Securing Safebook

Secure Data Access Control and Key Management for Safebook

WAQAS LIAQAT ALI

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Securing Safebook

Secure Data Access Control and Key Management for Safebook

Waqas Liaqat Ali

waqasl@kth.se

2013.08.25

Academic adviser and examiner: Prof. Gerald Q. Maguire Jr.
Supervisor at Eurecom: Dr. Melek Önen & Dr. Leucio Antonio Cutillo

Communication Systems
School of Information and Communications Technology
KTH Royal Institute of Technology
Stockholm, Sweden

Abstract

Online social networks have become a fast and efficient way of sharing information and experiences. Over the past few years the trend of using social networks has drastically increased with an enormous amount of users' private contents injected into the providers' data centers. This has raised concerns about how the users' contents are protected and how the privacy of users is preserved by the service providers. Moreover, current social networks have been subject to much criticism over their privacy settings and access control mechanism. The providers own the users' contents and these contents are subject to potential misuse. Many socially engineered attacks have exposed user contents due to the lack of sufficient privacy and access control. These security and privacy threats are addressed by Project Safebook, a distributed peer-to-peer online social networking solution leveraging real life trust. By design Safebook decentralizes data storage and thus the control over user content is no longer in the service provider's hands. Moreover, Safebook uses an anonymous routing technique to ensure communication privacy between different users.

This thesis project addresses privacy aware data management for Safebook users and a data access control solution to preserve users' data privacy and visibility utilizing a peer-to-peer paradigm. The solution focuses on three sub-problems: (1) preserving the user's ownership of user data, (2) providing an access control scheme which supports fine grained access rights, and (3) secure key management. In our proposed system, the user profile is defined over a collection of small data artifacts. An artifact is the smallest logical entity of a profile. An artifact could be a user's status tweak, text comment, photo album metadata, or multimedia contents. These artifacts are then logically arranged to form a hierarchical tree, call the User Profile Hierarchy. The root of the profile hierarchy is the only entry point exposed by Safebook from where the complete user profile can be traversed. The visibility of portions of the user profile can be defined by exposing a subset of profile hierarchy. This requires limiting access to child artifacts, by encrypting the connectivity information with specific access keys. Each artifact is associated with a dynamic access chain, which is an encrypted string and contains the information regarding the child nodes. A dynamic access chain is generated using a stream cipher, where each child's unique identifier is encrypted with its specific access key and concatenated to form the dynamic access chain. The decryption process will reveal only those child artifacts whose access keys are shared. The access keys are managed in a hierarchical manner over the profile hierarchy. Child artifacts inherit the parent's access key or their access key can be overridden with a new key. In this way, fine grained access rights can be achieved over a user's artifacts. Remote users can detect changes in a specific branch of a profile hierarchy and fetch new artifacts through our proposed profile hierarchy update service. On top of the proposed access control scheme, any social networking abstraction (such as groups, circles, badges, etc.) can be easily implemented.

Keywords: User Privacy, Access Control, Distributed Social Networks, Key Management

Sammanfattning

Online sociala nätverk har blivit ett snabbt och effektivt sätt att dela information och erfarenheter. Under de senaste åren har trenden med att använda sociala nätverk har ökat drastiskt med en enorm mängd av användarnas privata innehåll injiceras in i leverantörernas datacenter. Detta har väckt farhågor om hur användarnas innehåll skyddas och hur användarnas integritet bevaras av tjänsteleverantörerna. Dessutom har nuvarande sociala nätverk varit föremål för mycket kritik över sina sekretessinställningar och åtkomstkontroll. Leverantörerna äger användarnas innehåll och dessa innehåll är föremål för potentiellt missbruk. Många socialt konstruerade attacker har utsatt användarnas innehåll på grund av bristen på tillräcklig integritet och åtkomstkontroll. Dessa säkerhets-och privatliv hot hanteras av Project Safebook, en distribuerad peer-to-peer sociala nätverk online-lösning utnyttja verkliga livet förtroende. Genom design Safebook decentralizes datalagring och därmed kontrollen över användarens innehåll är inte längre i tjänsteleverantörens händer. Dessutom använder Safebook en anonym routing teknik för att säkerställa kommunikationen sekretess mellan olika användare.

Detta examensarbete behandlar sekretess medvetna datahantering för Safebook användare och åtkomstkontroll lösning för att bevara användarnas integritet och synlighet använder en peer to peer paradigm. Lösningen fokuserar på tre delproblem: (1) bevara användarens ägande av användardata, (2) att tillhandahålla ett system för åtkomstkontroll som stöder finkorniga åtkomsträttigheter, samt (3) säkra nyckelhantering. I vårt föreslagna systemet, användaren profilen som definieras över en samling av små data-artefakter. En artefakt är det minsta logisk enhet i en profil. En artefakt kan vara en användares status tweak, text kommentar, fotoalbum metadata, eller multimedieinnehåll. Dessa artefakter då är logiskt ordnade att bilda ett hierarkiskt träd, ring Användarprofil Hierarki. Roten till profilen hierarkin är den enda inkörsporten exponeras genom Safebook varifrån hela användarprofil kan passeras. Synligheten av delar av användarprofilen kan definieras genom att exponera en delmängd av profilen hierarki. Detta kräver att begränsa tillgången till barn artefakter, genom att kryptera uppkopplingen informationen med särskilda snabbtangenter. Varje artefakt är associerad med en dynamisk tillgång kedja, som är en krypterad sträng och innehåller information om de underordnade noder. En dynamisk tillgång kedjan genereras med hjälp av en ström chiffer, där varje barns unika identifierare är krypterad med dess specifika tillgången knapp och sammanfogas för att bilda den dynamiska tillgång kedjan. Dekrypteringsprocessen avslöjar endast de barn artefakter vars tillgång nycklar delas. De snabbtangenter hanteras på ett hierarkiskt sätt över profilen hierarkin. Barn artefakter ärva föräldrarnas tillgång nyckel eller deras åtkomstnyckeln kan åsidosättas med en ny nyckel. På detta sätt kan finkorniga åtkomsträttigheter uppnås över en användares artefakter. Fjärranvändare kan upptäcka förändringar i en viss gren av en profil hierarki och hämta nya artefakter genom vår föreslagna profil hierarki uppdateringstjänst. Ovanpå den föreslagna åtkomstkontroll system kan alla sociala nätverk abstraktion (t.ex. grupper, cirklar, märken, osv.) lätt genomföras.

Nyckelord: Användarnas Privatliv, Atkomstkontroll, Distribuerade Sociala Nätverk, Nyckelhantering

“Dedicated to my Parents, Family & Beloved Wife”

Acknowledgements

First, all praises and thanks to Allah Almighty. Without His countless blessing this thesis would not have been possible.

I would like to thank and highly acknowledge the support and wisdom of my immediate thesis supervisor Dr. Melek Onen & Dr. Cutillo at Eurecom France and express my deepest gratitude for their valuable guidance, assistance and inspiration during the course of this thesis project. My sincere thanks to Dr. Melek for believing in me and offering an opportunity to work with Eurecom.

I extend my deepest gratitude to my KTH supervisor, Professor Gerald Q. Maguire Jr. for his guidance throughout this thesis work. His valuable comments and supervision helped me to write this thesis work and improve the overall format of the report. Without his guidance and input this thesis would not have been possible.

I would also like to thank all the people whom I met in Stockholm and made my stay wonderful. Among all of them I would like to express my gratitude to Ahmad Kamal Mirza, Fahad Azeemi, Shariq Mobeen, Waqas Nouman and Muhammad Muaz for their moral support and guidance. I would like to thank Ahmed and Fahad for their valuable comments and inputs in this thesis works.

From my core of heart, special thanks to my parents and family for their unconditional love and support. Without their moral support and love this would not have been possible. Special thanks to my beloved wife for believing in me and encouraging throughout this thesis. I would not have been the same without their support and love.

Thank you all.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ABAC	Attribute Based Access Control
ACK	Acknowledgement
ACL	Access Control List
ACM	Access Control Matrix
API	Application Programming Interface
AS	Application Services
CRL	Certificate Revocation List
DAC	Dynamic Access Chain
DHT	Distributed Hash Table
EPL	Entry Point List
GUI	Graphical User Interface
LAN	Local Area Network
OSN	Online Social Network
PDV	Profile Dynamic View
PKI	Public Key Infrastructure
PPI	Personal Identifiable information
P2P	Peer-to-Peer
RBAC	Role-Based Access Control
SNA	Social Network Application
SNP	Social Network Provider
SNS	Social Networking Services
SSL	Secure Socket Layer
TBAC	Task Based Access Control
TMAC	Team Bases Access Control
TIS	Trusted Identification Service
TTP	Trusted Third Party
UPnP	Universal Plug and Play
WAN	Wide Area Network

1 Introduction

With the evolution of Internet and the information age, the way that the user interacts with the Internet has evolved from a document centric view to very complex social interactions. These interactions and services are based upon the social behavior of a user in a virtual world. This user behavior is very similar to the way the user interacts with other entities surrounding him/her in the user's social life. Users in the real world display very dynamic and versatile behaviors. They interact with other persons, communicate and exchange their thoughts, discuss different topics and issues, make new friends, share stories, and participate in adventures. In this way they socialize in the real-world. Due to the user's diverse behavior, a new concept has emerged in the Internet known as an Online Social Network (OSN). In my opinion the OSN concept can be expressed as:

An OSN is a set or suite of services that can model the social behavior of a user in a virtual world.

In order to model the social behavior of a user in a virtual world, we need some services or a platform which reflects the social network or social relationships among the different entities. Such services are often called Social Networking Services (SNSs). These services have revolutionized interaction patterns within the worldwide web and have become the most prominent services that the Internet has to offer*[1]*. With the available online services, even users with very limited technical skills can publish their personal data, share their interests, participate in different activities, and interact with other people with extreme ease *[1]*.

In the current internet, OSNs usually utilize a centralized web based approach to SNS. These services define the interaction style of the OSN and usually contain a subset of relationships from the physical world that users interact with *[1]*. Every user in the OSN has a set of relationships to other users, often referred to generically as "friends". These relationships can be spanned to form a social network graph, which can be traversed by users to explore/experience more users, thus making socializing easy and more interesting. The user usually creates an OSN profile, which is a virtual persona of a person and represents his/her interests and activities. The OSN applications and services facilitate sharing and injecting information into the OSN. This information can be viewed by or interacted with by other people (and software). The type of service usually depends upon the objective of the specific OSN. For example, an OSN for professional use is more business oriented and might offer career management services, knowledge sharing, business contacts, etc. Whereas an OSN for personal /private use will focus more on entertainment and sharing of private information, such as pictures, posts, contacts, videos, etc. *[1]*. In general starting to utilize an OSN begins with building a profile, then adding and maintaining a friends or contacts list. Users are kept up-to-date with their friend's activities by receiving notifications, which users publish to a selected group or to all the friends in their contacts list, when a user injects new data into the OSN. Along with this basic service, additional services such as messaging, instant chat, tagging, etc. are continuously evolving and being integrating into OSNs to enhance user satisfaction and to attract more users *[2]*.

Operating an OSN implies handling a huge amount of data injected by a diverse range of users. This data is often of a very sensitive nature, such as Personal Identifiable Information

(PPI), pictures, videos, text, etc. The users place a great deal of trust in their OSN service provider. A large amount of this user data is stored either in databases hosted and controlled by SNS providers or by third party service providers. This data is either accessible by the public or in some cases the SNS provider offers some (often configurable) access and privacy rules to that a user can use to control what portion of their data is visible to selected users or to all users. In some cases the user is aware of the potential for privacy problems and is offered some means to control access, however – this may or may not actually provide the protection that the user desires. The existence of an OSN depends upon a trust relationship between the users and their SNS provider. The users trust the provider and assume that their personal data will not be misused and that the privacy of each user will be respected. Secondly, each profile in the OSN is attributed to a person and in many cases the user believes and values the profile with same level of trust as he or she would trust the person in real life. Therefore any action and interactions with a profile are attributed to the presumed owner of the profile in the real world *[3]*.

The trust properties and interactive information exchange interface is owned by the SNS provider. This ownership may add to a user's satisfaction with the OSN, but may also pose a high risk of security threats to very sensitive user data. Studies have shown that users perceive many SNSs as offering weak security and that the SNS providers try to minimize the magnitude of this risk by allowing users to limit the visibility of their data and by other security measures, however many security loopholes can be exploited by parties with adverse interests *[4, 5]*. The exploitation of user data can be for commercial/business purposes or it can be plain malicious. The Wall Street Journal found that "the 10 most popular apps on Facebook were transmitting users' IDs to outside companies" *[6]*. Another factor which facilitates social engineering attacks is the lack of awareness of the risks by the users of the SNSs, thus people use these services every day without thinking of the potential security and privacy risks of a simple action such as accepting a friendship request from an unknown person or allowing an un-trusted third party application to access your data as these persons or applications could publish *their* choice of content on your wall and communication with your contacts, tag pictures, and much more *[1]*. Moreover, the *implicit* trust assumption about the OSN by users and their profiles adds to the problem *[3]*.

An analysis of privacy problems in current OSNs shows that, if all the parties are fully aware of potential privacy risks and competent to use SNS and the SNS providers implement sufficient privacy settings, then the OSN is still exposed to potential privacy violations via an omniscient service provider or an external/internal attacker who gains access to the OSN's databases *[1]*. All the data, in form of PPI, images, posts, messages, notes, and other user activities of participating entities in the OSN, are permanently stored in the database(s) of the provider, which potentially becomes a "big brother" who could exploit this huge collection of sensitive and valuable data in many ways and forms, that may violate the privacy of individuals or groups of participants *[3]*. The importance and sensitivity of this potential misuse can be comprehended by considering the market capitalization of these service providers. In 2005, the News Corp. acquisition of MYSPACE valued this OSN as being worth US$580 million[*]. Facebook Inc. is estimated to be worth US$15 billion *[7]*. Additionally, commercial activities, third party applications (which are very commonly integrated with a SNS to generate revenue), and the SNS provider's systems can be compromised by attackers, and together with attacks on the user profiles can result in disclosure of the users' private data. Such attacks have already happened *[8]*. Additionally, a

[*] As of In June 2011, Myspace was purchase for US$35M.

disgruntled employee might abuse their access to the SNS systems in order to exploit the user's data for their own personal goals or interests.

To address the above security risks in current OSN solutions, Leucio Antonio Cutillo and Refik Molva suggested a new approach to tackle the risk of privacy exposures of user data and potential misuse by the service provider by designing a distributed decentralized peer-to-peer (P2P) architecture. This architecture relies on mutual cooperation among a number of participating entities, who are also users of the OSN and by leveraging real life trust relationships among users to build trust in the system and to ensure a secure online privacy preserving mechanism *[3]*. These two design principles lead to their solution called **Safebook**, a decentralized P2P privacy preserving application.

Their proposed solution focuses on three aspects of OSN:

Decentralization	The objective is to provide an alternative solution to all of the data services provided by centralized servers (usually owned by OSN service providers or third party service providers). Services such as injecting information into the OSN, retrieving your own or a friend's data, lookup services, routing, secure communication, etc. are implemented using a P2P design, where every user in the OSN maintains some part of the P2P network's topology and participates in network management and operations at a micro level to ensure data distribution, retrieval, and availability.
Privacy Preserving Mechanism	On top of the decentralization layer, Safebook proposes a set of secure protocols for: *(a) Secure Data Management:* injecting and retrieving data without exposing the user's privacy and to ensure the integrity of data using encryption and authentication schemes; *(b) Communication:* prevents some potential attacks by providing end-to-end and hop-by-hop encryption & authentication and by utilizing anonymous routing techniques hiding sender/receiver identifiers and direct ownership of data packets. By using these techniques Safebook communication protocols achieve anonymity, unobservability, unlinkability, and untraceability; and *(c) Data Access Control:* a set of data encryption and key management techniques/protocols to exert control over who can access and see specific user content. Safebook manages friends in the form of batches (a local arrangement scheme) such as Family, Co-workers, Team, and Professional, etc. to limit and restrict access to the user's data as desired. Safebook provides fine grained access control settings to enable the user to set the visibility of any content with respect to the user's own batches.
Social Network Services	On top of these two layers, Safebook provides services which support the operations of OSNs, such as friends lookup, contact traversal, friendship request, pictures sharing, comments, etc.

In general this thesis addresses the privacy preserving mechanism of Safebook in the context of secure data management in P2P networks and the use of a data access control solution to preserve user data privacy and limit its visibility.

Chapter 2 provides general background information about OSNs and detailed information about Safebook. The chapter also reviews related work. Chapter 3 describes the access control requirements for an OSN, while chapter 4 describes how these requirements can be realized with Safebook. Chapter 5 will describe how the proposed solution is implemented and what the performance measurements are. Finally, Chapter 6 will summarize the conclusions of this thesis project, suggest some future work, and reflect upon some of the economic, social, and ethical issues considered during the course of this thesis project.

2 Background

This chapter gives a brief overview of concepts and terminologies which will form the basis of the work discussed in the later chapters. Since, an access control mechanism for a privacy preserving OSN is being investigated; the chapter begins with the fundamentals of an OSN, its architecture, and a functional overview; followed by a summary of the security objectives for an OSN. The cryptographic concepts that will be used in this thesis project will be discussed. Later sections give a brief overview of Safebook. The architecture of Safebook will be presented along with how the various security objectives are address by this proposed solution. Following this the need for access control in an OSN and some basic access control models are discussed. The chapter concludes with a summary of some related prior work.

2.1 Online Social Networks

Online social networking is an innovative internet service. Commercial versions of such services aim to provide more and more useful applications to satisfy their customers' needs. In the past few years Online Social Networks (OSNs) have become the most popular internet service (measured in numbers of users). The term social network describes the connections between entities in the form of a graph based on the social relationships between the entities. In a real life scenario, a person is related socially with different persons in a community around him or her, forming a graph of shared acquaintances and a network connecting persons and societies.

Figure 2-1 shows the social circle of Alice, where Alice knows a number of persons personally, her relationship to these persons are represented by directly connected edges, but in real life, we know persons indirectly via our acquaintances and such relationships are represented by indirect connections. As a whole this social graph represents an interaction pattern of Alice based on her personal interests and shared relationships. These social patterns can be very useful to disseminate information based on people's interests with shared social circles. For example, if Alice wants to share information with her friends regarding a low cost football equipment store, she will most likely share this information with her friends with whom she plays football or those who she knows have some interest in football. The information will most likely spread from her friends to their friends and so on. Thus, information dissemination in real life can be represented as information passing through social networks. It has become very common to connect people via social circles. Such an interaction model for the Internet community could incorporate the users' real life relationships and form a connected social circle. The systems that have been implemented to facilitate the development and maintenance of such social circles are called OSNs.

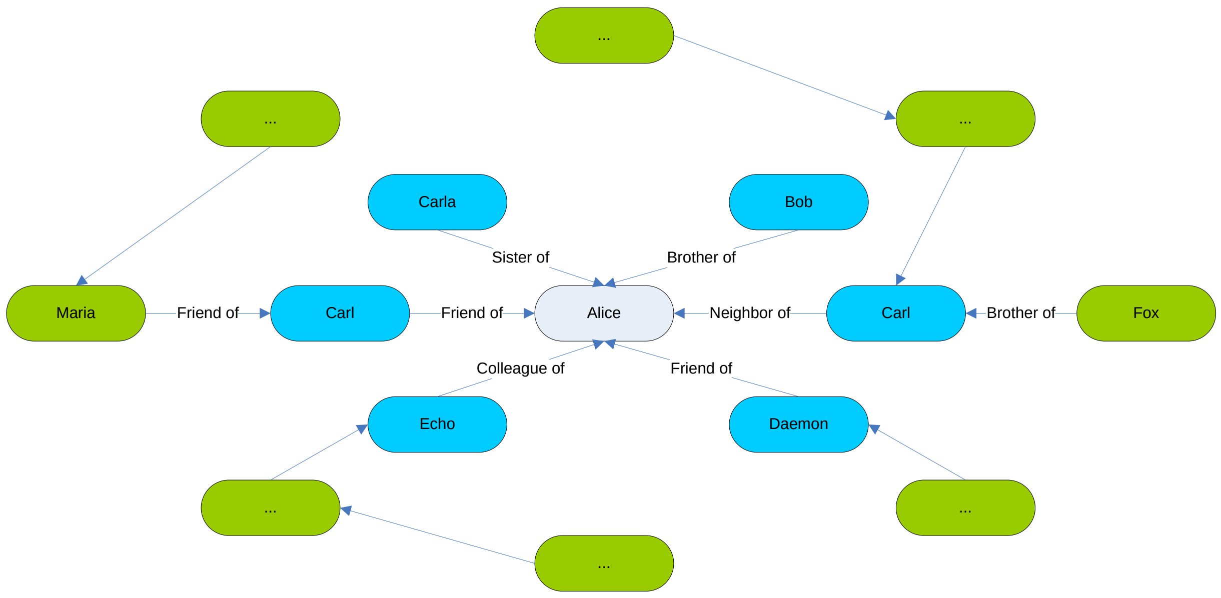


Figure 2-1: Social Circle of Alice

2.1.1 OSN Components

Some of the basic concepts for modelling an OSN are social networking services (SNS), social network provider (SNP), and social network application (SNA). Each of these is described further below.

Social Networking Services Social networking services (SNS) are an online accessible suite of services or a platform, which supports building social networks for online communities. These services facilitate users building their online profile, searching for friends and interest groups, adding and managing their relationships with other users based on relationships and interests, adding and sharing text and/or multimedia contents with members of their social circles, etc. These services are the core of any OSN and are subject to intense innovation.

Social Network Providers A social network provider (SNP) is an entity whom owns and provides a SNS to support social networking. The SNP provides users with data storage and management services, along with other services to realize the OSN. The SNP may or may not own all the infrastructure and facilities that are used to support their OSN.

Social Network Application A social network application (SNA) is usually a third party application, programmed using application programming interfaces (APIs) provided by the SNP to enhance the user's experience. Multiple OSNs can be coupled by third parties; hence a given user can use multiple SNPs in an integrated fashion. Such SNAs can support sharing interests and information across multiple OSNs.

2.1.2 User Interaction pattern with an OSN

The user's interaction with the entities involved in a OSN are presented in *[1]*, along with some of the possible risks of private data exposures. The interaction pattern shown in Figure 2-2 summarizes how the different entities in the OSN are interrelated. The service providers (SNPs) usually own the SNS, but the data servers can be provided by third parties. Usually the SNPs are sponsored by a number of stakeholders to advertise their products via the SNS to a large online community. These sponsors may wish access to users' private data in order to design new strategies to advertise products based on users' interests and preferences. The user interacts with the OSN via the SNS or SNAs provided by third parties.

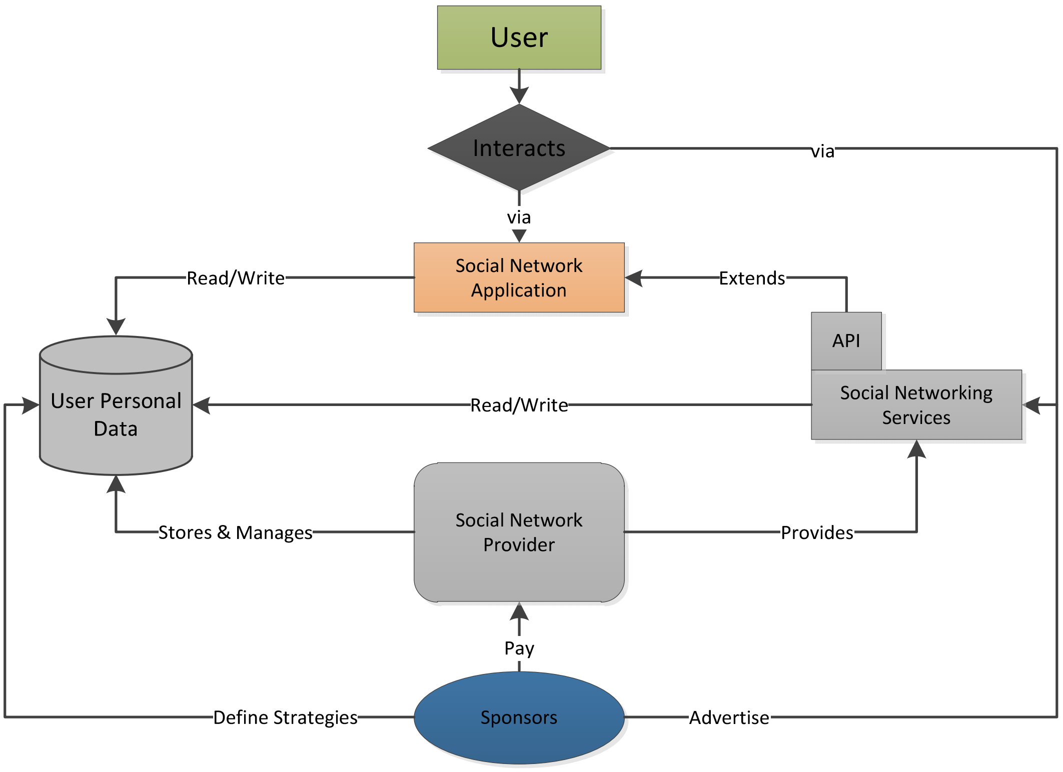


Figure 2-2: Interaction pattern between User and different entities of OSN (This figure is derived from [1].)

2.1.3 Functional Overview

In general, the functionality of any OSN can be classified as:

Networking Functions	The (social) networking functions are responsible for establishing graphs representing social relationships in the OSN. These functions build and maintain the social network graph and facilitate users looking up new friends, adding and deleting relationships, traversing the network graph, etc.
Data Functions	Data functions manage user-contents and ensure efficient content retrieval, as well as communicating content between users. These functions facilitate users inserting new content into the social network, retrieving their own contents and that of their friends, etc.
Access Control Functions	Access control functions enable a user to define the privacy of their contents in order to prevent unauthorized access to the user's private and sensitive information.

2.1.4 Data modelling

Usually the data in the OSN are modelled as:

Connections	These data artifacts store information about the user's relationships in the social network. The connections in the social graph show who is related to whom in the social graph.
Personal Data	The personal data can be in form of personal identifiable information (PII) (such as date of birth, father's name, etc.), posts and comments, multimedia contents, etc.

2.1.5 Security Objectives in an OSN

In social networks, the flow of information from one user to another involves many parameters; unlike traditional conversational communications, where all of the communicating parties are directly aware of each other. In traditional conversational communications the communications infrastructure is rather static, thus we can protect the communications using encryption to provide secure end-to-end protection to the users' communications. In contrast in social networks, sharing of information is not always via direct communication between two users; rather content can be and generally is shared by a number of acquaintances, based on their relationships and the trust that a user has with other users. However, in a large social network the credibility of these acquaintances is uncertain. Additionally, intermediate nodes cannot be trusted to deliver sensitive information *only* to those parties that the source of this information might want to have access to this information. Even if the information itself is encrypted and some end-to-end protection scheme is utilized, the intermediate nodes can learn the user's social network graph by observing the user’s communication patterns. Someone who is able to observe this traffic can deduce the user’s relationships with other nodes in the network based upon statistical analysis of the traffic flow(s). The traditional information security properties of confidentiality, integrity, and availability do not fully address the challenging environment of an OSN *[9]*. A more comprehensive notion of user protection in an OSN is user privacy *[1]*.

Ensuring user privacy is very important when it comes to dealing with real life data, as misuse of this data can lead to damages to the user’s personal integrity. A breach of the user’s privacy or destroying the integrity of the user’s data can result in social and economic damages and may lead to embarrassment of the user or even the loss of the user’s reputation. Along with preserving the privacy and integrity of user data, maintaining the availability of the user’s contents is an important security objective, if the OSN is to be considered reliable and dependable. Missing data or unavailability of data may damage the appeal of an OSN and could impair the service(s) it provides.

The following paragraphs will briefly explain the privacy, integrity, and availability security objectives for any OSN.

2.1.5.1 Privacy

Privacy in context of an OSN is the ability of an individual to limit the visibility of their contents to a desired and selected audience. In other words, privacy is the protection of the user’s information, while giving this user the ability to selectively disclose their information to members of their social circle(s). This is considered by many to be a very desirable property in the real life of a person, as people do not want to share everything with everyone. Personal information sharing among people is always subject to relationships and trust. Interference with this information sharing by others is never welcome. Therefore, in an OSN which deals with the user’s personal and sensitive information we need to protect the user’s privacy.

2.1.5.1.1 User Contents Privacy

In order to protect the privacy of the user’s contents, the OSN should provide the ability to define who is authorized to view what contents. This objective can be achieved by defining an appropriate *Data Usage Control*, which defines an appropriate access control over data artifacts for authorized users, along with its control over this data’s later usage *[1]*. This defines two important problems to be addressed, first an appropriate access control scheme, which allows the user to define fine grained access policies. The user profile usually contains data items, such as personal information, picture albums, blogs, interests, etc. The access mechanism should enable the user to control each data item or even control data within a given data item in order to protect the user’s privacy. Secondly, in addition to access control for authorized users, the user should be able to control usage of their content even after the content is accessed. This means that a user should be able to control how their content is processed in order to *preserve* the user’s ownership of this content and to prevent misuse of this content.

2.1.5.1.2 Communication Privacy

In OSNs the protection of communication among users is very important to preserve the user's privacy. Negligence in this regard may lead to very serious privacy leaks. In order to safeguard the user's privacy, the communications protocols should preserve the following properties *[1]*:

Anonymity	A requestor can access resources owned by another resource owner without revealing the requestor's identity.
Unobservability	During communication between two users, no third party should be able to gather any information concerning the communicating users and the contents being shared by these two users.
Unlinkability	For any two messages, no third party should be able to conclude whether these two messages were intended for same receiver or sent by the same sender.
Untracebility	No third party can build a useful history of actions of any user.

2.1.5.2 Integrity

The objective of preserving integrity in OSNs is to prevent a user's data and personal information from being tampered with or modified by unauthorized users. This also requires protection of the user's real identity in the OSN. We assume that user profiles are associated with a real person, thus preserving the integrity of a user's identity becomes an important security objective for a privacy preserving OSN. In current OSNs, creation of account is very easy and none of the major service providers ensures that the profile is actually associated with the corresponding person in the real world. Such a lack of integrity concerning the user's identity may lead to impersonation attacks, thus causing the OSN to be considered untrustworthy.

2.1.5.3 Availability

In OSNs availability concerns assuring the continuing operation of social networking services *despite* attacks or faults. If the OSN is unavailable, this may lead to a temporary or permanent loss of data. User access to the OSN's resources can be denied by denial-of-service attacks. If either the complete user profile or part of it is disrupted this could be very damaging in the case of professional and career oriented OSNs. Unavailability of user data and profiles makes the social networking less attractive and undesirable to users. Thus for any OSN platform, the availability of user generated content and profiles is a basic necessity that needs to be guaranteed (at least to some suitable level).

2.2 Related Work

This section describes some of the related work concerning data access control. Section 2.2.1 is a walkthrough of earlier access control models with their advantages and shortcomings. Section 2.2.2 will cover some proposed solutions which address the social networking paradigm together with a comparative study of their shortcomings.

2.2.1 Earlier Access Control Models

The problem of access rights was initially addressed in the form of *Access Control Lists* (ACLs) attached to resources. In this model, resources in the system are associated with a list of mappings between users and actions they can perform on the resource. Usually these actions are read, write, execute, delete, modify, etc. This approach has been used by many operating systems to define access rights for individual users or group rights to access/execute resources within a file system. Similar ACL models have been implemented in different network entities, such as servers, routers, firewalls, etc.

Another resource centric approach was introduced by Lampson in the form of an *Access Control Matrix (ACM) [12]*. In this model, the system resources/entities that need to be protected are modelled as a set of objects. These objects (such as user processes, memory segments, critical system resources, files, directories, etc.) need to be protected with specific access rights. The entities which access these objects (such as processes, applications, users, etc.) are defined as a set of subjects. These sets of objects and subjects are arranged in the form of a matrix where subjects are arranged along one axis and objects on other. This matrix relates each subject with access rights (taken from the set of rights) which this subject can perform on a specific object.

Due to the simplicity of the ACL and ACM there are number of limitations. These models consider each user as a distinct entity with a set of permissions for each resource. This implies each resource is separately associated with an ACL/ACM, and this can be very difficult to manage when there are a large number of users and resources. These models do not support advanced access permissions based on user's properties and relationships, moreover they do not define a common policy for a group of users.

In order to overcome the shortcomings of ACL/ACM as resource focused access control, Sandhu et al. *[12]* proposed a *Role-Based Access Control* (RBAC) model. In this model access is granted to a resource based on the relationship between the requester and the resource owner/organization. This relationship is defined as a role or function of the requestor in the organization, e.g. roles within an organization can be defined as 'Employee', 'Accountant', 'IT Staff', 'Software Developer', etc. Every role is associated with a set of permissions, specifying a certain level of access to the different resources and services of an organization. Moreover, users can have multiple roles and hence can hold a union of several permission sets. In this way a hierarchy of roles can be defined to allow flexible access control over an organization's resources. For example, if an accountant is an employee, then they can access the notice board and other general services for employees and they can access the company's spread sheets and specialized services based upon their accountant role.

Although RBAC can provide quite a flexible hierarchical access policy it lacks control over individual users within a certain role or group. This can be achieved by dividing roles into sub roles, but this approach lacks fine granularity. Additionally, assigning sub roles may not be feasible in all scenarios. The model organizes users as virtual groups in a weak binding through sharing of the same roles; this weak binding adds additional management complexity when specific groups are needed for specific tasks. Another approach to access control, with an effort to define more granular and context based access control is called Task Based Access Control (TBAC). Roshan K. Thomas *[12]* proposed defining dynamic access permission over tasks (a task can be a project, business operations, etc.) and its current context, instead of defining access policies for individual or group users. This model defines a task as a number of steps. Each step is coupled with a protection state and this state contains a set of permissions for the current context of this step *[12]*. In this way, as the task progresses, access permissions are dynamically changed according to the context of the current step. This model supports features such as typed-based, usage-based, and instance access to resources; furthermore users are authorized with strict runtime usage, expiry, and validity periods for access rights to specific resources *[12]*. Unlike RBAC, where permissions are statically associated with resources, TBAC is a live model, hence permissions are dynamically changed as the task proceeds.

TBAC is useful for task oriented problems, where concrete tasks can be defined with small steps and associated with access permissions to resources. Studies say it is not always possible to define access control for users based solely on contextual evidence or work flow. Roshen K. Thomas et al. proposed another access control model, Team Based Access Control (TMAC), which defines a set of permissions for a team/group. This model is different from RBAC, which loosely groups a number of users sharing the same role n to defined permissions for this role instead of a user group; whereas, TMAC defines access permissions directly for a team. Furthermore, using the RBAC approach, TMAC provides fine grained access permissions for individuals/sub-teams within a team, by assigning different roles to them and restricting/granting access to different resources. To be more formal, the formation of teams and associating a team with specific access permissions are based on two important factors: (1) **User Context** identifies users based on their current role in a team and (2) **Object Context** defines a set of specific objects/resources for a specific user context.

All the solutions mentioned above are considered identity based access models, where the identity of the users is known in advance, either directly or through roles, groups, etc. This is suitable for closed environments or for static distributed environments, where all parties are known in advance, but in a real open environment such as internet services the identity of the user/requestor cannot always be known. Moreover, these solutions lack expressiveness with regard to defining very fine grained access permissions. This fine granularity and freedom of expressiveness in an open environment is addressed though Attribute Based Access Control (ABAC) *[13–17]*. In ABAC access control decisions are based upon the association between a set of attributes/characteristics of the requestors/users and the resource/object owners. The user attributes are compared with the access rules and decisions are made, without actually knowing the identity of the requestor. These attributes can be very diverse, from simple attributes such as employee hiring date, projects, location, date of birth etc. to very specific credential certificates, signed attributes to access enterprise resources, etc. These attributes are usually a discrete set of values, hence very simple and easy to define access rules can be defined as decision criteria over these user attributes. For example, if the user wants to share an item of multimedia content only with people from the class of 2009 at KTH, then a simple attribute such as 'University Studies From' is sufficient. Furthermore, a simple rule can be formulated to deny a request to a resource based on the requestor's age or date of birth. Using the characteristics of users and defining a discrete set of attributes enables both fine grained and highly expressive access control *without* explicitly identifying the requestor. The advantage of not needing to know the user in advance, as required in the other access control models where users are known and roles are formally maintained by the organization, can also be seen as a disadvantage. Sensitive critical information and resources cannot be accessed simply based on attributes of the requestor, as doing so would require complicated rules based on multiple attributes in order to avoid any chance of improper access. Moreover in large organizations, such an access policy can be difficult to maintain and harmonize.

In addition to the above models, there are additional models such as Policy Based Access Control, which is a standardized and harmonized form of ABAC to support large organizations' specific governance objectives. In *[18]* Covington et al. proposed a Context Aware Access Control model where access permissions are changed dynamically with the changing environmental context of a user. This model extends RBAC with the concept of environment roles in order to provide security features for context aware applications.

Unfortunately, none of the above models fully satisfy the access control objectives for an OSN. However, these models can form a basis for a new dynamic and adaptive access control model for OSN. In social networks, it is a very common practice to organize friends into groups and then associate different types of data with particular groups. The user might want to restrict some data to be only accessible within a group with limited members or to share a particular picture/post with some friend outside this group. Some of these requirements can be addressed by team based access control, with teams as groups and then roles can be used to further define rules for individuals/sub-teams. However, due to the limitation of a role based approach, it is not always possible to define roles with sufficiently fine grained access control.

In addition to handling groups, an OSN also needs access control based on the characteristics of users. For example, a user might want to invites all his married friends in the 'Work Place' group accompanied by their families to attend a picnic. Unfortunately, it is not possible to form subgroups based upon on such characteristics using the classic approaches of group based access control models like RBAC, TMAC, or TBAC. This problem can be addressed by attribute based access control by defining attributes for each user which can be used to define access control. RBAC can be useful in defining relationships in an OSN based upon different roles, and then define access rules for each type of relationship. Similarly, context aware access control can be used to enable dynamic environment dependent access decisions. However, as of yet, no single address control model fully satisfies the requirements and objectives for access control in OSNs. Hence there is a need to design a flexible, expressive, and adaptive access control model to fit the constraints of dynamic OSN data and its users' access control needs. The next subsection will describe a proposal for such an access control model.

2.2.2 Proposed Access Control Models for OSNs

Social networking has proven to be a very challenging environment in which to implement access control which provides a fine grained access policy for users to control access to their data. Current social networks, such as Facebook, provide some privacy settings by which the user can limit access to their data to a limited audience. But such access control settings, no matter how good they may be, do not guarantee that the user's data will not be abused by the service provider. The user still has to trust the third parties that provide data storage, which is undesirable as these third parties might not preserve the user's privacy. To address this problem of third party access control enforcement, Filipe Beato et al. *[19]* proposed a solution with an objective of avoiding the need to trust the SNS provider. In this proposal access control is enforced at the client side, facilitating the user's control of their own data. To implement this access control access rights for a specific audience occurs through encryption. As the user profile contains information about the user's connections (such as represented by their circle of friends) and data (user contents such as posts, pictures, etc.), this solution classifies the user's profile into *Connection* and *Content* classes. Instance of these classes are arranged in a tree like structure to enable rule based access control over the user's private data. The *Connection* classes classify the social network user's connections into different groups and sub groups. Similarly, the *Content* classes classify the user's data into specific groups and subgroups, such as data related to family, work, etc. This characterization into content and connection classes forms a hierarchy of classes with nodes as groups and leaves as users or data items. Finally, access control is realized by relating the connection classes to the content classes. The access rights propagate from top to bottom in the hierarchy of classes, e.g. if the Friends connection class is mapped to the Hobbies content class, then all the subclasses of the Friends connection class will inherit access to the Hobbies content class. The data belonging to a content class is encrypted with a symmetric key associated with that class. This key is shared with the users of the associated connection class. By using a Public key Infrastructure (PKI) encryption scheme data confidentiality is achieved despite the data being stored on the SNS provider's data servers. Whenever a new connection (user) is added to the user's connection classes, there will be an exchange of public keys via an online key server, email, or some other means. These public keys will be stored locally and used to form the user's circle of trusts. Similarly, when a content class is added, a symmetric key is shared with the associated users in the content class to provide the desired access rights to a selected audience. In this scheme the service provider does not know the user's connections or what types of relationship these users share. Moreover, by storing only encrypted user contents the service provider is not aware of the actual contents.

