Department of Computer Security, **b**-it

Media Informatics

Master's Thesis

Certificateless Encryption Scheme Using Biometric Identity

Author:
Kumar Sharad
Matr. Nr. - 298748

Supervisors:
Prof. Dr. Joachim von zur Gathen
Dr. Michael Nüsken

Bonn, March 13, 2012

Acknowledgement

This master's thesis is the result of the help and cooperation that I received from my professors, colleagues, friends and family members. I would like to take this opportunity to express my gratitude to the people who have helped me in achieving my goals. This master's thesis has been written under the supervision of Prof. Dr. Joachim von zur Gathen and the guidance of Dr. Michael Nüsken, without their contributions this work could not have been completed.

Of the various people who contributed to my work, Dr. Michael Nüsken was specially helpful and patient in answering my questions and suggesting improvements. He was always supportive of my work and provided valuable feedback. His suggestions and ideas have helped in shaping this thesis.

I was first introduced to the field of cryptography when I took the course on basic cryptography offered by Prof. Dr. Joachim von zur Gathen in my first semester. Ever since I have been fascinated by this area and Prof. Dr. Joachim von zur Gathen has had a huge influence in driving my interests. It is due to him that I took subsequent coursework in the area of security and cryptography and this has finally culminated as my master's thesis. I am grateful to him for the inspiration, motivation and support he has given me.

Over the period of this thesis I have learned many things and I am very thankful to the people of *Department of Computer Security* at **b**-it, Bonn for this. They were always helpful to me whenever I sought their counsel and contributed in useful discussions. Finally, I would like to thank all my friends and family members for their valuable support and understanding.

Abstract

Often encryption is used to provide data privacy but alone it is not enough to ensure it. Generally the key which is used to decrypt an encrypted message is stored in the device along with the message itself and in the event of device compromise, privacy of data cannot be guaranteed. This thesis proposes a scheme which solves this problem by providing two-factor security for mobile devices. The proposed scheme generates key-pairs for identity based encryption using biometric information of the user which is semi-private. The scheme is *certificateless* and free from *key escrow*.

The work is mainly based on the ideas developed in the paper *Certificateless Encryption Schemes Strongly Secure in the Standard Model* by Dent, Libert & Paterson (2008). In this paper the authors proposed a construction based on *bilinear groups* to generate key-pairs using user's identity. We have modified this construction to include *biometrics* of the user. By doing so we further fortify the security of the scheme and this provides protection in the event of device compromise. The generation of key-pair is based on chosen secrets as well as the *biometric identity* of the user.

Decryption of a message requires the private key to be generated by combining pre-computed secrets stored in the device along with the biometric data which must be provided by the user every time he wishes to decrypt a message. The private key is deleted after decrypting the message and is never stored in the device. So when an adversary gains possession of the device with the encrypted messages in it, the device does not contain the entire private key to decrypt the messages but only a part of it. This part would have to be combined with the correct biometrics to which only the rightful owner has *easy* access. This property of the scheme provides two-factor security where the user needs to not only possess the device but also have the required biometric data. In the proposed attack model we treat biometric data as semi-private which means that in practice it is not readily available to the people who do not know the user, hence even if the device is compromised its messages cannot be trivially decrypted.

Contents

1. Introduction

1.1. Problem Definition. Traditionally in encryption schemes the encrypted message and the key needed to decrypt it are stored in the same device and the compromise of the device makes decryption of the message imminent by the attacker thus destroying all privacy. With mobile devices being increasingly used by a large portion of the population for communication, the possibility of device theft or loss is even higher. Such threats need to be accounted for and we need encryption schemes which provide security even after the device is compromised. One way to safeguard against these threats is to use *two-factor* authentication. A *two-factor* authentication mechanism provides security even after the first line of defence collapses. In case of mobile devices this would be equivalent to the adversary gaining control of the device. A *two-factor* authentication procedure would ensure that the adversary can not decrypt the messages stored in the device by using just the contents of the device. To form a second line of defence we propose the use of biometric data in the encryption scheme. The encryption scheme would use the biometrics of the user coupled with secrets stored in the device to encrypt and decrypt messages. Usage of biometric data has several advantages the most significant ones being its easy availability only to the legitimate user and inherent difficulty in replicating it.

Although biometric information is not entirely private we may consider it as *semi-private* since it is not readily accessible to an attacker and in normal circumstances it should be hard for the attacker to acquire the biometrics of an unknown user. Hence in order to decrypt a message an attacker would have to compromise the device as well as replicate the biometric data of the user. These two factors combined together pose a significantly harder challenge to the attacker as compared to before where he just had to compromise the device.

1.2. Goals. The issues pointed out in the problem definition require reinvestigation of security goals and threat models. The problem of privacy after device compromise has not been treated in detail by existing schemes and merely ensuring the privacy of data under ideal circumstances is far from adequate. To design an encryption scheme that refutes such attacks requires careful consideration of a number of aspects . In this section we describe the primary and auxiliary goals that must be achieved by such a scheme. These goals are discussed in further detail in the subsequent sections and here we just introduce the ideas which are treated in the text to follow. Below we formalise the primary and auxiliary goals that such an encryption scheme should attain.

- The scheme must provide protection against device compromise.
- The scheme must use secrets stored in the device coupled with biometrics to encrypt and decrypt messages, the advantages that we gain from this have been detailed in Section 8.
- The scheme must not be resource extensive since it is targeted for mobile devices.

- The scheme must be certificateless thus simplifying the Public Key Infrastructure (PKI) deployment and management. This would also reduce the computational costs endured by the mobile devices by eliminating the need to validate certificates.

These features provide us a starting point and we may now begin to conceptualise the design of an encryption scheme which supports these goals.

1.3. Approach Taken. To achieve the goals described in the previous section we use the ideas introduced in *Certificateless Encryption Schemes Strongly Secure in the Standard Model* by Dent, Libert & Paterson (2008). This scheme serves as a basis on which we build our construction, we borrow the presented ideas and modify them to design a scheme that supports our goals. The construction presented by Dent, Libert & Paterson possess properties that are very well aligned to the targets that we wish achieve and we highlight them below.

- *Identity Based* - Since we intend to use biometric identity to encrypt and decrypt messages it is only natural to use an Identity-Based Encryption (IBE) scheme. Also IBE schemes are certificateless and this meets another very key requirement hence we chose the construction mentioned above as it fits our needs well.
- *No Key Escrow* - IBE schemes suffer from the drawback of key escrow, however the construction presented by Dent, Libert & Paterson is free from this problem. This decreases the trust requirement that a user must put in a third party and makes it harder for the rogue trusted parties to misuse power.
- *Certificateless* - Even after being free from key escrow the scheme retains the important properties of IBE schemes and is certificateless.
- *Minimal Trust Requirements* - Normally IBE schemes require a high level of trust to be placed in a central authority which possesses the private keys of users and hence has the ability to read private communication. The construction of Dent, Libert & Paterson is free from such central control and the trust requirements are comparable to those one normally puts in a Certification Authority (CA), we discuss this point in more detail in Section 9.
- *Lightweight* - The scheme is lightweight which is essential since we wish to design the scheme for mobile devices. Such devices have modest computation power and memory and this should be taken into account while designing the construction.
- *Security* - The scheme is secure in the standard model which is the best possible security one can achieve.

Due to these factors we have chosen the construction presented by Dent, Libert & Paterson.

Historic Background Cryptographic schemes have evolved continuously to meet the needs of the current time, it all started with *symmetric-key cryptography* which was followed by *asymmetric-key cryptography*. These schemes although effective, were plagued by problems of proliferation of keys and infrastructure management. To resolve these issues the concept of *identity-based cryptography* was introduced. Later the idea of *self-certified keys* was explored and finally this evolution led to the development of Certificateless Public Key Cryptography (CL-PKC). The idea of Certificateless Public Key Encryption (CL-PKE) evolved as a result of the work done in the areas of IBE and public key cryptography. Each of the mentioned approaches tried to solve the problems posed by the previous generation of schemes. We discuss the motivations, history and evolution of these schemes in more detail in Section 2.

New Ideas Our work is mainly based on IBE and CL-PKE, in this thesis we present a practical solution to counter device compromise by using biometrics of the legitimate user as a second line of defence. With mobile devices becoming increasingly powerful and affordable we might soon be able to use biometrics seamlessly and hence our work is aimed at the future. We have constructed the scheme to work on standard hardware and no special set-up is required. The biometric data of the user never leaves the device and is neither stored, it is only used while encryption and decryption. This ensures that no one can harvest user data and launch attacks using that information. The introduction of biometric data in the original scheme has very little impact on the computational costs endured by any of the parties involved. These changes have almost no perceivable increase in the encryption and decryption times.

1.4. Results. We have derived a scheme using the construction proposed by Dent, Libert & Paterson (2008), our scheme attains all the goals that we defined in Section 1.2. We show that our construction is secure in the standard model and retains all the important features introduced by Dent, Libert & Paterson. To demonstrate our scheme we have developed an *Android* application which provides a proof of concept of the derived scheme. For sake of completeness we have also developed an *Android* application implementing the original scheme as proposed by Dent, Libert & Paterson.

The prototypes developed by us are a proof that the suggested changes to the original scheme are practical and achievable, it also throws light on the efficiency and convenience of using such schemes. This information is vital for further developments and improvements to the scheme.

1.5. Organisation. The rest of the paper is organized as follows. First we look at the evolution of public key cryptography and motivate the problem at hand in Section 2. Then we take a detailed look at the work already done in the area of certificateless encryption in Section 3. The concepts and definitions that have been used in this paper to construct the certificatelss encryption schemes are defined in

Section 4. We present an overview of an existing certificateless encryption scheme in Section 5 and in Section 6 propose modifications to it to achieve two-factor security. Then we look at various applications and advantages of certificateless encryption in Section 7 and Section 8 respectively. The security model of the original and the modified scheme is presented in Section 9 where we define oracles, adversaries and attackers. We present the concrete original construction and its security reduction in Section 10 this is followed by the concrete modified construction and its security reduction in Section 11. We discuss the implementation details in Section 12 and finally the paper ends with concluding remarks in Section 13.

2. Evolution of Cryptographic Schemes

Several cryptographic schemes exist today which have evolved over the years to meet the privacy goals which encompassed the challenges faced by the people at the time. As technology has evolved not only have the data consumption habits of people changed but it has also had a profound effect on the way we communicate. If we look at these schemes in detail we will discover that one goal is common to them all which is balancing convenience with security. In this section we look at how new cryptographic schemes have evolved successively. There has been a concerted effort to make data security more and more convenient and each new scheme has tried to achieve this, at times perhaps at the cost of privacy.

2.1. Symmetric-Key Cryptography. This is one of the earliest form of encryption schemes. The onset of *Symmetric-Key Cryptography* allowed parties to partake in private communication, users could now agree on keys which would allow them to communicate privately by encrypting messages using that key. But it had drawbacks like proliferation of keys since each pair of users needed a unique key to communicate privately this lead to every user having a separate key for all his communication channels. Also key exchange was not simple and users had to run key agreement protocols prior to sending messages to each other. These issues made key management and off-line communication difficult.

2.2. Public Key Cryptography. To solve the drawbacks of *Symmetric-Key Cryptography* the idea of *Asymmetric-Key Cryptography* or *Public Key Cryptography* came into existence. It solved the problem of key management by assigning two keys to each user, a *private key* and a *public key*, the user can freely distribute his *public key* while keeping the *private key* only to himself. This allowed him to receive messages encrypted with his *public key* which could be decrypted using his *private key*, this also solved the problem of key agreement and made off-line communication possible. However, this scheme presented new challenges to contend with. The authenticity of the users became an issue and certificates issued by a CA were used to verify that a certain key-pair and user identity were linked. During setting up a PKI one of the most challenging aspects is handling trust management, the conventional solution to this problem is to use certificates. Certificates are issued by trusted central authorities and cryptographically hard to forge but they are not easy to set-up and pose operational difficulties, these issues have been illustrated in further detail in Adams & Lloyd (2002). Deploying PKIs is a cumbersome task and many considerations need to be taken in order to make things work, there is no universal solution to the problem and normally one needs to take the *horses for courses* approach while setting them up. In Gutmann (2002) the authors examine various reasons for the limited success of PKIs and make suggestions to deploy them successfully. They also discuss the application specific approach which one needs to take to avoid problems, some of the most serious issues being certificate revocation, handling authorisation and audit, managing certificate chains, storage

and distribution. The computational cost of certificate verification is also an important point of contention, specially in the case of mobile devices whose usage is on the rise. These challenges have also been highlighted in Dankers, Garefalakis, Schaffelhofer & Wright (2002).

2.3. Identity-Based Cryptosystems. Due to the factors discussed we see that improper deployment and management of certificate PKIs can potentially compromise the security of a system hence we look further to simplify certificate management. In 1984 the notion *Identity-Based Cryptosystems and Signature Scheme* was proposed by Shamir (1984), it suggested the usage of user's unique identity like e-mail address, social security number or IP address to derive his public key which can be used to send him encrypted messages. This enabled parties to communicate securely without them requiring to exchange public or private keys, maintaining key directories or using services of a third party. The use of a trusted Private Key Generator (PKG) was suggested to provide the user with a smart card on joining the network, this card was tightly tied to the identity of the user and contained keys which would allow him to sign and encrypt the messages he sent and verify and decrypt the messages he received. The scheme had various advantages such as simplifying certificate management which is one of the hardest parts of setting up PKIs, now in order to send encrypted messages the users only required to know the identity of the party to which they intended to send the message and looking up information in the certificates was no longer necessary. Due to the involvement of a trusted PKG in the process the scheme had the drawback of *key escrow*, since the PKG was in possession of a master key which was used to generate private keys of the users in the system. A rouge PKG could destroy all privacy and consequently the system had a single point of failure whose compromise would have devastating effects on the users. Also in today's world it is impractical to assume the existence of such key generation centres due to nature of communication being global and a typical user possesses multiple identities which makes keeping a card for each identity unmanageable.

Shamir gave concrete construction only for a signature scheme but not for an encryption scheme. Many approaches were made in the subsequent years to present an IBE scheme like Desmedt & Quisquater (1986), Tsujii & Itoh (1989), Tanaka (1987), Maurer & Yacobi (1991) and Hühnlein, Jr. & Weber (2000). None of the solutions fully solved the problem and suffered from issues like collusion of users, long turnaround time for private key generation by PKG and requirement of tamper-resistant hardware. Finally, the first practical solution was presented in 2001 in Boneh & Franklin (2001) where the authors proposed an *Identity-Based Encryption from the Weil Pairing*. After this paper there was a surge in the area of IBE and this resulted in the development of many cryptographic primitives. An interactive identity-based key exchange protocol was presented by Sakai, Ohgishi & Kasahara (2000) followed by a non-interactive version by Smart (2002). Several signature schemes were also developed e.g. Cha & Cheon (2003), Hess (2003) and Paterson (2002b), a hierarchical identity based scheme was presented by Gen-

try & Silverberg (2002). Work was also carried out in the area of cryptographic work flows by Chen, Harrison, Moss, Soldera & Smart (2002), Paterson (2002a) and Smart (2003) and identity based cryptography was used as a mechanism to demonstrate this.

2.4. Fuzzy Identity-Based Encryption. With IBE schemes becoming popular attempts were made to use the biometric data of the user as identity and the first *Fuzzy Identity-Based Encryption* scheme was presented by Sahai & Waters (2005). It allowed for the use biometric identities by incorporating an error-tolerance property which correctly decrypted an encrypted message when the identity presented to decrypt the message was close to the identity used to encrypt the message. This was necessary for the scheme to work since biometric identity cannot be same every time it is sampled and hence the scheme needs to allow for some noise. Subsequently more efficient Fuzzy IBE schemes were presented by Baek, Susilo & Zhou (2007) and Furukawa, Attrapadung, Sakai & Hanaoka (2008). But due to *key escrow* being an inherent property of IBE the PKG could decrypt any cipher text, this also allowed the PKG to forge any user's signature and hence non-repudiation was not guaranteed by design. Use of multiple PKGs in an IBE scheme has been considered to avoid concentration of power but this requires more effort to manage communication and infrastructure. Even if the PKG is fully honest the compromise of the PKG's master key would have devastating consequences which would be more far reaching as compared to the breach of the CA's signing key in traditional PKI. Hence deployment of IBE schemes on a large scale is not suitable, for these reasons people continued to look for constructions which simplified certificate management without handing over too much power to the PKG.

2.5. Self-Certified Keys. Meanwhile the idea of *Self-Certified Keys* was introduced by Girault (1991) and later enhanced by Petersen, Horster & Horster (1997) and Saeednia (1997). A self-certified scheme also relies on the existence of a Trusted Third Party (TTP), here the users generate their own *private key* (sk) and corresponding *public key* (pk) and communicate pk to the TTP who creates a *witness* (w) by combining the identity (ID) of the user with pk. Several ways have been suggested to produce this witness, Girault used TTP's signature on some combination of pk and ID, Petersen, Horster & Horster used part of a signature and Saeednia used the result of inverting a trapdoor one-way function derived from pk and ID. This scheme allowed any party to extract pk from w and ID while only making it possible for the TTP to produce w from pk and ID. Although this scheme does not make use of certificates in the traditional sense it can be observed that the witness w is a type of lightweight certificate which binds the identity of the user to the correct public key. The scheme has an advantage as compared to the CL-PKC, it does not require any confidential communication between the TTP and the user. However, the private key needs to be generated before the public key, due to this the scheme cannot be used to enforce *cryptographic work flows* as described in Section 7. These schemes also lack concrete security proofs as pointed

by Saeednia (2003) and suffer from drawbacks which allow a rouge TTP to extract a user's private key.

2.6. Ceritificateless Public Key Cryptography. The idea of *Ceritificateless Public Key Cryptography* emerged from fact that people wanted to avoid the need for setting up infrastructure to support trust management using certificates. As we saw earlier IBE schemes did solve this issue however it was not without introducing the problem of *key escrow*, people now started working towards eliminating it without sacrificing the desirable properties of the IBE schemes. In a typical IBE scheme the private key of a user is entirely generated by the PKG and this is what makes the privacy of the system totally dependent upon the PKG. However, now it was suggested that perhaps only a part of the secret be generated by the PKG while the user holds on to the other part. This would eliminate the possibility of the PKG misusing his powers, additionally the scheme is kept certificateless while also defeating the attempts of a dishonest party to impersonate an user. A CL-PKC scheme is similar to the Identity Based Cryptography (IBC) scheme in the respect that it relies on the existence of a trusted third party which possesses a master key, the scheme also uses the identity of the user. These ideas were formally developed by Al-Riyami & Paterson (2003) and were derived from the scheme presented by Boneh & Franklin (2001) by making simple modifications. The authors suggested an intermediate between *Public Key Cryptography* and *Identity Based Cryptography* as *Certificateless Public Key Cryptography* and it eliminated the *key escrow* associated with the IBE schemes without the need of certificates. In principal there are three parties involved in a CL-PKC scheme, the trusted third party called Key Generation Center (KGC), the party sending the message called *Sender* and the party who receives the sent message called *Receiver*, we describe these terms formally in Section 4.2. The KGC uses his *master private key* along with the receiver's identity to generate a *partial private key* which the receiver then combines with a secret value to derive his full private key. Thus this key is known only to the receiver and key escrow is avoided. The receiver needs to authenticate his identity to the KGC who must then securely transmit the partial private key. Meanwhile, the receiver also computes his *public key* by combining the same secret value with the public parameters published by the KGC and distributes it freely. The generation of private key and public key is independent of each other and just requires the use of the same secret value. The sender can thus obtain the public key related to a certain identity and use it to send encrypted messages to the receiver.

3. Related Work

Let us take a brief look at the work done in the area of certificateless cryptography before the notion of CL-PKC was formalized. The idea of Certificate-Based Encryption (CBE) was introduced by Gentry (2003), it proposed a construction in which the user was required to use his secret key along with an up-to-date certificate to decrypt the message. The scheme tried to combine the implicit certification of IBE schemes with the no key escrow property of public key cryptography. Subsequently, an equivalence theorem between IBE, CBE and Certificateless Encryption (CLE) was presented by Yum & Lee (2004a,b). The generic transformations presented in this paper did not use random oracles but their results did not hold in the full security model developed by Al-Riyami & Paterson (2003) and were also shown to breakdown in the much weaker security model presented by Galindo, Morillo & Ràfols (2006).

Later Dodis & Katz (2005) formalized the problem of chosen-ciphertext security for multiple encryption and presented simple, generic and efficient constructions of multiple encryption schemes secure against chosen-ciphertext attacks in the standard model. They also proved that their methods can be applied to design CBE schemes without random oracles. However, their design did not hold in the security model presented by Al-Riyami & Paterson (2003) as their constructions were not designed to withstand the decryption queries for arbitrary public keys chosen adaptively by adversaries without the knowledge of matching secret.

More recently Liu, Au & Susilo (2006) highlighted the issue of *Denial-of-Decryption Attack* where the adversary replaces the user's public key by someone else's as a result when the user gets a message encrypted with that key and his identity, he is no longer able to decrypt it and the sender of the message is unaware of this. They propose a new paradigm called *Self-Generated-Certificate Public Key Cryptography* which addresses this problem and provide a generic construction using certificateless signature and certificateless encryption as the building block. Their construction is secure in the standard model but it does not hold in the full model presented by Al-Riyami & Paterson (2003). Huang & Wong (2007) presented a construction which is secure against the malicious-but-passive PKG attacker in the standard model but it does not allow a *Strong Type I Attacker* described in Section 9.2.2.

As discussed before the concept of CL-PKC was first introduced by Al-Riyami & Paterson they presented a scheme which was structurally similar and borrowed ideas from self-certified keys presented by Petersen, Horster & Horster (1997), Girault (1991), Saeednia (1997) and more recently CBE scheme proposed by Gentry (2003). In their work the authors specified certificateless encryption, signature and key exchange schemes and demonstrated how to support certificateless hierarchical schemes. Their construction was based on bilinear map on groups as described in Definition 4.1 and the security was reducible to the computational hardness of the Bilinear Diffie-Hellman Problem. Later in 2008 the first concrete and efficient construction for CLE secure in the standard model against strong adversaries was

presented by Dent, Libert & Paterson (2008). This scheme is secure against both *Strong Type I Attacker* and *Strong Type II Attacker* as described in Section 9.2.2. The construction is modelled upon the Waters' IBE scheme presented by Waters (2005) and modified using ideas from Al-Riyami & Paterson (2003) the security of this scheme is based on the hardness of the The Decisional 3-Party Diffie-Hellman Problem (3-DDH) defined in Definition 4.3 which is a slight and natural generalisation of the The Decisional Bilinear Diffie-Hellman Problem (DBDH) described in Definition 4.2. However, a new kind of threat was considered in Au, Mu, Chen, Wong, Liu & Yang (2007) where the adversaries maliciously generate system-wide parameters, the construction being presented here is not secure under this attack.

3.1. Proposed Modification. We propose a modification to the original construction to achieve two-factor authentication by inclusion of the user biometrics in the scheme. The derived scheme retains all the properties of the original scheme. In the modified version the receiver authenticates himself to the KGC by providing his public identity such as an email or an IP address and the hash of his biometric identity. The KGC then responds back in a secure manner transmitting the *partial private key* to the receiver which he generates combining a secret value with the receiver's public identity and biometric identity. The receiver then proceeds to generate his public key dependent on the partial private key and public parameters published by the KGC. Subsequently, this key is freely and widely published. Since we desire two-factor authentication it must be noted that the full private key is never stored in the device and is generated only while decrypting a cipher text and duly deleted after the decryption. The full private key is derived from the partial private key, the biometric identity and a secret value provided by the receiver. The sender can now obtain the public key related to a certain identity and use it to send encrypted messages to the receiver. These changes make it imperative to possess the correct biometric data to encrypt messages. In addition to the biometrics the device with stored secrets is also required by the receiver to be able to decrypt messages and this provides us a system with two-factor authentication.

4. Preliminaries

In this section we describe the various preliminaries which have been used throughout the thesis.

4.1. Notation. Certain terms are freely used in various places in the work presented. Here we define them formally.

- Adversary: The adversary is defined as the party who is trying to gain information about the ciphertexts by using the oracles as his disposal.
- Challenger: The challenger is defined as the party who presents the adversary with a challenge based on ciphertexts. After receiving the challenge the adversary tries to gain information about the ciphertexts.

4.2. Parties. The CL-PKE scheme consists of three parties which are

- *Sender*: This is the party which intends to send private messages and hence encrypts them using the CL-PKE scheme. In the original scheme the sender encrypts the message using receiver's public identity and *public key*. In the derived version of the scheme the sender encrypts the message using the receiver's public identity, biometric identity and *public key*.
- *Receiver*: This is the party which intends to receive private messages encrypted using the CL-PKE scheme and then decrypts the messages to read them, the receiver is also responsible for publishing his identity along with *public key* so that sender can use them encrypt the message. In the derived version of the scheme it is assumed that the sender possesses the receiver's biometric identity.
- *KGC*: The KGC is the party who is responsible for running the set-up for the CL-PKE scheme. It's duties include computing *master public key*, *master secret key* and authenticating an user. The KGC is also responsible for computing and securely communicating the *partial private key* for each receiver based on his identity. We define the details of the operations performed by the KGC in Section 10 and Section 11.

4.3. Definitions. In this section we formally describe the various definitions and theorems that are used throughout the thesis. The constructions and security reductions use them widely and this section aims to familiarize the reader with the notation used.

DEFINITION 4.1. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of order p for some large prime p. The bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ must satisfy the following properties*

(i) Bilinearity: $e(g^a, h^b) = e(g, h)^{ab}$ for any $(g, h) \in \mathbb{G} \times \mathbb{G}$ and $a, b \in \mathbb{Z}$.

(ii) Non-degenerate: $e(g,h) \neq 1_{\mathbb{G}_T}$ *whenever* $g,h \neq 1_{\mathbb{G}}$.

(iii) Computable: There is an efficient algorithm to compute $e(g,h)$ *for any* $g,h \in \mathbb{G}$.

