July 6, 2009

Contents

Chapter 1

Abstract

Publish-subscribe supports asynchronous interactions among processes in a distributed system. A process can describe its interest in messages by performing an operation called subscribe and will be notified about messages which match the specific interest.

Provision of basic security mechanisms such as authentication of publishers and subscribers and confidentiality of events and subscriptions is difficult in a publish-subscribe system.

Authentication is difficult to achieve due to the decoupled nature of interactions between the publishers and subscribers. Similarly confidentiality conflicts with the content based routing. Moreover, confidentiality is harder to address in broker-less environment, where the subscribers are clustered according to their interest.

In this thesis, new techniques to provide confidentiality and authentication in a broker-less content-based publish-subscribe built on P2P architecture are presented. Identity-based-encryption is used to provide authentication of publisher and subscriber and confidentiality of events. Furthermore, an algorithm is designed to cluster subscribers according to their subscriptions while preserving a weaker notion of confidentiality. Evaluation results show the feasibility of the technique in terms of dissemination latencies and message overhead.

Chapter 2

Introduction

Publish subscribe system is defined as "a wide-area communication infrastructure that allows information dissemination across geographically distributed and potentially unlimited number of publishers and subscribers" [2] , the roles in this system can be classified into three categories:

1. Publisher: Producer of information *(events)* which sent to the network. These events will be disseminated towards its destinations.
2. Subscriber: The consumer of the information sent by publishers. Subscribers specify their interest in the form of *(subscriptions)* and send it to the network.
3. Broker: Responsible for information *(events)* dissemination to reach the right consumers (subscribers) based on the subscriptions of these subscribers.

An important characteristic of the publish-subscribe system is the decoupling between publishers and subscribers in time, space, and flow. This decoupling improves the scalability and the effectiveness of the system. There are two classifications of publish-subscribe system:

- Topic-based Publish-Subscribe System: In topic-based system the event sent by the publishers marked with a topic and the subscribers subscribe to their interested topics, brokers perform matching based on the topic to route these events to the right subscribers.
- Content-based Publish-Subscribe System: Content-based system is more general and powerful model, where the subscribers have the added flexibility of choosing filtering criteria along multiple dimensions, using conditions and thresholds on the contents of the event, rather than being restricted to pre-defined topic fields.

Our interest in this thesis is the content-based publish-subscribe system. In traditional content-based publish-subscribe systems, brokers are organized into acyclic topology, these systems are simple but suffer from many scalability problems such as subscriptions are flooded to maintain the routing state at each broker. In order to gain scalability, self-organization and fault-tolerance, many content-based publish-subscribe systems are built based on peer-to-peer (P2P) architecture where the subscribers are connected to each others according to similarity of their interest.

Consider a publish-subscribe system disseminating confidential medical records of many patients, these secret records must be comprehensible only to legal doctors in the hospital. Patients subscribe to this system in order to receive some medical advices and remedies, patients prefer to keep their record secret and not revealed except to their doctors. There are two important requirements needed in such content-based publish-subscribe: *keeping the subscriptions and events confidential* and *guarantee the authentication of the events.*

Keeping the events confidential in publish-subscribe system refers to preserve the confidentiality of secret attributes in an event from unauthorized subscribers in the system. For example, the secret attribute *patientRecord* in the following event $e = \langle \langle topic, AidsTrial \rangle, \langle age, 30 \rangle, \langle patientRecord, record \rangle \rangle$ should be comprehensible to only a subscriber S who has subscribed for: $f = \langle \langle topic, =, AidsTrial \rangle, \langle age > 25 \rangle \rangle$, but the other subscriber $S^{'}$ with subscription: $f^{'} = \langle \langle topic, =, AidsTrial \rangle, \langle age > 35 \rangle \rangle$ must be unable to resolve this attribute. So the publish-subscribe network should be capable of matching the routable attributes in an event e (topic and age in the above example) against the restriction in the subscription filter f without obtaining any information about the secret attribute *patientRecord.*

Publish subscribe system built on P2P overlay network clusters the subscribers according to their interest, that's to avoid false positive events, for example, all subscribers interested in $\langle \langle topic, cancerRemedy \rangle \rangle$ must be adjacent to each other, *confidentiality of subscriptions* refers to preserve the subscription confidential between the adjacent subscribers, for example, two patients S_1 and S_2 with subscriptions: $f_1 = \langle \langle topic, =, cancerRemedy \rangle, \langle age > 10 \rangle \rangle$, $f_2 = \langle \langle topic, =, cancerRemedy \rangle, \langle age > 30 \rangle \rangle$ might be adjacent nodes on P2P network and they exchange many messages, but no one of them able to discover that the other is a *cancer* patient. Publish subscribe system should be capable of clustering these subscribers while keeping the subscriptions confidential.

Guarantee the authentication of the events refers to able any subscriber authenticates whether the received event is valid or not, e.g. suppose a doctor with a subscription $f_{doctor} = \langle \langle topic, =, AidsTrial \rangle, \langle age > 25 \rangle \rangle$ receives the *patientRecord* for some patient, he must be able to authorize that the received record is valid and right.

It is not trivial to provide confidentiality in content-based system, because it is against the routing mechanism in such a system *(how to route the secret events to subscribers without knowing their subscriptions?)*. On the other hand, achieving authentication is against the decoupling needed in the system *(how the subscriber authorizes the publisher without knowing him and vice versa?)*

The confidentiality and authentication for both publishers and subscribers must be achieved without a big overhead to the scalability and the throughput of the system, the main services and functions must be unaffected. Many algorithms were developed to achieve such security requirements, related-work section shows and discusses that, these solutions have many draw backs against the decoupling between publishers and subscribers and the key management mechanism between them *(renewing all the keys when new subscriber joins or leaves the system)*. Furthermore, no one addressed the issue of subscription confidentiality when subscribers are semantically clustered. In this thesis, new techniques have been developed to achieve the basic security requirements in publish-subscribe system. To verify their feasibilities they have been simulated and evaluated using *peersim* simulator [1].

The rest of this report is organized as follows. Chapter 3 sketches our reference publish-subscribe model and the threat model. Chapter 4 presents the related work. In chapter 5 we introduce the identity based encryption, then we present how to use it in our system in chapter 6. Chapter 7 contains the evaluations and results. And finally chapter 8 concludes the report with view on future work.

Chapter 3

Background And System Model

3.1 Content-Based Publish-Subscribe Model

Peer-to-peer [3] system is a dynamic and scalable set of peers where they could join or leave the system at any time. P2P overlay networks have interested much attention due to desired characteristics for a large-scale distributed environment, it is characterized by the direct sharing of resources among the peers in the network. Our content-based publish-subscribe system is based on P2P overlay network, where the subscribers are clustered according to similarity of their subscriptions. We classified all the nodes in the system into two categories:

1. Publisher: producer of events.
2. Subscriber: consumer of events and responsible for routing events to the right place*(there are no brokers in our system).*

In content-based publish-subscribe system [4] the notification is a set of pairs *(attribute,value)*, and the subscription is a conjunction of tuples *(attribute,operator,value)*, where $operators \in \{=, <, \leq, >, \geq\}$.

We assume content-based publish-subscribe system [3] provides a defined schema with m attributes $\{A_1, A_2, A_3, ...A_m\}$. All these attributes form the event space of the system, for example, a system with two attributes $\{temperature, humidity\}$ where: $0 \leq temperature \leq 100, 0 \leq humidity \leq 50$ has an event space as shown in figure 3.1. In this event space, the publishers set values for their events and subscribers set restrictions to get their interested events, e.g. a subscriber *S1* defines a subscription $[0 \leq temperature \leq 20, 25 \leq humidity \leq 50]$, and a publisher *P1* creates a notification $[temperature = 10, humidity = 40]$, an event is matched a subscription if all attribute values satisfy the restriction condition, the matching of notifications to the set of all sub-

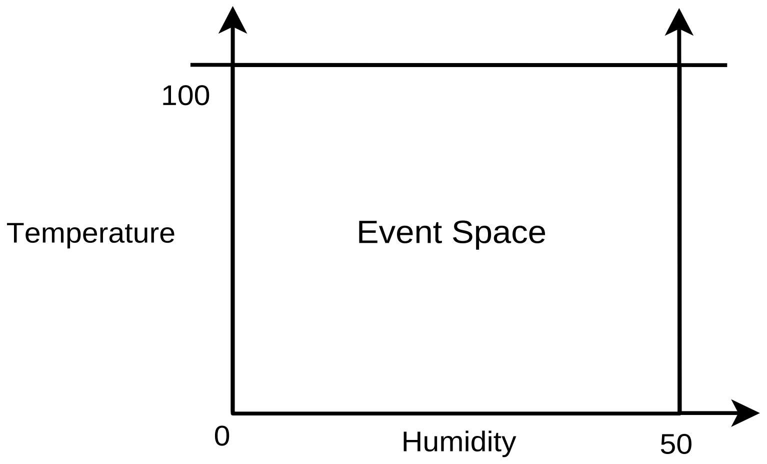


Figure 3.1: Example Of Event Space

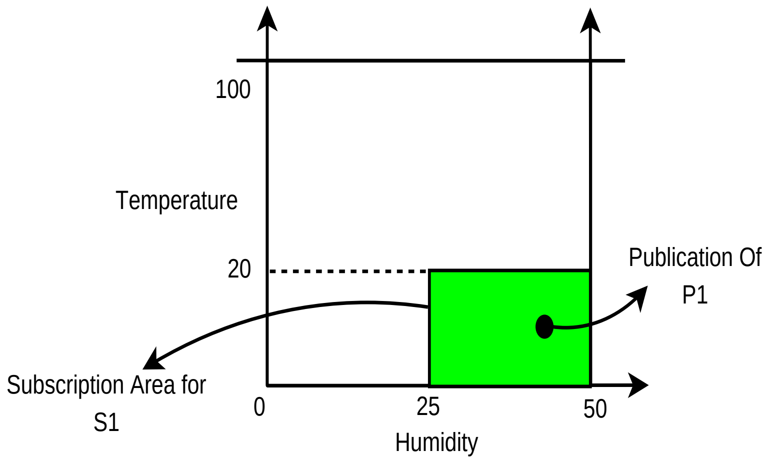


Figure 3.2: Matching between Subscription Area of S1 and Publication of P1

scriptions is done based on the contents (values of attributes), this is shown in figure 3.2.

Claim 3.1 *subscription A is contained by subscription B if: all events matched to subscriber subscribes for A are matched for a subscriber subscribes for B, or* $Region_A \prec Region_B$ *in the event space.*

In figure 3.3 the subscription of the subscriber *S1* contains the subscription of the subscriber *S2* $= [5 \leq temperature \leq 15, 30 \leq humidity \leq 40]$.

The Idea of spatial indexing [4] mechanism in publish-subscribe system is to divide the event space recursively into many regions as shown in figure 3.4, the first vertical line divides the event space into two regions labeled **0 and 1**, the second horizontal line divides the event space again into four regions **00,01,10,11**. The dividing axis is

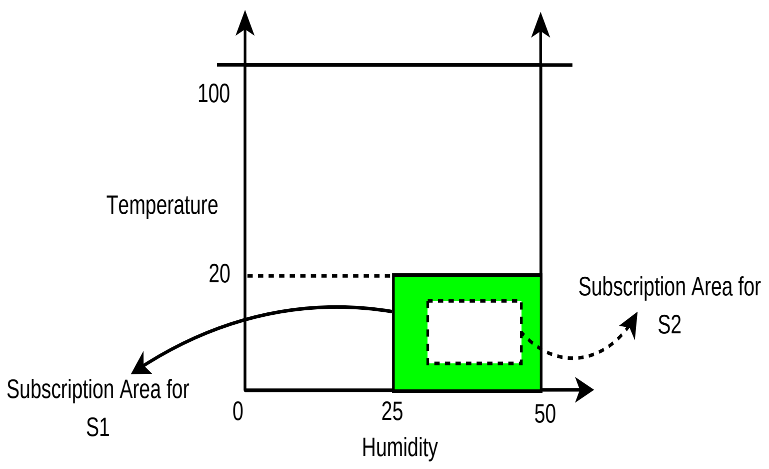


Figure 3.3: Containment Between Subscriptions

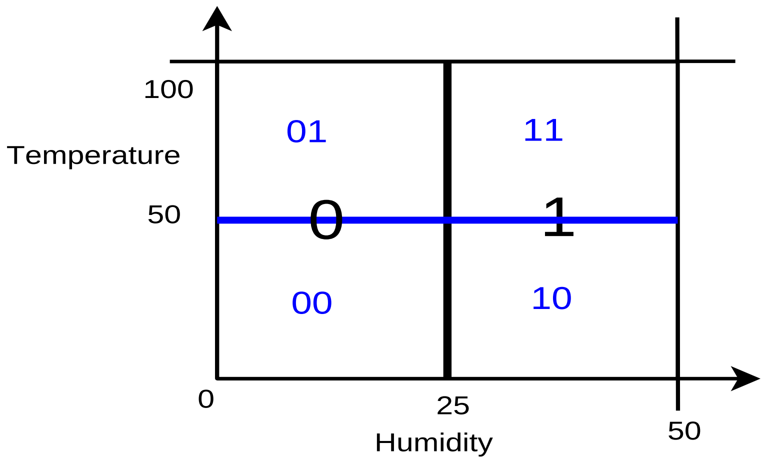


Figure 3.4: Division Of Event Space

selected alternatively (1,2,1,2). Each region is represented by bit string *(DZ)*, this bit string is generated by dividing each axis in the event space alternatively ($temperature_{axis}$, $humidity_{axis}$, $temperature_{axis}$, ...), the size of the region is related to the length of *DZ*, e.g. "110" is smaller than "11" and "1".

This spatial-indexing concept can be applied to any number of dimensions, but for simplicity reasons we discuss 2D spaces here. The mapping from the publish-subscribe domain to the spatial domain is done in two steps:

1. N-dimensional space S is created, where N is the number of unique attributes in the publish-subscribe domain.
2. Every attribute *aj* in the publish-subscribe domain maps to a dimension *dj* in the space S.

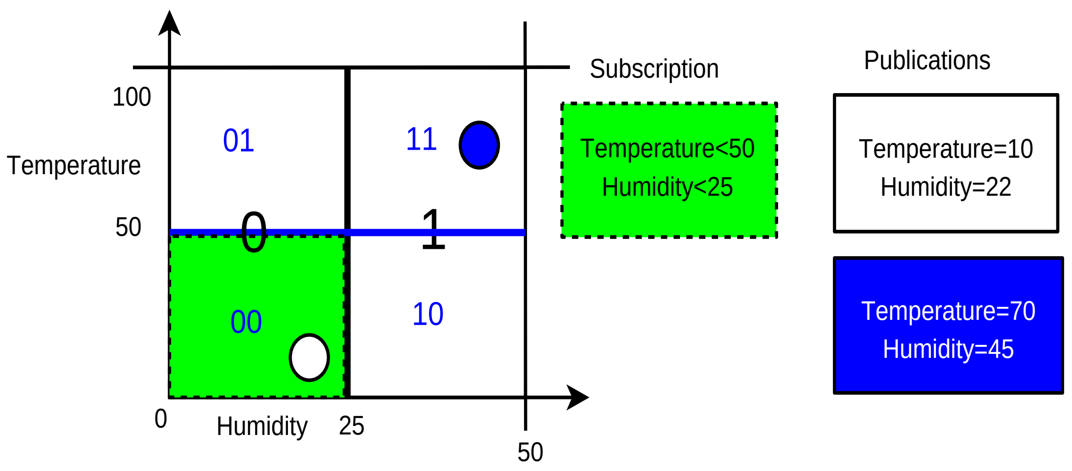


Figure 3.5: Subscriptions And Publications In Divided Event Space

In our system we can perform two basic operations on the *n-dimension* spatial-indexing space:

1. Advertise-to-area : where the publisher decides the area he will publish events on it.
2. Publish-to-point: where the publisher can publish a message at a given point, receivable by all subscribers whose subscription areas cover this point.
3. Subscribe-to-area: where the subscriber can define his area *(bit string (DZ))* of interest in this space, so he will receive all events delivered by the publishers in his subscription area.

In figure 3.5 the subscriber *S1* with subscription [$0 \leq temperature \leq 50$, $0 \leq humidity \leq 25$] specifies his subscription by *"00"*. Publications [$temperature = 10$, $humidity = 22$] and $[temperature = 70, humidity = 45]$ are represented by dots. Events sent by publishers will be delivered to all subscribers whose subscription areas contain the publication dots, so in figure 3.5 the subscriber *S1* will just receive the events related to $[temperature = 10, humidity = 22]$.

Claim 3.2 *if* $n_1 = length(DZ_1)$ *and* $n_2 = length(DZ_2)$ *then:*
if$(n_1 > n_2)$ *and* $DZ_1 == DZ_2$*(for length* n_2 *)* $\rightarrow$ DZ_2 *contains* DZ_1

Figure 3.6 shows 2-Dimensional event space divided recursively into 14 regions ("0", "1", "00"... "11", "000", "111"), subscription region labeled By S_{01} contains both S_{010} and S_{011}. Figure 3.7 shows the containment relation between all regions of subscription space in figure 3.6, the node with *"*"* subscription represents the whole subscription space.

Any subscriber subscribes for any region in figure 3.6 receives all the events related to his subscription and to all regions he contains. Taking into account that each sub-

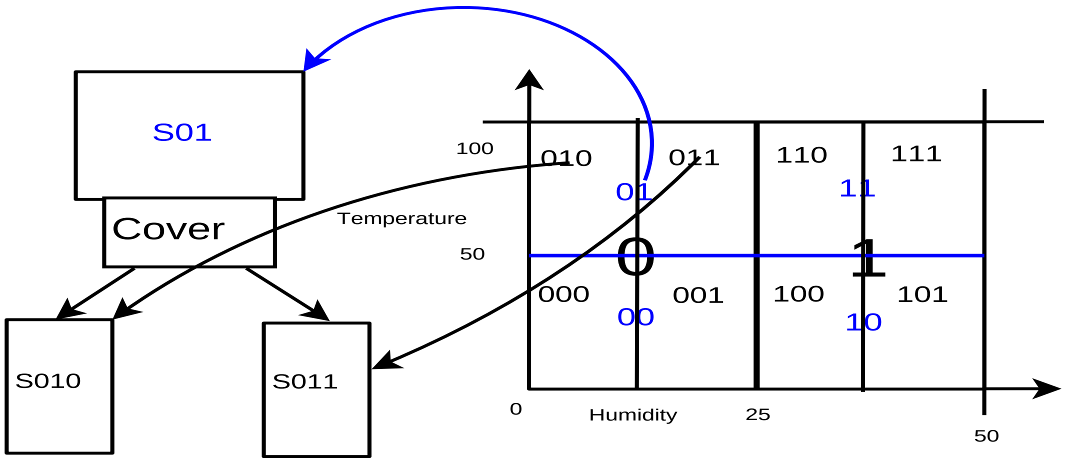
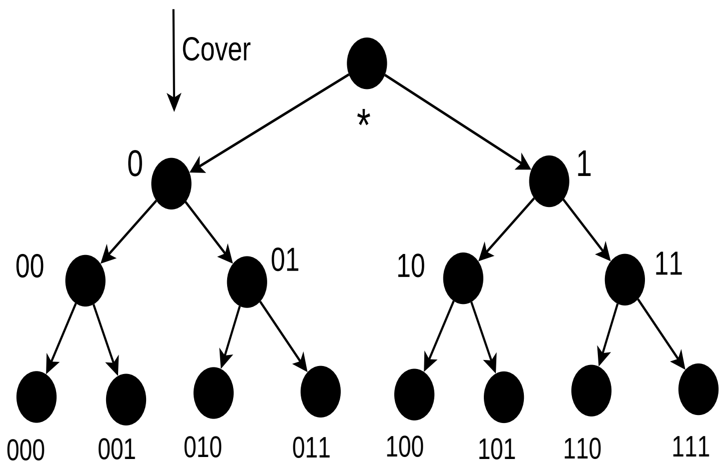


Figure 3.6: Event Space Divided into 14 regions



Subsciber	For	Labeled
S	*	S*
S	101	S101
S	1	S1
S	11	S11
S	0	S0
S	110	S110

Figure 3.7: Covering Tree of 14 Subscription Regions

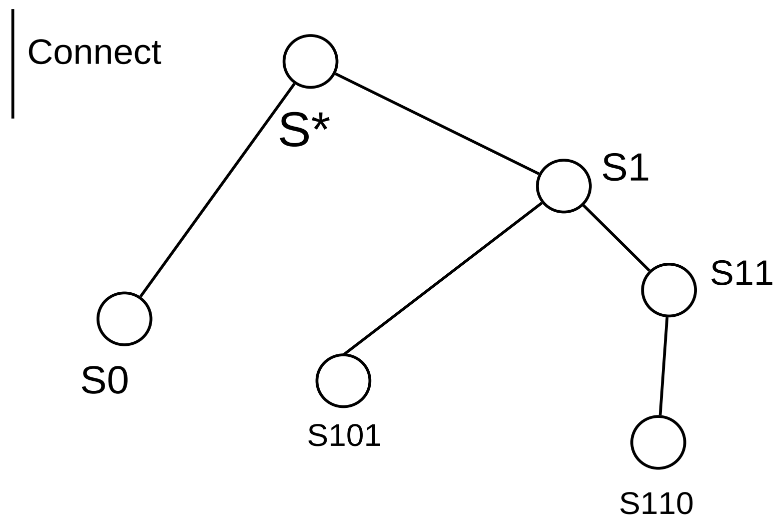


Figure 3.8: Connection Topology Of Subscribers

scriber can connect to *x* subscribers with *1* parent and *(x-1)* children, we can connect the subscribers in a topology like a tree where subscribers with a largest subscription region located higher in the tree, and each parent contains the subscriptions of his children. By this connection topology, we cluster the subscribers according to their subscriptions. Consider the subscribers shown in the table in figure 3.7 with *3* connection space for each subscriber *includes one parent*, the best way to connect these subscribers to each other is shown in figure 3.8.

