

CERIAS Tech Report 2014-8
PRIVACY IN SOCIAL MESSAGING AND IDENTITY MANAGEMENT
by Ruchith Fernando
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By RUCHITH UDAYANGA FERNANDO

Entitled
PRIVACY IN SOCIAL MESSAGING AND IDENTITY MANAGEMENT

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Bharat Bhargava, Ph.D.

Samuel S. Wagstaff, Jr., Ph.D.

Christopher W. Clifton, Ph.D.

Leszek T. Lilien, Ph.D.

To the best of my knowledge and as understood by the student in the *Thesis/Dissertation Agreement, Publication Delay, and Certification/Disclaimer (Graduate School Form 32)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Bharat Bhargava, Ph.D.

Approved by Major Professor(s): _____

Approved by: William J. Gorman, Ph.D.

07/10/2014

Head of the Department Graduate Program

Date

PRIVACY IN SOCIAL MESSAGING AND IDENTITY MANAGEMENT

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Ruchith Udayanga Fernando

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2014

Purdue University

West Lafayette, Indiana

To my parents.

ACKNOWLEDGMENTS

I would like to, first and foremost, express my deepest appreciation to my advisor, Professor Bharat Bhargava. You have been a great mentor to me, and this work would not be possible without your advice, encouragement, and support.

I would like to thank my committee members, Professor Samuel Wagstaff, Professor Chris Clifton and Professor Leszek Lilien for all your valuable feedback and encouragement.

A big thank you to Dr. Nabeel Yoosuf, Dr. Pelin Angin, Dr. Lotfi ben Othmane and Rohit Ranchal for your collaboration and support.

I would especially like to thank Dr. Sanjiva Weerawarana, for his encouragement for me to pursue my higher studies and incited me to strive towards my goals. In addition, a thank you to Ann Christine Catlin, who provided me a graduate assistant position for the full duration of my study. Your support has helped me immensely.

I would like to give a big shout out to the Sri Lankan community at Purdue, who have stood by me and made my stay an enjoyable one. Your friendship means a lot to me.

Last, but certainly not least, I would like to thank my parents, my two brothers and my wife for supporting me throughout the years. You all have been my pillars of strength through all the tough times and made the good times even better.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vii
LIST OF FIGURES	viii
ABSTRACT	x
1 INTRODUCTION	1
1.1 Privacy in Social Messaging	4
1.2 Privacy in Identity Management	5
1.3 Thesis Statement	6
2 PRIVATE ANONYMOUS MESSAGING	7
2.1 Introduction	7
2.2 Problem Definition	9
2.3 Related Work	10
2.4 Preliminary Notions	12
2.4.1 Hierarchical Identity Based Encryption	12
2.5 Proposed Solution	14
2.5.1 Peer Setup	15
2.5.2 Registering a Contact	15
2.5.3 A Contact Requesting an Update	15
2.5.4 Encryption and Update Response	16
2.5.5 Decryption of the Update	16
2.5.6 Peer Re-key	17
2.6 Security Evaluation	18
2.6.1 Update Request	19
2.6.2 Update response	20
2.6.3 Re-key	20
2.6.4 Peer Unlinkability	21
2.6.5 Event Causality	21
2.6.6 Protection Against Malicious Peers	22
2.6.7 Lifecycle of an Update Request	24
2.6.8 Verification of Response Generators	25
2.7 Implementation	26
2.7.1 Library	26
2.7.2 Proof of Concept Application	28
2.7.3 Experiment Framework	29

	Page
3 PRIVACY IN IDENTITY MANAGEMENT	35
3.1 Introduction	35
3.2 Problem	38
3.2.1 Overview	38
3.2.2 Problem Definition	39
3.3 Related Work	41
3.4 Solution	43
3.4.1 Generate a Claim (GenClaim)	44
3.4.2 Issue a Claim (IssueClaim)	44
3.4.3 Authenticate with a Claim	47
3.4.4 Authenticate with Multiple Claims	48
3.4.5 Issued Claim Revocation	51
3.5 Security Evaluation	54
3.5.1 Claim Definition	54
3.5.2 Claim Issuance	54
3.5.3 Authentication	56
3.5.4 Claim Revocation	56
3.5.5 Collusion Resistance	58
3.6 Implementation	59
3.6.1 Identity Provider	59
3.6.2 Identity Provider Client	60
3.6.3 Service Provider and Clients	61
3.6.4 Performance Evaluation	61
3.7 Integration with Service Compositions	67
3.7.1 Direct User Authentication	67
3.7.2 Service Provider Capabilities	69
4 CONSUMER ORIENTED PRIVACY PRESERVING ACCESS CONTROL FOR ELECTRONIC HEALTH RECORDS	70
4.1 Introduction	70
4.2 Motivation	72
4.3 Related Work	74
4.4 Preliminary Notions	76
4.4.1 IssueClaim	77
4.4.2 Authenticate	77
4.4.3 RevokeClaim	77
4.5 Proposed Solution	78
4.5.1 Initial Setup	78
4.5.2 Access Results	80
4.5.3 Access Revocation	81
4.5.4 HIE Access Policies	82
4.6 Evaluation	83

	Page
4.6.1 Privacy of Authentication Protocols	83
4.6.2 Privacy of Access Revocation	83
4.6.3 Value Added Services	84
5 REVOCATION OF CIPHERTEXT POLICY ATTRIBUTE BASED EN- CRYPTION KEYS	85
5.1 Ciphertext Policy Attribute Based Encrytion Keys	86
5.2 Revocation	86
5.2.1 Security Evaluation	88
6 SUMMARY	89
LIST OF REFERENCES	91
VITA	97

LIST OF TABLES

Table	Page
2.1 Operations of the public channel	30
3.1 Example set of claim public data published by an identity provider . .	45
3.2 An example of how the c_0' values are stored by an identity provider . .	52
3.3 An example of how the new set of c_0 values and new public parameter component g_1 are published by an identity provider	57
3.4 Claim definition creation time analysis	62
3.5 Claim instance issue time analysis	63
4.1 Entities and notations	74
4.2 Information stored for each user upon claim issuance	79
4.3 Example of published re-key information	81
5.1 User key information stored at the issuer	87
5.2 Re-key information to be published	88

LIST OF FIGURES

Figure	Page
1.1 Concept of a personal server that hosts all applications	3
2.1 A peer and her contacts	7
2.2 Contacts without the presence of the peer	8
2.3 Example of contact revocation action during update propagation	23
2.4 A screenshot of the demo application	29
2.5 Sample output of public channel messages	31
2.6 Example shell script to start an interact with a set of peers	34
3.1 Bob authenticating multiple times with Alice using his credentials . . .	38
3.2 Charlie authenticating multiple times with Alice using his credentials .	39
3.3 Typical setup of an identity management system	40
3.4 Identity provider using the request value given by a user to issue a claim	46
3.5 User uses a claim to generate an <i>AnonClaim</i> to initiate an authenticated session with the service provider	47
3.6 An example access structure and one possible technique to propagate the shares of a secret from the root node to the leaves	49
3.7 Design of the identity provider database. Text in bold indicates the primary keys of each entity.	60
3.8 Example of idptool usage	60
3.9 Setup of Amazon EC2 remote invocation experiment	63
3.10 Stacked times to authenticate with a service provider	65
3.11 Stacked times to authenticate with a service provider using concurrent key derivation	66
3.12 Comparison of key derivation times	66
3.13 A service composition with expected claims	67
4.1 Medical information exchange setup	72

Figure	Page
4.2 Entities of an identity management system	76
4.3 User setting up the claim definitions and issuing a read claim to a doctor	78
4.4 Lab sending an electronic health record to a health information exchange	79
4.5 Doctor authenticating with a health information exchange	80

ABSTRACT

Fernando, Ruchith Udayanga Ph.D., Purdue University, August 2014. Privacy in Social Messaging and Identity Management. Major Professor: Bharat Bhargava.

Messaging systems, where a user maintains a set of contacts and broadcasts messages to them, are very common. In a situation where a user only sends messages directly to a set of online contacts, a contact might miss a message if it is not available to receive it directly from the user. This work addresses the problem of a trusted contact's obtaining a message that it missed, from other trusted contacts of the user, while maintaining the anonymity of all participating contacts. A protocol is presented to facilitate this communication. An experimental framework is developed to evaluate various possible configurations of the entities involved.

The techniques developed to address the above problem are extended to address the problem of a user's authenticating with a service provider while ensuring that multiple sessions are unlinkable. The proposed approach achieves this by setting up an authenticated secure channel between the user and the service provider. Information exchanged for the setup of this secure channel is unique over multiple authentications. The proposed protocol is further enhanced to accommodate service provider policies that use credentials with relationship constraints among them. In such cases, the service provider will not be able to analyze and identify sets of users who authenticate with different credential subsets. The proposed credential revocation scheme allows an identity provider to revoke user credentials without compromising user privacy, even while relying on a public channel. Moreover, these protocols do not require the identity provider to remain online during authentication and revocation. Finally, details on how to adapt the proposed identity management system to privately manage healthcare records is presented as an application of the proposed system.

1 INTRODUCTION

While most of the web applications on the Internet interface with the user in a traditional client-server model, user privacy always remains as one of the non-functional requirements. User personal information is consumed by all such services. For example, when an individual attempts to sign up for a social networking application the service provider obtains his/her name and date of birth. Furthermore, when a user accesses these services via a mobile device, the service provider captures the sensor data related to user's location and other behavioral aspects. All of this information collected about users leads to profiling using novel machine learning techniques and can be used to infer more intimate personal information [1] [2] [3].

Service providers make the users agree to a terms of service or an end user license agreement when they sign up for their services. Moreover, agreements are drafted in such a way that allows the service providers to change those terms at will. Most importantly, as ignored by most users [4], the terms include clauses that specify how the user information may be used by the service provider. There are two types of user information collected by service providers. The first type is direct identity information that is obtained directly from the user. These may be collected at the point of registration or generated while using the service (E.g. images, written text). The second type of user information is derived information, such as the users' behavioral aspects as to how the users interact with the service. A service provider may share user information with other service providers. It is not possible for a user to foresee how his/her information may be exploited in any way in the future. Therefore, it is impossible to clearly determine what other disclosures are possible with information that is leaked at any point.

The number of attacks on web applications has increased [5] [6]. There are various insider and external attacks on online services. An insider attack may be as simple

as a disgruntled employee stealing a database of user information and disclosing it publicly or selling it to another party. External attacks are cases where an attacker or a group of attackers maliciously access service provider infrastructure to obtain user information or disrupt normal operation of services. These types of attacks are carried out by exploiting various vulnerabilities in service infrastructure and even using social engineering attacks on employees of the service [7]. Attacks continuously mounted on various web applications by organized groups, such as "Anonymous" [8], are good examples of external attacks on online services.

Another issue, due to the current architecture of the web applications, is the susceptibility of these services to censorship by various authorities. Government can sensor information available on the web. Service providers have mechanisms in place that allow an authority to dispute some user content (E.g. images, video) , enabling the service provider to remove that content even if the user is well within his/her rights to use the content in question in a manner he/she wishes. In many cases, the access to the Internet is controlled by the government which has control over the Internet service providers [9] [10] [11].

It is important to understand the legal obligations of service providers. Even if the service providers promise complete privacy to users and to never use their personal information for anything other than the explicit needs of the service provided, the service providers have to abide by the law. For example, consider a web based email service. The contents of users' emails are stored in databases of the service provider. Currently in the United States, under the 1986 Electronic Communications Privacy Act, a party will only require a warrant to access emails of a user if those emails are less than six months old. All emails prior to six months, at any point, stored by the service provider can be released, without a warrant, to an investigator. Such a policy is a threat to user privacy.

Under these circumstances, one of the best practices when using the Web would be to have no expectations of privacy at all. In this context, the user would be aware of the fact that any information that he/she generates or provides would be

public information at some point. This however is not practical. In searching for an answer to these privacy issues, it is interesting to note the concept of "FreedomBox" by Eben Moglen [12], a professor of law and legal history at Columbia University. According to Wikipedia *"The FreedomBox is an affordable personal server which runs only free software, with a focus on anonymous and secure communication"*. This concept attempts to bring the service implementation closer to the user, where the user will have more control on usage of his/her information.

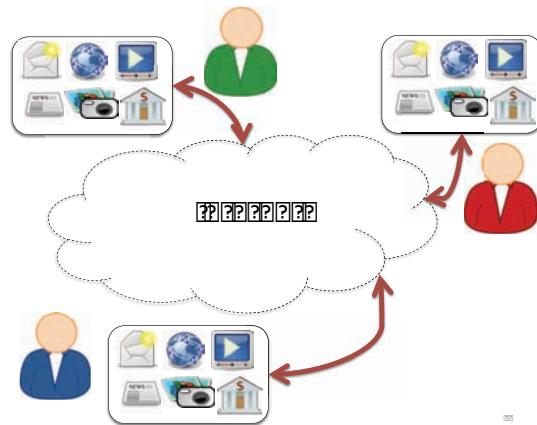


Figure 1.1. Concept of a personal server that hosts all applications

With this concept, the usage information of the service is confined to the boundaries of the user who is given significant control over how information is used. It is however interesting to note that users will still have to communicate over the untrusted Internet. For example, a user may be required to authenticate with another user. In such a situation, it is important to transfer identity information over the Internet in a privacy preserving manner. This research work explores privacy in social messaging in this model and develops a set of protocols. Moreover, this work utilizes those techniques to develop a set of authentication protocols to address a set of privacy issues in identity management.

1.1 Privacy in Social Messaging

One of the main use cases of online social networking platforms is the exchange of messages between users and their contacts. Social networks proved critical in several of the uprisings that had major political significance in the world. The problem of messaging addressed here is closely related to such scenarios.

There were several instances in the past where an authority was able to limit the propagation of information by disrupting access to the Internet, which prevented users from using popular social media applications [9] [10] [11]. In addressing this concern, the following requirements were identified for a messaging system that ensures user privacy and anonymity. A user in this system will have a set of personal contacts, who may not be known by each other, that wish to maintain their association with this particular user anonymous. This is equivalent to a whistleblower maintaining a list of journalist contacts, or a spy agency which has a set of spies all around the world. In both these cases all journalist and spies have no intention of revealing their associations. In this setup the main user needs to directly communicate a message securely only to his/her contacts. It is important to note that the user may only be able to propagate the message *directly* to only a subset of his/her contacts and may lose the ability to communicate any further. In such circumstance, the set of contacts who did not receive the message directly need to obtain the message via a public communications infrastructure.

In addressing this problem, a novel approach is proposed in which those users awaiting the message posts request to a public channel. The other contacts respond to these requests with the guarantee that by participating in this message exchange, no third party will be able to infer their identity or association with any party. A modification to the hierarchical identity based encryption [13] was used to achieve this. This scheme further allows the list of trusted contacts to be dynamic. Revocation of a contact is performed solely using public information that does not leak identity information about any of the participants.

1.2 Privacy in Identity Management

Digital identity management is the management of users' digital identity attributes for authentication and authorization purposes. There has been a significant amount of work towards ensuring user privacy in identity management. One of the foundations of this work is the laws of identity [14]. Windows CardSpace [15] by Microsoft and OpenID [16] are some of the most notable efforts that were based on the ideas of an identity metasystem proposed by the laws of identity. However, these platforms suffered from various privacy drawbacks. For example, implementations of these systems did not allow users to prove mere possession of a identity attributes without the potential of being profiled. There has been considerable research carried out [17], [18] in addressing some of these privacy concerns. These systems support features such as authentication using multiple attributes and authentication using attributes from different identity providers. Here an approach is proposed to solve these problems with new privacy features.

The proposed identity management system allows users to obtain certified credentials from various identity providers. Additionally, those users will be able to use those certified credentials for authentication and authorization purposes with service providers with the guarantees of unlinkability of transaction. Furthermore, users will be able to use credentials to efficiently satisfy complex authentication and authorization policies set by a service provider with the same level of privacy guarantees. Users will be able to mix and match credentials from different identity providers. The service providers are guaranteed that a user will not be able to collude with another to satisfy a service provider policy that involves multiple credentials.

1.3 Thesis Statement

The thesis statement is expressed in two main points:

- It is possible to propagate messages from a user to his/her trusted peers via a subset of the peers who have access to those messages, while maintaining anonymity of all participating peers.
- It is possible to establish an unlinkable authenticated secure channel between a user and a service provider without an online trusted third party, while adhering to service provider policies that contain identity claim relationships.

2 PRIVATE ANONYMOUS MESSAGING

2.1 Introduction

Social media services such as weblogs, online social networks and microblogging applications are being used to broadcast opinions. These services are susceptible to censorship and suppression by various authorities. Such control is feasible due to the current architecture of these services where they primarily adopt a client-server model. In the client-server model, services are hosted remotely and users contribute content these services but do not have direct control over this data. Any entity that can control the server that maintains data of such a service, also has complete control over the data.

It is interesting to explore possible approaches that can maintain the power of social media while being able to resist control by third parties. One such approach is discussed here.

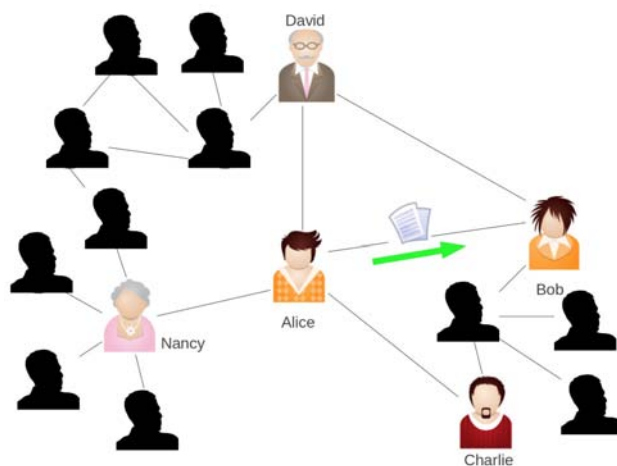


Figure 2.1. A peer and her contacts

Consider a peer-to-peer setup, where the peers are *only* connected to other peers who they personally trust. These are called *contacts* of a peer. A peer distributes a message to its contacts, (called an *update*) and all of the peers are expected to receive this update. A peer may directly communicate the message when a contact is available online. When there is only a subset of contacts available to the peer to directly communicate with, the peer sends any updates to one or more of those contacts. The set of contacts that need to obtain any missed updates of the peer, need to obtain these updates from contacts that already have the updates. This is necessary because, when those contact are available, the peer may not be available to directly connect with them and obtain updates. These updates must be obtained without revealing any identity information of any of the contacts. A novel cryptographic approach is presented here as the solution to this problem.

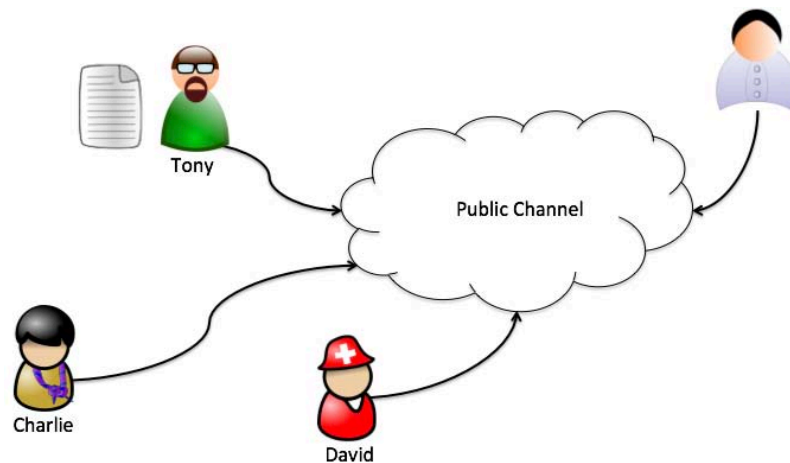


Figure 2.2. Contacts without the presence of the peer

Section 2.2 defines the problem and the requirements that need to be satisfied. Section 2.4 outlines the underlying techniques used in the proposed solution. The details of the proposed protocols are described in Section 2.5 followed by a detailed security analysis of the protocols. Section 2.7 presents the implementation details of the proposed cryptographic primitives.

2.2 Problem Definition

A peer in this system is a user who has a set of other peers registered with it as contacts. This peer registration is bi-directional. In other words, when peer A becomes a contact of peer B, peer B becomes a contact of peer A.

A peer intends to send messages to all its contacts. All of these messages are to be delivered to its contacts at that point of time. This is similar to the notion of microblogging (e.g., Twitter). Such a message is identified as an *update*.

Denote the peer generating an *update* as P and its contacts as the set $C = \{C_{P_i}\}$ where $i \in \{1, \dots, n\}$ where n is the number of contacts of P .

