M is the maximum number of leaf nodes any tree might have. A tree is satisfied if there exists a set of satisfied leaves that cover without overlap the whole interval $[1, M]$. We give simple examples to explain the main idea of the algorithm. If the Boolean expression is $A\ OR\ B$, both A and B will be labelled with $\langle 1, M \rangle$ because the presence of any of these attributes

satisfies the tree. If the Boolean expression is *A AND B*, the intervals could be $\langle 1, M/2\rangle$ for A, and $\langle M/2, M\rangle$ for B. In order to cover the whole interval, both A and B are required.

Algorithm 26 shows the details of the Label algorithm that is called to label each leaf node in a tree with the begin and end values of its interval. In the algorithm, n.leftLeaves denotes the total number of leaves appearing in the tree before node n in a pre-order traversal of the tree.

Algorithm 26 The Label algorithm.

Input: Node n.

```
1: if n is a leaf then
2:    return
3: else if n is an OR node then
4:    for all children c of n do
5:       c.begin ← n.begin
6:       c.end ← n.end
7:       Label(c)
8:    end for
9: else if n is an AND node then
10:   for first child c do
11:      c.begin ← n.begin
12:      c.end ← n.leftLeaves + c.size
13:      Label(c)
14:      curr ← c.end+1
15:   end for
16:   for all intermediate children c of n do
17:      c.begin ← curr
18:      c.end ← curr + c.size-1
19:      Label(c)
20:      curr ← c.end+1
21:   end for
22:   for last child c do
23:      c.begin ← curr
24:      c.end ← n.end
25:      Label(c)
26:   end for
27: end if
```

After the tree has been labelled, it can be matched against a set of satisfied leaves, where each leaf is represented by an interval $\langle begin; end\rangle$. The details of the match algorithm are given in Algorithm 27.

Algorithm 27 The Match Algorithm.

Input: I: set of intervals $\langle begin; end\rangle$ sorted by $begin$.
1: **return** $true$ or $false$.
2: matched ← Boolean Array of length $M + 1$
3: Initialize matched[i] to false for all i
4: matched[0] = true
5: **for all** intervals $\langle begin; end\rangle$ in I **do**
6: **if** matched $[begin - 1]$ **then**
7: matched $[end] \leftarrow true$
8: **end if**
9: **end for**
10: **if** matched[M] **then**
11: **return** $true$
12: **else**
13: **return** $false$
14: **end if**

6.2.2 Event routing optimizations

Routing optimizations can be classified as subscription-based optimizations and advertisement-based optimizations.

Subscription information is used by brokers to create event routing tables instead of simply broadcasting all events to all brokers. An event is forwarded to a neighbour broker only if the broker sent a subscription matching the event. Three subscription routing optimizations are used by current pub/sub systems: (i) covering-based, in which coverage relations are defined between subscriptions, (ii) merging-based, in which several subscriptions are merged into a more general subscription, and (iii) summarization-based [Triantafillou 2004] in which brokers forward to each other summaries of the subscriptions they have and then merge received summaries.

Covering is the most popular optimization used for example by SIENA, REBECA, and PADRES. Formally subscription covering can be defined as:

Definition 18 (Subscription covering)**.** *A subscription s_1 covers another subscription s_2 if s_1 matches all the events that are matched by s_2.*

Merging [Mühl 2001, Li 2005] is also widely used, sometimes together with covering. Formally, we can define subscription merging as:

Definition 19 (Subscription merging)**.** *A subscription s is called a merger of a subscription set S_i iff s matches all the events that are matched by the subscription in S_i. Subscription s is called a perfect merger if it matches exactly all the events matched by S_i and imperfect merger if it matches more events.*

In order to increase the efficiency of event routing, most systems require publishers to specify the schema of their future events through *advertisements*. An advertisement defines a Boolean expression of predicates and has the same structure as a subscription. An event message E conforms to an advertisement A if the

Boolean expression of A evaluates to true on E. All events sent by a publisher need to conform to one of its registered advertisements. Advertisement information is used by brokers to propagate subscriptions among each other and avoid broadcasting subscriptions in the network. Advertisements are used to create subscription routing tables. A subscription will be forwarded only to brokers that sent an advertisement for events that could match the subscription. Covering and merging optimizations can be applied to advertisement forwarding in the same way they are applied to subscriptions.

In the next section we survey how confidentiality preserving routing solutions handle event filtering and routing optimizations.

6.3 Related work

Efficient event filtering and routing are only marginally addressed by confidentiality-preserving schemes.

6.3.1 Confidential event filtering

From the solutions we reviewed in Chapter 3, only a few allow subscription indexing, but they can build routing tables with only one keyword, the topic [Srivatsa 2005, Shikfa 2009]. In [Srivatsa 2005], all subscribers subscribing to a topic w are given the same decryption key and token by the trusted key authority (KA). The token is sent to the broker and represents the subscription. The broker can then build a routing table using these tokens. Because the solution is vulnerable to inference attacks, it provides a probabilistic multi-path event routing scheme that flattens the occurrences of each topic at each router. In [Shikfa 2009], the encryption of the topic name is done using local keys established between neighbouring brokers, instead of being provided by a central authority, and as a result, the same topic encrypts to a different ciphertext at each broker.

Barazzutti et al. [Barazzutti 2012] propose a pre-filtering mechanism that allows faster encrypted filtering by reducing the number of calls to the encrypted matching function. This solution is intended to work with any privacy-preserving encrypting matching scheme. The solution works by extending both events and filters with a Bloom filter that encodes equalities such as "SYM=IBM". A simple bit-wise operation on the Bloom filters allows discarding a subset of non-matching subscriptions. The hash functions used to construct the Bloom filter are not known to the broker and are parametrized by an encryption key, shared by publishers and subscribers. Supporting disjunctions requires creating several Bloom filters per subscription, as a Bloom filter can only encode conjunctions.

6.3.2 Encrypted routing optimizations

Routing optimizations in confidentiality-preserving schemes are achieved using encrypted subscription covering. For example, the solutions from [Raiciu 2006,

Choi 2010, Nabeel 2009] allow the brokers to compute covering relations between encrypted subscriptions, but they require publishers and subscribers to share a secret key. This approach does not scale up because it requires re-keying whenever a participant leaves the system.

6.4 Solution details

We target a scalable solution for large pub/sub systems that allows brokers to efficiently match events against a large number of filters while preserving confidentiality. In order to provide a scalable solution, we create an index that allows identifying matching filters faster and does not require testing filters one by one. We assume incoming events are encrypted as in the previous chapters. An event consists of: (i) the message M that represents the content of the event and (ii) a set of attribute assignments $a_i = v_i$ that characterise M and are used for event filtering by the brokers. An attribute assignment has the form *attr_name=attr_value*, where *attr_value* can be either a string or a number. The message content M is encrypted with PE or ABE and the attributes are encrypted with the trapdoor algorithm Trap of SDE, a proxy based algorithm that requires the local broker to re-encrypt the attributes.

Filters represent conjunctions and disjunctions of predicates. A predicate has the form *attr_name op attr_value* where *op* can be one of $=, \leq, <, \geq$, and $>$. We represent filters as access tree structures as previously, with internal nodes representing AND or OR relation and leaf nodes representing predicates. To allow building an index, we encrypt leaf nodes using Trap–U on the user side and re-encrypt them using Trap–S on the broker side. The solution used in Chapters 3 and 4 used the KE algorithm which is probabilistic and does not allow creating an index. Figure 6.1 shows the main steps of filter encryption when fine-grained access control policies are supported using KP-ABE. As compared to Figure 4.3, we changed the leaf node encryption in steps (4) and (5).

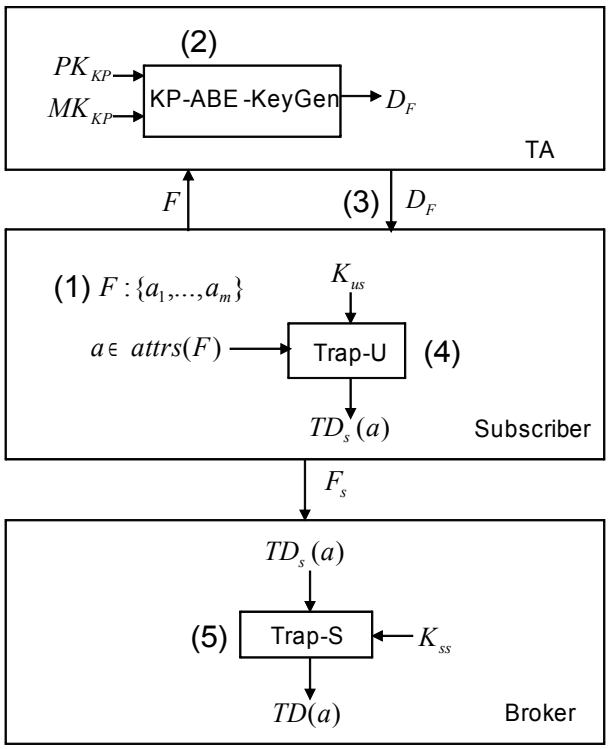


Figure 6.1: Filter generation and encryption.

6.4.1 A simple solution indexing predicates

We first describe a simple solution that indexes the leaf nodes of the trees, thus allowing for faster identification of matching filters. With this solution, filters do not need to be checked one by one as in the non-indexed scheme.

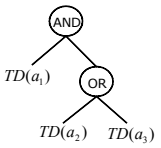
The broker indexes filters and the predicates contained in their leaf nodes using two data structures. First, a hash table called the *predicate index* maps predicates to the IDs of the filters containing them as shown in Table 6.2.

Table 6.2: Example of a predicate index.

Predicate	**Filter IDs**
$TD(a_1)$	$F_1, F_2, ...$
$TD(a_1)$	$F_3, F_4, ...$
$TD(a_3)$	$F_5, F_6, ...$
$TD(a_4)$	$F_2, F_3, ...$
$TD(a_5)$	$F_1, F_5, ...$
$TD(a_6)$	$F_4, F_6, ...$
...	...

Another table called the *filter index* maps filter IDs to the actual filters as shown in the example from Table 6.3.

Table 6.3: Example of a filter ID map.

Filter ID	Encrypted Filter
F_1	
F_2	...
...	...

The encrypted event filtering algorithm is given in Algorithm 28. When a new event E arrives, the broker queries the predicate index using the re-encrypted attributes $\{TD(a)\}_{a\in\gamma}$ of the event to retrieve the IDs of all filters containing one of the attributes. The broker then matches these filters one by one using the simplified filter matching function iTreeEval we introduced in the previous chapter and shown in Algorithm 25. This is a recursive function that verifies if the Boolean expression of the tree evaluates to true, given a set of satisfied predicates. This function is more efficient than the non-indexed matching function described in Algorithm 14 because it does not make calls to the SDE–Match function of SDE. To verify if a filter is satisfied, the broker calls the iTreeEval function on the root node of F^*.

Algorithm 28 Encrypted Event Filtering

Input: A predicate index PIdx, a filter index FIdx, a re-encrypted set of attributes A of an event E.

Output: a list L of filter IDs that are matched by the event E.

```
1: initialize a list of filter IDs L
2: for all re-encrypted attributes a ∈ A do
3:    if a belongs to PIdx then
4:       for all filter IDs f in PIdx(a) do
5:          add f to L
6:       end for
7:    end if
8: end for
9: for all filter IDs f in L do
10:   retrieve the filter F at FIdx(f)
11:   if iTreeEval(F.root, A) is false then
12:      remove f from L
13:   end if
14: end for
15: return L.
```

6.4.2 Indexing Boolean expressions

The previous solution from Section 6.4.1 indexes the predicates, but the index does not reflect the filter structure. In the following we describe a solution using the Interval ID algorithm of [Fontoura 2010] that includes the filter structure into the index.

When a new filter arrives, the broker runs the Label algorithm showed in Algorithm 26 to label each leaf node of the tree with an interval on the line $[1, M]$, where M is the maximum number of leaf nodes a filter might have and is fixed for the system at set up. Each leaf node contains a trapdoor encryption of a predicate of the form $attr\_name = value$. The broker then indexes under the predicate of the leaf node encrypted as $TD(a)$, the filter ID together with the label of the predicate in the filter. Table 6.4 shows an example of such an index. We note that the server does not need to store the actual filter as in the previous scheme, because the filter structure is already reflected in the predicate index.

Table 6.4: Example of a predicate index.

Trapdoor	**Label**
$TD(a_1)$	$F_1 : \langle 1; 7\rangle$ $F_2 : \langle 3; 5\rangle$ $F_5 : \langle 4; 7\rangle$
$TD(a_2)$	$F_2 : \langle 1; 2\rangle$ $F_4 : \langle 5; 7\rangle$
$TD(a_3)$	$F_1 : \langle 7; 9\rangle$ $F_3 : \langle 3; 5\rangle$ $F_4 : \langle 2; 7\rangle$
...	...