Definition 3.1 *False Positive Event: an event received by a subscriber but it does not match it's subscription.*

The topology of subscribers in figure 3.8 requires each subscriber knows the subscription of his neighbors, any new subscriber wants to join the system sends his subscription to random connected subscriber, this request will be forwarded through the network until reach the right position to connect. Also when receive any event the subscriber forwards this event to its neighbor based on their subscriptions.

In many content-based publish subscribe-system like the competitive system each subscriber tries to hide his subscription from others, furthermore, each subscriber must be able to authenticate the received events. Actually there are many security requirements needed in many content-based publish-subscribe systems, these requirements which we are trying to achieve in this thesis are:

1. **Subscriptions Confidentiality**: Confidentiality in networks means no illegal node can resolve or read a data not belongs to it, the definition of *illegal node* depends on the system. For our publish-subscribe system subscriptions confidentiality means that the subscription of any subscriber is hidden and unknown to other nodes in the system (subscribers and publishers). *Achieving such subscription confidentiality*

is difficult because it is against the mechanisms of connect new subscriber to the system and disseminate events to its right subscribers, in both of these mechanisms, subscriber must reveal his subscription in order to connect and receive his event).

2. **Publications Confidentiality**: Publications confidentiality guarantees that just the legal subscribers can reveal the sent events. *This requirement is against the routing mechanism in the system. In other words, how any subscriber decides the next hop to forward the event while he cannot resolve it?*

3. **Publications Authentication**: Authentication in networks means that any recieved message is sent by the legal sender, the receiver must be able to detect whether the received message is fake or not. In publish-subscribe system publications authentication means that any received event is just sent by the legal publisher, so there must be an ability to the subscriber to decide whether the received event is sent by the corresponding legal publisher. *This is difficult to achieve while sustaining the decoupling between publishers and subscribers. In other words, how the subscriber can detect whether the identity of the publisher is right even he does not know him?*

4. **Subscriptions Authentication**: Subscriptions authentication in our publish-subscribe system means that no illegal node (publisher or subscriber) can subscribe for some subscription and start receiving its corresponding events. So it must be a mechanism in publish-subscribe system to detect the identity of any subscriber and to check whether he can subscribe for any subscription or not. *This is again difficult to achieve while sustaining the decoupling between publisher and subscribers. In other words, how the publisher can detect the identity of the subscriber who receives his event while he does not know him?*

In case of achieving all these security requirements, we will gain a system that:

1. ***Subscriptions of subscribers are hidden, and no malicious node can fabricate any subscription and pretend as legal subscriber.***

2. ***Publications of publishers are hidden, and no malicious node can fabricate any fake event and send it as legal one.***

3. ***Any fake event will be stopped and will not be propagated through the whole network, so prevent denial-of-service attackers who overwhelm the network with huge number of fake events.***

4. ***Building the subscriber's tree and disseminating the events to the right subscribers while keeping all the subscriptions hidden and without a big overhead in the throughputs and performance latency in the system.***

3.2 Threat Model

Our system comprises of two entities: publishers and subscribers, here we present the threat model and the trust assumptions our new algorithm assumes in all these entities and in the underlying infrastructure network:

1. Publisher: We assume malicious publishers in our system. A malicious publisher might try to fabricate some events, discovers subscriber's subscriptions, discovers other publisher's events by trying to decrypt them or even by claiming that he is a subscriber, e.g. a publisher of "1100" pretends he is a subscriber for "1010" in order to discover the events sent by his competitive publisher.

2. Subscriber: We have an assumption of malicious subscribers in our system. A malicious subscriber might try to fabricate some events, discovers other subscriber's subscription, whether by decrypting events not related to his subscription or even claims that he is a subscriber for a certain event space region, e.g. a subscriber of "101" claims that he is a subscriber for "111".

3. Subscribers are honest in routing nodes between publishers to subscribers, so there is no intended event dropping in the system.

4. Subscribers don't inform others about their decrypted events, or about their secret keys.

5. The underlying IP-network may not provide confidentiality or authentication, but the underlying domain name server, network routers, and the related network infrastructure is secure and cannot be corrupted by an adversary.

6. We are using identity-based-encryption algorithm in our solution, we assume that we have a secure and protected trusted-third-party *(TTP)*, and there is a secure link between this *TTP* and other nodes in the system.

Chapter 4

Related Work

Many research and industrial projects focus on providing confidentiality and authentication in publish-subscribe system, this chapter gives an overview of some of these algorithms.

4.1 EventGuard Mechanism

EventGuard mechanism [2] is a reliable framework and a set of protection mechanisms for securing a publish-subscribe overlay service. It has two components, the first component is a group of security guards that can be plugged-into a wide-area content-based publish-subscribe system. The second component is a resilient publish-subscribe network design that is suitable of secure message routing, preventing denial-of-service attacks and failure of the nodes.

The reference model of EventGuard [2] supports:

1. *Publish, advertise and un-advertise* actions for the publishers. Publisher uses the advertise action to announce that he wants to publish some event, the un-advertise action means that the publisher will not send the corresponding event furthermore.

2. *Subscribe and un-subscribe* actions for the subscriber.

Figure 4.1 shows the architecture of EventGuard system, it is composed of many layers. The lowest one is the network layer, this layer could be TCP or UDP which used to exchange data between the nodes, the resilient publish-subscribe network used to prevent dropping message denial of service attack.

Security guards located in the second layer responsible for achieving many security requirements on different actions on EventGuard system. Figure 4.1 shows that this layer

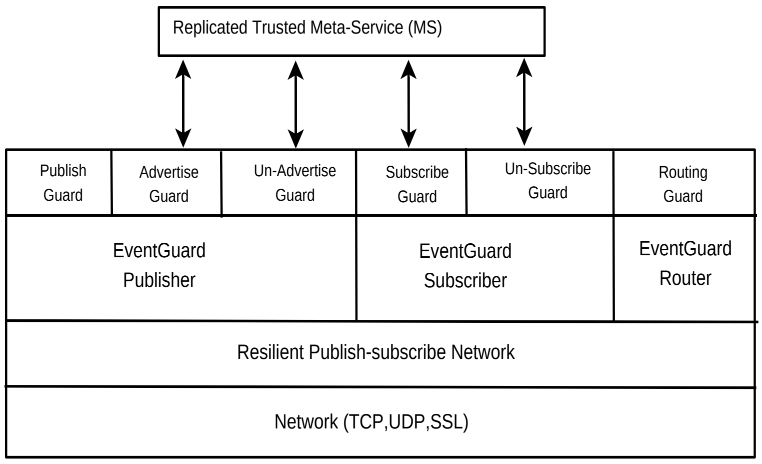


Figure 4.1: EventGuard Architecture [2]

is divided into: publisher, subscriber and router guards.

The higher layer is the replicated trusted Meta-Service. It is responsible for creating some needed blocks for EventGuard such as: tokens, keys and signatures. These blocks are used in different actions to secure the system. Tokens represent the topics in topic-based publish-subscribe system. Publishers in such system publish a token to a topic, and the subscribers subscribe for this token. Keys are essential to encrypt and decrypt messages between publisher and subscriber, and signatures are used to authenticate the received events.

Each security guard is responsible for securing different actions on the system:

1. Subscriber Guard: Subscribers handle the subscription process by corporation with the trusted-meta-service(MS) layer. Subscribers send their filters to the *MS* layer, which responds with a subscription permit, this permit allows the subscriber to securely subscribe for any topic in the system, by this, the subscriber will guarantee authentication, confidentiality of his subscription.

2. Advertise Guard: This guard is responsible for achieving authentication and confidentiality of advertisement action. Publishers gain a permit from the *MS* which used for their advertisements.

3. Publish Guard: Publisher uses the keys and tokens he gained from *MS* to make the transformation of his event to a secure one. Just the legal subscribers who have the corresponding decryption keys able to resolve this event.

4. Un-Subscribe and Un-Advertise guards: Legal subscribers and publishers have an *IDs* which they gained from the *MS* layer. these *IDs* allow them to un-subscribe and un-advertise respectively in a secure manner, so no malicious node able to fabricate any fake un-subscription or un-advertisement message in the system.

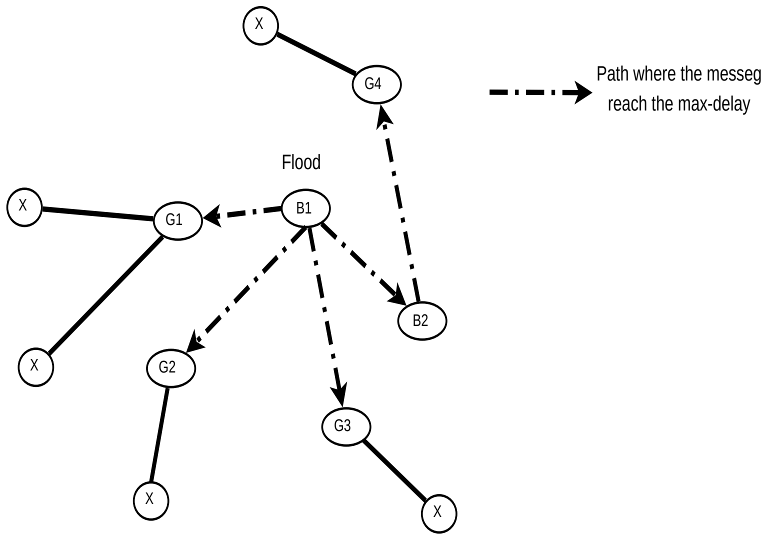


Figure 4.2: Preventing DoS Attack By Routing Guard In EventGuard System [2]

5. Routing Guard: This guard is responsible for preventing denial-of-service attack against publish-subscribe network. Each routing node depends on the time relation to prevent such attack, each EventGuard system message is signed by *MS*, there is also a defined *max-delay* time in the system, the routing nodes stop route any message that exceeds this maximum delay.

   Figure 4.2 illustrates that, in this figure nodes B1 and B2 are malicious nodes, node B1 performs the attack by send huge number of useless messages to other nodes in network. Nodes G1, G2 and G3 prevent and stop these fake messages to be propagated to others, but B2 moves these messages to G4 which stops it, so every *X* labeled node is free of this attack.

6. R-Resilient Network Guard: It is mentioned that EventGuard defines a network architecture in order to stop dropping-based denial-of-service attack. R-resilient network guard is responsible for achieving that. Any network called *r-resilient* if the percentage of the messages that the network can secure them from the dropping attack is *r*. This resilient guard is aim to minimize the number of packets that are affected by the dropping attack, also to reduce the load of the communication on the network. This is done by implement a guard which creates *a-ary* trees to achieve these goals, as a result there is more than one independent path (paths without shared nodes) to route the message from publishers to subscribers

EventGuard mechanism achieves authentication and confidentiality for publishers, subscribers and brokers. However, there are two important drawbacks in it: the first one is that EventGuard mechanism is more suitable for topic-based publish-subscribe system, it creates a token for each used topic. The second one is the unavoidable key management problem in this mechanism. The cost of key management depends on the number of subscribers, so the solution will not be scalable.

4.2 PS-Guard Mechanism

PS-Guard [5] mechanism aims to secure the data exchange between publishers and subscribers in publish-subscribe system. There are two related important factors this algorithm aims to achieve : how to make a secure data exchange while keeping flexible key management mechanism between nodes. Many other algorithms deal with key management by grouping the subscribers according to their interests. PS-Guard tries to find a solution where the key management does not depend on the number of subscribers, it achieves that by using two keys:

1. Authorization key *K(f)*, where *f* is the filter function of the subscriber.
2. Encryption key *K(e)*, where *e* is the event.

These keys *K(e)* & *K(f)* used to encrypt and decrypt the related secrete attributes in the event *e* respectively. This mechanism ensures that if the filter *f* for some subscriber matches an event *e*, then the subscriber can derive *K(e)* by using *K(f)*. The group key *K(e)* is defined by the publisher based on the subscriber's matching filter, so there are no unauthorized nodes could get the secrete attributes for the event *e*.

PS-Guard uses siena publish-subscribe network [5], it also uses the KDC (Key Distribution Center) [5] to handle authorization for the subscription, every subscriber has an authorization key *K(f)* for its corresponding filter.

A time epoch *id* is defined in the system, every period all subscribers renew their permissions, so any subscriber has to have the new *K(f)* for the new period if he interested in the events in the new epoch. This mechanism assumes un-guaranteed underlying network and malicious publisher, subscriber and routing nodes model. For example, malicious publisher attempts to discover publications of other publishers, malicious subscriber tries to reveal events which he does not match, routing nodes are honest in routing data towards its destination.

The main two algorithms used by PS-Guard are the *key management* and the *secure content based event routing*, in the following a description of these algorithms is given:

1. Key Management: The encryption and the authorization keys used to achieve many security goals by this mechanism. Key management algorithm is responsible for creating these keys, the basic idea of key management is to find a key space that each

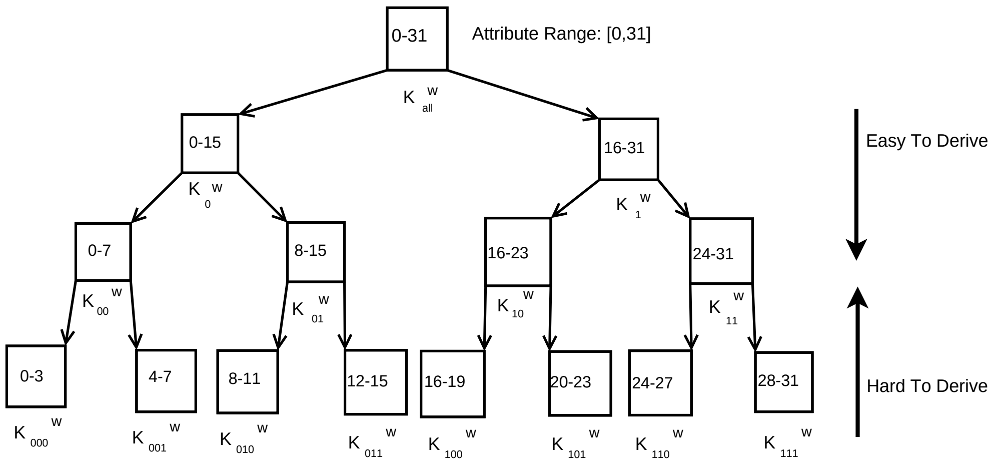


Figure 4.3: Key Tree Example Of Key Space With One Attribute with [0-31] Range [5]

subscriber can use his *K(f)* to find *K(e)*. This is done by using a hierarchal derivation, the produced key space is suitable for different publish-subscribe matching [5]: *string-prefix-suffix* matching, *category-based* matching and *numeric-attribute-based* matching.

Numeric-attribute-based matching supports the range condition between attributes in the event *e*. For example, assume a subscription filter f=$\langle price \in (a, b)\rangle$, this filter match any event e= $\langle price, c\rangle$ if a≤c≤b. So every subscriber can easily derive *K(e)* from *K(f)* if a≤c≤b, otherwise it is very hard to derive *K(e)*. Key management achieves this by map *K(e)* and *K(f)* to a common key space by using numeric-attribute-key-tree (NAKT).

Figure 4.3 shows an example of *key-tree* for a space consisting of one attribute (for simplicity). The range of this attribute is *[0-31]*. The important property of this key tree is: ***it is very easy to derive all child keys from the parent but not the opposite***. Publisher encrypts any event with the corresponding *K(e)* from this tree based of the event space. Subscriber uses his *K(f)* to derive *K(e)* related to his space and his children. As a result, no illegal subscribers can resolve events does not match his subscriptions.

2. Secure Content-Based Event Routing: PS-Guard aims to achieve scalable key management and to exchange the data securely between nodes. There are two techniques used for this secure data exchange, *tokenization* and the *probabilistic multi path event routing*. By applying these techniques the malicious nodes can just discover if there is a match between the event *e* and any subscription. Probabilistic multi path event routing aims to break timing analysis attack, which basically depends on analyze the frequency of the published events to reveal some secure data.

   Tokenization aims to achieve content-based routing, any node can decide if there is any match between the event *e* and the subscription filter *f*. Malicious routing nodes may attempt to break the confidentiality of routed attributes in the publish-subscribe message by observing the frequency of the events that match a given subscription filter. Depending on the distribution's frequency of different events, a malicious node can guess the topic included in the event. This is just happened to the routed attributes, but the secret attributes in an event are unaffected.

A probabilistic multi-path event routing constructs many independent paths used to route data between publishers and subscribers. So there is no clear frequency of the exchanged event. However, if there are many colluding nodes in the network, they are able to corporate and exchange data in order to calculate the right frequency of the exchanged event. To prevent that, the number of the independent paths increased as needed, and the events distributed among them randomly in order to prevent these colluding nodes resolve the event frequency. This technique minimizes the effect of denial-of-service attacks but there is a clear side effect by increasing of the communication load (cost).

PS-Guard guarantees the publication confidentiality because just the legal subscribers have the decryption keys of the encrypted events. However, it does not provide any subscription confidentiality of the subscribers.

4.3 Prefix-Preserving Encryption-Decryption Algorithm

Prefix-Preserving Encryption-Decryption [6] algorithm guarantees confidentiality against eavesdropper nodes in content-based publish-subscribe systems. In this scheme, events and subscription-filters are encrypted meanwhile the publish-subscribe network can route these encrypted events correctly based on the encrypted filters. As a result, these plain-texts are not revealed in the infrastructure for the purpose of security. This algorithm supports interval-matching as a filter function for subscriptions.

By assuming a content-based publish-subscribe systems that allow each filter function to be range-based, composed of intervals mapped to the event space. And by decomposing a subscription with multiple intervals into multiple subscriptions consisting of single ranges, it is sufficient only to consider intervals. Even though this resulted with more subscription cost, but string attributes (like names, not typically consisting of numbers) can be indexed and linearized in some manner. The suitable publish-subscribe model for this scheme is that where the filter function can only be *interval-matching* or *exact-matching* [6].

Interval-matching is defined as a boolean function $f_{[a,b]}\,(y)$, which returns true if and only if $y \in (a, b)$. We assume that a,b and y are all nonnegative integers. The exact-matching is a special case of interval-matching in which a is equal to b.

This schema is targeting private publish-subscribe systems over networks, it is proposed for those publish-subscribe systems in which publishers and subscribers are able to share some secret data, this is practical when publishers and subscribers are all from a same organization. This securing of events and subscriptions must be done while the roting nodes able to make correct routing decisions. Because of these limited targets, any group key distribution protocol can be used like ELK (Efficient Large-Group Key Distribution) [7].

A cryptographic scheme can be defined and used as follows: with a secret key, denoted by k, publishers and subscribers build a one-to-one mapping $F_K : (x_1, x_2, ...x_n) \rightarrow (y_1, y_2, ...y_n)$. Where, $(y_1, y_2, ...y_n)$ are the encrypted events of $(x_1, x_2, ...x_n)$ and F_K is the encrypt function. Before publish an event, publisher encrypts it by applying F_K, so the sent event will be $F_K(event)$. The subscriber generates a set of encrypted events represent his interest and subscribes to the network. Because the secret keys are shared between publishers and subscribers, the encrypted events they generate will match as long as the original events match. So the encrypted events can be routed correctly to subscribers via brokers while the plain-texts of publications and subscriptions are protected, after receiving an encrypted event, subscribers can resolve it because F_K is a one-to-one mapping.

One straightforward way to build the mapping F_K is to pad attributes into a multiple of 64-bit or 128-bit fields that is appropriate for applying standard encryption algorithms like AES or DES [8], by applying the encryption algorithm, we can obtain encrypted attributes, and the mapping F_K is established. The drawback of this simple scheme is that it can only provide exact-matching as a filter function. Interval-matching must be represented by a set of exact-matchings which may generate an intractable number of comparisons for each event. So we need a scheme that efficiently provides interval-matching as a filter function (discussed below).

In the following an overview of this algorithm scheme is given, this is divided into three parts: firstly, how an interval-matching problem can be converted into a set of prefix-matching problems?, then the prefix-preserving encryption and decryption algorithms are presented, finally a description of the protocol that enables content-based routing without revealing the plain-texts to the infrastructure is given:

1. Converting interval-matching problem into a set of prefix-matching problems: the transformation is based on the fact that an arbitrary interval can be mapped into a union of prefix-ranges, where the prefix-range is one that can be expressed by a prefix, e.g. the interval [32, 111], can be represented as 8-bit binary string: $[00100000, 01101111]$ which transformed into prefixes: $\{001*, 010*, 0110*\}$, where $(001*)$ represents $(32, 63)$, $(010*)$ represents $(64, 95)$ and $(0110*)$ represents $(96, 111)$. A recursive algorithm is used to transform any given interval into prefixes, then prefix-preserving encryption-decryption algorithms can be simply used to efficiently provide interval-matching as a filter function while keeping the confidentiality of the

publish-subscribe system.

2. Prefix-preserving encryption and decryption algorithms: This algorithm is depend on the result of transformation interval-matching into prefix-matching, so the routing nodes can route encrypted publications based on encrypted subscriptions, this is done by applying the encryption scheme proposed by Xu, Fan, Moon and Ammar [9] for prefix-preserving IP address anonymization.

   Figure 4.4(a) shows a plain-text tree of 4-bit plain-text. Prefix-preserving encryption function can be described as determining a binary variable for each non-leaf node (including the root node) of the plain-text tree, this variable determines whether the encryption function flips this bit or not. This encryption function rearranges this plain-text tree into a cipher-text tree, figure 4.4(c) shows the cipher-text tree generating from the encryption function shown in figure 4.4(b). For more details about encryption and decryption algorithms refer to [6].

3. Protocol that enables content-based routing without revealing the plain-texts to the infrastructure: Publishers encrypt their attributes using the encryption algorithm before they distribute them. When a subscriber wants to subscribe, he transforms his intervals into the corresponding prefixes using the transformation algorithm, before he subscribes, he applies the encryption function to every prefix. Routing nodes can route encrypted events correctly based on encrypted filter functions.