DEFINITION 4.2. *The Decisional Bilinear Diffie-Hellman Problem is to decide whether* $T = e(g,g)^{abc}$ *or a random element. Given* $g, g^a, g^b, g^c \in \mathbb{G}$, $T \in \mathbb{G}_T$ *and* $a,b,c \in_R \mathbb{Z}$. *A bilinear map is described by* e, *i.e.* $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ *with properties described in Definition 4.1.*

DEFINITION 4.3. *The Decisional 3-Party Diffie-Hellman Problem was first defined in Boneh & Franklin (2001) and its goal is to decide whether* $T = g^{abc}$ *such that* $a,b,c \in_R \mathbb{Z}$ *and* $(g^a, g^b, g^c, T) \in \mathbb{G}^4$ *for* $g \in \mathbb{G}$. *We define the advantage of a probabilistic polynomial-time algorithm* $\mathcal{A}$ *against this problem as*

$$Adv_{\mathcal{A}}^{3\text{-}DDH}(k) = \Big|\Pr\left(\mathcal{A}(g^a,g^b,g^c,T) = 1 \mid T = g^{abc} \wedge a,b,c \in_R \mathbb{Z}_p^{\times}\right) - \Pr\left(\mathcal{A}(g^a,g^b,g^c,T) = 1 \mid T \in_R \mathbb{G} \wedge a,b,c \in_R \mathbb{Z}_p^{\times}\right)\Big|$$

where k *is the security parameter. We assume* $Adv_{\mathcal{A}}^{3\text{-}DDH}(k)$ *to be negligible for all probabilistic polynomial-time algorithms* $\mathcal{A}$.

DEFINITION 4.4. *A hash function* H *randomly selected from a family of hash functions* $H \in_R \mathcal{H}(k)$ *is collision resistant if for all probabilistic polynomial-time algorithms* $\mathcal{A}$ *the advantage*

$$Adv_{\mathcal{A}}^{CR}(k) = \Pr\left(H(x) = H(y) \wedge x \neq y \mid (x,y) \in_R \mathcal{A}(1^k, H) \wedge H \in_R \mathcal{H}(k)\right)$$

is negligible as a function of the security parameter k.

5. Overview of the Original Construction

Our work is based on the construction proposed by Dent, Libert & Paterson (2008), in this paper the authors show a way to construct a practical certificateless encryption scheme which is secure in the standard model and free from key escrow. In this section we describe the constituents of this construction to give a bird's eye view to the reader. The individual components have been described in detail in Section 10.

5.1. Components of the Original CL-PKE. A CL-PKE scheme is defined by seven probabilistic, polynomial-time algorithms and is based on the ideas first proposed by Al-Riyami & Paterson (2003). We now take a look into the details of each of these algorithms. The subsequent algorithms make use of the user identity denoted as ID which is unique and publicly known, examples of such an identity could be email address, IP address or any other form of identification.

1. `Setup`: This algorithm is run by the KGC and takes as input the security parameter defined by 1^k. The algorithm returns the *master public key* mpk and the *master secret key* msk.

2. `Extract`: This algorithm is run by the KGC to extract the *partial private key*. The algorithm takes as input the *master public key* mpk, the *master secret key* msk and identity of the receiver $ID \in \{0,1\}^*$. The algorithm returns the *partial private key* d_{ID} of the receiver with identity $ID \in \{0,1\}^*$.

3. `SetSec`: This algorithm executed by the receiver generates a secret value x_{ID}, it takes as input the *master public key* mpk.

4. `SetPub`: This algorithm is run by the receiver and takes as input the *master public key* mpk and receiver's secret value x_{ID}. The algorithm outputs the *public key* $pk_{ID} \in \mathcal{PK}$ for the receiver.

5. `SetPriv`: This algorithm is run by the receiver to generate his *full private key* sk_{ID}. The algorithm takes as input the *master public key* mpk, the *partial private key* d_{ID}, the public identity ID and the receiver's secret value x_{ID}.

6. `Encrypt`: This algorithm is run by the sender. The algorithm takes as input the *master public key* mpk, the receiver's identity ID, the *public key* pk_{ID} of the receiver with identity ID and the message $m \in \mathcal{M}$. The algorithm returns the cipher text $C \in \mathcal{C}$ if $pk_{ID} \in \mathcal{PK}$ else it returns FAIL.

7. `Decrypt`: This algorithm is run by the receiver and takes as input the *master public key* mpk, receiver's *full private key* sk_{ID} and a cipher text $C \in \mathcal{C}$. It returns the message $m \in \mathcal{M}$ if C is a valid ciphertext else it returns FAIL.

5.2. Protocol Architecture. After discussing the various modules of the encryption scheme we can now look at how the entire machinery functions. We do this by describing the protocol involving the sender, the receiver and the KGC.

1. The scheme begins with the execution of `Setup` algorithm by the KGC which generates the system parameters and keys as described. These parameters and keys are later used by the the subsequent algorithms.

2. The next algorithm executed by the KGC is `Extract` and is initiated by the receiver when he wishes to compute his *private key*. Before running this algorithm the receiver must authenticate himself to the KGC as ID in the same way as he would to a CA. The computed *partial private key* d_{ID} is communicated to the receiver with identity ID in a secure manner. The computation of the receiver's *private key* is independent of the computation of his *public key* and only needs to be done before decrypting a message.

3. The first algorithm executed by the receiver is `SetSec` as described before. The algorithm is run once by the receiver before he can compute his *public* and *private keys*.

4. The next algorithm executed by the receiver is `SetPub` to compute his public key. It is run once by the receiver and the computed *public key* pk_{ID} is published and freely distributed. It is assumed that the public key space $\mathcal{PK}$ is publicly recognisable since it is defined using the *master public key* mpk. Public keys with matching private key should be easily recognisable from the malformed public keys, we show how to achieve this in the concrete construction presented in Section 10.

5. Before the receiver can decrypt the encrypted messages send to him he needs to run the algorithm `SetPriv` to obtain his *private key*. This algorithm is run once once by the receiver after he obtains the *partial private key* from the KGC after authenticating himself as seen in *Step A* and *Step B* in the encryption protocol.

6. To send encrypted messages the sender must run the `Encrypt` algorithm. To do that the sender must first obtain the receiver's *public key* which is freely and widely distributed by the receiver after running the `SetPub` algorithm. The messages are encrypted using receiver's identity ID which is public and his *public key*.

7. To decrypt the encrypted messages the receiver runs the `Decrypt` algorithm. This can be only done after the receiver has calculated his *full private key* by obtaining the *partial private key* from the KGC. We include a hash on the ciphertext because our encryption scheme is homomorphic. The hash acts as a signature on the ciphertext and defeats adversary's attempt to create valid encryptions by combining other encryptions and winning the

Indistinguishability Under Chosen Ciphertext Attack (IND-CCA) game described in Section 9.1.

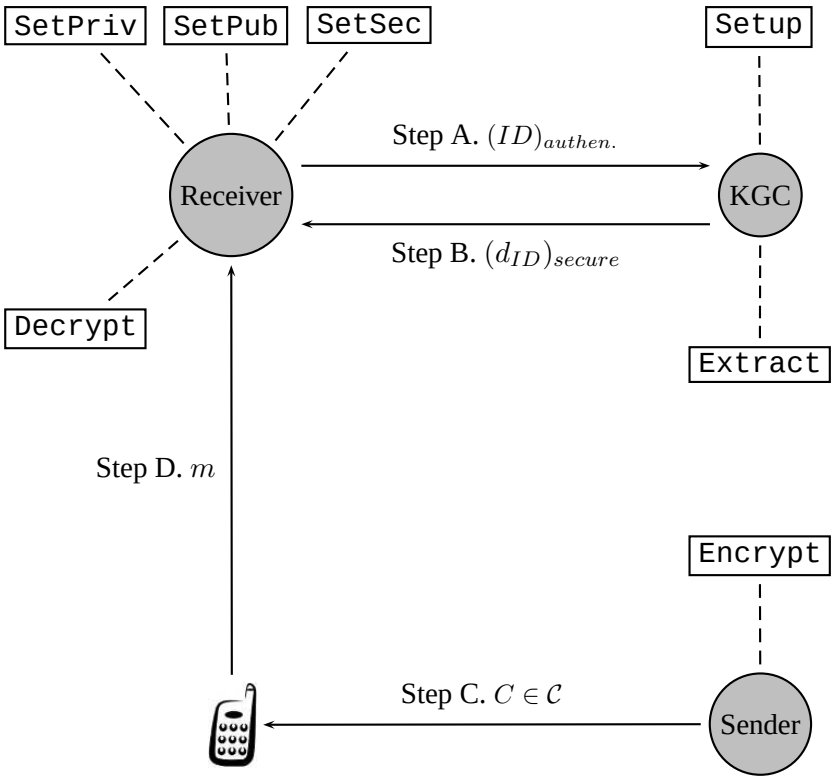

Figure 5.1: After the KGC runs `Setup` the receiver authenticates himself as ID in *Step A*. Subsequently, the KGC runs `Extract` and provides the receiver with d_{ID} in a secure manner as shown in *Step B*. After this the receiver runs `SetSec` and `SetPub`. The computed *public key* pk_{ID} is then published by the receiver. The sender obtains pk_{ID} and encrypts the message m by running `Encrypt`. The computed ciphertext $C \in \mathcal{C}$ is sent to the receiver's device as shown in *Step C*. Finally, the receiver runs `Decrypt` on his device after computing his *private key* using `SetPriv` and obtains the decrypted message m thus concluding the protocol.

6. Overview of the Derived Construction

By modifying the existing construction we aim to provide two-factor security such that even if the device is compromised, the encrypted messaged stored in the device should not be trivially accessible to the attacker. To achieve this we include user biometric data in the scheme. The *full private key* is never stored on the device and is generated before decrypting the message by combining the *partial private key* and a *secret value* stored in the device along with the biometric data of the rightful owner. The biometric data of the owner is never stored on the device and after decrypting the message the *full private key* is deleted. This makes it hard for the attacker to generate the *full private key* again even if he gains control of the device thus granting him access to the *partial private key* and the *secret value* stored in it.

The subsequent algorithms use the public identity of the user denoted as ID which is unique and publicly known, examples of such an identity could be the email address, IP address or any other form of identification. As a second factor in our authentication we use the biometric identity of the user denoted as BID. BID is derived from the biometric characteristics of the user and is therefore unique, some examples of such a biometric identity could be the fingerprint, face picture or voice. We treat the user's biometric data as *semi-private* which means that although it is easily available to the people who know the user still it is significantly harder to reproduce by someone who does not know the user. Hence to compromise the privacy of a user not only should the attacker obtain the device but also produce the owner's biometric data. In practice this is hard to achieve and this is what provides security to the user even in the event of loss or theft of the device.

6.1. Components of the Modified CL-PKE. Our construction is derived from the one presented in the last section and is defined by six probabilistic, polynomial-time algorithms. We now look into the details of each of these algorithms.

1. `Setup`: This is the first algorithm which is executed to set-up the system parameters, it is run by the KGC and takes as input the security parameter defined by 1^k. The algorithm returns the *master public key* mpk and the *master secret key* msk.

2. `Extract`: This algorithm is run by the KGC to extract the *partial private key* d_{ID}. The algorithm takes as input the *master public key* mpk, the *master secret key* msk, public identity of the receiver $ID \in \{0,1\}^*$ and the hash of the receiver's biometric identity BID, $F_h(BID)$.

3. `SetSec`: In this algorithm the receiver generates a secret value x_{ID}. The algorithm takes as input the *master public key* mpk.

4. `SetPub`: This algorithm is run by the receiver and takes as input *master public key* mpk, *partial private key* d_{ID} and the receiver's secret value x_{ID}. The algorithm outputs the *public key* $pk_{ID} \in \mathcal{PK}$ for the receiver.

5. `Encrypt`: This algorithm is run by the sender, the algorithm takes as input the *master public key* mpk, receiver's public identity ID, receiver's biometric identity BID, the *public key* pk_{ID} of the receiver with public identity ID and the message $m \in \mathcal{M}$. The output of this algorithm is the ciphertext $C \in \mathcal{C}$ if $pk_{ID} \in \mathcal{PK}$ else the algorithm returns FAIL.

6. `Decrypt`: This algorithm is run by the receiver and takes as input the *master public key* mpk, receiver's *partial private key* d_{ID}, receiver's secret value x_{ID}, receiver's public identity ID, receiver's biometric identity BID and the ciphertext $C \in \mathcal{C}$. Before decrypting, the *full private key* sk_{ID} is computed using the *master public key* mpk, the *partial private key* d_{ID}, the hash of receiver's public identity ID, $F_u(ID)$ and the hash of receiver's biometric identity BID, $F_h(BID)$. This key is then used to decrypt the message. The algorithm returns the message $m \in \mathcal{M}$ if C is a valid ciphertext else it returns FAIL.

6.2. Protocol Architecture.

1. Just like in the case of original scheme the modified scheme begins with the execution of `Setup` algorithm by the KGC which generates the system parameters and keys as described. These parameters and keys are later used by the the subsequent algorithms.

2. The next algorithm executed by the KGC is `Extract` and is initiated by the receiver when he wishes to compute his *private key*. Before running this algorithm the receiver must authenticate himself to the KGC as ID in the same way as he would to a CA and communicate $F_h(BID)$ securely. The computed *partial private key* d_{ID} is send back to the receiver with identity ID in a secure manner. Unlike the previous scheme in the modified version the receiver must obtain d_{ID} before he can compute his *public key*.

3. The first algorithm executed by the receiver is `SetSec` as described before. The algorithm is run once by the receiver before he can compute his *public* and *private keys*.

4. The next algorithm executed by the receiver is `SetPub` to compute his public key. It is run once by the receiver and the computed *public key* pk_{ID} is published and freely distributed. It is assumed that the public key space $\mathcal{PK}$ is publicly recognisable since it is defined using the *master public key* mpk. Public keys with matching private key should be easily recognisable from the malformed public keys, we show how to achieve this in the concrete construction presented in Section 11.

5. To send encrypted messages the sender must run the `Encrypt` algorithm. To do that the sender must first obtain the receiver's *public key* which is freely and widely distributed by the receiver after running the `SetPub` algorithm.

The messages are encrypted using receiver's personal as well as biometric identity. While personal identity is public and freely available, the biometric identity is *semi-private* and is only accessible to people who know the receiver. Just like in the earlier case, here we assume that the sender knows the receiver and hence possesses his biometric identity or data from which such an identity can be easily extracted, example of such data would be a face picture.

6. To decrypt the encrypted messages the receiver runs the `Decrypt` algorithm. This can be only done after the receiver has calculated his *full private key* after obtaining the *partial private key* from the KGC. This key is then used to decrypt the message and is deleted subsequently. The biometric identity BID is computed from data provided by the user during decryption and is never stored on the device. We include a hash on the ciphertext because our encryption scheme is homomorphic. The hash acts as a signature on ciphertext and defeats the adversary's attempt to create valid encryptions by combining other encryptions and winning the IND-CCA game described in Section 9.1.

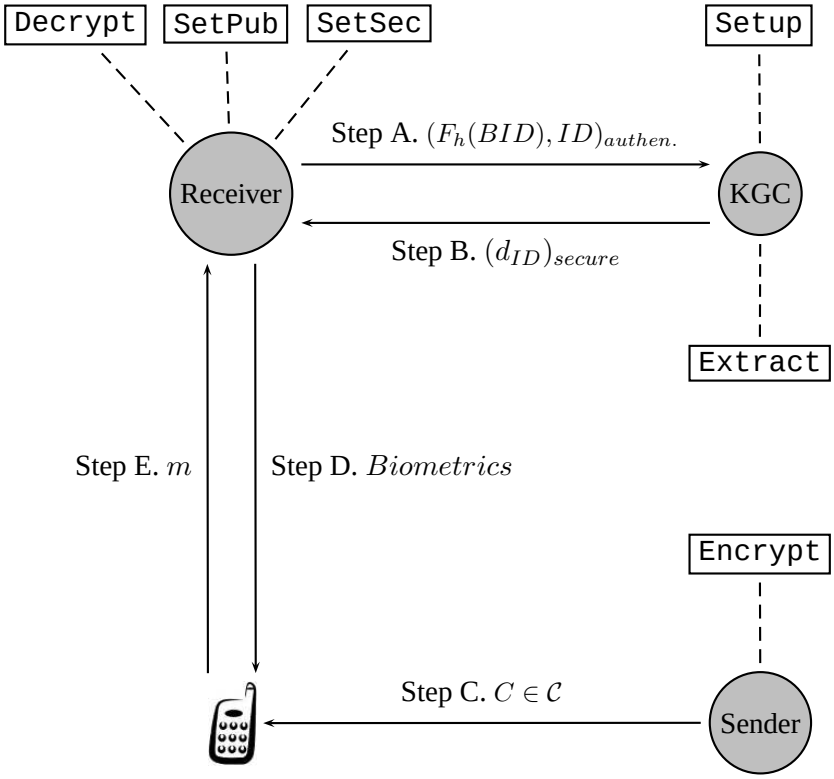

Figure 6.1: After the KGC runs `Setup` the receiver authenticates himself as ID and provides him with the hash of his biometric identity $F_h(BID)$ in *Step A*. Subsequently, the KGC runs `Extract` and provides the receiver with d_{ID} in a secure manner as shown in *Step B*. After this the receiver runs `SetSec` and `SetPub`. The computed *public key* pk_{ID} is then published by the receiver. The sender obtains pk_{ID} and encrypts the message m by running `Encrypt`. The computed ciphertext $C \in \mathcal{C}$ is sent to the receiver's device as shown in *Step C*. Then receiver proceeds on to provide his biometric data to the device as shown in *Step D*. Finally, the private key sk_{ID} is computed by the receiver and the ciphertext C is decrypted using the `Decrypt` algorithm. The receiver then obtains the decrypted message m as shown in *Step E* thus concluding the protocol.

7. Applications of CL-PKE

Due to the identity of the user being tied to the encryption, IBE schemes can be used for applications where identity based role separation is needed. Our scheme is an extension of the IBE scheme and is thus applicable to most situations in which IBE schemes can be employed. The possible applications of IBE schemes have been explored by Boneh & Franklin (2001), we present them below along with some additional applications.

- *Revocation of Public Keys*: The existing PKIs allow the possibility of certificate expiry and the users need to keep their certificates updated for them to be valid. Boneh & Franklin (2001) propose a way to model this in IBE schemes by coupling the identity with an expiration date, we can extend the same idea in CL-PKE schemes by issuing *partial private keys* based on identity coupled with an expiration date, eg. an identity like "`bob@company.com||current-year`" can be used. This will enforce the user to refresh his private key every year by obtaining corresponding *partial private key* and failure to do so will revoke his ability to decrypt ciphertexts. It should also be noted that such a mechanism allows sending messages in the future and Bob can only read those messages when the KGC issues him keys corresponding to the date specified. Thus we see that ephemeral keys can be very easily implemented using CL-PKE.

- *Managing user credentials*: The idea presented above can be extended further and can be employed to manage credentials. An identity like "`bob@company.com||current-year||clearance=secret`" can be used to encrypt messages, here Bob can only read decrypt a message if he has a clearance level of secret for the current year and this can be enforced with the help of a KGC.

- *Cryptographic Work Flows*: CL-PKE schemes have the property that the public key and the private key can be generated independently of each other, this property can be used to enforce *cryptographic work flows*. The sender can encrypt a message using the public key determined by the receiver and an identifier coupled with receiver's identity which the receiver can acquire only after accomplishing some task. Doing so forces the receiver to complete the task to gain access to the said identifier which he can then use to authenticate himself to the KGC and thus obtain the valid partial private key to decrypt the message received. An instance of such a scenario could be that *A* sends a money order to *B* which is encrypted using *A*'s public key and his identity coupled with proof that he sent the goods. Now, after *A* completes his part of the deal he is guaranteed to receive his money by decrypting the money order after obtaining the partial private key from the KGC, more such applications have been explored in Smart (2003) and Paterson (2002a).

The modified version of the scheme does not support this property since the public key and the private key can only be generated after obtaining the partial private key from the KGC. However, one must note that the modified scheme serves an entirely different purpose and hence cannot be expected provide all the features of the original scheme. The construction presented in Section 11 explains this point in further detail.

- *Delegation of Decryption keys*: This kind of application is suited to companies where there is a centralized KGC and he can delegate clearance to people who are allowed to access a message depending upon their role.
  - Use of ephemeral keys: Suppose one of the company's employees is going on a conference and he needs to take a company laptop with him for that period. Normally all the messages are encrypted with the same private key which is stored in the device and its loss cannot be recovered from. To ensure protection from loss of device, messages intended for the employee can be encrypted using his email and current date. The company's KGC can generate partial private keys for the duration of the conference which can be used to extract full private keys by the employee. In the event of loss of the device only the messages received during the conference period are compromised thus leaving the other messages untouched. On the other hand if the proposed modified construction is employed then biometrics would also be used in encryption and none of the messages would be compromised.
  - Delegation of duties: The company based KGC can also enforce credential management, messages could be encrypted based on their subject and the KGC can generate partial private keys to employees within whose domain those messages lie. For instance all the messages meant for *sales* department could be tagged with *sales* and all the employees working in that department could be given one partial private key. This would only give access to the people who possess a particular key to read a certain message, thus enforcing role based privacy.
- *Sharing Facilities*: The modified CL-PKE scheme can be useful in applications where the users need to share facilities yet require privacy. For instance the same mailbox can be shared by multiple users and the messages can be encrypted using email address as the public identity and biometric identity of the intended user. This would allow only the specific user to read the messages even though they are in the same mailbox. One foreseeable application could be forum moderation where messages can be posted publicly but require biometrics to decipher them.
- *Certificate-Based Encryption*: As proposed by Gentry (2003), CL-PKE schemes can be modified to provide *certificate-based encryption* by including the expiry information and public keys in the identity strings.

8. Advantages of the Modified Scheme

IBE schemes have gained popularity due to a number of advantages over the traditional public key cryptography, as discussed before CL-PKC evolved from IBC to further enhance it hence it also provides many benefits that the IBC schemes offer. In this section we explore some major advantages of the discussed CL-PKE schemes over the traditional public key cryptography.

- *Certificateless*: The deployment of PKIs is greatly simplified due to the independence from issuing certificates to manage trust. Issuing certificates is not strictly a technical exercise and involves social factors like proving one's identity and setting up CAs. Such procedures are time consuming and prone to human error as well as social engineering. Complete freedom from such a need hastens the deployment of PKIs and makes trust management entirely technical by removing human intervention and social aspects.

- *No Key Escrow*: IBE schemes were successful in providing certificateless PKIs as well but they relied on a trusted third party which possessed the private key of the users. This makes the scheme centralized by design with a single point of failure, also such power in the hands of the PKG leaves the possibility of surveillance open and denies the users full control of their own data. These factors deter the adoption of IBE schemes. CL-PKC successfully solves this problem by removing *key escrow* and ensures that a rouge KGC cannot destroy the privacy of the entire system. In IBE schemes the users need to trust the PKG to not abuse the private keys by launching passive attacks but in CL-PKC schemes they only need to trust the KGC to not actively replace the public keys.

- *Minimal Set-up*: Although IBE schemes require no set-up by the users prior to partaking in private communication, the cost at which this convenience came was too high. CL-PKC schemes require prior set-up on the part of the receiver of the message which is essentially just two steps as seen in Section 5 and Section 6.

- *Lightweight*: CL-PKC schemes are well suited to scenarios where computational power and bandwidth come at a premium, for example in mobile computation scenarios. As compared to traditional PKIs the infrastructure requirements for CL-PKC schemes are significantly low as there is no need to manage certificates. This saves the effort to transmit the certificates and check them, as shown by Dankers, Garefalakis, Schaffelhofer & Wright (2002) these factors are of considerable importance in the mobile domain and CL-PKC schemes give us an advantage.

- *Co-existence with IBE schemes*: As pointed out by Al-Riyami & Paterson (2003), the CL-PKE schemes described here are very similar to the IBE

schemes based on pairings. Thus the same infrastructure can be used to deploy them and both the schemes can peacefully co-exist.

- *Protection Against Device Compromise*: Both the original scheme as well as the derived scheme provide perfect forward secrecy. Hence, the compromise of the device along with the stored keys does not jeopardise the privacy of the stored messages.
- *Key Size*: The scheme uses relatively short public and private keys and therefore, is suitable for use in devices with limited resources, for instance mobile phones.
- *Using Biometrics*: The usage of biometric data in the encryption makes is convenient for the user to provide it since it cannot be lost or forgotten and is unique to every person. It is also relatively hard to duplicate, readily available and semi-private. Thus two-factor authentication is achieved without any significant change in the user behaviour.
- *Security*: Both the original and the modified CL-PKE schemes are secure in a fully adaptive adversarial model. The schemes provide indistinguishability under chosen cipher text attack and the security depends upon the *Decisional 3-Party Diffie-Hellman Problem* which we define in Definition 4.3.

9. Security Model

In the light of the discussion in the previous sections we have seen how a CL-PKE scheme is defined, now we take a look at the possible adversaries and define them formally. Standard security requirements of public key encryption schemes require that the encryptions are indistinguishable against a fully-adaptive chosen ciphertext attacker, i.e. it provides IND-CCA. In this definition there are two parties, the adversary and the challenger as described in Section 4.1, who participate in a sequence of games. The notion of IND-CCA security is formally defined in Section 9.1.

The security model that we define here is a natural generalisation of the fully adaptive, multi-user model presented by Boneh & Franklin (2001). To prove the security of the scheme in the lack of certificates and the presence of an adversary who has access to the master key requires defining the security model carefully. We need to allow the possibility of the adversary extracting the private keys of arbitrary users and choose the identity ID^* of the user on whose public key he is challenged. The compromise of the private keys of certain users should not jeopardise the privacy of users whose keys are still safe and our security model should encompass this requirement. However, we still need to do more to model the powers of an attacker. In traditional *public key cryptography* the public key is bound to the identity of the user by a certificate issued by the CA but in our case this is not possible and hence we need to allow the attacker to replace the public key of the user with a key of his choice. By doing so he might wish to decipher the encrypted messages sent to a certain user. But we will see that such an attack is rendered useless since messages are encrypted by binding them to the identity of the user. The decryption of messages requires the possession of the correct private key for a certain identity which can only be obtained with the cooperation of the KGC who provides the *partial private key* to derive the *full private key*. Modelling the response of a challenger whose public keys have been changed to key extraction and decryption queries should be done carefully.

We also need to take into consideration that the KGC might indulge in adversarial activities like eavesdropping on ciphertexts and making decryption queries. The KGC in our scheme is comparable to the CA in traditional PKI schemes. It is assumed that the CA does not generate certificates which authenticate arbitrary identities and public keys. Similarly, in our scheme we assume that the KGC does not replace public keys of the users. Since the KGC is in possession of the *partial private key*, he can generate private key of any user and if he chooses to replace the public keys as well then he has all the information to impersonate an user of his choice. However, we note that in our modified construction this is not possible before the user communicates the hash of his biometric identity to the KGC so there is an additional hurdle which a malicious KGC must overcome.