When the broker receives a new event, for each attribute trapdoor $TD(a)$ in the event, the broker identifies the entry in the predicate index corresponding to $TD(a)$ and retrieves the filter IDs and corresponding labels. The broker then sorts all such entries by filter ID and *begin* value in the label interval. For each filter ID, it runs the SDE–Match algorithm using the intervals sorted by *begin* value. The details of the event matching algorithm are given in Algorithm 29.

Algorithm 29 Filter matching.

Input: A predicate index PIdx and a set of trapdoors A.
Output: A L list of matched filters.

```
1: initialize a list L of filter IDs
2: initialize a list FI for storing pairs of filter IDs and intervals of the form ⟨begin; end⟩
3: for all attribute trapdoors TD(a) in A do
4:    if TD(a) belongs to PIdx then
5:       add to FI the content of PIdx(TD(a))
6:    end if
7: end for
8: for all distinct filter IDs f in FI do
9:    create a set I that contains all intervals ⟨begin; end⟩ pairs with f
10:   sort I by the begin value
11:   if match(I) returns true then
12:      add f to L
13:   end if
14: end for
15: return L.
```

This algorithm has larger pre-processing times as compared to the first algorithm, but event matching is faster.

6.5 Performance comparison of the schemes

We implement the two indexing schemes in Java and compare their performance. We tested the implementation on an Intel Core2 Duo 2.8 GHz with 3.48 GB of RAM. There are two main operations that a broker needs to perform: (i) add a new subscription to the index, and (ii) evaluate an incoming event on the indexed subscriptions. We compare the cost of these two operations for different datasets of filters. To analyse how the depth of the tree affects the performance of the schemes, we use different tree depths for each dataset.

We generate 5 datasets of 20,000 filters each for different tree depths. Figure 6.2 compares the indexing time of the two schemes for each of the datasets. As expected, the simple solution (SS) is faster than the Interval IDs (II) solution. That is because SS only indexes the leaf nodes of the tree, while II also indexes the structure of the tree.

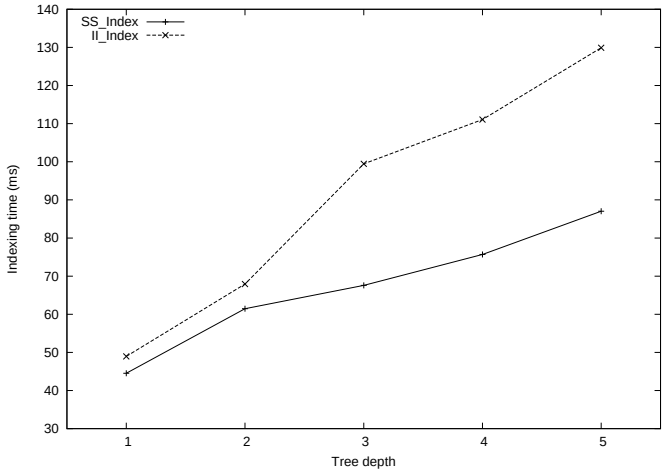


Figure 6.2: Indexing time of 20,000 filters for different depths.

We now evaluate query execution time on each of the 5 indexes. We choose the query (in the form of event attributes) that matches most filters and run it under both schemes. Figure 6.3 compares the performance as an average of running this query 1000 times. For small trees, the simple solution performs better, but when the depth of the tree grows, the Interval IDs solution is more efficient.

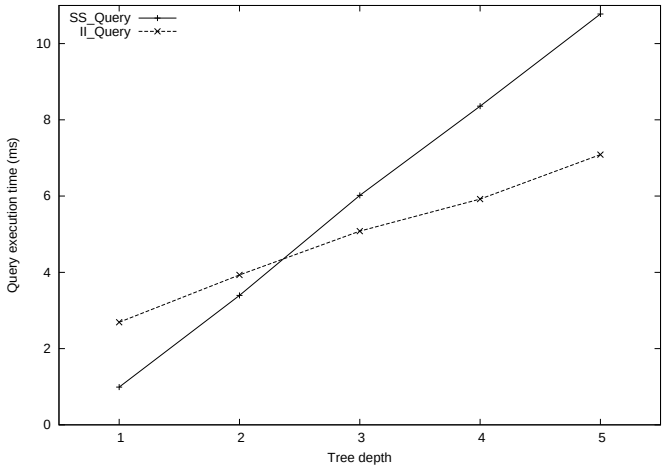


Figure 6.3: Query execution time on 20,000 filters.

We further compare the performance of the schemes for different index sizes using filters with depth 5. Figure 6.4 shows the times it takes to index sets of filters of different sizes using both of the methods. Once again, the SS scheme is more efficient from this point of view.

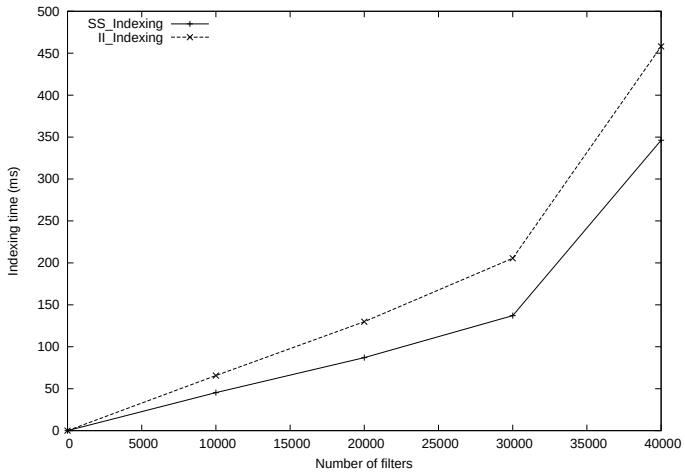


Figure 6.4: Indexing time for different numbers of filters.

Figure 6.5 compares the event matching times of the two schemes for different numbers of filters. The simple solution indexing only predicates is less efficient for very large filter sets, but it still manages to match 40,000 filters in about 22 ms on an average laptop, as compared to 14.5 ms for the method indexing Boolean expressions. For 10,000 filters it takes an around 5 ms for both methods. Servers usually have much better resources and the event matching time on such powerful machines could be significantly reduced. Moreover, because both methods are efficient even for large filter sets, they could prove practical for low capacity devices with limited resources at the edge of the pub/sub network that do not need to filter large amounts of publications or store that many filters.

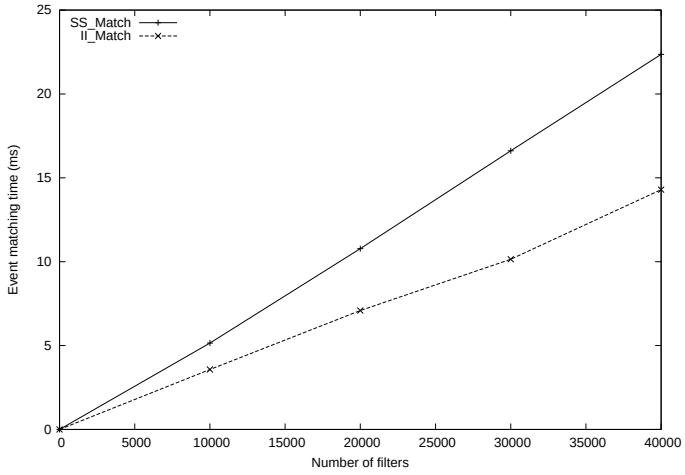


Figure 6.5: Event matching time for different numbers of filters.

We now compare these solutions with the non-indexed scheme. For filter depth=5, it takes 25 seconds to match 10,000, 50 seconds to match 20,000 filters,

75 to match 30,000 filters and 100 seconds to match 40,000 filters. This is much less efficient then the indexed methods, but on a powerful server for applications that do not have strict performance requirements, even this solution could be suitable.

6.6 Inference exposure

In the following we analyse the inference exposure of the two filtering approaches we introduced, the non-indexed solution from Chapters 3 and 4, and the indexed solutions from this chapter. In the non-indexed solution from Chapters 3 and 4, filter leaf nodes are encrypted with a probabilistic algorithm, and do not leak attribute frequency information. In this chapter we introduced another approach that indexes filters by the attributes stored in their leaf nodes. This solution significantly speeds up the event matching times, but allows the server to additionally learn statistical information about the attributes.

We only consider the stronger $\mathsf{DB} + \mathsf{DB}^{\mathsf{K}}$ threat model because the equivalent of the $\mathsf{Freq} + \mathsf{DB}^{\mathsf{K}}$ threat model translates into an adversary that knows the frequency of predicates in filters, but not the frequency of predicate associations in filters. This model can be easily solved for the indexed solution as in the previous chapter by constructing equivalence classes for leaf node attributes and computing their probabilities from the cardinalities of the classes. We already showed that under this threat model the exposure coefficient is small and may not even require protection mechanisms. In the case of the non-indexed scheme, the $\mathsf{Freq} + \mathsf{DB}^{\mathsf{K}}$ attack is not possible because the broker cannot compute the frequency of attributes from the static index.

6.6.1 Exposure of the non-indexed scheme

The non-indexed scheme only leaks the structure of the tree that encodes the filter. We assume that the server knows the plaintext filters and has access to the encrypted filters. For each un-encrypted filter, the server can compute its tree structure. The server then tries to match the computed structures with the structures of the encrypted filters it stores.

We first assume all filters refer to the same numeric attribute. From the structure of these filters, the broker is trying to infer the inequality. Analysing the Inequality Policy Generation Algorithm from Chapter 4 described in Algorithm 21, we observe that the structure of a filter $a < v_1$ is the same as the structure of the filter $a > v_2$, if v_2 is the complement of v_1, i.e., $v_2 = 2^n - 1 - v_1$, where n is the number of bits on which the values are represented. Tables 6.5 and 6.6 show the structures of the trees for $n = 4$. This means, that if the server knows a filter contains only a numeric inequality, the server has a $1/2$ probability of guessing the inequality from the structure of the filter. For the filters $a < 2^n - 1$ and $a > 1$, the server can guess the inequality with probability 1. If the two complement inequalities have different frequencies, the server can identify them.

Table 6.5: Filter structures 1.

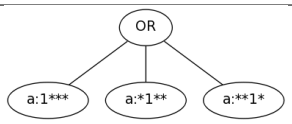
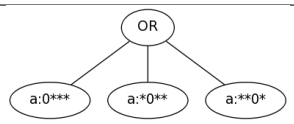
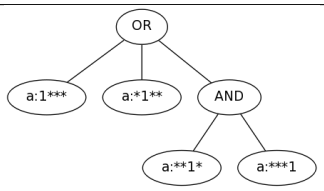
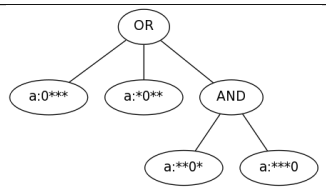
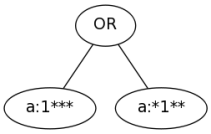
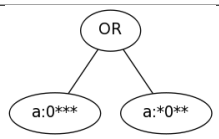
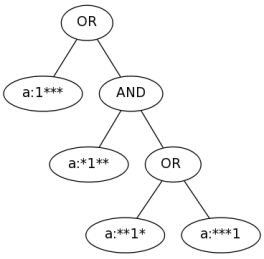
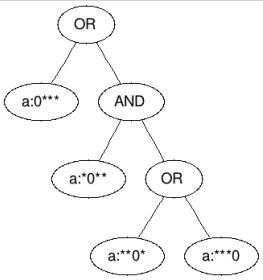
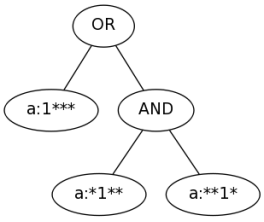
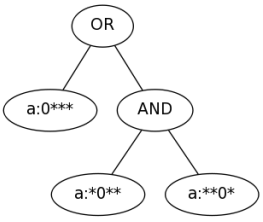
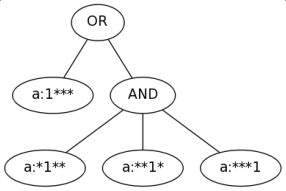
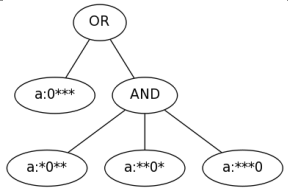
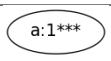
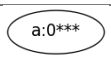
$a > 1 = 0001_b$	$a < 14 = 1110_b$
	
$a > 2 = 0010_b$	$a < 13 = 1101_b$
	
$a > 3 = 0011_b$	$a < 12 = 1100_b$
	
$a > 4 = 0100_b$	$a < 11 = 1011_b$
	
$a > 5 = 0101_b$	$a < 10 = 1010_b$
	
$a > 6 = 0110_b$	$a < 9 = 1001_b$
	
$a > 7 = 0111_b$	$a < 8 = 1000_b$
	

Table 6.6: Filter structures 2.