This algorithm provides a mechanism for routing the encrypted events towards it's destination without resolve them. Brokers able to efficiently make correct routing decisions based on encrypted events and predicate functions. However, an attacker can eavesdrop the connections and download the encrypted messages (publications or subscriptions). If he have certain number of (plain-text, cipher-text) pairs, he will be able to infer information from other cipher-texts by prefix-matching, because the encryption is prefix-preserving. Actually it is more dangerous when the number of (plain-text, cipher-text) pairs gathers at the eavesdropper side, because it will lead to more prefix information of encrypted messages to be revealed.

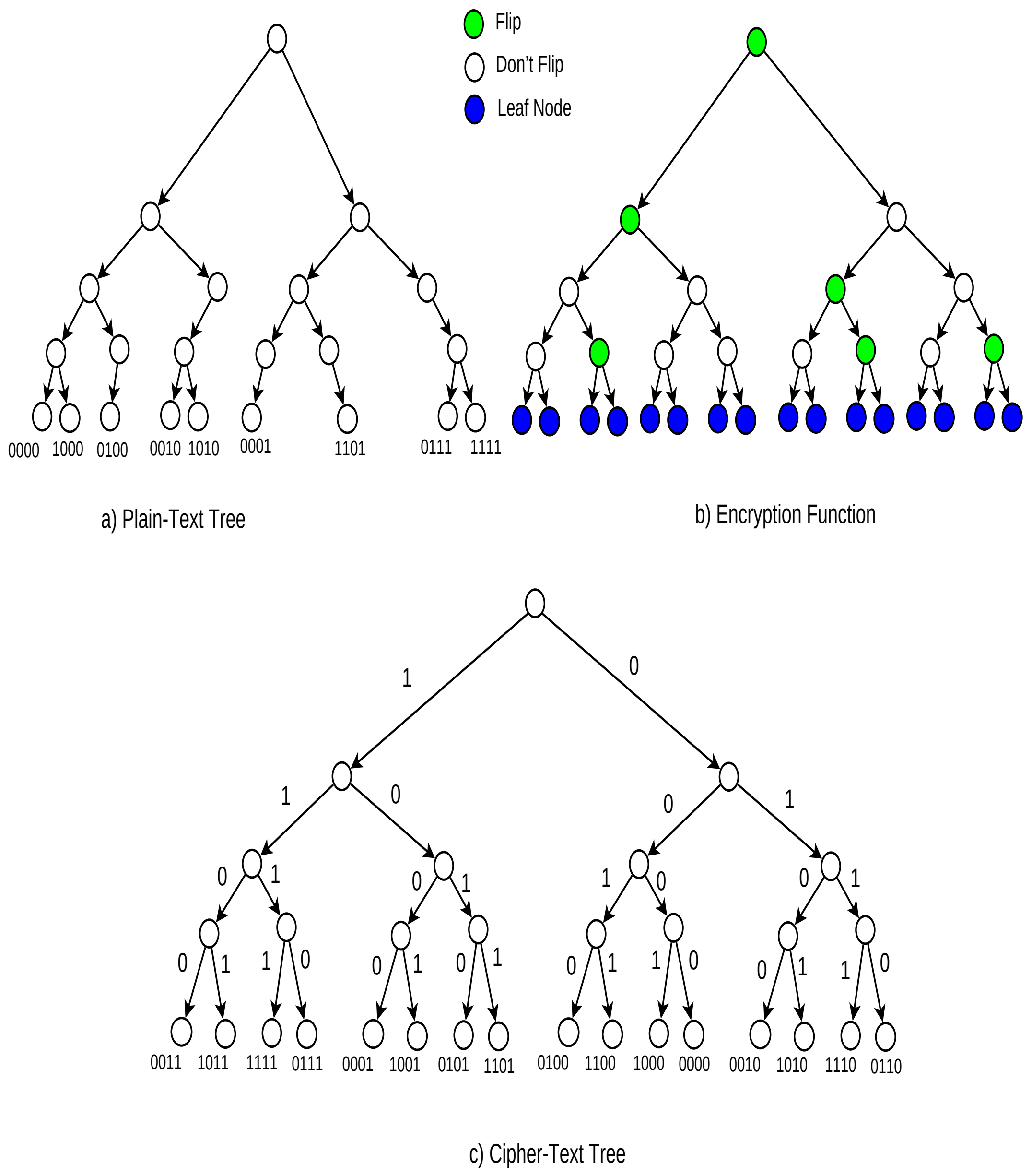


Figure 4.4: Prefix Preserving Encryption Example [6]

4.4 Anonymous Multi-Attribute Encryption with Range Query and Conditional Decryption

In this section, we introduce the concept of Anonymous Multi-Attribute Encryption with Range Query and Conditional Decryption *(AMERQCD)* [10]. In AMERQCD, a plain-text is encrypted under a point in multidimensional space. The resulted cipher-text hides both the plain-text and the point under which it is encrypted. In a range query, the owner of the master key can release the decryption key for an arbitrary hyper-rectangle in space, so, any cipher-texts encrypted under any point within the hyper-rectangle can be decrypted. However, the user who has the decryption key cannot discover any information on cipher-texts outside the range covered by his decryption key.

Searching on encrypted data recently captured a considerable amount of attention in the community [10]. Because it is very important to provide both functionality and privacy in any database application, some researchers proposed encryption schemes that allow keyword-based and specific types of comparison-based searches. AMERQCD provides a powerful technique that enables range query on encrypted data.

AMERQCD assumes the problem of designing an encryption scheme in which data entries consist of a pair *(X,Msg)*, where X is a point in a multi-dimensional lattice U_{Δ}, and *Msg* is an arbitrary string. AMERQCD encrypts the data entries and aims to achieve the followings:

1. Range Query and Conditional Decryption: Upon the request on a region $B \subseteq U_{\nabla}$, the owner of the master key frees a decryption key DK_B such that given a cipher-text C for some entry *(X,Msg)*, using the key DK_B, it is possible to decide whether $X \in B$ and to recover *Msg* from C iff $X \in B$.

2. Confidentiality: Given cipher-text C for entry *(Msg,X)*, it is hard to learn *Msg* from C, given that the decryption key for a region containing X is unknown.

3. Anonymity: Given cipher-text C for entry *(Msg,X)*, suppose a malicious user queried for regions $B_1, B_2, ..., B_q$ all of which do not contain X, then he cannot learn anything more about X from C, besides the fact that X does not fall in $B_1, B_2, ..., B_q$.

This problem is called the Anonymous Multi-Attribute Encryption with Range Query and Conditional Decryption(AMERQCD). Here we want to introduce the algorithms of AMERQCD(D) and AMERQCD(1) schemes. The needed definitions are mentioned in details in [10]. AMERQCD scheme consists of the following polynomial-time randomized algorithms:

1. *Setup*($\sum$, U_{∇}): Takes a security parameter $\sum$ and *D-dimensional* lattice U_{∇} as an inputs. It generates public key PK and master private key MK as an outputs.

2. *Encrypt(PK,X,Msg)*: Takes a public key *PK*, a point *X*, and a message *Msg* from the message space *M*. It generates cipher-text *C* as an output.

3. *DeriveKey(PK,MK,B)*: Takes a public key *PK*, a master private key *MK*, and a hyper-rectangle *B* and outputs decryption key for hyper-rectangle *B*.

4. *QueryDecrypt(PK,DK,C)*: Takes a public key *PK*, a decryption key *DK*, and a cipher-text *C* and outputs either a plain-text *Msg* or $\perp$, signaling decryption failure.

For each message $Msg \in M$, hyper-rectangle B $\subseteq U_\Delta$, and point $X \in U_\Delta$, the above algorithm must fulfill the following constraints:

$$\textit{QueryDecrypt(PK,DK,C)} = \begin{cases} Msg & \text{if } X \in B \\ \perp & \text{w.h.p., if } X \notin B \end{cases}$$

where C = *Encrypt(PK,X,Msg)* and DK = *DeriveKey(PK,MK,B)*.

In the following, we introduce an efficient AMERQCD(1) scheme based on Anonymous Identity Based Encryption (AIBE). Figure 4.5 shows the intuition of the AMERQCD(1) construction. An interval tree is built over integers from *1* to *T*. Each leaf node in the interval tree represents a unique integer in $[1, T]$. Any internal node represents a range covered by all leaf nodes in the subtree rooted at that node. In order to encrypt a message *Msg* and a value *x*, a part of cipher-text is produced for each node on the path from the root of the tree towards the leaf node representing *x*. Therefore, the cipher-text is *O(log T)* in length.

To issue decryption keys for a range $[s, t] \subseteq [1, T]$, let $\Lambda(s, t)$ represent a set of nodes covering $[s, t]$, portion of decryption key for each node in $\Lambda(s, t)$ is issued. Because any range can be represented by a collection of *O(log T)* nodes in the tree, the decryption key has length *O(log T)*.

The construction of *AMERQCD(1)* from an *AIBE* scheme is straightforward as shown in figure 4.5, in order to encrypt a message *Msg* under a point *x*, *Msg* must be encrypted under all nodes along the path *P(x)* from *x* towards the root, in order to liberate the decryption keys for a range $[s, t]$, the decryption keys for all node *IDs* in $\Lambda(s, t)$ must be released.

In order to be able to check whether a decryption is valid, prior to encryption, we append a message $Msg \in \{0, 1\}^m$ with a series of trailing *0s*, $0^{m'}$

- *Setup($\sum$,T)*: calls $Setup^*(\Sigma)$ and outputs *PK* and *MK*.

- *Encrypt(PK, x,Msg)*: yields $(c_1, c_2, ..., c_L)$, where $c_l = Encrypt^*(PK, L_l(x), Msg||0^{m'})(1 \leq l \leq L)$.

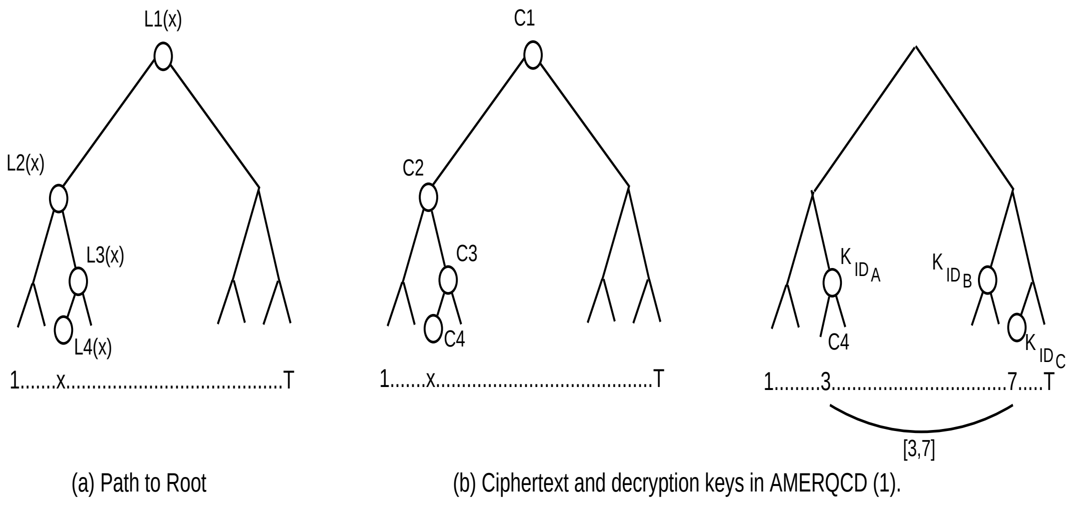

L1(x)
L2(x)
L3(x)
L4(x)
1.......x..T
(a) Path to Root
C1
C2
C3
C4
1.......x..T
K ID A
K ID B
C4
K ID C
1.........3..................................7.....T
[3,7]
(b) Ciphertext and decryption keys in AMERQCD (1).

Figure 4.5: An AMERQCD (1) scheme. (a) Path from the leaf node to the root. (b) Encryption to the point x = 3 and the keys released for the range [3,7]. [10]

- *DeriveKey(PK,MK, [s, t])*: yields a set $\{(l_{ID}, k_{ID})|ID \in \Lambda(s,t)\}$, where $k_{ID} = DeriveKey^*(PK, MK, ID)$ and l_{ID} is the depth of node ID in $\Gamma(T)$, DK has *O(log T)* size.
- *QueryDecrypt(PK,DK,C)*: where $C = (c_1, c_2, ..., c_L)$, defined as below:
  1. For each $(l, k_{ID}) \in DK$, compute: $V \leftarrow Decrypt^*(PK, k_{ID}, c_l)$. if V is of the form $\widehat{MSG}||0^{m'}$, then output $\widehat{MSG}$ as the decrypted message and exit.
  2. If for all $(l, k_{ID}) \in DK$, the previous step fails to produce a plain-text, then output $\perp$.

AMERQCD(D) and AMERQCD(1) are efficient schemes in terms of encryption cost, cipher-text size, decryption key size, and decryption cost. However, it is targeting different problem than our and it does not guarantee any authentication. Also, the problem of subscriptions confidentiality does not solved by these schemes.

4.5 Enabling Confidentiality in Content-Based Publish-Subscribe Infrastructures

This paper [11] makes a systematic analysis of providing subscribers and publishers confidentiality in publish-subscribe system. It provides a formal security model and shows that the maximum level of reachable security in this setting is restricted. It Tries to achieve confidentiality for commonly used applications and subscription languages in content-based publish-subscribe system.

This scheme aims to achieve confidentiality for both publishers and subscribers. The approach used by this scheme depends on the structure of notifications. This approach supports complex subscriptions composed from similar building blocks. By selecting the attributes that the broker is allowed to match, the amount of leaked information can be controlled.

This scheme supports confidentiality in content-based publish-subscribe system for different types of subscriptions. It adapts a scheme from Dawn Song [11] to support equality tests, it defines two novel schemes for range matches (numeric-valued attributes) and it uses a scheme from Eu-Jin Goh [11] for keyword matching (string-valued attributes).

1. Equality filtering: Song proposed a solution that enables searching on encrypted data. The idea of this scheme is to compute the secret value of an attribute by passing its plain-text value to a pseudo-random function, this pseudo-random function is keyed with the secret key. The encrypted subscription is the hidden value of the plain-text. Encrypted events are composed of two parts: a random nonce r, generated by the publisher, and the result of feeding r to a pseudo-random function, keyed with the hidden value of the notification's plain-text. This *Equal* scheme is shown in [11].

2. Keyword filtering: Substring matching is an important operation in content-based publish-subscribe systems. Goh mechanism depends on breaking the string into words and construct a bloom filter [11] to signal existence of a word in the string. The algorithm of Goh is shown in [11].

   An alternative scheme based on defining a dictionary that has one bit for every possible word in the string. This dictionary is shuffled using a pseudo-random permutation and blinded using pseudo-random functions and a random nonce. Events include the blinded dictionary, along with the random nonce. And the subscription contains the shuffled index of the word plus a "hidden" version of the index. The *Dictionary* scheme is shown in [11]. Compared to Goh algorithm, *Dictionary* does not generate false positive matches and does not impose any restrictions on the number of words in the document.

3. Numeric filtering: Numeric attributes filtering is supported by many implemen-

tations of content-based publish-subscribe systems. There are two forms of this numeric attributes: inequality test and range test.

In case of an inequality test, there are many constraints:
$\{" > p_1", " > p_2" ... " > p_l", " < p_1", " < p_2" ... " < p_l"\}$ defined in the system which called the dictionary. Subscriptions will be approximated with one of these constraints. Each event is considered to be a document containing the words in the dictionary that it matches. The Inequality scheme is defined in [11]. The overhead of this scheme is due to the size of the dictionary, equal to *2*l*.

To support range subscriptions, publishers and subscribers a-priori agree on a partitioning $P = \{p_1, p_2 ... p_l\}$ of D. Publishers encrypt the index of the subset N belongs to by using *Equality filtering* scheme defined above. The subscribers include as subscriptions encrypted versions of the indexes of the subsets in the partition they are interested in. In other words, several partitions of D, P_1,...,P_m are created, with different subset sizes and different starting offsets. Then a dictionary is created, which containing as words the index of the partition concatenated with the subset index, for all m partitions. A notification can be represented as a document with this dictionary by listing the subsets it is contained in. The subscription is approximated with one of the subsets in these partitions. The *range* scheme is defined in [11]. This scheme creates trade-off between the subscriptions-size and matching-time on one hand, and the number of false positives and the security achieved on the other hand.

This scheme compromised the decoupling needed between publishers and subscribers because it assumes the same keys for them and they have to priori agree on the partitioning. Also, many subscribers receive false positive events and there is no mechanism to achieve authentication between publishers and subscribers in this scheme.

Chapter 5

Identity Based Encryption(IBE)

In order to provide the security requirements needed in our content-based publish-subscribe we use the Identity Based Encryption(IBE) scheme for that.

5.1 Introduction

Public-Key Infrastructures (PKIs) [12], [13] and [14] were designed as mechanisms to handle certificates, but they became to be heavy in deployment and unwieldy to use. Shamir [15] tried to avoid the conventional problems associated with (PKIs) by introducing in 1984 the concept of identity-based cryptography where a public key can be any binary string identifying its owner non-ambiguously. Shamir motivated to simplify key-management and remove the need of public-key certificates the most that can be done. Since 1984 several practical solutions for identity based signatures (IBS) have been devised. But no feasible identity based encryption scheme (IBE) proposed until 2001 when Cocks [16] and Boneh Franklin [13] independently proposed schemes using quadratic residues and bilinear maps respectively.

In his paper in 1985, Adi Shamir [15] proposed the idea of Identity-Based Encryption (IBE) as a novel idea of a cryptographic scheme. This shceme enables any pair of users to securely communicate and to verify each other's signatures without the need to exchange private or public keys and without keeping key directories. In [12] Identity-Based Encryption(IBE) defined as: *"a public-key encryption scheme where any valid string, which uniquely identifies a user, is the public key of the user"*.

Un-like the Public-Key Infrastructure (PKI) where a public-key certificate is required to bind the key to its user. Instead, IBE requires a trusted central authority called a Private-Key Generator (PKG) for generation and distribution of private keys to registered identities, it removes several troubles associated with traditional PKI such as certificate lookup, certificate-revocation lists, life-cycle management, and cross-certification issues

Table 5.1: Differences Between IBE and Traditional Public-Key-Infrastructure(PKI) [12]

Features	ID based Public-Key Infrastructure (PKI)	Certificate Based PKI
Key Certification	NO	Yes
Private Key Generation	By Private Key Generator (PKG)	By user or Certification Authority(CA)
Key distribution	Requires a private protected channel and integrity for distributing a new private key from the TA to its owner	Requires an integrity protected channel for distributing a new public key from a user to his (Certification Authority)CA
Public key Retrieval	On the fly based on owner identifier's	From public directory or key owner
Escrow facility	Yes	No (Unless Public key generation is by Certification Authority(CA))

giving a greatly-simplified signature scheme and public-key encryption. Table 5.1 shows many differences between Identity-Based-Encryption and traditional PKI:

Figure 5.1 shows the basic secure communication steps of identity based encryption algorithm:

1. Alice encrypts the email using Bob's e-mail address, bob@c.com, as the public encryption key.

2. Bob contacts the key server, this key server asks a directory or other external authentication source to authenticate Bob's identity (ensures he is a valid user) and establish any other policy elements.

3. The key server returns Bob's private key, so Bob can decrypt the message, also he can use this private key to decrypt all future received messages.

Any IBE scheme like Cocks and Boneh Franklin schemes is made up of four algorithms:

- *"Setup: Generates system parameters params which are made public to all users in the system and a master-key sk_{PKG} which is known only to the Private Key Generator(PKG)."*

- *"Extract: Takes as input params, sk_{PKG} and any arbitrary ID and returns a private key sk_{ID} corresponding to that ID."*

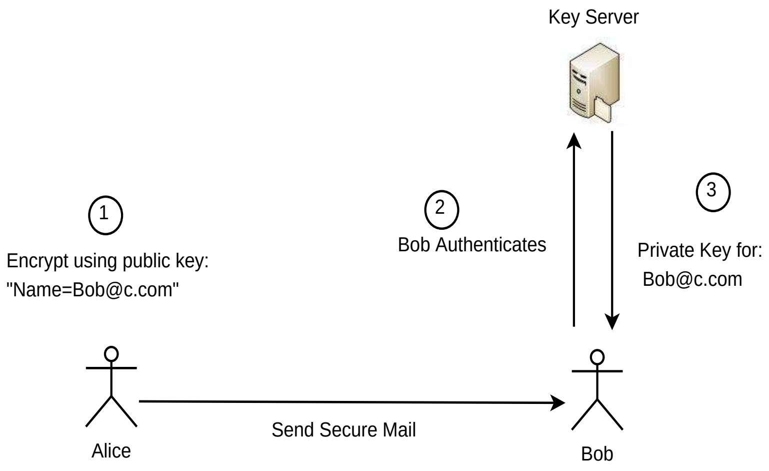


Figure 5.1: Identity Based Encryption [12]

- *"Encrypt: Takes as input params, the ID of the receiver and a plain-text message M and returns a cipher-text C."*
- *"Decrypt: Takes as input params, the private key sk_{ID} issued by the Private Key Generator(PKG) and the cipher-text C and returns the plain-text message M."*

5.2 Applicability To Content-Based Publish-subscribe system

There are many reasons we choose IBE scheme to provide the security requirements needed in our system. IBE has many advantages as:

1. No public key distribution infrastructure: Using the identity of the sender or receiver as sign key and encrypt key respectively eliminates the need for a public key distribution infrastructure, the authenticity of the public keys is guaranteed implicitly as long as the transport of the private keys to the corresponding user is kept secure.
2. Secure underlying cryptographic functions and secrecy at key generation center (KGC): the secret cryptographic functions at KGC prevents attacker of resolving and discovering the private keys.
3. Check of the identity before issuing private keys to the users, so prevent any illegal sender tries to fabricate messages.
4. Preserving the required decoupling between publishers and subscribers needed in our content-based publish-subscribe system.

In this thesis, we focus on Brent Waters scheme [14], which provides an efficient IBE scheme without random oracles. We use this scheme in the evaluation of our algorithm.