The solution uses an encryption wrapper to provide access control for a specific audience and can operate as an overlay on the current SNS providers' OSNs. This simple approach of dividing the profile into connection and content classes provides an easy to manage hierarchal access control mechanism for a user's groups. The solution addresses access control for a selected audience. However, defining access rules for individuals or members *within* a group is not supported. Additionally, users are not directly related with resources; rather the connection classes they belong to are related to content classes. The hierarchal propagation of access rights may lead to undesired accesses if not properly managed and monitored. Furthermore, the model does not support access control based on user and data attributes. Unfortunately, this rule based access control model does not fully satisfy the needs for an OSN.

Barbara Carminiti et al. *[20]* suggest another rule-based access control model in the context of a web based social network. In this solution the social network is modelled as a directed graph, where each edge represents the relationship type and a level of trust this direct relationship holds. A user's trust is subject to the depth of relationships with other users in the social network graph. For example, in the case of a friend of a friend of a friend relationship the depth is 3. The objective of their solution is to define an access policy for a web based social network, where a user can define access rights over their resources based on the Relationship Type, maximum Depth of Relationship, and the minimum Trust Level a requester should hold. The enforcement of access control is realized as follows, *a requester receives a set of access rules from the resource owner in response to a request to access to certain resource. These access rules state what relationship type, maximum relationship depth, and minimum trust level is allowed to access the resource. To gain access, the requester has to provide proof that they* *satisfying the required criteria.* The authenticity of the proof claimed by the requester is assured by using certificates. The correctness and trustfulness of the relationship type in an access rule is achieved by showing a signed relationship certificate. These certificates are exchanged between users stating which Relationship type and Trust level they hold, as part of making new relationships. Moreover, indirect/direct friends A and B can be verified by retrieving the chain of certificates, confirming the relationship path among them in the social network graph and provided this as proof to the resource owner. On other hand, proving the trust level is a complex operation which requires computing all possible connection paths between the requestor and the resource owner. Besides its inefficiency, this strategy has a weak notion of proving a specific trust level, as the requestor might only show paths with high trust levels. The author has addressed this problem with the introduction of a Central Node that is responsible for archiving and managing all the users' relationship certificates and calculating relationship trust levels. In this scenario, the requestor receives a chain of certificates signed by the CN, and then the requester reason or (a module which generates a proof based on the certificates received from the CN) generates a proof and forwards this proof to the resource owner, who then makes an access decision after evaluating the proof. The model proposes two implementations for the access control enforcement. First for Web-Based Social Networks, where users store their data in a central repository accessed via web, the Central Node and Reference Monitor[*] can be implemented by the SNS provider. Second, they have proposed a decentralized solution where each user owns its own data and enforces its own access policy. This architecture is based on two services: (1) the Central Node for certificate management and (2) a set of peripheral nodes representing each node in the system. The peripheral nodes are responsible for storing access rules and performing access control. This service can run on either the user's machine or it can be hosted on some server. Access control is enforced in the same way as discussed above, with an addition of peripheral nodes as communication and policy enforcement agents between two users.

The proposed model presents a flexible means to express access conditions in terms of relationship types, depth, and trust level. These conditions can be used in the decision to grant access to indirect users. This model gives the user flexibility when deciding how to propagate their data via their social graph, and limits the spread of this information by setting the appropriate depth for each given relationship type. It may seem inappropriate to grant access to indirect users, but in social networks this is a desired property. For example, it enables you to receive a comment on a tagged picture by friend of a friend. However, with this approach it is difficult to restrict access to a specific audience or group. The access rights are too general and it is difficult for the user to keep track of the users who can access a resource. In terms of preserving user privacy, the relationship types and network graph needs to be secured. Unfortunately, in this solution the Central node holds all the relationship certificates and thus knows the complete social graph of user relationships, which is not a desired property for a privacy preserving OSN. In the decentralized solution, relying on the central node as a trusted entity is also undesirable. Moreover, these authors did not mentioned any protection scheme for the user data hosted on third party servers and the solution proposed for web-based social network again puts all trust in the service provider, which should be avoided to prevent potential misuse of this information by the service providers.

[*] A trusted entity that enforces access rights based on user access policy.

In the solution presented in *[20]*, private relationships are not protected, but this problem is addressed by Barbara Carminiti et al. in *[21]* by an extension of same idea as presented in *[20]*, while protecting the relationship certificates by encrypting them with shared symmetric keys. This is achieved by considering certificates as just another type of data object with defined distribution rules and conditions. The certificate is encrypted with a certificate key and stored at the central node; therefore, the central node is not aware of relationships encoded in the certificates. A certificate key distribution protocol ensures that users receive a certificate key when it satisfies the distribution rule for that certificate. This protocol is activated whenever a user makes a new relationship or receives a new certificate key from its social network graph neighbourhood. Each node determines the need for further distribution of a certificate key based on its distribution rules and conditions.

The model proposed in *[21]* extended the access control model discussed in *[20]* and protects the relationship certificates. However, this protection is achieved at the cost of complex key management for certificate keys. Moreover, the central node is required to always be reachable. This makes the solution less scalable and less fault tolerant. In social networks relationships often changes, so the proposed revocation scheme needs to be properly managed to avoid unauthorized distribution of certificate keys. Once a relationship is revoked the central node is notified of the revocation and it removes the certificate along with maintaining a certificate revocation list (CRL). This means that the central node still acts as a trusted third party (TTP). Furthermore, the nodes need to verify the keys with the central node before accepting or distributing the keys, this makes the system very dependent on the central node thus it is even less fault tolerant and less scalable due to these frequent and repeated verifications.

These problems in the extension proposed in *[21]* for the protection of relationship certificates has been addressed by Josep Domingo-Ferrer *[22]* in another model based on public key cryptography. This model does not dependent on any central node or TTPs, thus making it an attractive solution. The idea for protecting social relationships is based on relaxation of the trust calculation condition proposed in *[20]*. Domingo-Ferrer suggests that considering all paths from A to B when calculating trust level is simply over protection and might lead to denial of a request despite a very high trust link existing, as the request could be rejected due the existence of another path with a low trust level. With this relaxation no central node is needed to maintain all the relationship certificates or to calculate trust levels. When a requestor sends a request for content to the resource owner, it receives in response a list of access rules. If a direct relationship exists, then it sends the relationship certificate encrypted with the owner's public key. After verification of the relationship certificate the resource owner decides how much the owner trusts the requester when it makes its decision to grant access to the object. For indirect paths, the owner receives a chain of relationship certificates signed with their private keys by each of the corresponding nodes in the path. Thus the owner can calculate the trust from the received chain of certificates and decide upon access.

Domingo-Ferrer's public-key protocol exposes some privacy of the user relationships. Each node has to disclose its relationship trust in the upstream, so that a resource owner can calculate the level of trust. Although the requestor is not aware of the certificate chain received by the resource owner each of the intermediate nodes is aware of the trust levels prior to it in the upstream. Moreover, with the advertisement of access rules for a resource, the owner relationships are exposed. The author proposed to fill the access rule with bogus entries to hide real relationships, but still these relationships can be exposed with an engineered attack.

The decentralization of a social network introduces a lot of new problems that need to be addressed. In the ideal situation data can stored in the user's local storage, to ensure the privacy of the data and to enforce the user's desired access control over it. However, this does not ensure availability of this user's data at all times. In P2P systems, this problem is addressed by replicating the same data over multiple peers to ensure high availability. Unfortunately, this replicated data is now a separate resource; hence the owner of the data loses their control over this new instance of their data. So in order to ensure data authenticity and to enforce an access control scheme on replicated data, Esther Palomar et al. *[23]* proposed a certificate based access control scheme for P2P networks without depending on any central TTP to sign certificates. In this model, each user categorizes its contents with several security labels. Other users are allowed to access data if they have at least the same security level as the content. These security clearance certificates are issued by the content owner. The authenticity of the replicated data is ensured by associating each item of content with a certificate and this certificate is signed by a list of signers. The signing of a content certificate issued by the owner is an iterative process, where the list of signers who have signed the certificate iteratively represent a verification of authenticity and integrity of a message along with the signature of the originator/resource owner. The certificate includes a puzzle, solving this puzzle produces the decryption key for the content. The difficulty of the puzzle depends on the security clearance certificate the requestor holds. So the requestor after selecting one peer from a list of peers holding an item of replicated content verifies the contents of a certificate by iteratively asking the list of signers. After verification of the certificate and the associated content, access is granted to the data by solving the puzzle in the certificate. The difficulty of the puzzle can be reduced by showing an appropriate security clearance certificate issued by the owner.

This certificate based model presents an approach which ensures authenticity and integrity of user data in a replicated environment. In distributed social networks replicating user data ensures high availability. This approach can be useful to ensure authenticity and integrity of the replicated data. Moreover, the proposed access control mechanism can support role-based access control, with security certificates as relationship/role certificates. In this approach the user does not have to distribute symmetric encryption keys to other users, instead the requestor after providing an appropriate certificate can gain access to an easier puzzle to get the key to decrypt the data. However, contents are replicated on other peers who have access to that content, and encrypting this data with new keys with an easier puzzle is responsibility of the peer rather than the resource owner. Such an approach involves many cipher operations, thus decreasing the performance of the replicating node along with potentially violating the privacy requirements of the resource owner. Furthermore, the iterative verification of the content certificate requires the availability of all peers who signed the certificate, which has a very low likelihood in a P2P network. Finally, the signing of certificates by a list of signers indicates some sort of relationship between the originator and signers, as they are not selected at random, but based upon trust levels, such information from the user's friendship graph needs to be protected. As a result, this approach is more suitable for a distributed file sharing application where the privacy requirements are relaxed than for privacy preserving social networks.

In social networks, users often define access policy based on the social relationships they have in their social network graph. Exploiting this approach, Kiran K. Gollu et al. *[24]* proposed an access control scheme for personal content in a social network which is focused on social relationships. The model uses the concept of Social Attestations, which certify a particular relationship between two users and is signed by a TTP. These attestation do not include access permissions, rather the attestation has four fields: an issuer, a receiver, a social relationship between the issuer and the recipient, and the relationship key. Access control is enforced by attaching ACLs containing the owner's public key, the public keys of users who can access the data, and a social relationship. To access the data, a requestor must have his/her public key in the ACL or a social attestation issued by the resource owner certifying the required relationship mentioned in the ACL. The authors of this model suggest exchanging public keys and social attestations through out-of-band mechanisms.

Using social attestation has many applications, Gollu et al. showed this in: (1) **Calendar**: social attestation can enforce fine-grained access control for different views of the calendar for different users; (2) **Social Distributed File Sharing**: social content such as pictures and posts can be distributed on top of P2P file sharing protocols such as Bittorrent, where the social attestation provide an access control scheme supporting role-based and object centric access control; (3) **Social Firewall**: access requests can be filtered based on the attestation provided by the requestor. Beside its different applications, the idea of attaching ACLs to a resource object poses a security risk especially in distributed systems. The approach is reasonable if the owner is hosting his own data, but if data is replicated on several peers, such an access control scheme is unreliable. Sharing, protection, and enforcement of user access policy in an environment where contents are replicated become a big challenge, which Gollu et al. did not mention. Moreover, dependence on a third party for signing the social attestations gives the trusted node complete knowledge of the social graph, which is undesired in privacy preserving OSNs.

In *[25]* Jonathan Anderson, et al. proposed a solution for a privacy-enabling social networks over untrusted networks. The core idea is to build a social network out of smart clients and a centralized data server where encrypted data blocks are stored. Anderson, et al. proposes an API on top of which a client application can be written and customized. The system is a simple client-server architecture, where the server ensures availability of data while clients are responsible for their data's confidentiality and integrity. The client API contains four software layers. Of these the **Application layer** provides basic atomic social network operations, which can be combined to realize a design according to a user's requirements. Moreover the applications will run in a secure sandbox limiting their access to the user's resources. The **Data Structure Layer** manages the user's profile & contents as a collection of discrete blocks. Each block is associated with a link. Each link contains information about the next connecting blocks, thus the data is arranged in the form of a tree. These links are used to enforce an access control mechanism over data blocks by hiding and showing different connections from the same link for different authorized users, thus a multi-view tree is formed based on the authorized user. This arrangement also hides the information regarding user data stored at the server from both users and server owners. The **Cryptography layer** ensures identity verification through out-of-band signalling. The links associated with data blocks are hidden through stream cipher encryption, in such a way that when user A decrypts a link it gets a connection abc, but when the same link is decrypted by B with its shared key, it is directed to some other xyz connection. This is achieved my concatenating different key's ciphers into a single link block. Moreover, a key management, mechanism for handling joint content and support for secure instant messaging is also supported by this layer. Finally, the **Network layer** provides an interface to communicate with the server through GET(ID) and POST(ID,x,C) methods, where ID is unique identifier, x is an encrypted data block, and C is the capability of the block.

This is a very unique approach of representing a social network. The server node is only a data repository and stores encrypted data with unique identifiers not associated with users. All access control is enforced at the client side. Unfortunately, Anderson, et al. did not mention any appropriate access control scheme and left this open for implementation by a third party. The information in the links are of vital importance, as if a single link is corrupted or lost, then the entire tree from that point is lost unless a backup of this link is maintained. Moreover, implementation of this design will result in very complex key and data management.

A distributed access control mechanism for P2P collaboration has been proposed by Christoph Strum et al. *[26]*. The main idea is to establish a global access control mechanism for a P2P system, similar to existing relational database management systems. The distributed global access control is established by mapping local access control components of participating peers and exporting parts of this access control information. The model organizes the network into different groups, where each group represents some company or organization. Each group is governed by some administrator peers. These administrator peers are preferably hosted on a dedicated server by the IT administrator of the company. The IT administrator defines each user's access rights over the organization's resources. The administrator peers are responsible for collecting local access control information from each peer in the group as a XACML document and combining them to form an Export Policy. Accesses to remote objects are achieved by mapping policies between the different groups. For example, if A is directly related to B, then a mapping policy will include rules to access remote objects, mutually agreed upon by the participating peers in the form of XACML contracts, stored at each peer, and these contracts can be revoked by the resource owner at any time. Access to indirect remote objects can be achieved by involving a grantor peer who is directly related to both peers. Mapping policies can be implemented as a distributed access control directory or by separate mapping policies stored at each peer and the resource owner can on request search for a required grantor through gossiping with his direct peers.

The solution provides good control and preserves ownership of its data, along with propagation of access rights to indirect users with the data owner consent and approval. This approach can be useful in social networks to grant access to a specific resource in relationships such as a friend of friend, where the intermediate node can be a grantor. Moreover, Strum, et al. assumed the administrator peers are more likely to remain available as they are server nodes with reliable network connections and running on reliable machines. In a P2P environment this assumption reduces the system's scalability and constrains applications. Furthermore, the model does not support replication of data in order to maintain full authority over the data by its owner. This is a very idealistic assumption, as in real networks peers needs to replicate data in order to increase its availability. This model is suitable for static P2P networks, rather than a dynamic network modelled by social networks.

Imen Ben Dhai et al. *[27, 28]* presented an access control model for social networks where access to resources is controlled by considering user properties and the various indirect/complex connections between users. Unlike other approaches where an access decision is based on proving your relationships or showing some sort of authorization to access the data item, in this approach access is gained by finding a connection path from a requestor to resource owner via the social graph based on the reachability criteria defined by the owner. The resource owner can define restrictions on its data, such as, only children of friends and friends of friends can access my adventure stories. As a result access is granted only to those requestors who have a reachability path with the specified attributes and relationships in the social graph on a path to the resource owner.

This solution offers a good approach to enable fine-grained access permissions based on reachability in the social graph. However, the complexity of finding the appropriate reachability path is too high. Moreover, the access control mechanism needs complete or partial knowledge of the social graph, in order to validate a request. While knowledge of a user’s relationships is sensitive information and needs to be protected, the proposed solution by Dhai et al. is based on completely exposing these relationships and attributes to some Reference Monitor, which can be a central entity or a distributed service, in order to have a compete view of social graph as needed to validate requests.

2.3 Safebook: A privacy preserving OSN leveraging real-Life trust

This section gives a brief review of an OSN framework called Safebook *[3]*, which aims to protect the user's privacy. Safebook's access control mechanism will be introduced. This access control mechanism is designed to enhance the user's privacy. This section includes a brief introduction to how the security objectives defined in the previous section are addressed by the Safebook framework.

2.3.1 Introduction to Safebook

Because OSNs are very popular they face a number of severe security and privacy threats. The risks of privacy violation are often addressed by service providers, but the existing mechanisms do not allow the user to control the privacy of their information on a fine grained scale. To tackle these problems, Leucio Antonio Cutillo and Refik Molva suggested a new approach to OSNs with the aim of safeguarding the user's privacy with respect to unauthorized access by intruders, other users, or service providers. The platform's design and architecture provide a defence against intruders or malicious users.

In current OSNs, the service provider is responsible for storing user data along with providing SNSs. In this model the service provider might misuse users' personal contents. Safebook ensures user privacy with respect to service providers by utilizing a decentralized architecture, where a number of independent users collaborate to provide privacy aware social networking services, secure data storage, and management services. The fundamental approach of Safebook is to exploit real life trust relationships in social networks in order to build trust and privacy preserving mechanisms for online applications *[3]*. Safebook's design integrates a number of social networking services and data management services that leverage real life trust in order to provide a privacy preserving OSN.

2.3.2 Architecture

The protection of user data and user privacy can be achieved to some extent by enhancing current OSNs by integrating various privacy and security mechanisms. However, even if the service provider is sincere these protection mechanisms pose a risk due to their centralize storage and data management. Attackers, malicious services/users, insufficient security, or unseen/overlooked security breaches all represent potential threats to the users' privacy. Moreover, service providers cannot always be trusted and they might misuse the users' content. Due to poor security a disgruntled employee or the service provider might sell the users' content. Rather than using a centralized server based data management system as most current OSNs use, a P2P architecture was proposed as an alternative to avoid the service provider having a monopoly on access to and storage of the user's content. Unfortunately, P2P systems lack *a priori* trust. Moreover, trust is an important element in a social networking environment. Safebook utilizes a decentralized P2P architecture to protect the user's privacy with respect to service providers and addresses the lack of *a priori* trust by leveraging the real life trust between users. Based on a user's real life trust in another person, services such as data storage, routing, and retrieval of users' profiles can be performed in the OSN by cooperating with users whom one trusts in real life.

Safebook is a three tier architecture [3] as shown in Figure 2-3. This architecture separates the social networking components from the underlying P2P application services and communication components. The objective of this multi-layered architecture is to reduce the coupling between operations at different levels, e.g. decoupling the social networking operations from basic P2P overlay lookup and maintenance operations. This makes the architecture more transparent and makes the implementation of each layer independent of the underlying implementations.

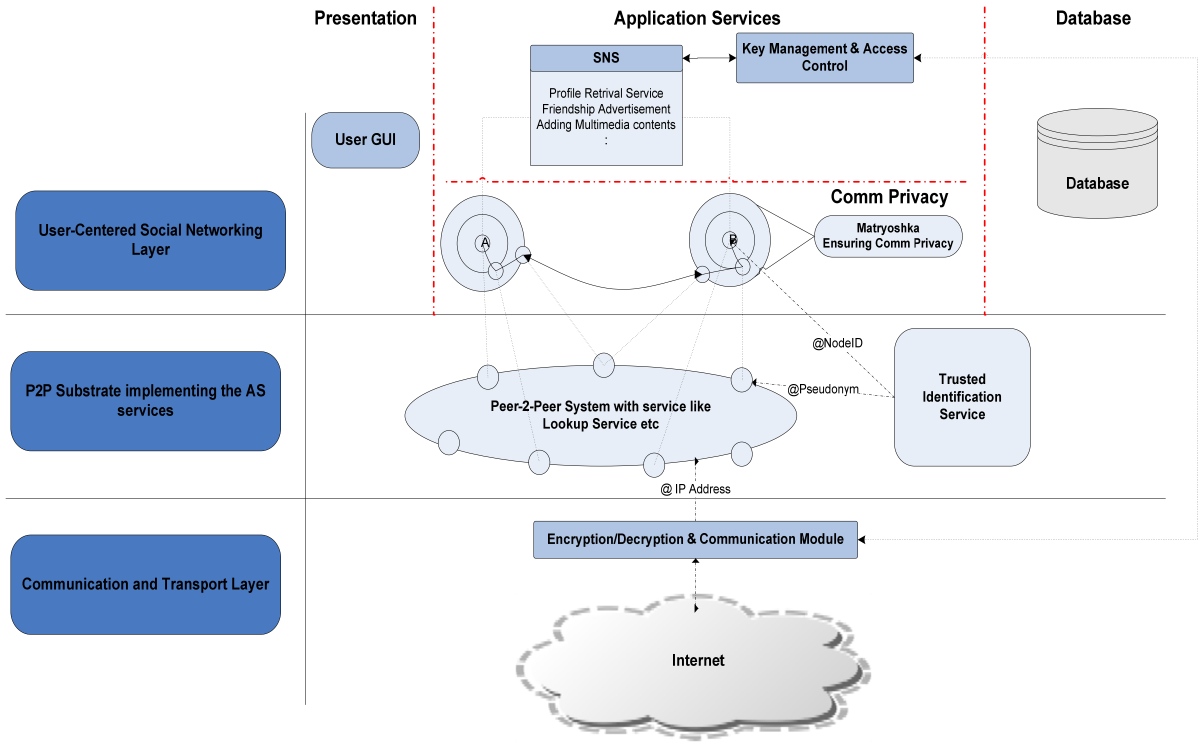

Presentation
Application Services
Database
SNS
Key Management & Access Control
Profile Retrival Service
Friendship Advertisement
Adding Multimedia contents
User GUI
Database
Comm Privacy
Matryoshka Ensuring Comm Privacy
User-Centered Social Networking Layer
A
B
@NodeID
P2P Substrate implementing the AS services
@Pseudonym
Peer-2-Peer System with service like Lookup Service etc
Trusted Identification Service
@ IP Address
Encryption/Decryption & Communication Module
Communication and Transport Layer
Internet

Figure 2-3: Safebook Architectural Overview

Each layer of this architecture addresses a set of privacy/security objectives and functional requirements. These three layers will be described in further detail in the following three subsections.

2.3.3 User-Centered Social Networking Layer

The user-centred social networking layer is the core of the Safebook implementation. This layer provides a set of services/operations to provide secure and privacy aware social networking services. The primary objectives of this layer are to provide a user friendly interactive interface for users, secure and privacy aware distributed services to support social networking, an appropriate key management and encryption scheme for access control of user content, and database services.

The privacy objectives of this layer are:

Decoupling the user layer from the P2P layer	The P2P layer is responsible for lookup services supported by a distributed hash table (DHT). If a node in the user layer triggers a lookup request in the P2P layer, then the intermediate nodes can relate the user with the request to access the data of a remote user. Similarly, by observing user requests, the OSN's network graph can be predicted to some extent. This threat is removed by decoupling the user identifier (User_ID) from the P2P node identifier (Node_ID). The mapping between a User_ID and a Node_ID is only available to trusted nodes (friends). In this way lookups cannot be associated with any user (i.e., to a real person).
Communication Privacy	It is very important in social networks to hide and safeguard the communications between two entities. Encryption alone is not sufficient to hide information about the user's network graph. By monitoring network traffic an attacker can partially or completely predict a user's network graph. This communication privacy is achieved by using anonymous routing techniques, where the SNS traffic, generated at the user layer is injected into the network through an anonymous routing overlay on top of the P2P overlay network.
Access Control and Key Management	Defining access control strategies in an OSN is always a challenging task. Traditional encryption schemes alone are not sufficient to define fine grained access control for user content. In Safebook access to user content is controlled by a number of parameters, such as relationships, attributes, activities, interests, etc. Allowing access to user content only by desired users or a selected subset of users is an important privacy objective.
Availability	A high probability of availability of user contents is ensured by a secure replication scheme.

The user-centered layer is further divided into the following components:

User GUI	The Safebook official release is accompanied by an interactive web based graphical user interface (GUI). This GUI can be implemented or customised by any interested vendor by using Safebook's application services (AS) application programming interface (API). The aim of this component is to provide the user with fine grained privacy settings via an interactive GUI.
Communication Privacy & Replication Overlay	This Safebook component insures the security objectives of anonymity, unobservability, unlinkability, and untracebility. The interaction between two social networking users can reveal lots of information regarding their relationships and trust levels. For example, if two users are communicating very frequently and transferring a large number of packets, then even though these packets are encrypted, the pattern of communication implies a good relationship between the two. Both the privacy of the information and the network graph of the user and their activities need to be

kept private. In order to provide this communication privacy, Safebook uses onion routing (an anonymous routing technique) to mask user requests and responses via a chain of connected and trusted users while hiding the USER_ID of the original message generator.

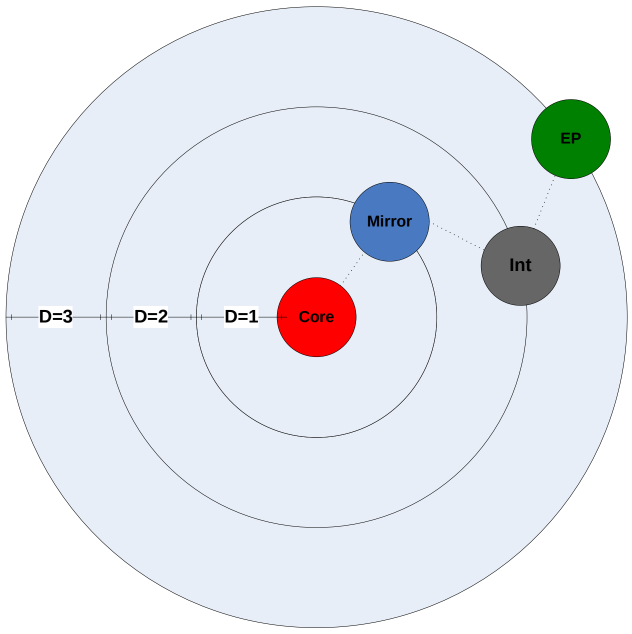


Figure 2-4: Matryoska Overlay with Depth=3

In the Safebook network, when a peer boots, its first task is to create a user overlay with a connected chain of trusted and willing users. Such an overlay is depicted in Figure 2-4. This overlay is called a Matryoska overlay and the chains are referred as Matryoska Chains. The Safebook node triggers a request to create a Matryoska Chain by sending a request to a set of available users in his or her contact/friend list. Users receiving a request for Matryoska creation may accept or deny this request based on their own capabilities and willingness to participate.

A Matryoska path/chain consists of five entities:

1. Depth of Chain (d)
2. Core Node (C)
3. Mirror Node (M)
4. Intermediate Node (Int)
5. Entry Point Node (EP)

2.3.3.1 Matryoska Overlay Creation

Considering a simple scenario, where *A* triggers a request for Matryoska creation, the process involves the following steps. If peer *A* is configured to create Matryoska chains with a depth of three (i.e., *d=3*), then when *A* (the core node) boots, it sends a Matryoska creation request with depth *d= 3-1=2* to peer *B*. If *B* accepts this request and agrees to become a mirror node for *A*, then *B* forwards the request to a set of available peers from *B*'s contact/friend list. We will assume that *B* forwards the request to *C* and *D* with depth *d=2-1=1*. Although *C* rejects the request, *D* accepts to participate. In this way *D* becomes the intermediate node in the chain. As the *d* parameter has not yet reached zero, *D* will select another node to complete the chain. Suppose *D* forwards the request to *E* with depth *d=1-1=0*, if *E* accepts the request then since *d* has reached zero, *E* will becomes an Entry Point (EP) node for the Core *A*. Node *E* will send a PATH COMPLETION ACKNOWLEDGEMENT (ACK) to its predecessor node in the path and the acknowledgement will be successively passed along the path until it reaches the Core Node.

Note: Except for Mirror Nodes, no other node in the path can predict the length of the chain as each intermediate node is only aware of its successor and predecessor in the chain. As a result only the mirror nodes can predict the core node. Furthermore, unless a node knows what the original depth is, it cannot even know that it is a mirror node.

2.3.3.2 Publishing Entry Points

OSNs are usually supported by lookup services to find other users either by their name, email address, birthday, or other attributes. Similarly, Safebook facilitates user lookup by publishing a name vector. This name vector consists of all names by which a user wants to be searchable in a Safebook OSN. This gives the Safebook user, complete control over their visibility in the network. This name vector is published along with an Entry Point List in the underlying DHT in the P2P overlay.

The publishing process involves the following steps. Upon creation of a successful Matryoska path with a defined depth, the EP node sends a completion acknowledgement to its predecessor and it to its predecessor, until this ACK is received by the core node. Once the core node receives this ACK it publishes the EP in the underlying DHT with a hash of each name vector entry as a unique search key. If the hash already exists, this indicates that the search key already exist, e.g. there exists another person with the same surnames, then the current EP along with Personal Identification Information (PPI refers to information which can uniquely identify a user, e.g. name, birthday, place of birth, address, email, phone number, etc.) is appended to the existing entry point list.

2.3.3.3 Data Replication

One of the important activities in Safebook is to replicate the user's data to achieve high availability. The user's data is replicated on its mirror nodes in the Matrysoka overlay. This leverages the user's trust in real life of these friends. Mirror nodes have a direct relationship with the core node and the level of trust is high. Moreover, the data is stored in encrypted form, so that its confidentiality is high. Additionally, the data's integrity is ensured by using digital signatures. The core node synchronizes its data with the mirror node when changes to the user's data are made. The mirror nodes handle all of the data requests on behalf of the core node. In this way if the core node is offline, then as long as at least one of the mirror nodes is online, then the user's content is accessible.

2.3.3.4 Matryoska Overlay Maintenance

The user's data will be accessible as long as the Matryoska chains are intact. If the chain is broken, then the mirror nodes appear to be offline to the requestor. For this reason Matryoska maintenance protocols are needed to keep the chains alive. If any intermediate node drops out of the path, then the preceding node in the chain will try to repair the chain by sending requests to its available friends for Matryoska path creation. Meanwhile, it will also notify its predecessor with an EP clean up request (indicating that the EP is no longer valid). The EP clean up notification should eventually reach the core node and the core node will remove the EP entry from the underlying DHT. Once the chain is repaired, a new EP is published by the core node. If a node fails to repair the chain, it notifies its predecessor of this failure and leaves the chain. In this case, the predecessor initiates the repair protocol unless it is the core node itself. In the worst cases the chain is terminated and the core node must begin to construct a new Matrysoka chain.

2.3.3.5 Application Services

Application services (ASs) are built on top of the Matryoska overlay and provide a suite of social networking services (SNSs), such as User Profile Retrieval, Friendship Advertisement, Insert Content, Set visibility and privacy of contents, etc.

2.3.3.6 Database

The user contents are stored encrypted in a local database. This database also stores data about remote users (to provide the replication discussed above). This database is also used by the GUI to display information for the user. This database allows a Safebook user to view, comment, upload new contents (even while offline), and schedules the events in the pending tray once internet connectivity is available.

2.3.4 P2P Substrate implementing the AS services

The purpose of this layer is to decouple the supporting P2P implementations and additional third party services from the application/user layer. The primary services of this layer, on which the Safebook user layer is based, are the Safebook Lookup Service and Trusted Identification Service (TIS). Moreover additional functionality can be added in the Safebook by adding supporting distributed services in the P2P substrate layer.

The primary services provided by the Safebook P2P substrate layer are:

Lookup Services	The Safebook lookup service implements a Kademlia[10, 11] DHT using the Safebook peers to build the underlying overlay. Kademlia supports iterative lookup, where the key is returned directly to the requestor. This scheme reveals the requestor and key holder identifiers. In order to hide the requestor and resource holders, the Kademlia lookup is modified to be a recursive lookup scheme, in that the key is returned following the same recursive path, thus hiding the identity of the originator and source due to circular nature of the DHT overlay. This service supports the following lookup operations:

Node IP Lookup	This lookup is used to find the IP address of a Safebook peer along with its status (i.e., whether the peer is online or in daemon state). The operation takes Node_IDs as input and for a successful lookup it returns an IP address and the state of the peer.
Entry Point Lookup	This lookup is used to find the Entry Point List (EPL) of a Safebook user. The operation takes a Name Vector as an input and returns the EPL for all matches. The name vector can be a single entry, such as a first name or multiple entries. For unique lookup when finding the EP for a friend, the hash of the friend's User_ID is used as the name vector.

Trusted Identification Service	The Trusted Identification Service (TIS) ensures that, every Safebook user is assigned a unique identifier in both overlay networks, i.e. the overlay in the user layer and in the P2P layer. It uses an out of band identification procedure and assigns each user a unique User_ID and Node_ID pair. The User_ID is computed as a hash of a set of user properties that uniquely identify the user in the real world, such as the user's name, date of birth, birth place, etc., whereas the node identifier is a pseudonym. The TIS does not take any part in the communication between Safebook users, thus is not a potential treat to privacy as it cannot exposure user contents.

2.3.5 Communication and Transport Layer

The objective of this component is to hide the complexity of the underlying secure communication layer from the application layer. This layer also solves the problems caused by IPv4 NAT traversal by using Universal Plug and Play (UPnP) and Secure Socket Layer (SSL) tunnelling techniques. The communication manager provides security APIs to support hop by hop and end-to-end encryption of messages to ensure confidentiality and integrity. This module utilizes UDP traffic for P2P DHT overlay messages and TCP traffic for user and Matryoska layer messages together with SSL wrapping to limit access to the traffic to specific network users.

The security objective of the communication and transport layer is to provide confidentiality and integrity for all communication between Safebook users. This is achieved by providing:

End-to-end protection	A message is encrypted with the intended receiver's public key and signed with the originating sender's private key. In this way intermediate nodes connected via insecure channels cannot interpret or change the messages.
Hop by hop protection	In order to protect the data in transit, hop by hop protection is used to encrypt the message with the intermediate receiver's private key and the encrypted message is signed with the sender's private key.

The security is implemented by the following modules:

Crypto Module	The cryptographic module provides APIs to encrypt, decrypt, and sign messages with different state of art standards.
Communication Manager – independent service	The communication manager is an independent service which manages all the communication complexities. The network IP settings and port forwarding occur through static configuration files or can be assigned dynamically using a UPnP service. The manager schedules all the received messages from the upper layers based on the message type and passes them to their respective handlers. Moreover, it uses the Crypto Module to provide end-to-end and hop by hop encryption.
SSL Tunnelling Wrapper	In normal network settings the communication manager is able to learn the public IP address along with the forwarded port numbers used by the NAT device. However, in networks where users do not have sufficient privileges to open ports in the NAT/firewall, we wrap their traffic and forward it through SSL tunnels using the SSL wrapper handlers, supported by the Safebook overlay network nodes that host SSL servers.

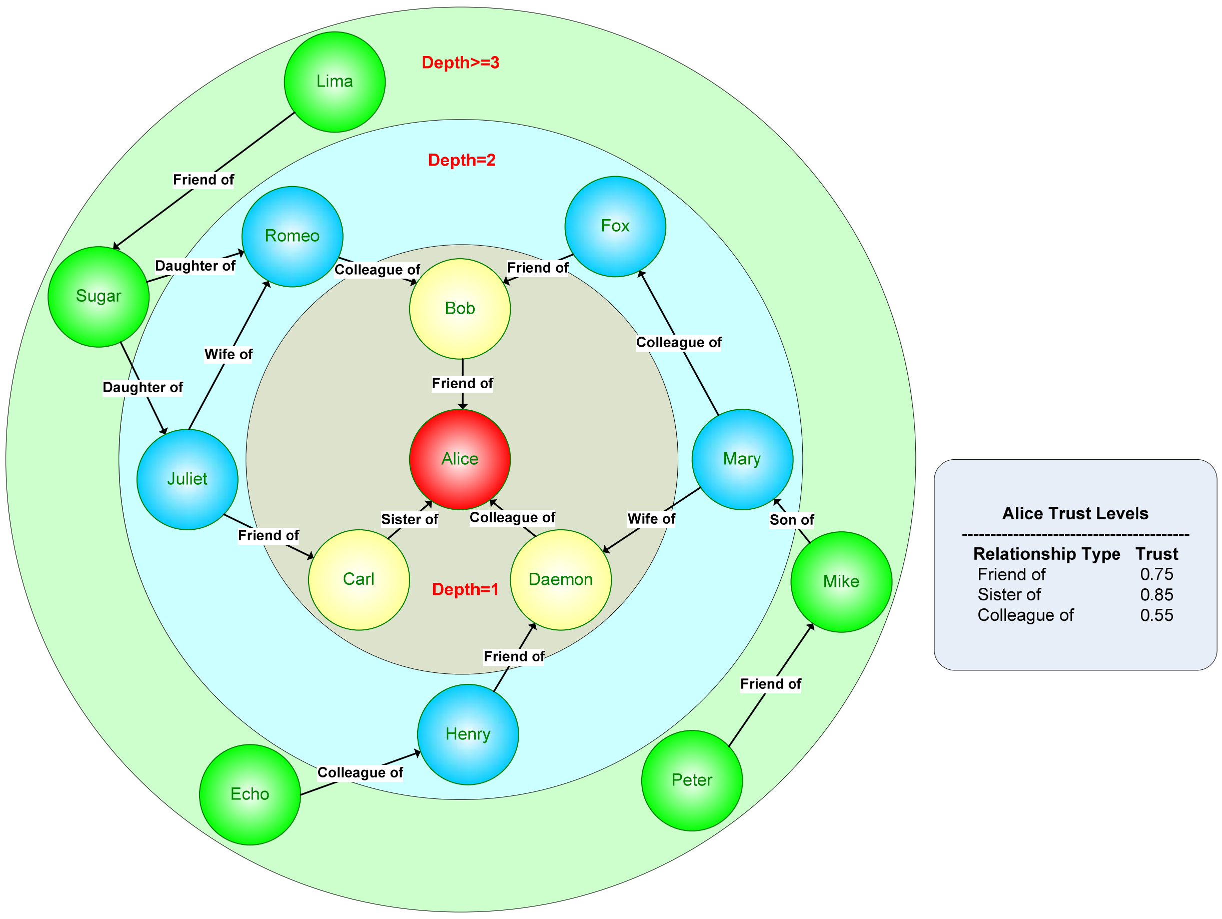
3 Access Control Requirements in a Social Network

This chapter describes the access control requirements in a social network. Section 3.1 explains how social networks can be realized over the internet and represented as a social graph based on real life acquaintances, along with social interactions and information flow. Section 3.2 analyses the requirements for an OSN based on user interaction. This is followed by modelling user interaction patterns and access control with a number of use cases.

3.1 Representing a Social Network

The social behavior of humans in the real world is based on acquaintanceships. These acquaintances surround the users and form an active bubble of his or her interactions, within which most of his or her activities occur. This bubble moves from time to time, with changes in a person's geographic location, activities, changes in relationship status, and many other factors. The same behavior can be modeled for online social networks, where the user starts building their social network by adding persons he or she knows in the real world and slowly this network grows by sharing and exploring this circle of acquaintances. As in real life, the sharing of information and daily life activities propagate from mouth to ear, hand to hand, etc. Much information which people share with their friends may concern subjects which his or her friends do not know in person, but are now shared by an acquaintance with him or her. Similarly the same behavior can be modeled in an online social network, by propagating the user's activities and information to his or her circle of acquaintances. However in real life, we do not share every aspect of our friend's life with other persons. More often we do not want to share our pictures with unknown persons; similarly a lot of other information should remain confidential or be limited to within a certain circle of acquaintances. However, the model which we proposed to represent the OSN as a social graph based on acquaintances, does not consider this filtering, which humans do based on social norms and morals. For this reason the second challenge in representing OSNs and user interaction patterns is how to limit access to the user's contents to some desired extent, based on social norms, morals, or explicitly defined by the user.

An OSN can be characterized as a connected graph based on relationships and trust which each relationship holds as illustrated in Figure 3-1. We will refer to this as a relational trust based social network (RTSN) graph. A user in a social network is represented as a node, connected with other users through edges, which represent types of relationships and associated trust levels. The edges are unidirectional and two connected nodes can have different types of relationships and different trust levels. The social network can be characterized as a graph of potential paths for information flow.