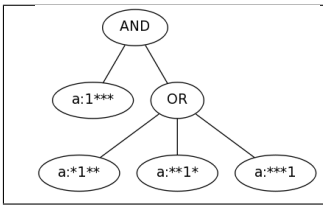
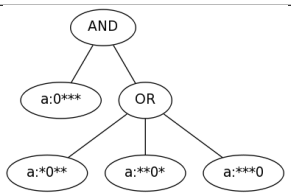
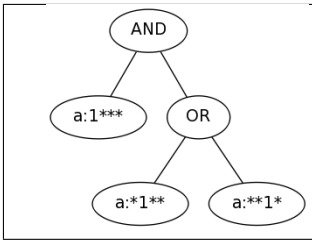
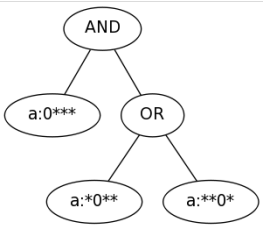
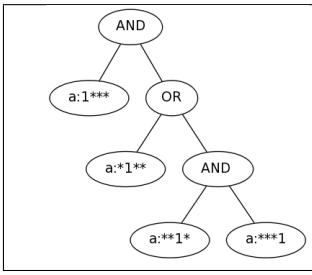
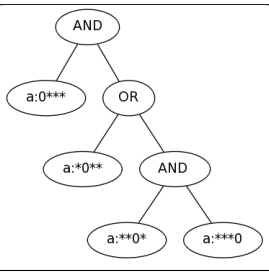
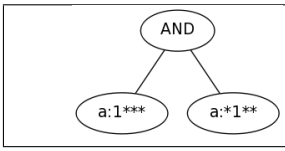
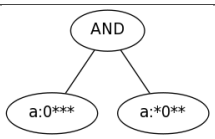
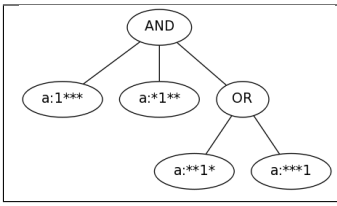
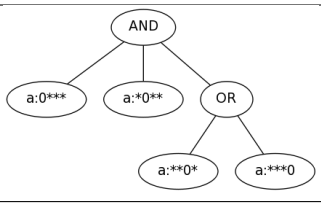
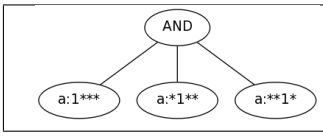
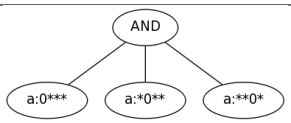
$a > 8 = 1000_b$	$a < 7 = 0111_b$
	
$a > 9 = 1001_b$	$a < 6 = 0110_b$
	
$a > 10 = 1010_b$	$a < 5 = 0101_b$
	
$a > 11 = 1011_b$	$a < 4 = 0100_b$
	
$a > 12 = 1100_b$	$a < 3 = 0011_b$
	
$a > 13 = 1101_b$	$a < 2 = 0010_b$
	

Table 6.7: Filters having the same tree structure.

a<3	d<8 and a>9
b and a>9	d>7 and a<6
b and a<6	b<8 and a<6
a>12	b and c and (d or e)
b and c and a<3	b and c and a>12
a<4 and d>3	a<4 and (d or e)
a>11 and d<12	a>11 and (d or e)
a<4 and d<12	a>11 and d<3
b and c and (a>7 or d<8)	a>7 and c and (d or e<8)
a>7 and d<8 and (e or f)	b<8 and a>9

However, when filters have more than one attribute, which is the case in large scale pub/sub systems, the probability of the server inferring the filter from its structure significantly decreases because many filters collide on the same structure. For example, the 20 filters in Table 6.7 have the same structure. The list is not exhaustive as many other filters would have the same structure.

We group all filters with the same tree structure t in a class C_t. We define the probability for the broker to guess a filter as:

$$\varepsilon_i = f_i \frac{1}{|C_t|} \tag{6.1}$$

where f_i is the number of occurrences of the filter, and $|C_t|$ is the cardinality of the class.

Let us take the following examples. First we assume we have 7 unique filters that collide on 3 unique tree structures as follows:

$C_{t1} = \{F_1, F_2\}$

$C_{t2} = \{F_3, F_4, F_5\}$

$C_{t3} = \{F_6, F_7\}$

The probabilities of guessing each filter can be computed as follows: $\varepsilon_1 = \varepsilon_2 = \frac{1}{|C_{t1}|} = \frac{1}{2}$, $\varepsilon_3 = \varepsilon_4 = \varepsilon_5 = \frac{1}{|C_{t2}|} = \frac{1}{3}$, $\varepsilon_6 = \varepsilon_7 = \frac{1}{|C_{t3}|} = \frac{1}{2}$.

We take another example in which filters are not unique. For example, a particular filter such as a specific stock might be very popular with many subscribers registering the same filter. We now assume that filters F_1 and F_4 occur more times and we have the following classes:

$C_{t1} = \{F_1, F_1, F_2\}$

$C_{t2} = \{F_3, F_4, F_4, F_4, F_5\}$

$C_{t3} = \{F_6, F_7\}$

The probabilities of each filter become: $\varepsilon_1 = f_1 \frac{1}{|C_{t1}|} = \frac{2}{3}$, $\varepsilon_2 = f_2 \frac{1}{|C_{t1}|} = \frac{1}{3}$, $\varepsilon_3 = f_3 \frac{1}{|C_{t2}|} = \frac{1}{5}$, $\varepsilon_4 = f_4 \frac{1}{|C_{t2}|} = \frac{3}{5}$, $\varepsilon_5 = f_5 \frac{1}{|C_{t2}|} = \frac{1}{5}$, $\varepsilon_5 = f_6 \frac{1}{|C_{t3}|} = \frac{1}{2}$, $\varepsilon_7 = f_7 \frac{1}{|C_{t3}|} = \frac{1}{2}$.

We compute the inference exposure of a set of filters as the average probability of guessing each filter.

$$\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\varepsilon_i = \frac{1}{n}\sum_{i=1}^{n}\frac{f_i}{|C_t|} = \frac{1}{n}\sum_{j=1}^{m}\sum_{F_i \in C_{t_j}}\frac{f_i}{|C_{tj}|} = \frac{m}{n} \tag{6.2}$$

where n is the number of unique filters and m is the number of classes. The last equality holds because the sum of the probabilities of all filters in one class is 1, i.e., $\sum_{F_i \in C_t} \varepsilon_i = 1$.

To compute the exposure of a set of filters, we need to count the number of unique filter structures and divide it by the total number of unique filters. Using this formula, we compute the exposure of the two filter sets we considered above to be 3/7 in both cases.

Using Equation 6.2 we first analyse how numeric inequalities expressed on different numbers of bits collide. This is due to the fact that the same inequality has a different tree structure when represented on different numbers of bits. We fist generate all inequities on 4 bits and obtain an exposure coefficient of 0.5357. We then generate all inequalities on 4 and 5 bits and notice that the coefficient decreases to 0.3523. Table 6.8 shows the coefficients for different filters sets obtained like that until 10 bits for which the coefficient decreases to 0.2535.

Table 6.8: Exposure coefficient for filters representing a single numeric inequalities.

Number of bits	**Exposure coefficient**
4	15/28=0.5357
4 and 5	31/88=0.3523
4, 5 and 6	63/212=0.2972
4, 5, 6 and 7	127/464=0.2737
4, 5, 6, 7 and 8	255/972=0.2623
4, 5, 6, 7, 8 and 9	511/1992=0.2565
4, 5, 6, 7, 8, 9 and 10	1023/4036=0.2535

To assess the inference exposure of the tree structure in large filter sets, we generate filter sets with different numeric and non-numeric attributes. We aim to create balanced sets of combinations of the possible attributes.

For a set of 262 unique filters with combinations of two non-numeric attributes and one numeric attribute on 4 bits we obtained the inference exposure $\varepsilon = 65/262 = 0.2481$.

For 892 unique filters with combinations of 3 non-numeric attributes and 2 numeric attributes on 4 and 5 bits respectively, we obtained $\varepsilon = 143/892 = 0.1603$.

For 1516 unique filters with combinations of 4 non-numeric and 3 numeric attributes one on 4 and two on 5 bits, we computed $\varepsilon = 161/1516 = 0.1061$.

The exposure of the non-indexed scheme is quite low because the same filter structure can correspond to a large number of subscriptions, consisting of both numeric and non-numeric attributes. The inference decreases with the number of filters and the number of attributes over which the filters are expressed.

6.6.2 Exposure of the indexed scheme

Following the $\mathsf{DB} + \mathsf{DB}^{\mathsf{K}}$ threat model, we assume the server knows the entire plaintext filters and their encryption. We use a similar approach as in the case of indexing events and create a graph from all the filters. There are several types of vertices in the graph: (i) one vertex for each leaf node attribute, (ii) one vertex per filter ID, and (iii) one vertex per each unique AND and OR relation in the filters. This differs from the graph we constructed in the previous section which only had two kind of vertices: one for attributes and one for each event. This graph is more complex because it also reflects the tree structure of each filter.

The AND and OR internal nodes connected to leaf nodes are added to the graph in the following way. Each unique internal node is added just once with a unique color. For example, if there are 3 filters of the form *attribute AND attribute*, we add only once the AND node to the graph. We illustrate this with a simple example shown in Figure 6.6.

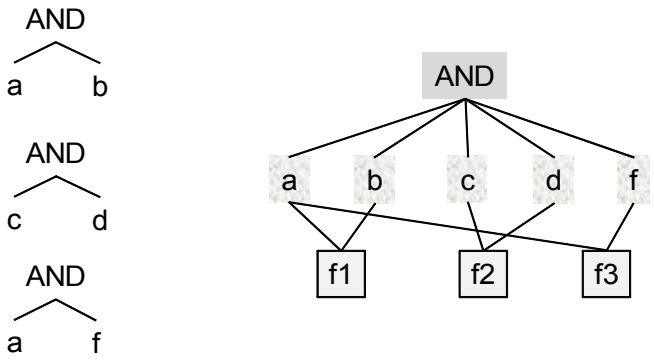


Figure 6.6: Filter index and corresponding associations graph.

Let us take a more complex example as in Figure 6.7. There are two kind of AND nodes and one type of OR node.

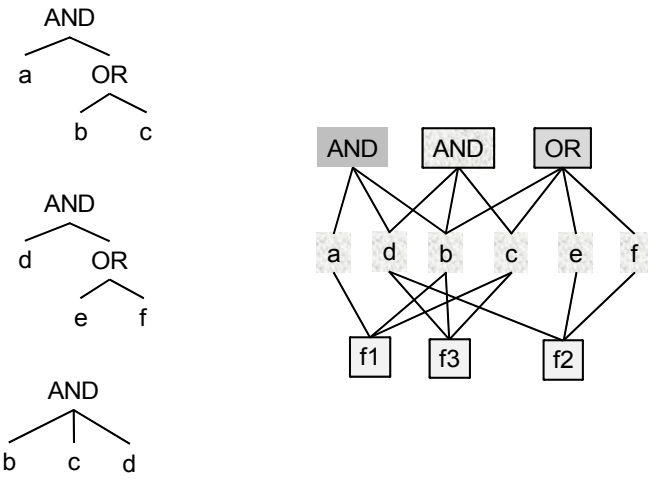


Figure 6.7: Filter index and corresponding associations graph.

To assess the inference exposure from the graph, in the previous section we computed the equitable partition of attribute vertices, which gives the probability of guessing each attribute. This method can also be applied here. In the first example from Figure 6.6, the equitable partition of attribute vertices is {a, (b f), (c d)}. Vertices in the same class can be substituted for each other in isomorphic graphs and the probability of inferring them is 1 over the cardinality of the class. We computed the average probability of guessing the attributes as the number of classes over the number of distinct attributes and in this case that is 3/5. For the example from Figure 6.7, the equitable partition of attribute vertices is {a, (b c), d, (e f)}, and the exposure coefficient is 4/6.