There however is a difference to the CA scenario, in traditional PKIs if the CA misbehaves then it is easy to point that out by observing the existence of two valid certificates for same identity but in our scheme a new public key can be created by

the user or the KGC and it is not possible to decide which is the case. Al-Riyami & Paterson (2003) showed that this can be avoided by allowing users to choose identifiers which bind their public keys and identities together. This will help to pinpoint a misbehaving KGC in the event there are two different working public keys for the same identity. There has been considerable debate on the whether the security model described here correctly captures the capabilities of an attacker against certificateless encryption, the issue has been discussed at length in Dent (2006b).

9.1. Chosen cipher text security.

DEFINITION 9.1. *Chosen cipher text security for Certificateless Public Key Encryption: We say that a CL-PKE scheme is semantically secure against an adaptive chosen ciphertext attack (IND-CCA secure) if no polynomially bounded adversary $\mathcal{A}$ of Strong Type I, Strong Type II, New Strong Type I or New Strong Type II described in Section 9.2.2 and Section 9.3.2 has a non-negligible advantage against the challenger in the following game.*

Setup *The challenger takes a security parameter k and runs the* `Setup` *algorithm. It gives $\mathcal{A}$ the resulting system parameters. If $\mathcal{A}$ is of Type I, then the challenger keeps the master secret key to itself, otherwise, it gives master secret key to $\mathcal{A}$.*

Phase 1 *$\mathcal{A}$ issues a sequence of requests, each request being either a partial private key extraction, a private key extraction, a request for a public key, a replace public key command or a decryption query for a particular user. These queries may be asked adaptively, but should respect the rules on adversary behaviour defined in Section 9.2.2 and Section 9.3.2.*

Challenge Phase *Once $\mathcal{A}$ decides that Phase 1 is over it outputs the challenge identity ID^* and two equal length plaintexts $M_0, M_1 \in \mathcal{M}$. Again, the adversarial constraints for the particular set-up apply. The challenger now picks a random bit $d \in \{0, 1\}$ and computes C^*, the encryption of M_d under the current public key pk_{ID^*} for ID^*. If the output of the algorithm* `Encrypt` *is FAIL, then $\mathcal{A}$ has immediately lost the game since it has replaced a public key with one not having the correct form. Otherwise, C^* is delivered to $\mathcal{A}$.*

Phase 2 *$\mathcal{A}$ issues a second sequence of requests as in Phase 1, again subject to the rules on adversary behaviour for the set-up at hand. In particular, no private key extraction on ID^* is allowed. Moreover, no decryption query can be made on the challenge ciphertext C^* for the combination of the identity ID^* and its public key pk_{ID^*} that was used to encrypt M_d.*

Guess *Finally, $\mathcal{A}$ outputs a guess $d \in \{0,1\}$. The adversary wins the game if $d = d'$. We define $\mathcal{A}$'s advantage in this game to be $Adv_{\mathcal{A}} = |\Pr(d = d') - 1/2|$.*

The adversaries have been formally defined in Section 9.2.2 and Section 9.3.2 but for the sake of completeness we present the constraints imposed on them here. The constraints imposed on the adversaries to win the IND-CCA game under discussion are

Constraints on Strong Type I attacker *A* Strong Type I *attacker $\mathcal{A}_I$ loses the game if*

- *$\mathcal{A}_I$ extracts the private key for ID^* at any point.*
- *$\mathcal{A}_I$ extracts the private key of any identity for which it has replaced the public key.*
- *$\mathcal{A}_I$ extracts the partial private key d_{ID^*} of ID^* after replacing the public key pk_{ID^*} and before the challenge being issued.*
- *In Phase 2, $\mathcal{A}_I$ makes a decryption query on the challenge cipher text C^* for the identity ID^* without replacing the public key pk_{ID^*} used to create the challenge ciphertext.*

Constraints on Strong Type II attacker *A* Strong Type II *attacker $\mathcal{A}_{II}$ loses the game if*

- *$\mathcal{A}_{II}$ extracts the private key for ID^* at any time.*
- *$\mathcal{A}_{II}$ extracts the private key of any identity for which he has replaced the public key.*
- *$\mathcal{A}_{II}$ outputs a challenge identity ID^* for which he has replaced the public key.*
- *In Phase 2, $\mathcal{A}_{II}$ makes a decryption query on the challenge ciphertext C^* for the identity ID^* without replacing the public key pk_{ID^*} used to create the challenge ciphertext.*

Constraints on New Strong Type I attacker *All constraints that were imposed on $\mathcal{A}_I$ also apply to $\mathcal{A}_I^{new}$ except, $\mathcal{A}_I^{new}$ is allowed to extract the partial private key d_{ID^*} of ID^* after replacing the public key pk_{ID^*} and before the challenge being issued. Additionally, a* New Strong Type I *attacker $\mathcal{A}_I^{new}$ loses the game if*

- *$\mathcal{A}_I^{new}$ extracts the biometric identity BID^* of the target identity ID^* at any point.*

Constraints on New Strong Type II attacker *The constrains imposed on $\mathcal{A}_{II}^{new}$ are exactly same as the ones that were imposed on $\mathcal{A}_{II}$.*

9.2. Security Model for the Original Scheme. In the following pages we try to model the requirements discussed in this section so far by defining oracles, attackers and other key components which complete the security model.

9.2.1. Oracles. The oracles present at the attacker's disposal are the following

- Request Public Key: The public key of an user is freely available to anyone, here the attacker provides an identity ID and the oracle returns the public key pk_{ID} corresponding to that identity, the oracle generates pk_{ID} if previously undefined.
- Replace Public Key: The attacker provides an identity ID and a public key $pk'_{ID} \in \mathcal{PK}$, and the oracle replaces the previous public key of ID with pk'_{ID}. Note that pk'_{ID} should be of correct shape and thus a valid public key, such valid keys can be generated with ease by anyone from the master public key.
- Extract Partial Private Key: The attacker provides an identity ID and the oracle returns the partial private key d_{ID} corresponding to that identity.
- Extract Private Key: The attacker provides an identity ID and the oracle returns the full private key sk_{ID} corresponding to that identity.
- Decrypt: The attacker provides an identity ID and ciphertext C, the oracle responds by constructing the private key sk_{ID} corresponding to the identity ID and its associated public key pk_{ID}. The oracle then returns the decryption of C under this private key. Here we need to observe that if the attacker has replaced the public key pk_{ID} with a key of his choice then the oracle will not decrypt using a corresponding private key and in general the decryption will fail. This models a realistic scenario since there is no way for the oracle to know the secret key corresponding to a replaced public key. However, if we assume that the oracle still decrypts correctly then although unrealistic but this would provide for a better security model. Hence we provide the attacker with a more powerful decryption oracle than possible under reasonable conditions by returning correct decryption of messages encrypted using a replaced public key.

9.2.2. Adversaries. We consider two kinds of attackers for a CL-PKE scheme which are

- Strong Type I attacker: This attacker is designed to model a third party who is trying to gain information about the plaintext by observing the corresponding ciphertext. Such an attacker $\mathcal{A}_I$ does not have access to the *master secret*

key but can request public keys, replace public keys with keys of his choice, extract partial private keys and make decryption queries, all for identities of his choice. However, there are certain restrictions on the actions which he could perform, specifically related to the target identity which is denoted by ID^*, the restrictions are

- $\mathcal{A}_I$ cannot extract private key for ID^* at any point.
- $\mathcal{A}_I$ cannot extract the private key of any identity for which it has replaced the public key. Allowing for such a possibility would be unreasonable since the public key has been replaced.
- $\mathcal{A}_I$ cannot extract the partial private key d_{ID^*} of ID^* if he replaced the public key pk_{ID^*} before the challenge was issued. Allowing this would enable the attacker to receive ciphertexts encrypted with a public key of his choice and possession of the partial private key will allow him to trivially decrypt by generating the corresponding private key.
- In *Phase 2* described in Section 9.1, $\mathcal{A}_I$ cannot make a decryption query on the challenge cipher text C^* for the identity ID^* unless the public key pk_{ID^*} used to create the challenge ciphertext has been replaced.

If a Strong Type I attacker indulges in any of the actions described above to answer the challenges then he might be able to achieve success but we define that as a loss.

- Strong Type II attacker: This attacker is designed to model the notion of an honest-but-curious KGC and the scheme should be safe from this kind of attacker. Such an attacker $\mathcal{A}_{II}$ does have access to the *master secret key* but is trusted not to replace the public keys of the users. However, we still allow $\mathcal{A}_{II}$ to replace the public keys under certain restrictions, this provides us a better model. The adversary $\mathcal{A}_{II}$ can compute *partial private keys* for itself from the *master secret key*. $\mathcal{A}_{II}$ can request public keys, extract private keys and make decryption queries, all for identities of its choice. However there are certain restrictions on the actions which he could perform, specifically related to the target identity which is denoted by ID^*, the restrictions are

  - $\mathcal{A}_{II}$ cannot extract private key for ID^* at any time.
  - $\mathcal{A}_{II}$ cannot extract the private key of any identity for which he has replaced the public key.
  - $\mathcal{A}_{II}$ does not query the partial private key oracle since it can compute d_{ID} for identity ID from msk which it possesses.
  - $\mathcal{A}_{II}$ cannot output a challenge identity ID^* for which he has replaced the public key.

– In *Phase 2* described in Section 9.1, $\mathcal{A}_{II}$ cannot make a decryption query on the challenge cipher text C^* for the identity ID^* unless the public key pk_{ID^*} used to create the challenge ciphertext has been replaced.

If a Strong Type II attacker indulges in any of the actions described above to answer the challenges then he might be able to achieve success but we define that as a loss.

9.3. Security Model for the Modified Scheme. As mentioned earlier by modifying the scheme we aim to provide two-factor security. We would like to ensure that even if the device is compromised the attacker cannot decrypt the encrypted messages although he may have the access to the secret values in the device which were used to encrypt the messages. We introduced the usage of biometric data to achieve this and now we define an appropriate adversarial model which captures the threat of the device being compromised. At this point we should also note that we assume that the biometric identity of the user is only accessible to him and people who know the user, this adds a social factor to the scheme. We cannot protect against an attacker who has easy access to the user's biometric data and also manages to compromise the device. Such an attacker has all information that the user has and hence there is no way to differentiate him from a valid user. We define oracles and attackers in the subsequent sections to capture our threat model.

9.3.1. Oracles. In addition to the oracles already defined in Section 9.2.1 the attacker has access to one further oracle:

- Extract Biometric Identity: The biometric identity of the user is treated as a semi-private value, which means that it is easily available to the user and people who know the user but is significantly harder for an attacker to obtain who does not know the user. This oracle returns the biometric identity BID of an user after the attacker provides a public identity ID.

9.3.2. Adversaries. Due to inclusion of biometrics our scheme permits for a stronger *Strong Type I Attacker* however *Strong Type II Attacker* essentially remains the same as defined previously in Section 9.2.2. We now formally define the attackers:

- New Strong Type I Attacker: This attacker in our modified scheme is similar to the *Strong Type I Attacker* apart from a few changes that we describe. The attacker $\mathcal{A}_I^{new}$ tries to model a third party who is trying to gain some information about the encrypted messages. $\mathcal{A}_I^{new}$ has compromised the device and hence has access to partial private key d_{ID^*} and the secret value x_{ID^*}, he may also replace the public key pk_{ID^*} of the identity ID^* whose device he has gained control of, although this does not give him any extra advantage since he already knows x_{ID^*}. Thus $\mathcal{A}_I^{new}$ can receive messages

encrypted using a public key of his choice while in possession of the partial private key, in the original scheme such a scenario would destroy all privacy but here due to the scheme using biometric identity BID^* of identity ID^* to encrypt data the attacker gains nothing. However, $\mathcal{A}_I^{new}$ does have certain restrictions imposed on his actions in addition to the ones that were imposed on $\mathcal{A}_I$, these are:

  - $\mathcal{A}_I^{new}$ cannot extract the biometric identity BID^* of the target identity ID^* at any point. This is in line with our previous assumption that $\mathcal{A}_I^{new}$ does not have access to biometrics of the users unknown to him.

- New Strong Type II Attacker: Just like the *Strong Type II Attacker* this attacker is designed to model an honest-but-curious KGC. $\mathcal{A}_{II}^{new}$ has all the powers which $\mathcal{A}_{II}$ had and additionally due to the modifications in the scheme he can also extract the biometric identity BID^* of the target identity ID^* as well as for the other identities. All the constraints which were applied to $\mathcal{A}_{II}$ also apply here additionally we also assume that $\mathcal{A}_{II}^{new}$ does not compromise the device of the target identity ID^*.

10. Concrete Original Construction

10.1. The Construction. The encryption scheme uses *bilinear map groups* namely $\mathbb{G}$ of prime order p for some large prime p, the requirements of such a bilinear map have been formally defined in Definition 4.1. We further require that the 3-DDH for $\mathbb{G}$ described in Definition 4.3 is intractable.

The scheme proceeds through a sequence of seven subroutines which are described in the pages to follow.

ALGORITHM 10.1. `Setup`.

Performed by the KGC — this is the first step in the encryption scheme in this step the KGC generates the *master public key* mpk and the *master secret key* msk after receiving the system security parameter k and bit length n of the *public identity* ID. Let $\mathbb{G}$ be a bilinear map group of order $p > 2^k$ and g be a generator for $\mathbb{G}$.

Input: $(1^k, n)$.

Output: (mpk, msk).

1. Choose $\gamma \xleftarrow{R} \mathbb{Z}_p^\times$.
2. Set $g_1 = g^\gamma$.
3. Choose $g_2 \xleftarrow{R} \mathbb{G}$.
4. Choose vectors $(u', u_1, \ldots, u_n)$, $(v', v_1, \ldots, v_n) \xleftarrow{R} \mathbb{G}^{n+1}$.
5. Write $ID = i_1 i_2 \ldots i_n$ and $w = w_1 w_2 \ldots w_n$ as bit strings with $i_j, w_j \in \{0, 1\}$.
6. Define hash functions

$$F_u : \begin{array}{ccc} \{0,1\}^n & \longrightarrow & \mathbb{G}, \\ ID & \longmapsto & u' \prod_{0 \le j \le n} u_j^{i_j} \end{array}$$

and

$$F_v : \begin{array}{ccc} \{0,1\}^n & \longrightarrow & \mathbb{G}, \\ w & \longmapsto & v' \prod_{0 \le j \le n} v_j^{w_j} \end{array} .$$

7. Choose a collision resistant hash function $H : \{0,1\}^* \to \{0,1\}^n$ as required in Definition 4.4.
8. Define the *master public key* as $mpk \leftarrow (g, g_1, g_2, u', u_1, \ldots, u_n, v', v_1, \ldots, v_n)$.
9. Define the *master secret key* as $msk \leftarrow \gamma$.

ALGORITHM 10.2. `Extract`.

Performed by the KGC — this step is executed after the *Receiver* authenticates himself as ID to the KGC. The KGC then generates the *partial private key* d_{ID} which is subsequently communicated to the *Receiver* in a secure manner.

Input: (mpk, msk, ID).

Output: d_{ID}.

1. Choose $r \xleftarrow{R} \mathbb{Z}_p^\times$.

2. Compute $d_{ID} \leftarrow (d_1, d_2) = (g_2^{\gamma} \cdot F_u(ID)^r, g^r)$.
3. Return d_{ID}.

ALGORITHM 10.3. `SetSec`.
Performed by the Receiver — in this step the *Receiver* computes a randomly chosen secret value x_{ID}.
Input: mpk.
Output: x_{ID}.

1. Choose $x_{ID} \xleftarrow{\$} \mathbb{Z}_p^{\times}$.
2. Return x_{ID}.

ALGORITHM 10.4. `SetPub`.
Performed by the Receiver — in this step the *Receiver* computes and freely distributes the *public key* pk_{ID}.
Input: (x_{ID}, mpk).
Output: pk_{ID}.

1. Compute $pk_{ID} \leftarrow (X, Y) = (g^{x_{ID}}, g_1^{x_{ID}})$.
2. Return pk_{ID}.

ALGORITHM 10.5. `SetPriv`.
Performed by the Receiver — in this step the *Receiver* computes his *private key* sk_{ID} which he uses to decrypt messages encrypted using pk_{ID} and ID.
Input: $(x_{ID}, d_{ID}, mpk, ID)$.
Output: sk_{ID}.

1. Choose $r' \xleftarrow{\$} \mathbb{Z}_p^{\times}$.
2. Set $(d_1, d_2) \leftarrow d_{ID}$.
3. Compute private key as

$$sk_{ID} \leftarrow (s_1, s_2) = \left(d_1^{x_{ID}} \cdot F_u(ID)^{r'}, d_2^{x_{ID}} \cdot g^{r'}\right).$$

4. Return sk_{ID}.

ALGORITHM 10.6. `Encrypt`.
Performed by the Sender — this step computes the encryption of message $m \in \mathbb{G}_T$.
Input: (m, pk_{ID}, ID, mpk).
Output: C.

1. If $e(X, g_1)/e(g, Y) = 1_{\mathbb{G}_T}$ then
2. Choose $s \xleftarrow{\$} \mathbb{Z}_p^{\times}$.

3. Set $(C_0, C_1, C_2) \leftarrow (m \cdot e(Y, g_2)^s, g^s, F_u(ID)^s)$.
4. Compute $w \leftarrow H(C_0, C_1, C_2, ID, pk_{ID})$.
5. Set $C_3 \leftarrow F_v(w)^s$.
6. Return $C = (C_0, C_1, C_2, C_3)$.
7. Else
8. Return FAIL .

In Step 1 in Algorithm 10.6 we check for the correctness of the public key pk_{ID}, if pk_{ID} is of the right shape then the ciphertext C is computed and returned else the algorithm aborts with FAIL.
In Step 4 in Algorithm 10.6 we include the hash $F_v(w)$ of w, since our encryption scheme is homomorphic this hash acts as a signature on ciphertext and defeats the adversary's attempt to create valid encryptions by combining other encryptions and winning the IND-CCA game described in Section 9.1.

ALGORITHM 10.7. `Decrypt`.
Performed by the Receiver — this decrypts the message encrypted using receiver's private key pk_{ID} and identity ID.
Input: (C, sk_{ID}, mpk).
Output: m.

1. Set $(C_0, C_1, C_2, C_3) \leftarrow C$.
2. Let $w \leftarrow H(C_0, C_1, C_2, ID, pk_{ID})$.
3. If $e(C_1, F_u(ID) \cdot F_v(w)) = e(g, C_2 \cdot C_3)$ then
4. Set $(s_1, s_2) \leftarrow sk_{ID}$.
5. Compute $m \leftarrow C_0 \cdot \dfrac{e(C_2, s_2)}{e(C_1, s_1)}$.
6. Return m.
7. Else
8. Return FAIL .

In Step 3 in Algorithm 10.7 we check for the validity of the ciphertext by checking the hash $F_v(w)$, if C is a valid ciphertext then we proceed with the decryption else the algorithm aborts with FAIL.

We check for completeness by substituting the values for $(C_0, C_1, C_2, s_1, s_2)$ in

$$(s_1, s_2) = \left(d_1^{x_{ID}} \cdot F_u(ID)^{r'}, d_2^{x_{ID}} \cdot g^{r'}\right).$$

This can be rewritten as

$$\begin{aligned}(s_1, s_2) &= \left((g_2^{\gamma} \cdot F_u(ID)^r)^{x_{ID}} \cdot F_u(ID)^{r'}, (g^r)^{x_{ID}} \cdot g^{r'}\right) \\ &= \left(g_2^{\gamma x_{ID}} \cdot F_u(ID)^{r x_{ID} + r'}, g^{r x_{ID} + r'}\right) \\ &= \left(g_2^{\gamma x_{ID}} \cdot F_u(ID)^t, g^t\right)\end{aligned}$$

where $t = rx_{ID} + r'$. Now, substituting the values we obtain

$$\begin{aligned} C_0 \cdot \frac{e(C_2, s_2)}{e(C_1, s_1)} &= m \cdot e(Y, g_2)^s \cdot \frac{e\left(F_u(ID)^s, g^t\right)}{e\left(g^s, g_2^{\gamma x_{ID}} \cdot F_u(ID)^t\right)} \\ &= m \cdot e\left(g_1^{x_{ID}}, g_2^s\right) \cdot \frac{e\left(F_u(ID)^s, g^t\right)}{e\left(g^s, g_2^{\gamma x_{ID}}\right) \cdot e\left(g^s, F_u(ID)^t\right)} \\ &= m \cdot \frac{e\left(g^{\gamma x_{ID}}, g_2^s\right)}{e\left(g^s, g_2^{\gamma x_{ID}}\right)} \\ &= m. \end{aligned}$$

Thus we conclude that the decryption of an encrypted message gives us back the original message m and the scheme functions correctly.

10.2. Security Reduction. In this section we define the security of our scheme. We base the security of our scheme on the intractability of the 3-DDH in the groups which are used by our construction. To capture the idea of security we first define theorems which highlight the advantage gained by an attacker in the described security model. After that we provide the proofs of these theorems.

THEOREM 10.8. *Suppose $\mathcal{A}_I$ is a Strong Type I adversary that runs in time t, makes at most q_d decryption queries, q_{ppk} partial private key queries, and q_{pk} private key queries. Then there exists*

- *an adversary $\mathcal{A}'$ against the 3-DDH that has advantage $Adv_{\mathcal{A}'}^{3\text{-}DDH}(k)$ and runs in time $\mathcal{O}(t) + \mathcal{O}\left(\varepsilon^{-2} \ln \delta^{-1}\right)$ for sufficiently small ε and δ, and*
- *an adversary $\mathcal{A}''$ against the collision resistance of the hash function H that runs in time $\mathcal{O}(t)$ and has advantage $Adv_{\mathcal{A}''}^{CR}(k)$*

such that the advantage $Adv_{\mathcal{A}_I}^{CL\text{-}CCA}(k)$ of $\mathcal{A}_I$ is bounded by

$$Adv_{\mathcal{A}_I}^{CL\text{-}CCA}(k) < 8(q_{ppk} + q_{pk})q_d(n+1)^2 \cdot (8 \cdot Adv_{\mathcal{A}'}^{3\text{-}DDH}(k) + \delta) + Adv_{\mathcal{A}''}^{CR}(k).$$

THEOREM 10.9. *Suppose $\mathcal{A}_{II}$ is a Strong Type II adversary that runs in time t, makes at most q_d decryption queries and q_{pk} private key queries. Then there exists*

- *an adversary $\mathcal{A}'$ against the 3-DDH that has advantage $Adv_{\mathcal{A}'}^{3\text{-}DDH}(k)$ and runs in time $\mathcal{O}(t) + \mathcal{O}\left(\varepsilon^{-2} \ln \delta^{-1}\right)$ for sufficiently small ε and δ, and*
- *an adversary $\mathcal{A}''$ against the collision resistance of the hash function H that runs in time $\mathcal{O}(t)$ and has advantage $Adv_{\mathcal{A}''}^{CR}(k)$*

such that the advantage $Adv_{\mathcal{A}_{II}}^{CL\text{-}CCA}(k)$ of $\mathcal{A}_{II}$ is bounded by

$$Adv_{\mathcal{A}_{II}}^{CL\text{-}CCA}(k) < 8q_{pk}q_d(n+1)^2 \cdot (8 \cdot Adv_{\mathcal{A}'}^{3\text{-}DDH}(k) + \delta) + Adv_{\mathcal{A}''}^{CR}(k).$$

Interpretation The advantage of the adversaries against the scheme presented in Section 10 is bound by the results of Theorem 10.8 and Theorem 10.9. Thus if either $\mathcal{A}_I$ or $\mathcal{A}_{II}$ exist such that it has a significant advantage against breaking the indistinguishability of the scheme then this would imply the existence of attackers with significant advantage against both solving the 3-DDH and breaking the collision resistant hash function H. Such attackers can be used to device algorithms which solve the 3-DDH and find collisions of H in polynomial time.

For the algorithms mentioned in the above theorems a solution can be conveniently found for which $\varepsilon \approx 0$ or $\delta \approx 0$ but such a solution provides no advantage to the adversary as it is too slow and the equation is rendered meaningless. To gain significant advantage in breaking the security of the scheme the adversary must find an algorithm with sufficiently small ε and δ such that the algorithm runs in polynomial time.

Framework The motivation behind our security reduction is that we want to use the attacker to solve the 3-DDH. However, our described protocol cannot be used to do this in its original form hence to do so we need to feed the attacker with values which are generated differently than in the original protocol. We create an environment which interacts with the attacker during the IND-CCA game. The attacker, assuming that he works within the scheme scenario, should not be able to detect any difference since that could change his entire behaviour. Thus the main challenge underlying the whole reduction is to change the original protocol to model the 3-DDH without inducing changes which the attacker can ascertain.

The idea behind proving Theorem 10.8 and Theorem 10.9 is that we want to reduce the challenge presented to the attacker to a sequence of randomly generated values which model the problem defined in Definition 4.3. The argument then follows that if the attacker cannot differentiate between a genuine challenge from the one which models the 3-DDH then he cannot predict what he is computing. Thus, in this modified scenario if he manages to win the IND-CCA game with a significant advantage then we can conclude that such an attacker can solve the 3-DDH. Thus we show that the security of our scheme rests on the intractability of the 3-DDH in $\mathbb{G}$. This is a very generic way of describing things and we define them formally in pages to follow.

We achieve our agenda by starting with the scenario which represents the situation when the scheme begins and then gradually change it in ways which the attacker cannot detect. Finally, we end up with a scenario based on which we can draw conclusions about attacker's advantage. We present such changes in the form of games that we describe subsequently.

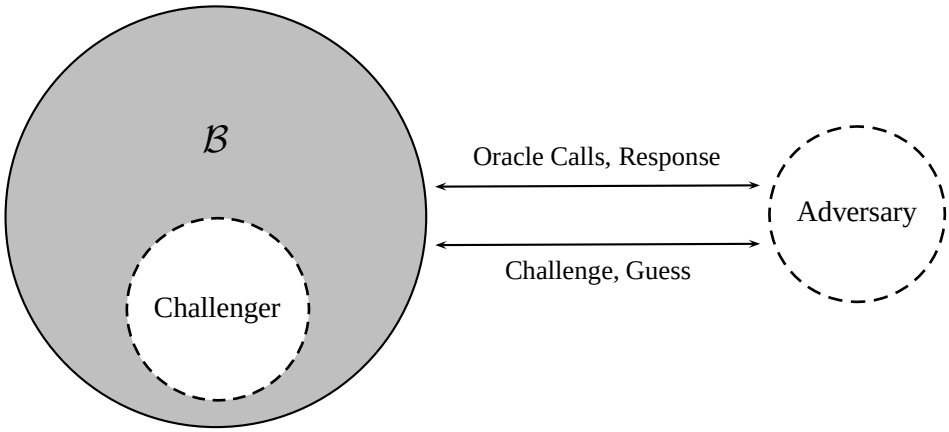

Figure 10.1: The arrangement of the security reduction. The *Challenger* feeds the attack environment $\mathcal{B}$ with values at different stages of game hopping. The attack environment $\mathcal{B}$ is responsible for setting up the scheme and answering oracle calls. The *Adversary* interacts with the attack environment $\mathcal{B}$ to get responses to the oracle calls that he makes. Finally, the attack environment $\mathcal{B}$ presents the *Adversary* with the indistinguishability challenge and the *Adversary* makes a guess.