5.3 IBE Scheme Without Random Oracles

A Random Oracle is an oracle, for example, a theoretical black-box that responds to every query with a truly random response chosen uniformly from its output domain. The first efficient and secure method for identity-based encryption was put by Boneh and Franklin [13]. They proposed a solution using efficiently computable bilinear maps that was shown to be secure in the random oracle model. Since then, there have been schemes shown to be secure without random oracles like Brent's scheme [14]. In the next section, we will give a brief introduction to bilinear maps, after that the Diffie-Hellman assumptions which used to prove the security of Brent's scheme will shown. Then we will describe Brent's scheme in details.

5.3.1 Introduction to Bilinear Maps [12]

A bilinear map is a function which is linear in both its arguments, we assume maps that establish relationship between cryptographic groups, let G_1 and G_1 be cyclic groups of order q, where q is some large prime, a bilinear map is a function e : $G_1 * G_2$ such that for all $v,u \in$ G1 and $a,b \in Z$:

$e(au,bv) = e(bu,av) = e(u,v)^{ab}$

These maps have another name as pairings because they connect a pair of elements from group G_1 to an element or a pair of elements in G_2. These maps can be degenerate, for example, maps all pairs $G_1 * G_2$ to the identity in G_2. In cryptography the interested maps are those which called admissible bilinear maps.

As mentioned in [12] the admissible bilinear map satisfies the following properties:

1. *"Bilinear: we say that a map* $e^{'}$: $G_1 {*} G_2 \rightarrow$G2 *is bilinear if* $e^{'}$(au,bv) = $e^{'}$(bu,av) = $e^{'}(u,v)^{ab}$ *for all u,v* $\in G_1$ *and all a,b* $\in Z$*".*

2. *"Non-degenerate: the map does not send all pairs in* $G_1 {*} G_1$ *to the identity in G2, i.e.* $e^{'}$(u,v) $\neq 1$, *for all u,v* $\in G_1$".

3. *"Computable: there is an efficient algorithm to compute* $e^{'}$(u,v) *for any u,v* $\in G_1$."

$e^{'}$ denoted the admissible bilinear map. G_1 is implemented using the set of all points on specific elliptic curve which form an additive group and G_2 is is a multiplicative group of large prime order.

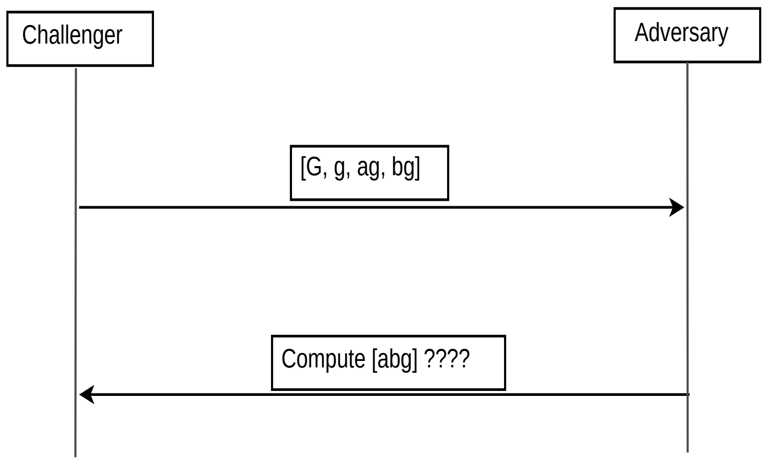


Figure 5.2: Computational Diffie-Hellman Assumption [12]

5.3.2 Computational Diffie-Hellman Assumption(CDH) [12]

Assume a cyclic group G of order q represented by the set of all points on an elliptic curve. The CDH assumption states that given $\langle g, ag, bg\rangle$ for a randomly chosen generator g and random a,b $\in \{0, 1....q-1\}$, it is computationally intractable to compute the value $\langle abg\rangle$, figure 5.2 illustrates that. The challenger stimulates the adversary by giving $\langle G, g, ag, bg\rangle$ parameters, depends on Computational Diffie-Hellman Assumption (CDH) assumption, it is computationally very hard for the adversary to compute $\langle abg\rangle$.

5.3.3 Bilinear Diffie-Hellman Assumption(BDH) [12]

Assume G_1 be an additive group of prime order p represented by the set of all points on an elliptic curve and G_2 be a multiplicative group of prime order q, if $e^{'}$: $G_1 {*} G_1 \rightarrow G2$ be an admissible bilinear map and let P be a generator of G_1. Then the Bilinear Diffie-Hellman (BDH) Assumption in $\left\langle G_1, G_2, e^{'}\right\rangle$ can be expressed as follows: Given $\langle P, aP, bP, cP\rangle$ for some a,b,c $\in Z_q^*$, it is computationally infeasible to compute W $= e^{'}(P, P)^{abc} \in G_2$ if the Computational Diffie-Hellman (CDH) problem is intractable, figure 5.3 shows that, in this figure the challenger stimulates the adversary by giving $\langle G_1, G_2, P, aP, bP, cP\rangle$ parameters, depends on Bilinear Diffie-Hellman (BDH) assumption, it is is computationally very hard for the adversary to calculate W $= e^{'}(P, P)^{abc} \in G_2$.

5.3.4 Brent's IBE Construction [14]

The construction of Brent's scheme is a modification of the Boneh and Boyen [17] scheme. **For both crypto and signature scheme shown below**, assume G be a group of prime

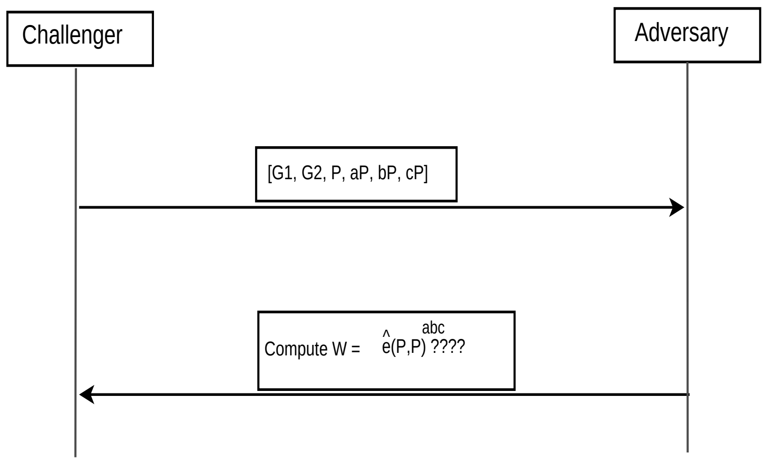


Figure 5.3: Bilinear Diffie-Hellman Assumption [12]

order, p, for which there exists a computable bilinear map into G_1. Let $e : G \times G \rightarrow G_1$ denote the bilinear map and g be the corresponding generator. The size of the group is determined by the security parameter. Identities will be represented as bit-strings of length n, a separate parameter unrelated to p. Identities also can be bit-strings of arbitrary length and n be the output length of a collision-resistant hash function, $H : \{0,1\}^* \rightarrow \{0,1\}^n$.

The Brent's crypto scheme is:

- **Setup**: The system parameters are generated as follows. A secret $\alpha \in Z_q$ is chosen at random. A random generator, $g \in G$, is chosen and set the value $g_1 = g^\alpha$ and choose g_2 randomly in G. Moreover, the authority chooses a random value $u^{'} \in G$ and a random n-length vector $U = (u_i)$, whose elements are chosen at random from G. The published public parameters are $g, g_1, g_2, u^{'}$, and U. The master secret is g_2^α.

- **Key Generation**: Let v be an n bit string representing an identity, v_i denote the *ith* bit of v, and $\nu \subseteq \{1, ..., n\}$ be the set of all i for which $v_i = 1$. (That is ν is the set of indicies for which the bit-string v is set to 1.) A private key for identity v is generated as follows. First, a random $r \in Z_q$ is chosen. Then the private key is constructed as:

$$d_v = \left(g_2^\alpha \left(u^{'} \prod_{i \in \nu} u_i \right)^r, g^r \right).$$

- **Encryption**: A message $M \in G_1$ is encrypted for an identity v as follows. A value $t \in Z_p$ is chosen at random. The cipher-text is then constructed as:

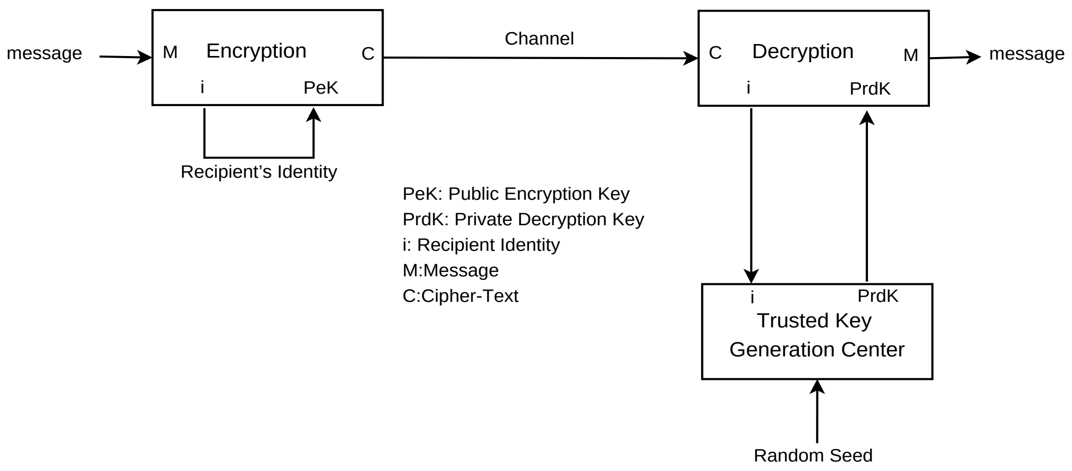


Figure 5.4: Brent's IBE Crypto-System

$$C = \left(e(g_1, g_2)^t M, g^t, \left(u^{'} \prod_{i \in \nu} u_i \right)^t \right).$$

- **Decryption**: Assume $C = (C_1, C_2, C_3)$ be a valid encryption of M under the identity v. Then C can be decrypted by $d_v = (d_1, d_2)$ as:

$$C_1 \left(e\left(d_2, C_3\right) / e\left(d_1, C_2\right) \right) = M$$

Brent's IBE crypto-system scheme can be used in any many-to-many system model. Sender encrypts the secret message and only the legal receiver can decrypt it. Figure 5.4 shows that, the sender encrypts the message using the receiver identity as the *public encryption key (PeK)*. The legal receiver uses his identity to get the *private decryption key (PrdK)* to decrypt the message. The trusted key generation party checks the identity of the receiver before grant him the *private decryption key (PrdK)*, the link to this trusted party is secure and protected.

The Brent's signature scheme is:

- **Setup**: The public key is generated as follows. A secret $\alpha \in Z_q$ is chosen at random. A random generator, g, is chosen and set the value $g_1 = g^\alpha$ and choose g_2 randomly in G. Moreover, the algorithm chooses a random value $u^{'} \in G$ and a random n-length vector $U = (u_i)$, whose elements are chosen at random from G. The published public key is $g, g_1, g_2, u^{'}$, and U. The master secret is g_2^α
- **Signing**: Let M be an n-bit message to be signed and M_i denote the *ith* bit of M, and $M \subseteq \{1, ..., n\}$ be the set of all i for which $M_i = 1$. A signature of M is generated as follows. First, a random $r \in Z_q$ is chosen. Then the signature is constructed as:

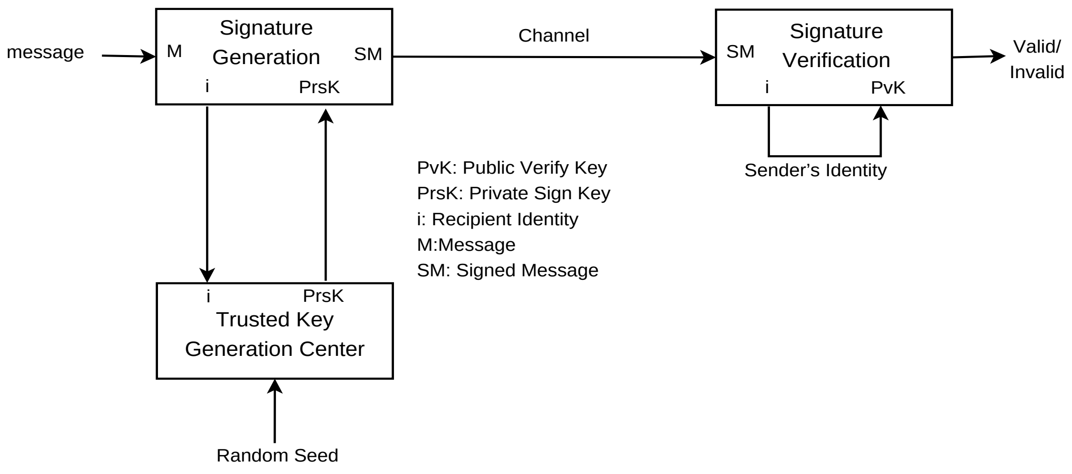


Figure 5.5: Brent's IBE Signature-Scheme

$$\sigma_M = \left(g_2^{\alpha}\left(u^{'}\prod_{i\in M} u_i\right)^r, g^r\right).$$

- **Verification**: Suppose we wish to check if $\sigma = (\sigma_1, \sigma_2)$ is a signature for a message M. The signature is accepted if $e(\sigma_1, g)/e(\sigma_2, u^{'}\prod_{i\in M} u_i) = e(g_1, g_2)$.

Using of Brent's IBE signature scheme is illustrated in figure 5.5. The sender sign the message with his *private sign key (PrsK)* which he got from the trusted key generation party. The receiver authenticates the sender using his identity as *public verify key (PvK)*. The key generation party checks the identity of any sender before send him the *private sign key (PrsK)* to prevent any malicious sender fabricates messages in the system.

So Brent's IBE scheme achieves confidentiality and authentication of messages in any many-to-many system models, in the next section we will describe how to use this scheme in our content-based publish-subscribe system.

Chapter 6

Authentication And Confidentiality In Content-Based Publish-Subscribe System

6.1 Basic Architecture And Construction Of Keys

We add the trusted third party(TTP) which is responsible for perform the basic cryptographic functions needed to secure our system. Then, our system will be composed of three components: publishers, subscribers and TTP. Figure 6.1 illustrates that. We aim to build the subscriber's tree and to disseminate the events to the right subscribers while keeping the subscriptions of subscribers hidden. Moreover, subscribers must be able to discover any fake event and stop it. We assume that the connection to TTP is secure and protected.

Trusted Third Party(TTP) contains the key generation center(KGC) which is basically responsible for creating different keys needed in our system. TTP maintains two keys: public-key(PK) and master key(MK), it can be realized as:

1. Either grant each node a smart card when first joins the network, this smart card contains the secret private keys.
2. Or central-server or group-of-servers to generate the secure keys for each node, the link to the server(s) must be protected and secure.

We show before that any region in the event space in our content-based publish-subscribe system can be represented by a bit-string *(DZ)* of *1's or 0's.* This bit-string is generated by dividing each axis in the event space alternatively. Regardless the number of axis or the divisions, all regions can be represented by these bit-strings.

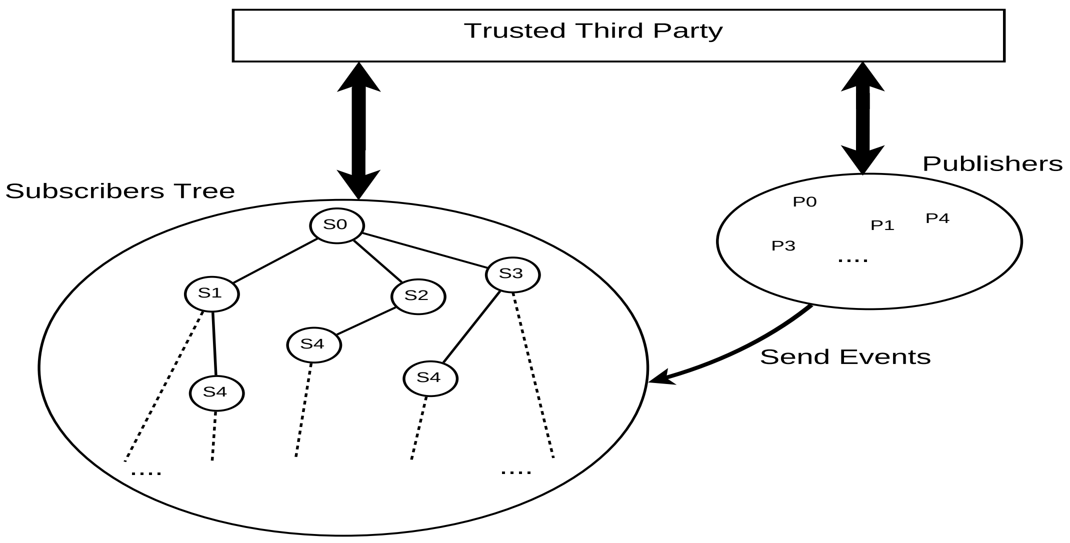


Figure 6.1: Content-Based Publish-Subscribe System with Trusted Third Party(TTP)

Any **region in this event space is used as the identity for the subscriber who subscribes for it**. To recognize that this identity is for the subscriber, we concatenate it with a defined string equals **"SUB"**. Figure 6.2 shows 2-dimensional event space and the identities of different subscribers in it.

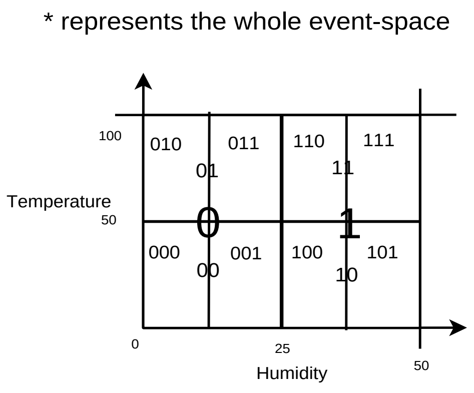
Any event in this event space is represented by a **point**, so it is located in many regions started from the biggest region towards the smallest one. **Publisher identities are represented by all regions where his event located**. To recognize their identities, we concatenate them with a defined string equals **"PUB"**. Figure 6.3 shows 2-dimensional event space and the identities of different publishers in it.

Keys in our system can be classified as:

1. ***Public Encryption Key*** ($Public_{EncKey}$): The identity of the received subscriber (any subscription region concatenated with "SUB"). This is a public key, any node (publisher or subscriber) can simply generate it without connection to the TTP.

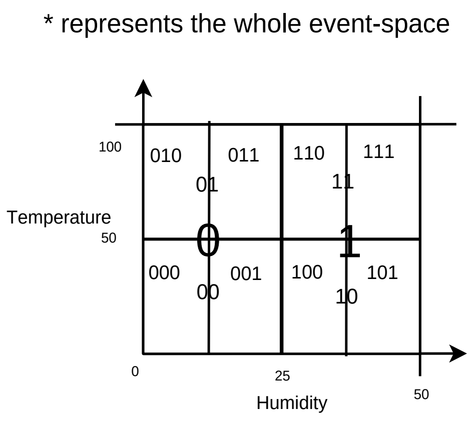
2. ***Private Decryption Key*** ($Private_{DecKey}$): Generated by the TTP and owned only by legal subscribers. Any subscriber asks TTP for this key by sending his identity, (as shown in figure 6.2). TTP assures that he is a legal subscriber for the event space region (included in the identity) before grant him this key.

3. ***Private Sign Key*** ($Private_{SignKey}$): Generated by TTP and owned only by legal publishers. A publisher asks TTP for his keys by sending his identities (as shown in figure 6.3). TTP assures that this publisher is allowed to publish events in the corresponding event space regions (included in the identities) before grant him the sign keys.



Subsciber	Identity of the subscriber
S*	"*SUB"
S1	"1SUB"
S10	"10SUB"
S11	"11SUB"
S0	"0SUB"
S01	"01SUB"
S001	"001SUB"

Figure 6.2: Subscribers Identity in 2-dimensional Event Space



Publisher	Identities of the publisher
P010	"*PUB" "0PUB" "01PUB" "010PUB"
P000	"*PUB" "0PUB" "00PUB" "000PUB"
P110	"*PUB" "1PUB" "11PUB" "110PUB"

Figure 6.3: Publishers Identities in 2-dimensional Event Space

4. ***Public Verify Key*** ($Public_{VerifyKey}$): The identity of the sender publisher (any subscription region concatenated with "PUB"), any node (publisher or subscriber) can simply generate it without connection to the TTP.

6.2 Authentication

6.2.1 Authentication Of Publishers And Subscribers

Publishers and subscribers ask TTP for their secret private keys ($Private_{SignKey}$ and $Private_{DecKey}$ respectively) as mentioned in the last section. Because TTP checks the identity of publishers and subscribers before grant them their secret keys, the authentication of publishers and subscribers is guaranteed in the system. Publishers guarantee that their events will just be received by the legal subscribers. Also, subscribers assure that their received events were published from legal publishers. This authentication between publishers and subscribers is achieved while they are decoupled (do not know each other's).

Figure 6.4 represents the responses of TTP to different subscribers and publishers in the system.