The problem of a contact obtaining an update that it missed anonymously is defined as follows:

- A peer P should be able to send its *update* M_P only to those contacts who are available online at the point of time it sends the update using direct connections to those peers. Denote the set of online contacts as $C^+ \subseteq C$ where $|C^+| \geq 1$.
- Those other contacts of P who were off-line at when P sent M_P should be able to obtain M_P when they are available online. Denote these contacts as C^- . $C^- \subset C$.
- Any $C_{P_i} \in C^-$ will be able to publish a query requesting an update of P . This is called an *update request* and is denoted by Q_P .
- Any $C_{P_i} \in C^+$ will be able to publish a response to a Q_P . This response is denoted by S_P , and an eavesdropper with polynomially bounded resources should not be able to compute the original M_P using S_P .
- The contact who provides S_P should not be able to learn who generated Q_P .
- The contact who generates Q_P and receives the corresponding S_P should be able to extract M_P but should not be able to learn who generated S_P .

- Only a member of C should be able to obtain the plaintext M_P using this protocol.
- When the composition of C changes to new set of peers C' with the removal of one or more contacts, P should be able to update private configuration of the members of C' with the issue of a public message.
- After such an update those peers in the set $C - C'$ should not be able to obtain an *update* of P .

For example let Alice be P . Alice has four contacts: Bob, Charlie, David and Nancy, out of which only Bob gets the *update* M_P . Therefore: $C = \{Bob, Charlie, David, Nancy\}$, $C^+ = \{Bob\}$ and $C^- = \{Charlie, David, Nancy\}$.

Note that a peer only trusts and has knowledge of its immediate contacts and is not aware of connections among those peers and their contacts. There are practical implementations of the notion of friend-only networks such as Freenet/Darknet [19] and GNUnet [20]. Furthermore, it is possible to rely on a public channel which does not maintain any information about clients that connect to it to publish messages. A client may connect to the public channel using onion routing [21] [22] to prevent any local eavesdroppers from identifying communications between clients and the channel.

2.3 Related Work

One can come up with a solution to this specific problem using public-key infrastructure. In this case, each user in this systems will be assigned a key pair, and P will be able to encrypt a message to all those contacts and publish each ciphertext. The users may retrieve these messages at will. This approach however is not attractive when P has a large number of contacts, due to computation, bandwidth, or timing issues. In such a scenario, it may be advantageous to push a single message to a contact using a technique as the one proposed here.

Another approach to this problem would be to use an anonymous proxy/group signature scheme [23]. In such schemes, all contacts become a proxy of P , and when they need an update message, they generate an ephemeral public key, sign as a proxy, and publish that as a request. Any contact can verify that the request is coming from a user within the group that belongs to P . This fact itself poses a problem for anonymity and privacy. This is where a third party having access to the public parameters of the signature scheme will be able to analyze request information per user. Furthermore, it is important to note that the signature scheme will have to be reset in the case of a change to the membership of the group of contacts, which may require the user to contact each contact directly using a private channel. In the solution proposed in this work, there is no requirement for a private channel and causal relationships between updating group membership, and message propagation actions can be established.

Broadcast Encryption [24] [25] is a technique that allows distribution of encrypted content to a set of subscribers. Multicast encryption [26], is another method for ensuring that only a set of entities in the network can decrypt a message. Privacy requirement in these schemes is that external parties are not able to decipher the messages. Specific requirement for anonymity between the nodes of the network is not addressed in these schemes.

Yi Lu et. al. [27] proposed a system that attempts to hide the association between the identity of a user and the data that user is interested in. Here a user relies on a proxy to obtain data from a supplier. This system relies on the trust of the proxy not to reveal the identities of the parties involved.

Instant messaging (IM) platforms, such as Internet Relay Chat [28], rely on a central server for a participant to connect and communicate. Also, there are other pseudo-peer-to-peer or hybrid systems, such as Skype [29]. All of these applications require users to provide identifiers when using these services and their usage can be monitored by the service provider. Also, due to recent service outage incidents, it is evident that these services are susceptible to be taken down by a motivated party.

There are efforts to make social networking more distributed without relying on a single centralized entity such as Google Plus or Facebook. One such effort is the Diaspora project [30]. This application allows users to setup their own installation of the social networking platform or join any of the existing installations. However, this still poses several problems since almost any attack that may be mounted on a centralized service provider may be mounted on any or those individual installations of the service. The proposed social messaging solution, only requires a public channel, and the information that is posted to this public channel will not leak identity information nor message data.

2.4 Preliminary Notions

A set of modifications to hierarchical identity based encryption scheme proposed by Boneh et. al. [13] is proposed in this work. The next section provides the necessary background about this scheme.

2.4.1 Hierarchical Identity Based Encryption

Identity based encryption, first proposed by Shamir [31], is a public key encryption scheme where the identity of an entity is used as the public key. The first complete solution for this was presented by Boneh et.al. [32]. Any party who intends to send a message to another will simply use a set of public parameters of a trusted authority along with the identity of the recipient to encrypt using this scheme. The recipient of the ciphertext will be able to obtain the corresponding private key from the third party (who executes private key generation algorithm for a given identity after authenticating the requester) and decrypt the ciphertext to obtain the plain text.

This idea of identity based encryption was extended to a hierarchy of identities by Horwitz et. al. [33] and Boneh et. al. [13]. These schemes allow a hierarchy of users. At each level the private key is used as an input to the key generation algorithm, along with the global parameters defined by the root, to generate private keys for users of

the next lower level. The Hierarchical Identity Based Encryption(HIBE) system by Boneh et. al. [13] is defined as follows:

Let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ be a bilinear map where \mathbb{G} is a group of prime order p . An identity is defined as $ID = (I_1, \dots, I_k) \in (\mathbb{Z}_p^*)^k$ where k is the depth of the hierarchy in which the ID belongs.

There are four algorithms: *Setup*, *KeyGen*, *Encrypt* and *Decrypt*. l is the allowed maximum depth of the hierarchy.

- *Setup*(l) generates the public parameters and the master key as follows:
 - Select a generator $g \in \mathbb{G}$ and a random value $\alpha \in \mathbb{Z}_p$
 - Set $g_1 = g^\alpha$
 - Pick random values $g_2, g_3, h_1, \dots, h_l \in \mathbb{G}$
 - $params = (g, g_1, g_2, g_3, h_1, \dots, h_l)$
 - $master-key = g_2^\alpha$
- *KeyGen*($d_{ID_{k-1}}, ID$) generates the private key of the given k^{th} level ID using a $k - 1$ level private key ($k \leq l$).

First suppose the $k - 1$ level private key was generated using the master key :

- Select a random value $r \in \mathbb{Z}_p$
- Output $d_{ID_{k-1}} = (a_0, a_1, b_k, \dots, b_l) =$

$$(g_2^\alpha \cdot (h_1^{I_1} \dots h_{k-1}^{I_{k-1}} \cdot g_3)^r, g^r, h_k^r, \dots, h_l^r)$$

Now the k^{th} level private key:

- Select a random value $t \in \mathbb{Z}_p$
- Output $d_{ID_k} =$

$$(a_0 \cdot b_k^{I_k} \cdot (h_1^{I_1} \dots h_k^{I_k} \cdot g_3)^t, a_1 \cdot g^t, h_{k+1}^t, \dots, h_l^t)$$

- $Encrypt(params, ID, M)$ encrypts a message $M \in \mathbb{G}_1$ using the public key $ID = (I_1, \dots, I_k)$:

- Select a random value $s \in \mathbb{Z}_p$
- Output $CT = (A, B, C) =$

$$(e(g_1, g_2)^s \cdot M, g^s, (h_1^{I_1} \dots h_k^{I_k} \cdot g_3)^s)$$

- $Decrypt(d_{ID}, CT)$ decrypts a given ciphertext of the above form (A, B, C) with the private key of the form $(a_0, a_1, b_k, \dots, b_l)$.

$$(A \cdot e(a_1, C)) / (e(B, a_0)) = M$$

2.5 Proposed Solution

A peer first registers a set of contacts and transmits an update directly to a subset of those contacts. The proposed set of protocols, allows a contact of a peer to request an update of from another contact. This request is made using a public channel that is monitored by other contacts of the peer. Another contact who has the update will generate a response and submit it as a response to the request. The original contact who posted the request will be able to process this response and obtain the update of the peer. In a situation where the peer removes a contact from his/her list of contacts, the re-key protocol updates the remaining peers' private information. This process ensures that the removed contact will not be able to obtain any further updates of the peer.

The details of setting up the parameters of a peer, registering contacts, contacts generating update requests to be processed by other contacts, update response to such a request and reset keys of a peer and how contacts update themselves, is discussed in this section.

2.5.1 Peer Setup

A peer, P , will setup parameters similar to a two level HIBE system parameters. This will generate the public parameters and the master key of the peer as follows:

- Select generators $g, g_2, h_1 \in \mathbb{G}$ and a random value $\alpha \in \mathbb{Z}_p$
- Set $g_1 = g^\alpha$
- Pick random values $g_3, h_2 \in \mathbb{G}$.
- $params = (g, g_1, g_2, g_3, h_1, h_2)$
- $master-key = g_2^\alpha$

2.5.2 Registering a Contact

When a peer P registers a C_{P_i} it will create a new random first level identifier $I_{r_{i1}} \in \mathbb{Z}_p$ and corresponding private key $(d_{I_{r_i}})$. The private key and the identifier will be communicated to C_{P_i} using a secure channel. $d_{I_{r_i}}$ is of the form $(g_2^\alpha \cdot (h_1^{I_{r_{i1}}} \cdot g_3)^{r_i}, g^{r_i}, h_2^{r_i})$, where the value $r_i \in \mathbb{G}$ is randomly chosen.

- C_{P_i} keeps both $I_{r_{i1}}$ and $d_{I_{r_i}}$ private along with the public parameters of P
- P stores the tuple $(I_{r_{i1}}, r_i)^1$

2.5.3 A Contact Requesting an Update

When P sends an *update* message, it may send the update directly to available contacts by encrypting the message using their corresponding identifiers. The interesting case is when a contact, $C_{P_{req}}$, needs to obtain the latest *update* of P and P is no longer available online. In such a situation, as highlighted by in the requirements,

¹This is used to update contact parameters in the case of a re-key.

$C_{P_{req}}$ will be able to generate a request for P 's update, Q_p . This is generated as follows:

Suppose the identifier assigned to $C_{P_{req}}$ by P is I_{r_1}

- Select a random value $I_{r_2} \in \mathbb{Z}_p$

- Set $ID_{req} = h_1^{I_{r_1}} \cdot h_2^{I_{r_2}}$

- Update Request to be published:

$Q_P = (PID, ID_{req})$, here PID is an identifier string of P known to all P 's contacts.

$C_{P_{req}}$ publishes (PID, ID_{req}) and any of P 's other contacts will be able to respond to this request. This request information can be made publicly available using a common medium. The steps of creating the response are described next.

2.5.4 Encryption and Update Response

When a contact of P observes the tuple (PID, ID_{req}) and decides to serve this request it will first encrypt the latest *update* message M_P from P using the following modified encryption function ($Encrypt'$) and P 's public parameters $params_P$.

$Encrypt'(params_P, ID_{req}, M_P) :$

- Select a random value $s \in \mathbb{Z}_p$
- $CT_{resp} = (e(g_1, g_2)^s \cdot M, g^s, (ID_{req} \cdot g_3)^s) = (A, B, C)$

The contact publishes the tuple $(PID, ID_{req}, CT_{resp})$ as the response S_P .

2.5.5 Decryption of the Update

The contact that generated the update request will obtain the response from the public channel. Now it can generate the corresponding private key using the first level private key it possesses, using I_{r_2} (used to generate ID_{req}) as the second level identifier. Suppose the first level private key is $d_{I_{r_1}} = (a_0, a_1, b_2)$, then:

- Private key for $ID_{req} : d_{ID_{req}}$

$$\begin{aligned}
 &= (a_0 \cdot b_2^{I_{r_2}} \cdot (h_1^{I_{r_1}} \cdot h_2^{I_{r_2}} \cdot g_3)^t, a_1 \cdot g^t) \\
 &= (a_0', a_1')
 \end{aligned}$$

- Finally, to obtain M_P the contact decrypts $CT_{resp} = (A, B, C)$:

$$(A \cdot e(a_1', C)) / (e(B, a_0')) = M_P$$

2.5.6 Peer Re-key

The set of contacts at a peer C can change in two ways:

- When a new contact joins.
- When an existing contact is removed.

When a new contact ($C_{P'}$) joins, the peer P can carry out new contact registration, and this does not require any changes to the parameters. The new contact will be able to request updates of the peer from its other contacts in the set $(C - C_{P'})$. However, when P needs to remove a contact $C_{P'}$ from the list of contacts, it has to update its parameters. At this point, the peer will have to update the remaining set of contacts with the new parameters. Due to this change in parameters, the contacts' private keys must be updated as well. In the approach presented here, a peer can generate information that the set $(C - C_{P'})$ can use to reconfigure their private keys. Importantly, this re-key information can be delivered via a public channel without compromising private keys of the contacts.

In peer setup, the generated configuration is of the form $params = (g, g_1, g_2, g_3, h_1, h_2)$ and $master-key = g_2^\alpha$ where $g_1 = g^\alpha$ and $\alpha \in \mathbb{Z}_p$ is random. In the case of re-key a peer :

- Generates a new random value $\alpha' \in \mathbb{Z}_p$
- Sets $master-key = g_2^{\alpha'}$

- Set $g_1 = g^{\alpha'}$

With this change, P will have to update the private keys of the contacts. Note that in the contact registration process, P stored the tuple (I_{r_i}, r) for each contact, C_{P_i} .

To update contacts, first generate a random value $u \in \mathbb{Z}_p$, initialize a list (id'_i, A_i) , and for each contact $C_{P_i} \in C$:

- Generate the first component of the private keys of the contacts as $g_2^{\alpha'} \cdot (h_1^{I_{r_i}} \cdot g_3)^{r_i} = A_i$. This r value is from the stored (I_{r_i}, r) .
- Add $(u^{I_{r_i}}, A_i)$ to the (id'_i, A_i) list.

Finally, the complete re-key information to be published is as follows:

$$(PID, g_1, u, [(id'_1, A_1), \dots, (id'_n, A_n)]) ,$$

where $n = |C|$. Note that, the value id'_i is the identifier of C_{P_i} blinded using u where $id'_i = u^{I_{r_i}}$.

When a contact $C_{P_i} \in C$ obtains this information it will do the following :

- Update P 's public parameters by replacing the g_1 value with received value.
- Retrieve its identifier issued by P (I_{r_i}) and compute $id' = u^{I_{r_i}}$
- Obtain the updated first component of its private key from the list $[(id'_1, A_1), \dots, (id'_n, A_n)]$ using id' .

After these steps, C_{P_i} will use the new parameters and the new private key to encrypt and decrypt updates.

2.6 Security Evaluation

This section discusses of security and privacy features for each of the protocol steps presented in the previous section.

2.6.1 Update Request

A contact of peer P generates a random identifier for any other party to use in encryption of an *update* message (which is included in the update request Q_P). As described in Section 2.5.3, this request takes the form :

$$ID_{req} = h_1^{I_{r_1}} \cdot h_2^{I_{r_2}}$$

Here h_1 and h_2 are public values but I_{r_1} and I_{r_2} values are only known to the contact who generates the request. Note that ID_{req} takes the form of a Pedersen commitment value [34], where it unconditionally hides the I_{r_1} and h_2 values.

Theorem 2.6.1 Given the value $ID_{req} = h_1^{I_{r_1}} \cdot h_2^{I_{r_2}}$, where h_1 is a generator of a group \mathbb{G} of prime order p . $I_{r_1}, I_{r_2} \in \mathbb{Z}_p$ and $h_2 \in \mathbb{G}$. ID_{req} hides the values I_{r_1} and I_{r_2} .

Proof Let $ID_{req}' = h_1^{x_1} h_2^{x_2} \mod p$

Note that since h_1 is a generator, $\exists a \in \mathbb{Z}_p$ where, $h_2 = h_1^a$.

$$\Rightarrow ID_{req} = h_1^{x_1} h_1^{a(x_2)} \mod p = h_1^{(x_1 + a(x_2))} \mod p$$

Due to the hardness discrete logarithm problem, it is infeasible to compute the value $(x_1 + a(x_2))$ such that, $ID_{req}' = ID_{req}$.

Using brute force approach, if an attacker arrives at a value y , such that $ID_{req}' = h_1^y = ID_{req}$:

$$\Rightarrow h_1^y \mod p = h_1^{(x_1 + a(x_2))} \mod p$$

$$\Rightarrow y \mod (p-1) = (x_1 + a(x_2)) \mod (p-1)$$

Therefore, for any value of x_2 there exists a value x_1 . Therefore, ID_{req} hides the values I_{r_1} and I_{r_2} . ■

Therefore, it is possible to publish the ID_{req} value in the public channel without revealing I_{r_1} and I_{r_2} .

2.6.2 Update response

A contact of P responds to a Q_P with a response S_P , which is of the form (P, ID_{req}, CT_{resp}) . Here CT_{resp} is standard HIBE encryption of M_P using the identity I_{r_1}, I_{r_2} . This value is secure due to the security assurances provided by the standard HIBE scheme [13]. Hence any other party other than the contact who generated ID_{req} will not be able to learn any information about P 's update, M_P .

Furthermore, note that the contact who generates the response does not attach any identifiable information about him/her self to the response. Therefore this process does not leak any information regarding who generated S_P to the contact who generated Q_P .

2.6.3 Re-key

When a peer, P , removes a contact and re-keys the system, the information published is of the following form:

$$(P, g_1, u, [(id'_1, A_1), \dots, (id'_n, A_n)])$$

Here $g_1 = g^{\alpha'}$ is a component of the public parameters of P in the standard HIBE scheme. It is computationally hard to obtain the α' due to the hardness of the discrete logarithm problem.

A similar argument can be made about the blinded identity values in the map of A_i values. Here the u value is raised to the power of the first level identity of the contact ($I_{r_{i1}}$). Since the identity values are only known to those corresponding contacts, to an eavesdropper, this is a table of random values and A_i values.

Finally, the A_i values are of the form $g_2^{\alpha'} \cdot (h_1^{I_{r_{i1}}} \cdot g_3)^{r_i}$. Here the $I_{r_{i1}}$ value is private between the peer, P , and contact, C_{P_i} , whereas r_i and α' is private to P . Furthermore, based on the same argument as in the case of update request, the A_i value does not disclose any information about r_i or α' . Moreover, no one other than P will be able to compute the other two components of the private key issued to C_{P_i} .

Therefore, the tuple, (id'_i, A_i) , does not compromise the private key information or the identity of the contact.

After removing a contact and re-keying the parameters of a peer, the removed contact will still be able to issue a request for an update. Even if a current contact of the peer responds to such a request, the removed contact will not be able to decrypt and obtain the message due to the use of the new parameters.

2.6.4 Peer Unlinkability

An update request is defined as $Q_P = (PID, ID_{req})$, where PID is an identifier string of P . PID is known to all P 's contacts.

An observer of the public channel, used here for communication, may clearly gain information because the value PID is available as plain text. This can be avoided by blinding the PID value in the update request as follows:

- Define $PID \in \mathbb{Z}_p$
- Select a generator $g_r \in \mathbb{G}$
- $Q_P = (g_r^{PID}, g_r, ID_{req})$

This optimization however, forces the users of this scheme to compute the g_r^{PID} value for each of their contacts to identify an update request that they can serve. This process may be an expensive operation in a situation where the user has a large set of contacts. This optimization is also applicable in the other cases of the scheme where PID is used.

2.6.5 Event Causality

In solving the problem of propagating a message to a set of contacts, one may argue that the user can simply encrypt a message and publish that message for any

party to access. However, the method proposed here has several important advantages over this approach.

First, it is important to note that the set of trusted contacts may change at any time. If the message is encrypted once and published in a public channel, then all users who have access to valid keys at the point of generation of the ciphertext will be able to access the message. However, with the proposed approach, the message is communicated to one user at a time. At each point any user will be able to receive the message if and only if he/she is in the list of authorized users. This can be enforced using a public channel which assigns a sequence number to each published message. Using this channel one can establish a happens-before relationship between actions. Therefore, this channel will be useful in enforcing the processing of re-key information before processing any further messages. Using this approach a user may guarantee that his/her messages will not be propagated to a party that he/she no longer trusts.

Figure 2.3 shows an example of a situation where a peer initially has 9 contacts. First the peer sends an update to contact 1, who is the only contact available at that point. Next, contact 2 is available online and requests the latest update, which he/she receives from contact 1. The peer decides to remove contacts 6 and 8 from the list of trusted contacts and publishes re-key information. All contacts process this information. Finally, contacts 6 and 8 are available, and they may attempt to request updates of the peer. Even if the other contacts respond with valid responses, 6 and 8 will not be able to receive the correct plaintext update value. Due to the above change in the public parameters, the ciphertext values in the responses are generated using new parameters. Therefore, decryption attempts of the removed contacts fail to produce the plaintext.