LEMMA 10.10. *For all i, let S_i denote the event that an attacker is successful in Game i and thus outputs the correct guess d' of d, the advantage of the attacker is denoted by $Adv_i = |\Pr(S_i) - 1/2|$.*
Assume that Game i is a game where the attacker wins with success probability $\Pr(S_i)$. E denotes an event which may occur during the attacker's execution such that

- *E is detectable by the simulator;*
- *E is independent of S_i;*
- *Game i and Game $i+1$ are identical unless E occurs, in which case the game halts and outputs a random bit.*

Then we have

$$Adv_{i+1} = \Pr(\neg E) \cdot Adv_i.$$

PROOF (Lemma 10.10). The proof of the lemma is straight forward and was first presented by Dent (2006a). We note here that the games i and $i+1$ are identical when E does not occur. Therefore, we have

$$S_{i+1} \cap \neg E = S_i \cap \neg E,$$
$$\neg S_{i+1} \cap \neg E = \neg S_i \cap \neg E$$

and by the independence assumption

$$\Pr(S_i \cap E) = \Pr(S_i) \cdot \Pr(E).$$

During the game $i+1$ if E occurs then the game halts and outputs a random bit which implies $\Pr(S_{i+1} \mid E) = \frac{1}{2}$.
Thus,

$$\begin{aligned}\Pr(S_{i+1} \cap E) &= \Pr(S_{i+1} \mid E) \cdot \Pr(E) \\ &= \frac{1}{2} \cdot \Pr(E).\end{aligned}$$

Similarly, we have

$$\Pr(\neg S_{i+1} \cap E) = \frac{1}{2} \cdot \Pr(E).$$

Now,

$$\begin{aligned}|\Pr(S_{i+1}) - 1/2| &= \left|\Pr(S_{i+1} \cap E) + \Pr(S_{i+1} \cap \neg E) - \frac{1}{2}\right| \\ &= \left|\frac{1}{2} \cdot \Pr(E) + \Pr(S_i) \cdot \Pr(\neg E) - \frac{1}{2}\right| \\ &= \left|\frac{1}{2} \cdot (\Pr(E) - 1) + \Pr(S_i) \cdot \Pr(\neg E)\right| \\ &= \left|-\frac{1}{2} \cdot \Pr(\neg E) + \Pr(S_i) \cdot \Pr(\neg E)\right| \\ &= |\Pr(S_i) - 1/2| \cdot \Pr(\neg E)\end{aligned}$$

and thus we obtain $Adv_{i+1} = \Pr(\neg E) \cdot Adv_i$. □

We now proceed to prove Theorem 10.8 and Theorem 10.9, these theorems were originally presented and proved by Dent, Libert & Paterson (2008). Our proof follows the same pattern and we have explained the results obtained in further detail.

PROOF (Theorem 10.8). We proceed to prove Theorem 10.8 through a series of games which involve *Type I* attacker $\mathcal{A}_I$ who tries to guess the hidden bit d in the IND-CCA game described in Section 9.1. The attacker outputs a guess d' on conclusion of the sequence of games.

Game 1 After the scheme has been specified as described in Section 10.1 we begin *Game 1*, in this game we designate the list of identities with which the *Type I* attacker $\mathcal{A}_I$ interacts and the manner of those interactions. The actual attack environment with which attacker interacts is denoted by $\mathcal{B}$, it generates the

master key, the public parameters, and the initial user's public keys and secret values. It is assumed that $\mathcal{B}$ can answer decryption queries without the knowledge of matching secret values for changed public keys. In this real attack we assume $PPK = \{ID_1, \ldots, ID_{q_{ppk}}\}$ be the list of identities for which the attacker asks for a *partial private key* extraction and let $PK = \{ID'_1, \ldots, ID'_{q_{pk}}\}$ be the list of identities for which the attacker asks for a *private key* extraction. Let $D = \{w_1, \ldots, w_{q_d}\}$ be the list of strings involved in decryption queries, where $w_j = H(C_0, C_1, C_2, ID_j, pk_j)$. The identity and the public key under attack by the attacker is denoted by (ID^*, pk_{ID^*}). $C^* = (C_0^*, C_1^*, C_2^*, C_3^*)$ denotes the challenge cipher text and $w^* = H(C_0^*, C_1^*, C_2^*, ID^*, pk_{ID^*})$. The challenge ciphertext is handed to the attacker as defined in Section 9.1 and the attacker must respond accordingly, this concludes *Game 1*.

Game 2 In this game we change the generation of the *master public key* in a way that does not effect its distribution. We achieve this as follows, the attack environment $\mathcal{B}$ picks $a, b \in_R \mathbb{Z}_p^\times$ and sets $g_1 = g^a, g_2 = g^b$, chooses $\kappa_u, \kappa_v \in \{0, \ldots, n\}$. Let τ_u, τ_v be integers such that $(n+1)\tau_u, (n+1)\tau_v < p$, we specify the particular values in *Game 7*. The environment selects $x'_u \in_R \mathbb{N}_{<\tau_u}, x'_v \in_R \mathbb{N}_{<\tau_v}$ and vectors $(x_{u,1}, \ldots, x_{u,n}) \in_R \mathbb{N}^n_{<\tau_u}, (x_{v,1}, \ldots, x_{v,n}) \in_R \mathbb{N}^n_{<\tau_v}$. It also picks $y'_u, y'_v \in_R \mathbb{Z}_p$ and vectors $(y_{u,1}, \ldots, y_{u,n}), (y_{v,1}, \ldots, x_{y,n}) \in_R \mathbb{Z}_p^n$. The remaining *master public key* elements are chosen as follows

$$u' = g_2^{x'_u - \kappa_u \tau_u} g^{y_{u'}}, \quad u_j = g_2^{x_{u,j}} g^{y_{u,j}} \quad \text{for } 0 \leq j \leq n \tag{10.11}$$

and

$$v' = g_2^{x'_v - \kappa_v \tau_v} g^{y_{v'}}, \quad v_j = g_2^{x_{v,j}} g^{y_{v,j}} \quad \text{for } 0 \leq j \leq n. \tag{10.12}$$

As seen the values of $(x'_u, x'_v, x_{u,j}, x_{v,j})$ and $(y'_u, y'_v, y_{u,j}, y_{v,j})$ are chosen differently this is because they are used in (10.11) and (10.12) to define (u', u_j, v', v_j). The exponents of g and g_2 are meaningful modulo p only hence we need to restrict the choice of $(x'_u, x_{u,j})$ and $(x'_v, x_{v,j})$ to $\mathbb{N}_{<\tau_u}$ and $\mathbb{N}_{<\tau_v}$ respectively.

We note that the number of possible values of u_j and v_j are $p\tau_u$ and $p\tau_v$ respectively because of the choice of the exponents in (10.11) and (10.12). However, the distribution of the *master public key* remains unchanged because all the elements of $\mathbb{G}$ are equally likely to be chosen. This is due to the fact that the exponents are randomly chosen from lists containing $p\tau_u$ and $p\tau_v$ elements where each exponent appears equally frequently. Therefore, *Game 1* is identical to *Game 2*, hence $\Pr(S_1) = \Pr(S_2)$ and thus $Adv_1 = Adv_2$.

Game 3 This game is the same as *Game 2* except the environment halts if the attacker submits a decryption query (C, ID, pk) for a well-formed cipher text $C = (C_0, C_1, C_2, C_3)$ where w is equal to the value of a previously submitted cipher text or w is equal to w^*. Such a legal decryption query means either

$C \neq C^*$ or $(ID, pk) \neq (ID^*, pk^*)$. Both situations reveal a collision of the hash function, thus we can construct an algorithm $\mathcal{A}''$ such that $|\Pr(S_2) - \Pr(S_3)| \leq Adv^{CR}_{\mathcal{A}''}(k)$, since the only difference between *Game 3* and *Game 2* is the condition enforced by the collision of the hash function hence the difference of the success probabilities is the lower bound on $Adv^{CR}_{\mathcal{A}''}(k)$.

We note that the algorithm $\mathcal{A}''$ which runs in polynomial-time does exist even though it requires to simulate a decryption oracle which is clearly a non-polynomial-time function. However, by modifying the way we define the master key using the values defined in the previous games it is possible to successfully decrypt using only the ciphertext and the public key, we show this in *Game 8*.

Game 4 In this game we modify the environment by flipping a coin $c_{mode} \in_R \{0,1\}$ at the beginning of the game. If $c_{mode} = 0$ then $\mathcal{B}$ expects that $\mathcal{A}_I$ will never extract the matching partial private key and choose to be challenged on the identity whose public key was replaced. If $c_{mode} = 1$ then $\mathcal{B}$ expects that $\mathcal{A}_I$ will extract the partial private key of the identity under attack at some point.
After the challenge is issued, if $c_{mode} = 0$ and $\mathcal{A}_I$ does not replace the public key then $\mathcal{B}$ aborts and simulates $\mathcal{A}_I$'s output as $d' \in_R \{0,1\}$. Similarly, $\mathcal{B}$ aborts if $c_{mode} = 1$ and $\mathcal{A}_I$ has replaced the challenge public key. The random variable c_{mode} is completely hidden from the attacker $\mathcal{A}_I$ thus he cannot know when the game is aborted. $\mathcal{B}$ aborts with a probability of $1/2$, this gives us $Adv_4 = \frac{1}{2} \cdot Adv_3$ according to Lemma 10.10.

Game 5 Before modifying *Game 4* we redefine $F_u(ID)$ and $F_v(w)$ with the help of specific choice of the values $(x_{u,j}, y_{u,j}, x_{v,j}, y_{v,j})$ from (10.11) and (10.12). To this end we define

$$J_u(ID) = x'_u + \sum_{j=1}^{n} i_j x_{u,j} - \kappa_u \tau_u, \qquad K_u(ID) = y'_u + \sum_{j=1}^{n} i_j y_{u,j},$$
$$J_v(w) = x'_v + \sum_{j=1}^{n} w_j x_{v,j} - \kappa_v \tau_v, \qquad K_v(w) = y'_v + \sum_{j=1}^{n} w_j y_{v,j},$$

where $ID = i_1 \ldots i_n$ and $w = w_1 \ldots w_n$ are n-bit strings. For any string $ID, w \in \{0,1\}^n$ we have

$$F_u(ID) = u' \cdot \prod_{j=1}^{n} u_j^{i_j}.$$

We can rewrite this as

$$\begin{aligned} F_u(ID) &= g_2^{x'_u - \kappa_u \tau_u} g^{y_{u'}} \cdot \prod_{j=1}^{n} (g_2^{x_{u,j}} g^{y_{u,j}})^{i_j} \\ &= g_2^{x'_u - \kappa_u \tau_u} g^{y_{u'}} g_2^{\sum_{j=1}^{n} i_j x_{u,j}} g^{\sum_{j=1}^{n} i_j y_{u,j}} \\ &= g_2^{x'_u + \sum_{j=1}^{n} i_j x_{u,j} - \kappa_u \tau_u} g^{y_{u'} + \sum_{j=1}^{n} i_j y_{u,j}} \\ &= g_2^{J_u(ID)} \cdot g^{K_u(ID)}. \end{aligned}$$

Similarly, we obtain

$$F_v(w) = v' \cdot \prod_{j=1}^{n} v_j^{w_j}.$$

This can be rewritten as

$$F_v(w) = g_2^{J_v(ID)} \cdot g^{K_v(ID)}.$$

Game 5 is identical to *Game 4* except when the attacker $\mathcal{A}_I$ outputs its guess d' of d then the environment $\mathcal{B}$ checks whether $J_u(ID^*) = J_v(w^*) = 0 \bmod p$. If $J_u(ID^*) \neq 0$ or $J_v(w^*) \neq 0$ then $\mathcal{B}$ aborts and simulates $\mathcal{A}_I$'s output choosing uniformly randomly $d' \in_R \{0, 1\}$. The values $(x'_u, x_{u,1}, \ldots, x_{u,n})$ and $(x'_v, x_{v,1}, \ldots, x_{v,n})$ are hidden from the attacker and hence it can only come up with ID^* such that $J_u(ID^*) = 0$ by chance.

Therefore we have

$$\begin{aligned} &\Pr\left(J_u(ID^*) = 0 \mod p\right) \\ &= \Pr\left(J_u(ID^*) = 0 \mod p \mid J_u(ID^*) = 0 \mod \tau_u\right) \cdot \\ &\quad \Pr\left(J_u(ID^*) = 0 \mod \tau_u\right) \\ &= \frac{1}{n+1} \cdot \frac{1}{\tau_u} \\ &= \frac{1}{\tau_u(n+1)}. \end{aligned}$$

We note here that $\Pr\left(J_u(ID^*) = 0 \mod p \mid J_u(ID^*) = 0 \mod \tau_u\right)$ is precisely $\frac{1}{n+1}$ due to the fact that under the given circumstances $J_u(ID^*)$ is a multiple of τ_u which would in turn imply that $x'_u + \sum_{j=1}^{n} i_j x_{u,j}$ is also a multiple of τ_u. Now, there is exactly one value out of the possible $n+1$ values that κ_u can take to make $J_u(ID^*) = 0$. Also, we have $\Pr\left(J_u(ID^*) = 0 \mod \tau_u\right) = \frac{1}{\tau_u}$ because this can

happen only if $J_u(ID^*) = 0$ hence there is exactly one choice for $J_u(ID^*)$ out of the possible τ_u choices. This leads us to the given probability.

Similarly, we obtain $\Pr\left(J_v(w^*) = 0 \mod p\right) = \dfrac{1}{\tau_v(n+1)}$ since $J_v(w^*) = 0$ purely by chance. Now applying the game hopping described in Lemma 10.10 leads us to $Adv_5 = \dfrac{Adv_4}{\tau_u \tau_v (n+1)^2}$.

Game 6 In this game we modify the way the environment $\mathcal{B}$ generates the challenge ciphertext. $\mathcal{B}$ picks up a random value $c \in_R \mathbb{Z}_p^\times$ and sets $C_1^* = g^c$. Let identity ID^*'s public key at the challenge phase be denoted by $pk_{ID^*} = (X^*, Y^*)$. $\mathcal{B}$ flips a coin $d^* \in_R \{0,1\}$ and computes

$$C_0^* = m_{d^*} \cdot e(Y^*, g_2)^c,$$
$$C_2^* = C_1^{*K_u(ID^*)} = (g^c)^{K_u(ID^*)}$$

and

$$C_3^* = C_1^{*K_v(w^*)} = (g^c)^{K_v(w^*)}$$

where $w^* = H(C_0^*, C_1^*, C_2^*, ID^*, pk_{ID^*})$. The returned ciphertext $(C_0^*, C_1^*, C_2^*, C_3^*)$ has the correct distribution since $J_u(ID^*) = J_v(w^*) = 0$ and hence we have $Adv_6 = Adv_5$.

Game 7 In this game we modify *Game 6* such that after $\mathcal{A}_I$ outputs his guess d' the environment $\mathcal{B}$ checks if one of the following conditions are true

- $c_{mode} = 0$ and $J_u(ID_i) = 0 \mod \tau_u$ for some $ID_i \in PPK$ with $i \in \{1, \ldots, q_{ppk}\}$.
- $J_u(ID_j) = 0 \mod \tau_u$ for some $ID_j \in PK$ with $j \in \{1, \ldots, q_{pk}\}$.
- $J_v(w_\ell) = 0 \mod \tau_v$ for some $w_\ell \in D$ with $\ell \in \{1, \ldots, q_d\}$.

We define E as the event that any of the aforementioned conditions hold. We observe that Dent's game hopping technique cannot be applied at this stage since even though E is recognisable there is no surety that it is independent of S_6. Attacker $\mathcal{A}_I$ can model his queries by choosing PK and PPK depending upon m_d in such a way that $\Pr(E)$ is significantly different in different query sequences. We use a re-normalisation technique suggested in Waters (2005) to circumvent this problem. We derive a non-negligible lower bound for $\Pr(\neg E)$ for any set of oracle queries. We estimate the probability that E occurs during a particular set of oracle queries that are made while running $\mathcal{A}_I$ and then add *artificial aborts* to ensure that $\mathcal{A}_I$ aborts with exactly the probability given by this lower bound. We now derive the theoretical lower bound.

For $c_{mode} = 1$ we find

$$\Pr(\neg E) = \Pr\left(\bigwedge_{ID \in PK} J_u(ID) \neq 0 \mod \tau_u \wedge \bigwedge_{w \in D} J_v(w) \neq 0 \mod \tau_v \mid J_u(ID^*) = 0 \mod \tau_u \wedge J_v(w^*) = 0 \mod \tau_v\right)$$

$$= \Pr\left(\bigwedge_{ID \in PK} J_u(ID) \neq 0 \mod \tau_u \mid J_u(ID^*) = 0 \mod \tau_u\right) \cdot \Pr\left(\bigwedge_{w \in D} J_v(w) \neq 0 \mod \tau_v \mid J_v(w^*) = 0 \mod \tau_v\right).$$

Now, considering the first term of right hand side

$$\begin{aligned}
&\Pr\left(\bigwedge_{ID \in PK} J_u(ID) \neq 0 \mod \tau_u \mid J_u(ID^*) = 0 \mod \tau_u\right) \\
&= 1 - \Pr\left(\bigvee_{ID \in PK} J_u(ID) = 0 \mod \tau_u \mid J_u(ID^*) = 0 \mod \tau_u\right) \\
&\geq 1 - \sum_{ID \in PK} \Pr(J_u(ID) = 0 \mod \tau_u \mid J_u(ID^*) = 0 \mod \tau_u) \\
&\geq 1 - \frac{q_{pk}}{\tau_u}.
\end{aligned}$$

Similarly, the second term of right hand side

$$\Pr\left(\bigwedge_{w \in D} J_v(w) \neq 0 \mod \tau_v \mid J_v(w^*) = 0 \mod \tau_v\right) \geq 1 - \frac{q_d}{\tau_v}.$$

Hence we have

$$\Pr(\neg E) \geq (1 - \frac{q_d}{\tau_v})(1 - \frac{q_{pk}}{\tau_u}).$$

For $c_{mode} = 0$ we obtain

$$\Pr(\neg E) = \Pr\left(\bigwedge_{ID \in PK \cup PPK} J_u(ID) \neq 0 \mod \tau_u \wedge \bigwedge_{w \in D} J_v(w) \neq 0 \mod \tau_v \mid J_u(ID^*) = 0 \mod \tau_u \wedge J_v(w^*) = 0 \mod \tau_v\right)$$

$$= \Pr\left(\bigwedge_{ID \in PK \cup PPK} J_u(ID) \neq 0 \mod \tau_u \mid J_u(ID^*) = 0 \mod \tau_u\right) \cdot \Pr\left(\bigwedge_{w \in D} J_v(w) \neq 0 \mod \tau_v \mid J_v(w^*) = 0 \mod \tau_v\right).$$

Now, handling the equation just like we did earlier we have

$$\Pr(\neg E) \geq (1 - \frac{q_d}{\tau_v})(1 - \frac{q_{pk} + q_{ppk}}{\tau_u}).$$

Putting the above results together we get

$$\Pr(\neg E) \geq \begin{cases} (1 - \frac{q_d}{\tau_v})(1 - \frac{q_{pk}}{\tau_u}) & \text{if } c_{mode} = 1 \\ (1 - \frac{q_d}{\tau_v})(1 - \frac{q_{pk} + q_{ppk}}{\tau_u}) & \text{if } c_{mode} = 0. \end{cases}$$

On setting $\tau_v = 2q_d$, $\tau_u = 2q_{pk}$ if $c_{mode} = 1$ and $\tau_u = 2(q_{pk} + q_{ppk})$ if $c_{mode} = 0$ we obtain $\Pr(\neg E) \geq 1/4$. This should be done in accordance to the specifications of τ_u and τ_v provided in *Game 2*.

As mentioned before this is just a theoretical lower bound for not aborting, to employ game hopping we need to ensure that the probability of not aborting is exactly $1/4$. We estimate the probability that a certain sequence of oracle queries made by the attacker $\mathcal{A}_I$ may cause an abort by repeatedly picking the values $x'_u, x_{u,j}, x'_v$ and $x_{v,j}$ and checking if these values cause an abort for the sequence of oracle queries that $\mathcal{A}_I$ has made. This does not require rerunning the attacker $\mathcal{A}_I$ but simply checking whether the simulator aborts as mentioned before. Also we do not constraint the values of $x'_u, x_{u,j}, x'_v$ and $x_{v,j}$. We must note that in order to have no impact on the attacker's behaviour due to these changes we have to ensure that the master public key value stays same. Hence, we may assume that y values are chosen so that master public key elements are as in the original execution of $\mathcal{A}_I$. It might appear at the outset that we need to solve the discrete logarithm problem to achieve this but looking at the definition of $x'_u, x_{u,j}, x'_v$ and $x_{v,j}$ in *Game 2* it becomes clear that this is not the case. We know that $g_2 = g^b$ hence a change in the x values can be adjusted by picking a suitable y value without the need to solve the discreet logarithm problem.

The probability that we do not abort for a given sequence of oracle queries made by $\mathcal{A}_I$ is given by η', i.e. $\Pr(\neg E) = \eta'$. We approximate the probability for η' given by the repeated sampling of the x values by η''.

Using the Chernoff bound we see that $\Pr(|\eta' - \eta''| \geq \varepsilon) \leq \delta$ when we consider $\mathcal{O}\left(\varepsilon^{-2}\ell n\delta^{-1}\right)$ samples and $\varepsilon, \delta \geq 0$. To attain a definite abort probability we force an artificial abort with probability $\dfrac{\eta'' - 1/4}{\eta''}$ whenever $\eta'' \geq 1/4$. In those cases $\mathcal{B}$ assumes that $\mathcal{A}_I$ outputs a random d'.

Now the probability of the abort can be estimated by

$$\begin{aligned}
\Pr\left(\text{Abort}\middle|\, |\eta' - \eta''| < \varepsilon\right) &= \Pr(\text{Natural Abort}) + \Pr(\text{Artificial Abort}) \\
&= (1 - \eta') + \frac{\eta'' - 1/4}{\eta''}\eta' \\
&= (1 - \eta') + (\eta'' - 1/4)\frac{\eta'}{\eta''} \\
&\leq (1 - \eta') + (\eta'' - 1/4)\frac{\eta'}{\eta' - \varepsilon} \\
&\leq (1 - \eta') + (\eta'' - 1/4)(1 + \frac{\varepsilon}{\eta' - \varepsilon}) \\
&\quad \text{as } \eta' \geq 1/4, \text{ we can estimate} \\
&\leq (1 - \eta') + (\eta'' - 1/4)(1 + \frac{4\varepsilon}{1 - 4\varepsilon}) \\
&\leq (1 - \eta') + (\eta' + \varepsilon - 1/4)(1 + 5\varepsilon), \\
&\quad \text{since } \varepsilon \leq \frac{1}{20} \text{ for sufficiently small } \varepsilon \\
&\leq (1 - \eta' + \eta' + \varepsilon - 1/4) + 5\varepsilon(\eta' - 1/4) + 5\varepsilon^2 \\
&\leq 3/4 + \varepsilon + 5\varepsilon + 5\varepsilon^2, \text{ since } (\eta' - 1/4) \leq 1 \\
&\leq 3/4 + 6\varepsilon + 5\varepsilon^2 \\
&\leq 3/4 + 7\varepsilon.
\end{aligned}$$

Now, we have

$$\begin{aligned}
\Pr(\text{Abort}) = \Pr\left(\text{Abort}\middle|\, |\eta' - \eta''| < \varepsilon\right) \cdot \Pr\left(|\eta' - \eta''| < \varepsilon\right) + \\
\Pr\left(\text{Abort}\middle|\, |\eta' - \eta''| \geq \varepsilon\right) \cdot \Pr\left(|\eta' - \eta''| \geq \varepsilon\right) \\
\leq (3/4 + 7\varepsilon) \cdot 1 + 1 \cdot \delta \\
\leq 3/4 + 7\varepsilon + \delta.
\end{aligned}$$

Therefore, the abort does not occur with a probability of at least

$$\begin{aligned}
1 - \Pr(\text{Abort}) &\geq 1 - 3/4 - 7\varepsilon - \delta \\
&\geq 1/4 - 7\varepsilon - \delta.
\end{aligned}$$

Now employing Lemma 10.10 we obtain

$$\begin{aligned} Adv_7 &\geq Adv_6\,(1/4 - 7\varepsilon - \delta) \\ &\geq (Adv_6 - \delta)\,(1/4 - 7\varepsilon) \end{aligned}$$

For $\varepsilon \leq 1/56$, we get

$$\begin{aligned} Adv_7 &\geq Adv_6\,(1/4 - 1/8) \\ &\geq \frac{Adv_6 - \delta}{8} \end{aligned}$$

Game 8 In this game we change the way we treat $\mathcal{A}_I$'s queries. Let $A = g^a$ where $a \in_R \mathbb{Z}_p^\times$ and unknown to $\mathcal{B}$. The generation of the master public key is changed, g_1 is generated depending upon the value of c_{mode}.

- If $c_{mode} = 0$ then $\mathcal{B}$ sets $g_1 = A$ without the knowledge of the master secret a.
- If $c_{mode} = 1$ then $\mathcal{B}$ sets $g_1 = g^\gamma$ with $\gamma \in_R \mathbb{Z}_p^\times$, and stores γ for later use.

We now model our response to the queries based on the value of c_{mode}.