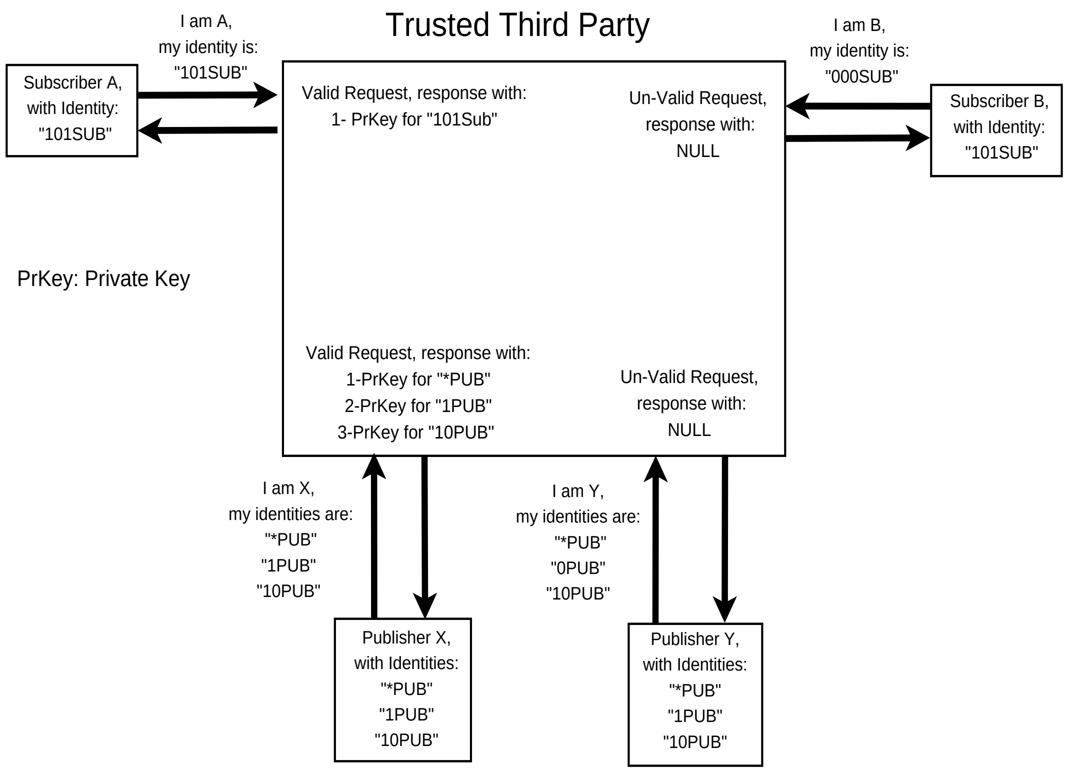

Trusted Third Party
I am A, my identity is: "101SUB"
Subscriber A, with Identity: "101SUB"
Valid Request, response with: 1- PrKey for "101Sub"
Un-Valid Request, response with: NULL
I am B, my identity is: "000SUB"
Subscriber B, with Identity: "101SUB"
PrKey: Private Key
Valid Request, response with: 1-PrKey for "*PUB" 2-PrKey for "1PUB" 3-PrKey for "10PUB"
Un-Valid Request, response with: NULL
I am X, my identities are: "*PUB" "1PUB" "10PUB"
I am Y, my identities are: "*PUB" "0PUB" "10PUB"
Publisher X, with Identities: "*PUB" "1PUB" "10PUB"
Publisher Y, with Identities: "*PUB" "1PUB" "10PUB"

Figure 6.4: TTP Responses To Subscribers And Publishers

6.3 Confidentiality

6.3.1 Confidentiality Of Events

When the publisher wants to publish any event, he encrypts and signs it using $Public_{EncKey}$ and $Private_{SignKey}$ respectively. Encryption and signing are done using the keys correspond to the region in the event space. **Any legal events must end with 4 bytes of zeros, so the publisher appends his events with these bytes**.

Figure 6.5 shows an example of how the publisher *(P010)* builds his encrypted event. In this figure, publisher *(P010)* encrypts and signs the event four times using the encryption and singing keys correspond to ("*", "0", "01" and "010") event space regions. He puts the encrypted events and their signatures in consecutive locations in the resulted packet, starting from the largest region ("*") towards the smallest one ("010") in the event space (figure 6.5(B)).

The publisher encrypts and signs the event **logarithmic** times (the number of his identities). Algorithm 6.1 shows the Encrypt_and_Sign_Event_Algorithm executed by publishers before disseminate events to the subscriber's tree.

The Encrypted-And-Signed-Events packet shown in 6.5(B) is sent to the subscriber's tree (shown in figure 6.1) and disseminated to the right subscribers *(the mechanism of routing this packet is discussed later).* When the subscriber receives an encrypted event, he tries to decrypt it using his $Private_{DecKey}$, also he checks the signature of this event using $Public_{VerifyKey}$ to assure it sent from a legal publisher.

Because the publisher inserts the encrypted events and their signatures in a consecutive locations in Encrypted-And-Signed-Events packet, **subscriber expects the location of his presumed encrypted event and the signature for this event in the this packet.** For example, the presumed encrypted event for the subscriber with identity *"*SUB"* is the first item of Encrypted-And-Signed-Events packet, and the presumed signature is the second item of this packet, on the other hand, subscribers with identities *"0SUB"* and *"1SUB"* expect their encrypted events at the third location and their signatures at the fourth location of this packet and so on.

Any subscriber receives the Encrypted-And-Signed-Events packet tries to **decrypt the entry of this packet located at his presumed event location** and **verify its signature in the consecutive location**. A subscriber **decides whether the decrypted event is valid or not by checking the last four bytes of it (must be 4 bytes of zeros for the valid event)**. Figure 6.6 shows an example of the *S101* subscriber, the presumed encrypted event index for this subscriber is *6*(start indexing from 0).

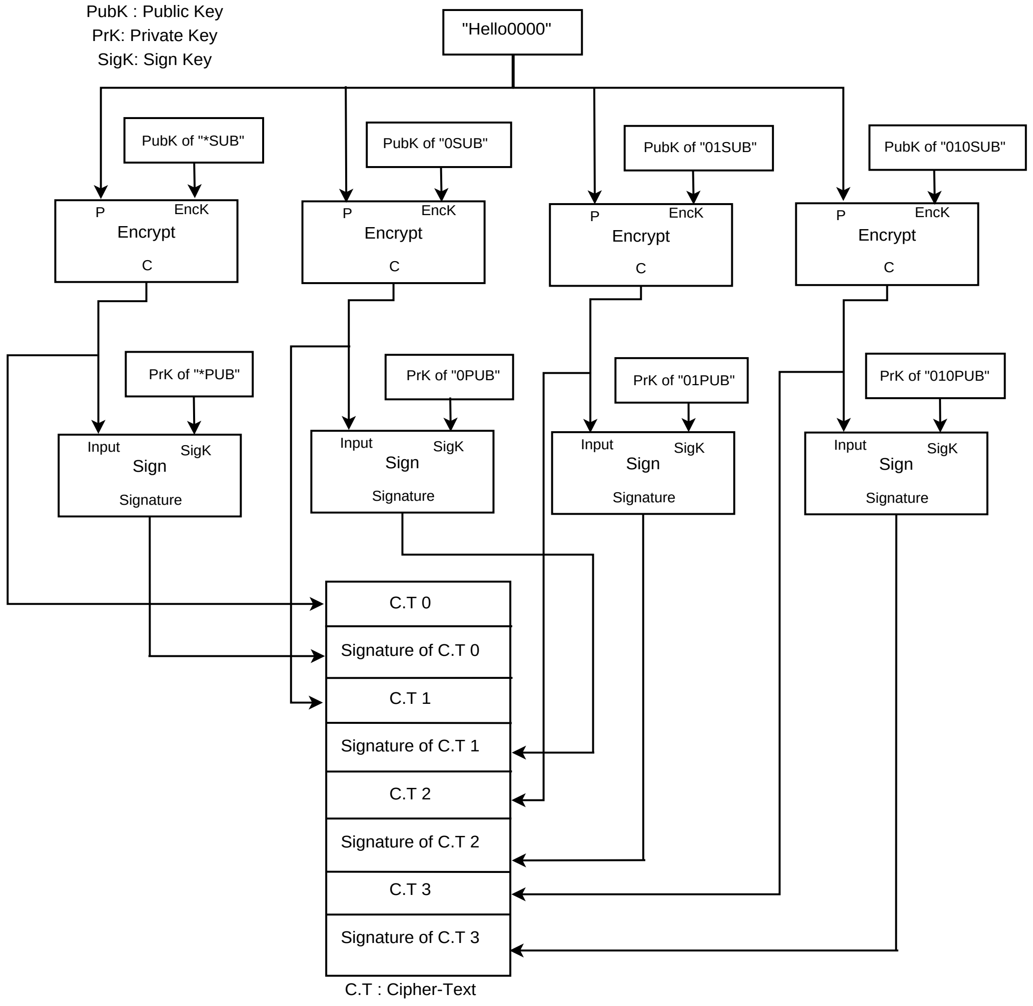
Algorithm 6.2 shows the Decrypt_and_Verify_Event_Algorithm executed by sub-

Publisher	P010
Identities of P010	"*PUB" "0PUB" "01PUB" "010PUB"
Private Keys of P010	PrK of "*PUB" PrK of "0PUB" PrK of "01PUB" PrK of "010PUB"

To send an Event = "Hello"
1- Append 4 bytes of 0's at the end ==>
Event = "Hello0000"
2- look at (b)

(A) Identities and Private Keys of P010 Publisher

EncK : Encryption Key
P: Plain-Text
C: Cipher-Text
PubK : Public Key
PrK: Private Key
SigK: Sign Key



(B) Build The Encrypted-And-Signed-Events Packet

Figure 6.5: Publish Event of Publisher P010

Algorithm 6.1 (Encrypt_and_Sign_Event_Algorithm), Executed By Publisher.
Input: Plain-Text.
Output: Encrypted-And-Signed-Events Packet.

```
$Event_{PlainText}$ = Plain-Text.
$Event_{PlainText}.Append(\text{"0000"}_{Bytes})$
$Result = NULL$
$EncryptedResult = NULL$
$SignatureOF_{EncryptedResult} = NULL$
for all $identity \in$ Publisher Identities do
    $EncryptedResult$ = Encrypt $Event_{PlainText}$ using $Public_{EncKey}$ of $identity$.
    $SignatureOF_{EncryptedResult}$ = Sign $EncryptedResult$ using $Private_{SignKey}$ of $identity$.
    $Result.Add(EncryptedResult)$
    $Result.Add(SignatureOF_{EncryptedResult})$
end for
return $Result$.
```

scribers when receive Encrypted-And-Signed-Events packet.

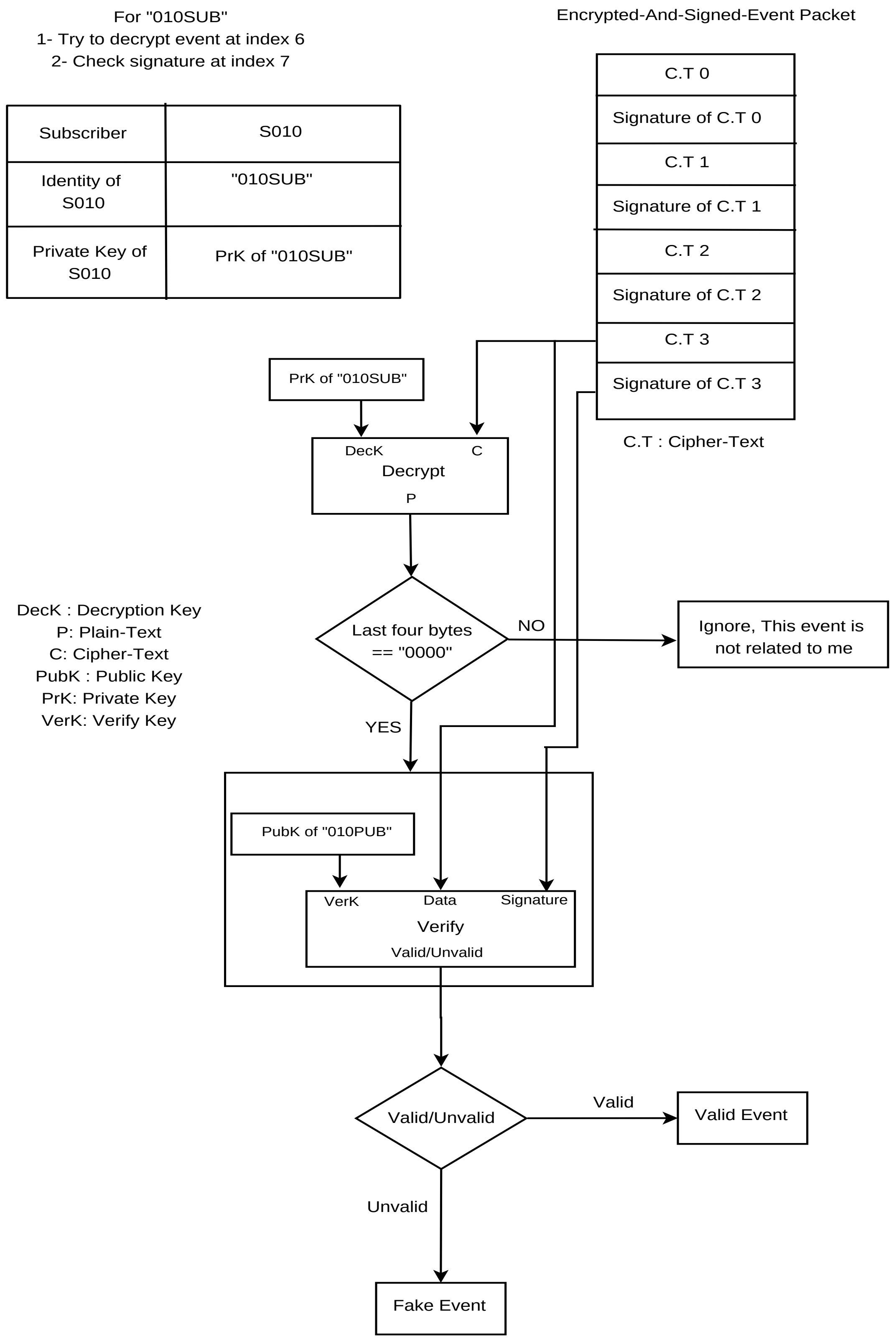


Figure 6.6: Subscriber S010 Actions When Receving Encrypted-And-Signed-Events Packet

Algorithm 6.2 (Decrypt_and_Verify_Event_Algorithm), Executed By Subscriber.
Input: Encrypted-And-Signed-Events Packet (Indexing is started from 0)
Output: Matched-event or Un-Matched-Event.
Comments are between {}.

```
Received_Packet = Input.
$Event_{PlainText} = NULL$
$Index_{OfEncyptedEvent} = NULL$
$Index_{OfSignature} = NULL$

if $identity \equiv$ "*SUB" then
    $Index_{OfEncyptedEvent} = 0$

    $Index_{OfSignature} = 1$
else
    $Index_{OfEncyptedEvent} = 2 * (identity.length - 3)$ {without the length of "SUB"}

    $Index_{OfSignature} = Index_{OfEncyptedEvent} + 1$
end if
$Event_{PlainText}$ = Decrypt $Received\_Packet.GetAtIndex(Index_{OfEncyptedEvent})$ using $Private_{DecKey}$.

if $Event_{PlainText}$ Ended with 4 bytes of "0000" then
    Verify The Signature at $Received\_Packet.GetAtIndex(Index_{OfSignature})$ using $Public_{VerifyKey}$.

    if Signature Is Valid then
        Read The Event {Valid Event}

        return Matched-Event
    else
        Ignore {Fake Event}

        return Un-Matched-Event
    end if
else
    Ignore {The Event is not related to this subscriber}

    return Un-Matched-Event
end if
```

6.3.2 Subscriptions Confidentiality

Figure 6.1 shows the subscriber's tree in our system. It is assumed that this tree is built in a way that the subscribers with large subscription are in the higher level in this tree. **Each subscriber in this tree can connect to *one* parent and has a maximum number of children equals $\boldsymbol{n}$.** We aim to cluster subscribers based on their subscriptions. Here, we introduce the technique used to build the desired subscriber's tree while keeping the subscriber's subscription confidential.

The mechanism used to enable any new subscriber joins the system is handled with the corporation of TTP, we mentioned before that any subscriber joins the system asks TTP for his $Private_{DecKey}$, he further asks the TTP for another secure packet which we called **Connection Request(CR)**. This CR enables him to connect to the right position in the subscriber's tree.

Based on the idea that the subscriber with larger subscription area is higher in the tree, then for each subscriber, there are some subscribers that he can connect to as a child. We introduced in the *Background* section the meaning of *Containment subscription.* **In order to build the desired subscriber's tree topology it is needed that any subscriber must contain the subscriptions of his children**. For example, subscriber with identity "010SUB" can connect to other subscribers with identities ("*SUB", "0SUB", "01SUB", "010SUB"). **So, for each subscriber there is a logarithmic number of subscribers he can connect to.**

TTP performs the following when any subscriber asks for his CR:

1. **Check if this subscriber is valid, does he want to subscribe for the right subscription region?**
2. **Find all identities of other subscribers he can connect to. (Identities of subscribers contain his subscription) (logarithmic number of subscribers).**
3. **Encrypt secret strings ended with 4 bytes of zeros using $Public_{EncKey}$ of these identities. (logarithmic number of encryptions).**
4. **Send this request to this legal subscriber.**

Figure 6.7 shows an example of the subscriber *S010* when he asks the TTP for his CR in order to join the system. *S010* sends his CR to random connected subscriber in the subscriber's tree. The receiver subscriber of the CR tries to decrypt any of the cipher-texts *(C.T 0, C.T 1, C.T 2,C.T 3) in figure 6.7(B)*, **(the decryption is valid if the last four bytes equal "0000")**. Moreover, the connected subscriber must authenticate that this CR is built by *TTP* by using the public parameters of TTP he has. In case the receiver subscriber successfully decrypts and verifies the CR he accepts *S010* as a child, actually just the subscribers with identities *("*SUB", "0SUB", "01SUB", "010SUB")* can do that.

In case new subscribers with identities *"*SUB"* or *"1SUB"* want to join the system. Their CR sent by the TTP contains just one or two entries respectively, look at figure 6.8(A). Assume that there is an already connected subscriber (denoted by A) in the system with identity *"*SUB"*. In case A receive the CRs shown in figure 6.8(A), he can simply discover the subscription of the sender. (If the sender identity is *"*SUB"* $\rightarrow$ the subscription of the sender equals *"*"*. If the sender identity is *"1SUB"* $\rightarrow$ the subscription of the sender either *"*"*, *"1"* or *"0"*).

In order to prevent the received subscriber of expecting the identity (subscription) of the sender subscriber, a random data is added to the CR. So the received subscriber cannot detect the subscription of the sender based on the number of $C.T$ in CR. This is illustrated in figure 6.8(B). In this figure a random data is added to the CR shown in figure 6.8(A).

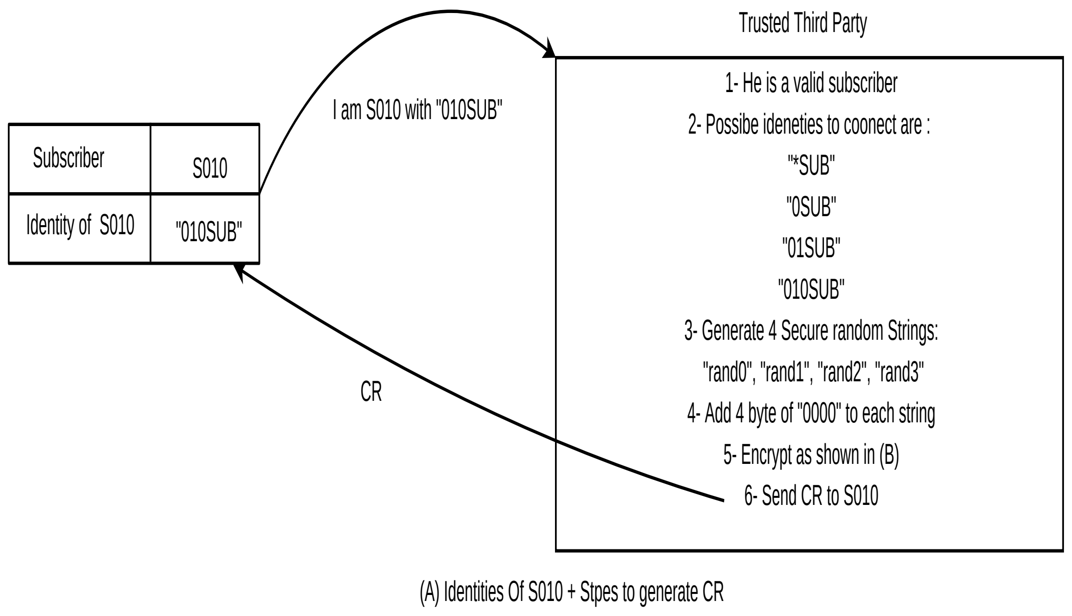
There is an ability that the received subscriber detects the identity(subscription) of the sender subscriber based on the index of the decrypted $C.T$ in figure 6.8(B). For example, the first $C.T$ is for *"*SUB"*, the second C.T belongs either to *"1SUB"* or *"0SUB"* and so on, to avoid that TTP shuffles the CR request, so there is no indication between the index of $C.T$ and the identity of the sender subscriber, this is shown in figure 6.8(C).

As a result the building of CR by TTP modified to the following steps:

1. **Check if this subscriber is valid, does he want to subscribe for the right subscription region?**
2. **Find all identities of other subscribers he can connect to. (Identities of subscribers contain his subscription) (logarithmic number of subscribers)**.
3. **Encrypt secret strings ended with 4 bytes of zeros using $Public_{EncKey}$ of these identities. (logarithmic number of encryptions)**.
4. **Add some random data to the CR**.
5. **Shuffle the whole CR packet randomly**.
6. **Send this request to this legal subscriber.**

The subscriber who receives CR tries to decrypt and verify it using algorithm 6.3. Based on the result of this algorithm the connected subscriber determine whether he can accept the new subscriber as a child or not.