Alice Trust Levels

Relationship Type	Trust
Friend of	0.75
Sister of	0.85
Colleague of	0.55

Figure 3-1: A relational-trusts based graphical representation of OSN (All relations are with respect to Alice)

3.1.1 A Relational-trust Social Network (RTSN)

A RTSN consists of a graph showing connections, relationships, contact lists, and social circles. Each of these is described in more detail below.

3.1.1.1 Connection

A connection between two nodes in a social network is defined by the path between these two nodes within the social network. For example, Alice in Figure 3-1 is connected with Echo through Daemon and Henry. The relations are unidirectional, which implies if Bob is friend of Alice, it is not necessary the case that Alice is also friend of Bob. Bob can associate a different relationship such as “College Friend” to Alice. So the connections to Alice are unidirectional and typed (as shown in Figure 3-1).

3.1.1.2 (Social) Relationship

A (social) relationship is a level of association which a user uses to label its connection with another node in its social network. These social relations are characterized by *depth*, *label*, and *trust level*. The *depth* is a relationship parameter which is based upon the number of acquaintances along the path between two persons. For example, a Friend-of-Friend relationship is a *depth* 2 relationship. Whereas, *label* is a “tag name”, usually a meaningful name adopted from a real life social relationship, such as family, friends, colleagues, etc. The *trust level* is a value with indicates the degree to which the user trusts this relationship. Trust level ranges from 0.0, indicating totally untrusted, to 1.0, indicating completely trusted.

A depth 1 relationship which does not share any acquaintance is a direct relationship. While a relationship with depth greater than 1 and sharing at least one acquaintance is an indirect relationship.

3.1.1.3 Contacts List

Each user in a social network has a set of nodes to which they are connected with direct connections, this represents a contact list. The contact list of a user is the gateway via which this user can explore and traverse the social network graph. Moreover, the contact list forms the basic social circle of a user, as described next.

3.1.1.4 Social Circles

A social circle defines the interaction pattern of a user. It consists of a set of users, with which this user interacts most often. This set of users can be arranged by a user to form groups (i.e., sub-circles) or a bubble of active interactions may naturally form such a sub-circle. For example, Alice may have several users in her contacts list, but she may not communicate very frequently with all of her contacts. Her interactions will most often be confined to few active friends, who might be classmates, family friends, etc. These active interaction patterns form the user’s social circles, within which most of the user’s activities lie. These social circles are similar to bubbles which keep moving over time. The impact of these social bubbles is due to the fact that these social network interactions are highly likely to lead to real world social interactions, change of geographical locations, changes of relationship status, changes in work place, initiation of new activities, etc. These factors influence the user’s social life, hence the impact of the interaction pattern(s) of the user in an online social network are represented as social bubbles.

3.1.2 Social Interactions and Information Flow

This subsection describes how the user interacts with the social networks and how the information is propagated and retrieved. Moreover, this section will identify the requirements for the social networks and a number of use cases based on the user social interactions

3.1.2.1 Profile traversal

In OSNs, the user sees the network as a graph, as shown in Figure 3-2. Where the user is the root of the tree and the user’s contact list members are leaves at depth one. Similarly, the contacts lists of the user’s contacts are represented by a second level of the graph, and so on. In this way, the user can traverse the graph and explore the child profiles connected within the user’s personal social network. In this figure we have shown the connections with and associated relationship and trust level pair. In an idealized scenario, where trust is not considered, the user can traverse up to n levels, where n is restricted by the social network’s size. However when we consider trusted links, the user will only access those profiles which meet some minimum trust level. In this way, the user will only access a subset of the tree. This subset represents the social circle of the user's acquaintances based upon persons with whom the user has some degree of trust.

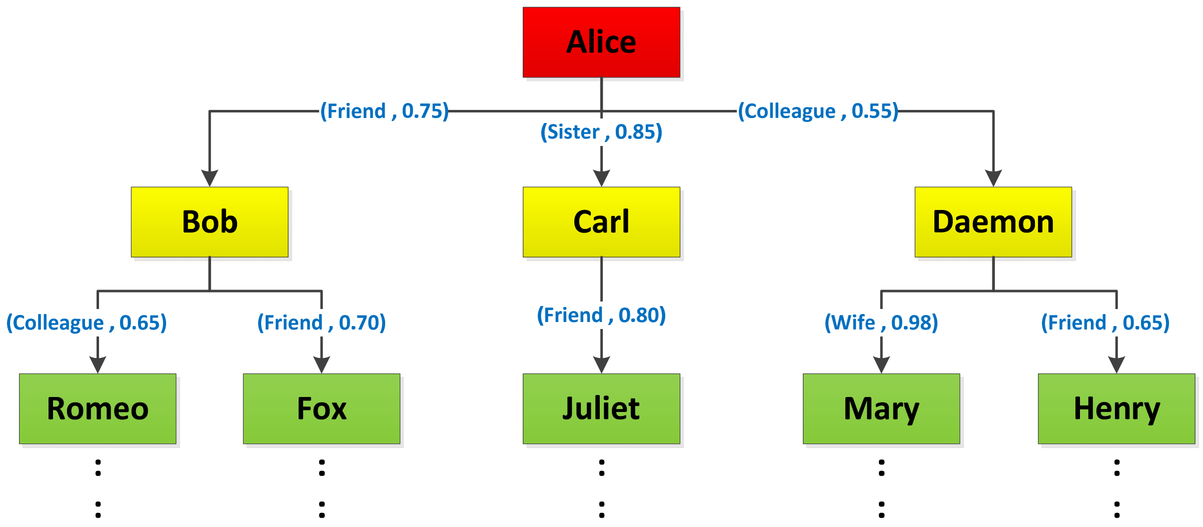


Figure 3-2: User profile traversal tree

3.1.2.2 *Information Propagation*

The information flow of user contents and activities is determined by traversal of the user's profile tree. This information flow is primarily restricted by the relationship types, link trusts, and reinforced by reachability conditions. The trust levels of each relationship form the basis for the propagation of information about the user's activities. In general for an active and interactive social network, information flow can be classified into active notifications or content sharing.

3.1.2.2.1 Activity Notifications

One of the services of any OSN is to keep track of the user's social circle's activities. The user wants to be aware of what his friends are planning and what activities they have undertaken. It would be inadvisable to rely on only the profile traversal property of the social network, where the user traverses only their user profile and manually explores the contents and activities of those that they can reach via this graph. Instead the user should be automatically informed by notifications concerning his or her friend's activities. However, the challenge is to see that the user only receives *relevant* and *meaningful* notifications. With respect to the profile tree, these notifications originate from the leaves and propagate upward to the root of the profile tree, thus the parent node can decide how to filter the notifications received from its children. This filtering is done according to some criteria and access control mechanism so that notifications only propagate those which a parent node wants to share with its parent. For example in Figure 3-2, Daemon has two child nodes, Mary his wife and a friend Henry. Daemon has Alice as a parent node. Alice expects to receive notifications from Daemon about his activities. However, Daemon may not want to share all of his wife's activities with Alice, hence Daemon will restrict some notifications, for example filtering out update's to his wife's picture gallery, comments on her personal activities, etc. while still allowing basic notifications concerning her birthday, basic profile information, general activities, and some user contents to propagate to Alice. As a result Alice will be notified about only a *subset* of Mary's activities, whereas Daemon is aware of all

Mary’s activities that have been shared by Mary. There are two aspects to consider: (1) Mary decides what to share with Daemon and her contacts and (2) Daemon can additional filter this information to limit the propagation of Mary Activities. Similarly, for Henry, Daemon may apply more relaxed access rules, thus allowing Henry’s activity notifications to propagate to Alice, subject to these access rules.

3.1.2.2.2 Content Sharing

Profile traversal and notifications describe *how* to reach a certain resource via the social network. Notifications signal the availability of new user content. These notifications are short messages that are published with defined propagation rules. Once the intended user receives a notification, this notification may trigger a lookup of the associated resource (i.e., the user content). Clearly in order to protect the privacy of this content we need to define an access control mechanism (this will be discussed in the next paragraph). In order to limit the sharing of content, each user defines the access and privacy rights for each of their resources in terms of relationships, trust, depth of relationship, and other reachability conditions.

3.1.2.2.3 Access Control Component

Access control limits the propagation of information. The access granted to another user (or process) is defined by access polices and privacy settings that have been set by the user for this user’s contents. The traditional access control schemes do not fully supports all the requirements and needs of social networks. One of the most important requirements is that the access control component ensure the expected (and permitted) social behavior of the social network, while protecting the user’s privacy *and* granting access to desired users, groups, sub-groups, and others based upon conditional access to the user’s resources. In a social network access control has two aspects:

1. **Who** can see my contents?
2. **Whose** activates should I be notified about?

Both of these aspects of access control for OSNs will be examined in the next section.

3.2 Requirement Analysis for Access Control

The primary objective of Safebook is to ensure the privacy of user contents in a OSN, while still enabling a user to share the contents that they wish to share with only those whom they wish to share it with. This privacy is partially ensured by the Safebook design itself, due to its secure and privacy aware communication components, as was discussed in section 0. However, communication privacy alone is insufficient to handle the user’s needs with regard to sharing contents. Therefore, there is a need for an access control mechanism to control access to user contents in a comprehensive way, while meeting the user’s requirements for data dissemination while keeping the social network secure and interactive.

In order to define an access control mechanism for Safebook, first we need to define and understand the user’s requirements for access control and the social interaction patterns governing these access rules. The access control mechanism is comprised of two entities, first the user profile (a representation of the user contents in the form of a dossier) and second the access rules themselves. These access rules will be discussed in Chapter 4. The requirements analysis for the desired Safebook access control will be done in context of the user profile as defined in the next subsection.

3.2.1 Safebook User Profile

A Safebook user profile is represented as a collection of data artifacts and an access control component as shown in Figure 3-3.

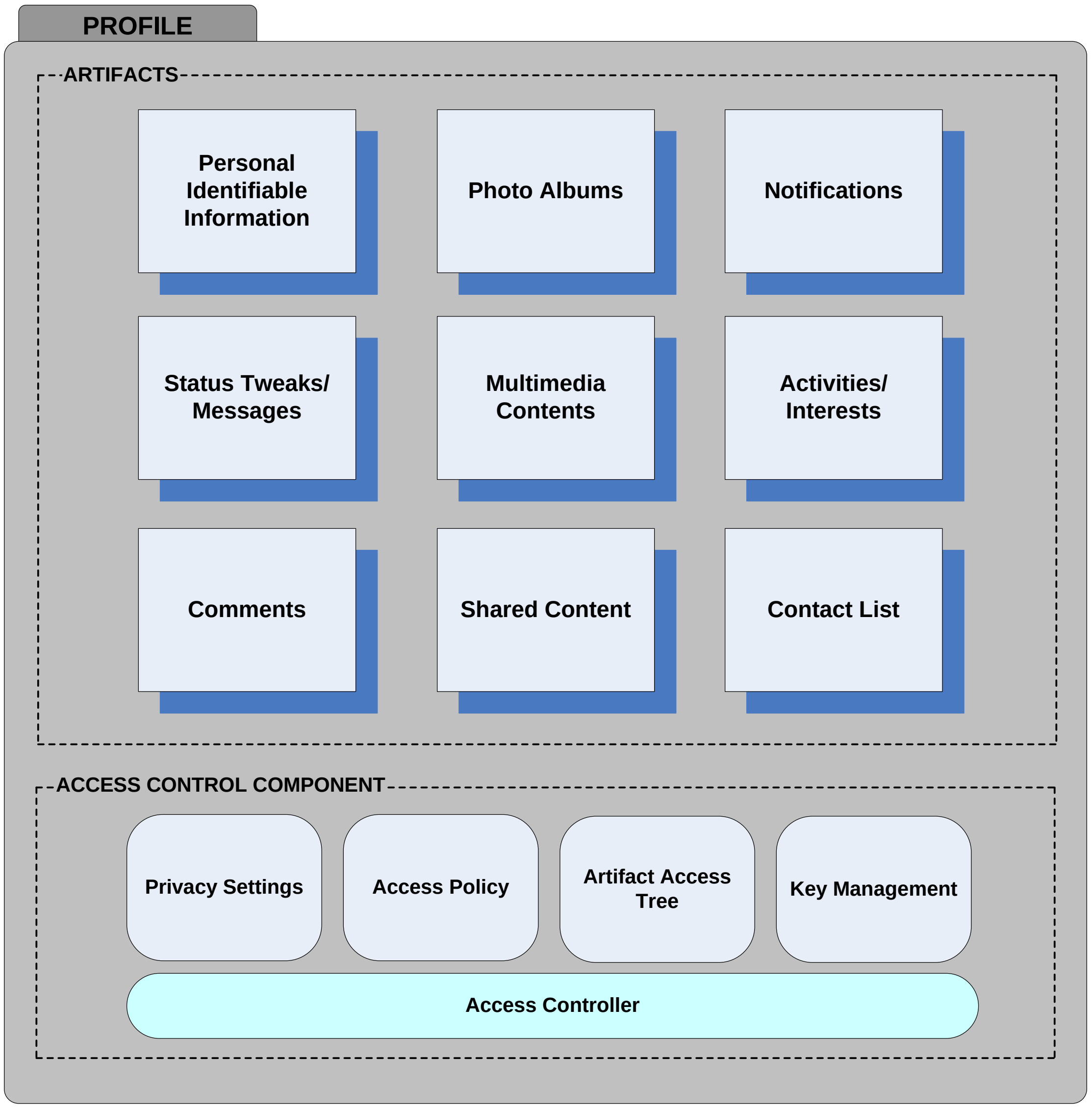


Figure 3-3: The two components of a Safebook user profile

3.2.1.1 Artifacts

Each artifact represents a uniquely identifiable data object with a set of attributes, for which an access rule can be defined. These artifacts can be grouped using universal labels, such as those used in Figure 3-3. Each label corresponds to a set of attributes which are inherited by all the artifacts with that label. Using this approach, similar types of data, either in the form of binary contents or logical contents is stored with the same artifact label. Each label represents a table in the database which stores the user’s profile. The idea behind representing a profile as a collection of artifacts is to make it as flexible and extensible as possible. New artifact labels can easily be added to the Safebook artifact collection.

3.2.1.1.1 Artifacts: General Labels

The artifact labels can be of several types, depending on the user's needs and to support future innovations. In Safebook, we begin by defining some general labels, which can present almost all activities of current online social networks. Brief descriptions of all of these currently defined labels are given in Table 3-1.

Table 3-1: Artifact labels

Personal Identifiable Information	As the name implies, these are the data artifacts which could uniquely identify the user in the real world. These artifacts include the user's name, nationality, gender, date of birth, email, phone number, work place, etc.
Status Tweaks/Messages	Status tweaks or messages are short textual messages; usually used to indicate the user's mood, current activity, etc. These messages are written on your profile wall and usually propagate to your contacts as a function of your privacy settings.
Comments	Comments are textual messages which are always associated with some artifact. For example, if Alice uploads a picture of her new car. Bob will receive a notification of this new picture and then he can view the picture and write a comment about this picture, such as "Waooo Nice ride…" Bob's artifact is a comment which is associated with Alice's artifact. In this way a number of connected comments can be attached to a single artifact. This chain of comments concerning a single artifact will be referred to as an activity thread or simply as a thread.
Photo Album	Sharing and uploading pictures on social networks has become the *de facto* service of any OSN. The user's pictures are usually organized into user defined photo albums. Each photo album has associated attributes such as: name, place, date, event, description, etc. If no album is defined by the user, then the pictures are stored in the default album of the user's profile. The pictures in the photo album may each have their own attributes, such as: title, description, place, etc. Moreover, individual pictures as well as whole photo albums can receive comments from your friends and these comments form a thread.
Multimedia Content	Safebook supports multimedia contents, such as audio clips, video clips, etc. in a similar way as for photo albums. The multimedia contents are usually stored in large files which are distributed as small chunks for easy and quick access. However, the details of the distribution and retrieval of multimedia data is outside the scope of this thesis. We will simply assume that an item of multimedia content will be handled as a single resource, i.e., an artifact.

Shared Content	Shared artifacts refer to those artifacts not owned by the user (i.e., they are owned by another user who owns the shared resource). In a social network, such artifacts are very common and include tagged photographs, comments, etc. Such tagged resources become part of a user's artifact repository and access rules can be defined for them.
Notifications	Notifications are small textual messages, which contain short descriptions regarding a friend's activities. These notifications are disseminated to a user defined group according to the user's access settings. In a OSN these notifications are used to make other users in the social network aware of the user's events and activities. From an implementation point of view, the reception of a notification can automatically trigger a fetch operation for the resource or the fetch operation might only be triggered if the user explicitly requests the resource.
Activities/Interests	Activities/Interests represent a user's interest groups, forums, fan page, etc. Usually, such an activity poses a potential threat of privacy leaks, because the user's information is exposed to an unknown set of users and this information could be exploited for purposes other than what the user might want. For example, by sharing a single picture in a forum another user might gain access to the user's complete album (in which this picture is located), which may not be desired by the picture's owner. Thus, for such artifacts, a more restricted access policy referred to as a sandbox is suggested to preserve user privacy and to prevent potential misuse of the user's information. For this reason these types of artifacts are handled in a sandbox environment and granted very limited access to the user's resources and information, unless they are explicitly granted access.
Contacts List	The connectivity of a social network is based upon the contacts of all of the individual users. These contacts connect a user with other users and recursively form a connected (social) graph. While all the users in the social network could be part of a global graph, the users communicate and interact within their own particular social circles, i.e., subgraphs. The Contact List refers to a collection of Safebook nodes which share a direct relationship status. These nodes are the first point of interactions and act as gateways to other parts of the social network. The artifacts relating to the contact list are treated just as other data artifacts and thus privacy and access rules can be defined. Moreover, these artifacts also play a role in defining access rules and policy.

3.2.1.1.2 Activity Thread

Every artifact in a Safebook profile is associated with a Thread ID when it is stored. This Thread ID is a random integer which represents a complete activity. Resources such as comments are linked to the Activity/Thread Owner's Thread ID. When Alice uploads pictures, the Safebook engine will assign a unique random Thread ID for each of these artifacts. If Bob comments on this picture, then his comment will be associated with the Thread ID of the picture upon which he commented, thus forming a chain of resources linked to a single activity.

3.2.1.2 Access Control Component

An access control component is a collection of a number of sub-components and a controller, as shown earlier in Figure 3-3. These components define and maintain a specific activity, the details of which will be explained in later chapters. A brief introduction to these components is shown in Table 3-2.

Table 3-2: Access control sub-components

Privacy Setting	Manages and maintains privacy rules for the user's artifacts and their propagation in social graph
Access Policy	It is a list of access rules governing some relationship type, user defined groups, individual users or particular data artifacts
Artifact Access Tree	It's a logical arrangement of user data artifacts where every artifact is connected with its child artifacts and the connections/link holds the access rules
Key Management	This component is responsible for security protocols which manage, maintain store, retrieve and distribute user encryption keys.

In addition to these sub-components, there is an Access Controller, which handles and processes each request based on the defined access rules. Similar to the different types of artifact, the access control component can also easily integrate new functional components.

3.2.2 Modelling User Interaction Patterns and Access Control

Before we define the access control model for a social network, we first need to understand how a user interacts with a OSN and what the user expects from this OSN. In this subsection we will identify possible ways for a user to interact with their contents and how they might share an artifact with their contacts. We will do this while keeping in mind the hierarchal model of information propagation shown in Figure 3-2. This subsection will identify possible use cases for artifact access control in a OSN.

The model suggested for Safebook is based on defining access control over a user's OSN based interactions. This approach offers flexibility and adaptability to new ideas and innovations. The user's social interactions can be categorized into three categories: direct relationships, indirect relationships, and user activities and interests. In the following paragraphs we will describe each of these different categories of access control use cases concerning social interactions.

3.2.2.1 Direct Relationships

The most basic category of social interactions of a user in a social graph (Figure 3-2) occurs at the first level of the tree, i.e. the child nodes of the parent in the profile traversal tree. These child nodes or siblings of a parent share a direct relationship with at least a user's selected minimum trust level. These nodes are the first point of interaction of a user with their social network. So, the first and most basic criterion for access control is based on the relationships of a user with his/her interaction circle, i.e., with his/her child nodes. These siblings are referred as contacts or members of the user's contacts list.

3.2.2.1.1 Badges

In the context of our Safebook based social network direct relationships are also referred to as badges. A badge consists of a label and a number of attributes. We can divide the different use cases for access control with regard to direct relationships into uses cases with unique badges or with multiple badges.

3.2.2.1.1.1 Unique Badges

The use case for a unique badge represents an ideal scenario in a social network, where the parent has a set of child nodes associated with a *disjoint* set of badges. In other words, each node in the contacts list has only one badge. This scenario is illustrated in the Figure 3-4, where Alice is the parent node and her contacts list includes Bob, Carla, and Daemon. As a social network is a relationship based network, each child must be associated with some specific relationship; here these are identified by badges. Alice can define a number of badges and associate different attributes with each badge. In the scenario shown in Figure 3-4, Alice defines 3 badges: Friends, Family, and Work. The nodes of Alice's contacts list are associated with these badges and different attributes are set according to Alice's relationship with each of these contacts. For example, in this contacts list Bob has a "Friends" badge and this badge has a Relationship Type attribute of "Friend". Similarly, Carla has a relationship as a sister of Alice, hence Carla will be assigned a "Family" badge with an attribute set to "Sister". This approach can define very specific relationship types, thus access control can be defined in very small granular steps.

The second part of the Figure 3-4 illustrates how Alice can define access to her artifacts based upon their badges and their associated attributes. There are two approaches to defining this access control. The first is to associate badges with the artifacts by defining an access policy for each badge. The second approach is to associate the artifacts with the access conditions defined over badges and their attributes. We will first consider the latter approach, as it offers greater flexibility and more control. The access rules will be defined by using a Direct-Union-Interaction rule (DUI).

Figure 3-4 shows how DUI can be used to define access based upon a user's badges and their attributes. Here "Direct" refers to rules defined over a badge or an attribute, such as a "Friends" badge or a "Sister" attribute. Here "Union" refers to computing a logical OR between several badges/attributes, such as "Friends OR Family", "Brother OR Sister", etc. Finally, "Intersection" refers to the logical AND of badges/attributes, e.g. "Friends AND Family", "Family AND Sister", etc. An access rule can be defined with complex union and intersection conditions, but generally in real social networks such complex conditions are very rare, hence we restrict our model to only simple conditions.

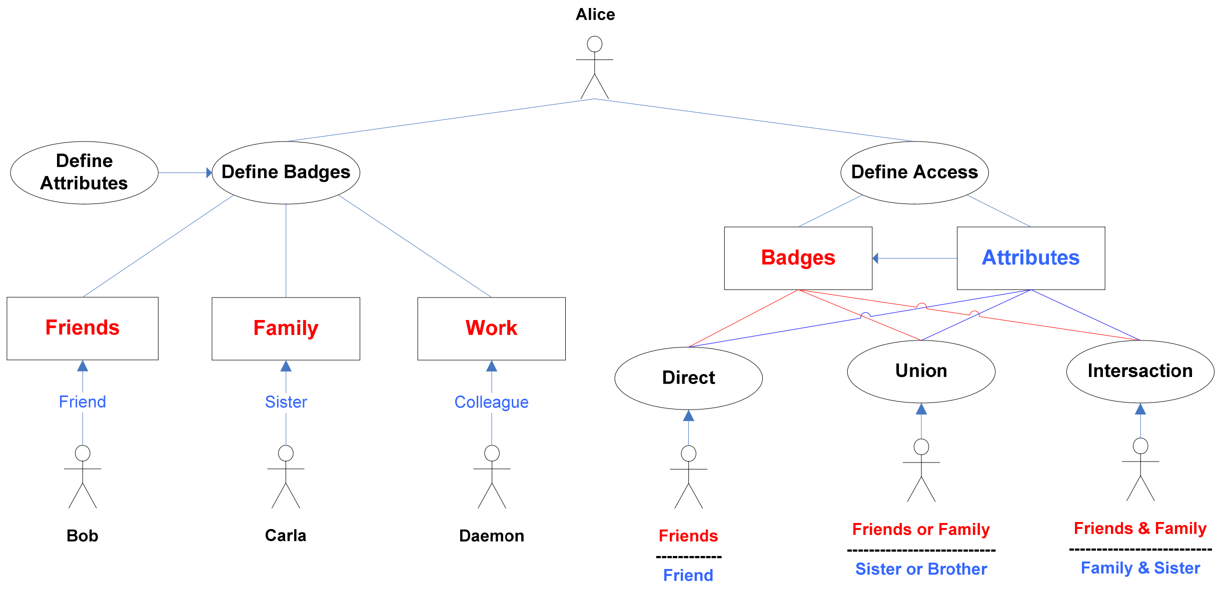


Figure 3-4: Interaction Pattern of Alice with Unique Badges

3.2.2.1.1.2 Multiple Badges

Multiple badges are needed to represent real life scenarios where one user has several different relationships with others. A simple example is that a brother is a family member, but can also be a friend. In the latter case a brother will be part of both the friends circle and the family circle. Similarly, in terms of an OSN, a brother might wish to be updated with the events which his sister is planning with her friends, via a message such as “How about Soccer tonight?”. Moreover; he is also part of Alice’s family. In such scenario, the user can assign several badges to a particular contact. Figure 3-5 illustrates a scenario where Bob is Friend of Alice as well as her Brother, thus Bob has two Badges: Friends and Family.

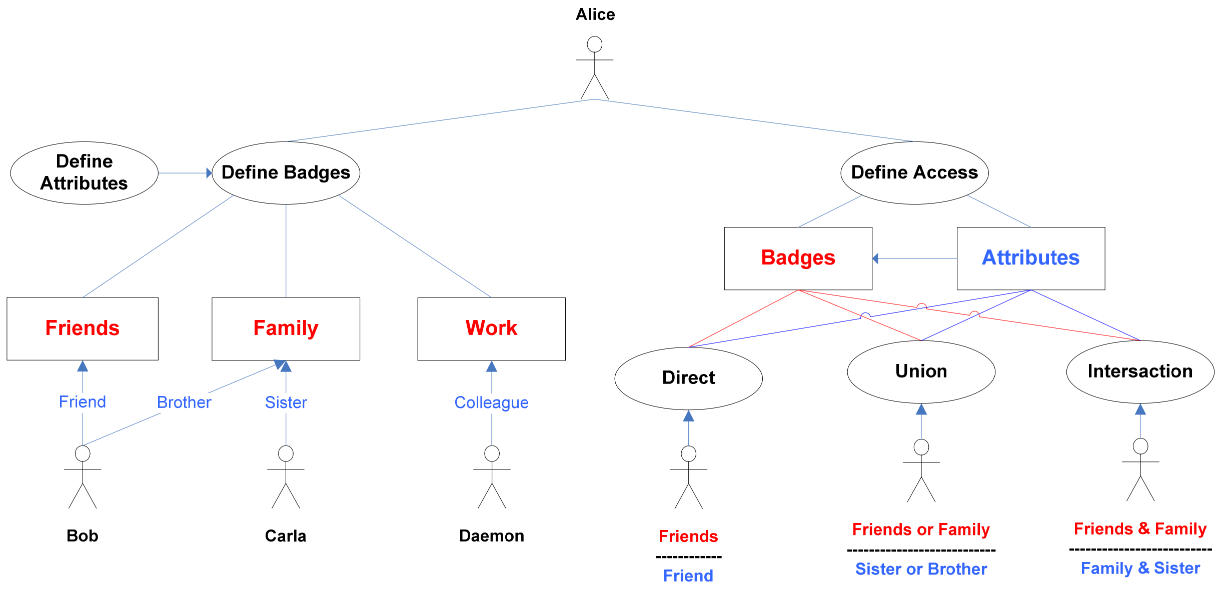


Figure 3-5: Interaction Pattern of Alice with Multiple Badges

3.2.2.1.2 Membership Groups

Access classification of the user’s contacts based on the relationships or badges is **not** sufficient to support a social networking environment, because it is not always the case that the user communicates or shares only with a particular type of badge or only with a specific set of relationship types. In real life, a person tends to share many activities with different circles, but still the person expects to maintain some privacy of his/her activities. Additionally, there is a need to know in advance who will participate in these activities. Consider a scenario where Alice is organizing a birthday party for her child. Alice will invite a number of her child’s friends and family members to her home to celebrate the child’s birthday. So it would be unexpected that someone without any formal acquaintance to Alice’s child would join the party, as Alice expects the party to be attended only by a subset of her child’s closest friends and relatives. This behavior is incorporated in an OSN by introducing membership groups. The user can create a membership group based on some activity and know *a priori* who can view the contents to which this group is given access. These are groups are only intended to organize members of the user’s contacts list. So regardless of the relationship type, any contact in the user’s contacts list can be assigned to be a member of a specific group. Thus a group is composed of nodes that have different relationships to the user and this group’s access policy will be applied to the artifacts to which the user has given this group access.

Group membership can be divided into single and multiple membership cases.

3.2.2.1.2.1 Single Membership

Figure 3-6 presents a group membership scenario where the members are grouped into disjoint sets. It is a simple and idealistic scenario where a contact in the user’s contacts list may belong to at-most one group. Within a group the contacts with different relationships and badges are grouped together, thus these contacts can interact within this closed circle. In this manner we can protect the privacy of each and every member of this group and no content will be shared outside of this group’s scope. As shown on the left in Figure 3-6, Alice can define a number of groups. Each group is associated with a set of group attributes. These attributes can include group name, purpose, motivation, administrator, etc. In this scenario, Alice has defined two groups “High School” group, where Alice has invited her former high school friends. In this scenario Bob and Daemon are part of this group.

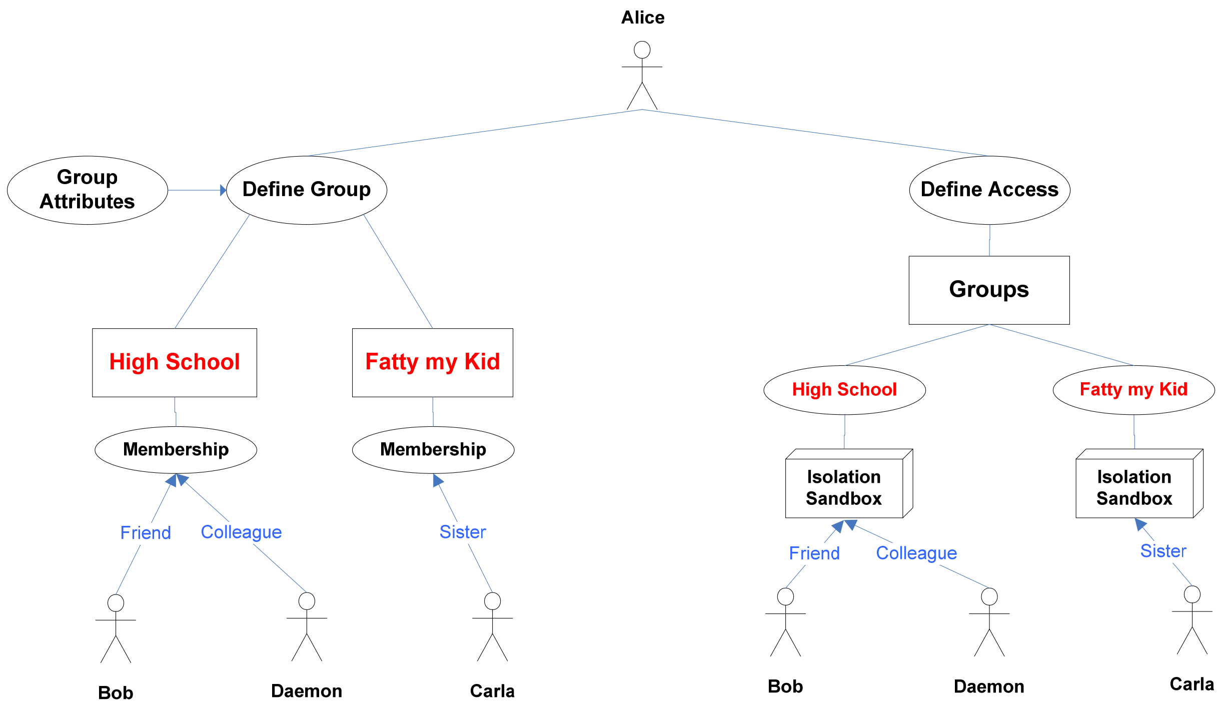


Figure 3-6: Interaction Pattern of Alice with Single Group Membership

Similarly, in the other group “Fatty my Kid”, as the name implies Alice plans to shares the activities of her child and her sister Carla is member of this group. In the right half of Figure 3-6 we can see that Alice has defined an access policy for the data artifacts belonging to each particular group. So the artifacts which are shared within particular groups remain isolated from the other groups. This isolation corresponds to an “Isolation Sandbox”. The purpose of each isolation sandbox is to protect the activities concerning a particular artifact in a group’s scope. For example, Alice can share the same picture with both the groups, thus the same data artifact is within two different groups’ scopes. However, the isolation sandbox will make sure that activities such as tagging, comments, etc. always remain within a single scope and are not propagated to other groups sharing the same artifact.

3.2.2.1.2.2 Multiple Memberships

Multiple membership scenarios represent a more realistic view of a social network. The creation of OSNs are motivated by the social behavior of humans in real life, thus a person may be part of several different activities. Therefore in an OSN context, a single contact can be part of multiple groups that have been defined by the user. Such a scenario is shown in Figure 3-7, where Daemon takes part in both of groups belonging to Alice.

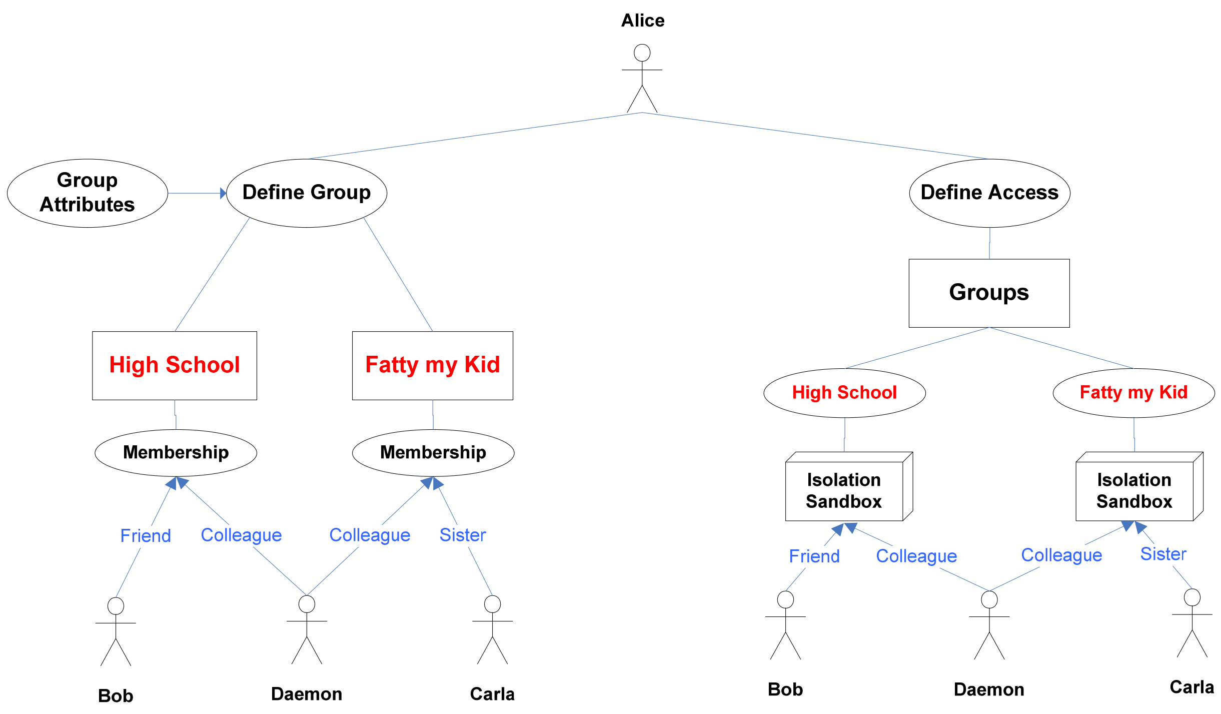


Figure 3-7: Interaction Pattern of Alice with Multiple Group Membership

3.2.2.1.3 Circles

Membership groups are a very effective way to organize a user's contacts and to provide privacy for specific activities. However, membership groups lack flexibility in terms of regrouping a user's contacts due to the strong coupling to group membership. For example, if Alice removes Daemon from her "High School" group, then the Daemon will get a notification that, he can no longer access this group. This is normal for any membership group, but this is less attractive in the context of an OSN where events are very dynamic and group membership changes frequently. A better and easier way is to avoid the concept of group membership and instead weakly couple contacts into circles. These circles represent a local view of a user's contacts in the form of specific groups (circles). The user can define a circle with specific members or arrange their contacts by specifying membership in a circle based on user attributes. The important idea is that the circle represents a local arrangement of a user and the members of the circle are not aware of what circles this user belongs to. This further guards the user's privacy about their other contacts. The circles interaction pattern can be model in terms of unique circles, multiple circles, or individuals and sub-groups.

3.2.2.1.3.1 Unique Circles

In order to comprehend the unique circle scenario we consider a simple case where the members of circles form a disjoint set, thus each user can belong to at-most one circle of the user's defined circles. A user can define different circles and associate certain attributes with each circle. Figure 3-8 shows a unique circles scenario where Alice has defined two circles. The first circle "Stockholm" has an attribute "place=Stockholm" and all of her contacts living in Stockholm are grouped in this circle. This is a very natural criterion for a user, as their social circles and interactions are frequently characterized by geographical locations. For example, when a person living in Stockholm moves to Paris for a new job, the new contacts and friendships which this person makes in Paris are unlikely to be related to this person's social circle in Stockholm, thus based on user attributes such as place the user can form a circle representing their geographic based circles.

Moreover, other user attributes can be utilized for other purposes. For example, if Alice wants to invite her Stockholm friends for a dinner, but wishes to invite only her married friends, then Alice will create a circle with 'Place' and 'Relationship Status' as a selection criteria to send invitations to her party. The second circle which Alice has defined is called "Best Friends". However, it is not possible to identify Alice's best friends simply based upon her contacts' attributes. So in this case Alice explicitly adds Bob as her best friend. This model will give great flexibility and will enable a user to define real life access control scenarios in OSNs.

The right half of Figure 3-8 shows how Alice has defined access for her data artifacts over her user defined circles. Here Alice uses a DUI rule to define an access policy for her data artifacts. Alice can define access rules such as "Stockholm OR Best Friends", in order to share the artifacts within different circles. This is different from the case of membership groups where the access policy is restricted to a specific group, but the artifacts can have access rules based on Union and Interaction with *different* circles. Once the access policies are defined for the artifacts the access remains isolated within the circle's scope. The isolation sandbox plays the same role as was the case for membership groups.