2.6.6 Protection Against Malicious Peers

So far, the protocols presented here expect all contacts of a peer to redistribute the update messages to requests *correctly*. In other words, contacts respond with

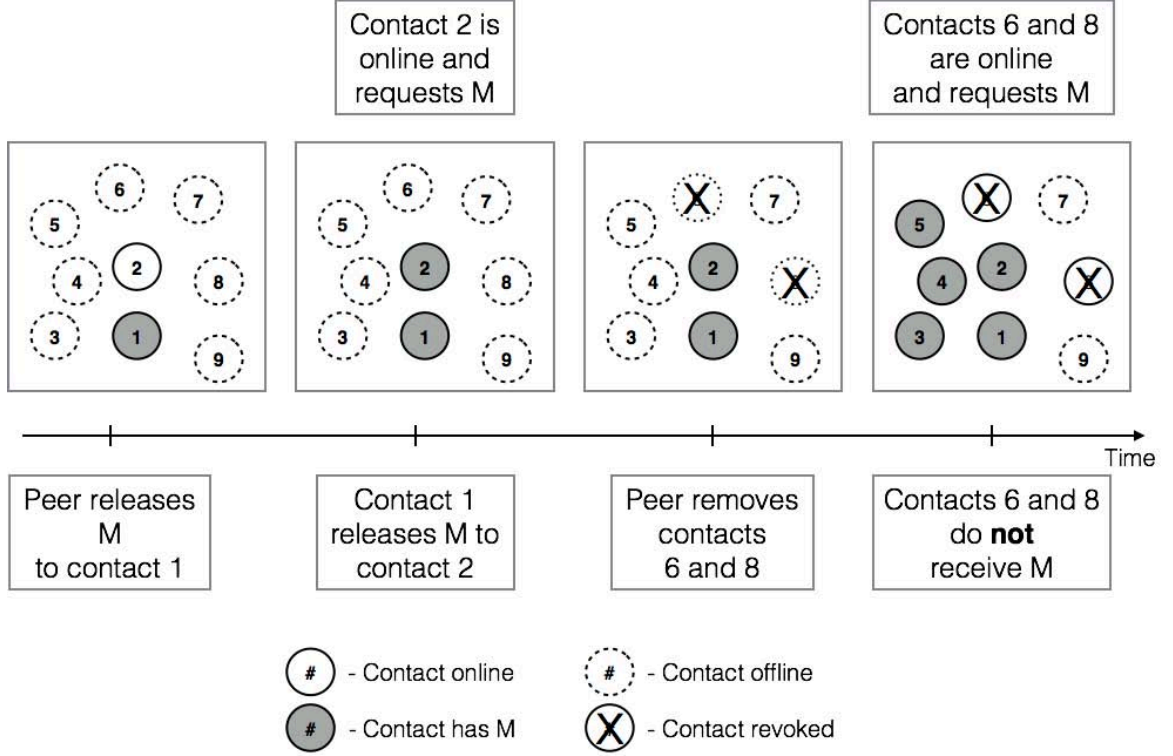


Figure 2.3. Example of contact revocation action during update propagation

the same message that they received. With this assumption, a contact has no mechanism to verify whether the peer generated the message the contact received. This assumption can be relaxed with a minor alteration of the protocols.

The peer, as he/she generates the system parameters, generates a public-private key pair. Let $k_{P_{pr}}$ denote the private key and $k_{P_{pu}}$ denote the public key. $k_{P_{pu}}$ is provided to each contact along with the system parameters during the contact setup protocol.

The peer signs each update, M , it sends out using $k_{P_{pr}}$ to generate σ_M . Each contact verifies σ_M using $k_{P_{pu}}$ and the update M . Suppose a malicious party attempts to generate an update M , and claim that it was from the peer. Since this party is not able to generate a valid σ_M , none of the contacts will accept M .

2.6.7 Lifecycle of an Update Request

When a contact submits a request to the public channel, online contacts of the peer will respond to the request. The number of contacts who responds to the request can be limited using an approach that attaches an expiration time to the request along with a mechanism for the contact to *close* the request.

Automatic Expiration of Update Requests

The public channel attaches a timestamp to an update request based on the time it received the update request. Any entity can query the public channel and receive a set of active requests. The public channel responds with a set of unexpired requests. Each of these is considered an active request. The expiration time window is set by the administrator of the public channel, and this value is known to all parties.

The update response is of the form $(PID, ID_{req}, CT_{resp})$. The public channel attaches this response to the request based on the ID_{req} value. Each contact who responds to a particular request stores the ID_{req} value. These stored values are used to avoid the same request being served twice.

Update Requests Closed by a Requester

When a contact generates a request, it will generate a key pair (k_{reqpu} and k_{reqpr}) of a signature scheme. Note that, a new key pair must be used for each request. The public key, k_{reqpu} , is added to the update request. The contact submits the value $(PID, ID_{req}, k_{reqpu})$ to the public channel. At the point, where the contact is satisfied with received responses, he/she can send a *request close* message to the public channel. This message should be signed using the k_{reqpr} value.

The public channel verifies the signature value using k_{reqpu} , which was posted along with the relevant update request. If the signature verification is successful, the request is marked as *closed*. When the public channel is queried for pending update

requests, it will exclude those closed requests from the list of requests included in the response. If a malicious user attempts to close an update request, that user will not be able to provide a valid signature on the *request close* message. Therefore, such an attempt will fail.

Both automatic expiration of an update request and closing of requests by the requester minimizes the burden on the contacts to generate responses.

2.6.8 Verification of Response Generators

The protocol presented here allows *any* participant to respond to a request submitted to the public channel. Therefore, even a party who is not a valid member of the peer may respond. Since such a party will not have the update message and a valid signature from the peer, this response can be discarded. However, note that the requester still has to decrypt the response before verifying the original signature. The possibility of a non-contact of a peer responding to a request can be blocked with the following addition to the protocol.

- As a response to $Q_P = (PID, ID_{req})$ a contact submits an intent to respond of the form $R_P = (ID_{req}, ID_{resp})$, where $ID_{resp} = h_1^{I_{resp_{r1}}} \cdot h_2^{I_{resp_{r2}}}$. The $I_{resp_{r1}}$ value is the identifier assigned to the contact by the peer and $I_{resp_{r2}} \in \mathbb{Z}_p$ is a random value.
- The requester generates a random value $v \in \mathbb{G}_1$ and computes $ct_v = \text{Encrypt}'(params_P, ID_{resp}, v)$
- The value (ID_{resp}, ct_v) is submitted to the public channel.
- The contact who submitted R_P decrypts ct_v to obtain v .
- Response to the original request, S_P , is created as defined in Section 2.5.4 and v is included in the message published: (PID, ID_{req}, v)

- The requester inspects the public channel for response messages with the value v and decrypts the response.

Since only a valid contact of the peer can recover v from ct_v value, the requester is guaranteed that the response was generated by such a party.

2.7 Implementation

The proposed scheme was implemented in Java as a library using Java Pairing Based Cryptography [35] library. This work is available under LGPL at “anon-encrypt” project hosted in Google Code [36]. Unit tests were developed to ensure the correctness of each functionality. These unit tests are integrated into the build script of the library. A framework of peers and a public channel was developed to simulate and experiment with the proposed protocols. Also, a proof of concept application was developed to demonstrate features of the library and the proposed protocols.

2.7.1 Library

The basic functionality of generating a new set of parameters, generation of private keys for the peer, generation of temporary private keys for a contact, generation of a request, encryption and decryption of a message, and generation and processing of re-key information was developed as a library. Each value that needs to be transmitted from one entity to another is encapsulated as an object. These objects include a method (*serializeJSON()*) to generate and output JSON [37] representations of themselves and a constructor to parse such JSON representations to initialize. All classes of the library are contained in the *org.ruchith.ae.base* package.

Parameter Generator

The *org.ruchith.ae.base.AEParameterGenerator* class generates a new set of parameters as defined in Section 2.5.1. This outputs an instance of the *org.ruchith.ae.base.AEParameters* class. This *AEParameters* object can be serialized as JSON to be published.

Peer Key Generator

A peer application uses the *org.ruchith.ae.base.RootKeyGen* class to create private keys for contacts. The *genKey()* method generates an instance of the *org.ruchith.ae.base.AEPrivateKey* class. A serialized format of an *AEPrivateKey* object and an *AEParameters* object are sent to a contact at the point of registration.

Contact Key Generator

The *org.ruchith.ae.base.ContactKeyGen* class provides the functionality required by a contact. This class is initialized using a contact private key and parameters issued by a peer. The method *getTmpPubKey()* allows a contact to generate a fresh request value to publish. The *getTmpPubKey()* method generates the corresponding private key to decrypt a response.

Cipher Implementation

The encryption and decryption functions are implemented in the *org.ruchith.ae.base.Encrypt* and *org.ruchith.ae.base.Decrypt* classes. A plain text element ($\in \mathbb{G}_1$) is represented as an *it.unisa.dia.gas.jpbc.Element* object. The *doEncrypt()* method of the *Encrypt* class encrypts such an element using a given request value. This encryption operation outputs an instance of *org.ruchith.ae.base.AECipherText*. The ciphertext is serialized into JSON to be published. The *doDecrypt()* method of the *Decrypt* class will instantiate an *AECipherText* instance using the serialized

ciphertext and decrypts it using the given private key. In practice, the ciphertext here will be the encryption of the key used to encrypt the payload with a symmetric key encryption algorithm.

Text Encoder

It is important to note that the cipher implementation works on elements of the group \mathbb{G}_1 . Therefore, the following encode and decode functionality is required to be able to encrypt arbitrary sequences of bits.

$$\begin{aligned} \text{encode} &: \{0, 1\}^* \rightarrow \{\mathbb{G}_1\}^* \\ \text{decode} &: \{\mathbb{G}_1\}^* \rightarrow \{0, 1\}^* \end{aligned}$$

The *org.ruchith.ae.base.TextEncoder* class supports both these functionality and is initialized with the public parameters. The *encode()* method returns an array of elements $\in \mathbb{G}_1$ given a byte array and the *decode()* method returns a byte array, given an array of elements.

Re-key

The functionality required to revoke a contact and re-key a peer is provided in the *org.ruchith.ae.base.ReKey* class. This is initialized with the current parameters and *update()* function creates a new master key and the new g_1 value. The *getPublicInfo()* method generates an instance of an *org.ruchith.ae.base.ReKeyInformation* class using a map of (I_{r_i}, r) values of the contacts. This information is serialized and published into a public channel.

2.7.2 Proof of Concept Application

The proof of concept application was developed using the above library. This application uses a database with one table to hold all information of a contact, such

as contact's parameters, private key assigned to the peer by the contact, and common name for the contact. Apache Derby [38] was used as the database management system. The database instance is maintained in a configuration directory (called *.ae*) located in the user's home directory.

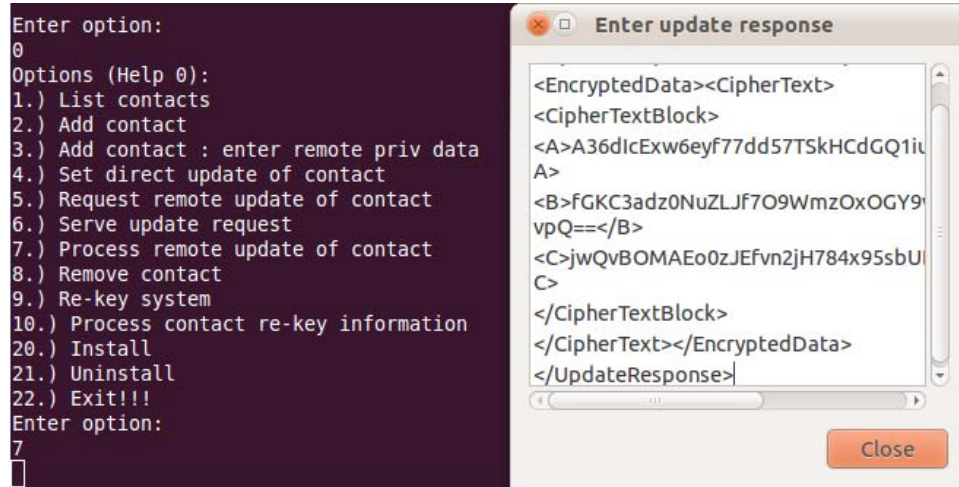


Figure 2.4. A screenshot of the demo application

2.7.3 Experiment Framework

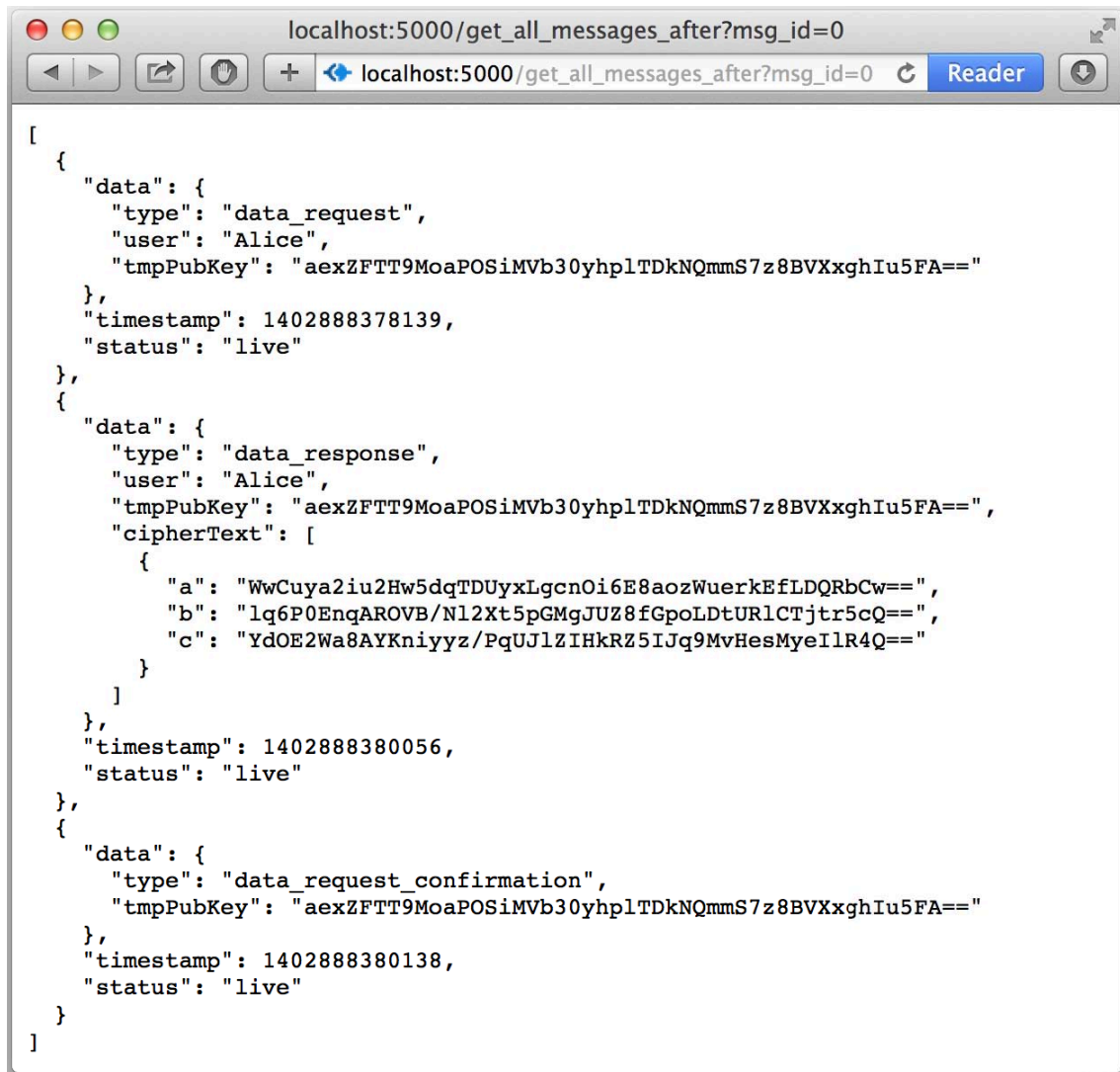
A framework of peers and a public channel was developed to perform a set of experiments with the proposed protocols. The public channel exposes a set of operations for a peer to use. The peer application is designed to accept a set of commands to perform required actions and uses the public channel for communication.

Public Channel

The public channel was developed as a node.js web application. This application exposes a set of operation end points. These operations are invoked by the peer application to send and receive messages and read the status of the public channel. Table 2.1 lists these operations.

Table 2.1.
Operations of the public channel

Operation	Description
<code>add_message</code>	Adds a given message to an ordered list.
<code>get_all_messages_after</code>	Returns all public messages starting from the given index.
<code>add_direct_message</code>	Adds a message to a list reserved for a peer. A peer uses this operation to send a message directly to a contact.
<code>get_all_direct_messages</code>	Returns the list of direct messages sent to a peer. The name of the peer is the name of the invoker of the operation.
<code>set_message_index_of_peer</code>	Sets a value given by the caller as its current update number.
<code>get_message_index_of_peer</code>	Returns the current update number of a peer. This is used as the trigger for a contact to request the latest update of a peer.



```
[
  {
    "data": {
      "type": "data_request",
      "user": "Alice",
      "tmpPubKey": "aexZFTT9MoaPOSiMvb30yhplTDkNQmmS7z8BVXxghIu5FA=="
    },
    "timestamp": 1402888378139,
    "status": "live"
  },
  {
    "data": {
      "type": "data_response",
      "user": "Alice",
      "tmpPubKey": "aexZFTT9MoaPOSiMvb30yhplTDkNQmmS7z8BVXxghIu5FA==",
      "cipherText": [
        {
          "a": "WwCuya2iu2Hw5dqTDUyxLgcnOi6E8aozWuerkEfLDQRbCw==",
          "b": "lq6P0EnqAROVb/Nl2Xt5pGMgJUz8fGpoLDtURlCTjtr5cQ==",
          "c": "YdOE2Wa8AYKniyyz/PqUJlZIHkRZ5IJq9MvHesMyeIlR4Q=="
        }
      ]
    },
    "timestamp": 1402888380056,
    "status": "live"
  },
  {
    "data": {
      "type": "data_request_confirmation",
      "tmpPubKey": "aexZFTT9MoaPOSiMvb30yhplTDkNQmmS7z8BVXxghIu5FA=="
    },
    "timestamp": 1402888380138,
    "status": "live"
  }
]
```

Figure 2.5. Sample output of public channel messages

Format of Public Messages

Each public message contains a *type* attribute. A public message can be either a “data_request,” “data_response,” or “data_request_confirmation.” An example of these three are shown in Figure 2.5. A “data_request” message contains the name of the peer, and the base64 encoded ID_{req} value, whereas a “data_response” message

additionally contains the ciphertext value. A “data_request_confirmation” message is only sent when a peer receives a valid message as an indication to *close* the request.

The public channel sets a time-to-live value for each of these messages. Each message is initially marked as *live* and is assigned a timestamp value. These messages are marked as *expired*, after the difference of the current time and timestamp on the message, exceeds the time-to-live value. Peers only process those messages that are marked as *live*.

Peer

The peer application is also designed as a nodejs web service. This application requires two arguments, the name of the peer and a port number. The application listens on the given port number and exposes two operations, *stop* and *action*. The *stop* operation immediately kills the process. The *action* operation carries out either the registration of a contact, sending a direct message, or removing a contact based on incoming instructions. The peer application was designed to be able to craft scripts for experimentation using Unix tools. Specifically, *curl* is used to invoke actions on each peer. Figure 2.6 shows an example script to start a main peer, Alice, and contacts, Bob, Charlie and David. In this experiment Alice sends an update message to Bob, removes Charlie and sends a new update message to Bob. Only Bob and David finally receives the latest message as expected. Timing delays between each event are added using the *sleep* command.

The peer application includes three independent threads that interact with the public channel. First thread obtains any available direct messages intended for the peer. The second reads the message count of each contact and makes necessary requests for updates. The third thread processes any new public messages. Timing of each of these threads is configurable, in terms of the interval of time each of them waits between runs.

Experiments

Two main experiments were carried out to evaluate correct operation of the main protocol features.

The correct operation of response to an update request is evaluated using one main peer with two contacts. The main peer sends a direct message to one contact and updates the public channel with the message count. The contact who did not receive the direct message notices that there's a new message from the main peer and publishes a new "data_request" message. The contact responds to this message by publishing a "data_response" message. The contact who published the original request can decrypt the response and output the original message. Finally, a "data_request_confirmation" message is sent to the public channel to close the request. Figure 2.5 shows the messages in the public channel at the end of this experiment.