We said that the number of children per each subscriber is limited. So there is a possibility that the connected subscriber accepts the connection of the new subscriber but


Trusted Third Party
I am S010 with "010SUB"
Subscriber
S010
Identity of S010
"010SUB"
1- He is a valid subscriber
2- Possibe ideneties to coonect are :
"*SUB"
"0SUB"
"01SUB"
"010SUB"
3- Generate 4 Secure random Strings:
"rand0", "rand1", "rand2", "rand3"
4- Add 4 byte of "0000" to each string
5- Encrypt as shown in (B)
6- Send CR to S010
CR
(A) Identities Of S010 + Stpes to generate CR

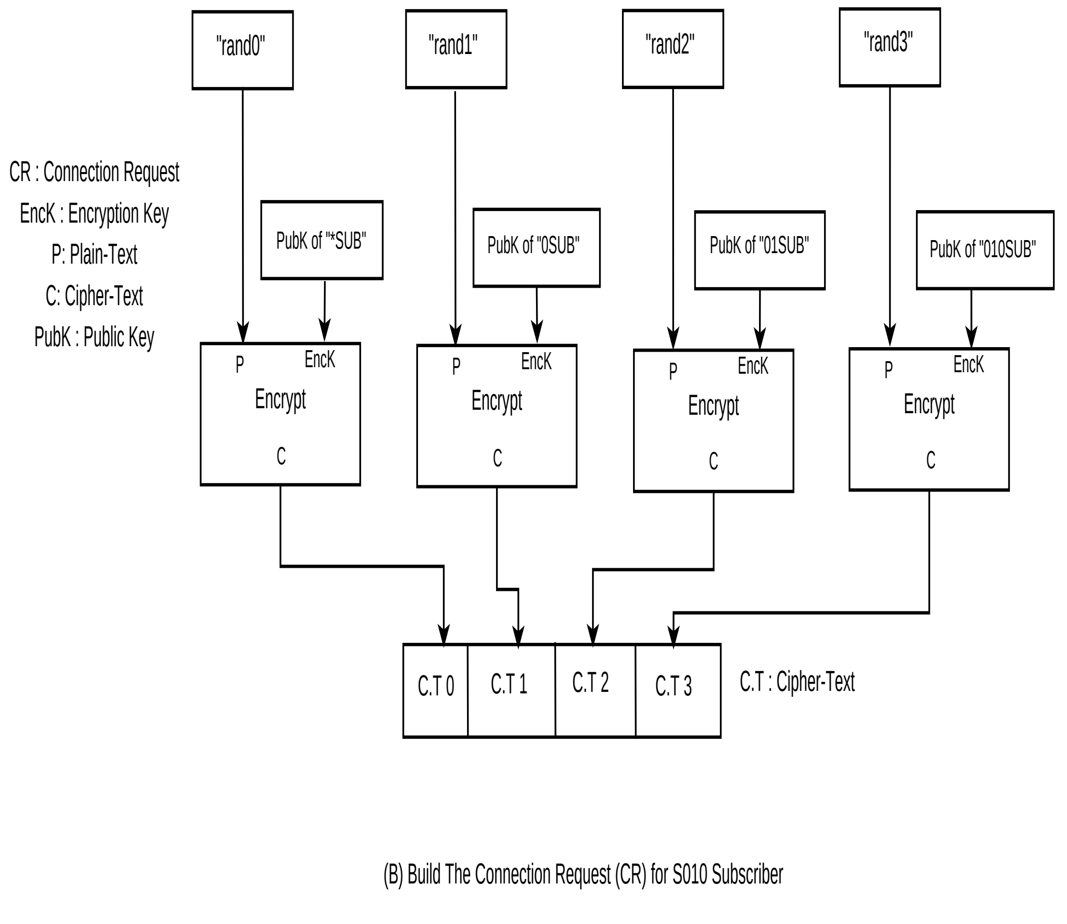


Figure 6.7: Connection Request For Subscriber S010

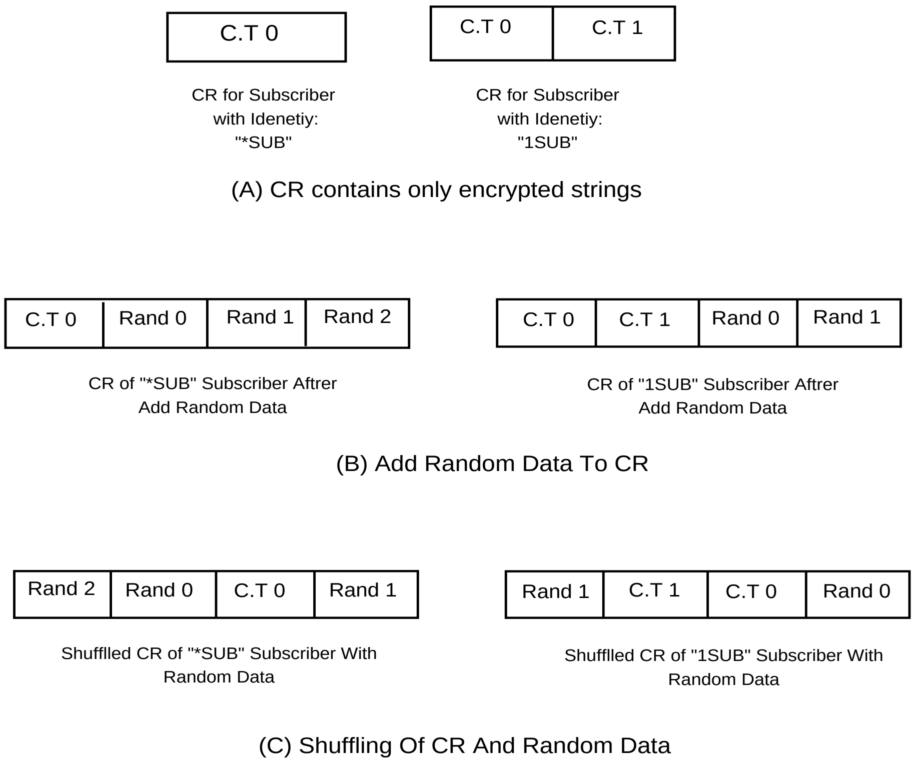

C.T 0
C.T 0
C.T 1
CR for Subscriber with Idenetiy: "*SUB"
CR for Subscriber with Idenetiy: "1SUB"
(A) CR contains only encrypted strings
C.T 0
Rand 0
Rand 1
Rand 2
C.T 0
C.T 1
Rand 0
Rand 1
CR of "*SUB" Subscriber Aftrer Add Random Data
CR of "1SUB" Subscriber Aftrer Add Random Data
(B) Add Random Data To CR
Rand 2
Rand 0
C.T 0
Rand 1
Rand 1
C.T 1
C.T 0
Rand 0
Shufflled CR of "*SUB" Subscriber With Random Data
Shufflled CR of "1SUB" Subscriber With Random Data
(C) Shuffling Of CR And Random Data

Figure 6.8: Modifications Of CR For Subscribers With Identities "*SUB" and "1SUB"

Algorithm 6.3 (Decrypt_and_Verify_CR Algorithm), Executed By the Connected Subscriber.
Input: CR (Indexing is started from 0)
Output: Accept or Reject the connection of the sender.
Comments are between {}.

```
$Event_{PlainText} = NULL$

$Index_{C.T} = 0$ {Initialize to 0}

$Size_{OfCR} = CR.Size()$

for $Index_{C.T} < Size_{OfCR}$ do

  $Event_{PlainText}$ = Decrypt $CR.GetAtIndex(Index_{C.T})$ using $Private_{DecKey}$.

  if $Event_{PlainText}$ Ended with 4 bytes of "0000" then

    Verify That the $CR$ is built by TTP using $TTP\_Public\_Parameters$

    if $CR$ Is Verified then

      return Accept Connection {Valid and authorized CR}

    end if
  end if
end for

return Reject Connection {Can't accept it as a child}
```

he does not have free space for this connection. Moreover, the connected subscriber may reject the received connection request. In order to make our system scalable enough to accept the connection of any new subscriber regardless of his subscription, any connected subscriber receives CR packet will execute algorithm 6.3. Based on the returned result of this algorithm (accept or reject connection) the connected subscriber decides to:

1. Accept the connection of the subscriber as a child.
2. Make a swap with him (the receiver subscriber disconnects from his parent and be a child of the new subscriber, and the new subscriber be a child of the old parent of the receiver subscriber).
3. Forward the received *CR* to his parent or children.
4. Disconnect one of the children randomly, and accept the connection of the new subscriber as a child.

In our system, we assume that we have a subscriber with identity *"*SUB"* initially in the system. Any new subscriber can connect to the system by send his CR to one subscriber selected randomly from the connected tree (just $S*$ at the beginning). As a result, the subscriber's tree incrementally grows, figure 6.9 illustrates that. In order to determine which decision to do, the connected subscriber executes algorithm 6.4. In the following we discuss different cases resulted when any connected subscriber receives CR:

- Received CR decrypted successfully: if any connected subscriber decrypts CR successfully and he has enough space, he accepts this new subscriber to connect to him as a child. If he does not have enough space he will send this CR to his children. If no one of the children accept or swap the connection with the new subscriber → He will disconnect one of them randomly and accepts the new subscriber as a child. Figure 6.10 shows these cases in more details.
- Received CR cannot be decrypted successfully: in case the connected subscriber cannot decrypt the received CR, he checks whether it sent from his parent or not, if it is not → He simply forwards this CR to his parent, this is represented in figure 6.10 using the abbreviation **C.D.S.P**.

  In case the parent is the sender that means he can decrypt the CR but he has no free space to connect to this new subscriber → In this case the connected subscriber sends his CR to the new subscriber, then there are two possible cases:

  1. Either the new subscriber can't decrypt the CR → In this case the child sends a $Failed_{Connection}$ message to the parent, and the parent will disconnect one of his children randomly as mentioned and shown in the previous note and figure 6.10 respectively (he disconnect one of them if all of them reply with $Failed_{Connection}$ message).

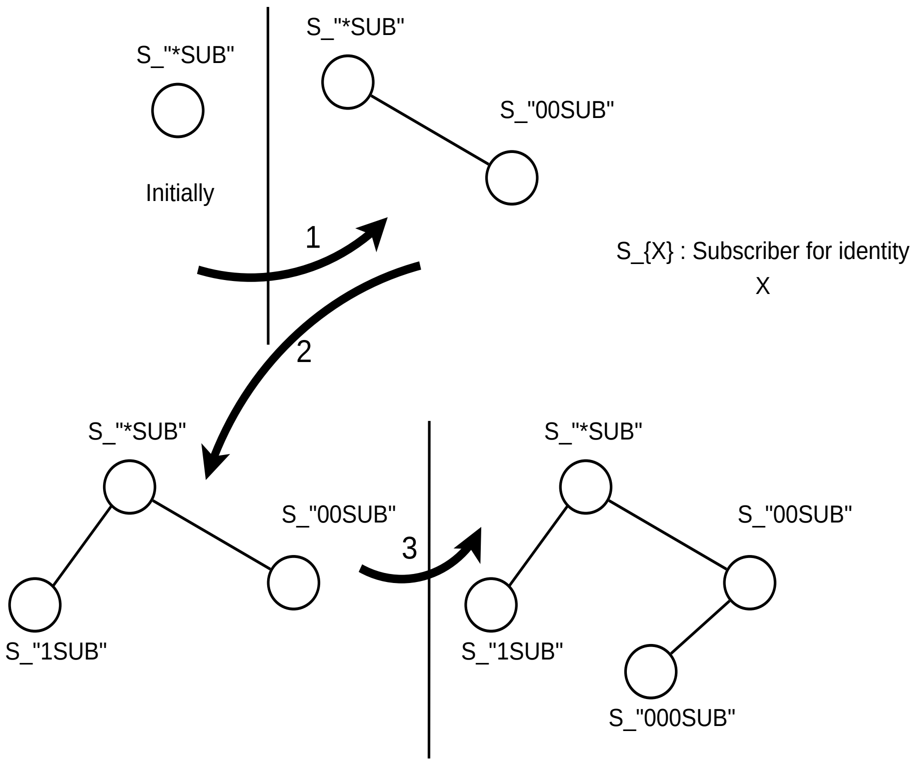


Figure 6.9: Growing Of Subscribers Tree

Algorithm 6.4 (Handling_CR_Algorithm), Executed By the Connected Subscriber.
Input:CR
Output: None.
Comments are between {}.

```
$Result_{Returned} = NULL$

$Result_{Returned}$ = Decrypt_and_Verify_CR Algorithm($CR$)

if $Result_{Returned}$ == Accept Connection then
  if There is enough space then
    CONNECT.
  else
    Send $CR$ to all child's.
    if All Child's return $Failed_{Connection}$ then
      Disconnect one of the child's randomly and CONNECT.
    end if
  end if
else
  if Received From Parent then
    Send My $CR$ to the source.
    if Source's Decrypt_and_Verify_CR Algorithm($CR$) == Accept Connection
    then
      Swap.
    else
      Send $Failed_{Connection}$ to the parent.
    end if
  else
    Send $CR$ to the parent.
  end if
end if
```

2. Or the new subscriber successfully decrypts the connection, that means the new subscriber covers the subscription of the connected subscriber → The connected subscriber will swap his position in the tree with this new subscriber, figure 6.11 illustrates the swap case in more details.

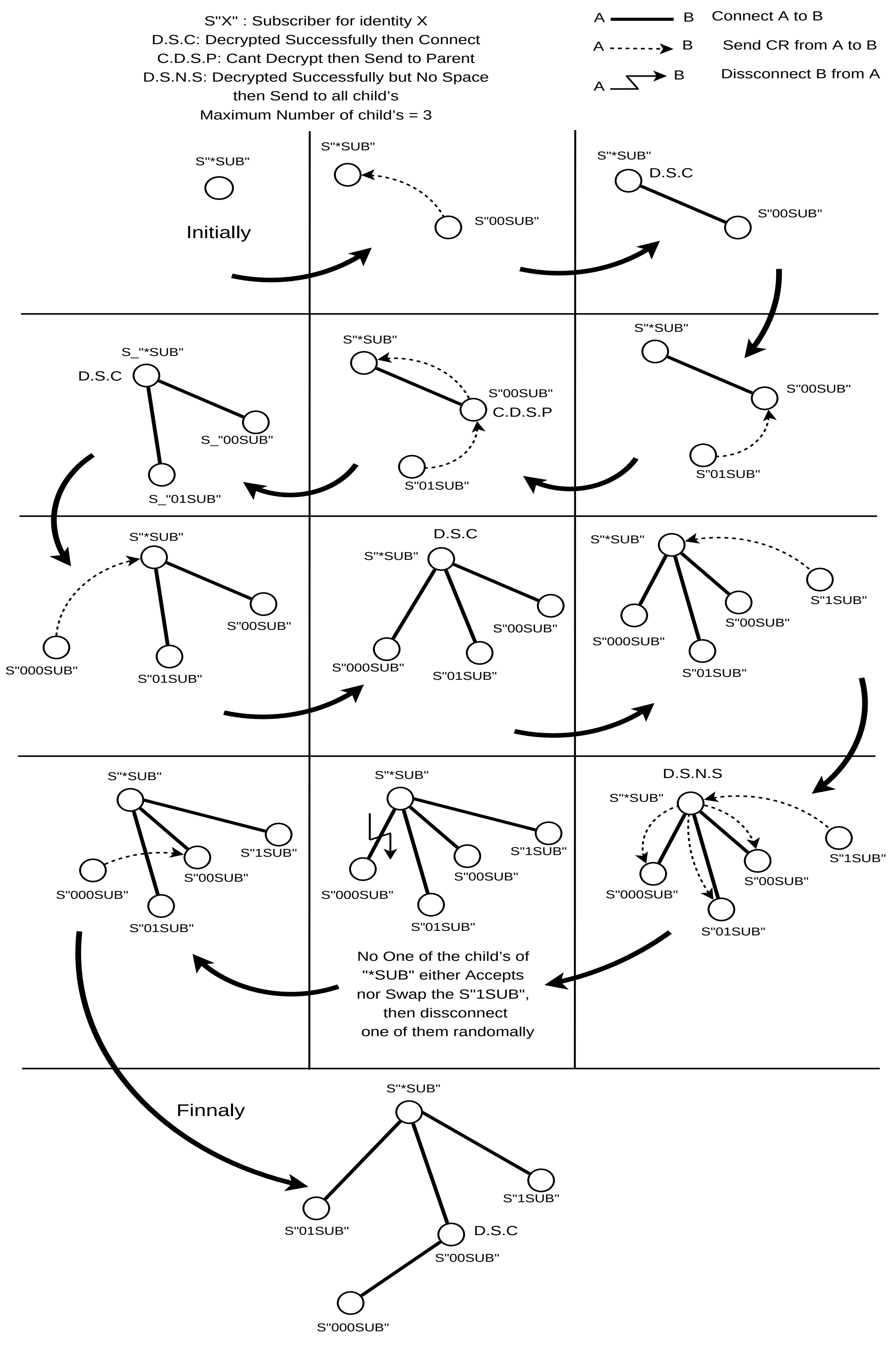


Figure 6.10: Growing Of The Subscribers Tree Using CR

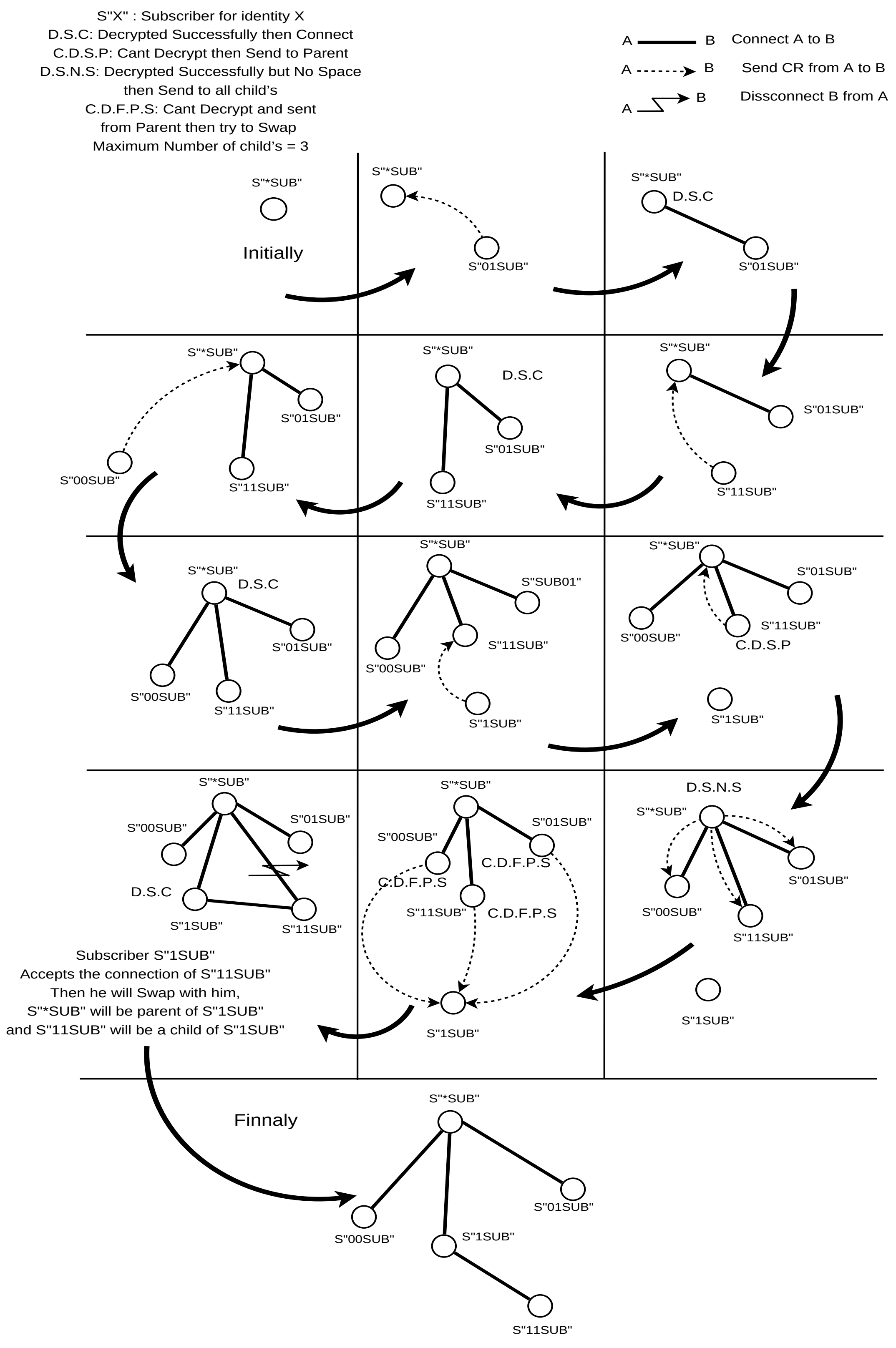

S"X" : Subscriber for identity X
D.S.C: Decrypted Successfully then Connect
C.D.S.P: Cant Decrypt then Send to Parent
D.S.N.S: Decrypted Successfully but No Space then Send to all child's
C.D.F.P.S: Cant Decrypt and sent from Parent then try to Swap
Maximum Number of child's = 3
A B Connect A to B
A B Send CR from A to B
A B Dissconnect B from A
S"*SUB"
Initially
S"01SUB"
D.S.C
S"00SUB"
S"11SUB"
S"SUB01"
S"1SUB"
C.D.S.P
D.S.N.S
C.D.F.P.S
Subscriber S"1SUB" Accepts the connection of S"11SUB" Then he will Swap with him, S"*SUB" will be parent of S"1SUB" and S"11SUB" will be a child of S"1SUB"
Finnaly

Figure 6.11: Swap Case In The Subscribers Tree

Weak Notion Of Confidentiality

Even though our technique above builds the subscriber's tree while keeping their subscriptions confidential, there is a possibility that an attacker able to expect the subscriptions of these subscribers based on the building rule of this tree: higher subscriptions are higher in the tree:

Lemma 6.1 *In case the attacker is the receiver of CR: If he decrypts the sender connection request successfully* $\overset{then}{\rightarrow}$ *the sender subscription is* ***less or equal*** *attacker subscription. Subscription confidentiality decreases when the receiver attacker moves towards leaves in the subscriber tree.*

Lemma 6.2 *In case the attacker is the sender of CR: If he connects to the receiver successfully* $\overset{then}{\rightarrow}$ *the receiver subscription is* ***greater or equal*** *attacker subscription. Subscription confidentiality decreases when the sender attacker moves towards the root in the subscriber tree.*

6.3.3 Events Dissemination

In figure 6.1 we show the subscriber's tree in our system and the publishers who send their events to this subscriber's tree. In the previous section, we introduce the technique responsible for building this subscriber's tree while keeping the subscriptions hidden between these subscribers. In this section, we discuss the mechanisms used by these subscribers in order to forward the events to their destinations.