However, the probability of guessing each attribute may not always be a good measure when the server is trying to match plaintext filters with encrypted filters. For example, if the server can determine that a particular filter is either *c AND d* or *d AND c*, the attribute exposure is 1/2 as c and d will form one class. However, in this case, we can consider that even if the server cannot tell which attribute is c and which one is d, the server can still identify the relation. This is another way of assessing the inference exposure of an index of filters and could be used in conjunction with the first one. To compute the filter exposure, instead of computing the equitable partition of attribute vertices, we will compute the equitable partition of filter ID vertices. Filters belonging to the same partition or class can be substituted for each other, so the broker cannot distinguish between them. To compute the classes of filters or equitable partitions, we use the Nauty algorithm [McKay] as before. The equitable partition of filter IDs for Figure 6.6 is {(1 3), (2)}. This means that filters 1 and 3 are inferred with probability 1/2 and filter 3 is inferred with probability 1. The average probability is the number of partitions over the number of filters. In this case, that is 2/3=0.67 which is slightly greater than 3/5=0.6 obtained as attributes exposure. The equitable partition of filter IDs for Figure 6.7 is {1,2,3} and the exposure coefficient is 1, which means that the

broker can identify each filter with probability 1, but not completely identify the correspondence between ciphertext attributes and plaintext attributes, because the attribute inference is 4/6.

Let us take a more complex example as in Figure 6.8.

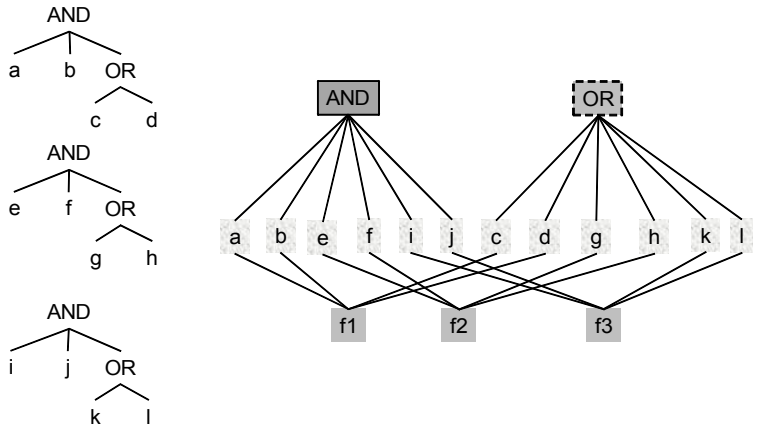


Figure 6.8: Filter index and corresponding associations graph.

The equitable partition of attributes is {(a b e f i j), (c d g h k l)} which gives an exposure coefficient of 2/12=0.17. The equitable partition of filter IDs is {(1,2,3)}, so the exposure coefficient is 1/3=0.33, significantly greater. However, we note that in this example, though the exposure coefficient appears to be quite large, in fact the index does not leak anything to the server, because the exposure coefficient equals the probability of a random guess. Perhaps a better inference exposure metric should cater for the probability of random guess. The attribute exposure coefficient is a better measure of inference exposure if we assume that the attacker had access to both plaintext and ciphertext values at one point, and then with this knowledge is trying to infer future encrypted filters about which it has no knowledge.

Chapter 7

Implementation and Integration with Different Middlewares

Contents

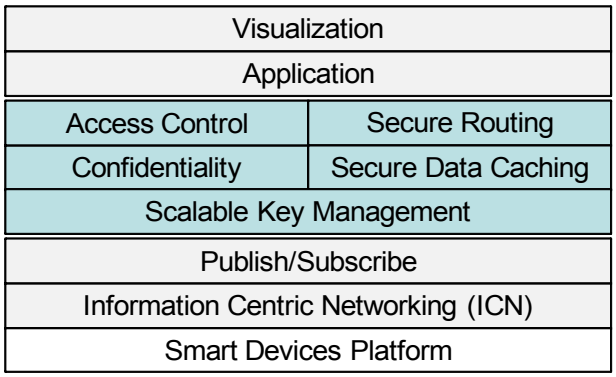
In this chapter we describe the main components of our implementation, how they interact and how they can be integrated with different middleware and applications. The goal was to create middleware-agnostic libraries that could be integrated with multiple systems.

7.1 Implementation overview

We start by giving an example of an architecture that shows how networking, security solutions and applications fit together and support each other. Figure 7.1 shows the main components organized as a logical stack. At the bottom, we have devices that generate data such as sensors, smart meters, electric vehicles, and mobile devices handled by users. An ICN or pub/sub system connects these devices and delivers data between them in an asynchronous manner that enables efficient multiparty communication and decoupling of publishers and subscribers. Data is identified by its name when using ICN or by content through attributes when using a pub/sub system. ICN only provides basic data identification and retrieval by name, and does not provide rich content-based networking as pub/sub systems do. To provide such functionality, a pub/sub systems needs to be built on top of the ICN. We note that the ICN layer of our architecture is optional, as the pub/sub system could be build either on an ICN or on any communication protocol. The middle layers in the figure represent the security solutions we described in this thesis, which

secure the communication over the pub/sub system. On top of our Scalable Key Management, we built solutions for Confidentiality & Access Control for securing events and subscriptions, and Secure Routing and Data Caching components. On top of our security components run different applications that make use of one or more of the security functionalities, and/or visualization and analysis components that display the data to users, or aggregate the data and transform it.

Figure 7.1: Components stack.



We further describe the implementation of the security components and show how they can be integrated with different middlewares. Our implementation consists of three layers shown in Figure 7.2:

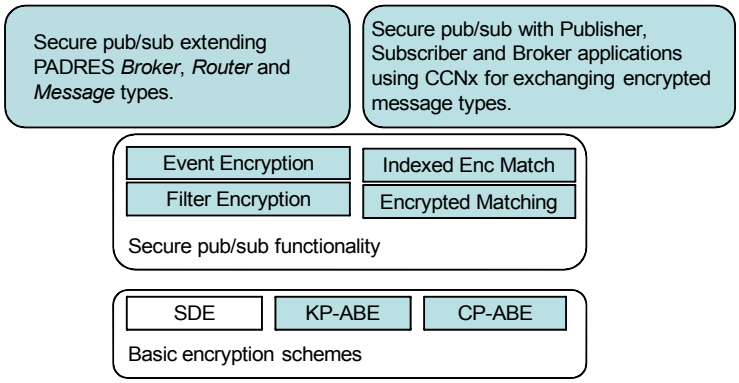
- **Basic encryption libraries**: Our scheme is based on three main encryption schemes: SDE, KP-ABE and CP-ABE. We used the SDE implementation of [Dong 2011], and implemented in Java KP-ABE and CP-ABE as described in [Goyal 2006a] and [Bethencourt 2007] respectively, using the Java Pairing Based Cryptography Library (jPBC)[1].

- **Libraries implementing our schemes**: We implemented the schemes as described in Chapters 3, 4, 5 and 6. These libraries allow encrypting an event and filter, and performing encrypted filtering.

- **Integration with PADRES and CCNx**: We demonstrate the usage of our schemes with two popular middlewares: a distributed pub/sub system called PADRES [Jacobsen 2010] and an ICN implementation of PARC called CCNx [Jacobson 2007]. We took different approaches to integrating the libraries with these middlewares. Because PADRES is a full-fledged pub/sub system, we created classes that extend the main message types in PADRES and the router and broker and integrated those with PADRES. Because CCNx does not provide a pub/sub system, we built simple Publisher, Subscriber and Broker applications that run on top of CCNx.

[1] http://gas.dia.unisa.it/projects/jpbc/

Figure 7.2: Libraries stack.

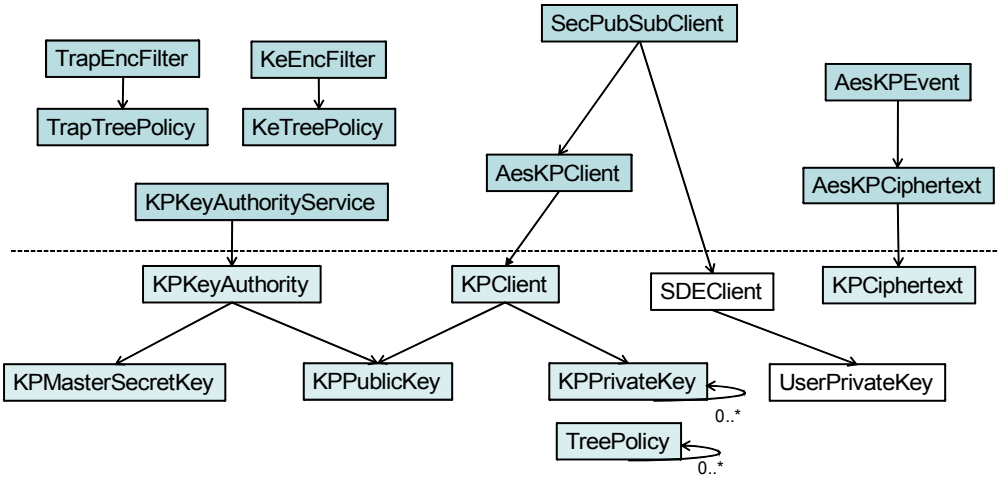


7.2 Libraries

In this section we describe the implementation of the basic encryption schemes (i.e., ABE implementation) and the secure pub/sub implementation. The following two sections will describe the integration with CCNx and PADRES respectively.

Figure 7.3 shows the most important classes needed for event and filter encryption. The bottom classes implement the basic encryption algorithms while the top ones implement our secure pub/sub schemes.

Figure 7.3: Diagram showing the main classes for event and filter encryption.



7.2.1 Basic encryption schemes implementation

We implemented KP-ABE as in [Goyal 2006a]. The main KP-ABE classes are shown at the bottom of Figure 7.3. KP-ABE requires a trusted Key Authority (KA), implemented by the class *KPKeyAuthority*, that generates the public (i.e.,

KPPublicKey) and the master secret (i.e., *KPMasterSecretKey*) keys. The KA distributes the public key to all senders and receivers, and stores securely the master secret key. We create a class *AesKPClient* that has the functionality for requesting the necessary keys and encrypting and decrypting messages using KP-ABE and AES as described in Chapter 4. KP-ABE is used to encrypt a random AES encryption key, while the actual content is encrypted using AES. Table 7.1 summaries the functionality of this class. In order to instantiate a client, we need a unique client name, and the hostname of the trusted KA for requesting the keys. Publishers encrypt messages under numeric or non-numeric attributes using the method *encrypt*. For example, a publisher can encrypt a message under the attributes "ambientdata", "airquality", and "trento". A subscriber that is allowed to receive such messages, will be issued by the KA a key (i.e, an instance of *KPPrivateKey*) for a specific access policy over attribute values. Such a policy could be "ambientdata and airquality and (trento or verona)". The subscriber requests the key using the method *getDecKey* and decrypts the ciphertext by calling *decrypt*.

Table 7.1: Summary of AesKPClient class.

Constructor
AesKPClient(String clientName, String kaHostname)
Key request methods
KPPublicKey getPublicKey()
KPPrivateKey getDecKey(String policy)
Encryption and decryption methods
AesKPCiphertext encrypt(String message, String[] attributes)
String decrypt(AesKPCiphertext cph, KPPrivateKey decKey)

The implementation for CP-ABE is similar and for lack of space we do not show the classes in the diagram. CP-ABE also requires a trusted Key Authority that generates public and master secret keys. The class *AesCPClient* shown in Table 7.2 provides the functionality for requesting the necessary keys and encrypting and decrypting messages. Messages in CP-ABE are encrypted under an access policy, while in KP-ABE messages are encrypted under attributes describing the message content. The decryption key generated by the Key Authority for each CP-ABE user is computed from the attributes of the user.

Table 7.2: Summary of AesCPClient class.

Constructor
AesCPClient(String clientName, String kaHostname)
Key request methods
CPPublicKey getPublicKey()
CPPrivateKey getDecKey(String[] attributes)
Encryption and decryption methods
AesCPCiphertext encrypt(String message, String policy)
String decrypt(AesCPCiphertext cph, CPPrivateKey decKey)

7.2.2 Secure pub/sub implementation

There are two main message types in our scheme: encrypted events and encrypted filters. Table 7.3 shows the components of an encrypted event using KP-ABE and AES to encrypt the message content, and the trapdoor algorithm Trap to encrypt attributes, as explained in Chapter 4. The publisherID is needed for the local broker to locate the server side of the key for re-encrypting the trapdoors.

Table 7.3: Summary of AesKPEvent class.