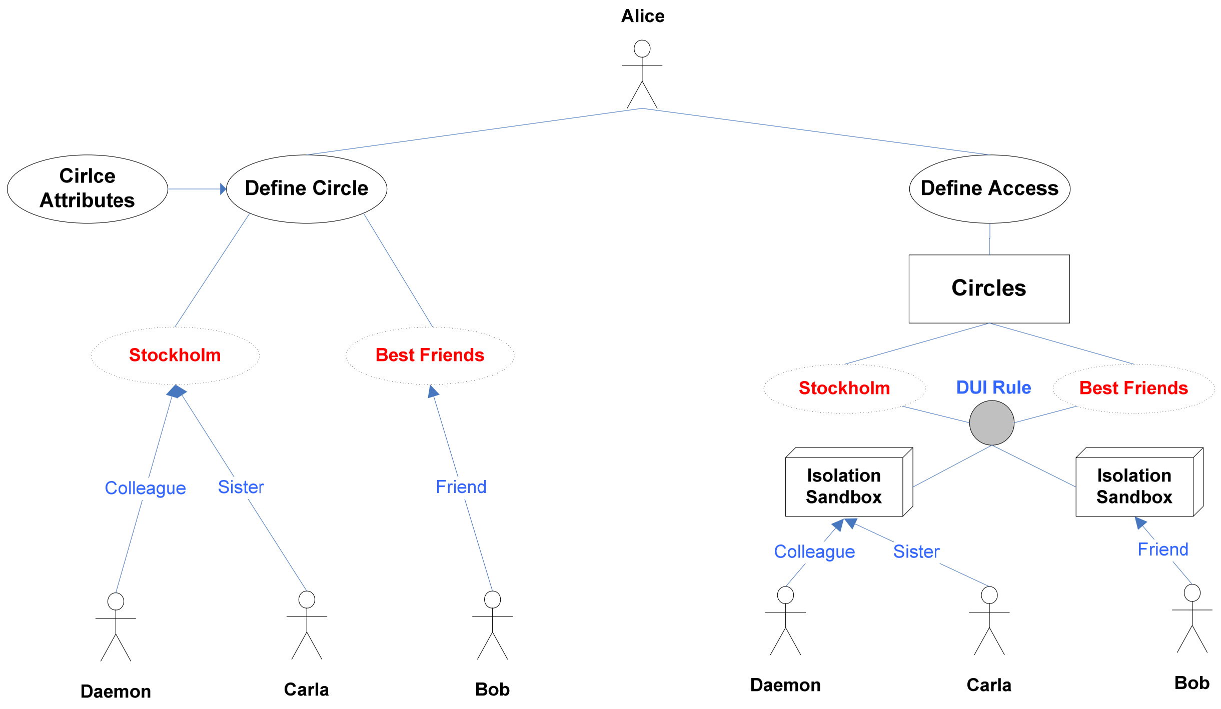


Figure 3-8: Interaction Pattern of Alice with Unique Circles

3.2.2.1.3.2 *Multiple Circles*

In a realistic social network scenario, a contact may belong to several user defined circles. This scenario is presented in Figure 3-9 where Carla belongs to both of the groups defined by Alice. In a latter section of this thesis we will discuss how the implementation will handle this case and what will be the impact and cost for doing so.

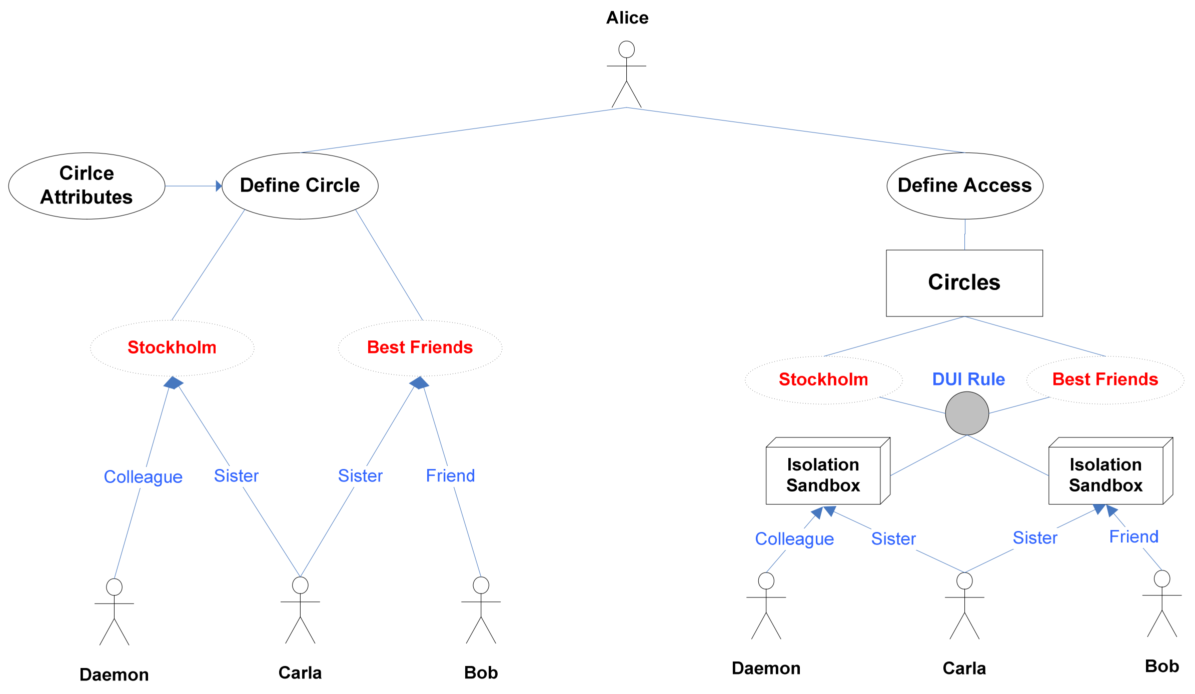


Figure 3-9: Interaction Pattern of Alice with Multiple Circles

3.2.2.1.3.3 *Individuals and Sub-Groups*

The utility of circles can be extended to membership groups as well by forming sub-circles within a group's scope. The left half of Figure 3-10 presents such a scenario, where Alice defines a circle "Baseball Club" within her "High School" group. This sub-circle can be used by Alice when she wants to make an announcement to all her friends in the "High School" group with whom she plays baseball at school. Alice wants to request her friends to delay the game for 30 minutes as she broke her bat and will buy a new bat before the game. This request is intended for only a specific audience within the group, thus Alice defines a sub- circle within her group and shares this information.

The right half of Figure 3-10 shows how access can be defined using an access hierarchy. This access hierarchy is defined at three levels. The root node is group access; at this level access policy for the complete group is defined. The second level is circle access; it defines a specific circle's access policy and inherits its access rights from the root note, i.e. the group's access rights. On the lowest level is individual access; this define specific rules for individuals within the group or circles, the individual access can be inherited from the group access or circle access.

Consider three artifacts A, B, and C as shown in Figure 3-10. Access for artifact A is defined at the root level, i.e. the group access for Alice's group "High School". Due to this group access all of the members directly belonging to this group have access to this artifact. In our scenario Carla is a direct member of this High School Group. Additionally, Alice has defined a circle in the group so that at the circle access level, the child of the root node inherits access to A while access to artifact B is defined at the circle level. As a result the members of “Baseball Club” can access artifacts defined at the circle level and the inherited artifacts from the root level.

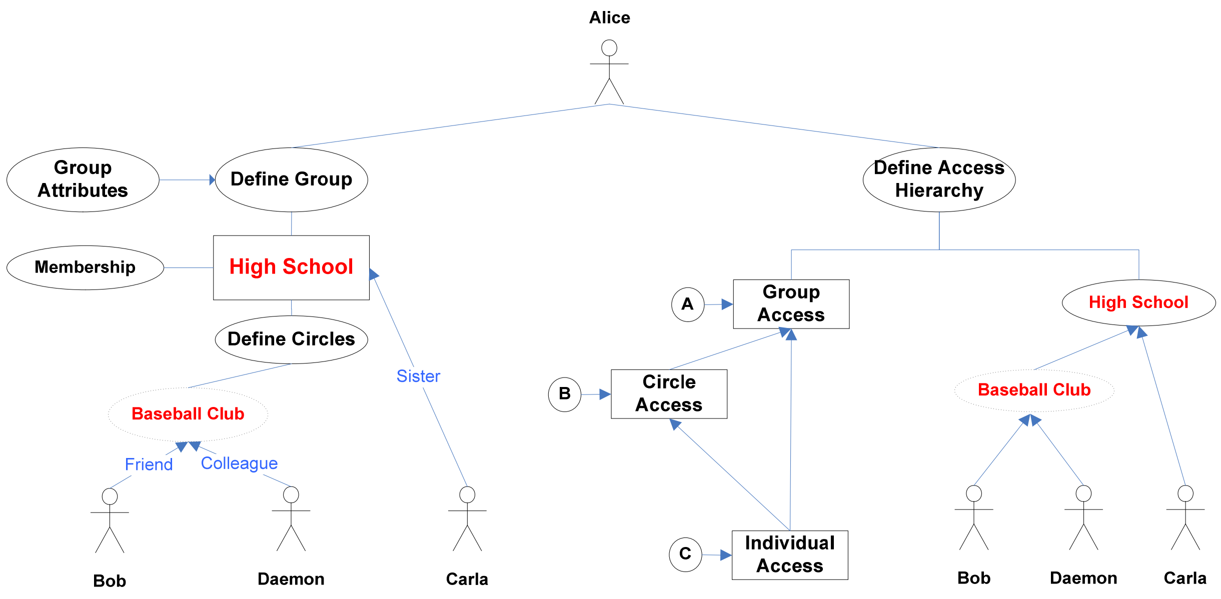


Figure 3-10: Interaction Pattern of Alice with Access hierarchy

Moreover, access to artifact C is defined at the third level, i.e. individual access. In this case, artifact C will inherit access from the circle access or from the group access. As a result artifact C is accessible to Daemon and Carla. In Figure 3-10 we can see that, Daemon will have access to A, B, and C, while Carla will have access to A as defined at the root and the artifact C as defined at the individual level.

3.2.2.2 Indirect Relationships

A social network disseminates information based on gossiping techniques. The information and activities of a user spread from one circle to another circle with which a user shares an acquaintance. But in the context of privacy aware social networks, this property of social networks needs to be restricted in order to ensure the privacy of a user’s contents. The access to a user’s contents by indirect relationships such as a “Friend-of-Friend” can be controlled and governed based upon propagation depth and trust levels.

3.2.2.2.1 Propagation Depth

The depth of a social graph is represented in Figure 3-11, where depth=1 represents direct contacts of a user, i.e. the other users in a user’s contacts list. Depth=2 represents the direct contacts of these contacts and similarly for depth>2. This parameter is a very simple and effective parameter to restrict access to a user’s artifacts. For example, Alice defines that her date of birth should be viewed by at most others with a propagation depth of 2, while her avatar should be restricted to those at a propagation depth of 1. So in this case only her direct friends can view her avatar, while a friend-of-friend can see her birthday but not her avatar.

In order to protect the user’s privacy, an intermediate node can notify others regarding Alice’s activity, but any interested notified node must send a fetch request for the artifact to Alice. Alice will only entertain this request if the request provides a relationship certificate which satisfies her established access criteria.

3.2.2.2.2 Trust Levels

The other way to restrict access to a user’s artifacts is by utilizing relationships’ trust levels. As a result information will only be shared with indirect relationships if the level of trust satisfies the user’s minimum required trust level. In order to simplify the system of trust levels, in our model trust is defined to have only three levels: high, medium, and low.

We consider a simple scenario where Alice defines that her date of birth can only be accessed by indirect relationships which can prove a high trust level. Referring to Figure 3-11, Alice has a friend Bob with a high trust level and Bob trusts his wife Carla with a high trust level and Mary with a medium trust level. So, Alice’s date of birth will be shared with Carla as well, because she can prove that she has high + high = high trust level. Similarly, Bob trusts Daemon at a high trust level and Daemon trusts Echo at a medium trust level, so the resulting trust level is high + medium = medium, and thus the information about Alice’s birthdate will not reach Echo. In this scenario only the friends of Alice in the first circle, i.e. depth=1, along with Carla will be aware of Alice’s birthday. Note that trust levels are only used for *indirect* friends.

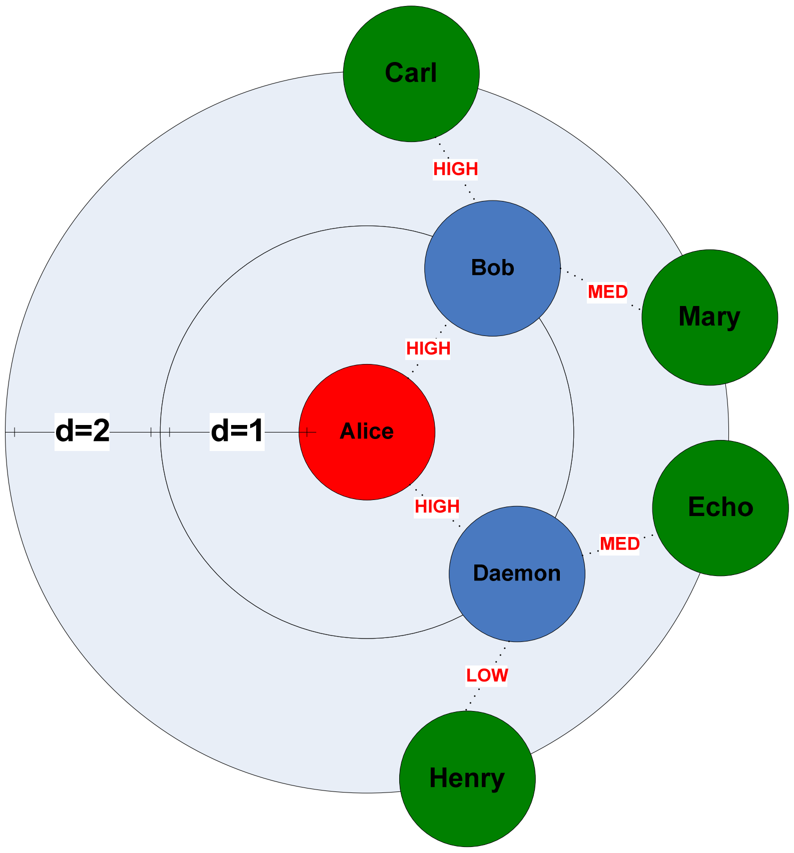


Figure 3-11: Information dissemination based on trust levels

3.2.2.3 *Activities & Interests*

Social networks present a very wide range of interaction patterns. The user is not confined simply to his/her contacts and social circles, but the user can always look for and participate in the wider social network. Based on interests and activities, a user can explore the social network to find information which interests him/her. For example, a user is interested in finding a wedding designer's clothing. The user can explore the social network looking for wedding designers and see on these designers' fan pages where they can learn additional information. Moreover, the interested user can subscribe to these pages and get notifications of activities concerning this designer's fan page. Such activities in the social network are beyond the scope of the user's contacts and social circles and thus require a strict access policy. These activities may expose users to an unknown set of peers in the social network. Furthermore the agendas and behaviors of these peers are not known in advance. There is a high risk of privacy leakage and malicious peers can gather information, facts, and preferences about their potential victims. Figure 3-12 represents a scenario, where Alice participates in different activities and interests via the OSN.

Alice can interact with interest groups in three different ways. First, Alice can create an interest group and become the administrator of the interest group's page. Second, Alice can join an existing interest group, in this way she can now access the group's contents and receive notification of this interest group's activities. Third, Alice might only view an interest group, as there are plenty of interest groups which are public and can be viewed without requiring membership. These interest groups have open memberships and any user in the social network can subscribe to this group's content.

The owner of an interest group has always the administrative right to add or remove any member he/she likes or restrict the visibility of the interest group. This access control is defined in a similar fashion to a membership group with slight modifications to give stricter access control over user artifacts. In addition to isolation sandboxes for activities in a group's scope the interest groups are also confined in a resource sandbox. The purpose of this resource sandbox is to protect the sensitive information about a user and this is achieved by decoupling the user artifacts from the regular stream to create separate dedicated streams, each belonging to a specific interest group. For example, if Alice wants to share her picture from a photo album with her football interest group, then in order to protect the original artifact and its links to other artifacts a clone is generated with new artifact id and access to this new artifact id is added to the interest group. This artifact is only related to this interest group via a separate encryption key so that an attacker cannot correlate any artifact found in the activity group with the user's activities. Figure 3-12 shows how access is define for Alice's interactions with interest groups. Alice can view different activities in a read only mode, which will protect Alice's private information from malicious third party applications and prevent these applications from placing anything on Alice's wall. Moreover, Alice is the administrator of the "Football" interest group and a member of the "Cooking" interest group. In both these activities the users have very limited access to Alice's profile and can only view the activities within the interest group's scope.

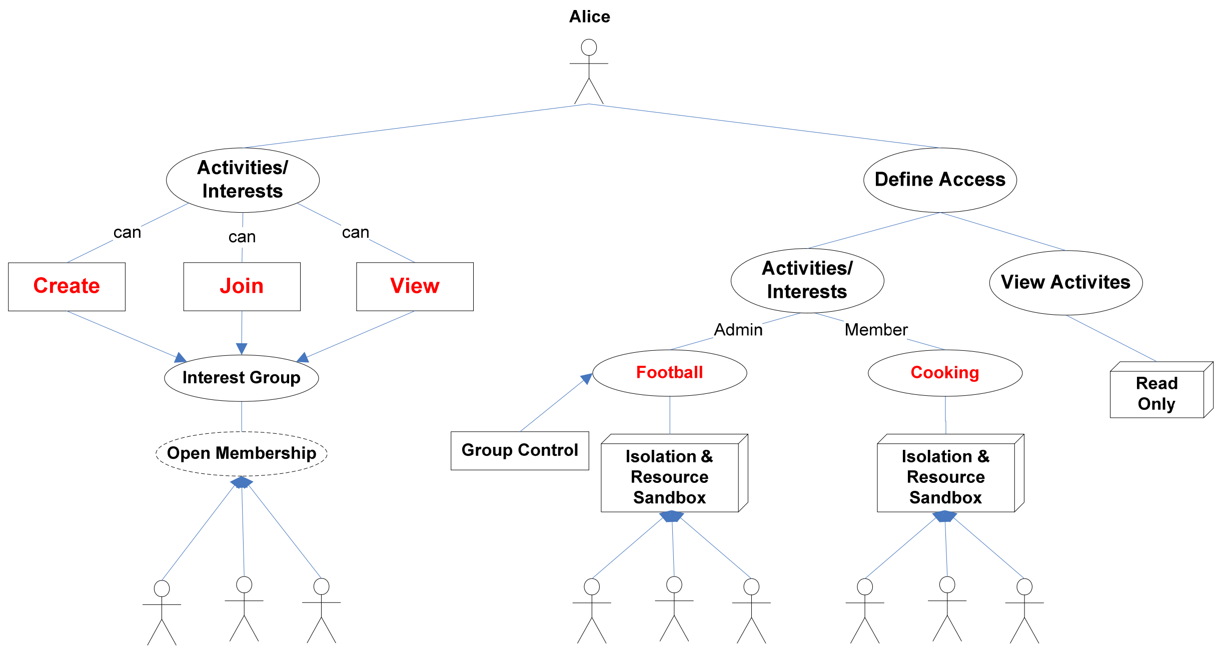


Figure 3-12: Interaction Pattern of Alice with Interest/Activity Groups

4 Safebook Access Control Model and Key Management

This chapter presents the details of Safebook's access control mode and examines how keys are managed. The first section describes the access layer. The second section introduces the concept of a profile hierarchy. The third section explains how we model this profile hierarchy. Section 0 describes an access hierarchy and key management. Section 4.5 describes Safebook's generic distribution process. The chapter concludes with a discussion of Safebook's social networking service abstractions.

4.1 Access Layer Orchestration

The Safebook access layer, as shown in Figure 4-1, is an orchestration of a number of service blocks centered on the key concept of a Profile Hierarchy. This profile hierarchy represents the logical arrangement of targeted user artifacts in a tree structure, which is used by the user interface to generate a complete user profile. The profile hierarchy represents the complete user profile in which artifacts are logically arranged. In social networks the user interface will likely only reveal a *subset* of a user's profile by linking dynamic access chains to the profile hierarchy. The resulting subset of a user's profile is called a profile dynamic view (PDV) and is based on the access rights of the viewer. The core function of the access layer is to maintain the user and their contact's PDVs using the services provided by the access layer.

The access layer exposes services via the Safebook API. These services can be used to generate a user profile from the PDV and by using social networking services abstractions to define the privacy and access control policies based on relationships, groups, and circles. Each of these building blocks is shown in Figure 4-1. These building blocks are each explained later in this chapter.

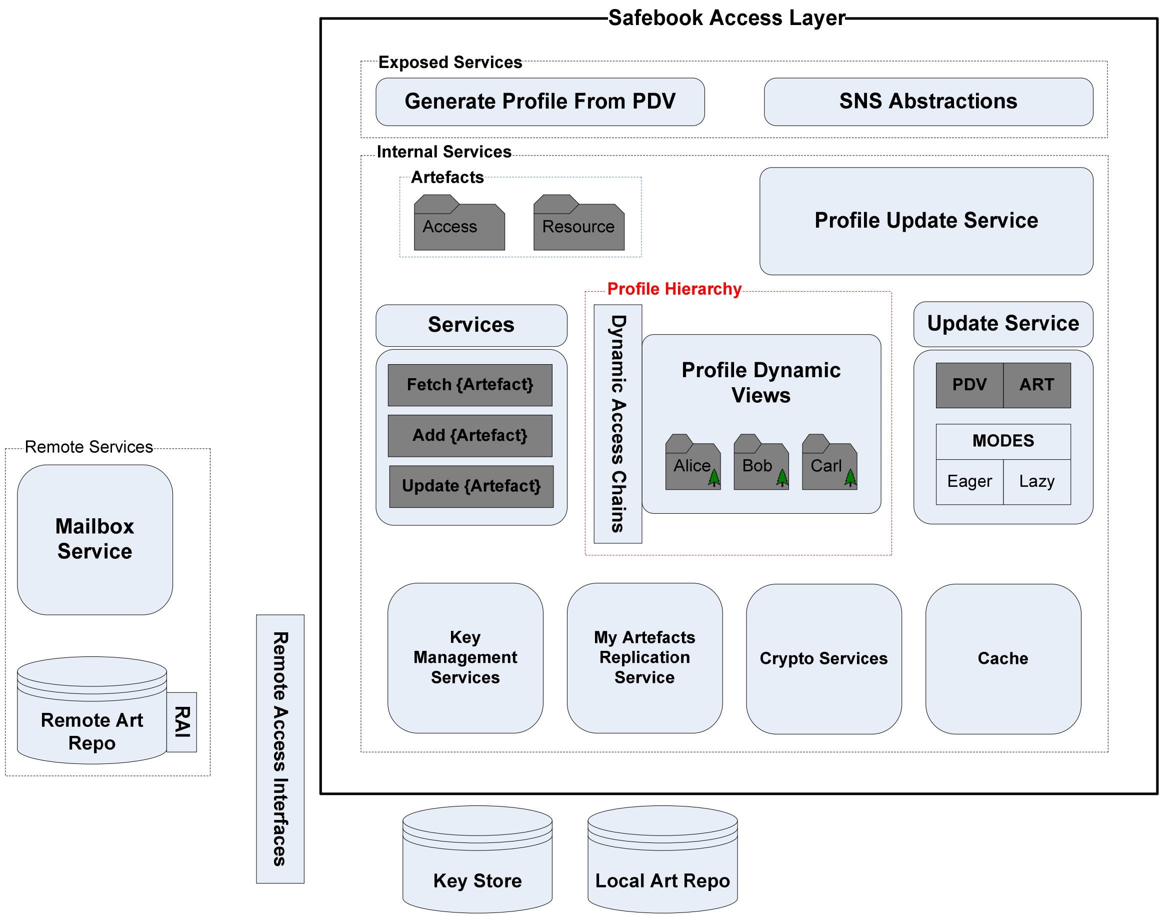


Figure 4-1: Safebook's Access Layer Orchestration

4.2 Profile Hierarchy

A profile hierarchy is the logical arrangement of user artifacts representing the natural flow of this user's profile. As illustrated earlier in Figure 3-3, a profile is a combination of many artifacts and each artifact is considered to be a fine grained logical social network entity for which access control can be defined. In other words, an artifact is the smallest unit in the Safebook access control scheme. These unique and standalone artifacts are raw material for the profile builder and they can be arranged or rearranged depending upon how the user wants their profile to look. A simplified template for a profile hierarchy is shown in Figure 4-2.

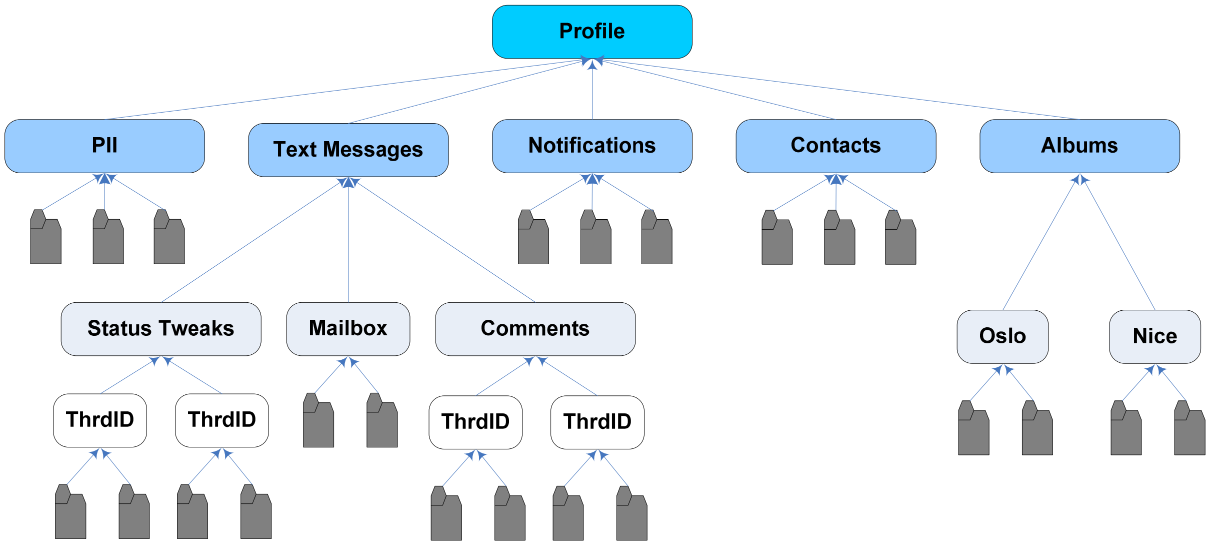


Figure 4-2: Profile Hierarchy Template for a User Profile

Figure 4-2 categorized the profile into five functional entities, i.e. personally identifiable information (PPI), text messages, notifications, contacts, and slbums. The profile builder uses this template to generate the user profile which will be rendered by a UI manager. The template itself is independent of any access control mechanisms and any vender can implement their own profile builder and use their own template. For simplicity, our model will consider the profile template shown in Figure 4-2.

The profile hierarchy entities can be classified into two categories:

1. Resource Entity

Labels Labels are those entities in the profile hierarchy which provides logical categorizations of the user profile data. For example, in Figure 4-2, the nodes Profile, PII, Notifications, etc. are labels and only provide a logical meaning within the profile hierarchy. These labels do not contain any user data and simply provide a skeleton for the user profile.

Data Data located at the leaves of the profile hierarchy graph and store the actual profile contents. These contents can be text messages, pictures, or other multimedia data. The data alone is not sufficient to generate the user profile, rather the profile builder uses the profile labels to generate the profile skeleton and then fills in the associated data element for each category.

2. Connectivity Entity

   Connectivity entities are the edges of the profile hierarchy graph, representing the association of each entity of the profile hierarchy with other entities. For example, the root entity Profile is linked with five other entities was shown in Figure 4-2.

4.3 Modeling Profile Hierarchy

The profile hierarchy is modeled as a unidirectional connected graph. The vertices of this graph represent resource entities, whereas the edges represent the association and connectivity of profile hierarchy as shown in Figure 4-3. The root node of the graph is the head of the profile hierarchy, binding all entities to a single entry point. This root node or profile entry point is the only entity which needs to be exposed for external access to this hierarchy graph.

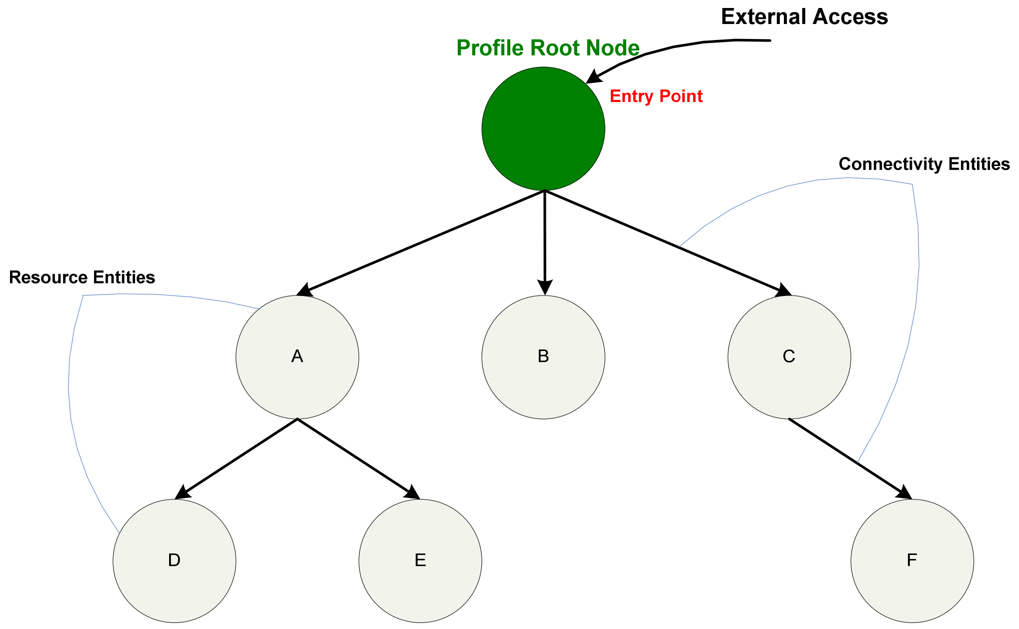


Figure 4-3: Profile Hierarchy Unidirectional Graph

4.3.1 System Entities & Representation

The profile hierarchy is a unidirectional graph and can be represented by an adjacency list based on two entities, i.e., resource entities and connectivity entities. An example of such an adjacency list is shown in shown in Figure 4-4.

Figure 4-4 presents the profile hierarchy defined in Figure 4-3 as an adjacency list. The vertices of the graph are placed in the Resource column and the edges associated with each vertex are presented in the Connectivity column. This adjacency list represents a unidirectional graph, thus Node A is linked with nodes D and E, but nodes D and E has no associated edges. The root node, shown as a green node is the head of the profile, and is the only entry point which needs to be exposed in order to fetch and generate a profile hierarchy.

This adjacency list will be used by the profile builder, to fetch the system's resource entities and to generate the profile hierarchy. Moreover, masking of the connectivity information will be exploited to generate a PDV as will be discussed later in this chapter.

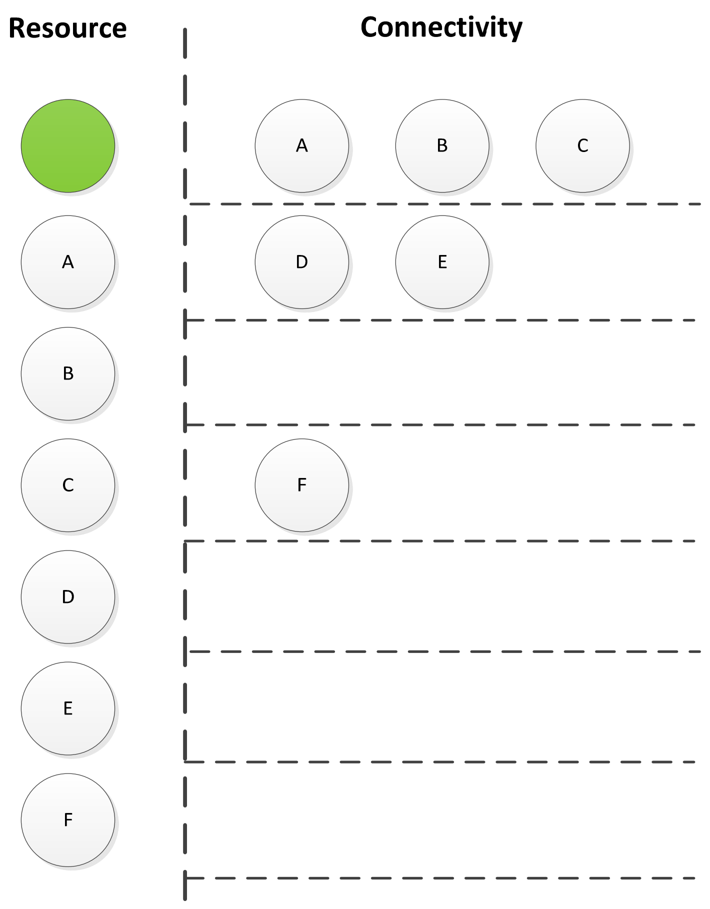


Figure 4-4: Adjacency List representing the profile hierarchy shown in Figure 4-3

4.3.2 Exporting a Profile Hierarchy to a Data Source

After defining the representation of the profile hierarchy, next we define how the external accesses this profile. In Safebook's P2P architecture, each node is considered as a potential data source, which is hosting the Matryoshka Core artifacts. In order to increase the availability, the artifacts are replicated over a number of mirror nodes (Figure 2-3) and the Safebook Communication privacy provides end-to-end and hope-by-hope protection, moreover it ensures anonymity, unobservability, untraceability, and unlinkability properties. There are two objectives to be achieved:

1. Replicate profile artifacts to increase their availability
2. Enforce access controls

The first objective is inherited from Safebook's design, thus user artifacts will be synchronized with copies places upon the mirror nodes. The second objective is addressed in this thesis. Unlike a centralized system, due to design in a P2P networks once data is replicated or downloaded by any other user, the owner of the data loses control over it. So, the mirror node cannot be used as an access controller for Safebook's core node. Therefore, access control must be ensured by the core node, while the mirror nodes are simply alternatively sources from which requestor can fetch the artifacts if granted access.

Figure 4-5 illustrates how user artifacts are exported to mirror nodes and then exposed to external world through a simple Fetch request.

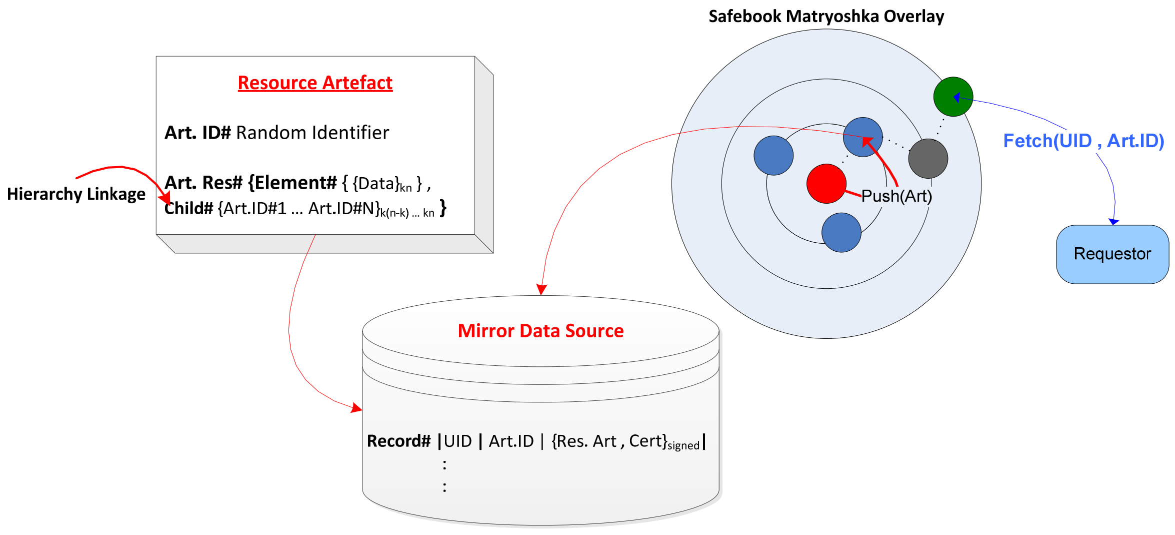

Safebook Matryoshka Overlay
Resource Artefact
Art. ID# Random Identifier
Art. Res# {Element# { {Data}kn } , Child# {Art.ID#1 … Art.ID#N}k(n-k) … kn }
Hierarchy Linkage
Push(Art)
Fetch(UID , Art.ID)
Requestor
Mirror Data Source
Record# |UID | Art.ID | {Res. Art , Cert}signed |

Figure 4-5: Exporting Profile Hierarchy Artifacts

A Resource Artifact is composed of two parts.

Art. ID	A random identifier to uniquely identify the artifact in user space.
Art. Res	An artifact resource is a tuple of ***Element*** and ***Access Chain*** *(AC)*. The *Element* contains the artifact *{Data}*$_{kn}$. The *Access Chain* contains the connectivity information and links to the next resource artifact in a profile hierarchy. Data is encrypted with a symmetric access key k_n, whereas the access chain is a stream cipher where an access key is generated from a set of symmetric access keys $K_{(n-k)}$ to K_n.

The mirror data sources are hosted on each mirror node of a Safebook Matryoshka Overlay. Each record in the data source consists of Safebook unique identifier (UID), an artifact ID, and a signed tuple of resource artifact and user certificate. When the UID and artifact ID are combined the result is a unique identifier in Safebook space. Moreover, the requestor can verify the artifact ID and UID from the signed resource artifact and user certificate respectively.

4.3.3 Profile Visibility

In an OSN it rare that the user wants to share all of his/her profile, other than those artifacts that they have explicitly made available via public groups and pages. It is more likely that a user reveals parts of his/her profile to different subsets of his/her contacts. So for any user in a Safebook OSN, a target profile is a derived subset (i.e., a view) of the actual profile. The visibility of the user's profile for a given requestor is defined by a *dynamic view*, which is determined by the access rights granted to this requestor by the user.

The PDV is controlled by revealing only a subset of edges of the profile hierarchy. The connectivity information can be tuned to hide branches of the artifact tree as appropriate for different users. Each user, based on the access that has been granted, will be able to access a subset of profile hierarchy. Figure 4-6 presents the dynamic views of Alice's profile with respect to Bob's and Carl's visibilities.

Alice's profile root has three child artifacts, with each child having different access rights. The PPI artifact is public; hence it is accessible to all the users. This PPI artifact has two child artifacts, where the Name artifact is public but the Avatar artifact is encrypted with access key KEY1.

Alice's profile is visible within the Safebook network with only its public information being visible; all of the other artifacts are only accessible to Alice's friends with whom she has shared access keys. Similarly, the Status and Album child artifacts of the root node are encrypted with access keys KEY1 and KEY2 respectively. The Album artifact has two child artifacts: Nice and Paris. The Paris artifact is using the inherited access from the Album artifact and is encrypted with same key KEY1, whereas the Nice artifact overrides this access with a new key KEY3.

Figure 4-6 shows how Alice's profile is logically represented in two different views for Bob and Carl. Bob has access key KEY1, whereas Carl has both KEY1 and KEY2. Bob can only access the Name (Public), Avatar (KEY1), and Status (KEY1) tweaks of Alice; while Carl can also access the Paris (KEY2) photo Album as well. Moreover, the Nice (KEY3) artifact is inaccessible to both Bob and Carl, as neither has KEY3.