The correct operation of removing a contact was tested using the script shown in Figure 2.6. In this experiment, Alice, the main peer initially has two contacts, Bob and Charlie. Alice removes Charlie and then sends a new message. Bob updates his private key and his copy of public parameters of Alice. Charlie attempts to request this message and receives a response encrypted with the new parameters. Charlie decrypts and outputs the received response, which is not equal to the message Alice sent. In fact, the output takes the form of a set of random bytes that are not ASCII. It was also observed that Charlie still can respond to other user's requests with incorrect messages. The protocol modifications presented in Section 2.6.8, to verify the contact that responds, were motivated by this behaviour.

```

#Start the main peer Alice
node app.js 8001 Alice &

#Wait for startup
sleep 1

#Add two contacts
curl "http://localhost:8001/action?action=add_contact&name=Bob"
curl "http://localhost:8001/action?action=add_contact&name=Charlie"

#Send a message to Bob
curl "http://localhost:8001/action?action=direct_message&to=Bob&message=MEET_AT_2"

#Start Bob and add Alice as a contact
node app.js 8002 Bob &
sleep 1
curl "http://localhost:8002/action?action=add_contact&name=Alice"

#Start Charlie and add Alice as a contact
node app.js 8003 Charlie &
sleep 1
curl "http://localhost:8003/action?action=add_contact&name=Alice"
sleep 2

#Alice: Remove Charlie
curl "http://localhost:8001/action?action=remove_contact&contact=Charlie"
sleep 1

#Alice: Send new update to Bob
curl "http://localhost:8001/action?action=direct_message&to=Bob&message=MEET_AT_5"
sleep 1

#Alice: Add David
curl "http://localhost:8001/action?action=add_contact&name=David"
sleep 1

#Start David
node app.js 8004 David &
sleep 1
curl "http://localhost:8004/action?action=add_contact&name=Alice"

```

Figure 2.6. Example shell script to start an interact with a set of peers

3 PRIVACY IN IDENTITY MANAGEMENT

3.1 Introduction

As the Web introduced personalized services to users, it was essential to identify the user and learn some identity attributes of that user. This requirement was addressed by developers in an application-centric manner where each application implemented its own solution.

One of the most popular methods used requires the user to sign up by providing a set of information about himself/herself along with a user name and a password. Although, such a mechanism is very simple to implement and convenient for the users, there are numerous issues with this approach. First, most of the users use the same password across most of the services they use. This is problematic because if one system is breached by an attacker, that attacker will be able to access all those other services where the user used the same password. It is true that there are techniques to prevent an attacker from obtaining user passwords even if the system was breached. However, there are reports of attacks – such as the case with LinkedIn [39] – where it was evident that there are services which still may store plaintext passwords of users, or even employ incorrect implementations of the password protection techniques.

There are reports of various attacks targeting online service infrastructure every day. Most of these attacks utilize vulnerabilities that exist in the service software to gain access to back-end systems where the attackers may obtain databases of user information. Since there are numerous applications that obtain a user's identity attributes – such as name, date of birth, address, credit card details – the attackers who obtain this information will be able to perform identity theft. Note that the service providers may be held liable for protecting users' personal information. The

service provider may also be forced to give away such information to authorities. This could be detrimental when it comes to users' goodwill towards the service provider.

Therefore, it is clear that storing personal identity information at various services is extremely dangerous. In analyzing this problem, it is important to understand the need of the service providers to obtain these identity attributes. For example, in most cases identity attributes, such as the date of birth, are required to prove that the user meets certain age restrictions every time the user procures the service. Another example is the use of address information to provide various location-sensitive services, which may not require the exact address but only more coarse-grained location information (e.g. for a weather alerting system). Therefore, it is evident that if there is a way to prove certain statements about a user's identity (or identity claims) to a service provider, it may not need to store identity attributes to derive these identity claims.

Social engineering attacks are successfully mounted on users to gain access to user credentials. One of the most common forms of such an attack is the *phishing attack* [40]. During a phishing attack, the attacker sends an email that captures the attention of the user of the target application. Such an email is crafted to make the contents appear to have the same look-and-feel as the target application. This email usually contains a hyperlink for the user to log into the target application. The link takes the user to the attacker's web application. At this point, a user who is concerned about some notice in the phishing email, enters the credentials and attempts to log into the application. Upon submission of this information, the attacker captures the login credentials and redirects the user to the actual application that the user intended to access. Rise of these types of threats call for novel techniques to protect the user against such attacks.

There were several interesting attempts by industry to address most of these issues. One such instance is Microsoft Passport system in which the Microsoft Corporation set up a service that users may sign up and provide their information. Other services can be registered with the Passport system to accept passport authentication. Users

were redirected to the Passport web application to login first before they are allowed into the intended service’s web application. However, due to the presence of only one authority to manage identity attributes of a user, this effort was not successful. This motivated further efforts and one breakthrough was the *Laws of Identity* [14] and the concept of *Identity Metasystem* [41]. The laws of identity discuss important aspects such as user control and consent, minimal disclosure of identity, bi-directional authentication of parties involved (where the users are assured that they are interacting with the intended party), as well as consistent and intuitive user experience for managing identities. The identity metasystem idea claims that there needs to be an interoperable system of systems where there are three main classes of systems. These are “identity providers,” “relying parties” and “subjects”. An *identity provider* is an authority that asserts various statements about the subject. These statements are called *claims*. A *subject* is any entity to which the claims apply. A *relying party* is an entity such as a service provider who intends to obtain user identity claim information for authentication or authorization.

Windows CardSpace was the first step towards realizing the concept of an identity metasystem. This was defined as an open specification and relied on the standard WS-* specification stack [42] [43] [44] [45] [46]. WS-* stack is a set of XML-based protocols that are independent of the communication protocols with multiple interoperable implementations. It was expected that due to the use of open standards, it would make it easier for the industry to adapt this technology. In CardSpace, an identity provider which manages identity claims in CardSpace verifies user’s information and is responsible for storing this information. Furthermore, the identity provider can issue tokens (using the Security Assertion Markup Language [47]) with certified claims about a user’s identity. These tokens are only meant for a particular relying party and are encrypted using that party’s public key. The user interacts with this system via a user interface that remains the same when interacting with all relying parties. The requirement for users to authenticate and remain unlinkable when using CardSpace was identified by Steuer et al. [48]. This proposed the use of

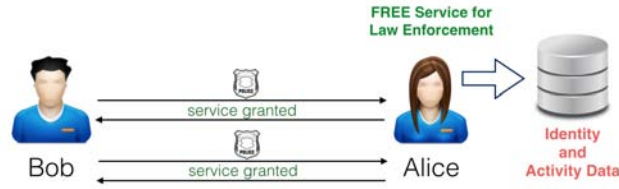


Figure 3.1. Bob authenticating multiple times with Alice using his credentials

a cryptographic commitment as the value of a claim to be included in the token issued by the identity provider. The main contribution to remove linkability of a user's transactions is a mechanism to generate a unique commitment value that depends on the previous commitment value used. Using this technique, the user is able to generate two authentication requests that cannot be linked to each other. However, it should be noted that this system still required the identity provider to be available online during the complete message exchange.

This work identifies a set of requirements that further strengthens the user against leaking his/her identity information and proposes a novel technique to solve these issues.

3.2 Problem

3.2.1 Overview

Consider the following scenario. Alice runs a coffee shop, which provides a free Internet access to members of law enforcement. Bob is a police officer who presents his credentials and uses the service.

If Bob uses traditional credentials, his interaction with Alice will pose several privacy issues. In the case where Bob uses the same identity information to authenticate himself with Alice multiple times, Alice can track Bob's usage patterns. In other words, Bob's subsequent transactions with Alice become linkable. Furthermore, if the credentials provided by Bob to Alice include other information about Bob, such

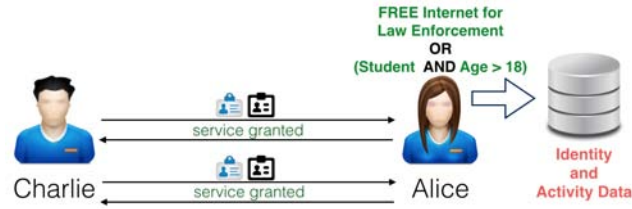


Figure 3.2. Charlie authenticating multiple times with Alice using his credentials

as his name and date of birth, then this will leak more information other than the fact that he is a member of law enforcement.

Suppose that Alice decides to extend the service to other members of the community. She now allows college students who are 18 or older to use the free Internet service. In this situation, Charlie, a college student, may have to provide not only his university ID but also his drivers license (proving his age) to obtain free Internet access. This potentially allows Alice to collect more information about Charlie than she should be able to.

The three main requirements to meet in these scenarios are:

- Any party with valid claims issued by a party trusted by Alice should be able to authenticate with Alice's service.
- Each successful authentication with Alice's service should be unlinkable with each other.
- Alice should not be able to obtain any information other than what is specified as required claims.

These high level requirements are formally defined in Section 3.2.2.

3.2.2 Problem Definition

A user needs to prove one or more identity claims to a service provider. These identity claims are expected to be issued by an identity provider that is trusted by the

service provider. Furthermore, the relying party may require identity claims issued by several different identity providers. Also, the user should not be able to collude with another user who possesses a subset of identity claims satisfying a relying party's policy. The scheme should not expect a trusted identity provider to be available online at the time when a user attempts to authenticate with a service provider.

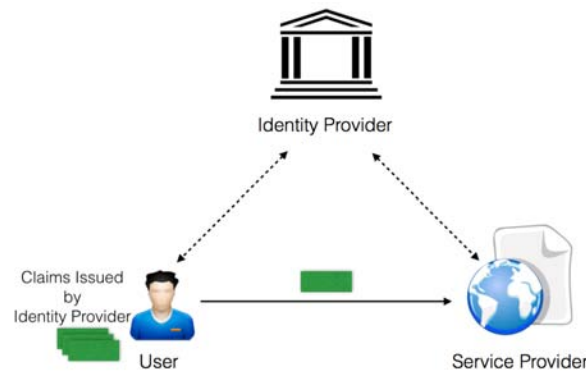


Figure 3.3. Typical setup of an identity management system

Case 1: Single Claim

Identity provider issues a credential with an identity claim to a user.

Service provider's policy requires the user to prove the possession of an identity claim issued by a particular identity provider in order to access the service.

User must be able to prove the ownership of a claim issued by the identity provider to the service provider multiple times without the risk of exposing any uniquely identifiable information.

Case 2: Multiple Claims

Identity provider issues a credential with a *set* of identity claims to a user.

Service provider's policy requires the user to prove the possession of a set of identity claims issued by particular identity providers in order to access the service. (This policy may specify any combination of these claims with *AND* and *OR* conditions.)

- Service provider should not be able to learn a subset of claims used by a user in a situation where the user decides between two or more options of the identity claim sets.
- User must be able to prove the ownership of claims issued by the identity providers to the service provider multiple times without the risk of exposing any uniquely identifiable information.
- The user cannot collude with another user.

3.3 Related Work

The identity management system proposed by Camenisch and Lysyanskaya [18] describes how a user can obtain credentials from an identity provider and use those credentials with unlinkability using zero-knowledge proofs. In addition to unlinkability, this scheme carries properties of one-show credentials, non-transferability and anonymity revocation. Belenkiy et al. [49] introduced the use of non-interactive proofs using P-signatures, where the user obtains a signature on a commitment of a message and generates a non-interactive zero-knowledge-proof-of-knowledge of the signature. This scheme requires the user to rely on multiple parties to register pseudonyms with to achieve unlinkability. The scheme proposed by Bhargav-Spantzel et al. [17] uses a technique based on aggregate signatures on commitments that are then used for aggregate zero-knowledge proof-of-knowledge protocols to present identity attributes to service providers. This scheme requires an online registrar of identity attributes that is equivalent to an online identity manager and allows the presentation of multiple identity attributes to a service provider in a single run of a zero-knowledge-proof protocol. Alpar and Hoepman [50] present the use of attribute-based credentials to set up an authenticated secure channel. An authenticated user is identified to the extent of the provided identity attribute proofs. Garman et al. [51] describe an approach based on techniques adapted from digital currencies where the user generates the credentials without the need for a trusted credential issuer.

In schemes that rely on signature based anonymous credentials, the user, or an application running on behalf of the user, selects a set of claims/credentials to be proven with a service provider based on the service provider’s policy. Such schemes have the potential to allow a service provider to carry out statistical analysis as it relates to complex service provider policies. For example, a service provider may require a user to prove possession of either one of two sets of claims (SET_1 OR SET_2). The user’s first step is to select which set of claims to use. This allows the service provider to arrive at statistical conclusions about the user population, which may compromise user privacy. The solution presented in this paper addresses this issue, assuring that the service provider learns no information above the required minimum about the set of claims used by a user in satisfying its policy. Camenisch et al. [52] also address handling such OR relationships using zero-knowledge proofs, which require two commitments and four linear relationship proofs by the user. The protocol proposed in this paper, allows the user to setup an authenticated secure channel with a service provider, with the user having to produce only a single request value as proposed in Section 3.4.4. Furthermore, the user can use a single request value to initiate an authenticated secure channel in the case of a service provider policy with AND conditions.

The problem of unlinkability in presenting identity claims using the Windows CardSpace platform was addressed by Steuer et al. [48]. This solution however, still requires the presence of an online identity manager and can handle a service provider policy only in the form of a simple set of required identity claims. Bhargava et al. [53] also extend Windows CardSpace to support zero-knowledge-proof-of-knowledge of identity attributes, but they do not address the problem of unlinkability.

Angin et al. [54] proposed an entity-centric approach for identity management in cloud computing using the concept of an active bundle (self-protecting data) [55] [56]. Identity attributes of a user are packaged in an active bundle, which includes a virtual machine (VM) that is executed in a trusted environment. At the point of execution

the VM determines the required set of identity information according to the associated policies.

Mobile devices maintain user identity attributes. Therefore a number of papers attempt to address the problems of identity attributes being maliciously leaked [57] [58]. However, the problems addressed in this work do not include identity attribute leakage due to malware or identity inferences using machine learning techniques [2] [3].

Protocols presented in this work allow a user to setup an authenticated secure channel with a service provider using identity claim instances issued by a trusted identity provider. The user generates an authentication request that does not disclose any identity information. This request value is used by the service provider to generate an encrypted response. The service provider is assured that the user is only able to decrypt the response only if he/she possesses claim instances required according to the service provider policy. This two-step interaction hides identity claim choices made by the user, from the service provider.

3.4 Solution

The solution consists of the following algorithms and protocols:

- Generate a claim (GenClaim)

Identity provider carries out GenClaim to setup a claim definition.

- Issue a claim (IssueClaim)

This is a protocol used by the identity provider to issue a claim instance to a user.

- Authenticate with a single claim.

This is a protocol carried out between a user and a service provider, in which the user who possesses a claim instance sets up an authenticated secure channel with the service provider.

- Authenticate with multiple claims

This is a protocol carried out between a user and a service provider, in which the user sets up an authenticated secure channel with the service provider, under the constraints of a policy using multiple claims.

- Claim revocation

Identity provider can revoke a claim definition or an issued claim.

3.4.1 Generate a Claim (GenClaim)

An identity provider creates an identity claim by generating the following values. The *claimkey* value is maintained privately by the identity provider and the public component *params* are published along with the claim metadata.

- Select generators $g, g_2, h_1 \in \mathbb{G}$ and a random value $\alpha \in \mathbb{Z}_p$
- Set $g_1 = g^\alpha$
- Pick random values $g_3, h_2 \in \mathbb{G}$.
- $params = (g, g_1, g_2, g_3, h_1, h_2)$
- $claimkey = g_2^\alpha$

Claim metadata includes the expiration timestamp and a description of the claim. For example, a set of identity claims published by an identity provider can be a table as shown in Table 3.1. This information is to be consumed by service providers who trust the identity provider to authenticate users that possess claims issued by that identity provider.

3.4.2 Issue a Claim (IssueClaim)

When a user needs a particular claim to be issued by an identity provider, the user first obtains the public parameters of the claim. As shown in Figure 3.4, the

Table 3.1.
Example set of claim public data published by an identity provider

Claim ID	Public Component	
	<i>params</i>	Metadata
<i>student</i>	$(g, g_1, g_2, g_3, h_1, h_2)_{student}$	2020-12-31, ...
<i>faculty</i>	$(g, g_1, g_2, g_3, h_1, h_2)_{faculty}$	2025-08-31, ...
...
<i>claim_i</i>	$(g, g_1, g_2, g_3, h_1, h_2)_i$	<i>timestamp_i</i> , ...
...
<i>claim_n</i>	$(g, g_1, g_2, g_3, h_1, h_2)_n$	<i>timestamp_n</i> , ...

user then generates and submits a request (req) of the following form to the identity provider:

- User selects a random value $ID_{claim} \in \mathbb{Z}_p$
- Set $req = h_1^{ID_{claim}}$ where h_1 is extracted from the public component of the claim published by the identity provider
- Send req to the identity provider

The identity provider authenticates the user with a mechanism that is beyond the scope of this paper. For example, the identity provider may require that the user signs req with his/her public key certificate (which he/she used at the point of registering with the identity provider). Note that the req value can be forwarded to the identity provider over a public channel.

Upon receiving the authenticated req from the user, the identity provider runs *IssueClaim*, defined as follows:

- Select a random value $r \in \mathbb{Z}_p$
- Output $claim = (c_0, c_1, c_2) = (claimkey \cdot (req \cdot g_3)^r, g^r, h_2^r)$

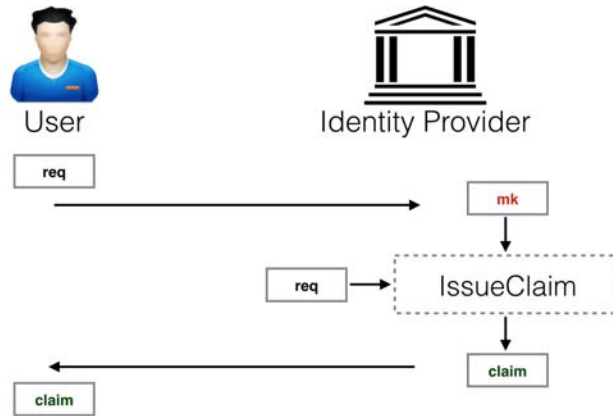


Figure 3.4. Identity provider using the request value given by a user to issue a claim

The issued claim must be returned to the user over a secure channel. This is the only requirement for a closed channel for communication between parties involved in all protocols defined in this work. This is also a practical requirement to meet since the user authenticates with the identity provider, at which point they can set up a secure channel (for example, using SSL). Upon receiving the issued claim, the user stores the tuple $\langle ClaimID, ID_{claim}, claim, params_{claim} \rangle$.

3.4.3 Authenticate with a Claim

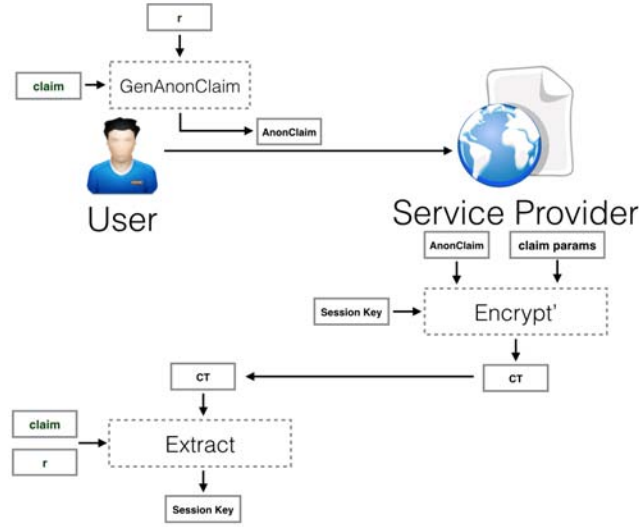


Figure 3.5. User uses a claim to generate an *AnonClaim* to initiate an authenticated session with the service provider

When a user needs to authenticate with a service provider who requires the user to possess a particular claim, the two parties carry out the following steps, as shown in Figure 3.5:

- User selects a random value $r \in \mathbb{Z}_p$
- Set $AnonClaim = h_1^{ID_{claim}} \cdot h_2^r$,
where $h_1, h_2 \in params_{claim}$

- Send *AnonClaim* to the service provider
- Service provider generates a *session key* that is a symmetric encryption key
- Service provider invokes *Encrypt'*, generates *ct* and sends *ct* to the user.
Encrypt' is the modified HIBE *Encrypt* function proposed in Section 2.5.4
- User invokes *Extract* to recover the session key

Extract

The *Extract* function is used by the user to obtain the session key. This function first generates the corresponding ephemeral private key using the value *r* and the *claim* instance issued to the user by the service provider. Then this private key is used to decrypt *ct* using the HIBE [13] Decrypt function.