Attributes
String publisherID
AesKPCiphertext ciphertext
// Client computed trapdoors
BigInteger[][] trapdoors_u
// Broker re-encrypted trapdoors
Set<String> trapdoors
boolean reencrypted
Constructor
AesKPEvent(AesKPCiphertext cph, String publisherID)

Table 7.4 shows the components of an encrypted filter using the KE algorithm to encrypt leaf node attributes as described in Chapters 3 and 4. A subscription filter expressed as conjunctions and disjunctions of equalities and inequalities, is parsed as a *TreePolicy* object. The *KPPrivateKey* issued by the Trusted Authority is computed from this policy. To encrypt the filter, the subscriber encrypts the leaf nodes of a *TreePolicy* either with the keyword encryption algorithm KE or the trapdoor encryption algorithm Trap. The details of the *KeTreePolicy* class are given in Table 7.5.

Table 7.4: Summary of KeEncFilter class.

Attributes
String subscriberID
KeTreePolicy policy
boolean reencrypted
Constructor
EncFilter(String subscriberID, KeTreePolicy policy)

Table 7.5: Summary of KeTreePolicy class.

Attributes
// The threshold value of the node.
int threshold
// The client computed trapdoor if leaf node.
BigInteger[] trap_u
// The broker re-encrypted trapdoor if leaf node.
String trapdoor
// Children policies, null for leaf nodes.
ArrayList<EncTreePolicy> children
Constructor
KeTreePolicy(TreePolicy filter, SDEClient sdeClient)

The client which can be either a publisher or subscriber or both, has the following functionality:

Table 7.6: Summary of SecPubSubClient class.

Attributes
String clientID
AesKPClient kpClient
SDEClient sdClient
Constructor
SecPubSubClient(String clientID, String kpAuthorityHostname, SDEClient sdClient)
Methods
AesKPEvent encryptEvent(String message, String[] attributes)
KeTreePolicy generateFilter(String subscrFilter)
KPPrivateKey requestDecryptionKey(String accessPolicy)
String decrypt(AesKPCiphertext cph)

Figure 7.4 shows the three kinds of encrypted brokers we described: the non-indexed broker from Chapters 3 and 4 (i.e., *EncBroker*), the indexed broker implementing the simple indexed solution from Chapter 6 (i.e., *IndexBroker*) and the broker implementing the interval ID algorithm from Chapter 6 (i.e., *IntervIdBroker*). All the brokers have an SDEServer instance that stores the server side keys of the clients connected to the broker and re-encrypts events and filters created by those clients.

Figure 7.4: Diagram showing the main broker classes.

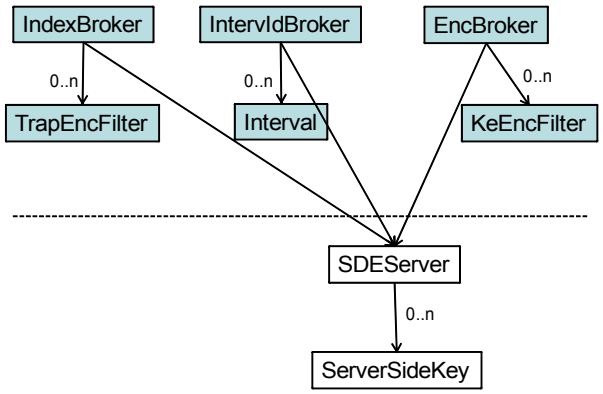


Table 7.7: Summary of EncBroker class.

Attributes
SDEServer sdeserver
Set<KeEncFilter> filterStore
Constructor
EncBroker(SDEServer sdeserver)
Methods
handleFilter(KeEncFilter filter)
handleEvent(AesKPEvent event)
eventRenc(AesKPEvent event)
filterRenc(KeEncFilter filter)
boolean match(KeTreePolicy policy, Set<String> trapdoors)

In the following we describe in more detail the integration with each middleware.

7.3 Integration with CCNx

There are two main message types in CCNx: *Interest* which is used to request data by name and *ContentObject* used to supply data in response to a matching Interest. An Interest contains a *ContentName* which is created from a CCNx URI like *ccnx:/ccnx.org/ambientdata/temperature/trento*. A ContentObject also contains a ContentName and the data payload as a byte array. This allows sending any kind of data object that can be serialized. A ContentObject matches an Interest, if the ccnx URI of the Interest is a prefix of the URI of the Content. CCNx does not provide any mechanisms for key management or complex routing, but only symmetric encryption algorithms. Moreover, in CCNx, interests are cancelled once they are answered, and subscriptions valid for a longer period of time are not supported. To provide a secure pub/sub system on top of CCNx, we create Publisher, Subscriber and Broker applications that exchange messages encrypted with our scheme over CCNx.

We use the Java interface of CCNx to write and read our serializable AesKPEvent messages. We create a publisher class (see Table 7.8) that encrypts and publishes events on CCNx under a specific naming expressed as a ccnxUri, and a subscriber class (see Table 7.9) that expresses interest in a ccnxUri, receives events published in response to the interest, and decrypts them. The encryption and decryption operations are done using the functionality of AesKPClient. Because CCNx cancels Interest objects once they are answered, the subscriber implements a *listen* method that keeps subscribing to the ccnxUri in order to receive updates, simulating the communication model of a pub/sub system. When a new message is received, $decryptMessage$ is called to decrypt and display the message.

Table 7.8: Summary of CCNPublisher class.

Attributes
SecPubSubClient encryptor
String ccnxUri
CCNSerializableObject<AesKPEvent> writeEvent
Method
publish(String message, String[] attributes)

Table 7.9: Summary of CCNSubscriber class.

Attributes
SecPubSubClient decryptor
String ccnxUri
CCNSerializableObject<AesKPEvent> readEvent
Methods
getDecKey(String accessPolicy)
subscribe(String ccnxUri)
listen()
decryptMessage(AesKPEvent message)

In CCNx, messages are requested and forwarded by name. Because name-based routing can be too simplistic in many scenarios, we add a CCNBroker that enables attribute-based routing. This feature allows users to express additional constraints on the attributes of the data such as *year=2012* or *temperature<16*. Routers or brokers use these constraints to filter-out undesired messages. To enable encrypted filtering, we create a *CCNEncFilter* class that contains a ccnxUri and extends KeEncFilter.

Table 7.10 shows the functionality of the CCNBroker class.

Table 7.10: Summary of CCNBroker class.

Attributes
Hashtable<ContentName, List<CCNEncFilter>> subscriptionStore
Methods
listenSubscriptions()
listenEvents(ContentName name)
getMatches(ContentName name, AesKPEvent event)
publishEvent(String ccnxUri, AesKPEvent event)

Figure 7.7 shows the flow of messages between subscriber, broker and publisher. The Broker first opens a *readSubscription* interface to listen for *CCNEncFilter* objects published on a dedicated *subscriptionUri*. Subscribers respond to the broker's Interest with their encrypted filters. A *CCNEncFilter* contains a ccnxUri and the encrypted conditions (Step 1). When receiving such a message, the CCNBroker re-encrypts the filter, stores it in the Subscription Store (Step 2), and subscribes to the ccnxUri of the filter.

Publishers that have content matching the ccnxUri of interest, respond with an AesKPEvent message (Step 3). The CCNBroker receives this messages on a *readEvent* interface. When a new message arrives, the broker retrieves from the Subscription Store all filters indexed under a ccnxUri that is a prefix of the ccnxUri of the event. The broker identifies the filters with encrypted conditions that are matched by the attributes of the event (Step 4). For each filter that matches, the broker re-publishes the message on an ccnxUri specific to the subscriber that issued the encrypted filter. The subscriber then decrypts the message using KP-ABE (Step 5).

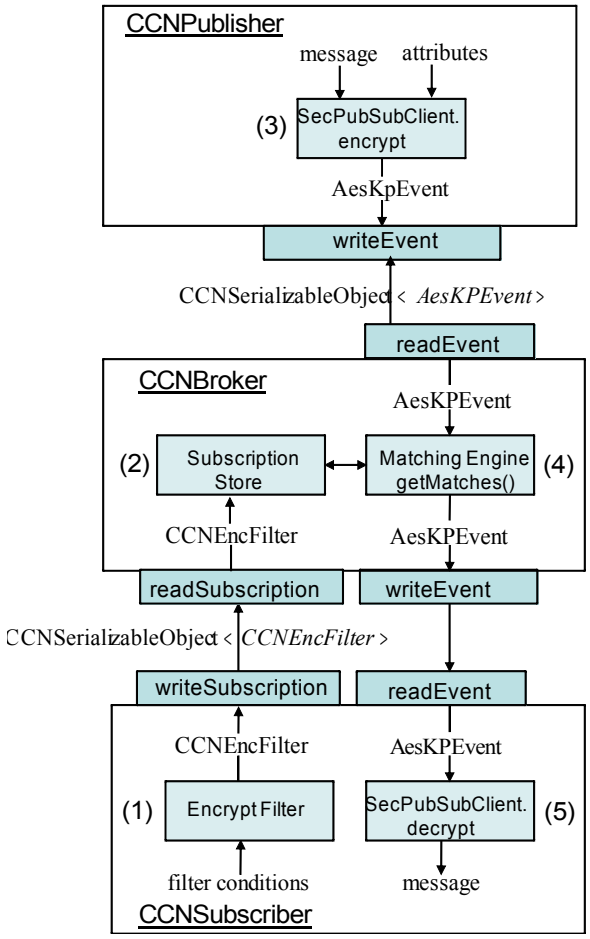


Figure 7.5: Encrypted routing over CCNx.

The application can be distributed to form a graph topology in which internal CCNx nodes run a CCNBroker application. A broker behaves as a publisher or subscriber, or both, to its neighboring brokers.

7.4 Integration with PADRES

In the following we show how our libraries can work with PADRES [Jacobsen 2010], a popular pub/sub system, more sophisticated and mature than CCNx.

7.4.1 PADRES

We integrated our scheme with the Publish/Subscribe Applied to Distributed Resource Scheduling (PADRES) middleware. PADRES is a very scalable pub/sub system designed for large-scale event management applications. PADRES is designed for event-driven enterprise applications such as supply chain and logistics, workflows, business processes and job scheduling, RFID and sensor networks, and

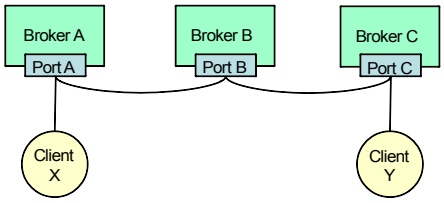


Figure 7.6: A simple PADRES network.

service oriented architectures. In these applications, an event triggers the execution of other events and jobs, which makes a scalable and reliable content-based routing middleware such as PADRES very suitable for event delivery.

PADRES consists of a set of clients connected by brokers organized in a peer-to-peer overlay network. Clients can connect to brokers through different binding interfaces such as Java Remote Method Invocation (RMI) and JMS. Figure 7.6 shows a simple PADRES network in which two clients are connected by a network of 3 brokers. Both clients can subscribe to events and publish events. A PADRES subscription is a conjunction of predicates, where a predicate has the form *[attribute,operator,value]*. Messages have a mandatory tuple describing the *class* of the message. Message routing is based on the publish-subscribe-advertise model from SIENA [Carzaniga 2001]. In order to publish an event, a publisher needs to advertise first the class and attributes of the event. Advertisements are used to create the Subscription Routing Table (SRT), based on which subscriptions are forwarded. The SRT is a list of *[advertisment, last hop]* tuples. A subscription is forwarded to a next hop if it overlaps an advertisement in the SRT. Hence, a subscription is routed hop by hop to the last hop broker that sent an advertisement that matches the subscription. Subscriptions are used to create Publication Routing Tables (PRT), used to route publications. The PRT consists of tuples of the form *[subscription, last hop]*. If a publication matches a subscription, it will be forwarded hop by hop until it reaches the subscriber. Brokers route publications to subscribers by matching them against the registered subscriptions. To efficiently match subscriptions, PADRES implements Rete [Forgy 1982], an efficient pattern matching algorithm.

7.4.2 Confidential PADRES

PADRES does not provide any confidentiality mechanisms, instead all messages are sent unencrypted between clients and brokers, and brokers have full access to the content of events and filters.

We extended PADRES to provide confidentiality of events and filters. We added a third entity to the PADRES network, the Key Management Authority (KMA), and extended the functionality of the PADRES broker and client. We also extended the *Publication* and *Subscription* data types to support sending encrypted publications (or events) and subscriptions (or filters).

The KMA is responsible for generating the public parameters and the secret keys for encryption and decryption. When started, the KMA runs the $Init(1^k)$ algorithm described in Section 6.4. The KMA which needs to be started before starting the brokers and clients exposes API for getting the public parameters and keys, generating the SDE key pairs when a new user joins the system, requesting a decryption key corresponding to a filter, and requesting a decryption key for the client's attributes.