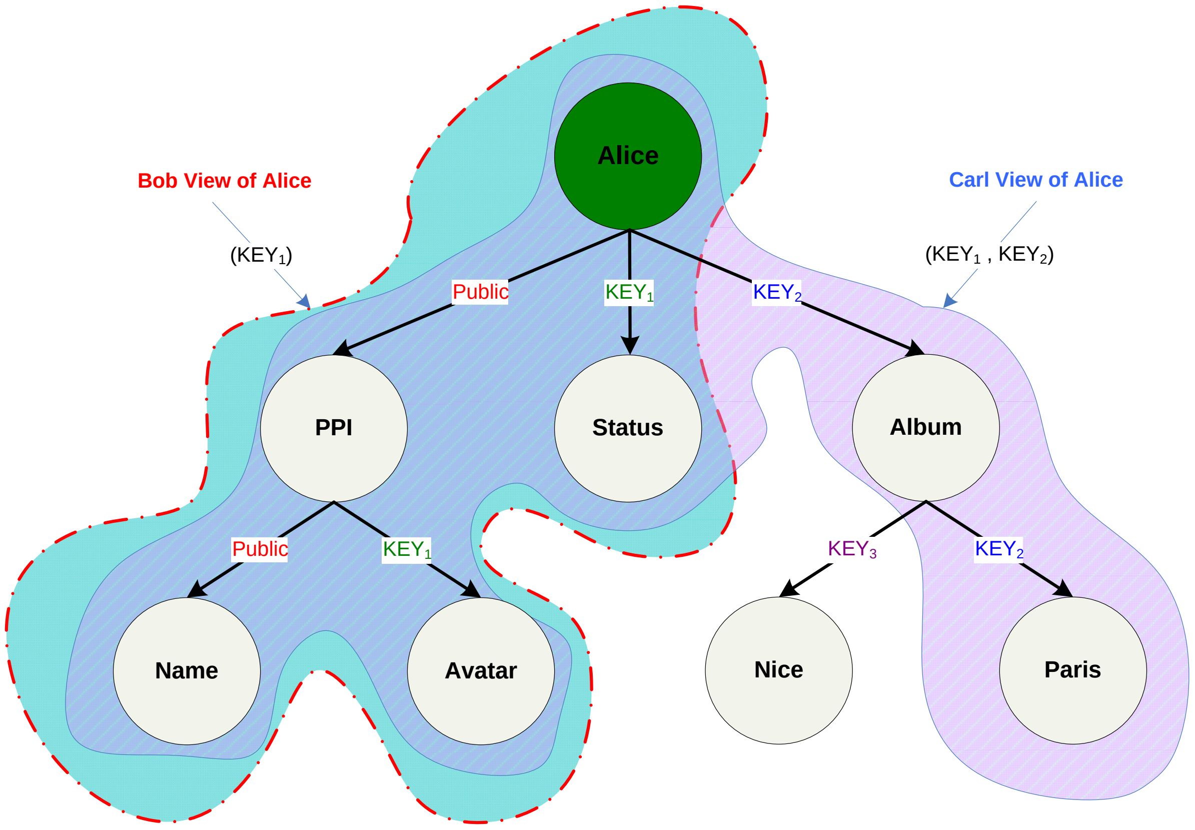


Figure 4-6: Logical Profile Dynamic View of Alice w.r.t Bob and Carl

In a Safebook network, each artifact is distributed autonomously with no dependence on other artifacts rather than being organized in terms of logical arrangement (as shown in Figure 4-6). The connectivity information is embedded in each artifact as described in Figure 4-4 and recursive profile retrieval is used to fetch the complete target profile. Each artifact in Figure 4-6 will be stored in the Safebook network as shown in Figure 4-7.

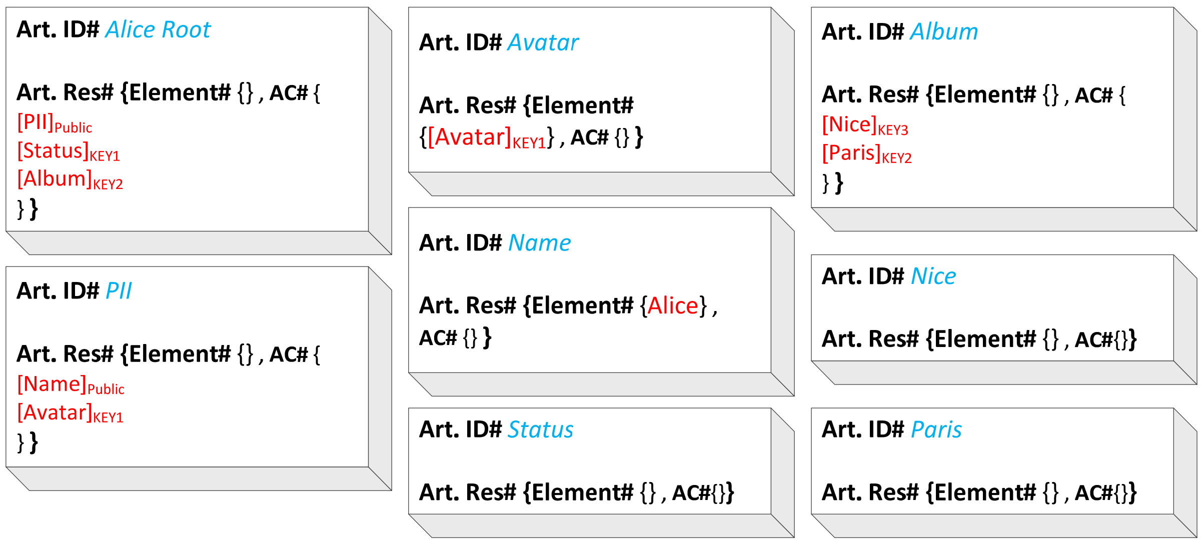

Art. ID# Alice Root
Art. Res# {Element# {} , AC# { [PII]Public [Status]KEY1 [Album]KEY2 } }
Art. ID# Avatar
Art. Res# {Element# {[Avatar]KEY1} , AC# {} }
Art. ID# Album
Art. Res# {Element# {} , AC# { [Nice]KEY3 [Paris]KEY2 } }
Art. ID# Name
Art. Res# {Element# {Alice} , AC# {} }
Art. ID# PII
Art. Res# {Element# {} , AC# { [Name]Public [Avatar]KEY1 } }
Art. ID# Nice
Art. Res# {Element# {} , AC#{}}
Art. ID# Status
Art. Res# {Element# {} , AC#{}}
Art. ID# Paris
Art. Res# {Element# {} , AC#{}}

Figure 4-7: Alice's artifacts with embedded connectivity links in a Safebook Network

Consider the root node whose artifact ID for the purposes of illustration in this discussion is "Alice Root", rather than a random identifier. An artifact resource consists of:

Element	In the case of the root node this element is empty in this example, but it may contain attributes and data related to the root node. The element data is encrypted with the symmetric access key of the artifact unless it is defined as public.
Access Chain	The access chain is an encrypted string which contains the identifiers of child artifacts. The string is created by concatenating the encrypted identifiers after they have been encrypted with their respective access keys.

4.3.3.1 Access Chain

The access chain is generated by using a stream cipher to dynamically create an encrypted string of each of the child artifact's identifiers with multiple access rights. Each child artifact identifier is encrypted with the respective access keystream and concatenated together with random padding. In this way a cipher string is generated, which is encrypted with multiple keystreams. The requestor can decrypt only those parts of the access chain which corresponds to a shared keystreams.

4.3.3.1.1 Creation of an access chain

The creation of an access chain depends on two parameters: an artifact identifier space and a symmetric access key space. The artifact identifier will be encrypted using a symmetric key by using a stream cipher approach. This symmetric key will be used to generate a keystream to encrypt the artifact identifier bit by bit. The keystream length should be at least equal to the artifact identifier, size but for security purposes, the keystream should have a long period, to avoid predictability and repetition. In Safebook, the artifact identifier space is defined as 2^{64}-1 and the symmetric key space is 2^{256}-1. Thus the length of each identifier and access key is 64 and 256 bits respectively. The creation of an access chain is shown in Figure 4-8.

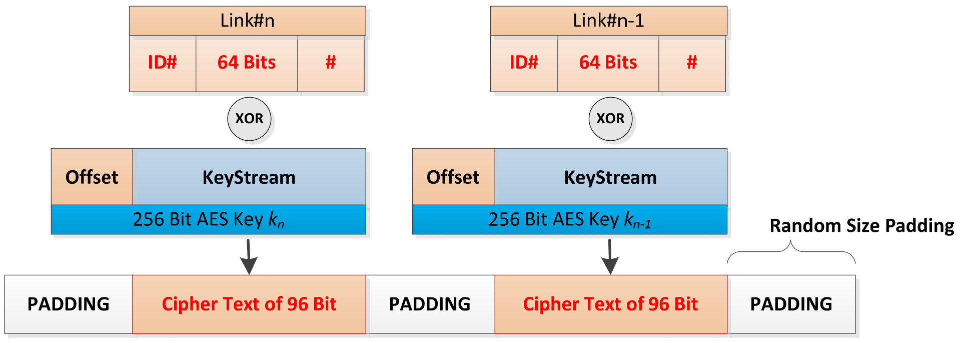


Figure 4-8: Creation of Access Chain

Random padding is added to avoid symmetry in the generated cipher string. The total length of each link is:

$$Length_{Link}(\{ID\# + 64\ bits\ Identifier + \#\}) = 24 + 64 + 8 \rightarrow 96\ Bits$$

The respective keystream to encrypt each link is:

$$KeyStream_{k_n} = k_n[Offset : (Offset + Len(Link)]$$

Where k_n is the AES 256 bit key and offset is:

$$Offset = Rand(seed)\%(Length_{k_n} - Length_{Link})$$

The cipher text is generated by XOR the corresponding bits of the link and the generated keyStream. The complete chain is created by concatenating all the cipher text generated from the encryption of links with their respective access keys and additional random sized padding.

$$\boldsymbol{Access\ Chain} = \boldsymbol{ConcatWithRandPadding}(\{Link_1\}_{k_1}, \{Link_2\}_{k_2}, \dots, \{Link_n\}_{k_n})$$

4.3.3.1.2 Decryption

The access chain is a string of binary bits, which embeds the information regarding the child artifact links. The links are extracted by using a stream cipher and the process is shown in Figure 4-9.

All the access keys which are associated with the target profile are concatenated to create the keystream for decrypting the embedded links in the access chain. The generated keystream is given as input to a shift register and a bit by bit XOR operation is done. Once the encryption key is exactly in phase with the cipher text, the XOR operation will emit the embedded link, otherwise a random stream of bits are generated. In this way, only those links will be revealed which corresponds to the keys shared with the target user, all of the other links will remain hidden in the access chain.

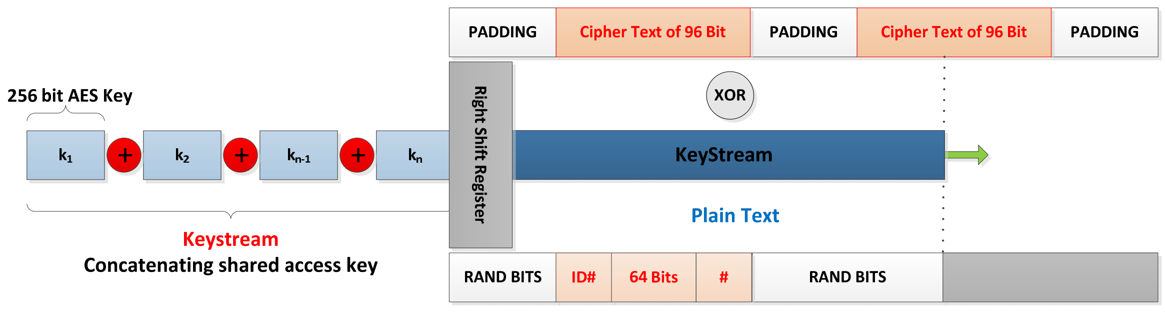


Figure 4-9: Access Chain Decryption Process

4.3.4 Decoupling an Artifact's Element from Connectivity

The resource artifact embeds the connectivity information of the profile hierarchy, thus an artifact has dependency on other artifacts. This raises the following concerns regarding the need to decouple the connectivity information from the artifact element:

Tight Coupling An artifact is tightly coupled with its connectivity information. This means, if slight change in connectivity is made, then whole artifact needs to be updated in the Safebook network and this will be a costly operation.

Hierarchy Update The update process is running continuously to stay up to date with the user profiles. If a complete artifact is downloaded only to fetch the connectivity links, this will add a lot of overhead and the update operation will be very costly. By decoupling the connectivity from the artifacts we will only need to update the connectivity information and will only need to download new artifacts or modified ones, thus reducing the overhead.

Security For security reasons, keeping both the artifact and its connectivity information together is not secure. If a key for an artifact is compromise, then the connectivity information is revealed which may expose further artifacts.

Artifacts can be categorized into two categories based on the information they hold. These two types of artifacts are:

Resource artifact This type of artifact holds element data, which can be public or encrypted with a symmetric key taken from the resource key space.

Access artifact An access artifact holds the connectivity information associated with a resource artifact. The access identifier is same as the resource artifact identifier to which it corresponds, thus there is one-to-one mapping between resource and access artifacts.

Figure 4-10 presents the decoupled environment, where each artifact in the resource space is represented with an artifact with same identifier in the access space. In order to secure the system and completely decoupling the resource and access space, the symmetric access keys in the resource space are not reused in the access space. In this way if an access key is compromised there is no impact on the resource space and vice versa. Moreover mapping between the keys occurs via a Safebook secure mailbox or Safebook's friendship request protocol.

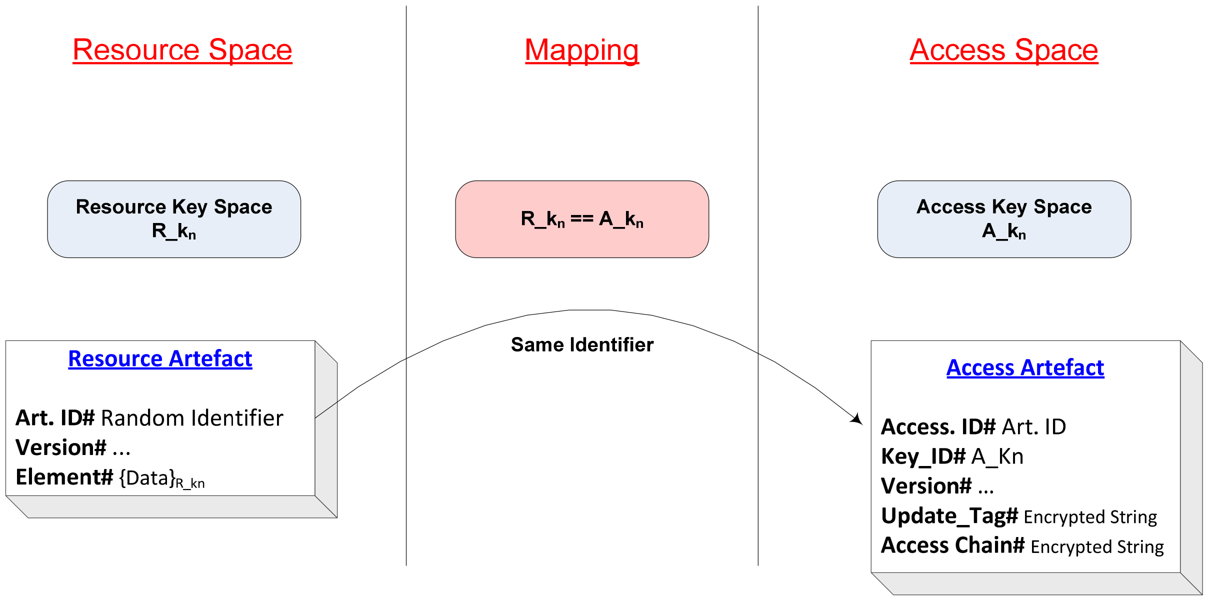


Figure 4-10: Decoupled environment for resource and access control

The links in the access chains are encrypted with the symmetric keys A_k$_n$ from the access key space, whereas the element in the resource artifact is encrypted with a key R_k$_n$ from the resource key space. This gives greater flexibility as changing the access keys in the access hierarchy will not impact the resource space. The artifacts in the access space do not contain user data such as pictures, messages, multimedia etc. rather they only contain one access chain string and are relatively small in size, so modification of the access hierarchy are not as costly as would be required when re-encrypting data in the resource space.

4.4 Access Hierarchy & Key Management

In this section we will examine the access hierarchy and how keys are managed, distributed, and revoked.

4.4.1 Types of Access Keys

In the Safebook system, there are three types of access keys: public key infrastructure asymmetric key pairs, symmetric resource keys, and symmetric access keys.

4.4.1.1 *Public key infrastructure asymmetric key pairs*

For each user in the Safebook system, there are two asymmetric key pairs which correspond to these two spaces:

User Space / UID	The user space represents the user in the social interaction layer. All of the protocols in this layer use this key pair.
P2P Space / NID	A separate PKI key pair is used in the underlying P2P layer which provides lookup and other services.

4.4.1.2 *Symmetric Resource Keys (R_k_n)*

The element data of a resource artifact is encrypted with AES using a 256 bit symmetric key. A resource key is represented with the notion: R_k_n. A resource key has a one-to-many relationship with an artifact and access keys as shown in Figure 4-11.

4.4.1.3 *Symmetric Access Keys (A_k_n)*

The symmetric keys which are generated in the access space to represent the access hierarchy of the profile are symmetric access keys and they are represented with the notion: A_K_n. The access key has a one-to-one mapping with a resource key and a one-to-many relationship with the artifact as shown in Figure 4-11.

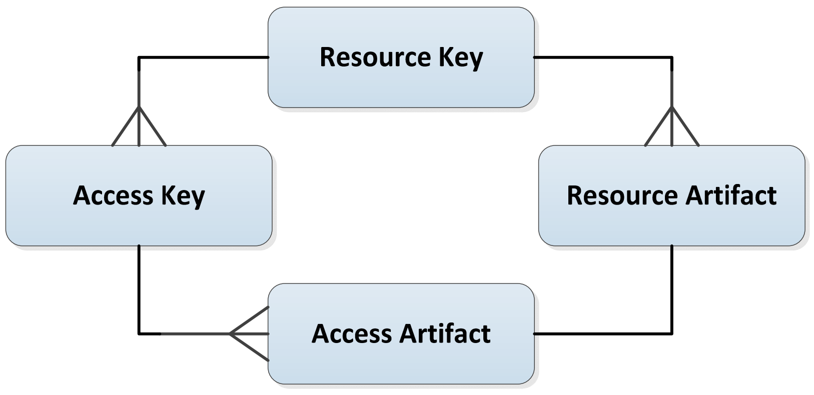


Figure 4-11: Relationship between the system entities

4.4.2 Granularity of Access Control

The granularity of access control over a user profile in the Safebook system is defined over artifacts in the profile hierarchy. As noted earlier an artifact in the Safebook network represents the smallest meaningful unit of information. For example, the name of a user can be defined as a separate artifact for which access control can be defined. So, the design by default provides the possibility to define access control over every piece of information in the user profile.

4.4.3 Access Hierarchy

The access hierarchy is a reflection of the profile hierarchy's connectivity information. Each resource artifact is represented by an access artifact. The access artifact contains all the child artifacts' links. The access chains are generated using an access key to encrypt the information. The mappings between the access and resource keys are shared with the users with whom a user wishes to provide access to specific artifacts. The access hierarchy looks similar to the one explained in Figure 4-6.

4.4.3.1 Objectives

The access hierarchy has the following objectives:

- To easily reflect changes in the access hierarchy for remote requestors.
- To manage the profile's organization and access rights through the access hierarchy.
- To ensure consistency of the access hierarchy and enable fine grained granularity of access rights.

4.4.3.2 Access Hierarchy Update Process

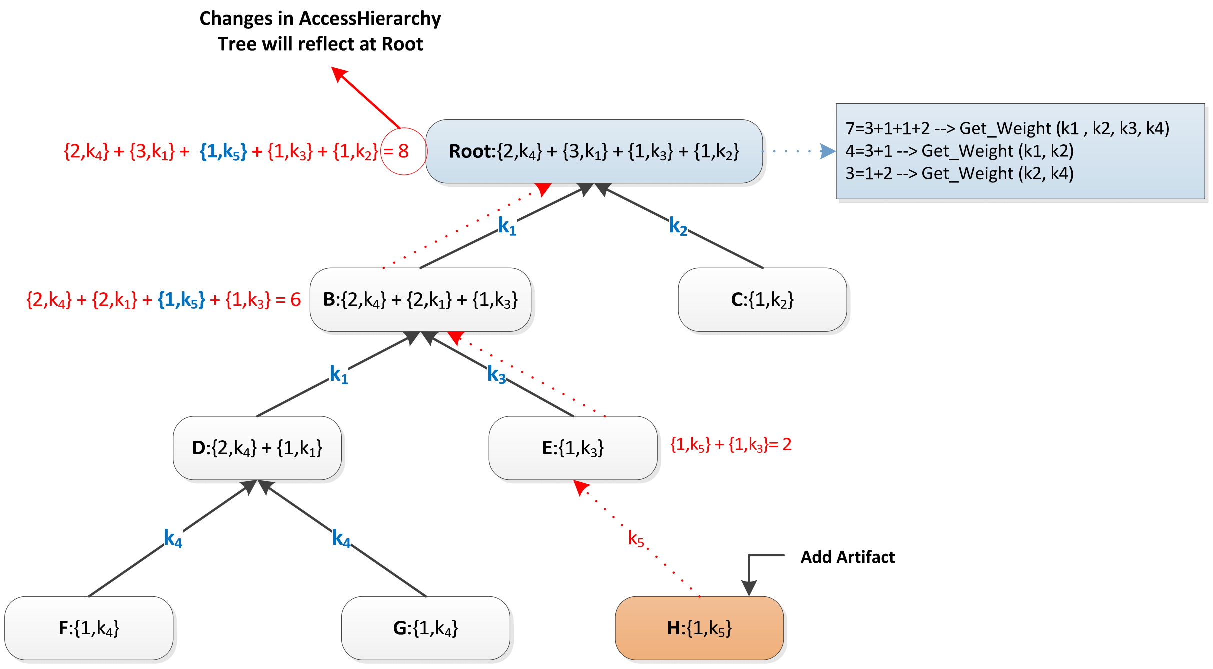
The users in a Safebook network needs to maintain synchronization with their contact's profile access hierarchy in order to detect and act based upon changes in this profile. Normally, changes in the hierarchy are hidden and only revealed once the complete access hierarchy is updated. The update process is run periodically. As this operation is very costly each update of the complete hierarchy will consume lots of network resources. An easier approach is to detect a change before further traversing the hierarchy.

The update process introduces the following new elements in the resource and access artifacts:

Version Label	The value of the version label is a random number assigned to each artifact when it is created. Both the resource and access artifacts share the same version as the identifier. This label is incremented by a random number each time the artifact is modified.
Weight	The weight of an access artifact is the sum of the artifact version and the versions of its children. This is a dynamic value and depends on the provided access keys. As shown in Figure 4-12 only the child links corresponding to provided keys will be included in the formation of an artifact weight. Moreover, weight is a vector where each element of the vector represents a key, whereas, the value is computed by summing all the child artifacts' versions, encrypted with the indexed key.

Update Tag An update tag is an encrypted string based on the concept of access chaining. This tag holds the information regarding an artifact and its child artifacts' weights. Each weight is encrypted with the respective access key using the stream cipher and concatenated. The creation and decryption process is same as explained in Figure 4-8 and Figure 4-9.

Figure 4-12 represents the access hierarchy update process and the red color represents the impacts following the insertion of a new artifact H in the access hierarchy. For simplicity all the artifacts in this example were assigned a default version value of 1. The new artifact has a initial version value of 1 and its weight is also 1 because it has no child artifacts. Artifact H is added as a child of artifact E with an access key k5. Artifact E re-commutes its weight as *[{1,k5},1] = 2* and the update tag will be changed accordingly. Artifact E will propagate these changes to its parent artifact B. Similarly, artifact B will also re-compute its weight and update tag and inform its predecessor. In this way the changes will eventually be reflected at the root and the root weight will be incremented from 7 to 8. In this way, a remote user can detect access hierarchy changes by comparing the weight of its target profile root with the newly fetched root weight.



Changes in AccessHierarchy Tree will reflect at Root
$\{2,k_4\} + \{3,k_1\} + \{1,k_5\} + \{1,k_3\} + \{1,k_2\} = 8$
Root:$\{2,k_4\} + \{3,k_1\} + \{1,k_3\} + \{1,k_2\}$
7=3+1+1+2 --> Get_Weight (k1 , k2, k3, k4)
4=3+1 --> Get_Weight (k1, k2)
3=1+2 --> Get_Weight (k2, k4)
k_1
k_2
$\{2,k_4\} + \{2,k_1\} + \{1,k_5\} + \{1,k_3\} = 6$
B:$\{2,k_4\} + \{2,k_1\} + \{1,k_3\}$
C:$\{1,k_2\}$
k_1
k_3
D:$\{2,k_4\} + \{1,k_1\}$
E:$\{1,k_3\}$
$\{1,k_5\} + \{1,k_3\} = 2$
k_4
k_4
k_5
Add Artifact
F:$\{1,k_4\}$
G:$\{1,k_4\}$
H:$\{1,k_5\}$

Figure 4-12: Access Hierarchy Update Process

4.4.3.3 Formation of an Access Hierarchy

The formation of an access hierarchy needs to handle the following cases:

- Adding a root artifact
- Adding a resource artifact
- Modify a resource artifact
- Deleting a resource artifact
- Changing access rights

All these cases are presented in the flowcharts shown in Figure 4-13 to Figure 4-18.

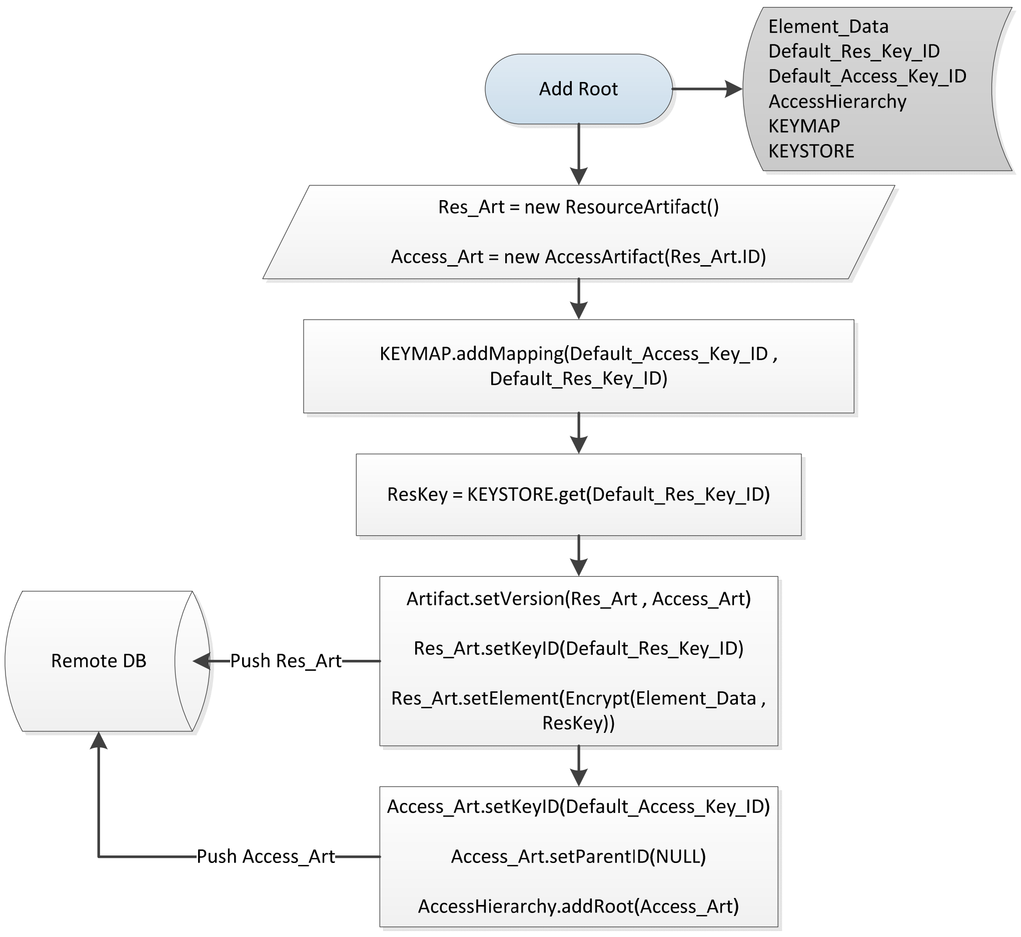

Add Root
Element_Data
Default_Res_Key_ID
Default_Access_Key_ID
AccessHierarchy
KEYMAP
KEYSTORE
Res_Art = new ResourceArtifact()
Access_Art = new AccessArtifact(Res_Art.ID)
KEYMAP.addMapping(Default_Access_Key_ID , Default_Res_Key_ID)
ResKey = KEYSTORE.get(Default_Res_Key_ID)
Artifact.setVersion(Res_Art , Access_Art)
Res_Art.setKeyID(Default_Res_Key_ID)
Res_Art.setElement(Encrypt(Element_Data , ResKey))
Remote DB
Push Res_Art
Access_Art.setKeyID(Default_Access_Key_ID)
Access_Art.setParentID(NULL)
AccessHierarchy.addRoot(Access_Art)
Push Access_Art

Figure 4-13: Adding a root artifact

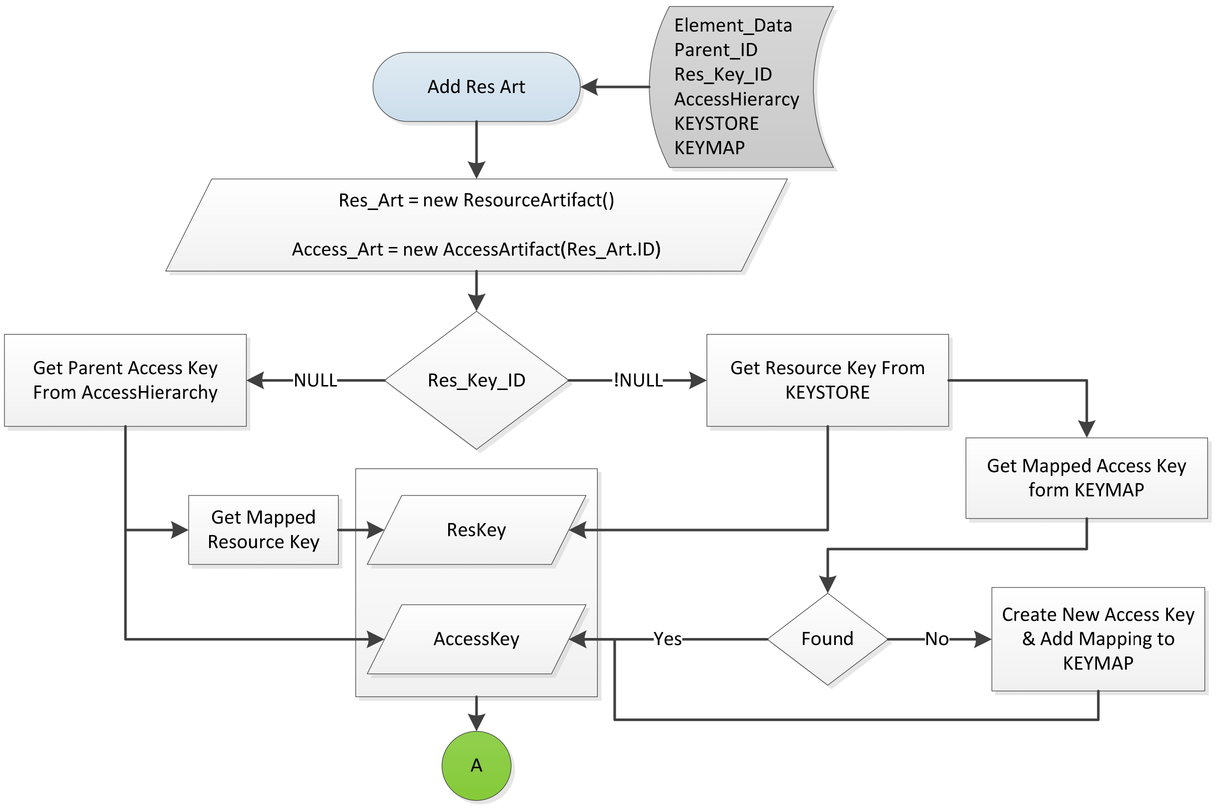

Element_Data
Parent_ID
Res_Key_ID
AccessHierarcy
KEYSTORE
KEYMAP
Add Res Art
Res_Art = new ResourceArtifact()
Access_Art = new AccessArtifact(Res_Art.ID)
Get Parent Access Key From AccessHierarchy
NULL
Res_Key_ID
!NULL
Get Resource Key From KEYSTORE
Get Mapped Access Key form KEYMAP
Get Mapped Resource Key
ResKey
AccessKey
Yes
Found
No
Create New Access Key & Add Mapping to KEYMAP
A

Figure 4-14: Adding a resource artifact

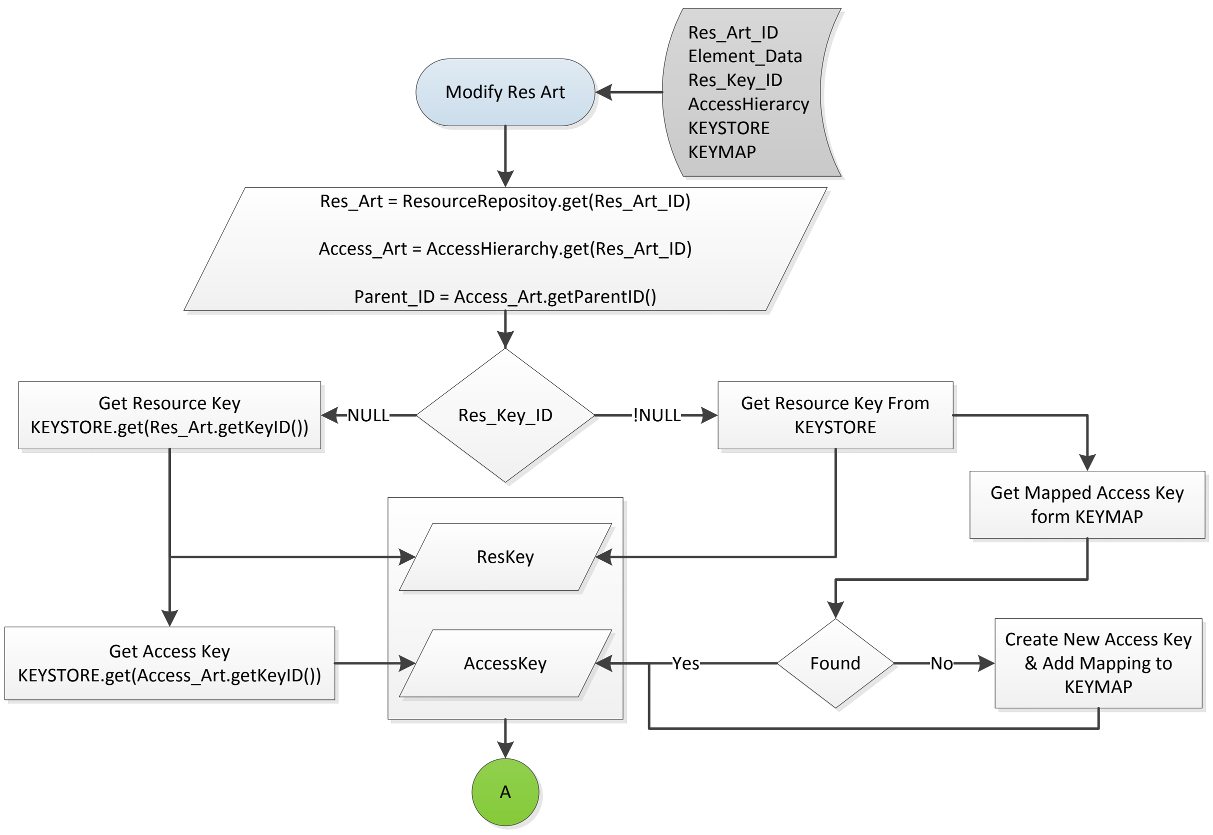

Res_Art_ID
Element_Data
Res_Key_ID
AccessHierarcy
KEYSTORE
KEYMAP
Modify Res Art
Res_Art = ResourceRepositoy.get(Res_Art_ID)
Access_Art = AccessHierarchy.get(Res_Art_ID)
Parent_ID = Access_Art.getParentID()
Get Resource Key KEYSTORE.get(Res_Art.getKeyID())
NULL
Res_Key_ID
!NULL
Get Resource Key From KEYSTORE
Get Mapped Access Key form KEYMAP
ResKey
Get Access Key KEYSTORE.get(Access_Art.getKeyID())
AccessKey
Yes
Found
No
Create New Access Key & Add Mapping to KEYMAP
A

Figure 4-15: Modifying a resource artifact

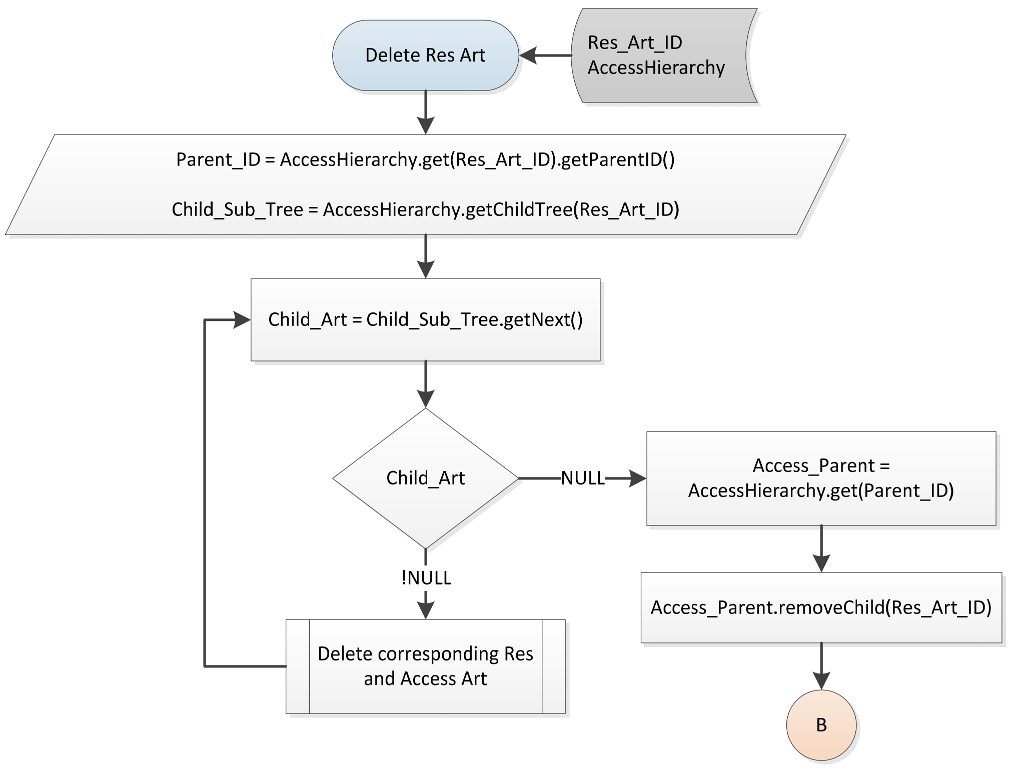

Delete Res Art
Res_Art_ID
AccessHierarchy
Parent_ID = AccessHierarchy.get(Res_Art_ID).getParentID()
Child_Sub_Tree = AccessHierarchy.getChildTree(Res_Art_ID)
Child_Art = Child_Sub_Tree.getNext()
Child_Art
NULL
Access_Parent = AccessHierarchy.get(Parent_ID)
!NULL
Access_Parent.removeChild(Res_Art_ID)
Delete corresponding Res and Access Art
B