- $key_r = (k_{r_0}, k_{r_1}) = KeyGen(claim, r)$
 $(k_{r_0}, k_{r_1}) = (c_0 \cdot c_2^r \cdot (h_1^{ID_{claim}} \cdot h_2^r \cdot g_3)^t, c_1 \cdot g^t),$
 where random value $t \in \mathbb{Z}_p$
- $Decrypt(key_r, ct)$ outputs session key

Once the user extracts the session key, it may encrypt future communication of the current session with the service provider using the session key. Also, in cases when the service provider allows access to protected content for the authenticated user, it can optionally send that content encrypted using the session key along with the *ct* value.

3.4.4 Authenticate with Multiple Claims

The notion of a monotonic access tree, discussed by Bethencourt et al. [59], is used to address the case of multiple claims. A service provider requires a user to

authenticate with more than one claim, and the service provider's policy may enforce certain constraints on those claims as well. These constraints are AND and OR combinations of claims. This constitutes an access tree in which AND and OR constraints are defined as $n - of - n$ and $1 - of - n$ threshold gates, respectively. The leaves contain a set of shares of an ephemeral key, which are encrypted using the user-provided *AnonClaim* values using *params* of each claim. Figure 3.6 shows an example of such an access tree.

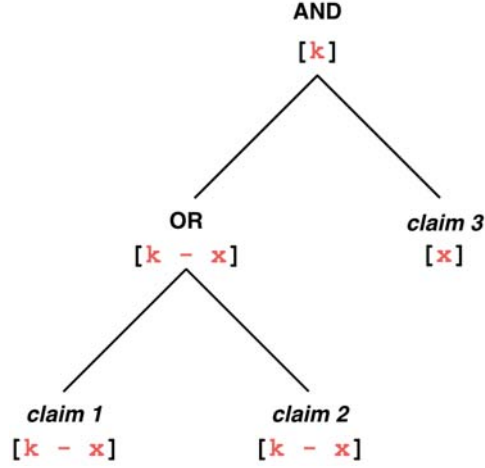


Figure 3.6. An example access structure and one possible technique to propagate the shares of a secret from the root node to the leaves

The service provider returns the access tree with the encrypted leaves to the user. The access tree is decrypted by the users, and the ephemeral key is recovered to be used as a session key. The above process is formally defined as follows:

The set of claims expected by the service provider is denoted by the set $C_{\mathcal{SP}} = \{c_1, c_2, \dots, c_i, \dots, c_n\}$, where $c_i = params_{claim_i}$. A user evaluates the policy of the service provider and identifies a subset of claims that the user is willing to disclose for authentication. This subset is denoted by $C_{\mathcal{U}} = \{c_1, c_2, \dots, c_j, \dots, c_m\}$, where $m \leq n$.

User \mathcal{U} generates the set *AnonClaims* as follows:

- Select a random number $r \in \mathbb{Z}_p$

- $AnonClaims_{\mathcal{U}} = (\forall c \in C_{\mathcal{U}} : h_{1c}^{ID_c} h_{2c}^r)$, where $h_{1c}, h_{2c} \in c$
- Select random values $r_1, r_2 \in \mathbb{Z}_p$
- $AnonClaims_{\mathcal{SP}'} = (\forall c \in C_{\mathcal{SP}}, c \notin C_{\mathcal{U}} : h_{1c}^{r_1} \cdot h_{2c}^{r_2})$, where $h_{1c}, h_{2c} \in c$
- Set $AnonClaims = AnonClaims_{\mathcal{U}} \cup AnonClaims_{\mathcal{SP}'}$

The set $AnonClaims$ is sent as the authentication request to the service provider. The service provider invokes $Encrypt''$ to generate the encrypted session key.

Encrypt''

Define an access tree T where the leaves are members of $C_{\mathcal{SP}}$. A session key is set as the value of the root node of T . The session key is split into shares at each inner node, which is either an AND or an OR node, based on a $n - of - n$ or a $1 - of - n$ scheme. Shares s_i are assigned to leaves. These shares can be combined when moving up the tree structure via the threshold gates to reconstruct the session key.

- $ct = (\forall c \in C, AnonClaim_c \in AnonClaims :$
 $ct_c = Encrypt'(AnonClaim_c, c, s_i), T)$

The service provider sends ct to the user. The user extracts the session key from ct using $Extract''$, as described next.

Extract''

- $Keys = (\forall c \in C_{\mathcal{U}} : KeyGen(claim_c, r))$, where $claim_c$ is a claim instance
- $\forall ct_c \in ct, Key_c \in Keys : s_c = Decrypt(Key_c, ct_c)$
- Replace leaves of T with s_c values
- Reconstruct session key using the s_c values and the threshold schemes applicable to each non-leaf node of T

Limiting the Number of *AnonClaim* values

In the above setup the user is required to send the n *AnonClaim* values to satisfy the service provider's policy. This can be changed to a single *AnonClaim* instance with a modification to the system as follows:

- Identity management system is defined using a single group \mathbb{G} of sufficiently large prime order p , and set $h_1, h_2 \in \mathbb{G}$
- The user sets the ID_{claim} value to be the same for all issued claims the user obtains

With this setup, the user can simply make the initial authentication request using $AnonClaim = h_1^{ID_{claim}} h_2^r$ with random value $r \in \mathbb{Z}_p$.

In response to the authentication request, the service provider sets up the access structure T , as defined in *Encrypt''*, and returns to the user: $ct = ((\forall c \in C : ct_c = \text{Encrypt}'(AnonClaim, c, s_i), T)$. *Extract''* is used to extract the session key from ct by the user.

It is important to note that changing m *AnonClaim* values to a single *AnonClaim* instance is particularly practical when an enterprise sets up an internal identity management system where it is possible to enforce the constraint of the having same h_1 and h_2 values of claim parameters.

3.4.5 Issued Claim Revocation

An important feature in any identity management system is revocation of issued credentials. In a typical identity management system, there is a unique identifier that can be used by the identity provider in setting up a list of revoked identities. Relying parties can block authentication attempts using such lists as blacklists.

In the proposed system, there is no such unique identifier per issued claim. Therefore, in the event that an identity provider revokes a claim issued for a user, it needs to regenerate the claim's *claimkey* and public parameters. Since the issued claim

Table 3.2.
An example of how the c_0' values are stored by an identity provider

User	Claim	req	c_0'
Alice	student	$req_{Alice_{student}}$	$c_0'_{Alice_{student}}$
Alice	adult	$req_{Alice_{adult}}$	$c_0'_{Alice_{adult}}$
Bob	student	$req_{Bob_{student}}$	$c_0'_{Bob_{professor}}$
Charlie	student	$req_{Charlie_{student}}$	$c_0'_{Charlie_{student}}$

values are generated based on the public parameters, this further requires the identity provider to issue new claim instances for all valid users. The revoked user is not issued a new claim and will be unable to authenticate with relying parties, who use updated public parameters for the claim.

Section 3.4.2 defined the value of a claim issued by the identity provider as:

$$claim = (c_0, c_1, c_2) = (claimkey \cdot (req \cdot g_3)^r, g^r, h_2^r)$$

Consider the first component of the above claim value:

$$c_0 = claimkey \cdot (req \cdot g_3)^r = claimkey \cdot c_0'$$

An identity provider stores each c_0' value of each claim that it issues. Table 2 shows an example of how an identity provider may maintain this information. It is important to note that the identity provider is trusted to discard the value of r used to generate the $claim$ ¹. This relieves the identity provider from securing all c_0' components. Furthermore, if an attacker obtains the set of stored c_0' values, they will not be of any help to the attacker in compromising any particular user. This will be further discussed in Section 3.5.

¹Chapter 2 identified the possibility of regenerating and publishing only the first component of a contact key using the stored value r (which was used to generate the key). In contrast, this protocol does not require the issuer to store the value r .

Let req_u denote the req value used by the user, u , running IssueClaim (as defined in Section 3.4.2), $c_0'_u$ denote the corresponding c_0' value, U denote all users with issued claim instances for a particular claim, and u_{revoke} denote the owner of the claim being revoked.

An identity provider performs the following steps to revoke an issued claim:

- Select a random value $\alpha' \in \mathbb{Z}_p$
- Set $g_1' = g^{\alpha'}$
- $claimkey' = g_2^{\alpha'}$
- Select random $s \in \mathbb{Z}_p$
- Generate the set of tuples $C_0 : (\forall u \in U, u \neq u_{revoke} : < (req_u)^s, claimkey' \cdot c_0'_u >)$

The identity provider updates the published values of the claim's public parameters, by replacing g_1 with g_1' , and publishes the set of tuples C_0 along with s . All these values can be published in a manner where the service providers and users can access this information without any further communication with the identity provider. Furthermore, the identity provider may sign this information using a PKI private key for integrity protection.

Any user that obtained a claim instance from the identity provider may update his/her claim's c_0 value with the following steps:

- Obtain a copy of C_0 and s value.
- Set $x = (h_1^{ID_{claim}})^s$
- Look up C_0 with using x in order to obtain
 $< x, c_{0x} >$
- Replace the claim's c_0 with c_{0x}

3.5 Security Evaluation

This section discusses of security and privacy features for each of the solution steps presented in the previous section. Each step is examined closely to emphasize the importance of the security and privacy guarantees provided by the system.

3.5.1 Claim Definition

As described in Section 3.4.1, an identity provider creates a claim definition as an instance of the HIBE scheme with depth 2 [13]. The public parameters and metadata of the claim are made public. Importantly, the *claimkey* value, which is the master key of the scheme, is held secret by the identity provider. This is the only information the identity provider is required to secure with respect to a defined claim.

3.5.2 Claim Issuance

Claim issuance requires an exchange of messages between a user and an identity provider. A user initially sends $req = h_1^{ID_{claim}}$, where ID_{claim} is a random value. The *req* value can be sent to the identity provider in the clear. Due to the hardness of the discrete logarithm problem, an eavesdropper of the channel with computationally bounded resources, who obtain *req*, will not be able to recover the ID_{claim} value without a brute force attack.

It is important to note that an identity provider authenticates a user before issuing an identity claim instance. This scheme does not enforce any specific authentication scheme between the identity provider and the user.

The identity provider returns $claim = (c_0, c_1, c_2) = (claimkey \cdot (req \cdot g_3)^r, g^r, h_2^r)$. Note that *claim* must be transmitted to the user over a secure channel, which prevents an eavesdropper from obtaining *claim*. The following PKI-based scheme can serve as the authentication scheme, which can ensure that *claim* is readable only by the user:

- The user has an asymmetric key pair $(pub, priv)$ to be used for encryption and signing.
- The identity provider registers the user along with any required metadata about the user and stores the user's public key (pub) as a trusted user.
- The user signs the req value with $priv$ to generate σ_{req} and sends (req, σ_{req}, pub) to the identity provider.
- The identity provider authenticates (req, σ_{req}) by verifying σ_{req} , and generates $claim$.
- The identity provider encrypts $claim$ using pub and returns the encrypted $claim$ to the user. This provides the assurance that only the authenticated user is able to decrypt the message and obtain $claim$.

In the case when the user needs to be assured of the authenticity of the identity provider, both parties can use each others asymmetric key pairs to sign and encrypt communication. In practice, this may be trivially realized using a scheme such as SSL with mutual authentication.

At the point of claim issuance, the identity provider is required to maintain the req and the c_0 component of the issued claim instance for revocation purposes. It should also securely discard c_1 and c_2 components. In the case of an attack, where an attacker can obtain the list of c_0 s, that information will not yield any assistance in disrupting the use of issued claim instances. Due to the missing c_1 and c_2 components of the issued claims, an attacker will not be able to extract a session key during an authentication attempt with a service provider. Furthermore, if the attacker obtains req , the attacker will be able to create an *AnonClaim* value, which is of the form $(req \cdot h_2^r)$, to initiate the authentication with a service provider. However, due to the absence of a complete *claim* the attacker will not be able to recover the session key.

It should be noted that any entity can submit *AnonClaim* requests of the form $h_1^x \cdot h_2^y$ to a service provider. The service provider responds with an encrypted

session key value as well. This allows the possibility of a denial-of-service attack on the service provider. However, depending on the implementation, the service provider can employ mechanisms to throttle authentication requests to thwart such attacks.

3.5.3 Authentication

A user generates the $AnonClaim = h_1^{ID_{claim}} \cdot h_2^r$ value as the initial request to a service provider. An attacker who obtains this message will not be able to derive either ID_{claim} or r values. Note that $AnonClaim$ takes the form of a Pedersen commitment value [34], where it unconditionally hides the ID_{claim} and h_2 values. Therefore, it is possible to send this value to the service provider over a public channel. The ct value, as discussed in Section 3.4.3, is essentially a ciphertext of the HIBE scheme [13]. Therefore ct does not divulge any information to an eavesdropper with computationally bound resources. In essence, the authentication protocol with a service provider can safely take place over a public channel.

Consider the case where the service provider may require a policy such as:

$$ClaimSet_1 \text{ OR } ClaimSet_2$$

Based on the approach described in Section 3.4.4, the service provider sets up an access tree where the leaves are encrypted with different claims and the session key is hidden at the root. The user decides which set of claims to use to decrypt the leaves of the access structure to obtain the session key. Since this process occurs at the users site, the service provider will not learn which set of claims were used.

3.5.4 Claim Revocation

Section 3.4.5 proposed publication of a set of values updating current valid claim instances, upon revocation of claim instances. Table 3 shows an example of how this data is organized.

As discussed in Section 3.5.2, the c_0 component of $claim$ can be made public and is useless without the c_1 and c_2 components. The “User” field does not list any

Table 3.3.
An example of how the new set of c_0 values and new public parameter component g_1 are published by an identity provider

g_1, s	
User	c_0
$(req_{Alice})^s$	c_{0Alice}
$(req_{Bob})^s$	c_{0Bob}
$(req_{Charlie})^s$	$c_{0Charlie}$

identifiable information about a user who possesses a valid claim instance. This field contains user's req_{User} values blinded using a random value s , where $(req_{User})^s = h_1^{(ID_{claim} \cdot s)}$. In the case when the req value is never exposed, an attacker will not be able to identify the user upon a claim revocation. This property holds true over multiple cycles of claim instance revocations as well.

It should be noted that in the case when the req values is available to an attacker, the attacker will be able to identify the presence of the user's updated claim component in the revocation information. To eliminate this possibility, leading to the knowledge of the number of currently valid claims, the identity provider can optionally include bogus c_0 values for revoked claims, or pad the total number of entries in the table. Additionally, to prevent an attacker from attempting to mount a denial of service attack by invalidating claim instances of users, claim revocation information is signed by the identity provider.

As discussed earlier, depending on the implementation, users and an identity provider might use asymmetric keys for authentication and confidentiality of exchanged messages. In such a situation req values are not available to an eavesdropper.

Finally, in the case in which an identity provider can be available online, it can carry out an authenticated secure exchange with the user to provide the new c_0 . In this scenario, the identity provider will only have to publish the new g_1 with respect to the claim. Presence of a new g_1 indicates that all users who own an issued claim

instance of the claim must obtain a new c_0 . A user sends the req to identity provider and the service provider encrypts the response, which includes the new c_0 , with the user's public key.

3.5.5 Collusion Resistance

Section 3.4.4 presented the approach for supporting complex service provider policies based on multiple identity claims. The basic scheme requires the user to send a set of

$$AnonClaims_{\mathcal{U}} = (\forall c \in C_{\mathcal{U}} : h_{1c}^{ID_c} h_{2c}^r)$$

values to the service provider. Since these $AnonClaims_{\mathcal{U}}$ values do not disclose any information about the ownership, it is impossible to avoid a situation where two users collude to obtain fraudulent authentication successfully.

The setup described in Section 3.4.4, to limit the number of $AnonClaim$ values, resolves this issue of collusion. The setup requires the user to obtain claim instances using the same ID_{claim} value, enforces the use of a single group \mathbb{G} , and requires that $h_1, h_2 \in \mathbb{G}$ are constant. Due to this modification, ID_{claim} becomes the master key of the set of claims of the user. At this point, if a user is to collude with another, the user will have to give up the master key.

A user initially generates a claim from an identity provider which is the root identity provider. An identity provider will ensure that the user uses the same $req = h_1^{ID_{claim}}$ value during issuance by verifying that the user has a claim issued by a parent identity provider by generating $x = req \cdot h_2^s$ where random value $s \in \mathbb{Z}_p$. The x value is used as an $AnonClaim$ value to encrypt the new issue claim value. The value s is sent to the user along with the ciphertext of the newly issued claim.

Moreover, a user may transfer an $AnonClaim$ value and the corresponding key value to decrypt the encrypted session key, to another user. To avoid such a delegation, the service provider may select a random value $r' \in \mathbb{Z}_p$, set $AnonClaim' = AnonClaim \cdot h_2^{r'}$ and use $AnonClaim'$ to encrypt the session key. The ciphertext

value and r' are returned to the party attempting to authenticate. Now, this party must generate a new key using $KeyGen(claim, (r + r'))$. This modification requires the possession of the *claim* value, which can only be carried out by the user who owns the *claim*.

3.6 Implementation

This section provides an overview of the developed proof of concept application. The complete source code is available as an opensource project [60]. The implementation is based on the Java Pairing Based Cryptography Library (jPBC) [35].

3.6.1 Identity Provider

The identity provider was developed as a Java library. This includes the a database to store defined claims, user registration information, issued claim information, and revocation information. The high-level database schema is shown in Figure 3.7.

Identity provider front-end implementation is written as a web application using the NodeJS/Express framework. This allows defining claims, creating user entries, issuing claims, and revoking issued claims. In addition, this has a set of JSON-based² REST interfaces for service provider and user interaction with the identity provider. The identity provider application is bootstrapped with a private key and the corresponding public key certificate. All public information published by the identity provider is signed with a private key.

A user can be added to this system by providing his/her name and public key certificate. The public key is used to authenticate a user's claim issuance requests and to encrypt issued claim instances.

²JSON (JavaScript Object Notation) is a lightweight data-interchange format, which is easy for humans to read and write, and easy for machines to parse and generate [37].

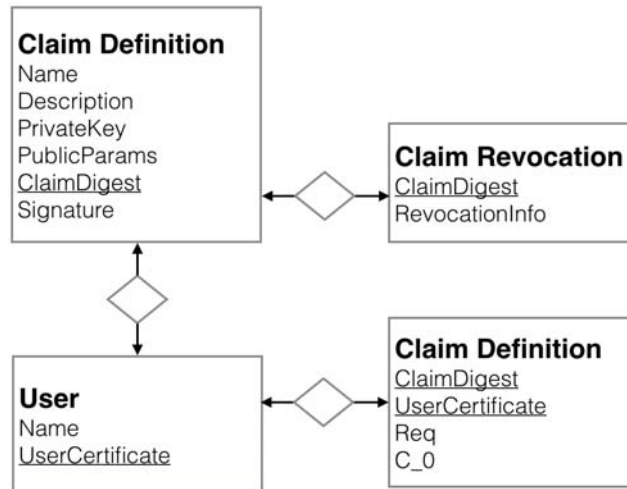


Figure 3.7. Design of the identity provider database. Text in bold indicates the primary keys of each entity.

3.6.2 Identity Provider Client

A user interacts with the identity provider using a client application, *idptool*. This allows a user to connect to the identity provider application using a URL. The user can obtain the list of available claim definitions, and request him/her a claim instance. Issued claim instances are stored in a *claim wallet directory* in the user's home directory. Figure 3.8 shows an example usage of the *idptool* application.

```

$./idptool -action list -url http://ec2-54-82-231-14.compute-1.amazonaws.com:3000
Identity provider at http://ec2-54-82-231-14.compute-1.amazonaws.com:3000 can issue claims for:
student
professor

$./idptool -action req_claim -url http://ec2-54-82-231-14.compute-1.amazonaws.com:3000 -claim student
-user alice -keystore keys/alice.jks -storepass alicekey -alias alice -keypass alicekey
Claim 'student' stored in the claim wallet.
Claim file: w8jHHAj73Z9cHeViRXBi39NMjSAT2c4MWIRdNSYEAwrbv9mUPMFkPyPVnTg1KlOZCqlTnJzGCEs6KxDJWg

```

Figure 3.8. Example of idptool usage

3.6.3 Service Provider and Clients

The service provider is implemented as a node.js RESTful service. This exposes two authentication operations. The first authentication operation generates a challenge based on a *student* claim. The other generates a challenge based on two claims, *student* AND *candidate*.

Each authentication operation encrypts a session key and returns the ciphertext to the client. In the two claim case, the session key is split into two components and encrypted using the two claims. A random element of the target group of the pairing function is used as a session key. This element is split by subtracting it using another random element.