We extended the functionality of the PADRES broker and client as follows. We enriched the broker with functionality for adding a new client and revoking a client. When a client is added, the broker stores the corresponding server-side key of the client, and when a client is revoked, the broker simply removes the client's key pair from the key store. Figure 7.7 shows a simplified architecture of the extended PADRES broker. Messages received are passed to the *Input queue*. The input queue processes messages sequentially, sending them one by one to the *Router*. The router consists of a *Preprocessor*, a *Matching engine*, a *Forwarder* and a *Post-processor*. We modify the functionality of the *Router* and create a new class *EncRouter* with the following modifications. First, we modify the *PreProcessor* to re-encrypt the publication and subscription. This operation is performed only by the broker directly connected with the client. Second, we add another matcher to the matching engine, *EncMatcher* which matches encrypted publications against encrypted subscriptions.

Finally, we extend the client with encryption and decryption functionality.

Our extended implementation can work both with encrypted and non-encrypted messages. In fact, control messages that allow brokers and clients to discover each other and build the network topology are sent unencrypted. We only encrypt application-specific events and filters.

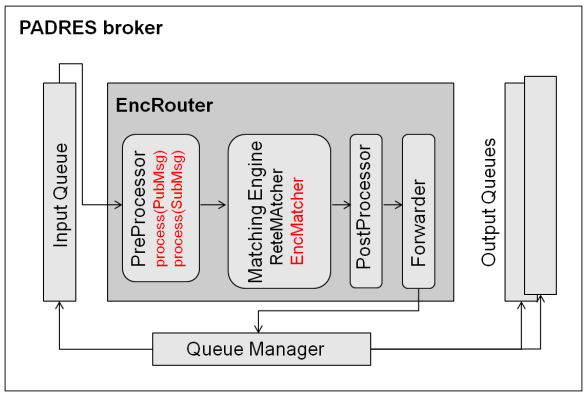


Figure 7.7: PADRES router extended with encryption functionality.

7.4.3 Using advertisements with PADRES

Events in PADRES have a compulsory attribute called *class*, which is the first attribute of an event. The class defines the message type. We create an application

in which events have a class attribute and publishers first send Advertisements of the class. This allows brokers to create more efficient routing tables. Instead of forwarding all subscriptions to all neighbouring brokers, a broker only forwards subscriptions to a neighbour that send an advertisement for the class of the subscription.

We simulate the EV charging scenario with and without advertisements. We use 4 clients and 3 brokers, organized in the topology shown in Figure 7.8.

Figure 7.8: EV Charging Scenario.

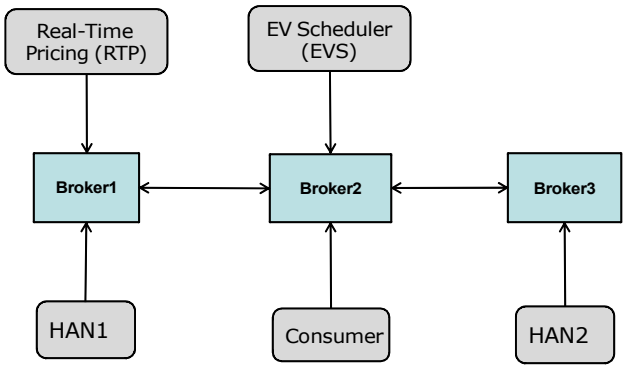


Figure 7.9 shows the routing tables when no advertisements are used. The brokers simply broadcast all subscriptions to each other.

Figure 7.9: Routing tables without advertisements.

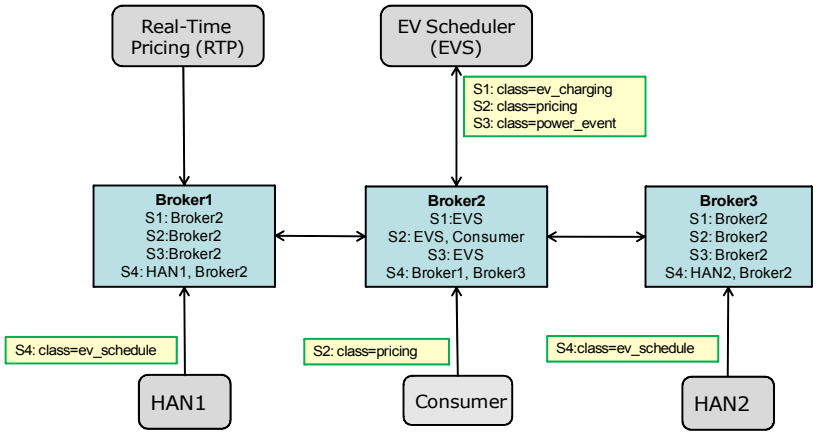
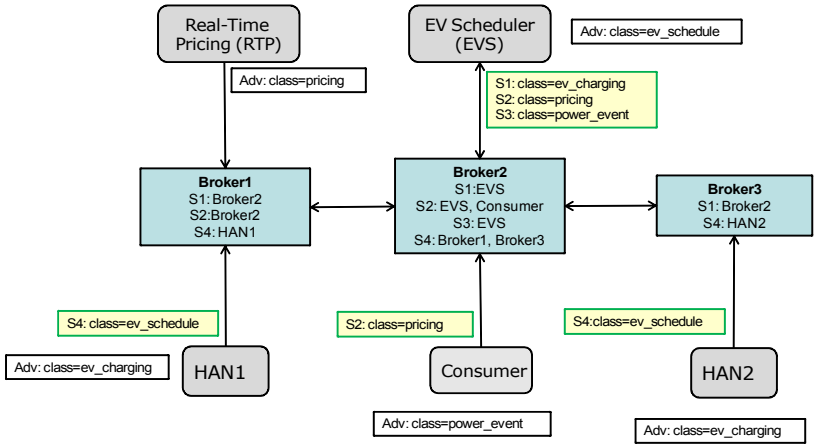


Figure 7.10 shows the routing tables when advertisements are used. In this case, a broker only forwards a subscription to a neighbour broker if it received an advertisement from that broker matching the subscription. The size of the routing tables at Brokers 1 and 3 decreases by 50% for Broker 3 and 25% for Broker 1, thus reducing the event matching time and network load. Thus, when Advertisements are used, the size of the routing tables decreases and as a result, event matching

and routing are more efficient.

Figure 7.10: Routing tables with advertisements.



Chapter 8

Conclusions

As content distribution is becoming the predominant usage of the Internet, content-based networking solutions such as publish/subscribe and Information Centric Networking (ICN) are gaining more and more importance and attention from the research community as a promising solution to the future Internet architecture. The switch from a host-centric to a data-centric Internet, enables consumers to express *what* content they are looking for instead of *where* the content can be found. At the network layer, identifying content rather than location allows more efficient networking and energy efficiency by duplicating and caching content in the network. At the same time, this communication model enables efficient multi-party communication and supports applications where senders and receivers are decoupled. In this thesis we described several such emerging applications like Smart Energy Systems, Smart Cities and remote monitoring of patients in eHealth. An efficient, decoupled communication model is critical for these applications as they evolve from a house, to a neighbourhood and ultimately to a city, country, or global scale.

Though a lot of effort has gone into designing efficient routing and caching solutions, security challenges have only been marginally addressed and the proposed solutions have many limitations, the most important one being the lack of scalability, as most solutions require establishing and exchanging secret keys between publishers and subscribers. This approach does not scale because in many cases subscribers are not aware of the sources of information, and hence, establishing shared keys is infeasible. Additionally, mobile and energy saving devices may become unavailalbe making synchronous communication between servers and receivers impossible. Moreover, as users join and leave the system, re-keying would add too much overhead and be too costly, thus reducing the efficiency of the system.

As the usage of mobile devices, body sensors, smart meters, ambient sensors, and security cameras increases, so does the amount of sensitive data generated by them that needs greater protection. The future Internet requires both better networking solutions able to deliver data between decoupled entities and make data available in the network, but also requires better security and privacy solutions targeted for this new communication model. Securing the future Internet requires content-centric security and privacy solutions, thus moving the protection mechanisms closer to the data and making them data specific.

In this thesis we find that our attribute-based approach plays a major role at many levels. Attributes are essential to naming and data routing as they describe the content of the data. They are also essential to describing the access rights of each

user, thus they support fine-grained access control policies. Moreover, attributes describe users through credentials or roles. Our attribute-based approach supports data confidentiality and enables fine-grained access control policies, while at the same time meeting scalability requirements by not requiring shared keys between publishers and subscribers.

One of the major contributions of this thesis is a novel solution for providing confidentiality and fine-grained access control policies in pub/sub systems. Our solution embeds security mechanisms in the data itself and the keys instead of relying on secure communication channels or trusted third party for enforcing data access policies. We achieve this by designing an encryption scheme based on KP-ABE, CP-ABE and multi-user SDE. Our scheme supports both publication and subscription confidentiality while at the same time eliminating the need for publishers and subscribers to share secret keys. Moreover, our scheme allows publishers to define additional constraints in the form of CP-ABE policies about who can access the content of events. Another novel contribution of our scheme is support for complex encrypted filters able to express conjunctions and disjunctions of equalities and inequalities. Although events and filters are encrypted, brokers can still perform event filtering without learning any information about events or filters. We demonstrated how to apply our scheme to a real-world e-health application that provides confidentiality of the data exchanged over a pub/sub system.

Furthermore we showed how our scheme can be used to provide confidentiality of in-network cached data, an important requirement for ICN, and how to support complex queries on encrypted databases in a multi-user setting. An extensive state-of-the art review revealed that our scheme is the first one to support both complex queries and multi-users who are able to read and write to the databases without sharing keys. An inference exposure analysis of our index showed that our scheme slightly increases the exposure as compared with direct encryption, but with the advantage of supporting exact range queries. The exposure of the scheme is very low and is acceptable when the attacker only knows frequency information of plaintext values and has access to the whole encrypted database. A stronger threat model in which the attacker knows both the encrypted and unencrypted database would require inference protection mechanisms.

Finally, we implemented our schemes as a set of middleware-agnostic libraries and integrated them not only with a distributed pub/sub system called PADRES, but also with a popular Information Centric Networking implementation called CCNx. By using an indexing solution with our encrypted matching scheme, and encrypted Advertisments to reduce the size of the routing tables, our scheme exhibits orders of magnitude greater scalability and can work with large-scale pub/sub systems that need to match thousands of filters in a range of milliseconds.

As future work, more security properties could be added to address other kind of attacks. For example, to protect against active attackers that corrupt messages, an integrity solution can be added. An integrity solution could provide mechanisms for each user to sign publications and subscriptions using the unique key from our scheme that is assigned to each user when the user joins the system. To protect

against dishonest publishers, an Accountability mechanism that enables subscribers to rate content could be designed. Another important open problem is analysing the attributes and access policies that specific applications require. Such analysis could reveal the number of attributes that are used in the system and per event, the complexity of the policies, and how to automate part of the system setup to increase usability by relieving the user of the burden of setting up policies and configuring devices. Furthermore, the performance of our scheme and the added overhead in terms of processing time and bandwidth can only be assessed properly when clear application specific requirements are available. Several on-going projects such as Pecan Street could provide insights into user behaviour and application requirements.

Appendix A

Appendix

A.1 Appendix Publications

Journals

- **M. Ion**, G. Russello, and B. Crispo, "Design and Implementation of a Confidentiality and Access Control Solution for Publish/Subscribe Systems", *Elsevier Computer Networks (COMNET)*, February, 2012.
- M. R. Asghar, **M. Ion**, G. Russello, and B. Crispo, "ESPOON_ERBAC: Enforcing Security Policies in Outsourced Environments", *Elsevier Computers & Security (COSE)*, December, 2012.

Conferences

- M. R. Asghar, **M. Ion**, G. Russello, and B. Crispo, "ESPOON: Enforcing Encrypted Security Policies in Outsourced Environments". In *Proceedings of the Sixth International Conference on Availability, Reliability and Security (ARES)*, 22-26 August 2011, p. 99-108. IEEE, 2011.
- M. R. Asghar, **M. Ion**, G. Russello, and B. Crispo, "Securing Data Provenance in the Cloud", In *Proceedings of IFIP iNetSec*, 2011.
- **M. Ion**, G. Russello, and B. Crispo (2010). "Supporting publication and subscription confidentiality in pub/sub networks". *Security and Privacy in Communication Networks*, 272-289.
- **M. Ion**, G. Russello, and B. Crispo, "Providing Confidentiality in Content-based Publish/Subscribe Systems". In *Proceedings of the International Conference on Security and Cryptography (Secrypt)*, Athens, Greece, July 2010.