In our system, the topology of subscriber's tree needs every parent's subscription covers the children's subscription, this topology makes providing an encryption mechanism that guarantees confidentiality of subscription very difficult. The parent can decrypt every event which he forwards to his children, by maintaining histories he can simply discover the children's subscription after a while. However, there are two mechanisms can be used to mitigate this problem:

1. Subscribers can divide their subscription into many regions and connect to many parents $\overset{DrawBacks}{\rightarrow}$ more decryption keys per subscriber and *k-anonymous* subscription regions for each subscriber.

2. Forward all matching events to all children $\overset{DrawBacks}{\rightarrow}$ false positive events may happen, but it will be stopped in the next hop (we use this mechanism in our evaluation).

We present the mechanism of encrypting and signing events applied by publishers in algorithm 6.1 and figure 6.5. Subscribers use algorithm 6.2 to decrypt and verify the received events.

After the publisher builds the desired event, he can forward it to some subscriber selected randomly from the subscriber's tree. When the subscriber receives the event, he depends on the result of algorithm 6.2 to decide whether to forward the received event to his neighbors *(child's and parent)* or not, this is shown in algorithm 6.5.

Algorithm 6.5 (Event_Dissemination_Algorithm), Executed By the Connected Subscriber.
Input: Encrypted-And-Signed-Events Packet.
Output: None.
Comments are between {}.

```
Returned-Result = Decrypt_and_Verify_Event_Algorithm($Input$)

if Returned-Result == Matched-Event then
  Send $Input$ To:
    1. Parent (Unless he is the sender of $Input$).
    2. All Child's (If the sender of $Input$ is one of the children exclude that child).
else
  if The sender of $Input$ is the parent then
    Ignore. {False-positive-event}
  else
    Send $Input$ to parent.
  end if
end if
```

Figure 6.12 shows an already built subscriber's tree, and a publisher sends his event to this tree. Each subscriber executes algorithm 6.5 to determine the next hop to forward the Encrypted-And-Signed-Events packet.

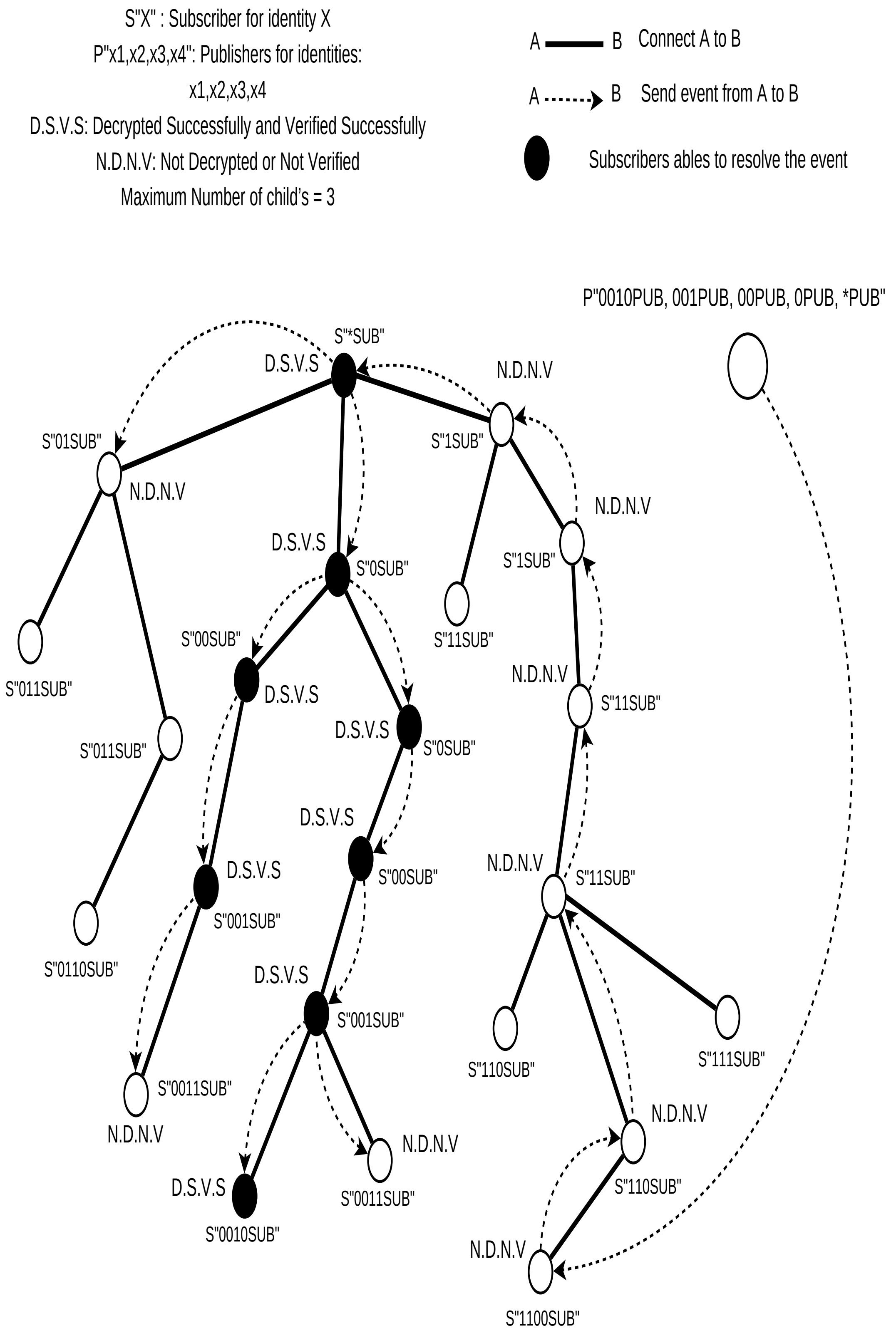

S"X" : Subscriber for identity X
P"x1,x2,x3,x4": Publishers for identities: x1,x2,x3,x4
D.S.V.S: Decrypted Successfully and Verified Successfully
N.D.N.V: Not Decrypted or Not Verified
Maximum Number of child's = 3
A B Connect A to B
A B Send event from A to B
Subscribers ables to resolve the event
P"0010PUB, 001PUB, 00PUB, 0PUB, *PUB"
S"*SUB"
D.S.V.S
N.D.N.V
S"01SUB"
S"1SUB"
S"0SUB"
S"00SUB"
S"011SUB"
S"11SUB"
S"001SUB"
S"0110SUB"
S"0011SUB"
S"0010SUB"
S"110SUB"
S"111SUB"
S"1100SUB"

Figure 6.12: Event Dissemination Example

6.4 Algorithm Properties

Here we show the properties of our techniques which we used to build the subscriber's tree and to disseminate events to the right destinations.

1. **Subscriptions Confidentiality**: The subscriber's tree is built and the events are disseminated to the right destinations without revealing the subscription of any subscriber. CRs are built by TTP and the received subscriber authenticates it using $TTP\_Public\_Parameters \overset{so}{\rightarrow}$ No malicious node can build many fake CRs and use them to discover the subscription of any subscriber. (e.g. the attacker builds many CRs using different $Public_{EncKey}$ and send them one after another until the receiver accepts the connection $\rightarrow$ The subscription of the receiver equals the identity used as a $Public_{EncKey}$ in the CR which the receiver decrypt it.)

2. **Publications Confidentiality**: Any event is encrypted using the $Public_{EncKey}$ of the received subscribers, just the legal subscribers who have the corresponding $Private_{DecKey}$ able to decrypt this event, $\overset{so}{\rightarrow}$ publications of the events are confidential.

3. **Subscriptions Authentication**: When any subscriber joins the system, he asks the TTP for his $Private_{DecKey}$ and CR. TTP checks whether he is a legal subscriber before grant him these secret stuffs. As a result all subscribers in the system are authorized.

4. **Publications Authentication**: Legal publishers gain their $Private_{SignKey}$ keys from the TTP. Subscribers authenticate the received events using $Public_{VerifyKey}$, $\overset{so}{\rightarrow}$ no malicious node can fabricate any fake event because it does not have the corresponding $Private_{SignKey}$ keys.

In the event dissemination technique, there is no routing mechanism used by the subscribers to determine whether the received encrypted event belongs to their children or not. As a result, many false positive events may be received by many subscribers, we will see that in the next section where we evaluate our techniques.

6.5 Forward And Backward Secrecy

Including the time in the subscription is an important feature in many content-based publish-subscribe systems. For example, subscriber wants to subscribe for one month for a specific event. We can easily include the time in our event space by add extra axis that represents the whole **epoch** and divides it as needed. Publishers and subscribers define their identities according to their interested period of time, and they must redefine it for

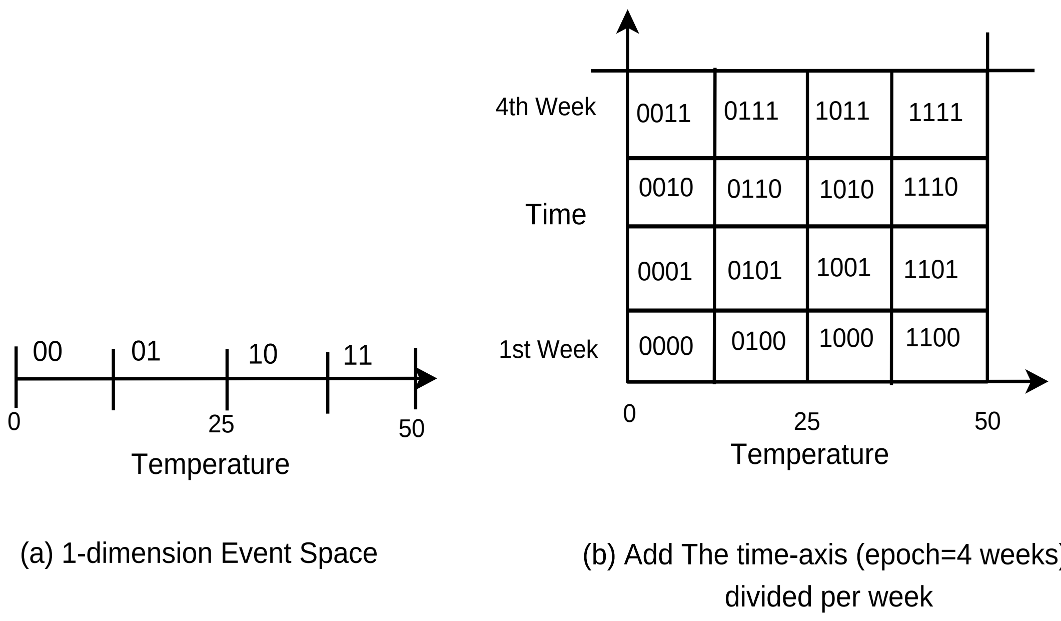


Figure 6.13: Involve the Time-Axis to 1-dimension Event Space

the new epoch. Figure 6.13 illustrates the involving of the time in an event space with one attribute (for simplicity).

This mechanism of modifying the identities and the secret keys of publishers and subscribers according to their interested periods of time guarantees the backward and forward secrecy in our system. For example, the new subscriber will not be able to decrypt the previous events in the system (backward secrecy). And the subscriber who left the system will not be able to discover and decrypt any future event sent after his departure (forward secrecy).

Chapter 7

Evaluation

The Evaluation chapter particularly deals with the question about the overhead resulted by our techniques which we used to build the subscriber's tree and to disseminate the events to its right destinations while achieving the confidentiality of events and subscriptions on one hand, and the authentication of publishers and subscribers, on the other hand. We call our system, which uses these techniques described in the last chapter the *(Secure-system)*. We will compare it with another system where the subscriptions and the events are not confidential and there is no mechanism to authenticate publishers or subscribers, we call this system the *(Unsecured-system)*. In the *(Unsecured-system)* each subscriber knows the subscriptions of his neighbors (parent and children).

7.1 Experimental Setup

In order to simulate our system for extremely large scale number of subscribers and publishers, we use the *PeerSim* [1] simulator with the following simulation setup:

1. The user defines the number of subscribers in the system denoted by *Ns*.
2. The user defines the number of publishers in the system denoted by *Np*.
3. The Event space is divided recursively four times into 31 regions represented by *(*), (0), (1), (00)...(11), (000)...(111), (0,0,0,0)...(1,1,1,1)*.
4. There is one subscriber with identity *"*"* initially in the system.
5. There are two mechanisms of assigning the subscription to the *Ns* subscribers, *random* and *uniform* (discussed later).
6. There is one mechanism of assigning event regions for each publisher (discussed later).

7. Each subscriber has one parent and *Log(Ns)* children.
8. When any subscriber wants to connect to the system, he selects one subscriber from the subscriber's tree randomly.
9. When any publisher wants to publish an event, he selects the destination subscriber form the subscribe's tree randomly.

Assign event space regions to publishers

We have 16 smallest regions in the event space represented by *(0,0,0,0), (0,0,0,1)...(1,1,1,1)*. We assign these regions one per publisher starting from *(0,0,0,0) towards (1,1,1,1)* until we finish assigning to the last publisher (if there are still un-assigned publishers we repeat from *(0,0,0,0)* and so on.

Assign subscriptions to subscribers randomly

For each subscriber, we generate random subscription selected randomly from the *30* regions of the event space (exclude "*").

Assign subscriptions to subscribers uniformly

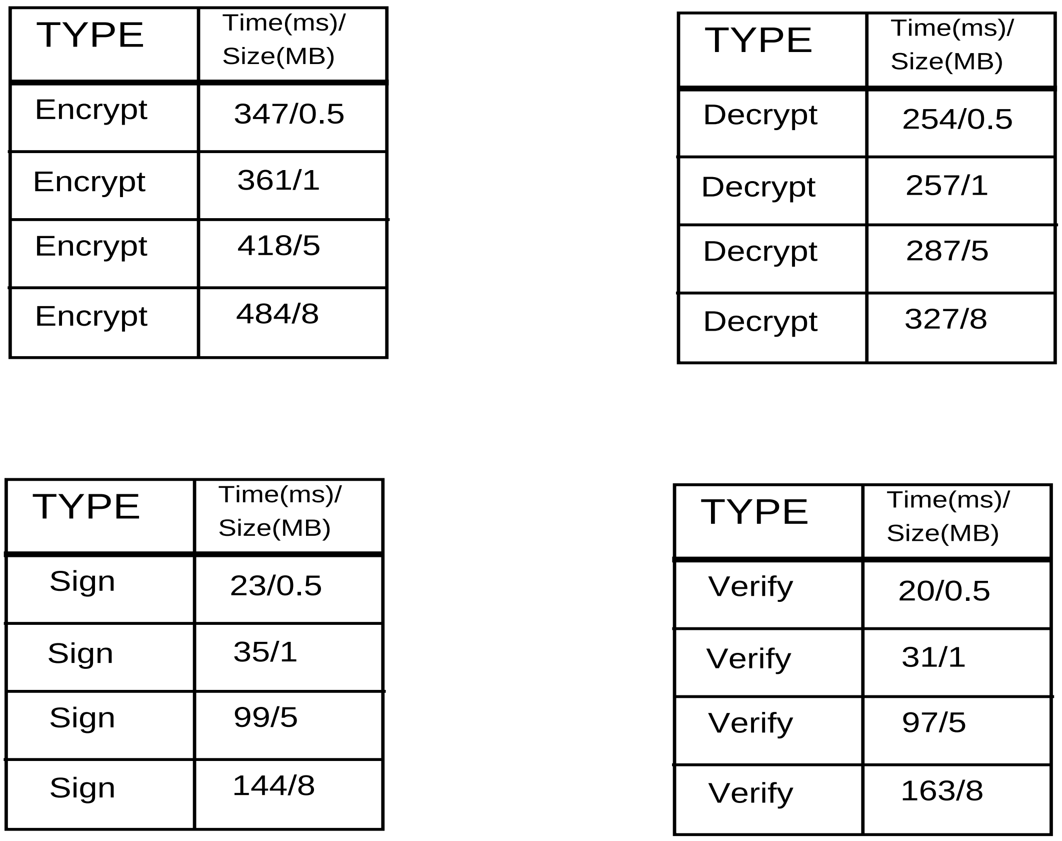
We have four levels of subscriptions according to the levels in the subscription tree:

1. First level: *2* regions with identities *"0"*, *"1"*.
2. Second level: *4* regions with identities *"00"*, *"01"*, *"10"*, *"11"*.
3. Third level: *8* regions with identities *"000"*, *"001"*, ..., *"111"*.
4. Fourth level: *16* regions with identities *"0000"*, *"0001"*, ..., *"1111"*.

We assign subscriptions to subscribers staring from the first level towards the fourth level. At each level, we assign the following number of subscribers *2*, *4*, *6*, *8* respectively until we finish assigning all subscribers. If we reach the fourth level and there are still some unassigned subscribers we repeat from the first level. Assigning subscriptions to the subscribers at each level is done uniformly, for example, if there are *10* subscribers at the first level then *5* will have the identity *"0"* and the other five will have the identity *"0"*. Now, suppose we have a system with *Ns=110* then the subscribers located at these levels like: *12* subscribers for the first level, *24* subscribers for the second level, *34* subscribers for the third level and *40* subscribers for the fourth level. At each level, the subscription regions are assigned uniformly to these subscribers.

Table 7.1: Specifications of the notebook used for evaluation

CPU	RAM	HD	OS	Java
Centrino Duo 2.00 GHz	2GB	107GB	WinXP SP2	JRE6



TYPE	Time(ms)/ Size(MB)
Encrypt	347/0.5
Encrypt	361/1
Encrypt	418/5
Encrypt	484/8

TYPE	Time(ms)/ Size(MB)
Decrypt	254/0.5
Decrypt	257/1
Decrypt	287/5
Decrypt	327/8

TYPE	Time(ms)/ Size(MB)
Sign	23/0.5
Sign	35/1
Sign	99/5
Sign	144/8

TYPE	Time(ms)/ Size(MB)
Verify	20/0.5
Verify	31/1
Verify	97/5
Verify	163/8

Figure 7.1: Throughput Of IBE-Java Library

To obtain the results, we use a notebook with the specifications denoted at table 7.1, also we use an IBE-java library *(nuim.cs.crypto which is supported by NUI Maynooth Crypto Group)* with throughputs shown in figure 7.1:

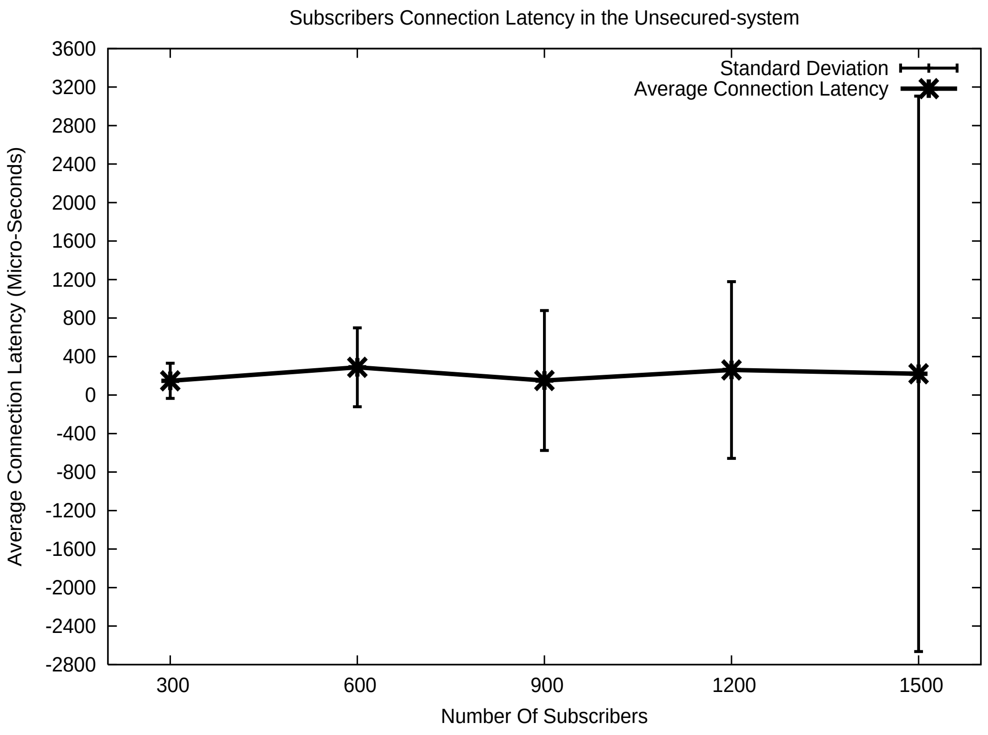


Figure 7.2: Subscribers Connection Latency in the *(Unsecured-system)*

7.2 Evaluation Results

In the following, the evaluation results are shown, starting with the presentation of the connection latency for the subscribers, then the event dissemination latency and the ratio of the false positive events resulted from our techniques. Finally, we present the ratio of the saved nodes which our system protects them from an DoS attacker. The values shown in the figures represent the mean values of these measurements. The standard deviations are represented by the vertical line at each value.

For subscribers connection and events dissemination evaluations, we compare the latency between the *(Secure-system)* and *(Unsecured-system)*.

7.2.1 Subscribers Connection

In the subscriber's connection evaluation, we used the uniform way of assigning subscriptions to the subscribers which we have shown in the *Experimental Setup* section. We calculate the average time needed by each subscriber to connect to the system in both *(Unsecured-system)* and *(Secure-system)*, figures 7.2 and 7.3 represent the latency in both systems respectively.