- *Request Public Key* for an identity ID:
  - If $c_{mode} = 0$, $\mathcal{B}$ picks $x_{ID} \in_R \mathbb{Z}_p^\times$ and returns $pk_{ID} \leftarrow (g^{x_{ID}}, g_1^{x_{ID}})$
  - If $c_{mode} = 1$, $\mathcal{B}$ picks $x_{ID} \in_R \mathbb{Z}_p^\times$ and returns $pk_{ID} \leftarrow (A^{x_{ID}}, A^{\gamma x_{ID}})$
- *Replace Public Key* for an input $(ID, (\tilde{X}, \tilde{Y}))$: $\mathcal{B}$ checks if $(\tilde{X}, \tilde{Y})$ of the correct shape and then replaces the public key of ID.
- *Extract Partial Private Key* for an identity ID:
  - If $c_{mode} = 0$, $\mathcal{B}$ aborts if $J_u(ID) = 0 \mod \tau_u$ just like previous game. Otherwise it follows that $J_u(ID) \neq 0 \mod \tau_u$ and thus $J_u(ID) \neq 0 \mod p$ and $\mathcal{B}$ picks $r \in_R \mathbb{Z}_p^\times$ and returns $d_A = (d_1, d_2)$ where

$$d_1 \leftarrow F_u(ID) \cdot g_1^{-K_u(ID)/J_u(ID)}$$

and

$$d_2 \leftarrow g^r \cdot g_1^{-1/J_u(ID)},$$

this can be written as

$$\begin{aligned} d_1 &= F_u(ID) \cdot (g)^{-aK_u(ID)/J_u(ID)} \\ &= F_u(ID) \cdot (F_u(ID)^{1/K_u(ID)} \cdot \\ &\quad g_2^{-J_u(ID)/K_u(ID)})^{-aK_u(ID)/J_u(ID)}, \\ &\quad \text{since } F_u(ID) = g_2^{J_u(ID)} \cdot g^{K_u(ID)} \\ &= F_u(ID) \cdot (F_u(ID)^{-a/J_u(ID)} \cdot g_2^a) \\ &= g_2^a \cdot F_u(ID)^{\tilde{r}} \end{aligned}$$

and

$$d_2 = g^r \cdot (g^a)^{-1/J_u(ID)} = g^{\tilde{r}},$$

where $\tilde{r} = r - \dfrac{a}{J_u(ID)}$.

– If $c_{mode} = 1$, $\mathcal{B}$ uses $msk = \gamma$ to calculate the partial private key using the construction.

∘ *Extract Private Key* for an identity ID: $\mathcal{B}$ aborts if $J_u(ID) = 0 \mod \tau_u$ just like previous game. Otherwise it follows that $J_u(ID) \neq 0 \mod \tau_u$ and thus $J_u(ID) \neq 0 \mod p$. Let $pk_{ID} = (X, Y)$ be the original public key for ID, $\mathcal{B}$ picks $t \in_R \mathbb{Z}_p^\times$ and returns $sk_{ID} = (s_1, s_2)$ where

$$(s_1, s_2) = \left(F_u(ID)^t \cdot Y^{-K_u(ID)/J_u(ID)}, g^t \cdot Y^{-1/J_u(ID)}\right).$$

Now, using $F_u(ID) = g_2^{J_u(ID)} \cdot g^{K_u(ID)}$ we have

– If $c_{mode} = 0$, then the secret value is x_{ID} and implicitly defined master key value is a, hence $(s_1, s_2) = \left(g_2^{ax_{ID}} \cdot F_u(ID)^{\tilde{t}}, g^{\tilde{t}}\right)$ where $\tilde{t} = t - \dfrac{ax_{ID}}{J_u(ID)}$.

– If $c_{mode} = 1$, then the implicitly defined secret value is ax_{ID} and the master key value is γ, hence $(s_1, s_2) = \left(g_2^{a\gamma x_{ID}} \cdot F_u(ID)^{\tilde{t}}, g^{\tilde{t}}\right)$ where $\tilde{t} = t - \dfrac{a\gamma x_{ID}}{J_u(ID)}$.

∘ *Decrypt* a valid ciphertext $C = (C_0, C_1, C_2, C_3)$ encrypted for an identity ID using the public key $pk_{ID} = (X, Y)$ which may or may not have been replaced by the attacker: Let $w = (C_0, C_1, C_2, ID, pk_{ID})$, $\mathcal{B}$ aborts if $J_v(w) = 0 \mod \tau_v$ just like the previous game and chooses $d' \in_R \{0, 1\}$. Otherwise it follows that $J_v(w) \neq 0 \mod \tau_v$ hence $J_v(w) \neq 0 \mod p$ and $C_3 = \left(g_2^{J_v(w)} g^{K_v(w)}\right)^s$ and $C_1 = g^s$ where $s \in_R \mathbb{Z}_p^\times$.

Now, $\mathcal{B}$ extracts

$$g_2^s = \left(\frac{C_3}{C_1^{K_v(w)}}\right)^{1/J_v(w)}$$

and computes $e(Y, g_2)^s$, this allows for the computation of $m = \dfrac{C_0}{e(Y, g_2)^s}$ regardless of the fact whether (X, Y) is the original public key or not.

Changing the generation of the master key does has no effect on $\mathcal{B}$'s ability to answer $\mathcal{A}_I$'s queries like in *Game 7* and the distribution of the master key remains unchanged, hence we have $Adv_8 = Adv_7$.

Game 9 In this game we modify the generation of the ciphertext again. Using variables $b, c \in_R \mathbb{Z}_p^\times$ defined in *Game 2* and *Game 6* respectively. We set $C_1^* = g^c$ and $T = A^{bc}$.

- If $c_{mode} = 0$, let $pk_{ID^*} = (X^*, Y^*)$ be identity ID^*'s current public key. $\mathcal{B}$ flips a binary coin $d^* \in_R \{0,1\}$ and computes

$$\begin{aligned} C_0^* &= m_{d^*} \cdot e(X^*, T) \\ &= m_{d^*} \cdot e(g^{x^*}, g^{abc}) \\ &= m_{d^*} \cdot e(g^{ax^*}, g^{bc}) \\ &= m_{d^*} \cdot e(Y^*, g_2)^c. \end{aligned} \tag{10.13}$$

  It then computes $C_2^* = (g^c)^{K_u(ID^*)}$, $w^* = H(C_0^*, C_1^*, C_2^*, ID^*, pk_{ID^*})$ and $C_3^* = (g^c)^{K_v(w^*)}$. If $J_v(w^*) \neq 0 \mod p$ or $J_u(ID^*) \neq 0 \mod p$ then $\mathcal{B}$ aborts like in *Game 5* otherwise it returns $(C_0^*, C_1^*, C_2^*, C_3^*)$.

- If $c_{mode} = 1$, $\mathcal{B}$ retrieves x_{ID^*} such that $pk_{ID^*} = (A^{x_{ID^*}}, A^{\gamma x_{ID^*}})$, flips a binary coin $d^* \in_R \{0,1\}$ and computes

$$\begin{aligned} C_0^* &= m_{d^*} \cdot e(Y^*, g_2)^c \\ &= m_{d^*} \cdot e(A^{\gamma x_{ID^*}}, g^b)^c \\ &= m_{d^*} \cdot e(g^{a\gamma x_{ID^*}}, g^{bc}) \\ &= m_{d^*} \cdot e(g, g^{abc})^{\gamma x_{ID^*}} \\ &= m_{d^*} \cdot e(g, T)^{\gamma x_{ID^*}} \end{aligned} \tag{10.14}$$

  It then computes $C_2^* = (g^c)^{K_u(ID^*)}$, $w^* = H(C_0^*, C_1^*, C_2^*, ID^*, pk_{ID^*})$ and $C_3^* = (g^c)^{K_v(w^*)}$. If $J_v(w^*) \neq 0 \mod p$ then $\mathcal{B}$ aborts like in *Game 5* otherwise it returns $(C_0^*, C_1^*, C_2^*, C_3^*)$.

Since we have $J_v(w^*) = 0 \mod p$, these changes do not affect the distribution of the challenge ciphertext and we have $Adv_9 = Adv_8$.

Game 10 In this game we change the challenge phase again. $\mathcal{B}$ only retains $g_2 = g^b$ and $C_1^* = g^c$ and forgets the values b, c. Challenge is constructed as shown in equations (10.13) and (10.14) but T is chosen randomly, $T \in_R \mathbb{G}$. The simulator only uses the values g^a, g^b, g^c and never touches a, b, c. The transition between *Game 9* and *Game 10* is based upon the indistinguishability of $T = g^{abc}$ from $T \in_R \mathbb{G}$ and both games are equal unless there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ which can tell the difference between the two values. This is clearly an instance of the 3-DDH which we wanted to achieve from the very beginning. Since the only difference between *Game 9* and *Game 10* is the condition enforced by the indistinguishability of $T = g^{abc}$ from $T \in_R \mathbb{G}$ hence the difference of the

success probabilities of *Game 9* and *Game 10* is the lower bound on $Adv^{3\text{-}DDH}_{\mathcal{A}'}(k)$. Therefore we have

$$|\Pr(S_9) - \Pr(S_{10})| \leq Adv^{3\text{-}DDH}_{\mathcal{A}'}(k)$$

Additionally C_0^* now reveals no information about m_{d^*} and is completely independent of d^*, hence $\Pr(S_{10}) = 1/2$. This brings game hopping to an end.

We now combine the results obtained from the previous games. We have

$$Adv_7 = Adv_8 = Adv_9 \leq Adv^{3\text{-}DDH}_{\mathcal{A}'}(k)$$

and

$$Adv_5 = Adv_6 \leq 8 \cdot Adv_7 + \delta$$

also

$$Adv_5 = \frac{Adv_4}{\tau_u \tau_v (n+1)^2}$$

where $\tau_u \leq 2(q_{ppk} + q_{pk})$ and $\tau_v = 2q_d$. And thus

$$\begin{aligned} Adv_4 &\leq 2(q_{ppk} + q_{pk}) \cdot 2q_d(n+1)^2 \cdot Adv_5 \\ &\leq 4q_d(q_{ppk} + q_{pk})(n+1)^2 \cdot (8 \cdot Adv_7 + \delta). \end{aligned}$$

We also have

$$Adv_3 = 2 \cdot Adv_4$$

and

$$\begin{aligned} Adv_1 = Adv_2 &= |\Pr(S_2) - 1/2| \\ &\leq |\Pr(S_2) - \Pr(S_3)| + |\Pr(S_3) - 1/2| \\ &\leq Adv^{CR}_{\mathcal{A}''}(k) + Adv_3. \end{aligned}$$

Combining the above equations we finally have

$$Adv_1 < 8q_d(q_{ppk} + q_{pk})(n+1)^2 \cdot (8 \cdot Adv^{3\text{-}DDH}_{\mathcal{A}'}(k) + \delta) + Adv^{CR}_{\mathcal{A}''}(k). \qquad \square$$

PROOF (Theorem 10.9). The proof for Theorem 10.9 is similar to the proof of Theorem 10.8 with differences in *Games 7, 8, 9* and *10*. We note that $\mathcal{A}_{II}$ never makes a partial private key query and has the access to the master secret key $msk = \gamma$ at the beginning of the game.

In *Game 7* and *Game 8*, $\mathcal{B}$ treats all queries as in case of $c_{mode} = 1$ and hands $msk = \gamma$ to $\mathcal{A}_{II}$. In *Game 9* all queries are handled as in the case of $c_{mode} = 1$ and the challenge ciphertext is computed using (10.14). Finally, in *Game 10* we combine all the results according to the changes described in *Game 8* and *9* to obtain the final equation. $\square$

Discussion The proofs of Theorem 10.8 and Theorem 10.9 show that the scheme is secure against the adversaries described in Section 9.2. Thus in a manner of speaking breaking the scheme under the imposed constraints would be equivalent to solving the 3-DDH problem and breaking the collision resistant hash function H. These are hard problems and basing our security on them gives an idea of the hardness of breaking the indistinguishability of the scheme.

At this point we should note that at the onset of the games we assume the knowledge of the number of oracle queries the attacker makes, we denote this by q_d, q_{ppk} and q_{pk}. This might not be possible in general hence to achieve this we make an assumption about the number of queries made. If our assumption seems too small or too large then we adjust the values accordingly and try again. For the purpose of our proofs we start with the correct assumed values. These proofs also show that certificateless schemes can be constructed that are secure in the standard model. We demonstrate how this construction can be implemented efficiently in Section 12.4.

11. Concrete Derived Construction

11.1. The Construction. The new encryption scheme is derived from the scheme previously described in Section 10. We take the originally proposed construction and modify it to include biometrics. As discussed earlier the motivation behind doing so is to achieve an encryption scheme which provides two-factor security. To encrypt messages the sender uses the receiver's public key as well as biometrics and decryption of messages requires the receiver to provide both the private key and his biometrics. This in turn safe guards the receiver against device compromise. Since the proposed scheme is an extension of the one discussed earlier, it also uses *bilinear map groups* and we again require the 3-DDH for $\mathbb{G}$ described in Definition 4.3 to be intractable in such groups.

The modified scheme proceeds through a sequence of six subroutines which are described in the pages to follow.

ALGORITHM 11.1. `Setup`.

Performed by the KGC — this is the first step in the encryption scheme in this step the KGC generates the *master public key* mpk and *master secret key* msk after receiving the system security parameter k and bit length n of the *public identity* ID. Let $\mathbb{G}$ define bilinear map group of order $p > 2^k$ and g be a generator for $\mathbb{G}$.

Input: $(1^k, n)$.

Output: (mpk, msk).

1. Choose $\gamma \xleftarrow{R} \mathbb{Z}_p^*$.
2. Set $g_1 = g^\gamma$.
3. Choose $g_2 \xleftarrow{R} \mathbb{G}$.
4. Choose vectors $(h', h_1, \ldots, h_n)$, $(u', u_1, \ldots, u_n)$, $(v', v_1, \ldots, v_n) \xleftarrow{R} \mathbb{G}^{n+1}$.
5. Write $BID = k_1 k_2 \ldots k_n$, $ID = i_1 i_2 \ldots i_n$ and $w = w_1 w_2 \ldots w_n$ as bit strings with $k_j, i_j, w_j \in \{0,1\}$.
6. Define hash functions

$$F_h : \begin{array}{ccc} \{0,1\}^n & \longrightarrow & \mathbb{G}, \\ BID & \longmapsto & h' \prod_{0 \le j \le n} h_j^{k_j} \end{array},$$

$$F_u : \begin{array}{ccc} \{0,1\}^n & \longrightarrow & \mathbb{G}, \\ ID & \longmapsto & u' \prod_{0 \le j \le n} u_j^{i_j} \end{array}$$

and

$$F_v : \begin{array}{ccc} \{0,1\}^n & \longrightarrow & \mathbb{G}, \\ w & \longmapsto & v' \prod_{0 \le j \le n} v_j^{w_j} \end{array}.$$

7. Choose a collision resistant hash function $H : \{0,1\}^* \to \{0,1\}^n$ as required in Definition 4.4.
8. Define the *master public key* as
$mpk \leftarrow (g, g_1, g_2, h', h_1, \ldots, h_n, u', u_1, \ldots, u_n, v', v_1, \ldots, v_n)$.

9. Define the *master secret key* as $msk \leftarrow \gamma$.

ALGORITHM 11.2. `Extract`.
Performed by the KGC — this step is executed after the *Receiver* authenticates himself as ID to the KGC and then securely communicates $F_h(BID)$. The KGC then proceeds to compute the *partial private key* d_{ID} which is subsequently communicated to the *Receiver* in a secure manner.
Input: $(mpk, msk, ID, F_h(BID))$.
Output: d_{ID}.

1. Choose $r \xleftarrow{R} \mathbb{Z}_p^{\times}$.
2. Compute $d_{ID} \leftarrow (d_1, d_2, d_3) = (g_2^{\gamma} \cdot F_u(ID)^r, (g \cdot F_h(BID))^r, F_h(BID)^{\gamma})$.
3. Return d_{ID}.

ALGORITHM 11.3. `SetSec`.
Performed by the Receiver — in this step the *Receiver* computes a randomly chosen secret value x_{ID}.
Input: mpk.
Output: x_{ID}.

1. Choose $x_{ID} \xleftarrow{R} \mathbb{Z}_p^{\times}$.
2. Return x_{ID}.

ALGORITHM 11.4. `SetPub`.
Performed by the Receiver — in this step the *Receiver* computes and freely distributes the *public key* pk_{ID}.
Input: (x_{ID}, mpk, d_{ID}).
Output: pk_{ID}.

1. Compute $pk_{ID} \leftarrow (X, Y, Z) = (g^{x_{ID}}, g_1^{x_{ID}}, d_3^{x_{ID}})$.
2. Return pk_{ID}.

ALGORITHM 11.5. `Encrypt`.
Performed by the Sender — this step computes the encryption of message $m \in \mathbb{G}_T$.
Input: $(m, pk_{ID}, mpk, ID, BID)$.
Output: C.

1. If $e(X, g_1)/e(g, Y) = 1_{\mathbb{G}_T}$ then
2. Choose $s \xleftarrow{R} \mathbb{Z}_p^{\times}$.
3. Set $(C_0, C_1, C_2) \leftarrow (m \cdot e(Y \cdot Z, g_2)^s, (g \cdot F_h(BID))^s, F_u(ID)^s)$.
4. Compute $w \leftarrow H(C_0, C_1, C_2, ID, pk_{ID})$.
5. Set $C_3 \leftarrow F_v(w)^s$.

6. Return $C = (C_0, C_1, C_2, C_3)$.
7. Else
8. Return FAIL

Just like Step 1 in Algorithm 10.6, in Step 1 in Algorithm 11.5 we check for the correctness of the public key pk_{ID}, if pk_{ID} is of the right shape then the ciphertext C is computed and returned else the algorithm aborts with FAIL.
We include the hash $F_v(w)$ of w in Step 4 in Algorithm 11.5 for the reasons previously described in Step 4 in Algorithm 10.6.

ALGORITHM 11.6. `Decrypt`.
Performed by the Receiver — this decrypts the message encrypted using *Receiver*'s private key pk_{ID}, public identity ID and the biometric identity BID.
Input: $(C, d_{ID}, x_{ID}, mpk, ID, BID)$.
Output: m.

1. Set $(C_0, C_1, C_2, C_3) \leftarrow C$.
2. Let $w \leftarrow H(C_0, C_1, C_2, ID, pk_{ID})$.
3. If $e(C_1, F_u(ID) \cdot F_v(w)) = e(g \cdot F_h(BID), C_2 \cdot C_3)$ then
4. Choose $r' \xleftarrow{R} \mathbb{Z}_p^{\times}$.
5. Set $(d_1, d_2) \leftarrow d_{ID}$.
6. Compute the private key as

$$sk_{ID} \leftarrow (s_1, s_2) = \left(d_1^{x_{ID}} \cdot F_u(ID)^{r'}, d_2^{x_{ID}} \cdot (g \cdot F_h(BID))^{r'}\right).$$

7. Compute $m \leftarrow C_0 \cdot \dfrac{e(C_2, s_2)}{e(C_1, s_1)}$.
8. Delete sk_{ID}.
9. Return m.
10. Else
11. Return FAIL

In Step 3 in Algorithm 11.6 we check for the validity of the ciphertext by checking the hash $F_v(w)$, if C is a valid ciphertext then we proceed with the decryption else the algorithm aborts with FAIL.

We check for completeness by substituting the values for $(C_0, C_1, C_2, s_1, s_2)$

in

$$(s_1, s_2) = \Big((g_2^{\gamma} \cdot F_u(ID)^r)^{x_{ID}} \cdot F_u(ID)^{r'}, \\ ((g \cdot F_h(BID))^r)^{x_{ID}} \cdot (g \cdot F_h(BID))^{r'} \Big)$$

$$= \left(g_2^{\gamma x_{ID}} \cdot F_u(ID)^{r x_{ID} + r'}, (g \cdot F_h(BID))^{r x_{ID} + r'} \right)$$

$$= \left(g_2^{\gamma x_{ID}} \cdot F_u(ID)^t, (g \cdot F_h(BID))^t \right)$$

where $t = r x_{ID} + r'$. Now, substituting the values we obtain

$$C_0 \cdot \frac{e(C_2, s_2)}{e(C_1, s_1)} = m \cdot e(Y \cdot Z, g_2)^s \cdot \frac{e\left(F_u(ID)^s, (g \cdot F_h(BID))^t\right)}{e\left((g \cdot F_h(BID))^s, g_2^{\gamma x_{ID}} \cdot F_u(ID)^t\right)}$$

$$= m \cdot e\left(g_1^{x_{ID}} \cdot d_3^{x_{ID}}, g_2\right)^s \cdot \\ \frac{e\left(F_u(ID)^s, (g \cdot F_h(BID))^t\right)}{e\left((g \cdot F_h(BID))^s, g_2^{\gamma x_{ID}}\right) \cdot e\left((g \cdot F_h(BID))^s, F_u(ID)^t\right)}$$

$$= m \cdot \frac{e\left((g \cdot F_h(BID))^{\gamma x_{ID}}, g_2^s\right)}{e\left((g \cdot F_h(BID))^s, g_2^{\gamma x_{ID}}\right)}$$

$$= m.$$

Thus we conclude that the decryption of an encrypted message gives us back the original message m and the scheme functions correctly.

11.2. Security Reduction. The security reduction of the derived scheme proceeds very similar to that of the original scheme presented in Section 10.2. The security of the derived construction is also based on intractablility of 3-DDH in the groups that are used by the construction. We again define a theorem to capture the idea of security in the new model and use games to prove it.

THEOREM 11.7. *Suppose $\mathcal{A}^{new}$ is either a New Strong Type I or New Strong Type II adversary that runs in time t, makes at most q_d decryption queries, q_{bid} biometric identity extraction queries, and q_{pk} private key queries. Then there exists*

- *an adversary $\mathcal{A}'$ against the 3-DDH problem that has advantage $Adv_{\mathcal{A}'}^{3\text{-}DDH}(k)$ and runs in time $\mathcal{O}(t) + \mathcal{O}\left(\varepsilon^{-2} \ln \delta^{-1}\right)$ for sufficiently small ε and δ, and*
- *an adversary $\mathcal{A}''$ against the collision resistance of the hash function H that runs in time $\mathcal{O}(t)$ and has advantage $Adv_{\mathcal{A}''}^{CR}(k)$*

such that the advantage $Adv^{CL\text{-}CCA}_{\mathcal{A}^{new}}(k)$ *of* $\mathcal{A}^{new}$ *is bounded by*

$$Adv^{CL\text{-}CCA}_{\mathcal{A}^{new}}(k) < 8q_{pk}q_dq_{bid}(n+1)^3 \cdot (16 \cdot Adv^{3\text{-}DDH}_{\mathcal{A}'}(k) + \delta) + Adv^{CR}_{\mathcal{A}''}(k).$$

Interpretation The advantage of the adversaries against the scheme presented in Section 11 is bound by the results of Theorem 11.7. Thus if either $\mathcal{A}^{new}_{I}$ or $\mathcal{A}^{new}_{II}$ exist such that it has a significant advantage against breaking the indistinguishability of the scheme then this would imply the existence of attackers with significant advantage against both solving the 3-DDH and breaking the collision resistant hash function H. Such attackers can be used to device algorithms which solve the 3-DDH and find collisions of H in polynomial time.

For the algorithms mentioned in the above theorems a solution can be conveniently found for which $\varepsilon \approx 0$ or $\delta \approx 0$ but such a solution provides no advantage to the adversary as it is too slow and the equation is rendered meaningless. To gain significant advantage in breaking the security of the scheme the adversary must find an algorithm with suitable ε and δ such that the algorithm runs in polynomial time.

Framework The framework is exactly similar to what was earlier presented in Section 10.2 and we again want to use the attacker to solve the 3-DDH. Like earlier, we change the original protocol by introducing changes which are undetectable by the attacker. Finally, we present the attacker with the 3-DDH modelled as our challenge and try to make conclusions about the attacker winning the IND-CCA game. Thus in this modified scenario if he manages to win the IND-CCA game with a significant advantage then we can conclude that such an attacker can solve the 3-DDH. Thus we show that the security of our scheme rests on the intractability of the 3-DDH in $\mathbb{G}$. We present the changes to the protocol in the form of games that we describe subsequently. The arrangement of the security reduction also remains same as shown in Figure 10.1.

PROOF (Theorem 11.7). The proof proceeds just like the proof of Theorem 10.8 through a sequence of games which involves the New Type I attacker $\mathcal{A}^{new}_{I}$ who tries to guess the hidden bit d in the IND-CCA game. The attacker outputs a guess d' on conclusion of the sequence of games. Here we highlight the modifications that we make to the proof of Theorem 10.8 to prove the given theorem.

Game 1 This game is same as the *Game 1* defined in the the proof of Theorem 10.8 but we define one further quantity here. Let $BE = \{ID''_1, \ldots, ID''_{q_{bid}}\}$ be the set of identities for which the attacker makes biometric identity extraction queries. BID^* denotes the biometric identity of the target identity ID^*. Also, PPK defined in *Game 1* of the proof of Theorem 10.8 is never used.

Game 2 In addition to the values defined earlier in *Game 2* of the proof of Theorem 10.8, we select $\kappa_h \in \{0, \ldots, n\}$ and let τ_h be an integer such that $\tau_h(n+1) < p$. The environment selects $x'_h \in_R \mathbb{N}_{<\tau_h}$ and the vector $(x_{h,1}, \ldots, x_{h,n}) \in_R \mathbb{N}^n_{<\tau_h}$. It also picks $y'_h \in_R \mathbb{Z}_p$ and vector $(y_{h,1}, \ldots, y_{h,n}) \in_R \mathbb{Z}^n_p$. The remaining *master public key* elements are chosen as follows:

$$h' = g_2^{x'_h - \kappa_h \tau_h} g^{y_{q'}}, \quad h_j = g_2^{x_{h,j}} g^{y_{h,j}}. \quad \text{for } 0 \leq j \leq n \tag{11.8}$$

Again, the distribution of the *master public key* remains unchanged due to the reasons discussed earlier in *Game 2* of the proof of Theorem 10.8. Thus we have $\Pr(S_1) = \Pr(S_2)$ which implies $Adv_1 = Adv_2$.

Game 3 Same as *Game 3* of the proof of Theorem 10.8.

Game 4 In our modified security model $\mathcal{A}_I^{new}$ is allowed to make partial private key queries as well as replace the public key of the same identity even in case of ID^* hence we do not require to differentiate between the two cases like earlier done using c_{mode}. In this game we do nothing but for the ease of understanding and similarity between other games we just leave it as a place holder and we have $Adv_4 = Adv_3$.