Poster/Demo

- **M. Ion**, J. Zhang, and E.M. Schooler, "Toward Content-Centric Privacy in ICN: Attributed-based Encryption and Routing". In *Proceedings of ACM SIGCOMM workshop on Information-Centric Networking (ICN SIGCOMM)*, Hong Kong, August 2013.
- **M. Ion**, G. Russello, and B. Crispo, "Enforcing Multi-user Access Policies to Encrypted Cloud Databases". In *Proceedings of IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, Pisa, Italy, June 2011.

- **M. Ion**, G. Russello, and B. Crispo, "An Implementation of Event and Filter Confidentiality in Pub/Sub Systems and its Application to e-Health". In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, Chicago, IL, October 2010.

Bibliography

[Ahlgren 2012] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher and B. Ohlman. *A survey of information-centric networking.* Communications Magazine, IEEE, vol. 50, no. 7, pages 26–36, 2012. (Cited on page 2.)

[Ashayer 2002] G. Ashayer, H.K.Y. Leung and H.A. Jacobsen. *Predicate matching and subscription matching in publish/subscribe systems.* In Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on, pages 539–546. IEEE, 2002. (Cited on pages 97 and 98.)

[Bacon 2000] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel and M. Spiteri. *Generic support for distributed applications.* IEEE Computer, vol. 33, no. 3, pages 68–76, 2000. (Cited on page 9.)

[Bacon 2008] J. Bacon, D.M. Eyers, J. Singh and P.R. Pietzuch. *Access control in publish/subscribe systems.* In Proceedings of the second international conference on Distributed event-based systems, pages 23–34. ACM, 2008. (Cited on pages 4, 50 and 68.)

[Baek 2008] Joonsang Baek, Reihaneh Safavi-Naini and Willy Susilo. *Public Key Encryption with Keyword Search Revisited.* In ICCSA (1), pages 1249–1259, 2008. (Cited on pages 77 and 79.)

[Banavar 1999] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom and D. Sturman. *An efficient multicast protocol for content-based publish-subscribe systems.* In International Conference on Distributed Computing Systems, volume 19, pages 262–272. IEEE COMPUTER SOCIETY PRESS, 1999. (Cited on pages 9 and 96.)

[Bao 2008] Feng Bao, Robert H. Deng, Xuhua Ding and Yanjiang Yang. *Private query on encrypted data in multi-user settings.* In ISPEC'08: Proceedings of the 4th international conference on Information security practice and experience, pages 71–85, Berlin, Heidelberg, 2008. Springer-Verlag. (Cited on pages 28, 32, 77 and 81.)

[Barazzutti 2012] R. Barazzutti, P. Felber, H. Mercier, E. Onica and E. Rivière. *Thrifty privacy: efficient support for privacy-preserving publish/subscribe.* In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, pages 225–236. ACM, 2012. (Cited on page 101.)

[Belenkiy 2009] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya and H. Shacham. *Randomizable proofs and delegatable anonymous credentials.* Advances in Cryptology-CRYPTO 2009, pages 108–125, 2009. (Cited on page 80.)

[Bellare 2003] M. Bellare, A. Boldyreva and J. Staddon. *Multi-Recipient Encryption Schemes: Security Notions and Randomness Re-Use.* In PKC 2003: Public Key Cryptography, volume 2567. Springer-Verlag, 2003. (Cited on page 69.)

[Bethencourt 2007] John Bethencourt, Amit Sahai and Brent Waters. *Ciphertext-Policy Attribute-Based Encryption.* In Proceedings of the 2007 IEEE Symposium on Security and Privacy, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society. (Cited on pages 32, 54, 56, 67, 70, 81, 86 and 118.)

[Bittner 2005] S. Bittner and A. Hinze. *On the benefits of non-canonical filtering in publish/subscribe systems.* In Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on, pages 451–457. IEEE, 2005. (Cited on pages 95, 96, 97 and 98.)

[Boldyreva 2009] A. Boldyreva, N. Chenette, Y. Lee and A. Oâneill. *Order-preserving symmetric encryption.* Advances in Cryptology-EUROCRYPT 2009, pages 224–241, 2009. (Cited on page 79.)

[Boldyreva 2011] A. Boldyreva, N. Chenette and A. OâNeill. *Order-preserving encryption revisited: improved security analysis and alternative solutions.* Advances in Cryptology–CRYPTO 2011, pages 578–595, 2011. (Cited on page 79.)

[Boneh 2004] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky and Giuseppe Persiano. *Public Key Encryption with Keyword Search.* In Christian Cachin and Jan Camenisch, editeurs, Advances in Cryptology - EUROCRYPT 2004, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer Berlin / Heidelberg, 2004. (Cited on pages 77 and 79.)

[Boneh 2007] Dan Boneh and Brent Waters. *Conjunctive, subset, and range queries on encrypted data.* In TCC'07: Proceedings of the 4th conference on Theory of cryptography, pages 535–554, Berlin, Heidelberg, 2007. Springer-Verlag. (Cited on pages 77 and 80.)

[Bornhovd 2002] C. Bornhovd, M. Cilia, C. Liebig and A. Buchmann. *An infrastructure for meta-auctions.* In Advanced Issues of E-Commerce and Web-Based Information Systems, 2000. WECWIS 2000. Second International Workshop on, pages 21–30. IEEE, 2002. (Cited on page 8.)

[Bösch 2011] C. Bösch, R. Brinkman, P. Hartel and W. Jonker. *Conjunctive wildcard search over encrypted data.* In 8th VLDB Workshop on Secure Data Management, SDM 2011, Seattle, WA, USA, 2011. (Cited on pages 77 and 78.)

[Campailla 2001] A. Campailla, S. Chaki, E. Clarke, S. Jha and H. Veith. *Efficient filtering in publish-subscribe systems using binary decision diagrams.*

In Proceedings of the 23rd International Conference on Software Engineering, pages 443–452. IEEE Computer Society, 2001. (Cited on page 97.)

[Canetti 2007] R. Canetti and S. Hohenberger. *Chosen-ciphertext secure proxy re-encryption*. In Proceedings of the 14th ACM conference on Computer and communications security, page 194. ACM, 2007. (Cited on page 25.)

[Cao 2011] N. Cao, C. Wang, M. Li, K. Ren and W. Lou. *Privacy-preserving multi-keyword ranked search over encrypted cloud data*. In INFOCOM, 2011 Proceedings IEEE, pages 829–837. IEEE, 2011. (Cited on pages 77 and 80.)

[Carzaniga 2001] A. Carzaniga, D.S. Rosenblum and A.L. Wolf. *Design and evaluation of a wide-area event notification service*. ACM Transactions on Computer Systems (TOCS), vol. 19, no. 3, pages 332–383, 2001. (Cited on pages 1, 9, 31, 96 and 128.)

[Carzaniga 2003] A. Carzaniga and A.L. Wolf. *Forwarding in a content-based network*. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pages 163–174. ACM, 2003. (Cited on pages 95, 97 and 98.)

[Carzaniga 2011] A. Carzaniga, M. Papalini and A.L. Wolf. *Content-based publish/subscribe networking and information-centric networking*. In Proceedings of the ACM SIGCOMM workshop on Information-centric networking, pages 56–61. ACM, 2011. (Cited on page 2.)

[Ceselli 2005] A. Ceselli, E. Damiani, S.D.C.D. Vimercati, S. Jajodia, S. Paraboschi and P. Samarati. *Modeling and assessing inference exposure in encrypted databases*. ACM Transactions on Information and System Security (TISSEC), vol. 8, no. 1, pages 119–152, 2005. (Cited on pages xii, 86, 87, 88 and 92.)

[Chand 2003] R. Chand and PA Felber. *A scalable protocol for content-based routing in overlay networks*. In Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on, pages 123–130. IEEE, 2003. (Cited on page 96.)

[Chang 2005] Y.C. Chang and M. Mitzenmacher. *Privacy preserving keyword searches on remote encrypted data*. In Applied Cryptography and Network Security, pages 391–421. Springer, 2005. (Cited on pages 77 and 78.)

[Chen 2010] W. Chen, J. Jiang and N. Skocik. *On the privacy protection in publish/subscribe systems*. In Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on, pages 597–601. IEEE, 2010. (Cited on pages 24, 25, 32 and 96.)

[Cheung 2007] Ling Cheung and Calvin Newport. *Provably secure ciphertext policy ABE*. In CCS '07: Proceedings of the 14th ACM conference on Computer

and communications security, pages 456–465, New York, NY, USA, 2007. ACM. (Cited on page 69.)

[Choi 2010] S. Choi, G. Ghinita and E. Bertino. *A Privacy-Enhancing Content-Based Publish/Subscribe System Using Scalar Product Preserving Transformations.* In Database and Expert Systems Applications, pages 368–384. Springer, 2010. (Cited on pages 24, 25, 32, 96 and 102.)

[Cugola 2002a] G. Cugola, E. Di Nitto and A. Fuggetta. *The JEDI event-based infrastructure and its application to the development of the OPSS WFMS.* Software Engineering, IEEE Transactions on, vol. 27, no. 9, pages 827–850, 2002. (Cited on page 8.)

[Cugola 2002b] G. Cugola and H.A. Jacobsen. *Using publish/subscribe middleware for mobile systems.* ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6, no. 4, pages 25–33, 2002. (Cited on page 8.)

[Curtmola 2006a] R. Curtmola, J. Garay, S. Kamara and R. Ostrovsky. *Searchable symmetric encryption: improved definitions and efficient constructions.* In Proceedings of the 13th ACM conference on Computer and communications security, pages 79–88. ACM, 2006. (Cited on page 40.)

[Curtmola 2006b] Reza Curtmola, Juan Garay, Seny Kamara and Rafail Ostrovsky. *Searchable symmetric encryption: improved definitions and efficient constructions.* In CCS '06: Proceedings of the 13th ACM conference on Computer and communications security, pages 79–88, New York, NY, USA, 2006. ACM. (Cited on pages 77 and 78.)

[Dong 2008a] C. Dong, G. Russello and N. Dulay. *Shared and Searchable Encrypted Data for Untrusted Servers.* Lecture Notes in Computer Science, vol. 5094, pages 127–143, 2008. (Cited on pages 25, 28 and 32.)

[Dong 2008b] Changyu Dong, Giovanni Russello and Naranker Dulay. *Shared and Searchable Encrypted Data for Untrusted Servers.* In Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security, pages 127–143, Berlin, Heidelberg, 2008. Springer-Verlag. (Cited on pages 77 and 81.)

[Dong 2011] C. Dong, G. Russello and N. Dulay. *Shared and searchable encrypted data for untrusted servers.* Journal of Computer Security, vol. 19, no. 3, pages 367–397, 2011. (Cited on pages 38, 39, 40, 43, 44 and 118.)

[Eugster 2003] P.T. Eugster, P.A. Felber, R. Guerraoui and A.M. Kermarrec. *The many faces of publish/subscribe.* ACM Computing Surveys (CSUR), vol. 35, no. 2, page 131, 2003. (Cited on pages 1 and 7.)

[Fidler 2005] E. Fidler, HA Jacobsen, G. Li and S. Mankovski. *The PADRES distributed publish/subscribe system.* Feature Interactions in Telecommunications and Software Systems, VIII, 2005. (Cited on page 13.)

[Fontoura 2010] Marcus Fontoura, Suhas Sadanandan, Jayavel Shanmugasundaram, Sergei Vassilvitski, Erik Vee, Srihari Venkatesan and Jason Zien. *Efficiently evaluating complex boolean expressions.* In Proceedings of the 2010 international conference on Management of data, SIGMOD '10, pages 3–14, New York, NY, USA, 2010. ACM. (Cited on pages 96, 97, 98 and 105.)

[Forgy 1982] C.L. Forgy. *Rete: A fast algorithm for the many pattern/many object pattern match problem.* Artificial intelligence, vol. 19, no. 1, pages 17–37, 1982. (Cited on page 128.)

[Fotiou 2012] Nikos Fotiou, Dirk Trossen and George C Polyzos. *Illustrating a publish-subscribe internet architecture.* Telecommunication Systems, vol. 51, no. 4, pages 233–245, 2012. (Cited on page 2.)

[Ghodsi 2011] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan and J. Wilcox. *Information-centric networking: seeing the forest for the trees.* In Proceedings of the 10th ACM Workshop on Hot Topics in Networks, page 1. ACM, 2011. (Cited on page 2.)

[Goh 2003] E.J. Goh. *Secure indexes.* Cryptography ePrint Archive, Report, vol. 216, page 2003, 2003. (Cited on pages 77 and 78.)

[Golle 2004a] P. Golle, J. Staddon and B. Waters. *Secure conjunctive keyword search over encrypted data.* Lecture notes in computer science, vol. 3089, pages 31–45, 2004. (Cited on page 28.)