Two client scripts were implemented in JavaScript, which depends on the Java-based implementation of the authentication library. Both use the claim wallet created by the *idptool* application to obtain claim instances. The first script invokes the authentication operation that uses a single claim and the other invokes the operation that relies on two claims. each client decrypt the ciphertext and extracts the session key. In the second case, the client decrypts the two ciphertext values related to the two claims and adds the results together to obtain the session key.

For demonstration purposes, clients invoke another operation on the service by sending an HTTP POST request with the extracted session key as the payload. The service responds with “*Access Granted*” upon the receipt of a valid session key. If the session key is not valid, it responds with “*Access Denied*.” This implementation is evaluated with a the service hosted on an Amazon EC2 instance.

3.6.4 Performance Evaluation

A set of experiments was performed to evaluate the implementation. These experiments were carried out using two systems: an Apple MacBook Pro (CPU: 2.3 GHz Intel Core i7, Memory: 16 GB 1600 MHz DDR3, Operating System: OS X 10.9.3, Java(TM) SE Runtime Environment-1.7.0_45) and an Amazon EC2 instance (model:

Table 3.4.
Claim definition creation time analysis

Time to Generate (ms)	MacBook Pro	Amazon EC2
Claim Definition	14.58	17.56
Digest	0.13	0.13
Signature	0.98	1.28
Storage	0.99	4.04

m3.large, vCPU: 2 Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz, Memory: 7.5 GB, Operating System: Ubuntu Linux, Java(TM) SE Runtime Environment-1.7.0_21). Both systems use SSD storage.

Creation of Claim Definition

Average time to create a claim definition was calculated by averaging the creation of 1000 claim definitions. The identity manager implementation creates a claim definition, a SHA-512 digest of it, signs the content, and stores it in a MySQL database. The average time to store on the Mac was 1ms and was 4ms on the EC2 instance. Time taken to sign was 0.98ms on the Mac and 1.28ms on the EC2 instance. In both systems time taken to create the digest was negligible. The average time to create an identity claim definition was 14.58ms on the Mac and 17.56ms on the EC2 instance. Table 3.4 summarises these results.

Claim Issuance

The average time required to issue a claim instance was analyzed using the two systems. As described in Section 3.4.2, the user first creates a *req* value, which is used

Table 3.5.
Claim instance issue time analysis

Time to Generate (ms)	MacBook Pro	Amazon EC2
Claim Issue Request	1.78	2.12
Claim Instance	8.33	13.14

by the identity provider to generate a claim instance. The performance analysis test first bootstrapped the identity provider with 1000 users and created a single claim definition. It then generated a *req* for each of the 1000 users and invoked the identity manager implementation to create corresponding identity claim instances. Table 3.5 summarises the results.

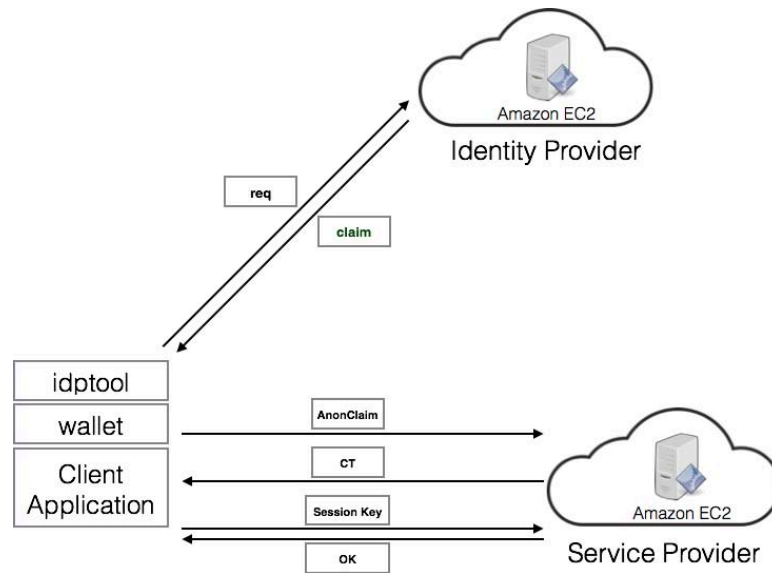


Figure 3.9. Setup of Amazon EC2 remote invocation experiment

The identity provider client application, *idptool*, was evaluated using the identity provider hosted at the Amazon EC2 server. The average time to obtain and store an issued a claim from the remotely hosted identity provider was 315.38 ms.

Authentication

Using a setup as shown in Figure 3.9, the client application authenticated with the service provider hosted at the Amazon EC2 server. First, authenticating with a single claim required an average time of 294 ms. Next authenticating with two claims accounted for an average time of 330 ms. These results are averaged over 100 interactions. Each interaction involves two invocations of the service provider to obtain a session key and present the extracted session key.

It is important to identify the component of time required to invoke a simple authenticate operation without using the proposed scheme. Such analysis will allow isolation of the processing time required for all the additional functions performed when using the scheme. An additional authentication operation was added to the service provider implementation. This authentication operation creates a session key and returns it to the client application. The client application extracts the session key from the response and invokes the service provider using the session key as an argument. The service provider responds with “*Access Granted*” if the session key is valid. Total interaction times of this setup was evaluated over 100 interactions. The average time taken by this base case was 174.69 ms.

This experiment was extended to obtain authentication times of up to 100 claims. A client would create a request with n claims and carry out the above authentication with a service provider, where n ranged from 1 to 100. In addition to the total time, times takes for request creation, authentication call, session key extraction and verification call, were recorded. During request creation, the client application creates the required *AnonClaim* values and their corresponding ephemeral private keys. The *AnonClaim* values are submitted to the service provider application in the authentication request. Service provider encrypts shares of a session key with these *AnonClaim* values using corresponding claims and returns the set of ciphertext values to the client application. The client application uses the ephemeral private keys to decrypt the ciphertext values and reconstructs the session key. Figure 3.10

shows the times taken to authenticate with each number of claims. The spikes in the authentication call durations can be attributed to random network delays.

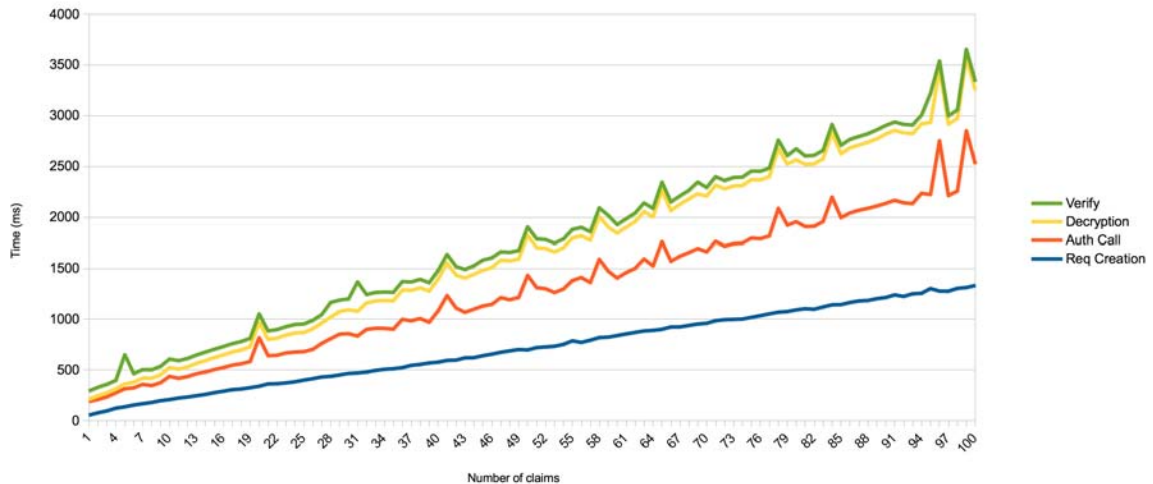


Figure 3.10. Stacked times to authenticate with a service provider

In this experiment, the request creation includes creation of the private keys. This time needs to be spent during the protocol if multiple *AnonClaim* values were used. However, as discussed in Section 3.4.4, it is possible to reduce the number of *AnonClaim* values to one *AnonClaim* value. This technique was used to generate a single *AnonClaim* while the multiple ephemeral private keys required were created using a multithreaded approach. In this approach, the main client application thread carries out the authentication call while each ephemeral private key is being created by its own independent thread. The stacked times taken in this approach are shown in Figure 3.11. This result shows a clear reduction in the total times.

Figure 3.12 compares the request creation times of the multiple *AnonClaims* approach with the single *AnonClaim* approach. This result clearly shows the efficiency of the single *AnonClaim* approach due to the possibility of concurrently computing the private keys. Moreover, it should be noted that, it is possible to carry out the encryption and decryption of the session key shares concurrently as well.

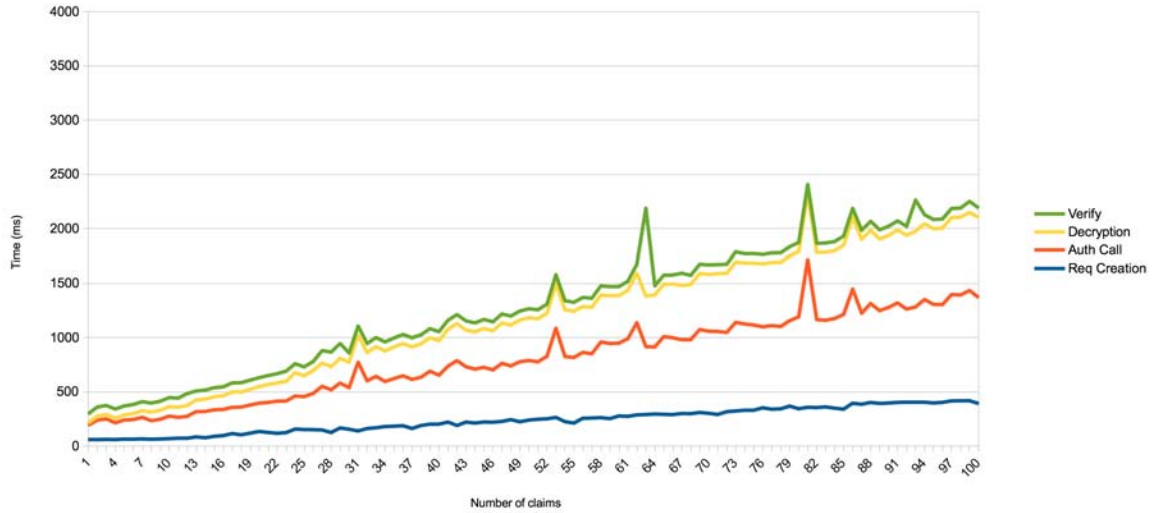


Figure 3.11. Stacked times to authenticate with a service provider using concurrent key derivation

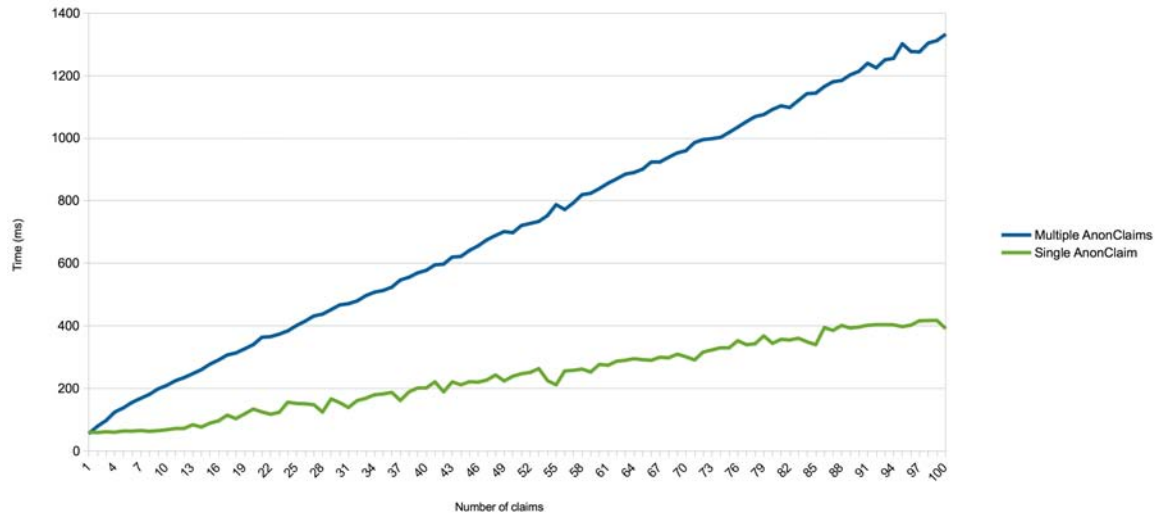


Figure 3.12. Comparison of key derivation times

These performance results show very low overhead due to the use of the proposed techniques compared to traditional means of authentication. Therefore, it is evident that the protocols proposed here are practical for the use in modern Web applications.

3.7 Integration with Service Compositions

In service oriented architecture (SOA), a service may compose of multiple services. Figure 3.13 shows an example of a service that depends on a set of other services. Such service compositions can utilize the proposed protocols using two different approaches. First approach allows direct authentication of the user via the main service. Second approach allows inner service providers to remain anonymous and use claims to prove ability to process user information.

3.7.1 Direct User Authentication

Direct user authentication scenario allows each inner services to ensure that the user owns all required claims according to each service policy.

Setup

The main service advertises the authentication policies of the services it immediately depends on. In other words, in the service invocation tree, the authentication policy of a parent service is the union of its own and its children. Each service includes the public key certificate of an asymmetric key pair in its policy in addition to the claim definitions.

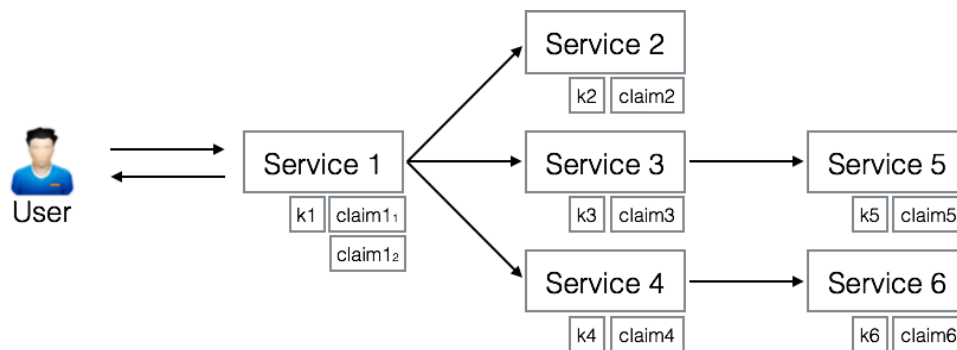


Figure 3.13. A service composition with expected claims

Authentication

To authenticate with such a scenario, a user generates a set of *AnonClaim* values. Note that, the user may just present a single *AnonClaim* value in the case where all the services use claim definitions with the same h_1 and h_2 values, as proposed in Section 3.4.4. The user sends the *AnonClaim* value to the main service. For example, this is Service 1 in Figure 3.13. Each service propagates this *AnonClaim* value to its immediate child services. Upon receiving this value, each service selects a random value $r_{service_i}$ and computes $AnonClaim_{service_i} = AnonClaim \cdot h_2^{r_{service_i}}$. Then, each service generates a session key value sk_i and encrypts it using $AnonClaim_{service_i}$ to obtain $ct_i = Encrypt'(params_{claim}, AnonClaim_{service_i}, sk_i)$. Values $ct_i, r_{service_i}$ along with service policy and service public key are returned to the caller. When a service receives such a response from a child service, it returns the values to the caller. This approach returns a set of $ct_i, r_{service_i}$ values to the user that invoked the main service.

After receiving the response from the main service, the user generates the corresponding private key values for each $AnonClaim_{service_i}$ value. These private keys are used to decrypt each ct_i value and obtain each session key value sk_i . In a situation where the user needs to carry out multiple requests with the service topology, he/she may use the public key value of each service to encrypt the corresponding session key.

The service may be able to provide a response to an authenticated user without the need for multiple service calls. In such a scenario, the service may encrypt the response payload with the session key value. This encrypted payload will be included in the response to the user along with the encrypted session key. User will be able to decrypt the encrypted payload only if he/she owns claims according to that service policy.

Note that, due to the use of the $AnonClaim_{service_i}$ values, each service ensures that is not possible for the owner of a claim to delegate a private key corresponding to an *AnonClaim* value.

3.7.2 Service Provider Capabilities

The main service may have the option of selecting one of several services to outsource operations. For example, an online vendor service may outsource credit card processing to one of many banks. A trusted authority will issue claims to these banks to certify that they are allowed to and are capable of processing credit card information. “*Allowed to process credit card information*” is the identity claim in this example.

The main service first contacts an inner service to obtain an *AnonClaim* value. This *AnonClaim* value is sent to the user at the start of a transaction. The user evaluates the *AnonClaim* value and claim definitions from the initial service and encrypts information with the *AnonClaim* value. These ciphertext values are provided to the main service which routes them to the inner service. A service can decrypt a particular ciphertext value, only if it own a claim issued by a trusted authority. Also, no other entity with the same capability will be able to decrypt the content as well.

4 CONSUMER ORIENTED PRIVACY PRESERVING ACCESS CONTROL FOR ELECTRONIC HEALTH RECORDS

4.1 Introduction

An investigation by the committee on Quality of Health Care in America, Institute of Medicine, performed in 1997 concluded that at least 44,000 and up to 98,000 people die in the United States each year because of medication errors [61]. We infer that among the causes of these errors are conflicting treatments and incomplete information. Several countries are working towards implementing national systems for Health Information Exchange (HIE). Such systems shall share Electronic Health Records among healthcare providers and reduce medical errors. The Netherlands was among the pioneers in this subject, however; the project was ceased because of privacy concerns [62]. Other European countries such as UK and France have also ceased their projects, with the exception of Denmark [63]. The critical issue for the success of these projects is consumer privacy.

HIEs have been provider-oriented with very little consumer involvement. HIEs share EHRs electronically between health care providers in accordance with nationally recognized standards [64]. These exchanges usually involve trusted third parties such as cloud services (aka EHR banks or HIE services) to facilitate EHRs' storage, sharing and access. However, consumers (patients or their authorized representatives) have no visibility or control in these transactions. They are typically carried out without consumers knowledge or approval. HIEs are plagued by privacy issues arising due to the opaque data sharing of consumer EHRs. Consumer risks and harms arising due to opaque data sharing of EHRs are well documented [65].

This work explores the privacy issues related to the access control of patient information and their medical data managed by external service providers. Consider

a medical laboratory that uses a cloud based EHR bank to store test results. The EHR bank exposes a service to provide access to the data it stores. Patients and their representatives, physicians, and medical laboratories can use this service to access the data. While there are various means of securing the stored sensitive data and accessing it safely, it is important to investigate the possible privacy implications, due to knowledge that may be inferred using side channels based on access patterns.

A patient consults Dr. A who prescribes a blood test. The patient visits a medical testing facility (lab) with Dr. As prescription and provides a blood sample for testing. Once the results are ready, the lab uploads these results to an EHR bank and notifies the doctor and the patient. Dr. A, using her credentials, obtains the test results using an HIE service exposed by the EHR bank. If this service employs traditional means of authentication, such as username-password pairs or public key infrastructure based authentication, which are *linkable*, the EHR bank can infer additional information about Dr. A. Furthermore, in the case where another physician Dr. B, a cancer specialist, accesses this information, the EHR bank will learn about Dr. Bs interest in this data. Using both physicians specialization, the EHR bank will be able to infer even more information. For instance, based on access patterns, the EHR bank will not only infer consumer information but will also be able to identify groups of physicians who collaborate frequently. Therefore, it is very important to provide means to limit the privacy implications. In order to provide visibility, the level of patient empowerment has to be enhanced in HIEs [66]. There is a strong need to add consumer mediation and control in HIE services to address privacy concerns [64]. This paper proposes an approach to nullify the effects of the use of distinguishable identities for authentication and authorization by a service. This approach enables consumers to establish and enforce individual privacy requirements by allowing a specific EHR (complete or partial) to be used in an HIE.