Game 5 In addition to the values of $F_u(ID)$ and $F_v(w)$ redefined in *Game 5* of the proof of Theorem 10.8, in this game we redefine $J_h(BID)$ before modifying *Game 4*. This is done with the help of specific choice of the values $(x_{h,j}, y_{h,j})$ from (11.8). The game proceeds similar to *Game 5* of Theorem 10.8. To this end we define

$$J_h(BID) = x'_h + \sum_{j=1}^{n} i_j x_{h,j} - \kappa_h \tau_h, \qquad K_h(BID) = y'_h + \sum_{j=1}^{n} i_j y_{h,j},$$

where $BID = k_1 \ldots k_n$ is a n-bit string. For any string $BID \in \{0,1\}^n$ we define

$$F_h(BID) = h' \cdot \prod_{j=1}^{n} h_j^{i_j}.$$

We can rewrite this as

$$F_h(BID) = g_2^{J_h(BID)} \cdot g^{K_h(BID)}.$$

Game 5 is identical to *Game 4* except when the attacker $\mathcal{A}_I^{new}$ outputs its guess d' of d then the environment $\mathcal{B}$ checks whether $J_u(ID^*) = J_v(w^*) = J_h(BID^*) = 0 \bmod p$. If $J_u(ID^*) \neq 0$ or $J_v(w^*) \neq 0$ or $J_h(BID^*) \neq 0$ then $\mathcal{B}$ aborts and simulates $\mathcal{A}_I^{new}$'s output choosing uniformly randomly $d' \in_R \{0,1\}$. Again we have $\Pr(J_h(BID^*) = 0 \mod p) = \dfrac{1}{\tau_h(n+1)}$, since $J_h(BID^*) = 0$

purely by chance. The reasons for this are the same as those that have been previously discussed in *Game 5* of the proof of Theorem 10.8. Hence, we have $Adv_5 = \dfrac{Adv_4}{\tau_u \tau_v \tau_h (n+1)^3}$.

Game 6 In this game we modify the way the environment $\mathcal{B}$ generates the challenge ciphertext. $\mathcal{B}$ picks up a random value $c \in_R \mathbb{Z}_p^{\times}$ and sets

$$\begin{aligned} C_1^* &= (g \cdot F_h(BID^*))^c \\ &= g^{c(K_h(BID^*)+1)}. \end{aligned}$$

Let identity ID^*'s public key at the challenge phase be denoted by $pk_{ID^*} = (X^*, Y^*, Z^*)$. $\mathcal{B}$ flips a coin $d^* \in_R \{0,1\}$ and computes

$$\begin{aligned} C_0^* &= m_{d^*} \cdot e(Y^* \cdot Z^*, g_2)^c, \\ C_2^* &= C_1^{*K_u(ID^*)/(K_h(BID^*)+1)} = (g^c)^{K_u(ID^*)} \end{aligned}$$

and

$$C_3^* = C_1^{*K_v(w^*)/(K_h(BID^*)+1)} = (g^c)^{K_v(w^*)}$$

where $w^* = H(C_0^*, C_1^*, C_2^*, ID^*, pk_{ID^*})$. The returned ciphertext $(C_0^*, C_1^*, C_2^*, C_3^*)$ has the correct distribution since $J_u(ID^*) = J_v(w^*) = J_h(BID^*) = 0$ and hence we have $Adv_6 = Adv_5$.

Game 7 In this game we modify *Game 6* such that after $\mathcal{A}_I^{new}$ outputs his guess d' the environment $\mathcal{B}$ checks if one of the conditions described below are true. We redefine the event E previously defined in *Game 7* of the proof of Theorem 10.8 as follows

- $J_u(ID_j) = 0 \mod \tau_u$ for some $ID_j \in PK$ with $j \in \{1, \ldots, q_{pk}\}$.
- $J_v(w_\ell) = 0 \mod \tau_v$ for some $w_\ell \in D$ with $\ell \in \{1, \ldots, q_d\}$.
- $J_h(BID_k) = 0 \mod \tau_h$ for some $ID_k \in BE$ with $k \in \{1, \ldots, q_{bid}\}$ where BID_k is the biometric identity corresponding to ID_k.

We define E as the event that any of the aforementioned conditions hold. Just like before we observe that Dent's game hopping technique cannot be applied at this stage since even though E is recognisable there is no surety that it is independent of S_6. Attacker $\mathcal{A}_I^{new}$ can model his queries by choosing PK and BE depending upon m_d in such a way that $\Pr(E)$ is significantly different in different query sequences. Hence, we again use re-normalisation technique suggested in Waters (2005) to circumvent this problem. We derive a non-negligible lower bound for $\Pr(\neg E)$ for any set of oracle queries. We estimate the probability that E occurs during a particular set of oracle queries that are made while running $\mathcal{A}_I^{new}$ and then

add *artificial aborts* to ensure that $\mathcal{A}_I^{new}$ aborts with exactly the probability given by this lower bound. We now derive the theoretical lower bound.

$$\Pr(\neg E) = \Pr\Bigg(\bigwedge_{ID\in PK} J_u(ID) \neq 0 \mod \tau_u \wedge \bigwedge_{ID\in BE} J_h(BID) \neq 0 \mod \tau_h \wedge \bigwedge_{w\in D} J_v(w) \neq 0 \mod \tau_v \mid J_u(ID^*) = 0 \mod \tau_u \wedge J_h(BID^*) = 0 \mod \tau_h \wedge J_v(w^*) = 0 \mod \tau_v\Bigg)$$

$$= \Pr\Bigg(\bigwedge_{ID\in PK} J_u(ID) \neq 0 \mod \tau_u \mid J_u(ID^*) = 0 \mod \tau_u\Bigg) \cdot \Pr\Bigg(\bigwedge_{ID\in BE} J_h(BID) \neq 0 \mod \tau_h \mid J_h(BID^*) = 0 \mod \tau_h\Bigg) \cdot \Pr\Bigg(\bigwedge_{w\in D} J_v(w) \neq 0 \mod \tau_v \mid J_v(w^*) = 0 \mod \tau_v\Bigg).$$

Following the steps like we did in *Game 7* of the proof of Theorem 10.8 leads us to

$$\Pr(\neg E) \geq (1 - \frac{q_{pk}}{\tau_u})(1 - \frac{q_d}{\tau_v})(1 - \frac{q_{bid}}{\tau_h}).$$

On setting $\tau_u = 2q_{pk}$, $\tau_v = 2q_d$ and $\tau_h = 2q_{bid}$ we obtain $\Pr(\neg E) \geq 1/8$. This should be done in accordance to the specifications of τ_u, τ_v and τ_h provided in *Game 2*. Since this is a theoretical lower bound for not aborting, to employ game hopping we need to ensure that the probability of not aborting is exactly $1/8$.

We estimate the probability that a certain sequence of oracle queries made by the attacker $\mathcal{A}_I^{new}$ may cause an abort by repeatedly picking the values $x'_u, x_{u,j}, x'_v, x_{v,j}, x'_h$ and $x_{h,j}$ and checking if these values cause an abort for the sequence of oracle queries that $\mathcal{A}_I^{new}$ has made. This does not require rerunning the attacker $\mathcal{A}_I^{new}$ but simply checking whether the simulator aborts as mentioned before. Also we do not constraint the values of $x'_u, x_{u,j}, x'_v, x_{v,j}, x'_h$ and $x_{h,j}$. We must note that in order to have no impact on the attacker's behaviour due to these changes we have to ensure that the master public key value stays same. Hence, we may assume that y values are chosen so that master public key elements are as in the original execution of $\mathcal{A}_I^{new}$. It might appear at the outset that we need to solve the discrete logarithm problem to achieve this but looking at the definition of $x'_u, x_{u,j}, x'_v, x_{v,j}, x'_h$ and $x_{h,j}$ in *Game 2* it becomes clear that this is not the case. We know that $g_2 = g^b$ hence a change in the x values can be adjusted by picking a suitable y value without the need to solve the discreet logarithm problem.

The probability that we do not abort for a given sequence of oracle queries made by $\mathcal{A}_I^{new}$ is given by η', i.e. $\Pr(\neg E) = \eta'$. As suggested in *Game 7* of the proof of Theorem 10.8, we approximate the probability for η' given by the repeated sampling of the x values by η''.

Using the Chernoff bound we see that $\Pr(|\eta' - \eta''| \geq \varepsilon) \leq \delta$ when we consider $\mathcal{O}\left(\varepsilon^{-2}\ell n\delta^{-1}\right)$ samples and $\varepsilon, \delta \geq 0$. To attain a definite abort probability we force an artificial abort with probability $\dfrac{\eta'' - 1/8}{\eta''}$ whenever $\eta'' \geq 1/8$. In those cases $\mathcal{B}$ assumes that $\mathcal{A}_I^{new}$ outputs a random d'.

Now the probability of the abort can be estimated by

$$\begin{aligned}
\Pr\left(\text{Abort}\middle|\, |\eta' - \eta''| < \varepsilon\right) &= \Pr(\text{Natural Abort}) + \Pr(\text{Artificial Abort}) \\
&= (1 - \eta') + \frac{\eta'' - 1/8}{\eta''}\eta' \\
&= (1 - \eta') + (\eta'' - 1/8)\frac{\eta'}{\eta''} \\
&\leq (1 - \eta') + (\eta'' - 1/8)\frac{\eta'}{\eta' - \varepsilon} \\
&\leq (1 - \eta') + (\eta'' - 1/8)(1 + \frac{\varepsilon}{\eta' - \varepsilon}) \\
&\quad \text{as } \eta' \geq 1/8, \text{ we can estimate} \\
&\leq (1 - \eta') + (\eta'' - 1/8)(1 + \frac{8\varepsilon}{1 - 8\varepsilon}) \\
&\leq (1 - \eta') + (\eta' + \varepsilon - 1/8)(1 + 10\varepsilon), \\
&\quad \text{since } \varepsilon \leq \frac{1}{40} \text{ for sufficiently small } \varepsilon \\
&\leq (1 - \eta' + \eta' + \varepsilon - 1/8) + 10\varepsilon(\eta' - 1/8) + 10\varepsilon^2 \\
&\leq 7/8 + \varepsilon + 10\varepsilon + 10\varepsilon^2, \text{ since } (\eta' - 1/8) \leq 1 \\
&\leq 7/8 + 11\varepsilon + 10\varepsilon^2 \\
&\leq 7/8 + 12\varepsilon.
\end{aligned}$$

Now, we have

$$\begin{aligned}
\Pr(\text{Abort}) = \Pr\left(\text{Abort}\middle|\, |\eta' - \eta''| < \varepsilon\right) \cdot \Pr\left(|\eta' - \eta''| < \varepsilon\right) + \\
\Pr\left(\text{Abort}\middle|\, |\eta' - \eta''| \geq \varepsilon\right) \cdot \Pr\left(|\eta' - \eta''| \geq \varepsilon\right) \\
\leq (7/8 + 12\varepsilon) \cdot 1 + 1 \cdot \delta \\
\leq 7/8 + 12\varepsilon + \delta.
\end{aligned}$$

Therefore, the abort does not occur with a probability of at least

$$\begin{aligned}
1 - \Pr(\text{Abort}) &\geq 1 - 7/8 - 12\varepsilon - \delta \\
&\geq 1/8 - 12\varepsilon - \delta.
\end{aligned}$$

Now employing Lemma 10.10 we obtain

$$\begin{aligned} Adv_7 &\geq Adv_6\,(1/8 - 12\varepsilon - \delta) \\ &\geq (Adv_6 - \delta)\,(1/8 - 12\varepsilon) \end{aligned}$$

For $\varepsilon \leq 1/192$, we get

$$\begin{aligned} Adv_7 &\geq Adv_6\,(1/8 - 1/16) \\ &\geq \frac{Adv_6 - \delta}{16} \end{aligned}$$

Game 8 Just as shown in *Game 8* of the proof of Theorem 10.8, in this game we change the way we treat $\mathcal{A}_I^{new}$'s queries. Let $A = g^a$ where $a \in_R \mathbb{Z}_p^\times$ and unknown to $\mathcal{B}$. The generation of the master public key is done as follows, $\mathcal{B}$ sets $g_1 = g^\gamma$ where $\gamma \in_R \mathbb{Z}_p^\times$ and stores γ for later use.

We now model our response to the queries.

- *Replace Public Key* for an input $(ID, (\tilde{X}, \tilde{Y}))$: $\mathcal{B}$ checks if $(\tilde{X}, \tilde{Y})$ of the correct shape and then replaces the public key of ID.
- *Extract Partial Private Key* for an identity ID:
  - $\mathcal{B}$ uses $msk = \gamma$ to calculate the partial private key using the construction.
- *Request Public Key* for an identity ID:
  - $\mathcal{B}$ picks $x_{ID} \in_R \mathbb{Z}_p^\times$ and returns $pk_{ID} \leftarrow (A^{x_{ID}}, A^{\gamma x_{ID}}, d_3^{x_{ID}})$.
- *Extract Private Key* for an identity ID: $\mathcal{B}$ aborts if $J_u(ID) = 0 \mod \tau_u$ just like previous game. Otherwise it follows that $J_u(ID) \neq 0 \mod \tau_u$ and thus $J_u(ID) \neq 0 \mod p$. Let $pk_{ID} = (X, Y)$ be the original public key for ID, $\mathcal{B}$ picks $t \in_R \mathbb{Z}_p^\times$ and returns $sk_{ID} = (s_1, s_2)$ where

  $$(s_1, s_2) = (F_u(ID)^t \cdot Y^{-K_u(ID)/J_u(ID)}, (g \cdot F_h(BID))^t \cdot Y^{-1/J_u(ID)}).$$

  Now, using $F_u(ID) = g_2^{J_u(ID)} \cdot g^{K_u(ID)}$ we have
  - Implicitly defined secret value is ax_{ID} and the master key value is γ, hence

    $$(s_1, s_2) = (g_2^{a\gamma x_{ID}} \cdot F_u(ID)^{\tilde{t}}, g^{\tilde{t}} \cdot F_h(BID)^t) \text{ where } \tilde{t} = t - \frac{a\gamma x_{ID}}{J_u(ID)}.$$

- *Decrypt* for a valid ciphertext $C = (C_0, C_1, C_2, C_3)$ encrypted for an identity ID and biometric identity BID using the public key $pk_{ID} = (X, Y)$ which may or may not have been replaced by the attacker. Let $w =$

$(C_0, C_1, C_2, ID, pk_{ID})$, $\mathcal{B}$ aborts if $J_v(w) = 0 \mod \tau_v$ just like previous game and chooses $d' \in_R \{0,1\}$. Otherwise it follows that $J_v(w) \neq 0 \mod \tau_v$ and thus $J_v(w) \neq 0 \mod p$, $C_3 = \left(g_2^{J_v(w)} g^{K_v(w)}\right)^s$ and

$$\begin{aligned} C_1 &= (g \cdot F_h(BID))^s \\ &= \left(g_2^{J_h(BID)} g^{K_h(BID)+1}\right)^s \end{aligned}$$

where $s \in_R \mathbb{Z}_p^\times$. Now, $\mathcal{B}$ extracts

$$g_2^s = \left(\frac{C_3}{C_1^{K_v(w)/(K_h(BID)+1)}}\right)^{\frac{1}{J_v(w) - K_v(w) \cdot J_h(BID)/(K_h(BID)+1)}}$$

and computes $e(Y \cdot Z, g_2)^s$ this allows for the computation of $m = \dfrac{C_0}{e(Y \cdot Z, g_2)^s}$ regardless of the fact whether (X, Y) is the original public key or not.

Changing the generation of the master key does has no effect on $\mathcal{B}$'s ability to answer $\mathcal{A}_I^{new}$'s queries like in *Game 7* and and the distribution of the master key remains unchanged, hence we have $Adv_8 = Adv_7$.

Game 9 In this game we modify the generation of the ciphertext again. Using variables $b, c \in_R \mathbb{Z}_p^\times$ defined in *Game 2* and *Game 6* respectively. We set $C_1^* = g^{c(K_h(BID^*)+1)}$ and $T = A^{bc}$.

$\mathcal{B}$ retrieves x_{ID^*} such that $pk_{ID^*} = (A^{x_{ID^*}}, A^{\gamma x_{ID^*}}, F_h(BID^*)^\gamma)$, flips a binary coin $d^* \in_R \{0,1\}$ and computes

$$\begin{aligned} C_0^* &= m_{d^*} \cdot e\left(Y^* \cdot Z^*, g_2\right)^c \\ &= m_{d^*} \cdot e\left(A^{\gamma x_{ID^*}} \cdot F_h(BID^*)^\gamma, g^b\right)^c \\ &= m_{d^*} \cdot e\left(g^{a x_{ID^*}} \cdot F_h(BID^*), g^{bc}\right)^\gamma \\ &= m_{d^*} \cdot e\left(g \cdot F_h(BID^*)^{1/a x_{ID^*}}, g^{abc}\right)^{\gamma x_{ID^*}} \\ &= m_{d^*} \cdot e\left(g \cdot F_h(BID^*)^{1/a x_{ID^*}}, T\right)^{\gamma x_{ID^*}} . \end{aligned} \tag{11.9}$$

It then computes $C_2^* = (g^c)^{K_u(ID^*)}$, $w^* = H(C_0^*, C_1^*, C_2^*, ID^*, pk_{ID^*})$ and $C_3^* = (g^c)^{K_v(w^*)}$. If $J_v(w^*) \neq 0 \mod p$ then $\mathcal{B}$ aborts like in *Game 5* otherwise it returns $(C_0^*, C_1^*, C_2^*, C_3^*)$.

Since we have $J_v(w^*) = 0 \mod p$, these changes do not affect the distribution of the challenge ciphertext and we have $Adv_9 = Adv_8$.

Game 10 In this game we change the challenge phase again. $\mathcal{B}$ only retains $g_2 = g^b$ and $C_1^{*1/(K_h(BID^*)+1)} = g^c$ and forgets the values b, c. Challenge is constructed as shown in (11.9) in *Game 9* but T is chosen randomly, $T \in_R \mathbb{G}$. The simulator uses the values g^a, g^b, g^c and never touches a, b, c. The transition between *Game 9* and *Game 10* is based upon the indistinguishability of $T = g^{abc}$ from $T \in_R \mathbb{G}$ and both games are equal unless there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ which can tell the difference between the two values. This is clearly an instance of the 3-DDH which we wanted to achieve from the very beginning. Since the only difference between *Game 9* and *Game 10* is the condition enforced by the indistinguishability of $T = g^{abc}$ from $T \in_R \mathbb{G}$ hence the difference of the success probabilities of *Game 9* and *Game 10* is the lower bound on $Adv_{\mathcal{A}'}^{3\text{-}DDH}(k)$. Therefore we have

$$|\Pr(S_9) - \Pr(S_{10})| \leq Adv_{\mathcal{A}'}^{3\text{-}DDH}(k).$$

Additionally C_0^* now reveals no information about m_{d^*} and is completely independent of d^*, hence $\Pr(S_{10}) = 1/2$. This brings game hopping to an end, we now combine the results from the games. We have

$$Adv_7 = Adv_8 = Adv_9 \leq Adv_{\mathcal{A}'}^{3\text{-}DDH}(k)$$

and

$$Adv_5 = Adv_6 \leq 16 \cdot Adv_7 + \delta$$

also

$$Adv_5 = \frac{Adv_4}{\tau_u \tau_v \tau_h (n+1)^3}$$

where $\tau_u = 2q_{pk}, \tau_v = 2q_d$ and $\tau_h = 2q_{bid}$. And thus

$$\begin{aligned} Adv_4 &= 2q_{pk} \cdot 2q_d \cdot 2q_{bid}(n+1)^3 \cdot Adv_5 \\ &\leq 8q_{pk}q_dq_{bid}(n+1)^3 \cdot (16 \cdot Adv_7 + \delta). \end{aligned}$$

We also have

$$Adv_3 = Adv_4$$

and

$$\begin{aligned} Adv_1 = Adv_2 &= |\Pr(S_2) - 1/2| \\ &\leq |\Pr(S_2) - \Pr(S_3)| + |\Pr(S_3) - 1/2| \\ &\leq Adv_{\mathcal{A}''}^{CR}(k) + Adv_3. \end{aligned}$$

Combining the above equations we finally have

$$Adv_1 < 8q_{pk}q_dq_{bid}(n+1)^3 \cdot \left(16 \cdot Adv^{3\text{-}DDH}_{\mathcal{A}'}(k) + \delta\right) + Adv^{CR}_{\mathcal{A}''}(k).$$

To prove the same result for $\mathcal{A}^{new}_{II}$ we follow the exact same procedure except $\mathcal{A}^{new}_{II}$ cannot replace the public key of the target identity ID^* and never makes partial private key queries since it has the access to the master private key $msk = \gamma$ which he uses to compute it. Finally, in *Game 10* we combine all the results according to the changes described to obtain the final equation.

□

Discussion The proof of Theorem 11.7 shows that the derived scheme is secure against the adversaries described in Section 9.3. Just like in case of previous reduction breaking the scheme under the imposed constraints would be equivalent to solving the 3-DDH problem and breaking the collision resistant hash function H. These are hard problems and basing our security on them gives an idea of the hardness of breaking the indistinguishability of the scheme.

At this point we should note that at the onset of the games we assume the knowledge of the number of oracle queries the attacker makes, we denote this by q_d, q_{pk} and q_{bid}. This might not be possible in general hence to achieve this we make an assumption about the number of queries made. If our assumption seems too small or too large then we adjust the values accordingly and try again. For the purpose of our proofs we start with the correct assumed values. The proof shows that the original scheme presented by Dent, Libert & Paterson (2008) can be modified to provide privacy even in the event of device compromise. Additionally, the derived scheme is secure in the standard model. To fortify our claims further we also demonstrate a fully functional implementation of the derived scheme in Section 12.5.

12. Implementation

To prove that the ideas developed in this thesis work in practice we have developed a prototype for both the schemes discussed. The prototypes have been developed on *Android* platform and implement the concrete constructions presented in Section 10 and Section 11. In this section we describe the implementation details, motivations for the chosen design, internal workings of the schemes discussed and look at the various aspects of the developed applications.

12.1. Platform. As mentioned earlier the target of the modified scheme is to provide two-factor security for mobile devices hence our applications have been developed for the mobile platform. For implementation purposes we have chosen *Android* as the mobile operating system. There are several reasons for this choice and we highlight them below.

- *Android* and *iOS* are the two most common mobile operating systems, while *iOS* is not open source, *Android* is, to a large extent. Being open source is one of the key aspects which helps in designing secure systems, therefore Android seems to be the natural choice.

- *Android* is a Linux-based operating system and has been proven to support a diverse array of applications.

- Our applications have been developed using Java as the programming language since it is very well supported and has open source libraries for almost every task. *Android* has native support for Java and our applications do not need to be ported to suit the development platform.

- *Android* is the most popular mobile operating system at the moment with a smart phone market share close to 50%. Due to being open source and thus free, vendors can modify it to suit their phones. This allows easy adoption and hence it supports devices in a wide spectrum of prices. These features will encourage usage of the developed application due to being readily available.

12.2. Programming Language and Libraries. We have chosen Java as the language to develop our applications as it provides numerous features which are very beneficial for our development. We enlist these features below.

- *Open Source* - Java is an open source language, this provides the developer a deeper insight into its workings. It is of prime importance for security critical applications to use open source platforms. As stated very articulately by Kerckhoffs (1883) that one must not attempt to achieve security by obscurity, the security of a system must only depend upon the keys and not on the secrecy of the system itself.

- *Platform Independent* - Java is completely platform independent and the developed applications can be easily ported to mobile devices and desktops either of which could be running on various operating systems. This saves additional development effort and facilitates the acceptance of applications.
- *Object Oriented* - Java provides object oriented application development environment which is easy to maintain and scale. Due to being modular the applications can be conveniently integrated with other applications and this increases its value.
- *Application Programming Interface (API)'s* - Java has various API's which can be used for a number of purposes, this saves development effort and improves efficiency.
- *Well Supported* - Since Java is a widely used language it is well supported and new versions are rolled out with improvements regularly.

These reasons make a very compelling case for using Java and consequently it is our programming language of choice. As a result our applications can be run on mobile devices running Android as well as desktops.

Almost all the libraries used by our applications are standard Java and Android libraries. However we do require some special libraries to implement pairing-based cryptosystems and read QR codes. We discuss the details of these libraries below.

12.2.1. Bilinear Maps. Apart from the standard Java libraries the application uses the Java Pairing Based Cryptography (jPBC) library developed by Caro (2010). jPBC is an open source Java porting of the Pairing-Based Cryptography (PBC) library written in C and developed by Lynn (2007). The PBC library is an open source C library built on the GNU Multiple Precision Arithmetic Library (GMP). The GMP is responsible for performing the mathematical operations underlying pairing-based cryptosystems. The PBC library is designed to support implementation of pairing-based cryptosystems, it provides good speed along with portability. Some of the routines it provides are elliptic curve generation, elliptic curve arithmetic and pairing computation. The API is abstract and allows the programmer to use the PBC library with basic knowledge of pairings and group theory.

The jPBC library was written by Caro (2010) to port the functionalities provided by the PBC library to Java platform and it consists of two parts.

- A *Java porting* of the PBC library which supports computations of symmetric and asymmetric pairings.
- A *Java wrapper* of the PBC library to delegate all the computation to the C library.

Since our application is *Android* based therefore we only use the Java porting for development. The Java porting is further divided into several modules which are

- `jpbc-api`: This module contains the API exposed by the jPBC.
- `jpbc-plaf`: This module includes the default API's implementation and PBC's Java porting.
- `jpbc-pbc`: This module contains the API's implementation to be used when PBC is chosen as the computation engine, it also includes the PBC's Java wrapper.
- `jpbc-crypto`: This module provides the implementation of some sample cryptosystems.

Our application uses only `jpbc-plaf` and `jpbc-plaf` modules for development.

jPBC Internals The jPBC library allows usage of several types of pairings by defining curve type and other values in a parameter file. These pairing types are defined by default and are a part of the library. We use pairing *Type A* for our applications.

The *Type A* pairings are constructed on the curve $y^2 = x^3 + x$ over the field $\mathbb{F}_q$ for some prime $q = 3 \mod 4$. The pairing is symmetric and constructed using G1 as the base group which is the group of points $E(\mathbb{F}_q)$. Hence we have $\#E(\mathbb{F}_q) = q + 1$ and $\#E(F_{q^2}) = (q + 1)^2$. Thus the embedding degree is 2, and hence the target group $\mathbb{G}_T$ is a subgroup of $\mathbb{F}_{q^2}$. The order r of $\mathbb{G}_T$ is some prime factor of $q + 1$.

We can write $q + 1 = r \cdot h$. For efficiency, r is picked to be a Solinas prime, that is, r has the form $2a \pm 2b \pm 1$ for some integers $0 < b < a$. Below is a sample parameter file where the bit lengths are, `r = 181 bits`, `q = 1027 bits` and `h = 845 bits`. This provides good security for most applications.