Figure 4-16: Deleting a resource artifact

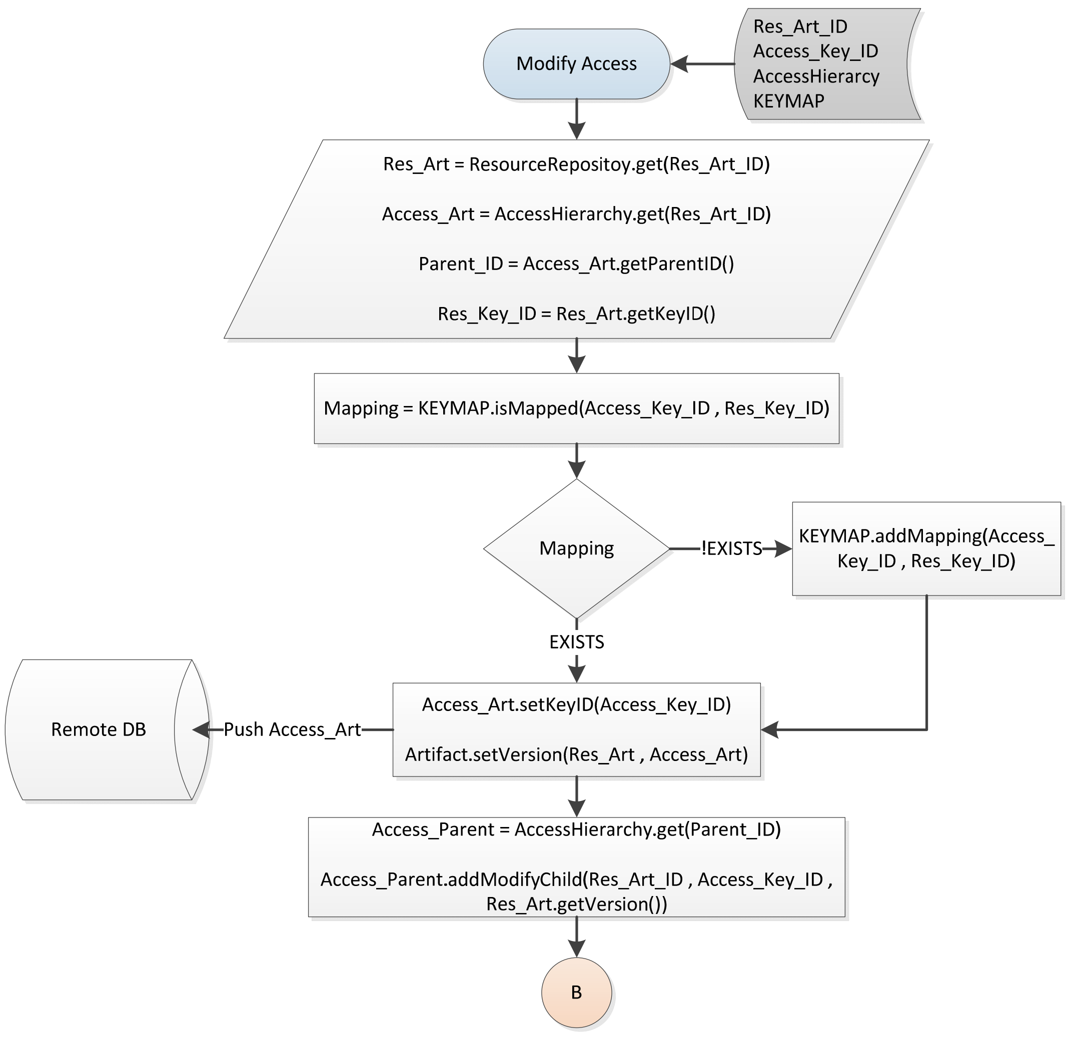

Res_Art_ID
Access_Key_ID
AccessHierarcy
KEYMAP
Modify Access
Res_Art = ResourceRepositoy.get(Res_Art_ID)
Access_Art = AccessHierarchy.get(Res_Art_ID)
Parent_ID = Access_Art.getParentID()
Res_Key_ID = Res_Art.getKeyID()
Mapping = KEYMAP.isMapped(Access_Key_ID , Res_Key_ID)
Mapping
!EXISTS
KEYMAP.addMapping(Access_Key_ID , Res_Key_ID)
EXISTS
Remote DB
Push Access_Art
Access_Art.setKeyID(Access_Key_ID)
Artifact.setVersion(Res_Art , Access_Art)
Access_Parent = AccessHierarchy.get(Parent_ID)
Access_Parent.addModifyChild(Res_Art_ID , Access_Key_ID , Res_Art.getVersion())
B

Figure 4-17: Modifying access rights

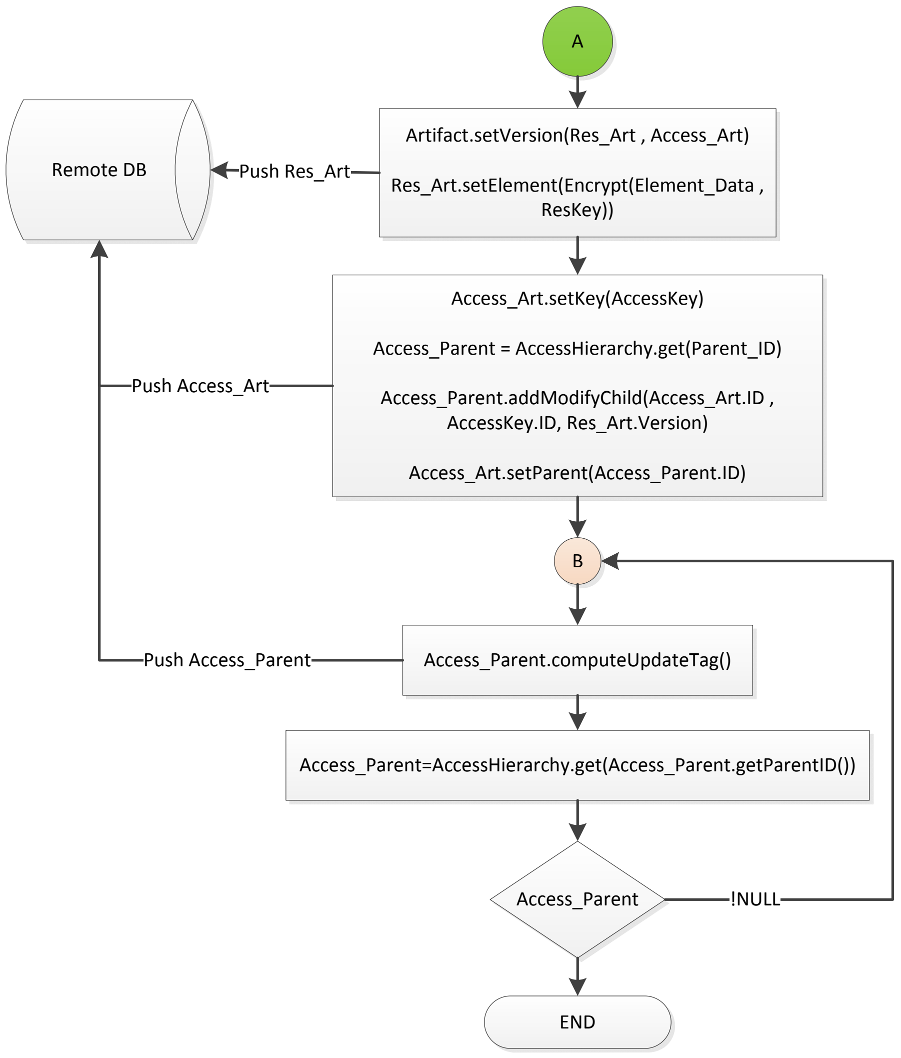

A
Artifact.setVersion(Res_Art , Access_Art)
Res_Art.setElement(Encrypt(Element_Data , ResKey))
Remote DB
Push Res_Art
Access_Art.setKey(AccessKey)
Access_Parent = AccessHierarchy.get(Parent_ID)
Access_Parent.addModifyChild(Access_Art.ID , AccessKey.ID, Res_Art.Version)
Access_Art.setParent(Access_Parent.ID)
Push Access_Art
B
Push Access_Parent
Access_Parent.computeUpdateTag()
Access_Parent=AccessHierarchy.get(Access_Parent.getParentID())
Access_Parent
!NULL
END

Figure 4-18: Post processing

4.4.3.4 Consistency Checks

The following rules need to be observed in order to keep the access hierarchy consistent:

- The access key defined at the root will be propagated to the child artifacts unless it is overridden by a new key.
- If a key B at Nth level of access hierarchy has an access dependency on upper hierarchy keys, e.g. A, then revealing key A should not reveal any other artifact in the hierarchy. For example, in Figure 4-6, the artifact *Nice* which is encrypted with KEY3 has a dependency on KEY2. If KEY2 is not shared with the requestor, then the artifact *Nice* will be lost. If KEY2 is shared, then the requestor can also access the artifact *Paris*, which is not desired. In order to break this connectivity, the Album encryption key will be changed to KEY4.

4.4.4 Distribution

The access keys are distributed through two processes:

Friendship Request Service	A friendship request service is provided by Safebook *[1]*. During the friendship handshaking protocol a number of access keys can be exchanged between both parties. The integrity and confidentiality of this exchange is ensured by the communication protocol.
Mailbox Service	Safebook also provides a mailbox service to exchange encrypted messages. A shared key is exchanged during the Friendship Request Protocol, which is use to encrypt the message, the message is then encrypted with target public key and signed with the sender private key. This ensures the authenticity, confidentiality and integrity of the message. The mailbox service is used for the following purposes: • To distribute mappings between the resource and access keys. • To distribute symmetric resource and access keys.

4.4.5 Key Revocation

In any cryptographic system the cost of key revocation is an important metric to measure. The Safebook access control scheme has been designed to make the cost of key revocation as low as possible. If any access key is compromised, and a key needs to be revoked, this is usually followed by re-encrypting all the affected data. If the data is large, such as a photo album then the re-encryption cost will be high. In a social network changes in access control are expected to be very frequently and this will result in frequent key revocations. For example, removing a user from a membership group will require revoking the group keys. In order to reduce the cost of re-encryption and key distribution a compromise is made based on the inherent limitations of P2P networks.

4.4.5.1 Limitations of P2P System

Some of the limitations of P2P systems are:

1. The fairness of a remote access controller cannot be guaranteed.
2. *For this reason the Safebook uses mirror nodes only as data source to increase availability, not to act as access controllers.*
3. To increase data availability, data needs to be replicated.
4. The owner losses control over their data once the data is distributed in the network.

4.4.5.2 Key Revocation Scheme

Unlike a centralized system, if a key is revoked in a P2P network all the affected artifacts have already been distributed hence access to them cannot be revoked. Therefore we can only protect data from *future* access by changing and revoking the key. The Safebook key revocation scheme is based on the structure of the access hierarchy. The resource and access artifacts are located in two separate spaces. Examining the access hierarchy reveals nothing about the resource artifacts. Moreover, if an access key or resource key is changed, it does not affect the other. If a resource key is compromised it will affect only one artifact, similarly if an access artifact is compromised, it will reveal the identifiers only of those child artifacts which inherit the same key. Knowing identifiers is not enough to fetch the corresponding resource artifact as these artifacts are encrypted with a resource key and the mappings are only shared with the desired contacts through trusted channels. The revocation scheme is shown in Figure 4-19 and Figure 4-20.

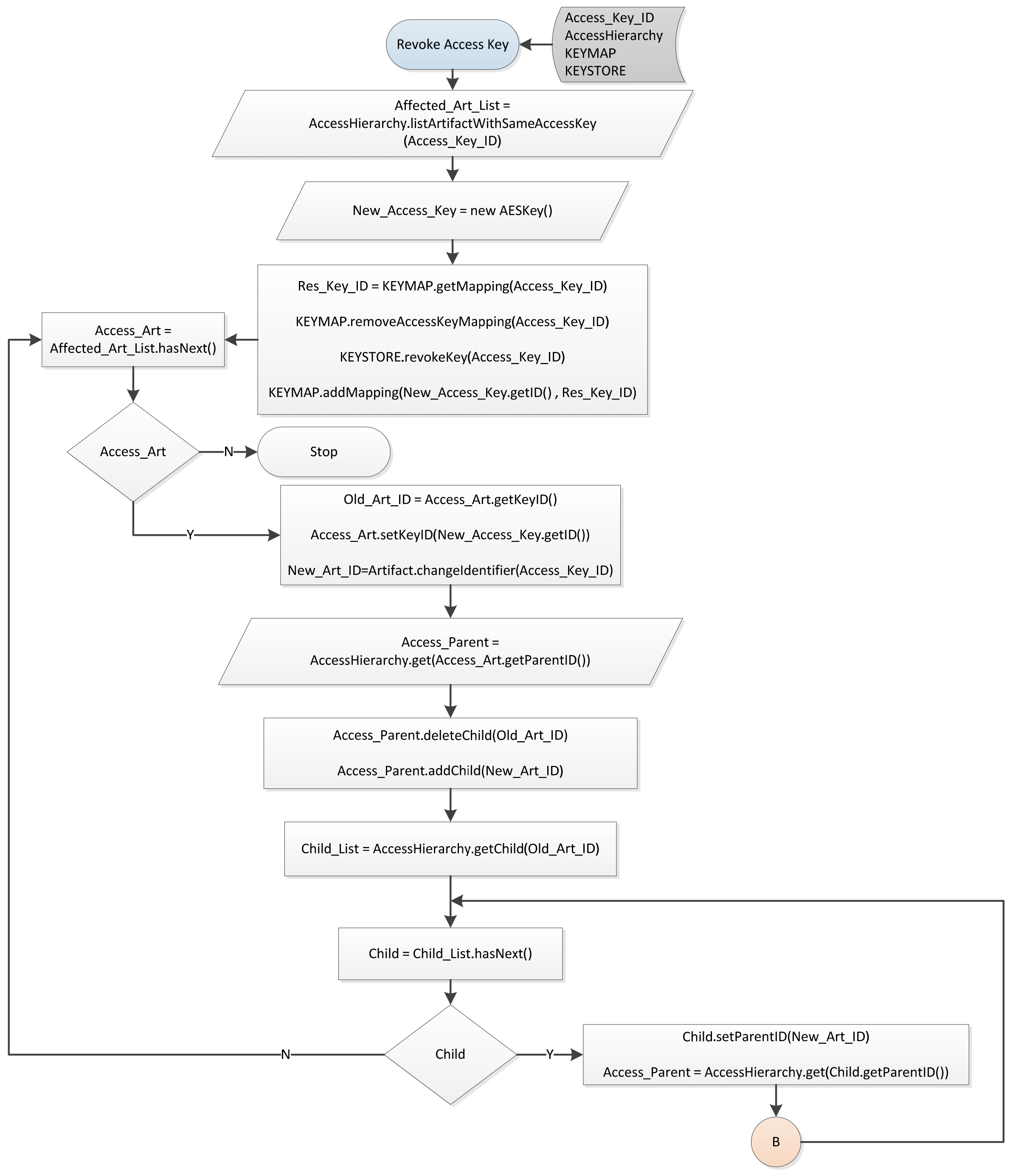

Revoke Access Key
Access_Key_ID
AccessHierarchy
KEYMAP
KEYSTORE
Affected_Art_List = AccessHierarchy.listArtifactWithSameAccessKey (Access_Key_ID)
New_Access_Key = new AESKey()
Res_Key_ID = KEYMAP.getMapping(Access_Key_ID)
KEYMAP.removeAccessKeyMapping(Access_Key_ID)
KEYSTORE.revokeKey(Access_Key_ID)
KEYMAP.addMapping(New_Access_Key.getID() , Res_Key_ID)
Access_Art = Affected_Art_List.hasNext()
Access_Art
N
Stop
Y
Old_Art_ID = Access_Art.getKeyID()
Access_Art.setKeyID(New_Access_Key.getID())
New_Art_ID=Artifact.changeIdentifier(Access_Key_ID)
Access_Parent = AccessHierarchy.get(Access_Art.getParentID())
Access_Parent.deleteChild(Old_Art_ID)
Access_Parent.addChild(New_Art_ID)
Child_List = AccessHierarchy.getChild(Old_Art_ID)
Child = Child_List.hasNext()
Child
N
Y
Child.setParentID(New_Art_ID)
Access_Parent = AccessHierarchy.get(Child.getParentID())
B

Figure 4-19: Access key revocation scheme

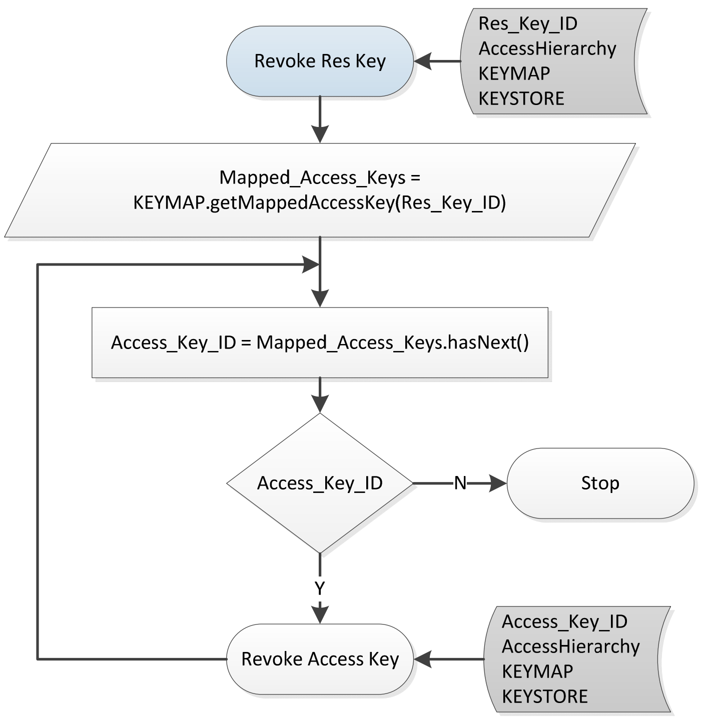


Figure 4-20: Resource key revocation scheme

When any key is revoked, there are two steps:

1. Change the affected access keys

   The affected access keys are changed and all the links in the affected artifacts of the access hierarchy are modified accordingly and then redistributed to the mirror nodes. The access artifacts are small in size and thus re-encryption and redistribution are not as costly as distributing complete resource artifacts.

2. Change the affected artifacts' identifiers

   In the first step the access keys are changed and thus users with the old access key can no longer see the affected artifacts in the hierarchy, but still they have the identifiers and they can fetch the resource artifact and decrypt it with the resource access key. In order to break this connection, the identifiers are changed, in this way the there is no connection to the old artifacts and the identifiers are selected randomly from a large address space, so the probability of guessing correctly is very low. The actual data is not re-encrypted, but the identifiers are changed locally and on the remote mirror nodes.

4.5 Safebook Generic Distribution Process

The Safebook generic distribution process has the following four steps: publish the profile root, find the user, distribute the access key, and push the artifact. After this an interested party can fetch an artifact or retrieve the profile. Finally in this section we describe how a profile can be updated.

4.5.1 Publish a Profile Root

The profile hierarchy is made accessible in a Safebook network by publishing the profile's root node. The Safebook API publishes the name vector (for visibility in a Safebook network) and Matryoshka chain entry points once the Matryoshka overlay is created. The profile root node is published with the Name Vector using the same service, as:

$$Publish_{Profile_{Root}} = Publish(\{ProfileRootArtID\ , NameVector\}, EP)$$

4.5.2 Find User

Safebook provides a Lookup service to find users. Users in a Safebook network are visible through a published name vector, which contains information, such as name, email address, country, etc. The lookup service takes any element as a parameter and returns a list of matching Entry Point Lists (EPLs) along with $ProfileRootArtID$ and $NameVector$.

4.5.3 Distribute Access Key

The access keys are exchanged either via the friendship request protocol or with the *PublishKey* service using the Safebook Mailbox service.

4.5.4 Push Artifact

The users artifacts are pushed into the Safebook Network by the Matryoshka replication service.

4.5.5 Fetch Artifact

The fetch request is usually followed by an EP lookup request. Once the user has the EPL, the fetch request can be generated to find and download the artifact. The EPL is automatically taken from the P2P overlay and the higher layers do not explicitly provide the EPL.

$$ResourceArtifact = FetchArt(UID, Art.\ ID, EPL)$$

4.5.6 Profile Retrieval

The user profile can be retrieved recursively by traversing the profile hierarchy. The Profile ROOT_ID is published with the EP list. So any Safebook user can look up this information and download the Profile Root Artifact. This node contains some public information which is accessible to all users and encrypted information only accessible by friends or members of the user's contacts list. The symmetric access keys are shared either via the friendship or PublishKey protocols. Only a user with a symmetric shared key can decrypt the artifact resource, containing a list of child artifacts' IDs and user data. The recursive process to retrieve the complete profile is shown in Figure 4-21.

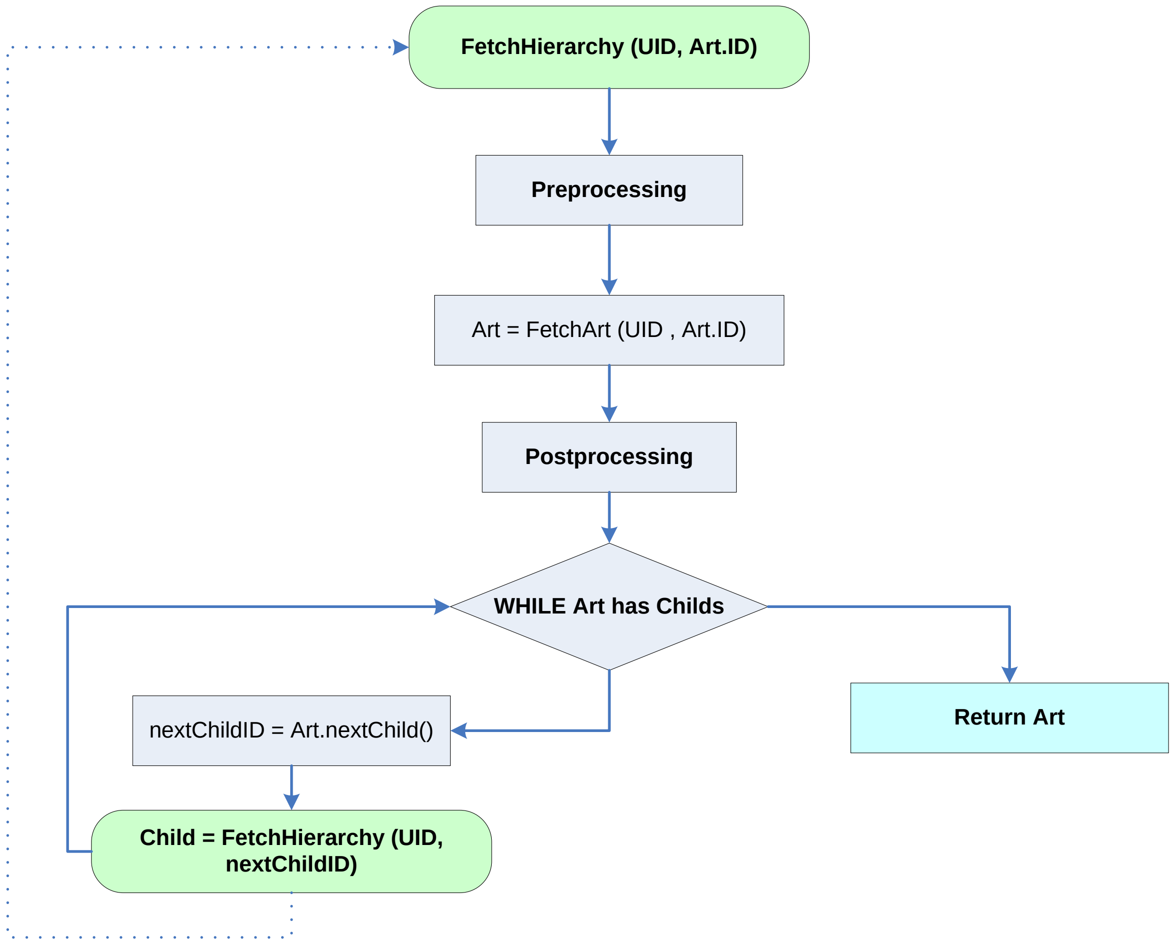

FetchHierarchy (UID, Art.ID)
Preprocessing
Art = FetchArt (UID , Art.ID)
Postprocessing
WHILE Art has Childs
Return Art
nextChildID = Art.nextChild()
Child = FetchHierarchy (UID, nextChildID)

Figure 4-21: Profile Retrieval

The profile is retrieved by providing the Profile Root ID, the Pre-processing and Post processing blocks are used to enforce the access control mechanism, fetching the desired key, decryption, etc.

4.5.7 Profile Update Service

Beside a user's own profile access hierarchy, a Safebook client also maintains PDVs of all the contacts in its contact list. As described earlier a PDV is a subset of an actual access hierarchy of a remote user based on the keys that have been shared. In order to remain synchronized with changes in the remote contacts' profiles, periodically the root access artifacts are fetched and their weights are compared with the weights of the stored PDV roots. If the weight is different than the stored PDV, then the respective profile is updated. This process is explained in Figure 4-22, only branches with changes in the access hierarchy which have different weights are fetched.

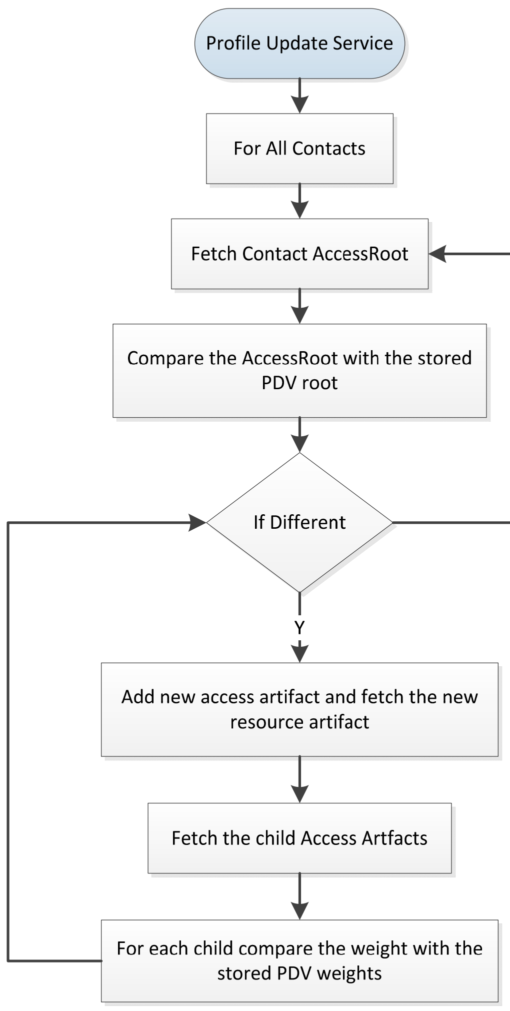

Profile Update Service
For All Contacts
Fetch Contact AccessRoot
Compare the AccessRoot with the stored PDV root
If Different
N
Y
Add new access artifact and fetch the new resource artifact
Fetch the child Access Artfacts
For each child compare the weight with the stored PDV weights

Figure 4-22: Profile Update Service

4.6 Social Networking Services Abstractions

Social networking services (SNS) are features offered by any SN provider. Designing SNS abstractions is outside of the scope of this thesis. However, in order to demonstrate that the proposed access control scheme supports the use cases discussed in Chapter 3, this section will explain a few cases with help of scenarios and examples.

4.6.1 Badges (Direct Relationships)

Badges refer to the relationship between users in the Safebook network. Usually these badges are based on real life relationships such as Friends, Family, Co Workers, etc. This SNS abstraction can be defined on top of the proposed access control scheme and the profile hierarchy will be defined as shown in Figure 4-23.

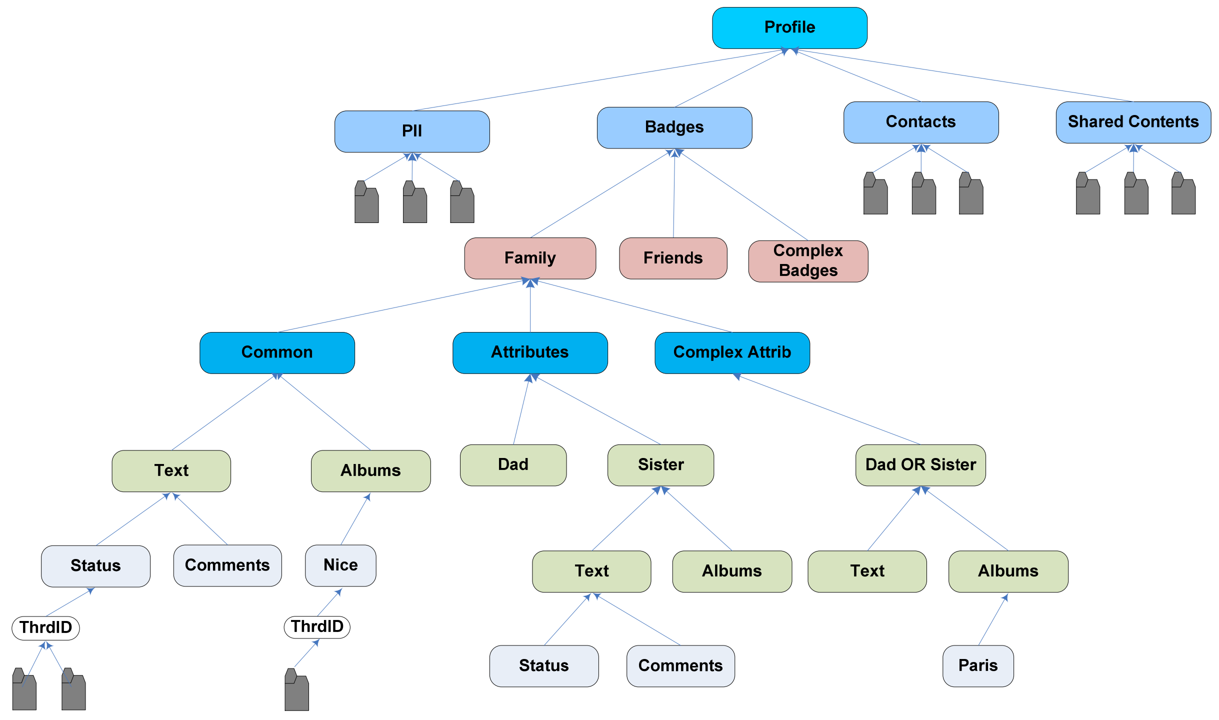


Figure 4-23: Profile Hierarchy with Badges

4.6.1.1 Badge Creation

A badge is a group of access keys packed together with a set of properties. Each property is associated with a group of access keys. As shown in Figure 4-23, the Family Badge has three properties. The Common Properties corresponds to access keys which will be shared among all the holders of this Family Badge. The attributes define further categorization of the Family Badge, e.g. Family badge has two attributes: Sister and Dad. Each attribute has its own set of access keys. Moreover, the property labeled as Complex Attributes defines Boolean AND and OR conditions over defined attributes.

4.6.1.2 Distribution

As the badges are based on the real life relationships, they must be validated by both parties. Badges cannot be distributed without a validation process. The badges are exchanged during the friendship request protocol or they can be exchanged through the mailbox service.

Consider the case when Alice wants to add Carl to her Family Badge. Carl is a brother of Alice, so in the Family Badge Alice adds an Attribute Brother with a new access key and sends a Relationship Approval Request with Labels as {Family , Brother}. Carl will review the labels, validate the given information and select a suitable Badge to return. In this case Carl will return a Badge {Family, Sister} with a signed certificate testify to this relationship. Once both the parties have validated the relationship labels, then the Badges are exchanged with appropriate properties. Alice will send Family badge with {Common, Attributes:Brother} and Carl will reply with a Badge with properties {Common, Attributes:Sister}. Moreover, the complex attributes are shared on demand when the conditions are created.

4.6.1.3 Scenarios

In this scenario we have three users, Alice, Bob, and Carl. Alice has a Family badge with properties as {Common, Attributes}, where the Attributes property has sub properties {Brother, Cousin}. Alice has added Carl and Bob in her contacts list and shared a Family badge with them. Carl is brother of Alice and Alice has shared the Brother Attributes, whereas with Bob Alice has only shared the Common property. Figure 4-24 illustrates this scenario.

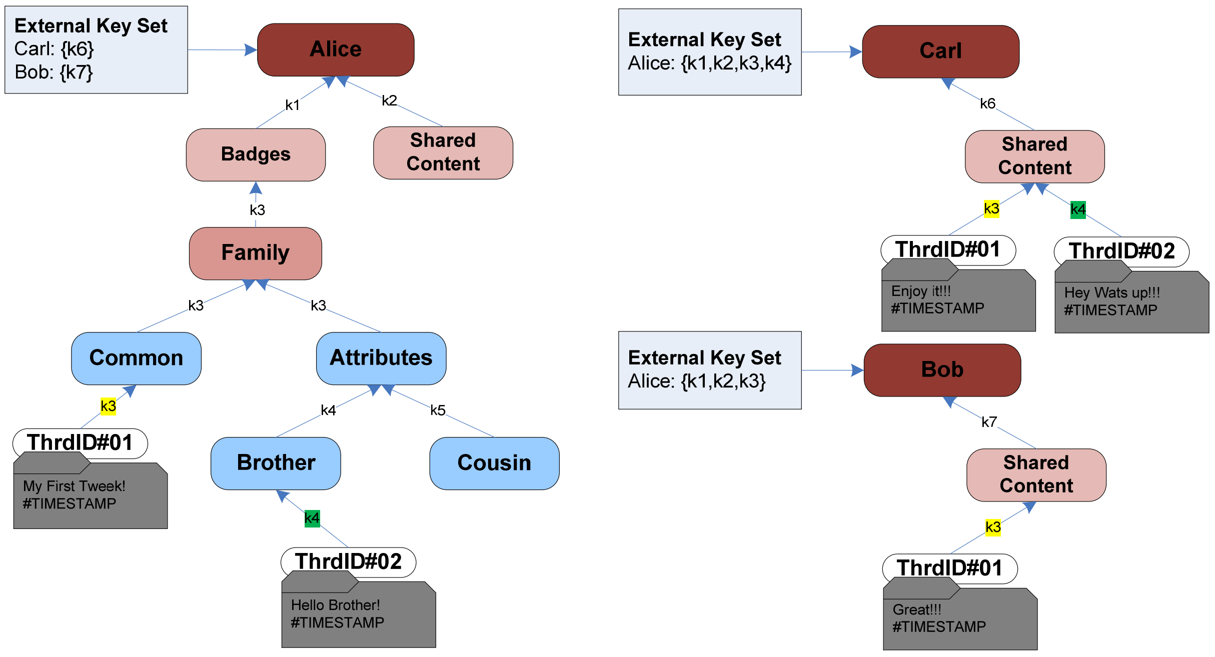


Figure 4-24: Badges Scenario, where Carl and Bob are contacts of Alice sharing same badge with different Properties

Each activity in the Safebook is represented with a thread having a unique threadID. Alice has added a new status message in her Family badge with access rights set to Common. The message is given a thread ID (in this case threadID=01), when Carl and Bob run their update process on Alice's profile, they will get this new message. In response Carl adds a comment on Alice's Status. This artifact is added as a child of Shared Content artifact with the same access key shared by Alice k3. The change in the access hierarchy of the Carl will be detected by Alice and the modification in the Shared Content branch is detected and the new artifact is fetched. The content of this artifact contains the message and the threadID. Thus this message is mapped to the threadID and Alice will have the Carl's comment. Similarly, Bob's comment is also received by Alice and mapped to the same thread. The timestamp is used to sort the messages in order to display them in the profile. Subsequently Alice adds another message to the Family Badge with access rights set to Brother. This will be only shared with Carl because he holds this attribute, whereas Bob will not be aware of this artifact.

4.6.2 Extended Relationships (Indirect Relationships)

The use of the indirect or extended relations can be demonstrated by considering facts revealed from Figure 4-24:

- Each user is responsible for its own artifacts.
- The privacy of each user is preserved.

  In scenario explained in Figure 4-24, Bob and Carl are not related to each other. This implies that on the Alice thread#01 Carl is not aware of Bob comment and vice versa.

In many cases, this interaction pattern may reduce the readability due to missing comments because of unauthorized access rights. The result is individuals messages grouped in a single thread instead of a conversation. So there must be some way to enable or disable relations such as "Friend-of-Friend", etc. In this section this process is demonstrated using the proposed access hierarchy.

Consider Figure 4-24, where Bob and Carl have a common contact Alice. Now Carl wants his activities to be reflected in these extended relations. This will be done in the following steps, shown in Figure 4-25:

- Carl sends an "Extended Relationship Request" to Alice with a defined propagation depth.
- Alice creates an Attestation which contains the Relationship it holds with Carl and the trust level. This trust level depends on the depth of the relationship. For direct relationships it is HIGH and the level reduces as the depth increases.
- This Attestation with the "Extended Relationship Request" is forwarded to Alice's contacts (i.e. Bob). On forwarding each request the propagation depth is decremented. When this parameter reaches zero, the request forwarding is stopped.
- On reception of "Extended Relationship Request", Bob analyzes the attached Attestation and decides whether to accept or reject the request.
  - If the request is accepted:
    - Bob sends an acknowledgment to Carl
    - Carl in response sends the access key of its "Shared Contents" artifacts.
    - Bob receives the access key {k6} and adds Carl to his extended contacts list to stay updated with his shared activates.
- Bob decrements the propagation depth, if it is not zero he forwards the request to members of his contacts list with his Attestation attached.

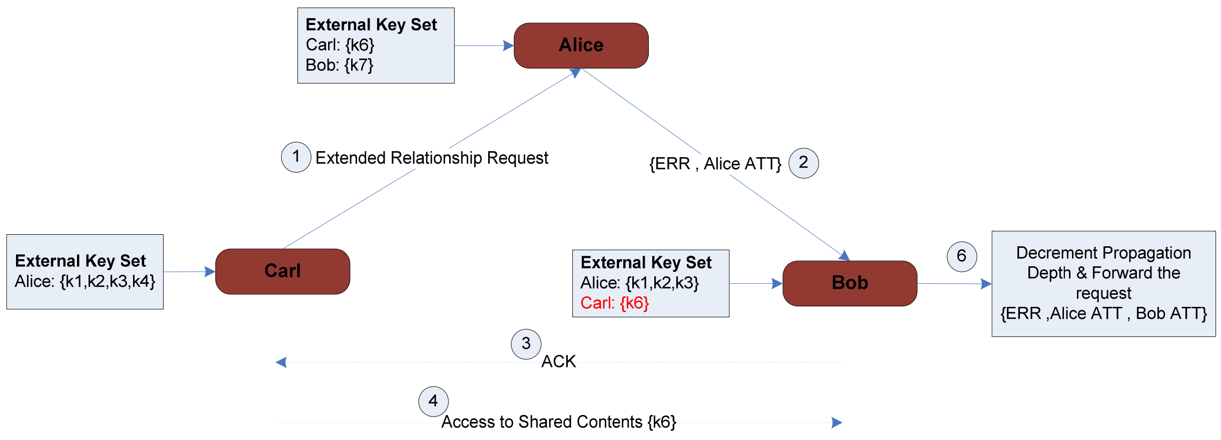


Figure 4-25: Extended Relationship Access Sharing Protocol

5 Implementation and Analysis

The proposed access control solution is implemented as a Java maven project using a test driven approach. The API aims to be generic and allow customization. The resulting system could be integrated with any P2P based social networking solution. The main parts of the implementation are discussed in the following section.

5.1 Components of the API

The complete functional overview of the API is shown in Figure 5-1. These components are functionally bundled into the following containers: artifact, access controller, cache, cipher, database handler, and Safebook Mock API. Each of these is described in a subsection below.

5.1.1 Artifact

This component consists of an Artifact Interface which provides a contract for the implementation. The contract is implemented by an abstract class called Abstract Artifact, providing common functionality for the underlying artifact types. The Access and Resource artifacts implements the Abstract Artifact class and add their own functional needs. Moreover, an Artifact Factory class is provided to store and load serialized artifact's objects from a database. The detail of the package is shown in Figure 5-2.

5.1.2 Access Controller

Access controller is the core of the API, and the entry point to access the API. Access Controller Interface provides all the methods explained in Chapter 3 to define the user access rights over the artifact hierarchy. The access controller uses the components of the API as described in the following paragraphs.

5.1.2.1 Hierarchy

The profile hierarchy is represented through the Hierarchy Interface as shown in Figure 5-3. In our solution the access artifacts are used to represent the profile hierarchy whereas the resource artifacts are decoupled from this information and stored separately. The Hierarchy contract is implemented by the Access Hierarchy class, which maintains the profile hierarchy and implements the operations for the profile hierarchy. The Access Hierarchy uses the Artifact Cache to store the access artifacts in the cache once they have been read from the database in order to increase performance.

5.1.2.2 Key Store Manager

The key store manager is a wrapper component over the Java Security API provided in the Java Standard Edition. The key store manager is responsible for storing and maintaining the encryption keys. The keys are stored in an encrypted key store file, where each key is associated with a password. The user name and password are provided through the Access Controller Configuration API.

5.1.2.3 Key Map

The Key Map class provides an interface to add and get mappings between the access and resource encryption keys. The mappings are stored in encrypted form in the database using the database handler API.

5.1.3 Cache

The caching service is designed over the JSR107 JCache standard *[29]* with the EHCACHE open source implementation *[30]*. The Abstract Cache provides a contract to implement function specific caches. It provides the methods getEntry() and putEntry(), and an abstract method retrieve(). The implementing cache class (such as the Artifact cache) implement the method retrieve() to define the logic to load the entry, e.g. in the case of the Artifact Cache the artifacts are loaded from the database using the database handler. Moreover, the Cache Key is an interface, and each cache implementation, provides its own cache key. A detailed class diagram is shown in Figure 5-4.