[Golle 2004b] Philippe Golle, Jessica Staddon and Brent Waters. *Secure Conjunctive Keyword Search over Encrypted Data.* In ACNS 04: 2nd International Conference on Applied Cryptography and Network Security, pages 31–45. Springer-Verlag, 2004. (Cited on pages 77 and 78.)

[Goyal 2006a] V. Goyal, O. Pandey, A. Sahai and B. Waters. *Attribute-based encryption for fine-grained access control of encrypted data.* In Proceedings of the 13th ACM conference on Computer and communications security, page 98. ACM, 2006. (Cited on pages 51, 60, 70, 118 and 119.)

[Goyal 2006b] Vipul Goyal, Omkant Pandey, Amit Sahai and Brent Waters. *Attribute-based encryption for fine-grained access control of encrypted data.* In CCS '06: Proceedings of the 13th ACM conference on Computer and communications security, pages 89–98, New York, NY, USA, 2006. ACM. (Cited on pages 80 and 81.)

[Hacigümüş 2002] H. Hacigümüş, B. Iyer, C. Li and S. Mehrotra. *Executing SQL over encrypted data in the database-service-provider model.* In Proceedings

of the 2002 ACM SIGMOD international conference on Management of data, pages 216–227. ACM, 2002. (Cited on pages 77 and 78.)

[Hapner 2002] M. Hapner, R. Burridge, R. Sharma, J. Fialli and K. Stout. *Java Message Service*. Sun Microsystems Inc., Santa Clara, CA, 2002. (Cited on pages 9 and 31.)

[Heimbigner 2001] D. Heimbigner. *Adapting publish/subscribe middleware to achieve Gnutella-like functionality*. In Proceedings of the 2001 ACM symposium on Applied computing, pages 176–181. ACM, 2001. (Cited on page 8.)

[Hore 2004] B. Hore, S. Mehrotra and G. Tsudik. *A privacy-preserving index for range queries*. In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, pages 720–731. VLDB Endowment, 2004. (Cited on pages 77 and 78.)

[Hore 2011] B. Hore, S. Mehrotra, M. Canim and M. Kantarcioglu. *Secure multidimensional range queries over outsourced data*. The VLDB Journal, pages 1–26, 2011. (Cited on pages 77 and 78.)

[Hwang 2007] Yong Ho Hwang and Pil Joong Lee. *Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System*. In Pairing, pages 2–22, 2007. (Cited on pages 77 and 81.)

[Jacobsen 2010] H.A. Jacobsen, A. Cheung, G. Li, B. Maniymaran, V. Muthusamy and R.S. Kazemzadeh. *The PADRES Publish/Subscribe System*. Principle and Applications of Distributed Event-based Systems. IGI Global, 2010. (Cited on pages 2, 3, 118 and 127.)

[Jacobson 2007] V. Jacobson, M. Mosko, D. Smetters and JJ Garcia-Luna-Aceves. *Content-centric networking*. Whitepaper, Palo Alto Research Center, pages 2–4, 2007. (Cited on pages 2 and 118.)

[Kamara 2012] Seny Kamara, Charalampos Papamanthou and Tom Roeder. *Dynamic searchable symmetric encryption*. In Proceedings of the 2012 ACM conference on Computer and communications security, pages 965–976. ACM, 2012. (Cited on pages 77 and 78.)

[Katz 2008] Jonathan Katz, Amit Sahai and Brent Waters. *Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products*. In Nigel Smart, editeur, Advances in Cryptology EUROCRYPT 2008, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer Berlin / Heidelberg, 2008. (Cited on pages 28, 77 and 80.)

[Khurana 2005] H. Khurana. *Scalable security and accounting services for content-based publish/subscribe systems*. In Proceedings of the 2005 ACM symposium on Applied computing, page 807. ACM, 2005. (Cited on pages 23 and 25.)

[Langheinrich 2000] M. Langheinrich, F. Mattern, K. Romer and H. Vogt. *First steps towards an event-based infrastructure for smart things.* In Ubiquitous Computing Workshop, PACT 2000. Citeseer, 2000. (Cited on page 8.)

[Li 2005] G. Li, S. Hou and H.A. Jacobsen. *A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams.* 2005. (Cited on pages 95, 96, 97, 98 and 100.)

[Li 2011] M. Li, S. Yu, N. Cao and W. Lou. *Authorized private keyword search over encrypted data in cloud computing.* In Distributed Computing Systems (ICDCS), 2011 31st International Conference on, pages 383–392. IEEE, 2011. (Cited on pages 77 and 80.)

[Lu 2011] Y. Lu and G. Tsudik. *Enhancing data privacy in the cloud.* Trust Management V, IFIPTM 2011, pages 117–132, 2011. (Cited on pages 77 and 80.)

[Maji 2012] A.K. Maji and S. Bagchi. *v-CAPS: A Confidentiality and Anonymity Preserving Routing Protocol for Content-Based Publish-Subscribe Networks.* Security and Privacy in Communication Networks, pages 281–302, 2012. (Cited on pages 24 and 25.)

[McKay] Brendan McKay and Adolfo Piperno. *Nauty and Traces: GRAPH CANONICAL LABELING AND AUTOMORPHISM GROUP COMPUTATION.* http://pallini.di.uniroma1.it/. (Cited on pages 93 and 115.)

[McKay 1981] Brendan D McKay. *Practical graph isomorphism.* 1981. (Cited on page 93.)

[Miklos 2002] Zoltan Miklos. *Towards an Access Control Mechanism for Wide-area Publish/Subscribe Systems.* In In International Workshop on Distributed Event-based Systems. IEEE Press, 2002. (Cited on pages 4, 50 and 68.)

[Mühl 2001] G. Mühl. *Generic constraints for content-based publish/subscribe.* In Cooperative Information Systems, pages 211–225. Springer, 2001. (Cited on pages 95, 96 and 100.)

[Nabeel 2009] M. Nabeel, N. Shang and E. Bertino. *Privacy-Preserving Filtering and Covering in Content-Based Publish Subscribe Systems.* Rapport technique, Purdue University, 6 2009. (Cited on pages 24, 25, 32 and 102.)

[Ondo 2006] K. Ondo and M. Smith. *Outside IT: the case for full IT outsourcing.* Healthcare financial management: journal of the Healthcare Financial Management Association, vol. 60, no. 2, page 92, 2006. (Cited on page 4.)

[Ostrovsky 2007] R. Ostrovsky, A. Sahai and B. Waters. *Attribute-based encryption with non-monotonic access structures.* In Proceedings of the 14th ACM conference on Computer and communications security, page 203. ACM, 2007. (Cited on page 69.)

[Paillier 1999] P. Paillier. *Public-key cryptosystems based on composite degree residuosity classes*. In Advances in CryptologyâEUROCRYPTâ99, pages 223–238. Springer, 1999. (Cited on page 79.)

[pec a] *Electric Car Owners All Plug In at Once*. http://www.pecanstreet.org/2012/08/electric-car-owners-all-plug-in-at-once/. (Cited on page 12.)

[pec b] *Pecan Street Project*. http://www.pecanstreet.org/. (Cited on page 12.)

[Pentikousis 2012] Kostas Pentikousis, Prosper Chemouil, Kathleen Nichols, George Pavlou and Dan Massey. *Information-centric networking [Guest editorial]*. Communications Magazine, IEEE, vol. 50, no. 7, pages 22–25, 2012. (Cited on page 1.)

[Popa 2011] R.A. Popa, C. Redfield, N. Zeldovich and H. Balakrishnan. *CryptDB: protecting confidentiality with encrypted query processing*. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pages 85–100. ACM, 2011. (Cited on pages 77 and 79.)

[Raiciu 2006] C. Raiciu and D.S. Rosenblum. *Enabling confidentiality in content-based publish/subscribe infrastructures*. Securecomm and Workshops, vol. 28, pages 1–11, 2006. (Cited on pages 23, 25 and 102.)

[Rhee 2010] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo and Dong Hoon Lee. *Trapdoor security in a searchable public-key encryption scheme with a designated tester*. J. Syst. Softw., vol. 83, no. 5, pages 763–771, 2010. (Cited on pages 77 and 79.)

[Ristenpart 2009] Thomas Ristenpart, Eran Tromer, Hovav Shacham and Stefan Savage. *Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds*. In Proceedings of the 16th ACM conference on Computer and communications security, pages 199–212. ACM, 2009. (Cited on page 18.)

[Segall 2000] B. Segall, D. Arnold, J. Boot, M. Henderson and T. Phelps. *Content based routing with elvin4*. In Proceedings of AUUG2K, 2000. (Cited on page 97.)

[Shao 2010] Jun Shao, Zhenfu Cao, Xiaohui Liang and Huang Lin. *Proxy re-encryption with keyword search*. Inf. Sci., vol. 180, no. 13, pages 2576–2587, 2010. (Cited on pages 77 and 81.)

[Shen 2009] E. Shen, E. Shi and B. Waters. *Predicate privacy in encryption systems*. Theory of Cryptography, pages 457–473, 2009. (Cited on page 80.)

[Shikfa 2009] A. Shikfa, M. Onen and R. Molva. *Privacy-Preserving Content-Based Publish/Subscribe Networks*. In Emerging Challenges for Security, Privacy

and Trust: 24th Ifip Tc 11 International Information Security Conference, SEC 2009, Pafos, Cyprus, May 18-20, 2009, Proceedings, page 270. Springer, 2009. (Cited on pages 21, 23, 25, 96 and 101.)

[Singhera 2008] Z.U. Singhera. *A workload model for topic-based publish/subscribe systems*. 2008. (Cited on page 8.)

[Song 2000a] Dawn Xiaoding Song, D. Wagner and A. Perrig. *Practical techniques for searches on encrypted data*. pages 44 –55, 2000. (Cited on pages 77 and 79.)

[Song 2000b] D.X. Song, D. Wagner and A. Perrig. *Practical techniques for searches on encrypted data*. In 2000 IEEE Symposium on Security and Privacy, 2000. S&P 2000. Proceedings, pages 44–55, 2000. (Cited on pages 25 and 28.)

[Srivatsa 2005] M. Srivatsa and L. Liu. *Securing publish-subscribe overlay services with EventGuard*. In Proceedings of the 12th ACM conference on Computer and communications security, pages 289–298. ACM, 2005. (Cited on pages 96 and 101.)

[Srivatsa 2007] M. Srivatsa and L. Liu. *Secure event dissemination in publish-subscribe networks*. In Proceedings of the 27th International Conference on Distributed Computing Systems, page 22. Citeseer, 2007. (Cited on pages 21, 23 and 25.)

[Triantafillou 2004] P. Triantafillou and A. Economides. *Subscription summarization: A new paradigm for efficient publish/subscribe systems*. In Distributed Computing Systems, 2004. Proceedings. 24th International Conference on, pages 562–571. IEEE, 2004. (Cited on page 100.)

[Wang 2006] H. Wang and L.V.S. Lakshmanan. *Efficient secure query evaluation over encrypted XML databases*. In Proceedings of the 32nd international conference on Very large data bases, pages 127–138. VLDB Endowment, 2006. (Cited on pages 77 and 78.)

[Whang 2009] S.E. Whang, H. Garcia-Molina, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Vee and R. Yerneni. *Indexing boolean expressions*. Proceedings of the VLDB Endowment, vol. 2, no. 1, pages 37–48, 2009. (Cited on pages 95, 97 and 98.)

[Xylomenos 2012] G. Xylomenos, X. Vasilakos, C. Tsilopoulos, V.A. Siris and G.C. Polyzos. *Caching and mobility support in a publish-subscribe internet architecture*. Communications Magazine, IEEE, vol. 50, no. 7, pages 52–58, 2012. (Cited on page 2.)

[Yang 2011] Y. Yang, H. Lu and J. Weng. *Multi-User Private Keyword Search for Cloud Computing*. In 2011 Third IEEE International Conference on Cloud

Computing Technology and Science, pages 264–271. IEEE, 2011. (Cited on pages 77 and 79.)

[Zhang] J. Zhang, Q. Li and E.M. Schooler. *iHEMS: An Information-Centric Approach to Secure Home Energy Management.* (Cited on page 2.)

[Zhu 2011] B. Zhu, B. Zhu and K. Ren. *Peksrand: Providing predicate privacy in public-key encryption with keyword search.* In Communications (ICC), 2011 IEEE International Conference on, pages 1–6. IEEE, 2011. (Cited on pages 77 and 79.)

[Zhuang 2001] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz and J.D. Kubiatowicz. *Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination.* In Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, page 20. ACM, 2001. (Cited on page 8.)

1