`param_type_a.txt`

```
type a


q 46580998768652845643382050909768057383427364176178662548400435380768762342065628936475112129870929579771480644427739378041784486773479853562930954566451274773934527992685401786815085428211467783641470348441066697220226465765969924641429885305459811174805539802499340269299050528609565521257835054329637378 19


r 30649910817317777167166839130958165414022662707415613 43
```

```
h 151977599694462091499092633437896417267123966707330122 0
  642365133032480824760576673164317062853388644631314794 9
  674446923820293427836654923432201182900089027234505142 7
  129683698149410512911997114723483146377405442651416064 7
  419776092918950130186100519432087 40

exp1 103

exp2 181

sign0 -1

sign1 -1
```

12.2.2. QR Codes. We discuss the simulation of biometric identities using QR codes in section Section 12.5.1. The application developed for the derived construction uses QR Droid services to read the QR codes. These services are part of a third party application called QR Droid developed by DroidLa (2012). The services provide functionality to scan and decode a QR code as well as encode text into a QR code.

For the derived version of the application we need to decode a QR code after scanning it. The decoded text string is then passed on to the application for further processing. The library has been conveniently integrated with our application and works seamlessly. On calling a method of the library the control is passed to the QR Droid application and the result is returned to the parent application on conclusion of the method. Details of each class file contained in this library is provided in Section 12.5.3.

12.3. Device Specifications. The development of both the original scheme and the modified scheme has been done on *HTC Incredible S* which has Android 2.3.5 running on it. However our application is independent of the device and would work on any device which has an Android 1.6 or higher installed on it. Our application requires a camera to read the QR codes and the QR Droid 4.1.2 application developed by DroidLa (2012) installed on the mobile device to decode the read QR code. The application can be installed free of charge from the Android market, it occupies 3.3 Mbs of space and is extremely light weight. The device should have a working internet connection to use email services, apart from this there are no special requirements.

Figure 12.1: The HTC Incredible S.

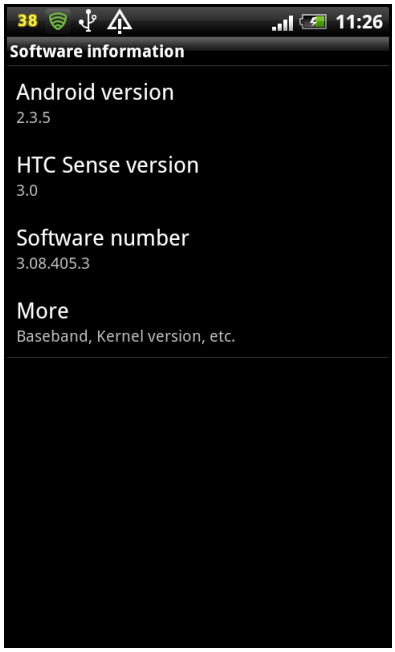

Figure 12.2: The software specification of the device.

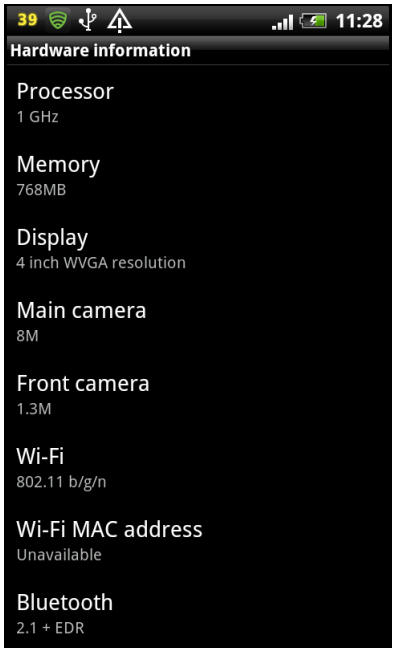

Figure 12.3: The hardware specification of the device.

12.4. Original Construction. In this section we discuss the various aspects of the implementation of the construction originally presented by Dent, Libert & Paterson (2008). First we take a look at the interface of the application and then we discuss the class structure where we provide the details of the role played by each class as well as the methods implemented by them.

12.4.1. Application Interface. Our application is a prototype of a mail client which implements the original scheme. For simulation purposes all three parties involved in the scheme namely the sender, receiver and KGC have been implemented in the same device. It allows a user to send encrypted mails using the receiver's email address. The email address is treated as an unique identifier which is used to encrypt the messages send to the owner of that address. The receiver can decrypt the encrypted messages by following the scheme and obtaining necessary values from the KGC after he has authenticated himself as the owner of the email address. We now present the application interface to explain the functioning of the scheme.

Figure 12.4: The main screen of the application provides the user with the option to read mail, compose mail, generate key and read about the application.

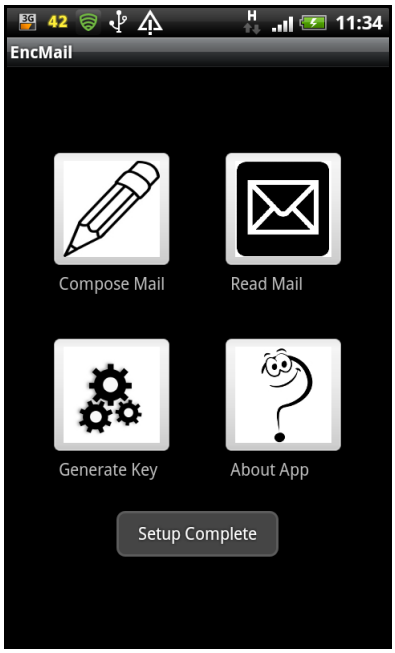

Figure 12.5: The first step in the scheme is to generate the system parameters. This is done by the KGC on tapping the *Generate Key* button.

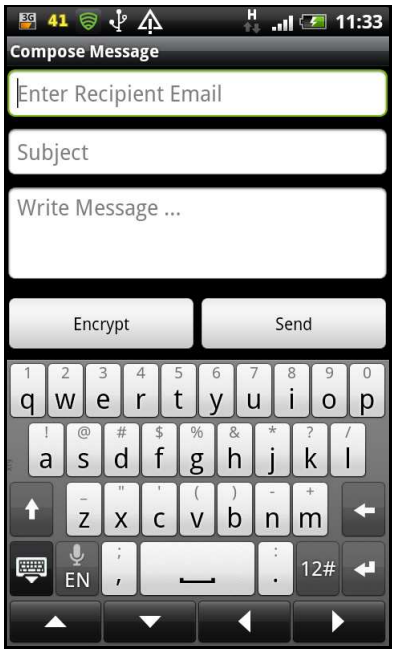

Figure 12.6: After the system parameters have been generated, the sender can compose messages by tapping the *Compose Mail* button from the main screen.

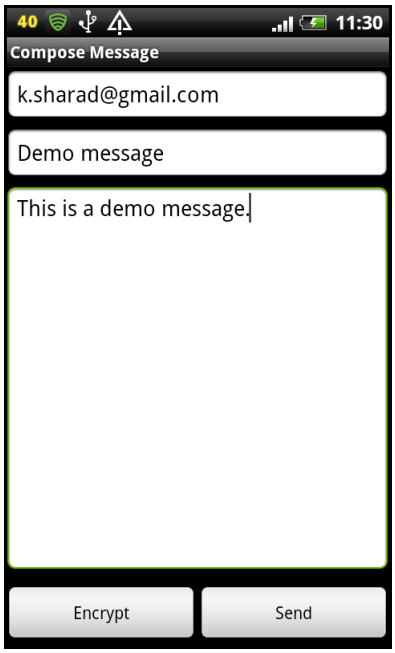

Figure 12.7: A sample message.

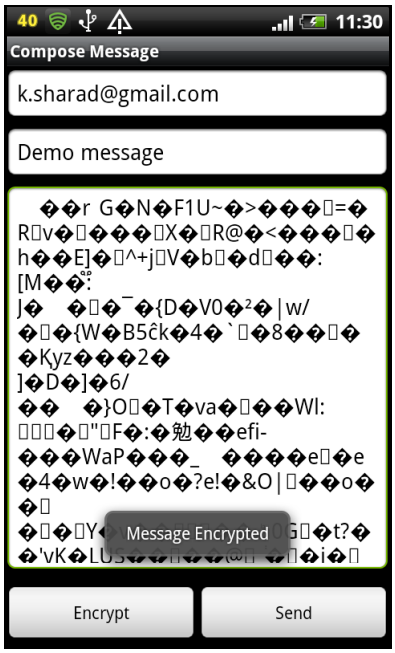

Figure 12.8: After obtaining the receiver's private key the sender encrypts the message by tapping the *Encrypt* button.

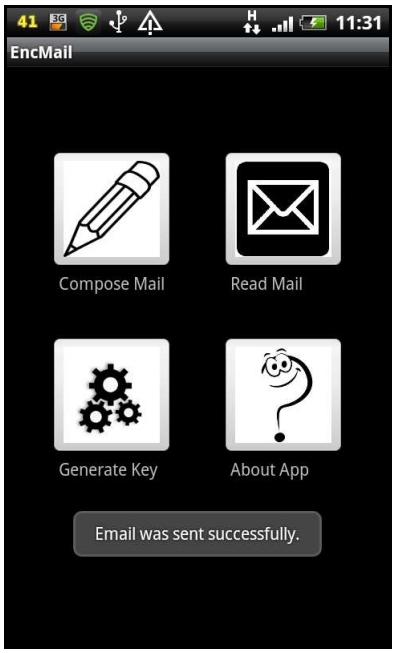

Figure 12.9: The sender can now send the encrypted message by tapping the *Send* button.

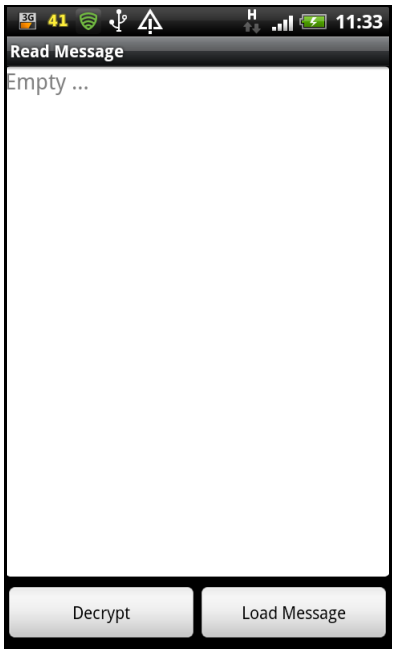

Figure 12.10: On tapping the *Read Mail* button from the main screen the receiver is directed to the interface to read messages.

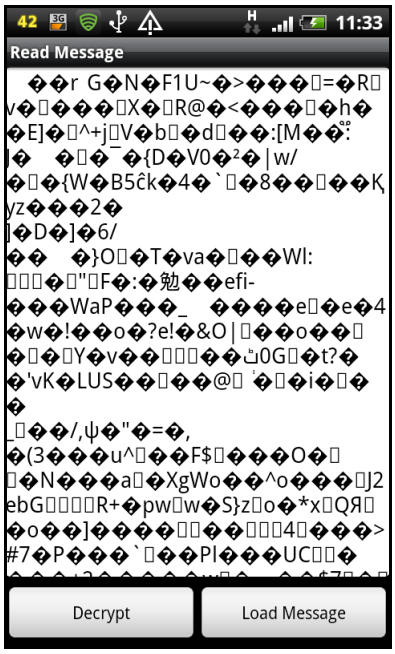

Figure 12.11: The receiver loads the encrypted message by tapping the *Load* button.

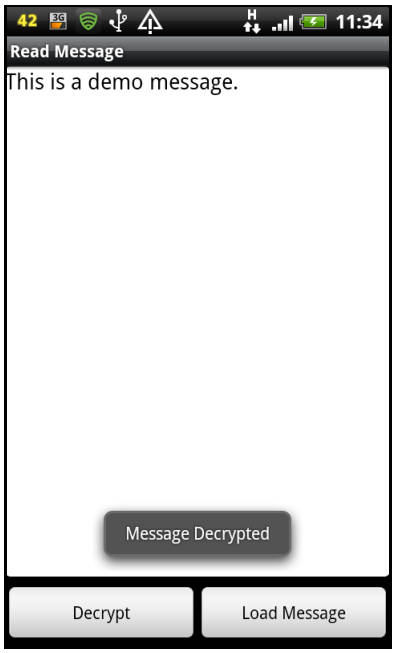

Figure 12.12: Finally, receiver decrypts the encrypted message by tapping the *Decrypt* button and is presented with readable text as send by the sender.

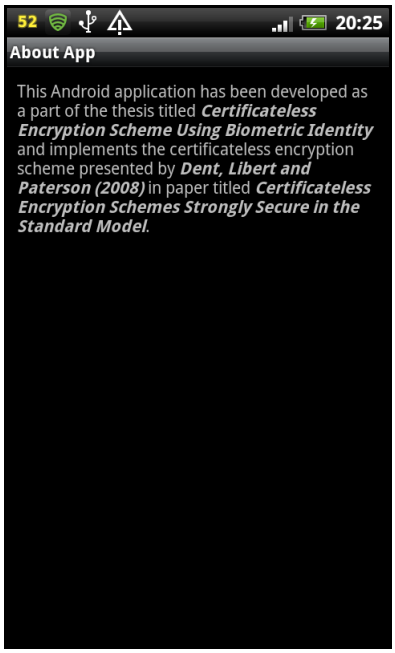

Figure 12.13: The user can read more about the application by tapping the *About App* button from the main screen.

12.4.2. Class Structure. The Android application of the original construction consists of 9 modules. These modules implement the various algorithms executed by the sender, receiver and KGC in form of classes. In this section we describe these classes and then present the class diagram to give a picture of the class structure.

1. `CertificatelessEncAppActivity.java`: This is the main class file of the application as described in Appendix A.1. It provides an interface to access other modules of the application.
2. `ComposeMessage.java`: This class as described in Appendix A.2 implements the interface for composing, encrypting and sending mails.
3. `Help.java`: This class as described in Appendix A.3, provides an interface with information about the application.
4. `KeyGenerationCenter.java`: In this class as described in Appendix A.4, we implement the algorithms executed by the KGC. It contains the logic for `Setup` and `Extract` algorithms.
5. `Mail.java`: This class as described in Appendix A.5, implements the logic to send out mails. It is accessible to the `ComposeMessage.java` class which uses it to send messages when called by the user.
6. `Methods.java`: This class as described in Appendix A.6, provides implementation of several methods that are used by other classes such as converting a string to binary, SHA-1 hash function, converting bytes to binary string and hash function used by the encryption scheme.
7. `ReadMessage.java`: This class as described in Appendix A.7, provides an interface to read encrypted messages after decrypting them.
8. `Receiver.java`: This class as described in Appendix A.8, implements the algorithms executed by the receiver. The logic for `SetSec`, `SetPub`, `SetPriv` and `Decrypt` algorithms is contained in this class.
9. `Sender.java`: This class as described in Appendix A.9, implements the `Encrypt` algorithm executed by the sender.

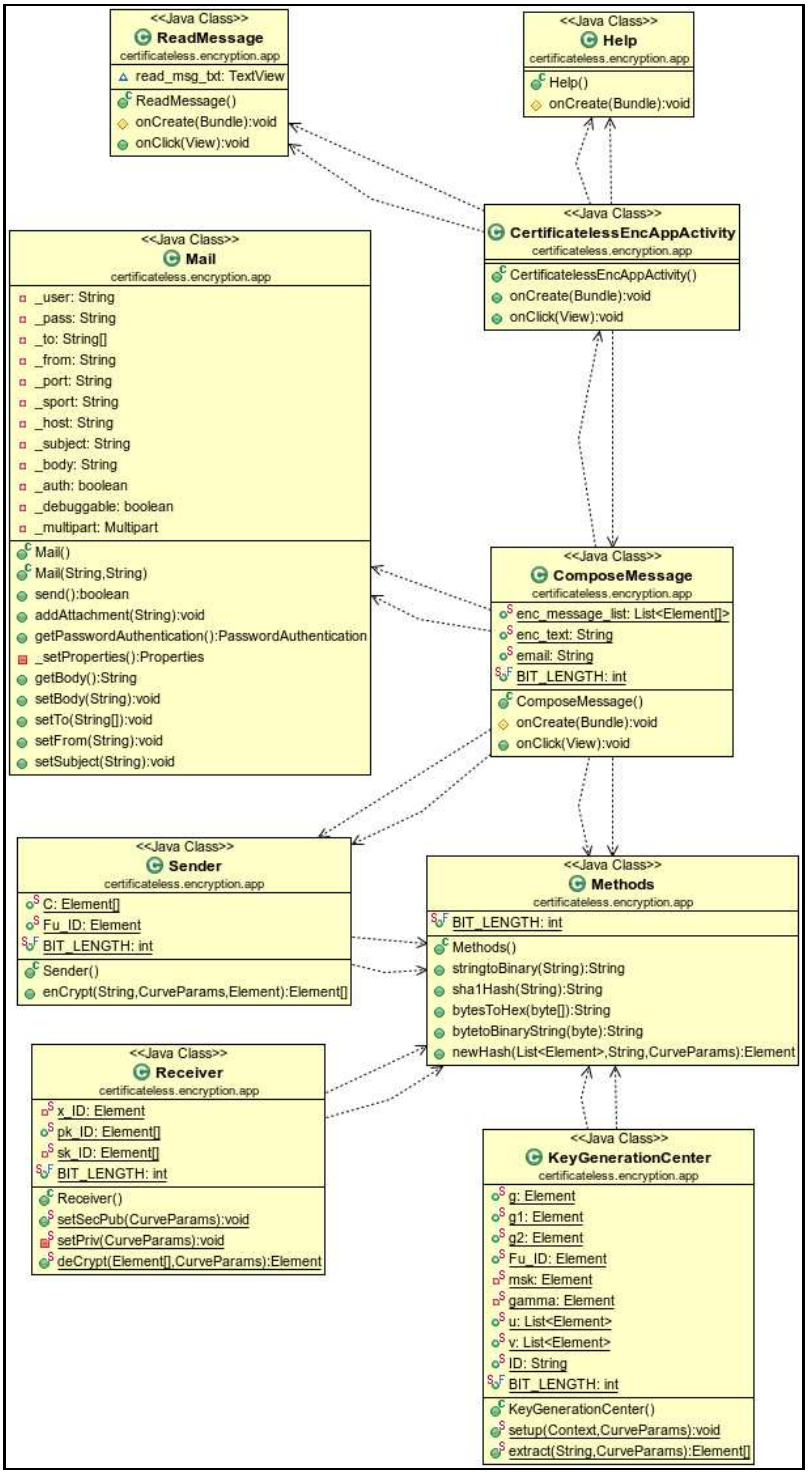

Figure 12.14: The class diagram showing the relationship among different classes

12.5. Derived Construction. After discussing the implementation of the original scheme in detail we are now ready to look at the changes brought on by the derived scheme and the impact of those changes. First, we describe the implementation details of extracting biometric identity of the user, then we present the application interface to understand the flow of the algorithm. Lastly, we discuss the class structure which explains the modular design of the application and the function of each module.

12.5.1. Biometric Identity. As discussed earlier the derived construction uses biometric identities to provide two-factor authentication. There are several possible ways to extract biometric identity of the user, most of them use physiological characteristics like fingerprint, DNA, face recognition, palm print, hand geometry, iris recognition, etc. We aimed to design the encryption scheme for standard smart phones and this leaves us only with the option of using face recognition or hand geometry, since measuring all the other characteristics requires special equipment and therefore is beyond the capabilities of a mobile device.

We decided to go with face recognition for our scheme since it is relatively well researched field as compared to hand geometry. Also software packages exist that provide basic face recognition whereas there are hardly any freely available software packages which implement biometric identity derivation using hand geometry. To use face recognition for deriving the biometric identity from facial pictures we thought of using the existing open source libraries. We specifically needed a library that provides feature extraction of facial images for reasons we elaborate later. We came across many hurdles in achieving this and we now describe them in detail.

Face recognition is an evolving field and not yet mature enough to provide fool prove solutions. Recognising a face is a challenging task and requires considerable effort. For our scheme we needed to extract an identity from a face picture, however we had certain requirements that needed to be fulfilled for the method to be of practical use to the scheme. We list the requirements of our scheme and challenges faced below.

- *Feature Extraction* - For our application the only method to obtain a biometric identity from a face picture is by using feature extraction as the device should store no information about the biometrics. Hence, the derived scheme needed a library to extract features from a face picture and use those features to generate a unique identity. None of the existing open source Java libraries support this feature and this is a serious impediment.

- *Existing Libraries* - The libraries that implement face recognition do so by comparing the input image of the face to a database of images already stored in the device. This mechanism cannot be employed for our purposes since this will not protect against device compromise. If the attacker gains control of the device then he would have access to the stored secrets of the device as well as the raw data from which biometrics is extracted. Hence he would

possess all the information that is needed to decrypt a message and such an attacker could not be stopped.

- *Performance* - We need to ensure a certain level of performance for our schemes to be functional. Face recognition is reasonably fast in desktop environments however the performance drops appreciably on mobile devices. This is mainly due to the use of Java and limited resources of the mobile devices. The libraries that provide fast face recognition are generally developed in C/C++ and Java porting although convenient cannot match the performance of the libraries written in C/C++. This makes the application sluggish and depreciates user experience.

- *Picture Quality* - Ideally face recognition is employed on biometric images to ensure best results. This is not possible in our case since the cameras installed on mobile devices are not advanced enough. Also we cannot expect an user to take perfect images every time, such a requirement would be unreasonable and make the scheme unusable. The application needs to work with relatively crude images of the user and still successfully extract the same unique identity every time. This is a very stiff requirement and not possible to achieve with the currently existing libraries. Also using multiple crude versions of a face picture to derive the same identity repeatedly impacts the entropy of the data extracted and thus leaves the possibility of the attacker exploiting this drawback open.

Due to these challenges we were unable to achieve the goal of implementing feature extraction using facial images. Face recognition is a involved process and developing a solution from scratch is a huge task. The prime objective of this thesis was to design and implement an encryption scheme which provides security in the event of device compromise. Developing a complete solution for face recognition is a very challenging research problem which is out of the scope of the work presented.

However, to prove that the ideas developed in the modified scheme do work in principle we have simulated the facial images using QR codes. Thus the application developed provides a proof of concept for our derived scheme. For demonstration purposes QR codes are quite similar to face pictures as far as our scheme is concerned. Some of the key features which make them useful for prototyping our construction are

- QR codes can be considered as a face picture and they help in explaining the idea of our scheme.

- Unique QR codes can be generated to model unique facial characteristic of each person.

- QR codes can be easily read and used for extracting the same unique identity every time.

- The time taken to read and decode a QR code is very low and has virtually no effect on the user experience.

Hence these features help us in demonstrating the exact functionality of our scheme. It shows how the actual scheme would function if there was a way to extract biometric identities using faces in an efficient way on mobile devices. Apart from this change the scheme is complete and implements the derived construction efficiently.

Figure 12.15: A sample QR code.

12.5.2. Application Interface. The application we have developed implements a proof of concept of the derived scheme. The application simulates the derived construction. For demonstration all three parties involved in the scheme namely the sender, receiver and KGC have been implemented in the same device. The application allows the sender to send messages encrypted using the receiver's public key, facial picture and unique public identity, we use email address of the receiver as his unique public identity. The receiver can decrypt the messages after generating his private key using his face picture and values obtained from the KGC as specified in the scheme.

The interface of the derived construction is very similar to that of the application presented earlier, however there are some key differences which we elaborate in the following pages. We now present the application interface to explain the functioning of the scheme.

Figure 12.16: The main screen of the application provides the user with the option to read mail, compose mail, generate key and read about the application.

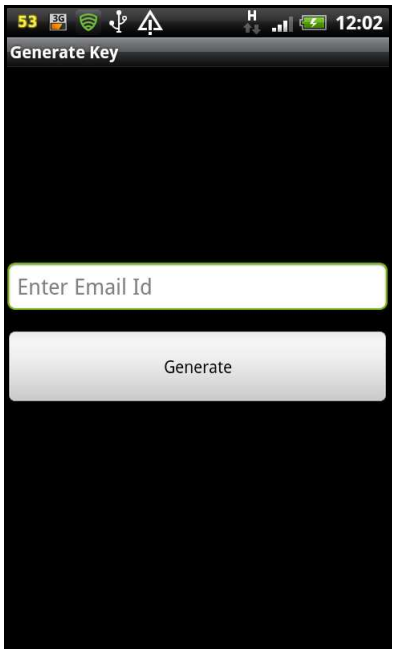

Figure 12.17: The generate key screen is started on tapping the *Generate Key* button and requires the receiver to enter his email id to generate keys.

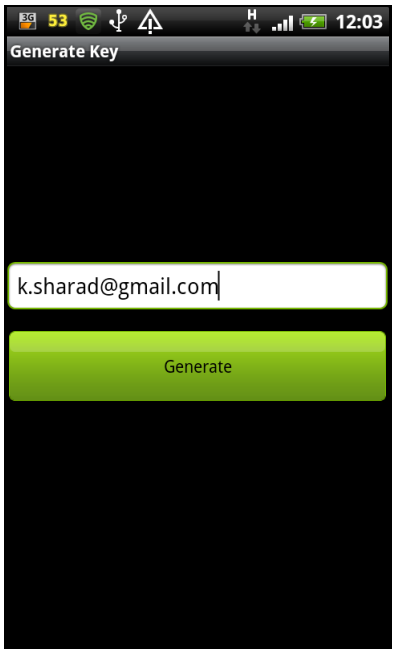

Figure 12.18: After the email id is entered, tapping the *Generate* button generates the system parameters and the receiver's public key.

Figure 12.19: Before the keys can be generated the receiver must also provide his face picture which has been simulated using a QR code in our case.

Figure 12.20: The public key is generated once the QR code is read.

Figure 12.21: The sender can now compose messages by tapping the *Compose Mail* button from the main screen.

Figure 12.22: A sample message.

Figure 12.23: After obtaining the receiver's public key the sender encrypts the message by tapping the *Encrypt* button. Then he is prompted to provide the receiver's face picture.

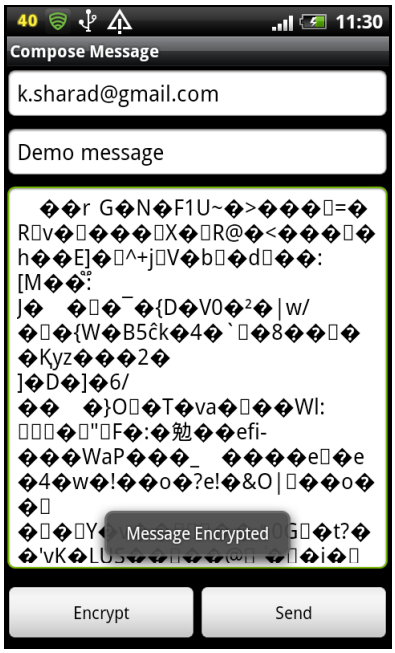

Figure 12.24: On reading the QR code the message is encrypted using the receiver's public key, email id and face picture.