5.1.4 Cipher

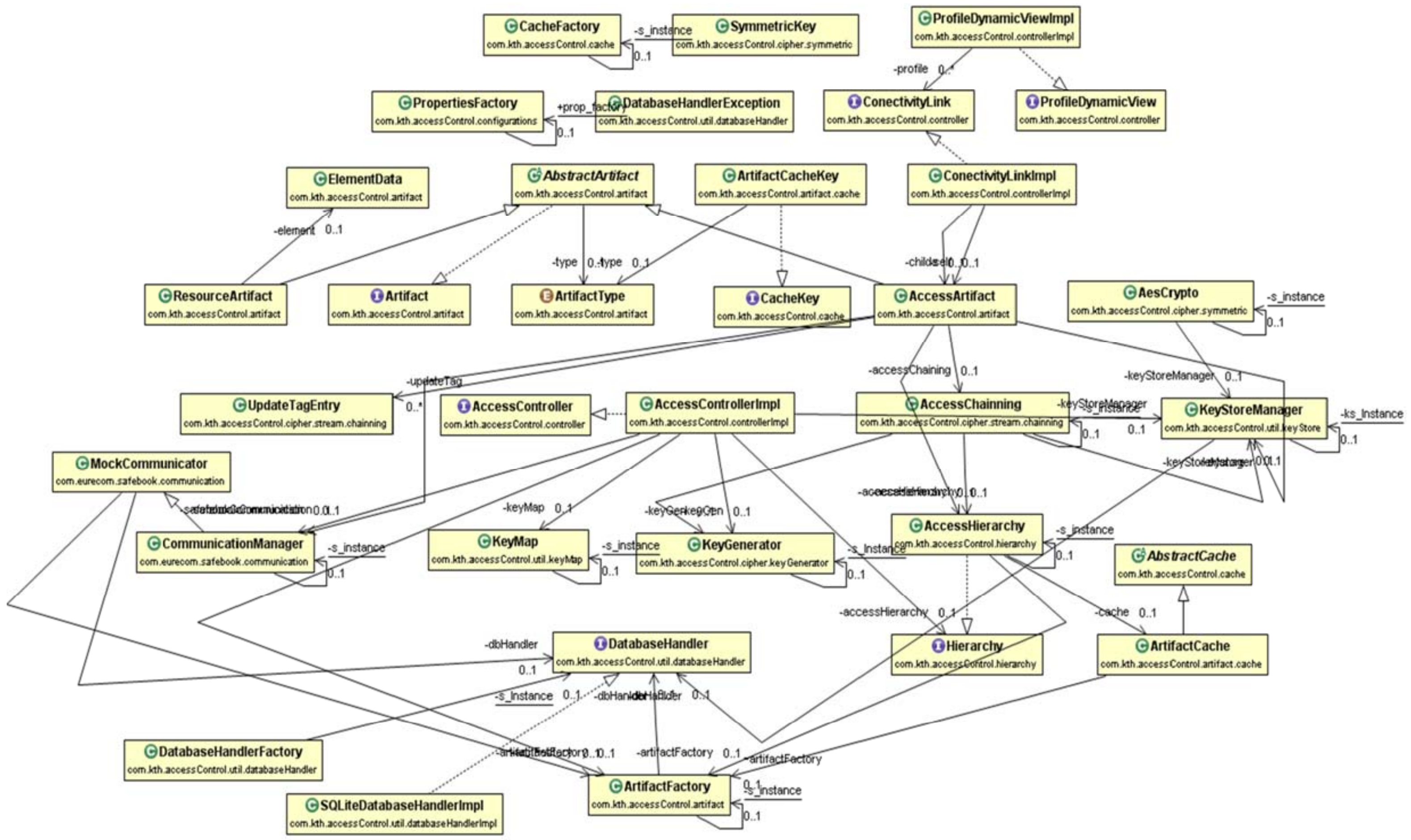
The cipher component deals with all the encryption functionality of the access controller. The AesCrypto class provides AES cryptographic functions using the Java Security API. The Key Generator class is used to generate unique encryption keys. Moreover, the Access chaining API implements the algorithms discussed in Chapter 3 for defining the dynamic access chains and extracting Artifact identifiers from these chains. A detailed class diagram is shown in Figure 5-5.

5.1.5 Database Handler

The Database Handler is a generic API to support multiple types of databases. The implementation is not specific to a database type. For use in this project an SQLITE database handler is provided to implement the Database Handler contract. The correct instance of database handler is acquired though the Database Handler Factory and the implementation class is provided through the Configuration class. Details of this are presented in the Figure 5-6.

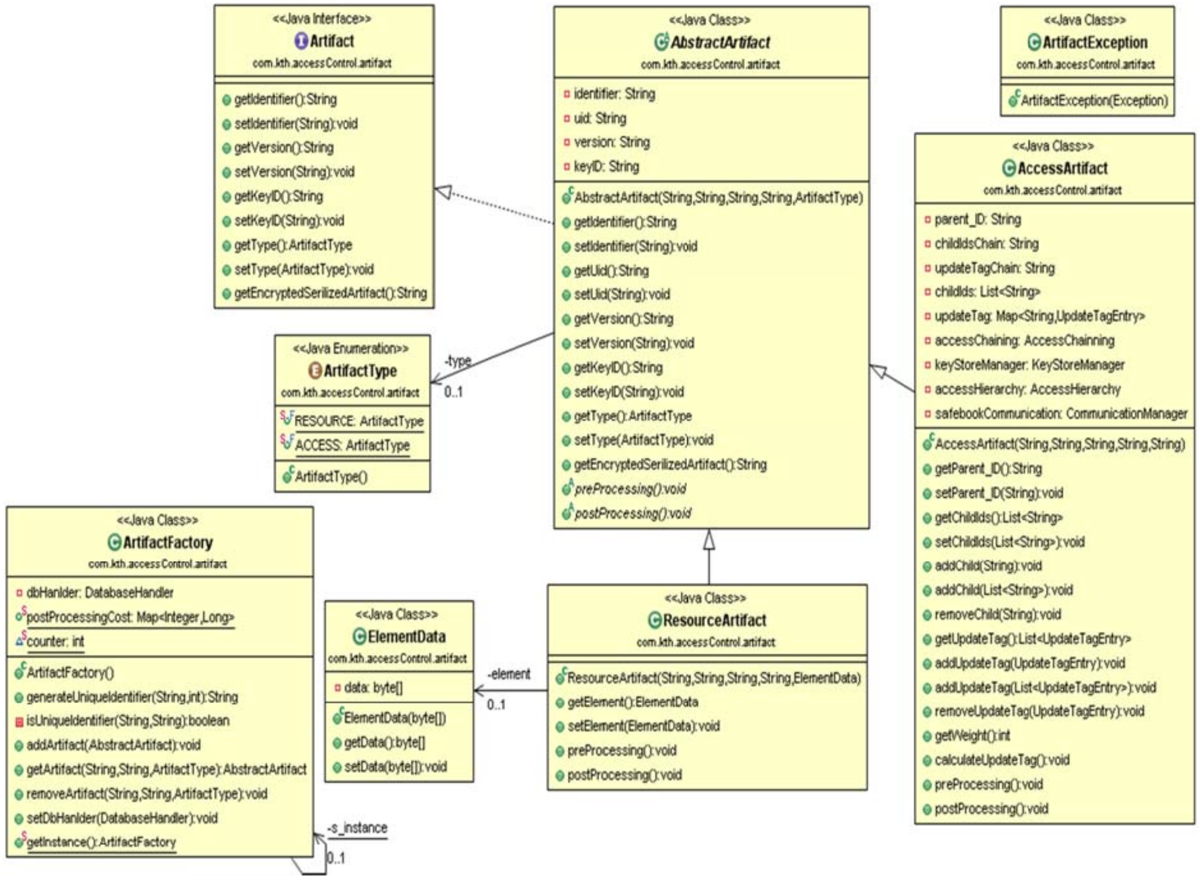
5.1.6 Safebook Mock API

The Safebook Mock API part of the Safebook API was mocked up to allow testing and analysis of the access hierarchy framework. The communication manager of Safebook is needed to push and get artifacts from the remote database. This is simulated by a mock communicator which uses a perfect to perfect (i.e., lossless) link with a delay of 1000 milliseconds. Further details are shown in Figure 5-6.



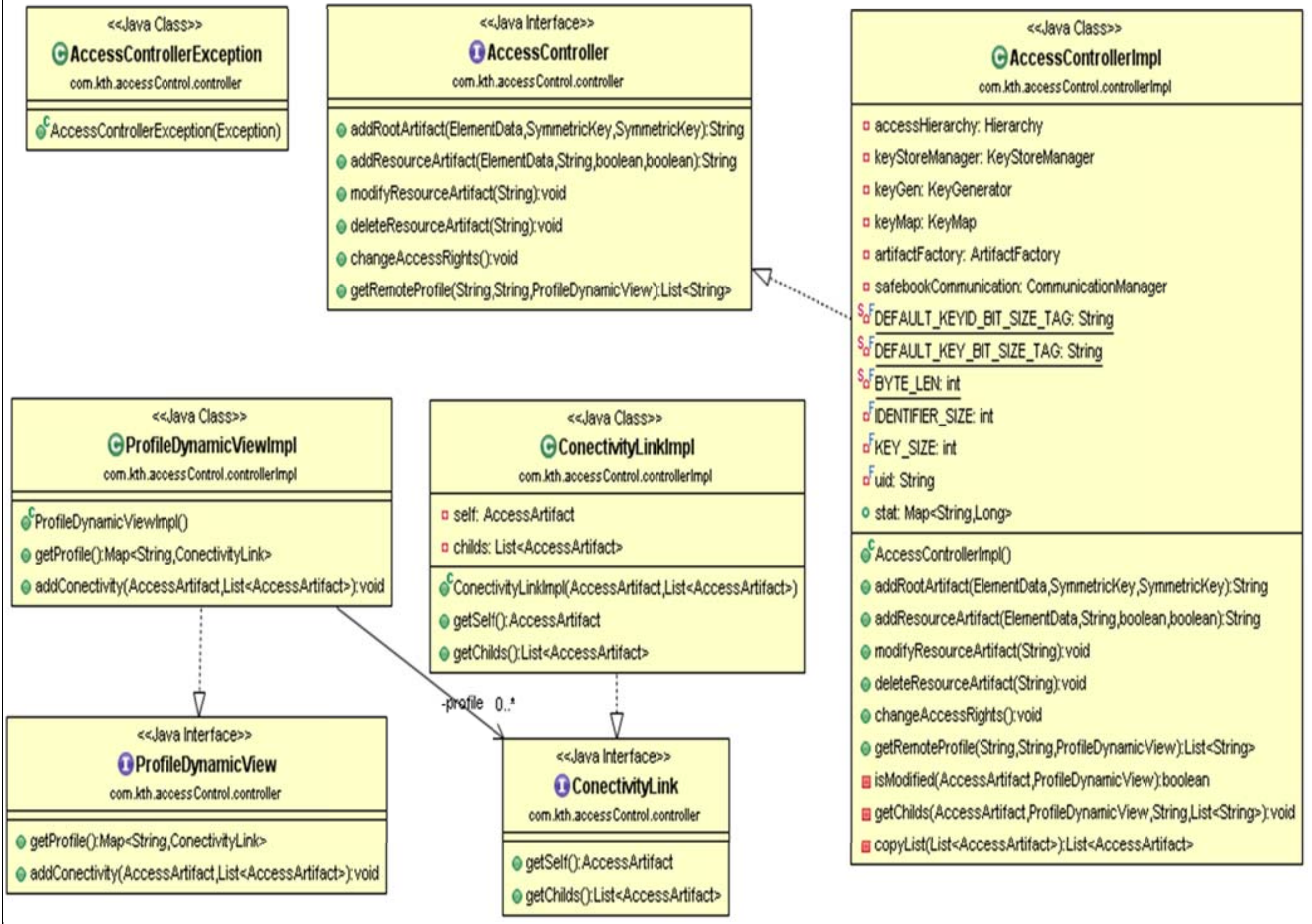
CacheFactory
com.kth.accessControl.cache
SymmetricKey
com.kth.accessControl.cipher.symmetric
ProfileDynamicViewImpl
com.kth.accessControl.controllerImpl
PropertiesFactory
com.kth.accessControl.configurations
DatabaseHandlerException
com.kth.accessControl.util.databaseHandler
ConectivityLink
com.kth.accessControl.controller
ProfileDynamicView
com.kth.accessControl.controller
ElementData
com.kth.accessControl.artifact
AbstractArtifact
com.kth.accessControl.artifact
ArtifactCacheKey
com.kth.accessControl.artifact.cache
ConectivityLinkImpl
com.kth.accessControl.controllerImpl
ResourceArtifact
com.kth.accessControl.artifact
Artifact
com.kth.accessControl.artifact
ArtifactType
com.kth.accessControl.artifact
CacheKey
com.kth.accessControl.cache
AccessArtifact
com.kth.accessControl.artifact
AesCrypto
com.kth.accessControl.cipher.symmetric
UpdateTagEntry
com.kth.accessControl.cipher.stream.chainning
AccessController
com.kth.accessControl.controller
AccessControllerImpl
com.kth.accessControl.controllerImpl
AccessChainning
com.kth.accessControl.cipher.stream.chainning
KeyStoreManager
com.kth.accessControl.util.keyStore
MockCommunicator
com.eurecom.safebook.communication
CommunicationManager
com.eurecom.safebook.communication
KeyMap
com.kth.accessControl.util.keyMap
KeyGenerator
com.kth.accessControl.cipher.keyGenerator
AccessHierarchy
com.kth.accessControl.hierarchy
AbstractCache
com.kth.accessControl.cache
DatabaseHandler
com.kth.accessControl.util.databaseHandler
Hierarchy
com.kth.accessControl.hierarchy
ArtifactCache
com.kth.accessControl.artifact.cache
DatabaseHandlerFactory
com.kth.accessControl.util.databaseHandler
ArtifactFactory
com.kth.accessControl.artifact
SQLiteDatabaseHandlerImpl
com.kth.accessControl.util.databaseHandlerImpl
-s_instance 0..1
-profile 0..*
+prop_factory 0..1
-element 0..1
-type 0..1
-childs 0..1
-s_instance 0..1
-updateTag 0..*
-accessChaining 0..1
-keyStoreManager 0..1
-ks_Instance 0..1
-keyMap 0..1
-keyGen 0..1
-s_instance 0..1
-accessHierarchy 0..1
-cache 0..1
-dbHandler 0..1
-artifactFactory 0..1

Figure 5-1: Access Controller API Components



<<Java Interface>>
Artifact
com.kth.accessControl.artifact
getIdentifier():String
setIdentifier(String):void
getVersion():String
setVersion(String):void
getKeyID():String
setKeyID(String):void
getType():ArtifactType
setType(ArtifactType):void
getEncryptedSerilizedArtifact():String
<<Java Class>>
AbstractArtifact
com.kth.accessControl.artifact
identifier: String
uid: String
version: String
keyID: String
AbstractArtifact(String,String,String,String,ArtifactType)
getIdentifier():String
setIdentifier(String):void
getUid():String
setUid(String):void
getVersion():String
setVersion(String):void
getKeyID():String
setKeyID(String):void
getType():ArtifactType
setType(ArtifactType):void
getEncryptedSerilizedArtifact():String
preProcessing():void
postProcessing():void
<<Java Class>>
ArtifactException
com.kth.accessControl.artifact
ArtifactException(Exception)
<<Java Class>>
AccessArtifact
com.kth.accessControl.artifact
parent_ID: String
childIdsChain: String
updateTagChain: String
childIds: List<String>
updateTag: Map<String,UpdateTagEntry>
accessChaining: AccessChainning
keyStoreManager: KeyStoreManager
accessHierarchy: AccessHierarchy
safebookCommunication: CommunicationManager
AccessArtifact(String,String,String,String,String)
getParent_ID():String
setParent_ID(String):void
getChildIds():List<String>
setChildIds(List<String>):void
addChild(String):void
addChild(List<String>):void
removeChild(String):void
getUpdateTag():List<UpdateTagEntry>
addUpdateTag(UpdateTagEntry):void
addUpdateTag(List<UpdateTagEntry>):void
removeUpdateTag(UpdateTagEntry):void
getWeight():int
calculateUpdateTag():void
preProcessing():void
postProcessing():void
-type
0..1
<<Java Enumeration>>
ArtifactType
com.kth.accessControl.artifact
RESOURCE: ArtifactType
ACCESS: ArtifactType
ArtifactType()
<<Java Class>>
ArtifactFactory
com.kth.accessControl.artifact
dbHanlder: DatabaseHandler
postProcessingCost: Map<Integer,Long>
counter: int
ArtifactFactory()
generateUniqueIdentifier(String,int):String
isUniqueIdentifier(String,String):boolean
addArtifact(AbstractArtifact):void
getArtifact(String,String,ArtifactType):AbstractArtifact
removeArtifact(String,String,ArtifactType):void
setDbHanlder(DatabaseHandler):void
getInstance():ArtifactFactory
-s_instance
0..1
<<Java Class>>
ElementData
com.kth.accessControl.artifact
data: byte[]
ElementData(byte[])
getData():byte[]
setData(byte[]):void
-element
0..1
<<Java Class>>
ResourceArtifact
com.kth.accessControl.artifact
ResourceArtifact(String,String,String,String,ElementData)
getElement():ElementData
setElement(ElementData):void
preProcessing():void
postProcessing():void

Figure 5-2: Artifact Components Class Diagram



<<Java Class>>
AccessControllerException
com.kth.accessControl.controller
AccessControllerException(Exception)
<<Java Interface>>
AccessController
com.kth.accessControl.controller
addRootArtifact(ElementData,SymmetricKey,SymmetricKey):String
addResourceArtifact(ElementData,String,boolean,boolean):String
modifyResourceArtifact(String):void
deleteResourceArtifact(String):void
changeAccessRights():void
getRemoteProfile(String,String,ProfileDynamicView):List<String>
<<Java Class>>
AccessControllerImpl
com.kth.accessControl.controllerImpl
accessHierarchy: Hierarchy
keyStoreManager: KeyStoreManager
keyGen: KeyGenerator
keyMap: KeyMap
artifactFactory: ArtifactFactory
safebookCommunication: CommunicationManager
DEFAULT_KEYID_BIT_SIZE_TAG: String
DEFAULT_KEY_BIT_SIZE_TAG: String
BYTE_LEN: int
IDENTIFIER_SIZE: int
KEY_SIZE: int
uid: String
stat: Map<String,Long>
AccessControllerImpl()
addRootArtifact(ElementData,SymmetricKey,SymmetricKey):String
addResourceArtifact(ElementData,String,boolean,boolean):String
modifyResourceArtifact(String):void
deleteResourceArtifact(String):void
changeAccessRights():void
getRemoteProfile(String,String,ProfileDynamicView):List<String>
isModified(AccessArtifact,ProfileDynamicView):boolean
getChilds(AccessArtifact,ProfileDynamicView,String,List<String>):void
copyList(List<AccessArtifact>):List<AccessArtifact>
<<Java Class>>
ProfileDynamicViewImpl
com.kth.accessControl.controllerImpl
ProfileDynamicViewImpl()
getProfile():Map<String,ConectivityLink>
addConectivity(AccessArtifact,List<AccessArtifact>):void
<<Java Class>>
ConectivityLinkImpl
com.kth.accessControl.controllerImpl
self: AccessArtifact
childs: List<AccessArtifact>
ConectivityLinkImpl(AccessArtifact,List<AccessArtifact>)
getSelf():AccessArtifact
getChilds():List<AccessArtifact>
-profile 0..*
<<Java Interface>>
ProfileDynamicView
com.kth.accessControl.controller
getProfile():Map<String,ConectivityLink>
addConectivity(AccessArtifact,List<AccessArtifact>):void
<<Java Interface>>
ConectivityLink
com.kth.accessControl.controller
getSelf():AccessArtifact
getChilds():List<AccessArtifact>

Figure 5-3: Access Controller Class Diagram

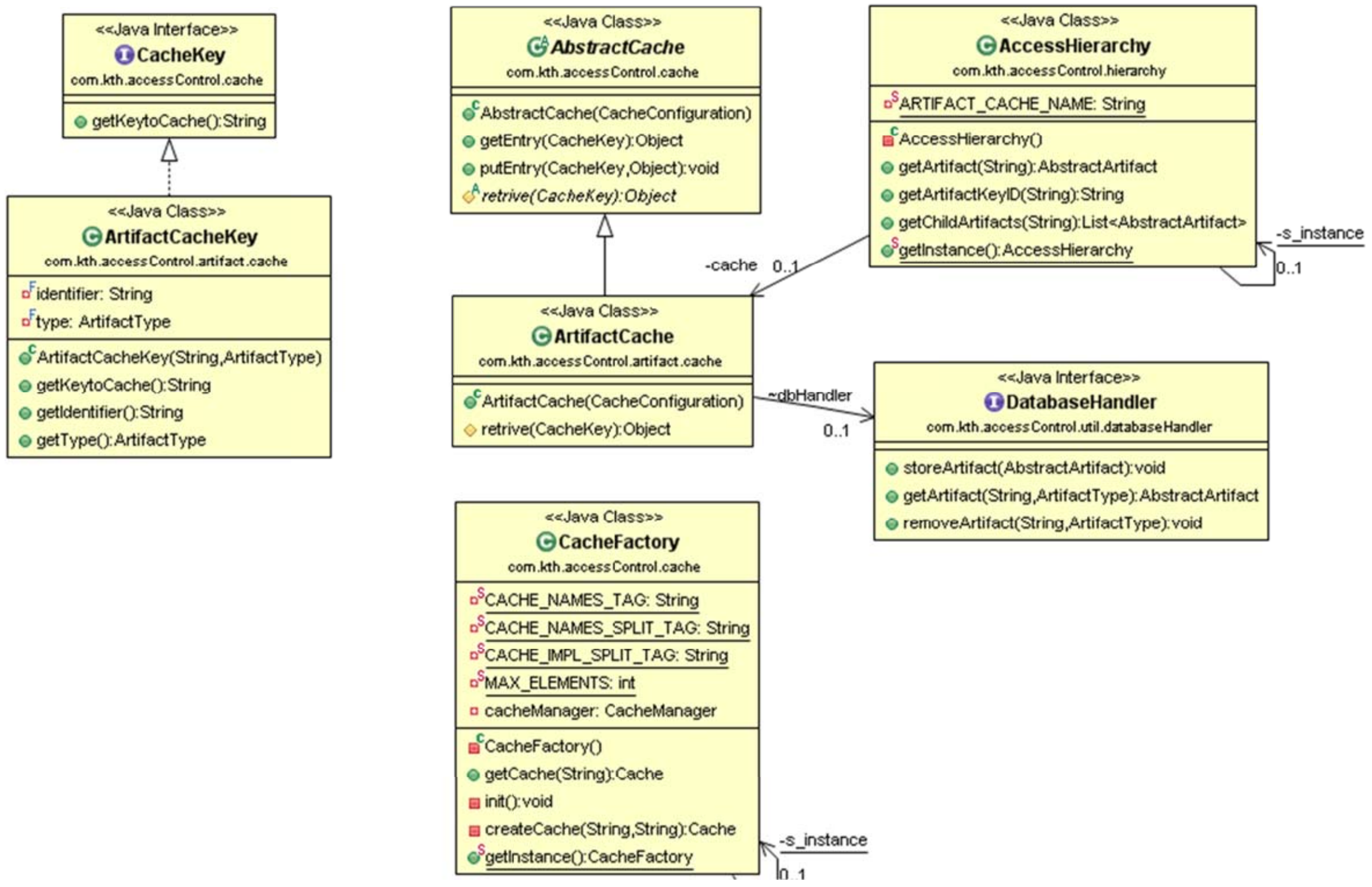
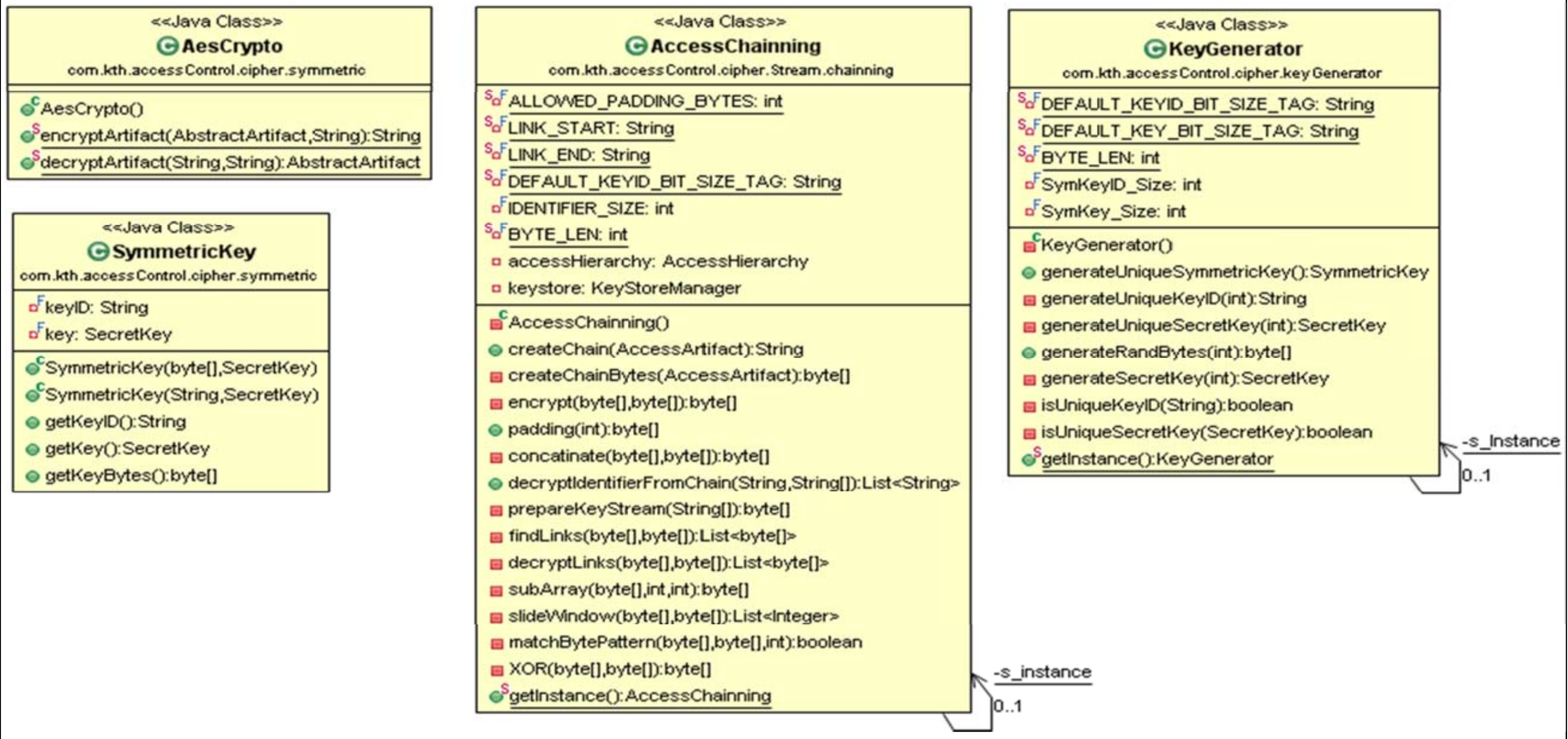
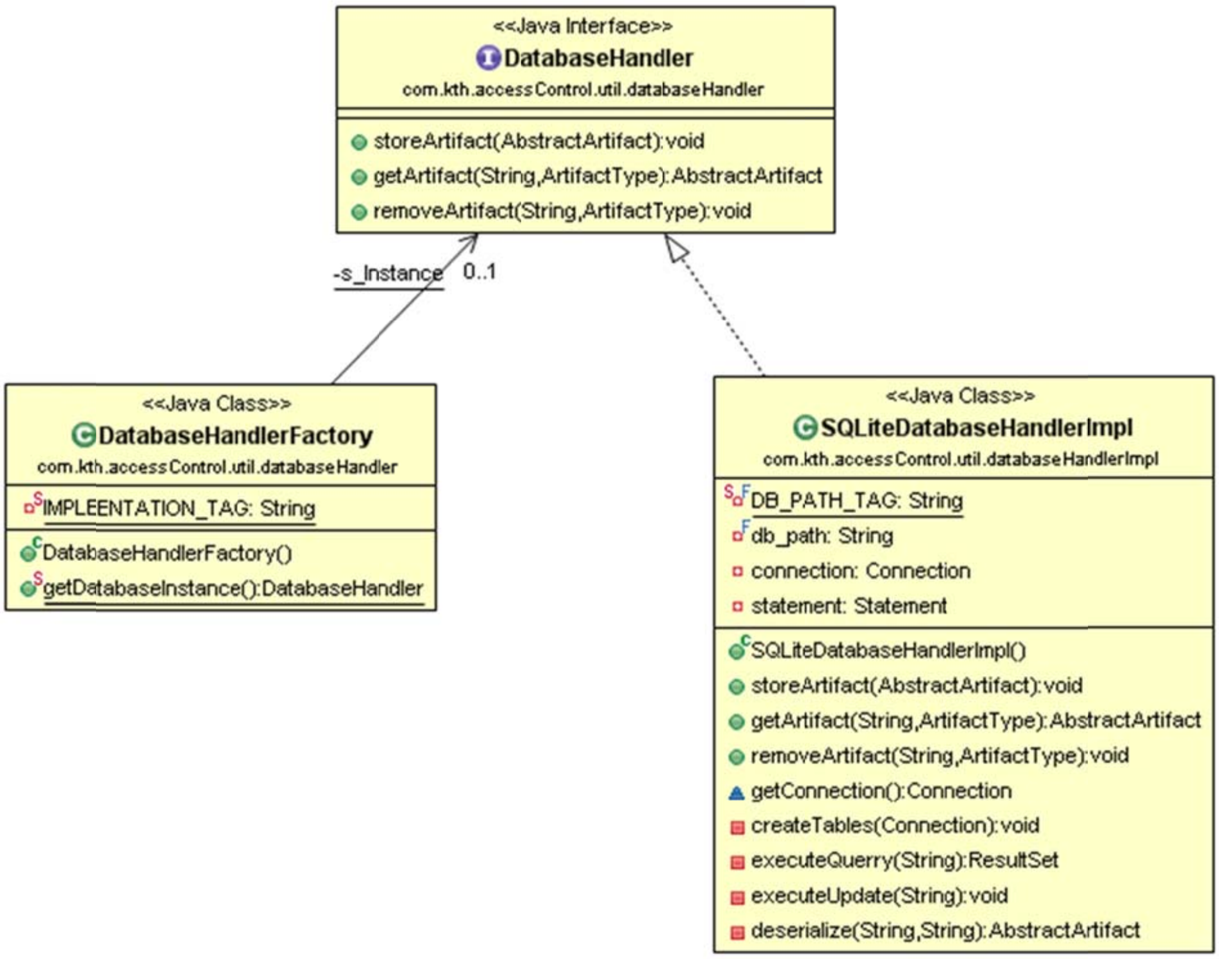

<<Java Interface>>
CacheKey
com.kth.accessControl.cache
getKeytoCache():String
<<Java Class>>
ArtifactCacheKey
com.kth.accessControl.artifact.cache
identifier: String
type: ArtifactType
ArtifactCacheKey(String,ArtifactType)
getKeytoCache():String
getIdentifier():String
getType():ArtifactType
<<Java Class>>
AbstractCache
com.kth.accessControl.cache
AbstractCache(CacheConfiguration)
getEntry(CacheKey):Object
putEntry(CacheKey,Object):void
retrive(CacheKey):Object
<<Java Class>>
AccessHierarchy
com.kth.accessControl.hierarchy
ARTIFACT_CACHE_NAME: String
AccessHierarchy()
getArtifact(String):AbstractArtifact
getArtifactKeyID(String):String
getChildArtifacts(String):List<AbstractArtifact>
getInstance():AccessHierarchy
-s_instance
0..1
-cache
0..1
<<Java Class>>
ArtifactCache
com.kth.accessControl.artifact.cache
ArtifactCache(CacheConfiguration)
retrive(CacheKey):Object
~dbHandler
0..1
<<Java Interface>>
DatabaseHandler
com.kth.accessControl.util.databaseHandler
storeArtifact(AbstractArtifact):void
getArtifact(String,ArtifactType):AbstractArtifact
removeArtifact(String,ArtifactType):void
<<Java Class>>
CacheFactory
com.kth.accessControl.cache
CACHE_NAMES_TAG: String
CACHE_NAMES_SPLIT_TAG: String
CACHE_IMPL_SPLIT_TAG: String
MAX_ELEMENTS: int
cacheManager: CacheManager
CacheFactory()
getCache(String):Cache
init():void
createCache(String,String):Cache
getInstance():CacheFactory
-s_instance
0..1

Figure 5-4: Cache Class Diagram



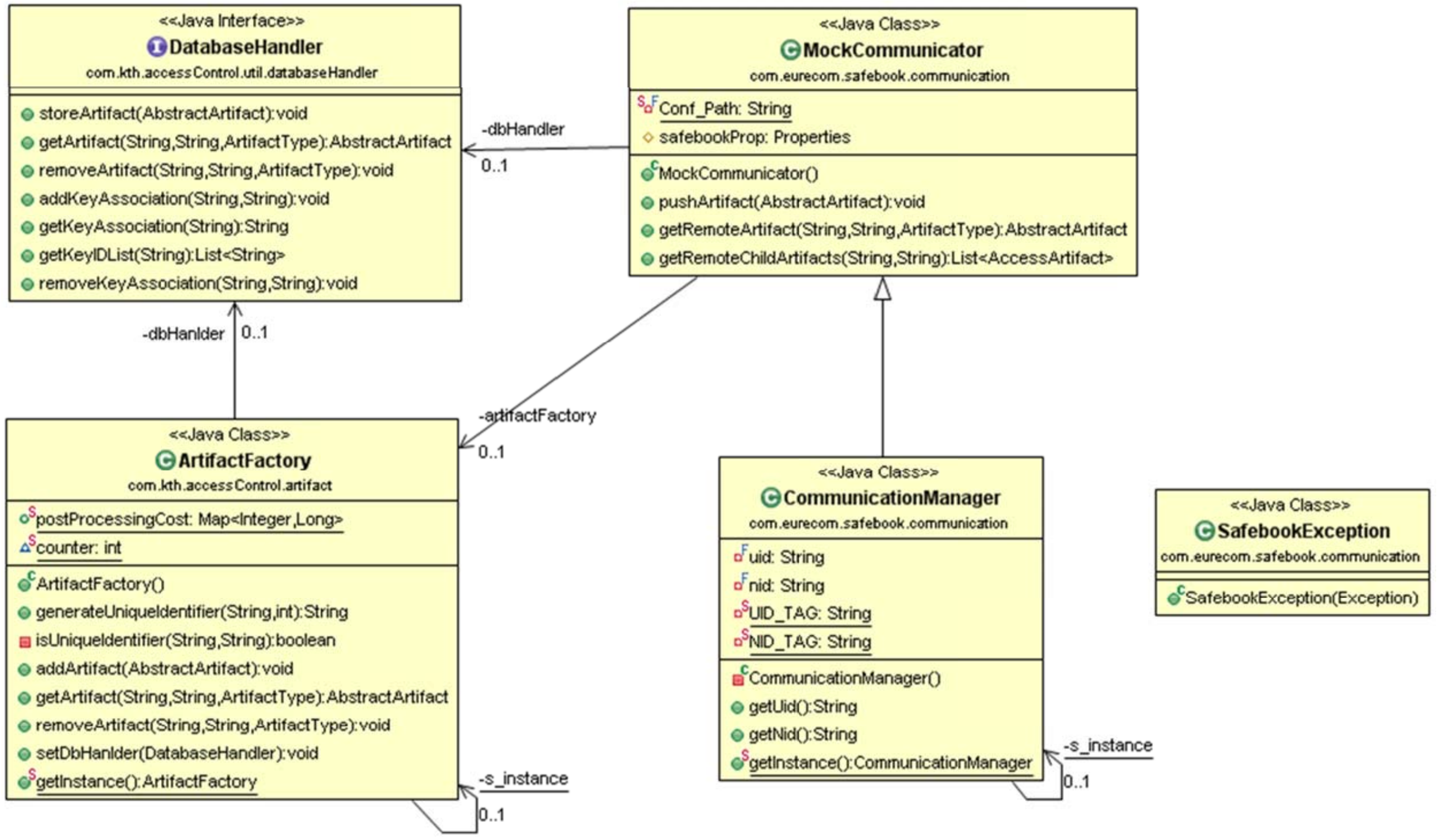
<<Java Class>>
AesCrypto
com.kth.accessControl.cipher.symmetric
AesCrypto()
encryptArtifact(AbstractArtifact,String):String
decryptArtifact(String,String):AbstractArtifact
<<Java Class>>
SymmetricKey
com.kth.accessControl.cipher.symmetric
keyID: String
key: SecretKey
SymmetricKey(byte[],SecretKey)
SymmetricKey(String,SecretKey)
getKeyID():String
getKey():SecretKey
getKeyBytes():byte[]
<<Java Class>>
AccessChainning
com.kth.accessControl.cipher.Stream.chainning
ALLOWED_PADDING_BYTES: int
LINK_START: String
LINK_END: String
DEFAULT_KEYID_BIT_SIZE_TAG: String
IDENTIFIER_SIZE: int
BYTE_LEN: int
accessHierarchy: AccessHierarchy
keystore: KeyStoreManager
AccessChainning()
createChain(AccessArtifact):String
createChainBytes(AccessArtifact):byte[]
encrypt(byte[],byte[]):byte[]
padding(int):byte[]
concatinate(byte[],byte[]):byte[]
decryptIdentifierFromChain(String,String[]):List<String>
prepareKeyStream(String[]):byte[]
findLinks(byte[],byte[]):List<byte[]>
decryptLinks(byte[],byte[]):List<byte[]>
subArray(byte[],int,int):byte[]
slideWindow(byte[],byte[]):List<Integer>
matchBytePattern(byte[],byte[],int):boolean
XOR(byte[],byte[]):byte[]
getInstance():AccessChainning
-s_instance
0..1
<<Java Class>>
KeyGenerator
com.kth.accessControl.cipher.keyGenerator
DEFAULT_KEYID_BIT_SIZE_TAG: String
DEFAULT_KEY_BIT_SIZE_TAG: String
BYTE_LEN: int
SymKeyID_Size: int
SymKey_Size: int
KeyGenerator()
generateUniqueSymmetricKey():SymmetricKey
generateUniqueKeyID(int):String
generateUniqueSecretKey(int):SecretKey
generateRandBytes(int):byte[]
generateSecretKey(int):SecretKey
isUniqueKeyID(String):boolean
isUniqueSecretKey(SecretKey):boolean
getInstance():KeyGenerator
-s_Instance
0..1

Figure 5-5: Cipher Component Class Diagram



<<Java Interface>>
DatabaseHandler
com.kth.accessControl.util.databaseHandler
storeArtifact(AbstractArtifact):void
getArtifact(String,ArtifactType):AbstractArtifact
removeArtifact(String,ArtifactType):void
-s_Instance
0..1
<<Java Class>>
DatabaseHandlerFactory
com.kth.accessControl.util.databaseHandler
IMPLEENTATION_TAG: String
DatabaseHandlerFactory()
getDatabaseInstance():DatabaseHandler
<<Java Class>>
SQLiteDatabaseHandlerImpl
com.kth.accessControl.util.databaseHandlerImpl
DB_PATH_TAG: String
db_path: String
connection: Connection
statement: Statement
SQLiteDatabaseHandlerImpl()
storeArtifact(AbstractArtifact):void
getArtifact(String,ArtifactType):AbstractArtifact
removeArtifact(String,ArtifactType):void
getConnection():Connection
createTables(Connection):void
executeQuerry(String):ResultSet
executeUpdate(String):void
deserialize(String,String):AbstractArtifact

Figure 5-6: Database Handler Class Diagram



<<Java Interface>>
DatabaseHandler
com.kth.accessControl.util.databaseHandler
storeArtifact(AbstractArtifact):void
getArtifact(String,String,ArtifactType):AbstractArtifact
removeArtifact(String,String,ArtifactType):void
addKeyAssociation(String,String):void
getKeyAssociation(String):String
getKeyIDList(String):List<String>
removeKeyAssociation(String,String):void
-dbHandler
0..1
<<Java Class>>
MockCommunicator
com.eurecom.safebook.communication
Conf_Path: String
safebookProp: Properties
MockCommunicator()
pushArtifact(AbstractArtifact):void
getRemoteArtifact(String,String,ArtifactType):AbstractArtifact
getRemoteChildArtifacts(String,String):List<AccessArtifact>
-dbHanlder
0..1
-artifactFactory
0..1
<<Java Class>>
ArtifactFactory
com.kth.accessControl.artifact
postProcessingCost: Map<Integer,Long>
counter: int
ArtifactFactory()
generateUniqueIdentifier(String,int):String
isUniqueIdentifier(String,String):boolean
addArtifact(AbstractArtifact):void
getArtifact(String,String,ArtifactType):AbstractArtifact
removeArtifact(String,String,ArtifactType):void
setDbHanlder(DatabaseHandler):void
getInstance():ArtifactFactory
-s_instance
0..1
<<Java Class>>
CommunicationManager
com.eurecom.safebook.communication
uid: String
nid: String
UID_TAG: String
NID_TAG: String
CommunicationManager()
getUid():String
getNid():String
getInstance():CommunicationManager
-s_instance
0..1
<<Java Class>>
SafebookException
com.eurecom.safebook.communication
SafebookException(Exception)

Figure 5-7: Safebook Mock API for Analysis

5.2 Analysis

This section analyses the proposed access control scheme according to a number of performance metrics. During the analysis the Safebook overhead has been ignored and only the access scheme has been tested with a number of stress tests to measure the performance of each component. The simulations of the Safebook system were run in an Eclipse Java Environment on an Intel® Core™ 2 Duo CPU T7300 @ 2.0 GHz with 3 GB of RAM running Microsoft Windows XP.