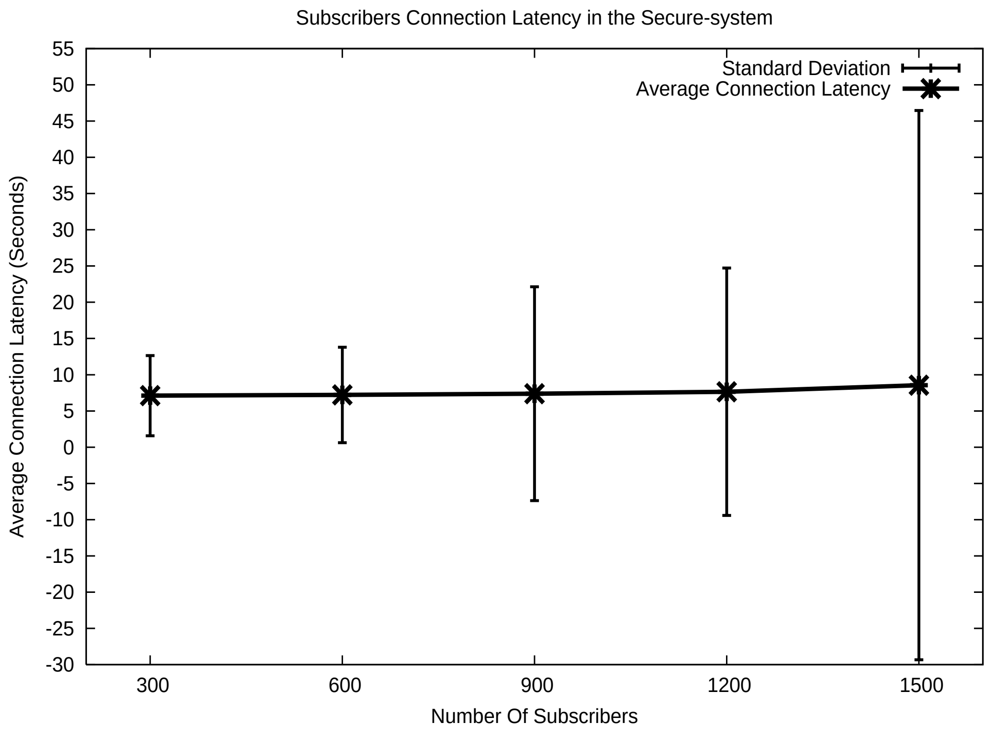


Figure 7.3: Subscribers Connection Latency in the *(Secure-system)*

As shown in these figures the latency of subscribers to connect the system increases when the *(Unsecured-system)* becomes *(Secure)* (from micro-seconds to seconds), that is resulted from the encryption, decryption, signing and verification actions of connection requests (CRs) needed in the *(Secure-system)*. In both figures, we can note that the average connection latency per each subscriber increased linearly when the number of subscribers increased *(300 to 1500)*, that is because there are more hops to reach the right location to connect in the subscriber's tree in case of the larger system.

The standard deviation increases when the number of subscribers increased. Subscribers join the system early suffers of small latency (few subscribers $\rightarrow$ Few hops to reach the right subscriber to connect). However, subscribers who join the system after many others already connected will suffer higher latency because they need to route their connection requests to many hops to reach the right destinations. The number of hops in the large systems is higher than the systems with few subscribers $\rightarrow$ The standard deviation value increases when the number of subscribers increased in both figures.

The change of the connection latency is linear when the number of subscribers increased in the *(Secure-system)* $\overset{so}{\rightarrow}$ *our technique of building the (Secure-system) is scalable enough to be applied on large systems with large number of subscribers.*

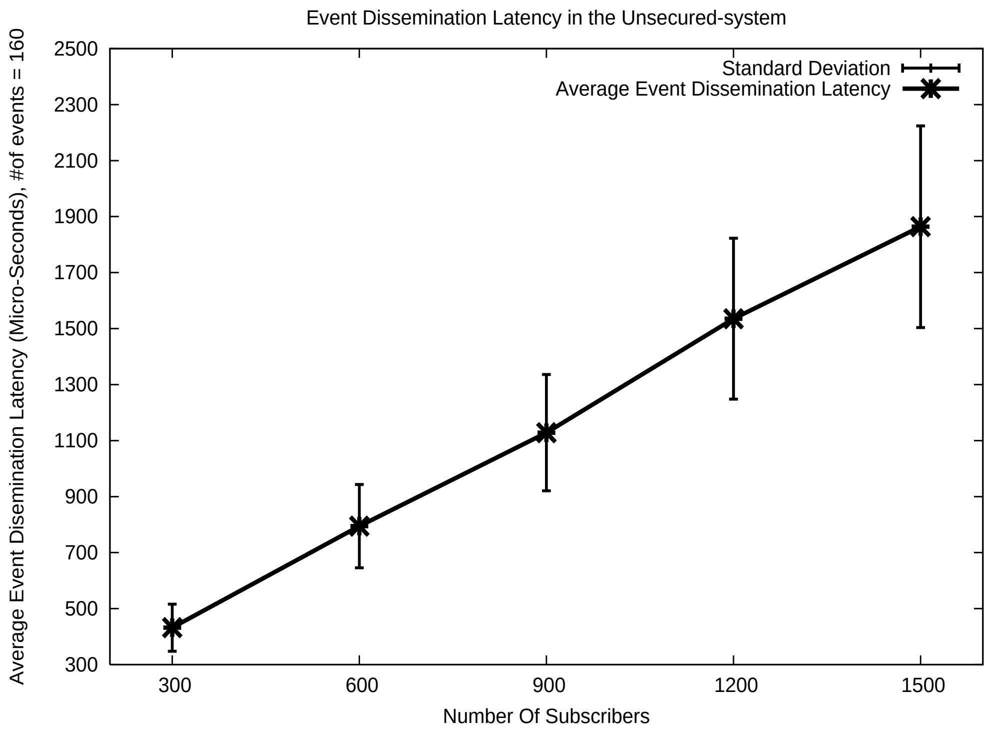


Figure 7.4: Event Dissemination Latency in the *(Unsecured-system)*

7.2.2 Events Dissemination

In the events dissemination evaluation, we used the uniform way of assigning subscriptions to the subscribers (*Experimental Setup* section), firstly the subscriber's tree is built using the secure connection technique, and after all subscribers connect to the system, publishers start sending their events. We use the mechanism of assigning event space regions to the publishers which defined in the *Experimental Setup* section. We have *160* publishers assigned uniformly to the smallest regions in the event space, so there are *10* publishers per each region. Each publisher publishes one event, so we have *160* events sent to the system and all these events disseminated to the right subscribers. We calculate the average time needed by each event to reach *(decrypted and verified)* to it's right subscriber in both *(Unsecured-system)* and *(Secure-system)*, figures 7.4 and 7.5 represent the latency in both systems respectively.

As shown in these figures the event dissemination latency increases when the *(Unsecured-system)* becomes *(Secure)* (from micro-seconds to seconds), that is resulted from the encryption, decryption, signing and verification actions of events needed in the *(Secure-system)*. The event latency needed in the *(Secure-system)* is shown in figure 7.5, this latency increases *(14 to 64 seconds)* when the number of subscribers increased *(300 to 1500)*, that is because in case of the larger systems the events routed into more hops before it reaches the right destinations. For the same reason, the standard deviation increases

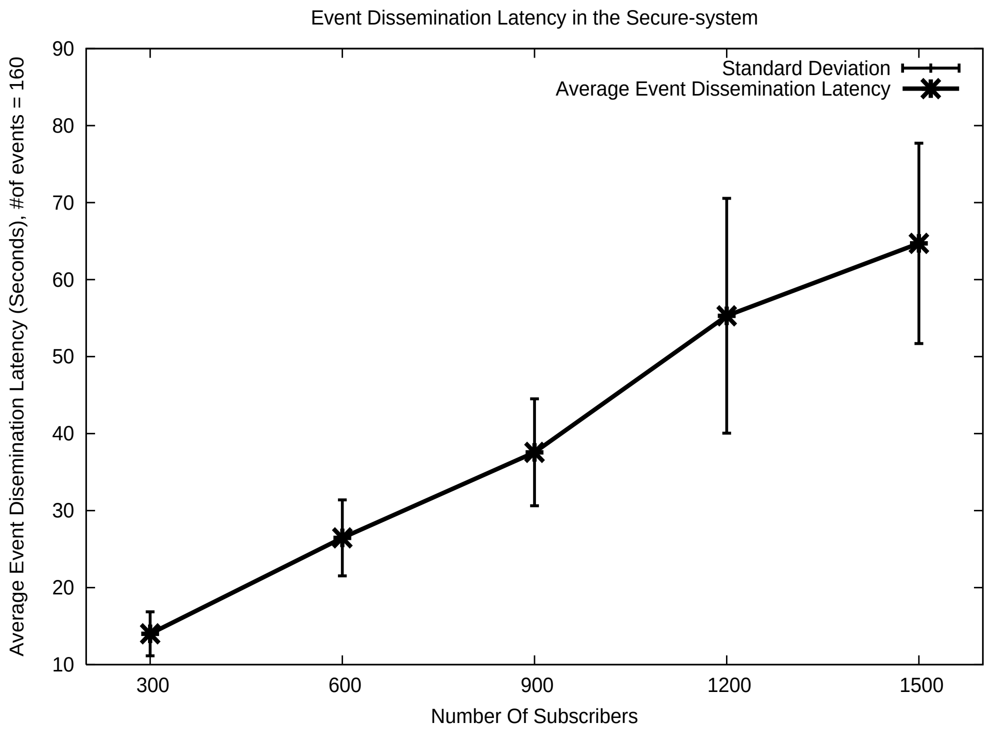


Figure 7.5: Event Dissemination Latency in the *(Secure-system)*

when the number of subscribers increased.

By using the same mechanism of assigning event space regions to the publishers and use the random mechanism of assigning the subscriptions to the subscribers as presented in *Experimental Setup* section, we calculate the ratio of false positive events received by each subscriber. We calculate that for 1600 publishers *(100 for each smallest region in the event space and each one sends one event* $\overset{so}{\rightarrow}$ There are 1600 events sent to the system). Figure 7.6 shows that. The ratio of false positive events received per each subscriber is relatively high because there is no mechanism for routing used in our event dissemination technique. The parent subscriber sends all matched events (decrypted and verified) to all children excluding the sender (see algorithm 6.5 in the last chapter).

7.2.3 Securing against DoS Attackers

In case we have attacker subscribers in the system who fabricates some events and overwhelms the system of these events. Other subscribers in the system must be able to discover these fake messages and stop forwarding them. In case of the *(Unsecured-system)*, these events will never be discovered and stopped, so any subscriber receives this event will handle it as a legal one and forward it to his neighbors $\overset{so}{\rightarrow}$ *Overhead of processing of these fake events on each subscriber receives it.* On the other hand, in our *(Secure-*

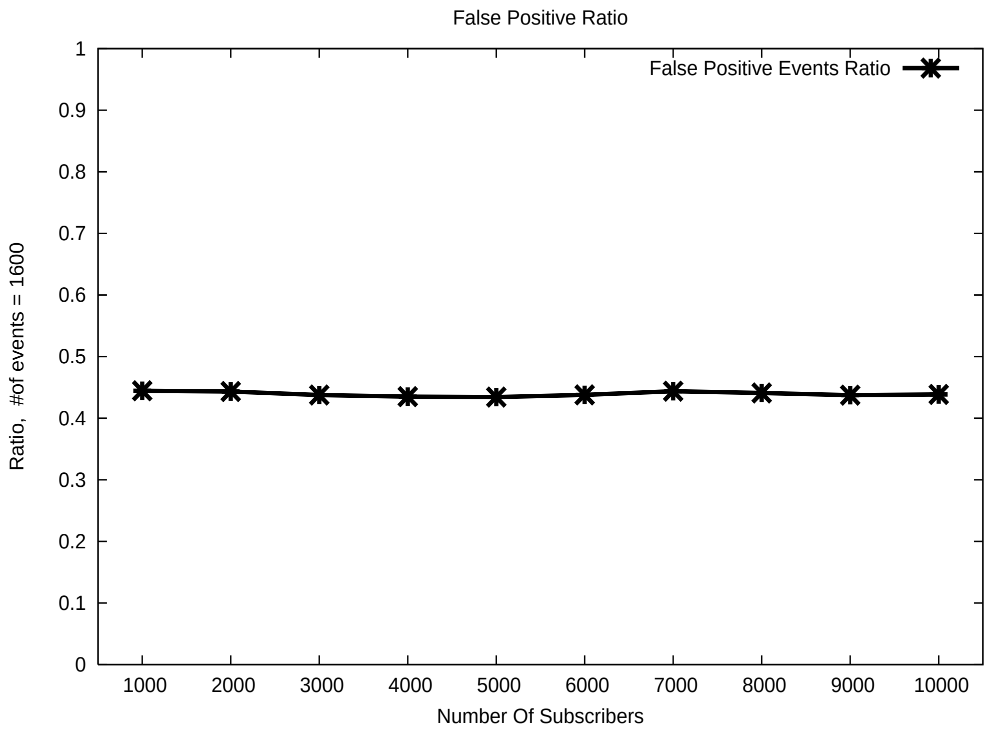


Figure 7.6: False Positive Ratio in the *(Secure-system)*, with 1600 Publishers

system), the attacker can build the fake event by encrypt it using $Public_{EncKey}$ but he can not sign it by the $Private_{SignKey}$ which is just known to the legal publishers $\overset{so}{\rightarrow}$ When any legal subscriber successfully decrypts this event he can discover that this event is fake *(by verify it using the corresponding $Public_{VerifyKey}$)*. This legal subscriber will not forward this fake event further. In case the subscriber unable to decrypt this fake event, he will forward is to his parent (look algorithm 6.5 in the last chapter).

In order to calculate the ratio of subscribers our secure technique saves them of receiving these fake events, we consider the following system:

1. *1000* subscribers *(the subscription is assigned uniformly)* connected to each others.
2. There is a ratio that defines the number of attackers of these subscribers.
3. These attackers randomly connected to the system.
4. Each one of these subscribers fabricates one event and attacks the system by sending it to all neighbors of him (parent and children), for example, in case the ratio of attackers equals *0.1* there are *100* fake events.

We calculate the number of attacked subscribers in both *(Unsecured-system)* and *(Secure-system)*. Figure 7.7 represents the ratio of subscribers whom our *(Secure-system)*

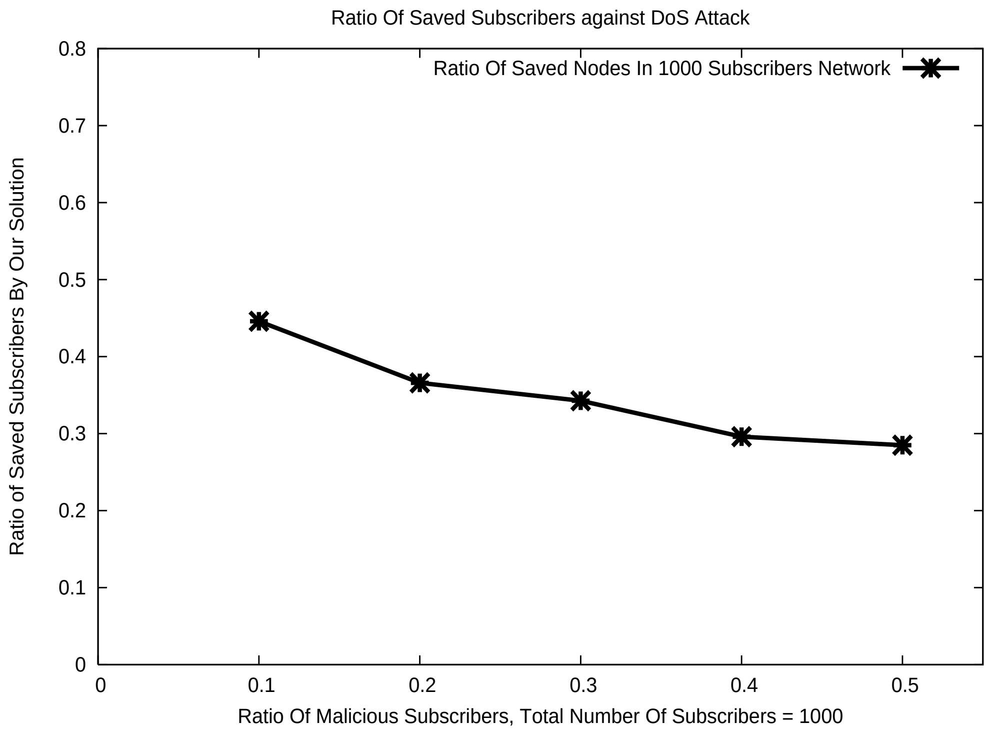


Figure 7.7: Ratio Of Saved Subscribers In *(Secure-system)* Compared To *(Unsecured-system)*

saves them of these fake events. For example, in the *(Unsecured-system)* the number of attacked subscriber = 100, and it is 70 in the *(Secure-system)* $\Rightarrow$ The ratio of saved nodes equals $\frac{(100-70)}{100} * 100\% = 30\%$.

Looking at figure 7.7 we can notice that the ratio of saved subscribers varies between *(0.45 to 0.30)*. The subscribers which our *(Secure-system)* can not save them of receiving fake events are unable to discover these fake events because they can not decrypt it. So this fake event will be forwarded until any subscriber decrypts it and discovers that it is not signed by the legal publisher and stops it. The ratio of saved subscribers decreases when the ratio of attackers increased because there are more fake events overwhelmed to the system by these attackers.

Chapter 8

Conclusion And Future Work

In this thesis, a new approach for achieving confidentiality of events and subscriptions, and authentication of publishers and subscribers in broker-less content-based pub/sub system built on P2P architecture is presented. This approach enables subscribers and publishers authenticate each other's while they are decoupled. Moreover, events confidentiality is guaranteed by allowing only legal subscribers resolve the sent events. Also, the subscribers are connected and clustered according to their interests without revealing their subscriptions.

This approach uses the identity based encryption (IBE) scheme to achieve the needed confidentiality and authentication. By cooperating with the trusted-third-party (TTP), four types of keys are generated in the system: private-sign-keys, public-verify-keys, public-encryption-keys and private-decryption keys.

Publisher's authentication is achieved by dedicating the private-sign-keys only to legal publishers, these keys are used to sign the sent events, and the subscriber authenticates the received event using the public-verify-key. On the other hand, authentication of subscribers is guaranteed because TTP assures the identity of any subscriber before grant him the private-decryption-key.

Moreover, publishers encrypt the events logarithmic times by public-encryption-keys, because the private-decryption-keys are only known to legal subscribers $\rightarrow$ Event confidentiality is guaranteed.

Based on his subscription, the new subscriber connects to the right position in the subscriber's tree without revealing his subscription. The trusted-third-party builds a connection request (CR) with a logarithmic size for this new subscriber. This new subscriber will connect to some subscriber in the subscriber's tree without knowing his subscription and vice versa. However, any attacker can expect the subscription possibilities of his neighbors based on the position of this attacker in the subscriber's tree.

In the evaluation chapter, the latencies for connection of any subscriber to the system and dissemination of the events to its right destinations are calculated, the results show there is a slight overhead by executing our techniques.

Up to now, we consider subscription maps to a single region in the event space but subscription with arbitrary ranges may map to many regions. This will increase the number of keys managed by a subscriber. Future research is to modify the approach to reduce the number of keys managed by subscriptions mapping to many regions.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] PeerSim Simulator, *PeerSim HOWTO: Build a new protocol for the PeerSim 1.0 simulator*. http://peersim.sourceforge.net/tutorial1/tutorial1.html.

[2] Mudhakar Srivatsa and Ling Liu, *Securing Publish-Subscribe Overlay Services with EventGuard*. College of Computing, Georgia Institute of Technology, mudhakar@cc.gatech.edu, lingliu@cc.gatech.edu

[3] Shou-Chih LO, *Design of Content-Based Publish/Subscribe Systems over Structured Overlay Networks*.

[4] Vinod Muthusamy and Hans-Arno Jacobsen, *Small-Scale Peer-to-Peer Publish/Subscribe*. Middleware Systems Research Group, Department of Electrical and Computer Engineering, Department of Computer Science, University of Toronto vinod,jacobsen@eecg.toronto.edu.

[5] Mudhakar Srivatsa and Ling Liu, *Secure Event Dissemination in Publish-Subscribe Networks*. College of Computing, Georgia Institute of Technology fmudhakar, lingliug@cc.gatech.edu.

[6] Jun Li, Chenghuai Lu and Weidong Shi, *An Efficient Scheme for Preserving Confidentiality in Content-Based Publish-Subscribe Systems*. College of Computing, Georgia Institute of Technology.

[7] Adrian Perrig, Dawn Song and J. D. Tygar, *ELK, a New Protocol for Efficient Large-Group Key Distribution*. University of California Berkeley fperrig,dawnsong,tygarg@cs.berkeley.edu.

[8] Joan Daemen and Vincent Rijmen, *AES Proposal: Rijndael*. Joan Daemen, Belgium, daemen.j@protonworld.com. Vincent Rijmen, Belgium, vincent.rijmen@esat.kuleuven.ac.be.

[9] Jun Xu, Jinliang Fan, Mostafa H. Ammar and Sue B. Moon, *Prefix-Preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme*. College of Computing Georgia Institute of Technology, Atlanta.

[10] John Bethencourt, T-H. Hubert Chan, Adrian Perrig, Elaine Shi and Dawn Song, *Anonymous Multi-Attribute Encryption with Range Query and Conditional Decryption.* School of Computer Science Carnegie Mellon University.

[11] Costin Raiciu and David S. Rosenblum, *Enabling Confidentiality in Content-Based Publish\Subscribe Infrastructures.* Department of Computer Science, University College London, c.raiciu|d.rosenblum@cs.ucl.ac.uk.

[12] Group 15, *Identity-Based Encryption.* November 20, 2007.

[13] Dan Boneh and Matthew Frankliny, *Identity-Based Encryption from the Weil Pairing.* dabo@cs.stanford.edu, franklin@cs.ucdavis.edu.

[14] Brent Waters, *Efficient Identity-Based Encryption Without Random Oracles.*

[15] A. Shamir, *Identity-based cryptosystems and signature schemes.* Proceedings of CRYPTO 84 on Advances in cryptology table of contents, pages 47-53, 1985.

[16] C. Cocks, *An identity based encryption scheme based on quadratic residues.* Proceedings of the 8th IMA International Conference on Cryptography and Coding, 2001.

[17] Dan Boneh and Xavier Boyen, *Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles.* Stanford University, dabo@cs.stanford.edu. Voltage Security, xb@boyen.org.