It is also important to note that, the consumer, as the owner or guardian of her EHRs, should have visibility and control over how these records are accessed by various parties. Consumer preferences may change over time, for e.g., depending on

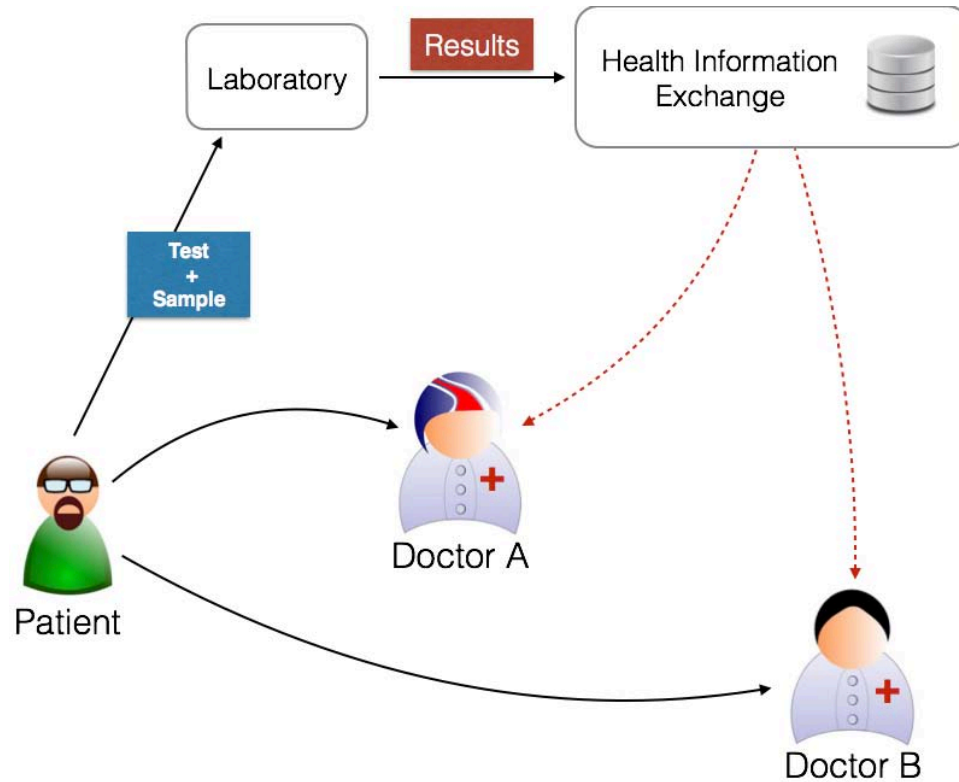


Figure 4.1. Medical information exchange setup

their health status so these features are essential for a trusted healthcare system. Also, consumer controlled HIEs are necessary for a diverse society because people have different privacy preferences. The techniques used for managing EHRs and HIEs must provide dynamic and fine-grained access control with revocation ability. They should be able to protect consumer privacy and the privacy of entities that participate in HIE. We believe consumer mediation and control in HIEs will resolve trust issues and privacy concerns in HIE services.

4.2 Motivation

Privacy preserving consumer centric EHR access is of utmost importance and has many benefits, as follows:

- **Accurate identification, association and propagation of health data.** Consumers are able to indicate any missing or mismatched information [66].
- **Reduce provider liability during HIEs.** Practitioners make decisions based on EHRs from an HIE. These EHRs may come from unknown providers which increases the malpractice liability. However, if consumers grant access to their digitally signed EHRs, the provider liability is reduced [66].
- **Increased consumer satisfaction by reducing linkability and inferencing.** Consumers may not want to make certain health conditions public, for e.g. a positive diagnosis of cancer. Allowing consumers to sequester and selectively disclose their information will improve consumer satisfaction [66].
- **Reduced legal consequences/confusions.** Law requires HIEs to filter certain information, for e.g. a positive diagnosis of AIDS, in order to protect consumer privacy. Moreover, inferencing based on other information, for e.g. prescriptions, consulting practitioners, pattern of specific medical tests etc., may still link consumers with the diagnosis. However, consumer approved sharing of EHRs is accepted by law [66].
- **Improved visibility.** EHR bank and HIE services are not government entities and are not required to explain how EHRs are shared (or sold). Consumers may be prone to economic losses due to the lack of transparency and accountability [66].
- **Reduced healthcare costs.** Consumers will be able to reuse existing EHRs such as test results leading to cost reductions.

Table 4.1 presents the notations used for each entity in the scenarios addressed. *L* generates a new *EHR* about a particular *U* and sends this record to *HIE* to store. *HIE* is responsible for controlling access to this data. *HIE* has access to plain text data of *EHR* and can provide value added services on this data, such as rendering

Table 4.1.
Entities and notations

Entity	Symbol
Generator of health records	L
Consumer of health records	D
Patient/Owner of health records	U
Health Information Exchange Service	HIE
Electronic Health record	EHR

the EHR using a graphical user interface. The following are the security and privacy properties that need to be provided by the system:

- U should be able to define (add/remove) who can access this information.
- HIE should not be able to identify U and subsequent accesses of HIE by U must be unlinkable.
- HIE should not be able to link multiple EHR s to any U .
- HIE should not be able to analyze access of different roles of consumers based on their credentials.
- Only D s authorized to access an EHR should be able to access it.
- Any D authorized to access an EHR should be able to access it without revealing any personal identifiable information.
- All access to EHR s by all D s should be unlinkable.

4.3 Related Work

Examples of commercial EHR services include Google Health [67] and Microsoft HealthVault [68]. Google Health initiative was unsuccessful and has been discontin-

ued. Privacy issues leading to the lack of trust was one of the reasons it was not adopted by masses. Microsoft HealthVault has not completely matured yet.

Protecting the confidentiality of the EHR while ensuring the usability of the system and limiting the resistance of the system actors to the use of the system is deemed an important challenge. Several approaches and solutions have been proposed or implemented. For instance, The Netherlands implemented a centralized national EHR system [69]. The system uses a passive confidentiality protection strategy: control and warn [70]. In this strategy, a monitoring system logs all accesses to EHR given that all care providers are able to access all patient data; while the warning system evaluates the compliance of each access with access rules that implement the law. Groot et al. found that the average unlawful accesses are about 9% from all requests by care providers [70].

Salih et Al. [71] proposed a distributed approach for protecting the confidentiality of EHR. The idea is to encapsulate the EHR data of each patient along with the access policies into an entity called active bundle [72] and to store the bundle in the repository of the primary health care provider of the patient. The active bundle encompasses a virtual machine that enforces the policies and performs protection mechanisms, such as apoptosis. The health care provider can search for the active bundle, query it locally for requested information, and update it as needed. The bundle can also record a log of accesses and call the main repository if required.

MyDataCan [73] supports the creation of consumer-controlled HIE services and helps consumers manage access to their EHRs [74] and [75]. It allows easy access to health data through apps and enables consumers to set goals and work towards improving their health.

There are multiple ongoing pilot programs for EHR banks [76]. For instance, a project at Madigan Army Medical Center focused on connecting DoD and VA healthcare systems.

4.4 Preliminary Notions

Figure 4.2 shows the typical setup of an identity provider which issues credentials to a user, who authenticates with a service provider using those credentials.

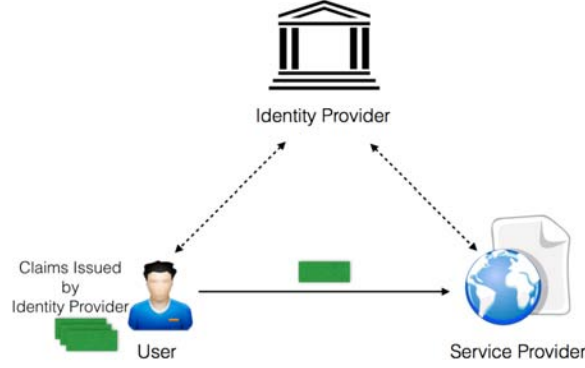


Figure 4.2. Entities of an identity management system

The identity management system defined in Chapter 3 defines the following operations:

GenClaim

IssueClaim

Authenticate

RevokeClaim

GenClaim

Let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ be a bilinear map where \mathbb{G} is a group of prime order p . GenClaim generates a new claim as a set of parameters $param = (g, g_1, g_2, g_3, h_1, h_2)$ and a key, $claimkey = g_2$, where g is a generator of \mathbb{G} and a random $g_2 \in \mathbb{Z}_p$, random $g_2, g_3, h_1, h_2 \in \mathbb{G}$ and $g_1 = g$. This is invoked by an identity provider to setup a new identity claim definition. Identity claim instances are issued to user based on this

definition. *claimkey* is kept private at the identity provider and $param, e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ is published to be accessed by users and service providers along with a semantic description of the claim.

4.4.1 IssueClaim

Claim issuance starts with a user requesting a claim instance from the identity provider using a *req* value defined as: $req = h_1^{ID_{claim}}$, where $ID_{claim} \leftarrow \mathbb{Z}_p$. The identity provider outputs $claim = (c_0, c_1, c_2) = (claimkey \cdot (req \cdot g_3)^r, g^r, h_2^r)$ and transmits *claim* to the user. This exchange is carried out over a secure channel between the user and the identity provider.

4.4.2 Authenticate

A service provider who wishes to authenticate a user with the possession of a claim issued by a particular identity provider will indicate such a requirement in its authentication policy. To authenticate, a user sets $AnonClaim = h_1^{ID_{claim}} \cdot h_2^r$, where $r \leftarrow \mathbb{Z}_p$. *AnonClaim* is sent to the service provider. The service provider generates a random *sessionkey* value and encrypts it using the *Encrypt'* defined in Section 2.5.4: $ct = Encrypt'(param, AnonClaim, session\ key)$. The service provider returns *ct* value to the user and the user then invokes $Extract(ct, param, claim)$ to obtain the *session key*.

In the case where the users does not possess a valid claim instance, it will not be possible to extract the session key hidden in the challenge sent by the service provider. Hence, the user will not be able to setup the authenticated secure channel.

4.4.3 RevokeClaim

In a situation where an issued claim is revoked, the identity provider generates a new $\alpha' \leftarrow \mathbb{Z}_p$ and publishes the new public parameters along with a table of blinded

values that current set of users can use to update their *claim* values. The published information does not compromise the security of the *claim* values held by any user.

4.5 Proposed Solution

As shown in Figure 4.1, the general workflow supported by this solution is as follows¹: Patient (U) visits a physician and the physician (D) requests a test. U visits a laboratory (L) with a prescription issued by D and L performs the test. L submits the generated results (EHR) to the health information exchange (HIE). U grants access to D where HIE enforces rules set by U . Using the following set of steps in creating and accessing each EHR instance will enforce all the privacy properties identified in Section 4.2.

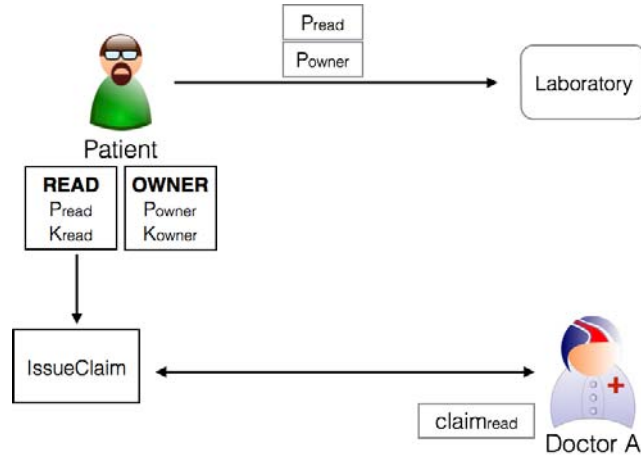


Figure 4.3. User setting up the claim definitions and issuing a read claim to a doctor

4.5.1 Initial Setup

When D requests a test, U creates two claim definitions $ClaimDef_{read}$ and $ClaimDef_{owner}$ using $GenClaim$ as discussed in Section 4.4. These are to be as-

¹Note that the solution is not limited to this example scenario used to establish the protocols.

sociated with the *EHR* instance to be generated as a result of the test. *D* carries out *IssueClaim* with *U* and obtains $claim_{read}$ based on $ClaimDef_{read}$. P_{read} and P_{owner} , which are parameters of $ClaimDef_{read}$ and $ClaimDef_{owner}$ respectively, are given to *L*. This workflow is shown in Figure 4.3.

As shown in Table 4.2, *U* stores the value $(req \ g_3)^r$, used in calculation of c_0 of $claim_{read}$, along with an identifier of *D*. These values are required when *U* needs to revoke access to any *D*.

Table 4.2.
Information stored for each user upon claim issuance

P_{read}		
Identity of <i>D</i>	c_0	req
Dr. Alice	$(req_{Alice} \ g_3)^{r_1}$	req_{Alice}
Dr. Bob	$(req_{Bob} \ g_3)^{r_2}$	req_{Bob}
Nurse Nancy	$(req_{Nancy} \ g_3)^{r_3}$	req_{Nancy}

Once the result (*EHR*) is ready, *L* sends *EHR* to *HIE* along with P_{read} and P_{owner} . *HIE* stores *EHR* and associates P_{read} and P_{owner} with it.

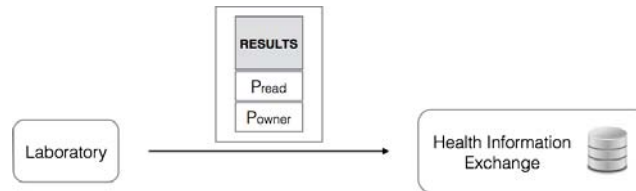


Figure 4.4. Lab sending an electronic health record to a health information exchange

4.5.2 Access Results

Upon receipt of an *EHR*, *HIE* may advertise the value g from P_{read} . This g serves as an identifier when interested parties attempt to access the *EHR*.

To obtain an *EHR* stored at *HIE*, *D* first sets up an authenticated secure channel with *HIE*. As shown in Figure 4.5, *D* first generates a fresh *AnonClaim* instance, and sends it to *HIE* along with g of P_{read} associated with the *EHR*.

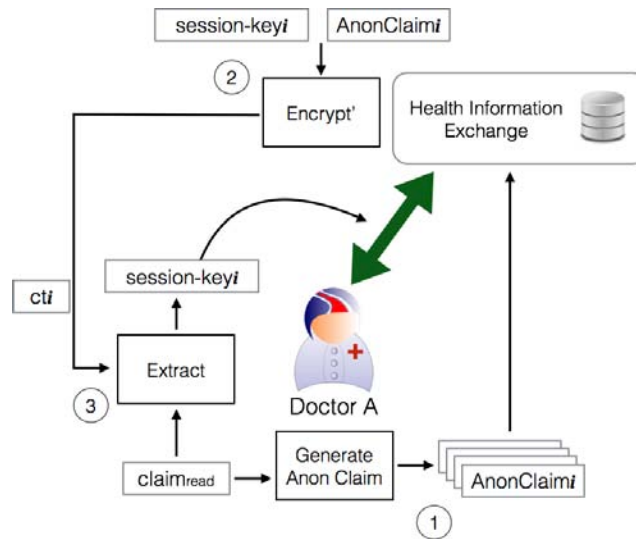


Figure 4.5. Doctor authenticating with a health information exchange

Upon receipt of *AnonClaim* value and g the *HIE* generate a challenge (*ct*) value using the following steps:

Generate ephemeral *session key* value. This is a key of any chosen symmetric key encryption algorithm, such as AES.

Look up the P_{read} value using g and calculate *ct* value:

$$ct = \text{Encrypt}(P_{read} \parallel \text{AnonClaim} \parallel \text{session key}).$$

The *ct* value is sent to *D* who will invoke $\text{Extract}(ct \text{ param } claim)$ to obtain the *session key* value. This *session key* is used to establish the secure channel between

D and HIE . Note that, HIE can send the encrypted EHR value using $session - key$ along with ct as well.

4.5.3 Access Revocation

In a situation where U decides to prevent any D_i from accessing an EHR , U carries out the following steps:

- Generate $\alpha' \leftarrow \mathbb{Z}_p$ where p is the order of \mathbb{G} in P_{read} .
- Set new $claimkey = g_2^{\alpha'}$.
- Remove D_i 's entry from the list of r values maintained (Eg. Table 4.2).
- Calculate new c_0 values for each remaining D s using stored $(req \cdot g_3)^r$ of each D .

$$c_0 = claimkey \cdot (req \cdot g_3)^r.$$

Each of these new c_0 values are listed in a table associated with blinded identity values of D . Table 4.3 presents an example of such a table with updated c_0 values.

Table 4.3.
Example of published re-key information

Updated P_{read}, β	
Blinded Identity of D	Updated c_0
$(req_{Alice})^\beta$	$claimkey \cdot (req_{Alice} \cdot g_3)^{r_1}$
$(req_{Bob})^\beta$	$claimkey \cdot (req_{Bob} \cdot g_3)^{r_2}$
$(req_{Nancy})^\beta$	$claimkey \cdot (req_{Nancy} \cdot g_3)^{r_3}$

U sets up an authenticated secure session as with HIE using P_{owner} and sends the updated P_{read} along with the list of updated c_0 values. HIE updates its records.

Note that, the g value of P_{read} did not change and this can still be used by a D to look up the information. A D can query HIE to obtain the latest P_{read} and the set of c_0 values and update its *claimkey* value by updating the c_0 value.

4.5.4 HIE Access Policies

It is important to note that the HIE may impose further constraints on the $EHRs$. These constraints may lead to potential compromise of consumer privacy. For example, due to regulations, HIE may require all physicians who access this information, prove that they are licensed. Also, the same HIE may allow a nurse to access this information provided she is the head nurse. This policy may be expressed as :

$$\begin{aligned} &\{(\text{Doctor OR Head Nurse}) \text{ AND } ClaimDef_{read}\} \\ &\text{OR} \\ &ClaimDef_{owner} \end{aligned}$$

Authentication scheme proposed in Chapter 3 can be used here to satisfy this policy, while preventing the HIE from inferring statistics about the usage of different groups of consumers. The secure channel establishment protocol remains the same as in Section 4.5.2 with the exception of the *session – key* being hidden at the root node of an access tree structure. The leaves of this access tree correspond to each claim required in the policy and the inner nodes represents the AND and OR conditions. The *session – key* is split and distributed down the access tree structure and the shares at the leaves are encrypted by the corresponding claims. U extracts the *session – key* by decrypting the leaves and combining key shares based on the access structure. The main advantage with this approach is that, HIE is not aware of the claims used by U to recover the *session – key*.

4.6 Evaluation

This work proposes a system for providing authenticated access to sensitive health records. The owner of the information is given complete control over managing access. This section presents a discussion of the main features of the system.

4.6.1 Privacy of Authentication Protocols

One of the main features of the proposed system is preservation of privacy of those parties who access the records. U as the owner of the records provides the two sets of public parameters P_{read} and P_{owner} to the generator of a record. This entity forwards the EHR along with P_{read} and P_{owner} to HIE . With the assumption that, the EHR does not contain any identity information, and since P_{read} and P_{owner} generated as defined in Section 4.4, the HIE is not able to associate any entity with the EHR .

During authentication, a consumer of the EHR initially sends $AnonClaim_i$ value. This is of the form $h_1^{ID_{claim}} \cdot h_2^{r_i}$, which is similar to the construction of the Pedersen commitment [34]. Therefore each $AnonClaim_i$ value unconditionally hides ID_{claim} and r_i values.

When the HIE has an access policy such as $ClaimSet_1$ OR $ClaimSet_2$, HIE constructs an access structure as defined in Section 4.5.4. The consumer has the option to extract the *session – key* value stored at the root of the access structure by decrypting the leaves using either $ClaimSet_1$ or $ClaimSet_2$. It is very important that this decision is not visible to the HIE . This prevents the HIE from being able to determine the distribution of groups of consumers who access each EHR using either $ClaimSet_1$ or $ClaimSet_2$.

4.6.2 Privacy of Access Revocation

An owner publishes a partial component of a consumer's private credentials during revocation. This is of the form $c_{0i} = claimkey \cdot (req \cdot g_3)^r = g_2^{\alpha'} \cdot X_i^r$. Security of this

construction can also be argued based on the Pedersen commitment [34], where c_{0i} unconditionally hides the r and α' . Note that, an eavesdropper may obtain req during claim issue protocol. However, even if such an attacker can construct $X_i = req \cdot g_3$ the above argument holds.

The c_0 values of each consumer is published as shown in Table 4.3. The key column of the table uses a value of the form req^β where $\beta \in \mathbb{Z}_p$ and $req = h_1^{ID_{claim}}$, where req is the value that the consumer initially sends the owner to obtain a credential. This initial exchange to obtain a claim occurs over a secure channel. Therefore, a party other than a consumer will not be able to query the re-key information table. Furthermore, an attacker who obtains multiple tables of re-key information will not be able to make any associations.

4.6.3 Value Added Services

A physician who possesses credentials to access several health records may *unlock* those records, group them, and sets up a session with the *HIE*, which provides a dashboard to carryout further analysis on those records. This will allow the *HIE* to identify a set of possibly related health records. This can also materialize, if the owner re-uses the same P_{read} to issue credentials to access multiple records. These situations may allow the *HIE* to possibly prompt another consumer who groups a subset of health records, which someone else might have grouped with additional records that he/she might find useful. Allowing such configurations may be safe with respect to owner/consumer privacy as long as the *HIE* does not learn any additional information about the owner or the consumers.