5.2.1 Performance Metrics

The access control mechanism proposed in this thesis is based on symmetric stream encryption. The profile is represented as a hierarchy of artifacts containing encrypted chains of data, which hides the links. The performance metrics are used to study the feasibility and applicability of the system in real distributed social networks.

5.2.1.1 Encryption Cost

Two types of encryptions are used in the solution, AES encryption is used with the symmetric access keys to encrypt the Artifacts. The same key is used as a key stream for the stream cipher used to create dynamic access chains.

5.2.1.1.1 AES

Figure 5-8 shows the AES Cipher performance. The processing time is plotted against the size of the plain text block. The processing time shows a linear trend for both the encryption and decryption modes of the AES API provided by the Java Crypto library. The cost of encrypting a 5 MB block takes on average 150 milliseconds which is very acceptable, particularly as on average the access artifact size is only a few kilobytes.

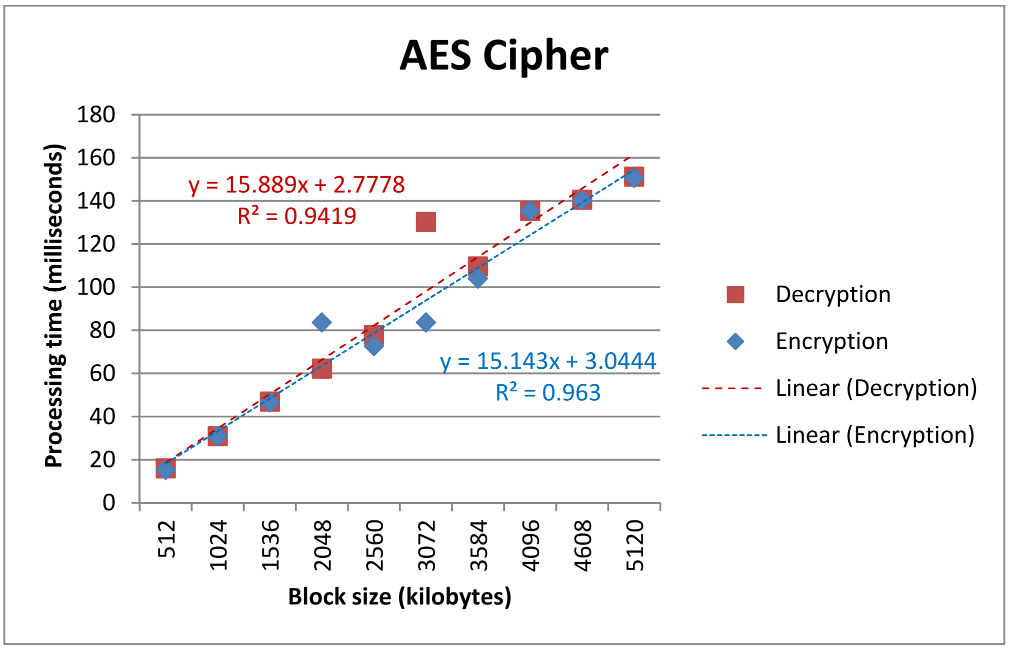


Figure 5-8: AES Cipher Graph: Processing Cost vs. Block Size

5.2.1.1.2 Chaining Creation/Decryption

Access chaining is the core of the design, so here the performance of the process is studied using several different cases.

The variables impacting the performance of chain creation are No of artifacts, Access Keys used and size of the access hierarchy cache. Figure 5-9 and Figure 5-10 represents the performance in two different scenarios. The first graph represents the case when only one access key is defined for all the child artifacts, whereas the second graph defines a unique key for *each* child artifact.

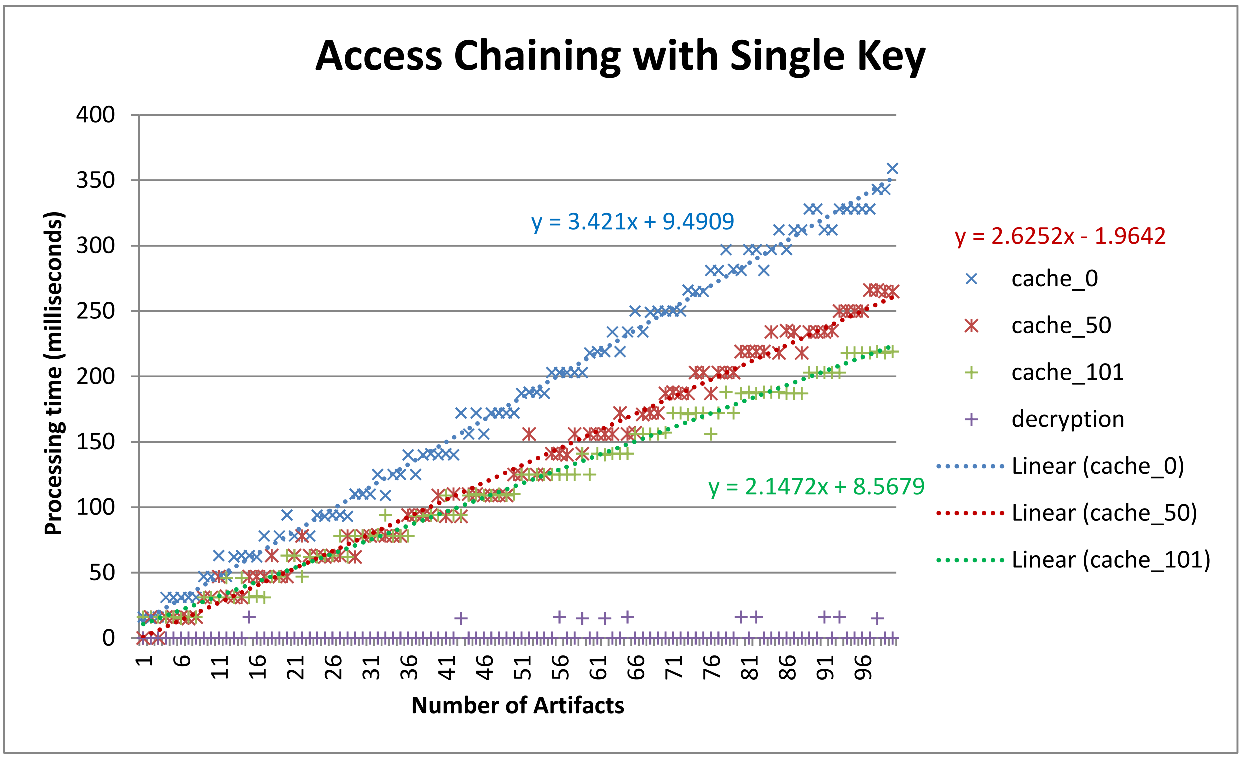


Figure 5-9: Access Chaining with Single Access Key: Number of artifacts vs. processing time

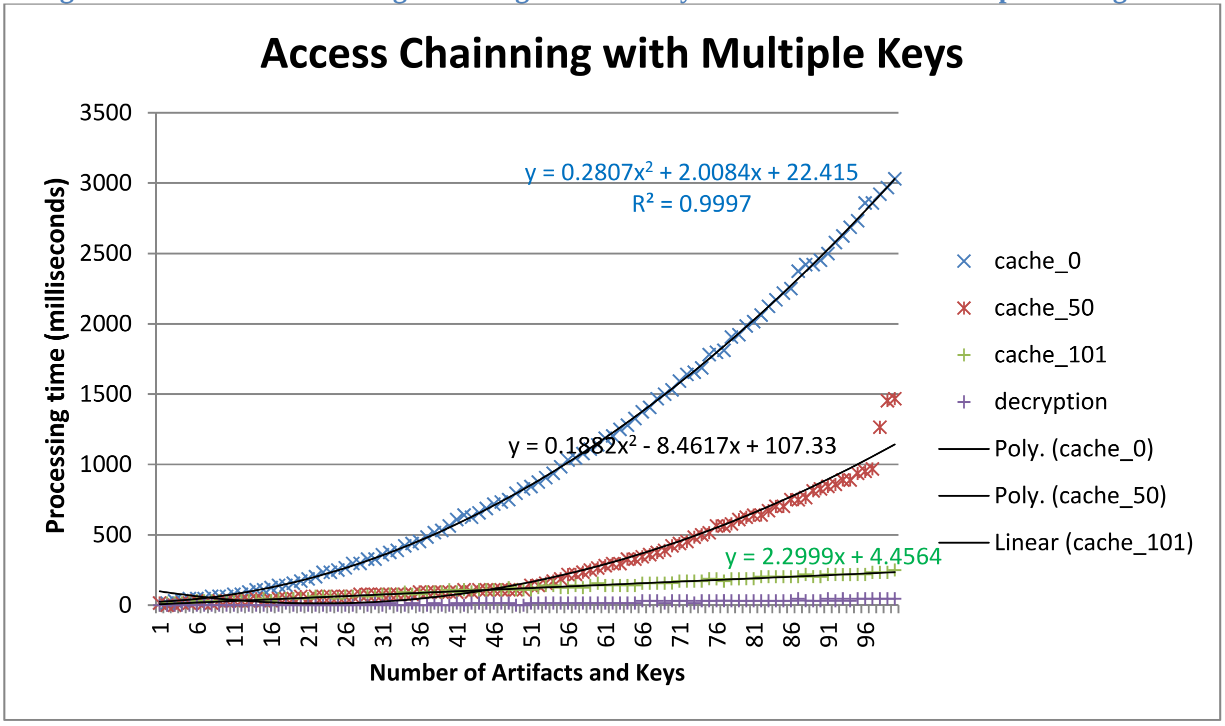


Figure 5-10: Access Chaining with Multiple Access Keys: x-axis represents the number of Artifacts and Access keys vs. processing time

In case of a single access key, as the number of artifacts increases the processing time increases linearly for all cache sizes. However, the behavior is not same for the case of multiple keys due to added cost of the deserialization process. The deserialization processing time depends on the number of keys associated with each UID. When caching is disabled, all the artifacts are read from the database, which stores serialized artifacts. During serialization the child identifiers and the update tags are converted into access chains because both these list are defined as transient. Moreover, each artifact is stored in encrypted form in the database and the format is same as that of the remote database. Therefore, during the deserialization process each chain is decrypted to access the artifact child identifiers and the artifact update tag. The process shows a significant decrease in the processing cost when caching is used as the deserialization post processing is not done.

The decryption cost for both the single and multiple access keys cases is very much smaller than the creation cost. For multiple keys a slight increase in the processing cost is observed due to the fact that the creation of key stream involves loading the keys from the key store.

Figure 5-11 shows the size of the chain versus the number of child identifiers an artifact holds. Here the size is linearly increasing.

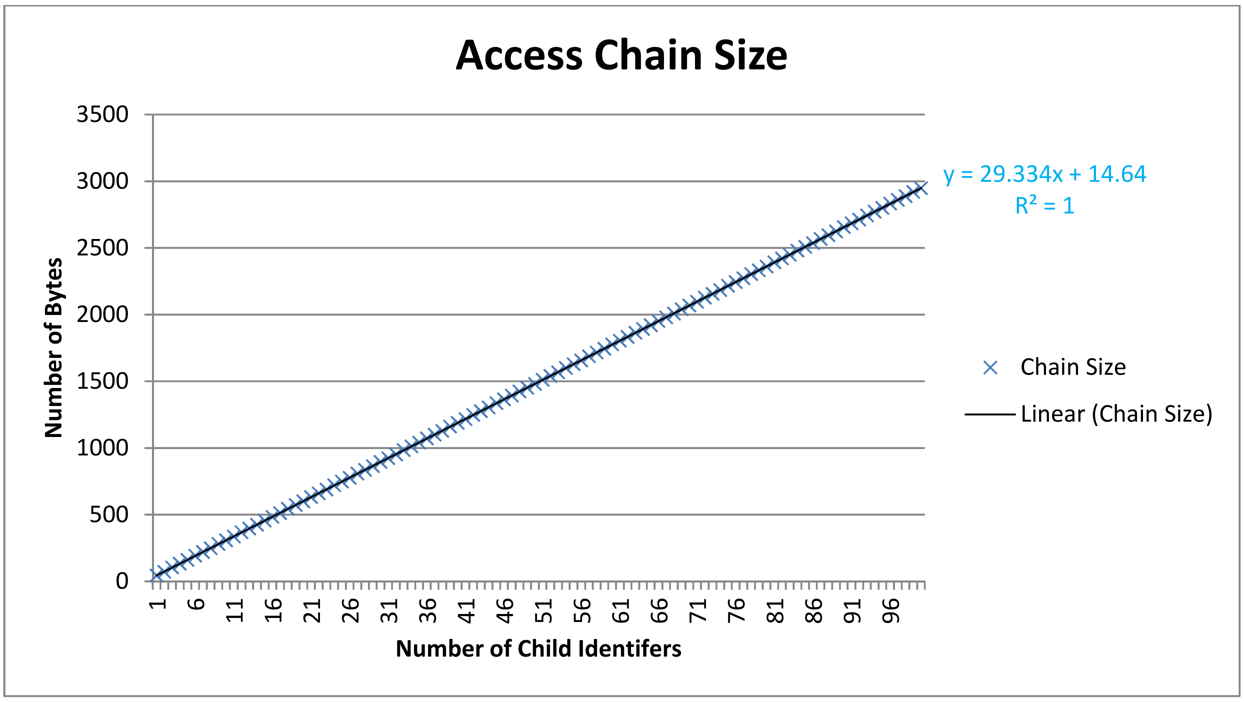


Figure 5-11: Access Chain Size: Number of Child Identifiers vs. Number of Bytes

5.2.1.2 Access Hierarchy Size

In order to access the remote profile, an access hierarchy needs to be downloaded. This access hierarchy is the set of exposed artifacts for a given user. For each user the size of the access hierarchy should be small enough to be easily transferred over the network and has a small memory footprint. Figure 5-12 shows that, the size of the access hierarchy increases linearly with a slope of 1.567 as new artifacts are added. If size of the access hierarchy is projected with this same slope of 1.567, the size of 10,000 artifacts is estimated to be ~15.23 MB. This means for an average of 100 friends each Safebook node has to maintain less than ~1,500 MB. If the compression is used this reduces the slope to 0.92 which means for an average of 10,000 artifacts with on average 100 friends ~898 MB of data needs to be stored and maintained. This data is still very small as compared to the actual data stored for each profile. As profiles contain a lot of multimedia contents, for example the average size of an image from a 5 mega pixel camera is more than 2 MB, whereas, an access hierarchy of 1,000 artifacts is less 1,600 kilobytes.

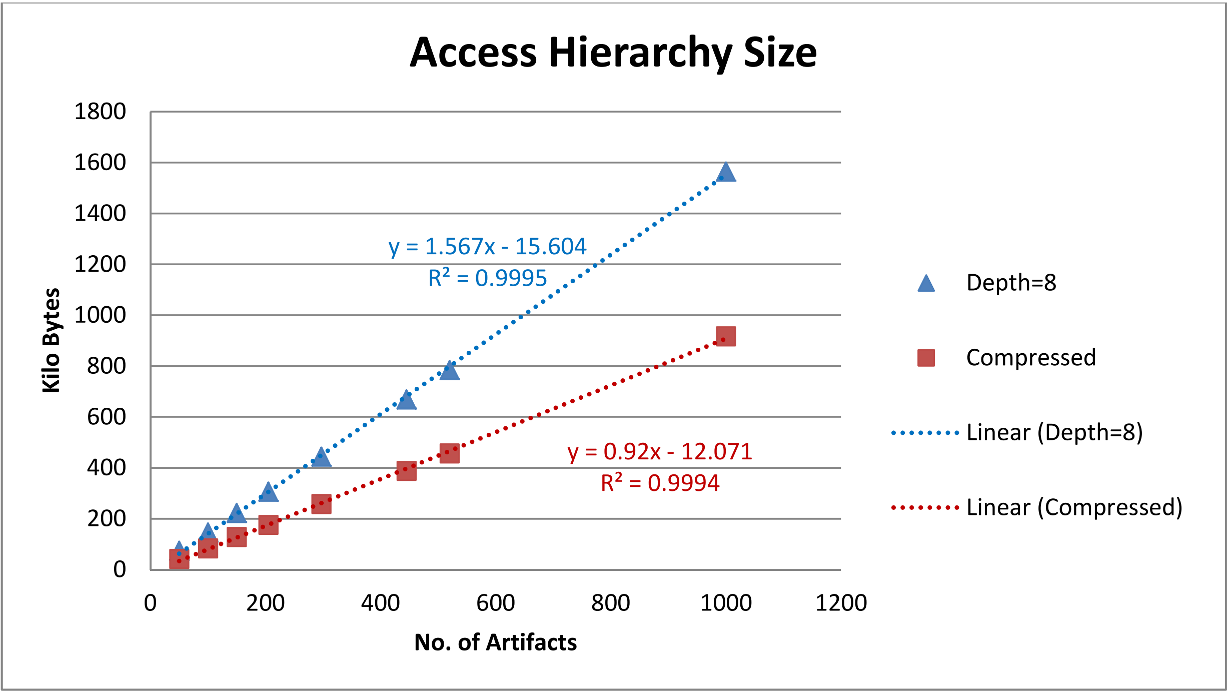


Figure 5-12: Access Hierarchy Size Graph

5.2.1.3 Access Hierarchy Update Cost

The update cost of the Access Hierarchy is directly depended on the depth of the hierarchy. Suppose the access hierarchy has a depth of N, then atmost N artifacts will be modified. The remote users have to update these N artifacts in order to update their corresponding Dynamic access views of their contacts. So, there are two costs to be considered, first the network overhead of sending the N artifacts to the remote repository and secondly the processing time to update the local hierarchy. For simplicity the network cost is considered constant so the maximum overhead in the first case is simply N Update Messages. The second cost is shown in the Figure 5-13.

In Figure 5-13, at each depth 3 artifacts are added with caching enabled and disabled. When caching is disabled all the artifacts are loaded from the database, and then once the update tag is updated they are again inserted into the database. This is showing a linear increase as the depth increases. When caching is enabled, the increase in the processing time is logarithmic instead of linear. This is due to the fact that artifacts are loaded from the cache instead of calling the database each time; but still the insert I/O operation is required. The update cost can be decreased by loading the access hierarchy into memory when the client boots and storing it in the database only when the user closes their client. However, this approach has the added risk of losing very important information in the event of a node failure or if the user does not close the client properly.

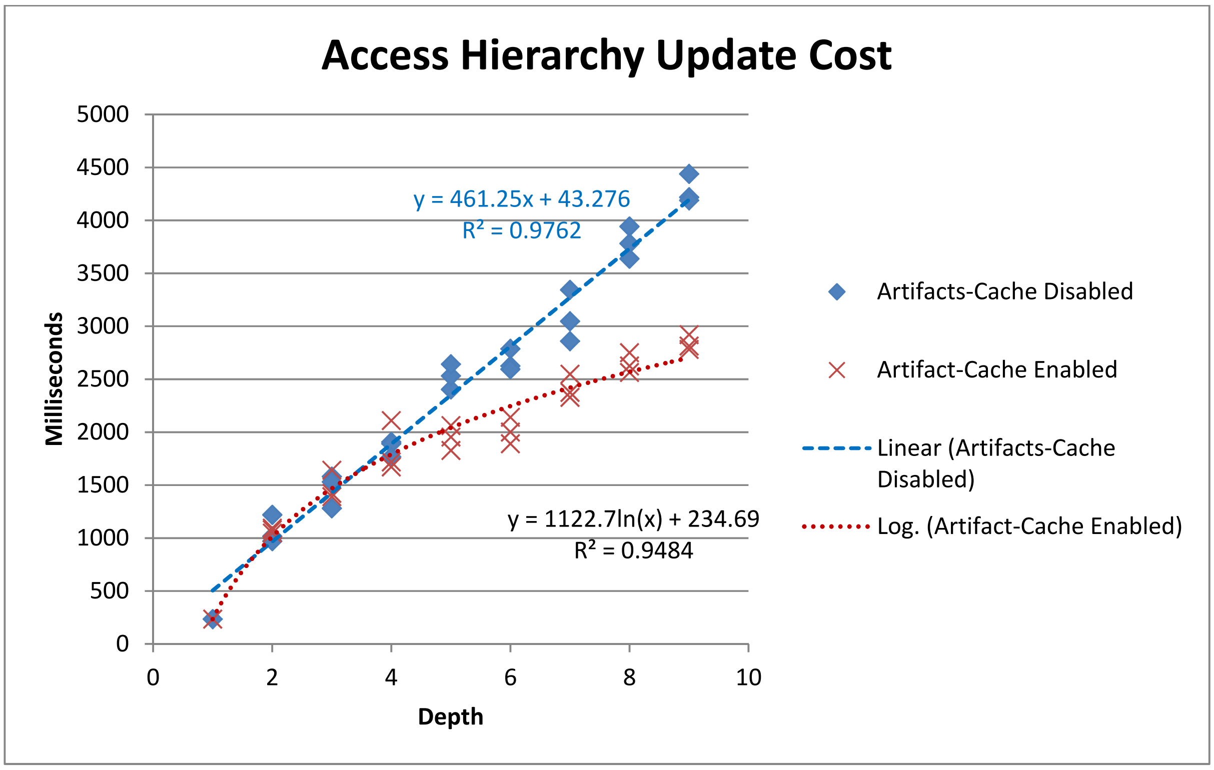


Figure 5-13. Access Hierarchy Update Cost Graph

5.2.1.4 Response Time

The response time is measured with respect to the remote user, the amount of time needed to detect the change (i.e., the response time) is:

$$Responce\ Time = Host\ Cost + Requestor\ Cost$$

Where:

$$Host\ Cost = Local\ Hierarchy\ Update + Push\ N\ Verital\ Artifacts\ to\ Remote\ Database$$

$$Requestor\ Cost = Download\ N\ Vertical\ Artifacts\ \&\ M\ Horizontal\ Artifacts + Processing\ Cost$$

The response is dependent on the horizontal cost, which corresponds to the number of child artifacts at a given depth and a vertical cost which is proportional to the depth. For the host there is no horizontal cost, as the addition of an artifact has no impact on the artifacts at that level, only vertical artifacts are modified to reflect the changes at the root; whereas on the requestor side there is a horizontal cost. When the requestor reads the Profile Root to detect a change, if the weight is different then it will download the child artifacts of the root, and detect which branch has changed and this continue this process until reaching depth N, so this is the basis of the horizontal cost.

The simulation results are shown in the Figure 5-14, where the update frequency of the remote profile view was set to 1,000 milliseconds and the network delay was simulated as 1,000 milliseconds. At each depth level, 3 artifacts are added at an interval of 10 seconds. The response time is calculated from the time the artifact is added into the access hierarchy until it is received by the remote user. The Access Controller is using the Mock Safebook Communication Manager to push the artifacts into a simulated remote database, which adds the network delay on each transaction. Figure 5-14 shows a linear increase in the response time with respect to the depth of the access hierarchy. These results show that the response time with a depth of 8 is less than 30 seconds. This should be sufficient for providing users with secure updates of the status of their friends in a distributed social network.

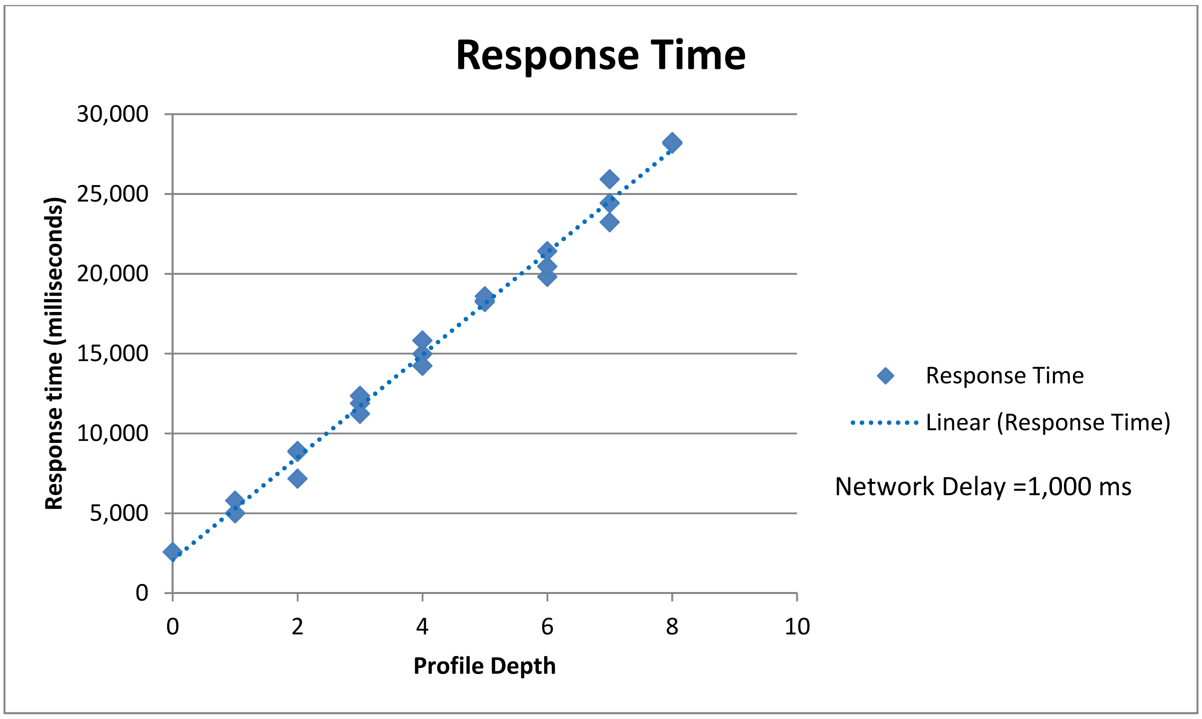


Figure 5-14: Response Time for the Access hierarchy with simulated network delay of 1000 milliseconds

6 Conclusion, Future Work & Reflections

This chapter summarizes the complete thesis work with a brief conclusion of the proposed solution along with some recommendation for possible improvements.

6.1 Conclusion

The privacy settings in the current OSNs are not sufficient to enable a user to define access control at a fine grain level. Moreover, users are forced to trust the SNS providers to protect their personal contents, from which the SNS provider could potentially gain profits by sharing with advertising companies and others. In this thesis we have demonstrated that an alternative to traditional centralized OSNs is a more secure distributed OSN, where each user is responsible for the distribution and control of his or her personal content with very fine grained access rights. The thesis defined an access control framework for Safebook, a distributed solution for online social network leveraging real life trust.

The proposed access control framework defines the user profile as a hierarchy of artifacts, where an artifact is the smallest possible logical entity in the Safebook network for which access control can be defined. The connectivity information and the resource contents of the artifact are categorized into Access Artifact and Resource Artifacts. The profile hierarchy is represented with the access artifacts while the child artifact identifiers are stored using dynamic access chains. These chains contain a string encrypted with multiple access keys using a stream cipher to hide the connectivity links. A requestor can only decrypt the link from the dynamic access chain whose access key he or she holds. In this ways a number of dynamic views can be created of the same profile hierarchy based on the number of access keys exposed.

The access keys associated with corresponding access and resource artifacts are in different key spaces and the mapping between the keys is only shared with the desired users, or in other words only those contacts the user wants to share their content with. This gives flexibility in the design of a revocation scheme where simply changing the access key for an Access Artifact and changing the Identifier for an artifact is sufficient to revoke access and avoids the need for re-encryption of all the resource artifacts (as these artifacts can be huge in size) and the redistribution of these artifacts can be very costly. Moreover, the thesis identifies the access control requirements for any social network and demonstrates several of them within the proposed access control framework.

The feasibility of the solution was demonstrated by implementing and simulating several different test cases. These test cases allowed us to measure the response time, access hierarchy update cost, and encryption cost. The response time for detecting a change in the remote profile at depth 8 is less than 30 seconds which is a reasonable delay for OSNs. Moreover, the update cost and encryption cost are less than 5 seconds at depth 8 and less than 250 milliseconds for 1,000 artifacts with caching enabled (respectively). These results indicate that the access control framework shows acceptable performance results while providing possibility to define fine grained access rights and moreover, the ownership of data is always preserved by design.

6.2 Future work

In the current implementation the database calls are not very optimized, hence the update process could be improved by calling the database calls at the end of the process in a single I/O operation. The artifacts can be chained into a single SQL command, which will be more efficient than calling the database separately for each artifact. Same is the case for the case when updating artifacts in the remote database, as the access artifacts are small in size, they can be easily chained into a single message and thus reduce the number of message exchanges between the core and mirror nodes. Moreover, the size of the access hierarchy can be reduced by classifying the artifacts in the form of bundles instead of individual artifacts. A classified bundle will be a single entity representing a number of access artifacts.

The performance of the access control mechanism can be improved further by using indexing over the access hierarchy to reduce the exchange of messages. The index tree can target a specific branch to update rather than always updating the root, which causes the exchange of addition messages. In this technique it will be useful to classify the branches in the profile hierarchy in terms of those which are more prone to changes and thus the update frequency for the different branches can be optimized. For example, if a notification service is added, the notification branch will need a higher frequency of updates in order to rapidly propagate these updates and then based on the updates received; specific branches can be directly updated instead of invoking the update process from the root. Moreover, the update tag contains information regarding the access keys, so this information can be used to reduce the horizontal cost by simply requesting only the branches which correspond to the change with respect to the access key.

Furthermore, the study can be extended to demonstrate other use cases identified in the thesis for the access control scenarios in the social networks. The results generated in this thesis should be further analysed and compared with other access control scheme. However, this comparison was out of the scope of this study.

6.3 Reflections

This thesis was motivated and encouraged by the community concerns over the privacy issues in the current Online Social Networks, which has become a de facto source of sharing life experiences. Over the course of study, I worked with the Safebook team and we identified several social aspects of user privacy along with customer needs and what they expect from a social network. This has been a step by step learning phase for me and I had a great opportunity to contribute in the improvements of the Safebook open source API. The research in this thesis address the importance of social and ethical values of user personal contents and provides a comprehensive solution to protect and control the distribution in a distributed paradigm. The solution is aimed to have a flexible access control scheme which coops up with the customer needs and I believe this thesis is a good contribution for the community.

References

[1] Leucio Antonio Cutillo, Mark Manulis, and Thorsten Strufe, ‘Security and Privacy in Online Social Networks’, in Handbook of Social Network Technologies and Applications, vol. Part 4, Borko Furht, Ed. Boston, MA: Springer US, 2010, pp. 497–522.

[2] Danah M. Boyd and Nicole B. Ellison, ‘Social Network Sites: Definition, History, and Scholarship’, *Journal of Computer-Mediated Communication*, vol. 13, no. 1, pp. 210–230, October 2007, DOI:10.1111/j.1083-6101.2007.00393.x.

[3] Leucio Cutillo, Refik Molva, and Thorsten Strufe, ‘Safebook: A privacy-preserving online social network leveraging on real-life trust’, *IEEE Communications Magazine*, vol. 47, no. 12, pp. 94–101, December 2009, DOI:10.1109/MCOM.2009.5350374.

[4] Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda, ‘All your contacts are belong to us’, presented at the Proceedings of the 18th international conference on World wide web WWW ’09, 2009, pp. 551–560, DOI:10.1145/1526709.1526784, Available at http://portal.acm.org/citation.cfm?doid=1526709.1526784.

[5] Tom N. Jagatic, Nathaniel A. Johnson, Markus Jakobsson, and Filippo Menczer, ‘Social phishing’, *Communications of the ACM*, vol. 50, no. 10, pp. 94–100, October 2007, DOI:10.1145/1290958.1290968.

[6] Emily Steel and Geoffrey A. Fowler, ‘Facebook in Online Privacy Breach: Top-Ranked Applications Transmit Personal IDs, a Journal Investigation Finds’, 17-October-2010. [Online]. Available: http://online.wsj.com/article/SB10001424052702304772804575558484075236968.html. [Accessed: 27-November-2012].

[7] Michael Arrington, ‘Modeling The Real Market Value Of Social Networks | TechCrunch’, 23-June-2008. [Online]. Available: http://techcrunch.com/2008/06/23/modeling-the-real-market-value-of-social-networks/. [Accessed: 30-July-2012].

[8] Michael Barbaro and Tom Zeller Jr., ‘A Face Is Exposed for AOL Searcher No. 4417749 - New York Times’, *New York Times*, NY, NY, USA, 09-August-2006, Available at http://www.nytimes.com/2006/08/09/technology/09aol.html?pagewanted=all.

[9] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, ‘Basic concepts and taxonomy of dependable and secure computing’, *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, January 2004, DOI:10.1109/TDSC.2004.2.

[10] Petar Maymounkov and David Mazières, ‘Kademlia: A Peer-to-Peer Information System Based on the XOR Metric’, in *Peer-to-Peer Systems*, vol. 2429, Peter Druschel, Frans Kaashoek, and Antony Rowstron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 53–65.

[11] Petar Maymounkov and David Mazières, ‘Kademlia: A Peer-to-Peer Information System Based on the XOR Metric’, in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 2002, pp. 53–65, DOI:10.1007/3-540-45748-8_5.

[12] Xukai Zou, Yuan-Shun Dai, and Yi Pan, *Trust and Security in Collaborative Computing*, vol. 2. World Scientific Publishing Company, 2008, ISBN: 981-270-368-3, 978-981-270-368-2.

[13] E. Damiani, S. De Capitani di Vimercati, and P. Samarati, ‘New paradigms for access control in open environments’, presented at the Fifth IEEE International Symposium on Signal Processing and Information Technology, 2005, Athens, Greece, 2005, pp. 540–545,

DOI:10.1109/ISSPIT.2005.1577155, Available at http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1577155.

[14] Piero A. Bonatti and Pierangela Samarati, 'A uniform framework for regulating service access and information release on the Web', *Journal of Computer Security*, vol. 10, no. 3, pp. 241–271, January 2002.

[15] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia, 'A logic-based framework for attribute based access control', in *Proceedings of the 2004 ACM workshop on Formal methods in security engineering FMSE '04*, 2004, pp. 45–55, DOI:10.1145/1029133.1029140, Available at http://portal.acm.org/citation.cfm?doid=1029133.1029140.

[16] E. Yuan and J. Tong, 'Attributed based access control (ABAC) for Web services', presented at the IEEE International Conference on Web Services, 2005. ICWS 2005, 2005, DOI:10.1109/ICWS.2005.25, Available at http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1530847.

[17] Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman, 'A Flexible Attribute Based Access Control Method for Grid Computing', *Journal of Grid Computing*, vol. 7, no. 2, pp. 169–180, November 2008, DOI:10.1007/s10723-008-9112-1.

[18] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dev, Mustaque Ahamad, and Gregory D. Abowd, 'Securing context-aware applications using environment roles', presented at the Sixth ACM symposium on Access control models and technologies SACMAT '01, 2001, pp. 10–20, DOI:10.1145/373256.373258, Available at http://portal.acm.org/citation.cfm?doid=373256.373258.

[19] Filipe Beato, Markulf Kohlweiss, and Karel Wouters, 'Enforcing Access Control in Social Network Sites', presented at the The 9th Privacy Enhancing Technologies Symposium 2009 (PETS 2009), Seattle, WA, USA, 2009, p. 13 pages, Available at http://www.cosic.esat.kuleuven.be/publications/article-1240.pdf.

[20] Barbara Carminati, Elena Ferrari, and Andrea Perego, 'Rule-Based Access Control for Social Networks', in *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, vol. 4278, Robert Meersman, Zahir Tari, and Pilar Herrero, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1734–1744.

[21] Barbara Carminati, Elena Ferrari, and Andrea Perego, 'Private Relationships in Social Networks', presented at the IEEE 23rd International Conference on Data Engineering Workshop, 2007, Istanbul, Turkey, 2007, pp. 163–171, DOI:10.1109/ICDEW.2007.4400987, Available at http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4400987.

[22] Josep Domingo-Ferrer, 'A Public-Key Protocol for Social Networks with Private Relationships', in *Modeling Decisions for Artificial Intelligence*, vol. 4617, V. Torra, Yasuo Narukawa, and Yuji Yoshida, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 373–379.

[23] E. Palomar, J.M. Estevez-Tapiador, J.C. Hernandez-Castro, and A. Ribagorda, 'Certificate-based Access Control in Pure P2P Networks', presented at the Sixth IEEE International Conference on Peer-to-Peer Computing, 2006 (P2P 2006), Cambridge, 2006, pp. 177–184, DOI:10.1109/P2P.2006.7, Available at http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1698608.

[24] Kiran K. Gollu, Stefan Saroiu, and Alec Wolman, 'A Social Networking-Based Access Control Scheme for Personal Content', presented at the ACM SIGOPS 21st Symposium on Operating

Systems Principles 2007 (SOSP ’07), Stevenson, Washington, USA, 2007, p. 1 page, Available at http://www.cs.toronto.edu/~kkgollu/docs/kkgollu_access.pdf.

[25] Jonathan Anderson, Claudia Diaz, Joseph Bonneau, and Frank Stajano, ‘Privacy-enabling social networking over untrusted networks’, presented at the The 2nd ACM workshop on Online social networks (WOSN ’09), 2009, pp. 1–6, DOI:10.1145/1592665.1592667, Available at http://portal.acm.org/citation.cfm?doid=1592665.1592667.

[26] Christoph Sturm, Klaus R. Dittrich, and Patrick Ziegler, ‘An access control mechanism for P2P collaborations’, in *Proceedings of the 2008 international workshop on Data management in peer-to-peer systems*, Nantes, France, 2008, pp. 51–58, DOI:10.1145/1379350.1379358.

[27] Imen Ben Dhia, ‘Access control in social networks’, in *Proceedings of the 2012 Joint EDBT/ICDT Workshops (EDBT-ICDT ’12)*, 2012, p. 227, DOI:10.1145/2320765.2320828, Available at http://dl.acm.org/citation.cfm?doid=2320765.2320828.

[28] Talel Abdessalem and Imen Ben Dhia, ‘A reachability-based access control model for online social networks’, presented at the Databases and Social Networks (DBSocial ’11), 2011, pp. 31–36, DOI:10.1145/1996413.1996419, Available at http://portal.acm.org/citation.cfm?doid=1996413.1996419.

[29] SR 107: JCACHE - Java Temporary Caching API, Oracle Corporation, Specs Available at http://jcp.org/en/jsr/detail?id=107

[30] Ehcache, Open Source Project, Available at http://ehcache.org/documentation

TRITA-ICT-EX-2013:199