5 REVOCATION OF CIPHERTEXT POLICY ATTRIBUTE BASED ENCRYPTION KEYS

Attribute based encryption (ABE) is a public key encryption scheme. ABE addresses the problem of encrypting data once, to be decrypted by a set of users who possess a set of attributes or credentials. Such a system allows a trusted third party to issue private keys to users with respect to attributes. A data owner may use the public key associated with these attributes to encrypt a message and publish it. This message can be now decrypted only by those users who have the private keys corresponding to the attributes of the public keys used to encrypt a message.

Bethencourt et. al. [59] describe the ciphertext policy attribute based encryption (CP-ABE) scheme, which presents four main algorithms: Setup, Encrypt, KeyGen, and Decrypt. Setup creates a public key (PK) and a master key (MK). A user encrypts a message (M), using an access structure (T) that is based on a set of attributes, using $\text{Encrypt}(\text{PK}, \text{M}, \text{T})$. The party who owns MK executes $\text{KeyGen}(\text{MK}, \text{S})$ to output a secret key (SK). S denotes the set of attributes included in the secret key. A user who has a secret key, use $\text{Decrypt}(\text{CT}, \text{SK})$ to obtain M, where CT is the ciphertext output by Encrypt.

Typically, Setup is carried out by a third party trusted by both data owners and users (e.g., a certificate authority). Such a trusted third party must be able to revoke a user's secret key. After revocation, the owner of the secret key will not be able to decrypt any future messages encrypted using the public key associated with the attributes owned by him/her.

This chapter presents an approach for the trusted third party to revoke a user who owns a CP-ABE secret key. This scheme does not rely on a private channel between the user and a trusted third party. This approach first resets the master key of the

scheme and then publishes a set of re-key information with components for each valid user. Furthermore, there is no requirement for this trusted third party to be online.

5.1 Ciphertext Policy Attribute Based Encryption Keys

This section provides an overview of the keys used in the CP-ABE scheme. Setup algorithm generates a master key (MK) and public key (PK). The party that runs Setup keeps the master key secret and makes the public key is available publicly. This entity generates a set of secret keys and transmits them to each user.

A public key of CP-ABE is generated as

$$PK = \mathbb{G}_0, g, h = g^\beta, f = g^{1/\beta}, e(g, g)^\alpha$$

where \mathbb{G}_0 is a bilinear group of prime order p , g is a generator of \mathbb{G}_0 and $e : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ is a bilinear map. α and β are randomly chosen from \mathbb{Z}_p .

The master key is

$$MK = (\beta, g^\alpha)$$

A secret key for a user is generated based on a set of attributes denoted by S . The owner of the master key runs $\text{KeyGen}(MK, S)$, which first selects random values $r \in \mathbb{Z}_p$ and $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$ and outputs

$$SK = (D = g^{(\alpha+r)/\beta}, \forall j \in S : \{D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j}\})$$

$$SK = (D, \forall j \in S : \{D_j, D'_j\})$$

H is a hash function defined as $H : \{0, 1\}^* \rightarrow \mathbb{G}_0$.

Note that, the value D is the only component of the secret key that is related to the master key. Therefore, in a situation where the master key changes the only component of each secret key that is affected is D . Revocation scheme presented in Section 5.2 relies on this property of CP-ABE secret keys.

5.2 Revocation

The issuer, at the point of creating a secret key for a user, creates a random value $r \in \mathbb{Z}$. This value will be stored at the issuer permanently along with an identifier of

the user. In addition to r , KeyGen requires the creation of a set of random $r_j \in \mathbb{Z}$ values for each $j \in S$, where S is a set of attributes. Note that these values are not stored at the issuer. Table 5.1 shows an example of the values stored for Alice, Bob and Charlie.

Table 5.1.
User key information stored at the issuer

User	Random (r)
Alice	r_{Alice}
Bob	r_{Bob}
Charlie	$r_{Charlie}$
Nancy	r_{Nancy}

Let the original set of users be denoted by U and let U' be the set of users with the secret keys to be revoked. After revocation, the set $(U - U')$ will still be able to use their secret keys to decrypt content encrypted using the corresponding PK . However, the set of users in U' will not be able to decrypt such content. Let $user_random(u)$ return the stored r value of a given user u . Following steps are taken to revoke keys of the users in U' .

- Generate new random values $\beta' \in \mathbb{Z}_p$ and $\alpha' \in \mathbb{Z}_p$
- Let set $RK = \forall u_i \in (U - U') : \{(u_i, D_i^{new})\}$
where, $D_i^{new} = g^{(\alpha' + user_random(u))/\beta'}$
- New public key: $PK' = \mathbb{G}, g, h = g^{\beta'}, f = g^{1/\beta'}, e(g, g)^{\alpha'}$
- Publish both PK' and RK as re-key information

Table 5.2 shows an example of re-key data of three users Alice, Bob and Charlie and the updated public key.

Table 5.2.
Re-key information to be published

User	D^{new}
Alice	D_{Alice}^{new}
Bob	D_{Bob}^{new}
Charlie	$D_{Charlie}^{new}$
PK'	

When re-key information is available, owners of secret keys may update their secret keys using published data. Furthermore, all parties should replace PK with PK' . Therefore, PK' will be used to encrypt data.

It is important to note that this scheme only allows revocation of a complete secret key. It does not allow revocation of attributes of an existing user. An issuer will be able to append additional attributes to the user by sending only components related to those attributes since the issuer stores the r value related each secret key. Note that, secret key takes the form, $SK = (D, \forall j \in S : \{D_j, D'_j\})$. To add a new attribute s_i to SK , the issuer computes $D_i = g^r \cdot H(j)^{r_i}$, $D'_i = g^{r_i}$, where random value $r_i \in \mathbb{Z}_p$. The values D_i and D'_i must be transmitted via a secure channel to the owner of SK .

5.2.1 Security Evaluation

During a revocation, the only information available to an eavesdropper are PK' and RK . Since $f = g^{1/\beta'}$ any party can compute the value $f \cdot D_i$. This yields $g^{(\alpha' + r_i)}$. Note that, α' and all r_i values are kept secret at the trusted third party. Therefore, due to the hardness of the discrete logarithm problem this does not leak any information about α' or r_i values.

Furthermore, since D_i^{new} is bound to a particular user due to the use of r_i value that is specific to that user, no user will find any other user's D_i^{new} component useful.

6 SUMMARY

This work initially addressed privacy issues of distributing messages in an anonymous social messaging scheme. In this scheme, one user has a set of contacts, and the contacts do not know each other. If one contact receives a message from the main user, he/she can redistribute this message to other contacts. Such redistribution occurs via a public channel and is based on requests from the contacts who need the message. During this message distribution, all parties should maintain their anonymity. A cryptographic protocol was presented to address this problem. A contact publishes a request to a public channel. Another contact responds to this request by encrypting a message with the proposed encryption scheme. The proposed protocol supports dynamic sets of user contacts. This is where a user may remove a contact from the trusted set of contacts. In such a situation, the remaining set of contacts is updated with new key material using the public channel. A detailed security analysis is presented to emphasize the privacy features provided by the protocol. As a proof of concept of the proposed functionality, the cryptographic primitives were developed as a library. This library was used to develop a demonstration application and an experimentation framework.

Based on the encryption scheme proposed in the above work, a novel identity management system was created to address the privacy concerns of linkable authentication of users. This scheme can tolerate an offline identity provider, where the user can authenticate with a service provider without any interaction with the identity provider. Furthermore, the issued claim instances can be used together to support complex authentication policies of the service provider, while preventing collusion among users who own subsets of required claims. An innovative approach was employed to revoke a user's credentials and update the set of authorized users using only public information. A detailed analysis of security and privacy features of the proposed system

was presented, followed by implementation details of the proof of concept application and performance evaluation. Results of the performance evaluation indicate the practicality of using the proposed authentication protocols in modern applications.

A novel approach was proposed to allow the user to have full control over his/her electronic health care records, hosted at a health information exchange, while improving privacy of those parties who access the records. Protocols between the owner, generator of data, health information exchange and the consumers of the health records were presented. The privacy properties guaranteed by the protocols are unlikability of subsequent usage sessions of consumer interactions with the health records and the inability of the health information exchange to profile classes of consumers in the case of complex access policies. In addition to these privacy properties, proposed techniques allow the owner to manage fine-grained access to the health records.

LIST OF REFERENCES

LIST OF REFERENCES

- [1] Brian Y. Lim and Anind K. Dey. Toolkit to support intelligibility in context-aware applications. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, Ubicomp '10, pages 13–22, New York, NY, USA, 2010. ACM.
- [2] Leonard H. Grokop, Anthony Sarah, Chris Brunner, Vidya Narayanan, and Sanjiv Nanda. Activity and device position recognition in mobile devices. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 591–592, New York, NY, USA, 2011. ACM.
- [3] A. Henpraserttae, S. Thiemjarus, and S. Marukatat. Accurate activity recognition using a mobile phone regardless of device orientation and location. In *Proceedings of the 2011 International Conference on Body Sensor Networks (BSN)*, pages 41–46, May 2011.
- [4] FoxNews. 7,500 online shoppers unknowingly sold their souls. <http://www.foxnews.com/tech/2010/04/15/online-shoppers-unknowingly-sold-souls/>. [Last accessed: July 2014].
- [5] R. Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security Privacy*, 9(3):49–51, May-June 2011.
- [6] Bill Miller and Dale Rowe. A survey scada of and critical infrastructure incidents. In *Proceedings of the 1st Annual Conference on Research in Information Technology*, RIIT '12, pages 51–56, New York, NY, USA, 2012. ACM.
- [7] Mat Honan. How apple and amazon security flaws led to my epic hacking. <http://www.wired.com/gadgetlab/2012/08/apple-amazon-mat-honan-hacking/>. [Last accessed: July 2014].
- [8] Wikipedia. Anonymous (group). [http://en.wikipedia.org/wiki/Anonymous_\(group\)](http://en.wikipedia.org/wiki/Anonymous_(group)). [Last accessed: July 2014].
- [9] The Huffington Post. Libya internet shut down amid protests later restored. http://www.huffingtonpost.com/2011/02/18/libya-internet-shut-down-n_825473.html. [Last accessed: July 2014].
- [10] Christopher Williams. How egypt shut down the internet. <http://www.telegraph.co.uk/news/worldnews/africaandindianocean/egypt/8288163/How-Egypt-shut-down-the-internet.html>. [Last accessed: July 2014].
- [11] Anne Barnard and Robert Mackey. Internet shutdown reported across syria. <http://thelede.blogs.nytimes.com/2012/11/29/internet-outage-reported-across-syria/>. [Last accessed: July 2014].

- [12] Jim Dwyer. Decentralizing the internet so big brother can't find you. <http://www.nytimes.com/2011/02/16/nyregion/16about.html>. [Last accessed: July 2014].
- [13] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Proceedings of Eurocrypt 2005*, LNCS. Springer-Verlag, 2005.
- [14] Kim Cameron. The laws of identity. <http://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf>. [Last accessed: July 2014].
- [15] David Chappell. Introducing windows cardspace. <http://msdn.microsoft.com/en-us/library/aa480189.aspx>. [Last accessed: July 2014].
- [16] Openid. <http://en.wikipedia.org/wiki/OpenID>. [Last accessed: July 2014].
- [17] Abhilasha Bhargav-Spantzel, Anna Cinzia Squicciarini, Rui Xue, and Elisa Bertino. Multifactor identity verification using aggregated proof of knowledge. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(4):372–383, July 2010.
- [18] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proceedings of Advances in Cryptology – EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer-Verlag, 2001.
- [19] Ian Clarke, Oskar Sandberg, Matthew Toseland, and Vilhelm Verendel. Private communication through a network of trusted connections: The dark freenet. <http://freenetproject.org/papers/freenet-0.7.5-paper.pdf>. [Last accessed: July 2014].
- [20] Gnunet. <https://gnunet.org/>. [Last accessed: July 2014].
- [21] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42:39–41, 1999.
- [22] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *In Proceedings of the 13th Usenix Security Symposium*, 2004.
- [23] David Chaum and Eugène Van Heyst. Group signatures. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'91, pages 257–265, Berlin, Heidelberg, 1991. Springer-Verlag.
- [24] Amos Fiat and Moni Naor. Broadcast encryption. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 480–491, New York, NY, USA, 1994. Springer-Verlag.
- [25] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, CRYPTO'05, pages 258–275, Berlin, Heidelberg, 2005. Springer-Verlag.

- [26] Daniele Micciancio and Saurabh Panjwani. Corrupting one vs. corrupting many: The case of broadcast and multicast encryption. In Ingo Wegener, Vladimiro Sassone, and Bart Preneel, editors, *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming – ICALP 2006*, volume 4052 of *Lecture Notes in Computer Science*, pages 70–82, Venice, Italy, 2006. Springer-Verlag.
- [27] Yi Lu, Weichao Wang, B. Bhargava, and Dongyan Xu. Trust-based privacy preservation for peer-to-peer data sharing. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 36(3):498–502, May 2006.
- [28] Internet relay chat protocol. <https://tools.ietf.org/html/rfc1459>. [Last accessed: July 2014].
- [29] Skype. <http://skype.com>. [Last accessed: July 2014].
- [30] Diaspora project. <https://joindiaspora.com/>. [Last accessed: July 2014].
- [31] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag.
- [32] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag.
- [33] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '02, pages 466–481, London, UK, UK, 2002. Springer-Verlag.
- [34] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, UK, 1992. Springer-Verlag.
- [35] The java pairing based cryptography library (JPBC). <http://gas.dia.unisa.it/projects/jpbc/>. [Last accessed: July 2014].
- [36] Ruchith Fernando. Project anon-encrypt at Google code. <http://code.google.com/p/anon-encrypt/>. [Last accessed: July 2014].
- [37] Javascript object notation (JSON). <http://www.json.org/>. [Last accessed: July 2014].
- [38] Apache derby. <http://db.apache.org/derby/>. [Last accessed: July 2014].
- [39] Avik Sarkar. 6.5 million of linkedin passwords stolen by cyber criminals. <http://www.voiceofgreyhat.com/2012/06/65-million-of-linkedin-passwords-stolen.html>. [Last accessed: July 2014].
- [40] Phishing. <http://en.wikipedia.org/wiki/Phishing>. [Last accessed: April 2014].
- [41] Microsoft Corporation. Microsoft's vision for an identity metasystem. <http://msdn.microsoft.com/en-us/library/ms996422.aspx>. [Last accessed: July 2014].

- [42] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Definition Language (WSDL). Technical report, W3C, March 2001.
- [43] Henrik F. Nielsen, Noah Mendelsohn, Jean J. Moreau, Martin Gudgin, and Marc Hadley. SOAP version 1.2 part 1: Messaging framework. Technical report, W3C, June 2003.
- [44] Kelvin Lawrence and Chris Kaler. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). Technical report, February 2006.
- [45] WS-Trust 1.3. Technical report, OASIS, March 2007.
- [46] Web Services Addressing (WS-Addressing). Technical report, W3C, August 2004.
- [47] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, March 2005.
- [48] Kevin Steuer, Jr., Ruchith Fernando, and Elisa Bertino. Privacy preserving identity attribute verification in windows cardspace. In *Proceedings of the 6th ACM workshop on Digital Identity Management*, DIM '10, pages 13–16, New York, NY, USA, 2010. ACM.
- [49] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *Theory of Cryptography*, pages 356–374. Springer-Verlag, 2008.
- [50] Gergely Alpár and Jaap-Henk Hoepman. A secure channel for attribute-based credentials. In *Proceedings of the 2013 ACM Workshop on Digital Identity Management*, DIM '13, pages 13–18, New York, NY, USA, 2013. ACM.
- [51] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium*, 2014.
- [52] Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 345–356, New York, NY, USA, 2008. ACM.
- [53] Bharat Bhargava, Noopur Singh, and Asher Sinclair. Privacy in cloud computing through identity management. In *International Conference on Advances in Computing and Communication*, 2011.
- [54] Pelin Angin, Bharat Bhargava, Rohit Ranchal, Noopur Singh, Mark Linderman, Lotfi Ben Othmane, and Leszek Lilien. An entity-centric approach for privacy and identity management in cloud computing. In *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*, pages 177–183, 31 2010–nov. 3 2010.
- [55] Lotfi Ben Othmane and Leszek Lilien. Protecting privacy of sensitive data dissemination using active bundles. In *Proceedings of the 2009 World Congress on Privacy, Security, Trust and the Management of e-Business*, CONGRESS '09, pages 202–213, Washington, DC, USA, 2009. IEEE Computer Society.

- [56] Rohit Ranchal, Bharat Bhargava, Lotfi Ben Othmane, Leszek Lilien, Anya Kim, Myong Kang, and Mark Linderman. Protection of identity information in cloud computing without trusted third party. In *Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems*, SRDS '10, pages 368–372, Washington, DC, USA, 2010. IEEE Computer Society.
- [57] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W. Freeh. Taming information-stealing smartphone applications (on android). In *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, TRUST'11, pages 93–107, Berlin, Heidelberg, 2011. Springer-Verlag.
- [58] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.
- [59] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 321–334, May 2007.
- [60] Ruchith Fernando. Project anon-encrypt at Google code. <http://code.google.com/p/anon-idm/>. [Last accessed: July 2014].
- [61] Linda T. Kohn, Janet M. Corrigan, and Institute of Medicine Molla S. Donaldson, Editors; Committee on Quality of Health Care in America. *To Err Is Human: Building a Safer Health System*. The National Academies Press, 2000.
- [62] "Dutchhealthcare". The rise and fall of the national EHR initiative in the Netherlands. <http://dutchhealthcare.wordpress.com/2011/04/05/the-rise-and-fall-of-the-national-ehr-initiative-in-the-netherlands/>, May 2011. [Last accessed: July 2014].
- [63] The Economist Intelligence Unit. Denmark: Electronic patient records. <http://www.reforminghealthcare.eu/economist-report/some-roads-ahead-innovative-approaches-in-five-west-european-countries/denmark-electronic-patient-records>. [Last accessed: April 2014].
- [64] Williams Claudia, Farzad Mostashari, Kory Mertz, Emily Hogin, and Parmeeth Atwal. From the office of the national coordinator: the strategy for advancing the exchange of health information. *Health affairs*, 31(3):527–536, 2012.
- [65] theDataMap. <http://thedatamap.org>. [Last accessed: April 2014].
- [66] James J. Cimino, Mark E. Frisse, John Halamka, Latanya Sweeney, and William Yasnoff. Consumer-mediated health information exchanges: The 2012 ACMI debate. *Journal of Biomedical Informatics*, February 2014.
- [67] Google Health. http://www.google.com/intl/en_us/health/about. [Last accessed: April 2014].
- [68] Microsoft HealthVault. <https://www.healthvault.com/us/en>. [Last accessed: April 2014].

- [69] Joseph Barjis. Dutch electronic medical record – Complexity perspective. In *Proceedings of the 43rd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, Jan 2010.
- [70] Perry Groot, Ferry Bruijsten, and Martijn Oostdijk. Patient data confidentiality issues of the dutch electronic health care record. In *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC)*, pages 151–157, 2007.
- [71] Raed M. Salih, Leszek Lilien, and Lotfi Ben Othmane. Protecting patients’ electronic health records using enhanced active bundles. In *Proceedings of the 6th International Conference on Pervasive Computing Technologies for Healthcare, Doctoral Consortium*, pages 1–4, 2012.
- [72] Lotfi Ben Othmane. *Active bundles for protecting confidentiality of sensitive data throughout their lifecycle*. PhD thesis, Kalamazoo, MI, USA, 2010.
- [73] MyDataCan. <http://mydatacan.org>. [Last accessed: April 2014].
- [74] William A. Yasnoff, Latanya Sweeney, and Edward H. Shortliffe. Putting health IT on the path to success. *JAMA*, 309(10):989–990, 2013.
- [75] Health Record Banking Alliance. <http://www.healthbanking.org>. [Last accessed: April 2014].
- [76] State of Washington Health Care Authority and Health Information Infrastructure Advisory Board. Washington state health information infrastructure: final report and roadmap for state action. Technical report, 2006.

VITA

VITA

Ruchith Udayanga Fernando received a B.Sc. (Hons.) in computer science and engineering from the University of Moratuwa, Sri Lanka in 2004. He worked as a Software Engineer in Sri Lanka and entered Purdue University in the fall of 2008. He obtained an M.S. in computer science in May 2012 and a Ph.D. in computer science in August 2014. In addition to his primary research work, Ruchith worked as a software developer with the Rosen Center for Advanced Computing where he contributed to multiple healthcare related projects. Ruchith was a member of the Purdue University Cricket team in the summer of 2012. His other interests include photography and travel.