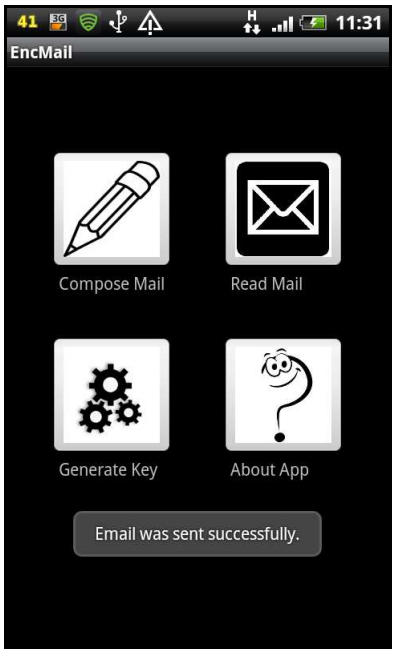

Figure 12.25: The encrypted message is sent by tapping the *Send* button.

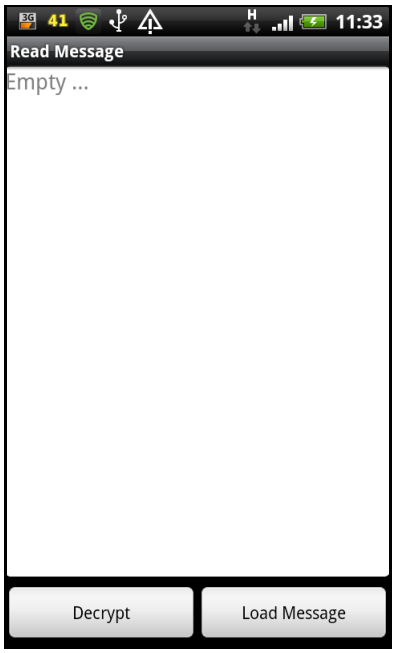

Figure 12.26: On tapping the *Read Mail* button from the main screen the receiver is directed to the interface to read messages.

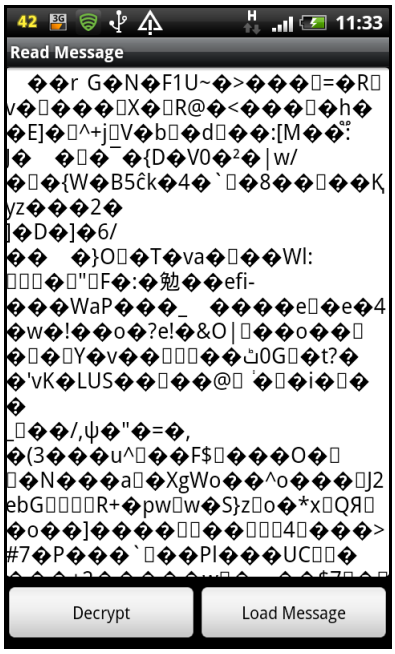

Figure 12.27: The receiver loads the encrypted message by tapping the *Load* button.

Figure 12.28: On tapping the *Decrypt* button the receiver is prompted to provide his face picture to decrypt the encrypted message.

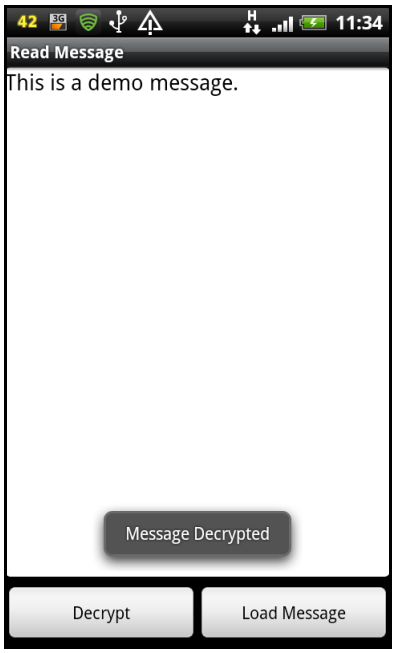

Figure 12.29: Finally, the receiver decrypts the message on providing the correct QR code and is presented with readable text as send by the sender.

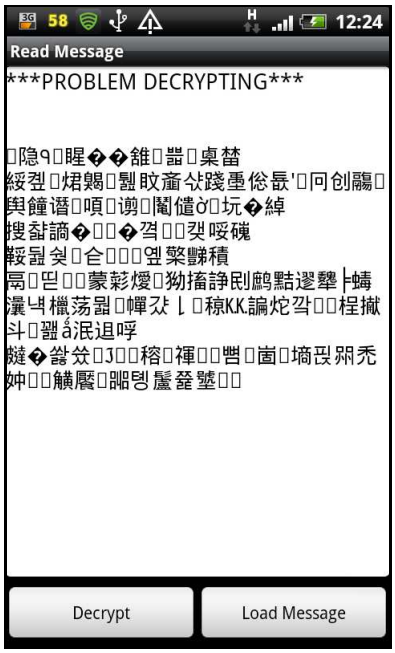

Figure 12.30: An error message is received if someone tries to decrypt by providing an incorrect face picture.

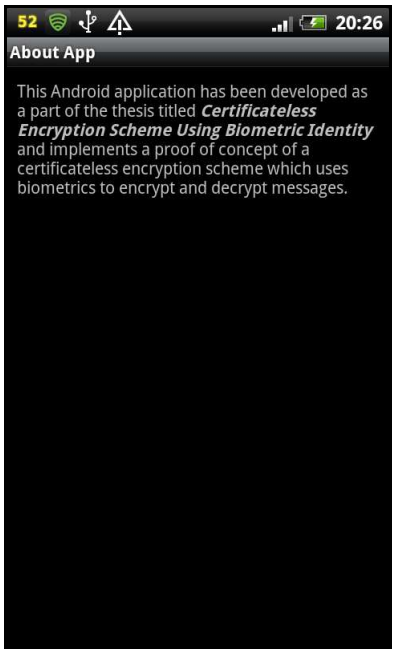

Figure 12.31: The user can read more about the application by tapping the *About App* button from the main screen.

12.5.3. Class Structure. The class structure of the derived scheme is very similar to that of original scheme as discussed previously. However, the methods that are defined in the classes are as per the specifications of the derived scheme. We also use some additional classes to simulate face recognition using QR Codes. These classes are part of the library which implements reading the QR codes using the third party application called *QR Droid* developed by DroidLa (2012).

Below we list all the classes that are used and define the ones that are new to this scheme.

1. `CertificatelessEncModAppActivity.java`: This is the main class file of the application as described in Appendix B.1. It provides an interface to access other modules of the application.

2. `ComposeMessage.java`: This class has same functionality as presented earlier in Section 12.4.2 and is described in Appendix B.2.

3. `Decode.java`: This class as shown in Appendix B.3, implements the methods to decode the QR code entered from the camera.

4. `Encode.java`: This class as shown in Appendix B.4, implements the methods to encode a text into a QR code entered from the camera. We never use this class but it is part of the library *QR Droid* and hence included for completeness.

5. `Help.java`: This class has same functionality as presented earlier in Section 12.4.2 and is described in Appendix B.5.

6. `KeyGenerationCenter.java`: This class has same functionality as presented earlier in Section 12.4.2 and is described in Appendix B.6.

7. `Mail.java`: This class has same functionality as presented earlier in Section 12.4.2 and is described in Appendix B.7.

8. `Methods.java`: This class has same functionality as presented earlier in Section 12.4.2 and is described in Appendix B.8.

9. `ReadMessage.java`: This class has same functionality as presented earlier in Section 12.4.2 and is described in Appendix B.9.

10. `Receiver.java`: This class has same functionality as presented earlier in Section 12.4.2 and is described in Appendix B.10.

11. `Scan.java`: This class as shown in Appendix B.11, provides the functionality to scan the QR code entered from the camera.

12. `Sender.java`: This class has same functionality as presented earlier in Section 12.4.2 and is described in Appendix B.12.

13. `Services.java`: This class as shown in Appendix B.13, implements the interface to access the classes `Decode.java`, `Encode.java` and `Scan.java`.

14. `Setup.java`: This class as shown in Appendix B.14, generates the various system parameters as well as the public and the private keys for the encryption scheme.

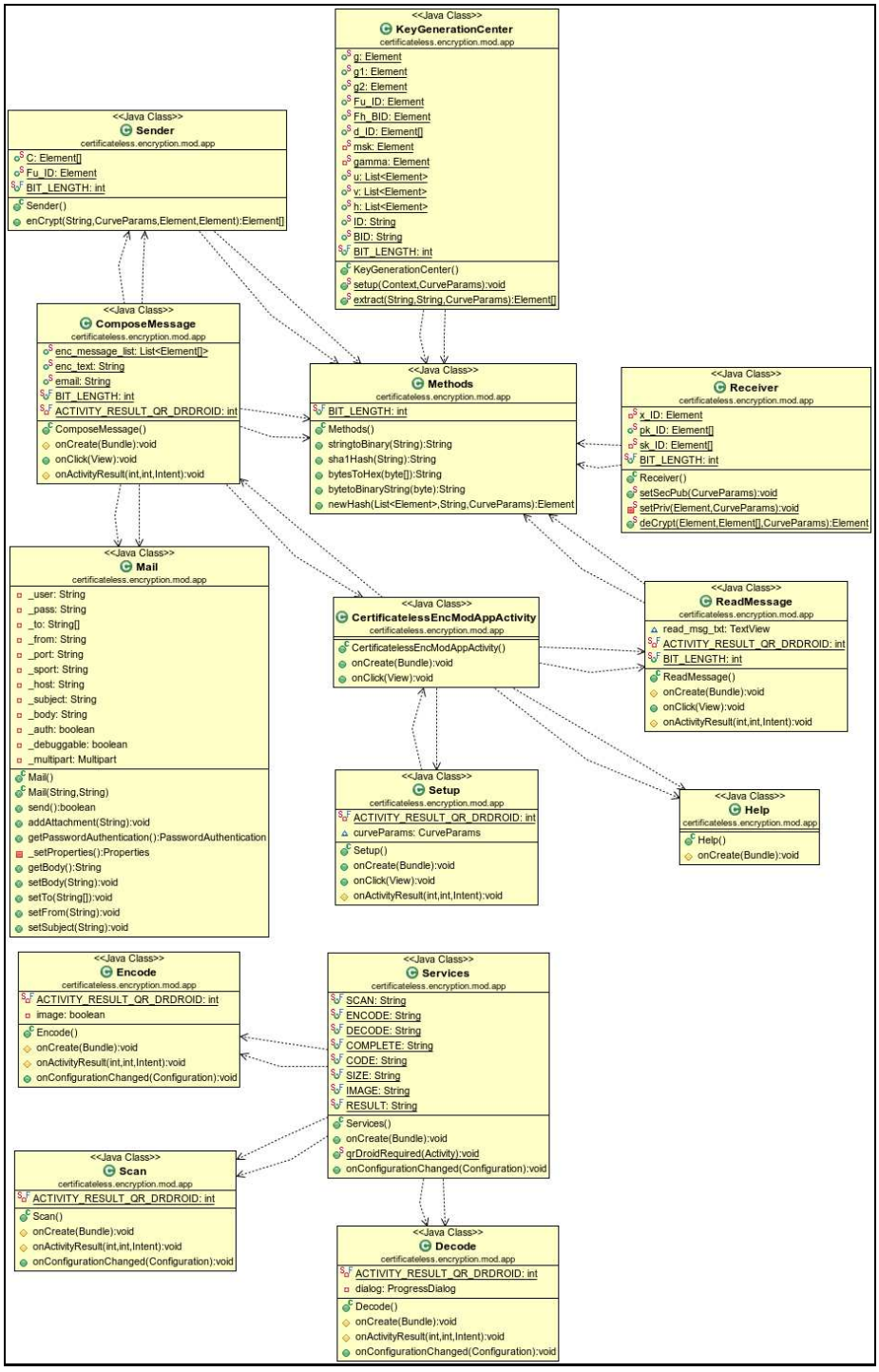

Figure 12.32: The class diagram showing the relationship among different classes

13. Conclusion

In this thesis we presented a new approach to identity based encryption based on a certificateless scheme. As we saw the scheme has been designed for mobile devices and protects the privacy of users in the event of device compromise. We saw the motivations and reasons behind developing such a scheme. Then we looked at the advantages and applications that our schemes have and explored their constructions in detail. We presented the security proofs for the original and the derived scheme followed by the implementation details. The Android applications developed for the original and the derived schemes proves the practical importance of these constructions. It also provides the proof that the ideas are not just theoretical but also work in practice. In this section we try to give an overview of the work presented and explore the areas for future work.

13.1. Contributions. Our work demonstrates the implementation of the scheme presented by Dent, Libert & Paterson (2008). We developed an Android application which is fully functional and can be used as a practical solution to communicate privately. Subsequently, we saw how this construction can be modified to meet a different security goal. We presented a construction which used biometric identities to encrypt and decrypt messages and provided a two-factor authentication. We derived this construction from the work done by Dent, Libert & Paterson to protect the privacy of users even if an adversary gains control of the device with the stored secrets. This scheme safeguards the privacy of the user in the event of device compromise. Additionally, we also proved that the derived construction is secure in the standard model. An Android prototype demonstrating the derived scheme was also developed. This proves that our new scheme is practical and efficiently implementable on mobile devices which have limited resources as compared to a desktop.

We showed both the schemes can be implemented efficiently using open source libraries developed by Lynn (2007) and Caro (2010). Before our work no previous Android application implementing *Certificateless Encryption Schemes Strongly Secure in the Standard Model* existed. The developed application can be used to learn more about the scheme and improve it further. Our work can be seen as a guideline which provides future developers with better understanding of the challenges faced when implementing pairing-based cryptographic applications on mobile devices. Android is still a relatively new platform and it gets updated rather frequently. The developed applications can give an insight to the open source community in developing features for Android which readily support cryptographic work. At the moment not all Java libraries are well supported by the Android platform. We live in a world where surveillance has never been easier and this has spurred the growth of applications which provide privacy and secrecy. These factors are going to be instrumental in shaping the development of Android as a complete operating system for mobile devices. Our applications help in highlighting the scope for further development to support cryptography natively in Android.

13.2. Challenges and Future Work. Although we have made considerable strides in providing privacy even in the event of device compromise, still there are certain facets of the scheme that can be improved. As we saw the lack of open source face recognition libraries in Android platform made implementation of biometrics difficult. Feature extraction is a extremely hard problem and clever solutions are needed for it to work successfully even with basic cameras that normal smart phones have. Ideally, a biometric picture is required to do this but for the scheme to be acceptable on a large scale other solutions need to be found. Perhaps we could also look at the possibility of extracting biometric identity from other data sources, the voice of the user could be an option. Android is a relatively new platform and functionalities like voice identification are not well supported. Voice identification is also a tricky problem which has not been fully solved irrespective of the platform we consider. Hopefully as the mobile phone technology matures we will see more solutions, future work can try to solve these issues.

Further improvements could also be done by making the encryption scheme faster. The applications developed by us use the Java library developed by Caro (2010) for all the pairing-based cryptography related operations. This library is a Java porting of the C library library originally written by Lynn (2007). Since Java has native Android support it is very convenient to use the jPBC library and it works out of the box however we do lose out on speed. Android has recently started providing native C/C++ support through *Native Development Kit* but this increases the application complexity and there are issues which need to be circumvented before one can switch completely to C/C++. Also, a performance improvement is not guaranteed by merely switching to a C/C++ library and careful design is required to derive any gains. This can be looked as an area of further investigation that would improve the performance of the scheme.

A new problem called Denial-of-Decryption (DoD) Attack in CL-PKC schemes was presented by Liu, Au & Susilo (2006). In this attack the adversary replaces the receiver's public key by someone else's public key. So when the sender tries to send an encrypted message to the receiver he uses the replaced public key and the receiver's identity to encrypt the message. Consequently, the receiver cannot decrypt the message and the sender remains unaware of this. The authors have coined the term *Denial-of-Decryption Attack* for this kind of threat and the name has been inspired from commonly known *Denial-of-Service Attack*. Both the original scheme and the derived scheme are vulnerable against such an attack. To circumvent this problem Liu, Au & Susilo propose a new paradigm called *Self-Generated-Certificate Public Key Cryptography* and provide a generic construction of a self-generated-certificate public key encryption scheme which is secure in the standard model. Their scheme uses certificateless signature and certificateless encryption as the building blocks. Also, the authors have described a certificateless encryption scheme with concrete implementation that is provably secure in the standard model. The work done by Liu, Au & Susilo is worth exploring and can improve the discussed constructions further. However, one must carefully consider the impact of such ideas on the discussed constructions and make sure that we do

not lose the benefits for which the schemes were originally designed.

In both the constructions discussed in this thesis we assume that the KGC does not actively launch attacks. A new kind of threat model was considered by Au, Mu, Chen, Wong, Liu & Yang (2007), this model did not impose such constraints on the KGC. The authors proceed on to show that the existing CL-PKE schemes are insure in such a security model where the adversaries maliciously generate system-wide parameters. Both the original as well as the derived construction are vulnerable against such a KGC. Au, Mu, Chen, Wong, Liu & Yang have claimed that the existing schemes still suffer from the key escrow problem. They also give new proofs to show that there are generic constructions which have been recently proposed for certificateless signature and certificateless encryption that are secure under the new threat models. The work done by Au, Mu, Chen, Wong, Liu & Yang merits careful study and evaluation in the light of the constructions presented in this thesis. If we can find concrete constructions which are secure in the new model then this will further strengthen the security. Again, the modifications must be done with care and attention to detail so that we can conserve the beneficial properties of the work discussed in this thesis.

Thus we see that the area of CL-PKE is relatively young and it is still evolving. This presents us with the scope for further improvements. As discussed the improvements can be structural as well as implementation based. With our work we solved some important problems but there is always scope for improvement and we need to do more to achieve perfection. We hope that our work inspires growth in the area of CL-PKC and people come up with ingenious solutions which further paves the path for knowledge and learning in the area of certificateless schemes.

Acronyms

3-DDH The Decisional 3-Party Diffie-Hellman Problem. 20, 48, 51, 52, 64, 66, 68, 71, 72, 79, 80

API Application Programming Interface. 83, 84

CA Certification Authority. 10, 14, 16, 26, 30, 37, 40

CBE Certificate-Based Encryption. 19

CL-PKC Certificateless Public Key Cryptography. 11, 16, 17, 19, 37, 111, 112

CL-PKE Certificateless Public Key Encryption. 11, 22, 25, 34, 35, 37, 38, 40, 41, 43, 112

CLE Certificateless Encryption. 19

DBDH The Decisional Bilinear Diffie-Hellman Problem. 20

DoD Denial-of-Decryption. 111

GMP GNU Multiple Precision Arithmetic Library. 83

IBC Identity Based Cryptography. 17, 37

IBE Identity-Based Encryption. 10, 11, 15–17, 19, 20, 34, 37

IND-CCA Indistinguishability Under Chosen Ciphertext Attack. 27, 31, 40–42, 50, 52, 54, 72

jPBC Java Pairing Based Cryptography. 83, 84, 111

KGC Key Generation Center. 17, 20, 22, 25, 26, 29–31, 34, 35, 37, 40, 41, 44, 46, 48, 68, 69, 87, 93, 97, 112

PBC Pairing-Based Cryptography. 83, 84

PKG Private Key Generator. 15–17, 19, 37

PKI Public Key Infrastructure. 10, 14–16, 34, 37, 40

TTP Trusted Third Party. 16, 17

References

C. ADAMS & S. LLOYD (2002). *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley Longman Publishing Co., Inc.

SATTAM S. AL-RIYAMI & KENNETH G. PATERSON (2003). Certificateless Public Key Cryptography. In *ASIACRYPT*, CHI-SUNG LAIH, editor, volume 2894 of *Lecture Notes in Computer Science*, 452–473. Springer. ISBN 3-540-20592-6.

M.H. AU, Y. MU, J. CHEN, D.S. WONG, J.K. LIU & G. YANG (2007). Malicious KGC attacks in certificateless cryptography. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, 302–311. ACM.

JOONSANG BAEK, WILLY SUSILO & JIANYING ZHOU (2007). New constructions of fuzzy identity-based encryption. In *ASIACCS*, FENG BAO & STEVEN MILLER, editors, 368–370. ACM. ISBN 1-59593-574-6.

DAN BONEH & MATTHEW K. FRANKLIN (2001). Identity-Based Encryption from the Weil Pairing. In *CRYPTO*, JOE KILIAN, editor, volume 2139 of *Lecture Notes in Computer Science*, 213–229. Springer. ISBN 3-540-42456-3.

ANGELO DE CARO (2010). Java Pairing-Based Cryptography Library. URL http://gas.dia.unisa.it/projects/jpbc/.

JAE CHOON CHA & JUNG HEE CHEON (2003). An Identity-Based Signature from Gap Diffie-Hellman Groups. In *Public Key Cryptography*, YVO DESMEDT, editor, volume 2567 of *Lecture Notes in Computer Science*, 18–30. Springer. ISBN 3-540-00324-X.

L. CHEN, KEITH HARRISON, A. MOSS, DAVID SOLDERA & NIGEL P. SMART (2002). Certification of Public Keys within an Identity Based System. In *Proceedings of the 5th International Conference on Information Security*, ISC '02, 322–333. Springer-Verlag, London, UK, UK. ISBN 3-540-44270-7. URL http://dl.acm.org/citation.cfm?id=648026.744529.

J. DANKERS, T. GAREFALAKIS, R. SCHAFFELHOFER & T. WRIGHT (2002). Public key infrastructure in mobile systems. *Electronics & Communication engineering journal* **14**(5), 180–190.

ALEXANDER W. DENT (2006a). A Note On Game-Hopping Proofs. *IACR Cryptology ePrint Archive* **2006**, 260.

ALEXANDER W. DENT (2006b). A Survey of Certificateless Encryption Schemes and Security Models. *IACR Cryptology ePrint Archive* **2006**, 211.

ALEXANDER W. DENT, BENOÎT LIBERT & KENNETH G. PATERSON (2008). Certificateless Encryption Schemes Strongly Secure in the Standard Model. In *Public Key Cryptography*, RONALD CRAMER, editor, volume 4939 of *Lecture Notes in Computer Science*, 344–359. Springer. ISBN 978-3-540-78439-5.

YVO DESMEDT & JEAN-JACQUES QUISQUATER (1986). Public-Key Systems Based on the Difficulty of Tampering (Is There a Difference Between DES and RSA?). In *CRYPTO*, ANDREW M. ODLYZKO, editor, volume 263 of *Lecture Notes in Computer Science*, 111–117. Springer.

YEVGENIY DODIS & JONATHAN KATZ (2005). Chosen-Ciphertext Security of Multiple Encryption. In *TCC*, JOE KILIAN, editor, volume 3378 of *Lecture Notes in Computer Science*, 188–209. Springer. ISBN 3-540-24573-1.

DROIDLA (2012). QR Droid. URL https://market.android.com/details?id=la.droid.qr&hl=en.

JUN FURUKAWA, NUTTAPONG ATTRAPADUNG, RYUICHI SAKAI & GOICHIRO HANAOKA (2008). A Fuzzy ID-Based Encryption Efficient When Error Rate Is Low. In *INDOCRYPT*, DIPANWITA ROY CHOWDHURY, VINCENT RIJMEN & ABHIJIT DAS, editors, volume 5365 of *Lecture Notes in Computer Science*, 116–129. Springer. ISBN 978-3-540-89753-8.

DAVID GALINDO, PAZ MORILLO & CARLA RÀFOLS (2006). Breaking Yum and Lee Generic Constructions of Certificate-Less and Certificate-Based Encryption Schemes. In *EuroPKI*, ANDREA S. ATZENI & ANTONIO LIOY, editors, volume 4043 of *Lecture Notes in Computer Science*, 81–91. Springer. ISBN 3-540-35151-5.

C. GENTRY & A. SILVERBERG (2002). Hierarchical ID-based cryptography. *Advances in Cryptology – ASIACRYPT 2002* 149–155.

CRAIG GENTRY (2003). Certificate-Based Encryption and the Certificate Revocation Problem. In *EUROCRYPT*, ELI BIHAM, editor, volume 2656 of *Lecture Notes in Computer Science*, 272–293. Springer. ISBN 3-540-14039-5.

MARC GIRAULT (1991). Self-Certified Public Keys. In *EUROCRYPT*, DONALD W. DAVIES, editor, volume 547 of *Lecture Notes in Computer Science*, 490–497. Springer. ISBN 3-540-54620-0.

P. GUTMANN (2002). PKI: it's not dead, just resting. *Computer* **35**(8), 41–49.

F. HESS (2003). Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography*, 310–324. Springer.

QIONG HUANG & DUNCAN S. WONG (2007). Generic Certificateless Encryption in the Standard Model. In *IWSEC*, ATSUKO MIYAJI, HIROAKI KIKUCHI & KAI RANNENBERG, editors, volume 4752 of *Lecture Notes in Computer Science*, 278–291. Springer. ISBN 978-3-540-75650-7.

DETLEF HÜHNLEIN, MICHAEL J. JACOBSON JR. & DAMIAN WEBER (2000). Towards Practical Non-interactive Public Key Cryptosystems Using Non-maximal Imaginary Quadratic Orders. In *Selected Areas in Cryptography*, DOUGLAS R. STINSON & STAFFORD E. TAVARES, editors, volume 2012 of *Lecture Notes in Computer Science*, 275–287. Springer. ISBN 3-540-42069-X.

A. KERCKHOFFS (1883). La cryptographie militaire. *Journal des sciences militaires* **9**(1), 5–38.

JOSEPH K. LIU, MAN HO AU & WILLY SUSILO (2006). Self-Generated-Certificate Public Key Cryptography and Certificateless Signature / Encryption Scheme in the Standard Model. *IACR Cryptology ePrint Archive* **2006**, 373.

BENJAMIN LYNN (2007). Pairing-Based Cryptography Library. URL http://crypto.stanford.edu/pbc/.

UELI M. MAURER & YACOV YACOBI (1991). Non-interactive Public-Key Cryptography. In *EUROCRYPT*, DONALD W. DAVIES, editor, volume 547 of *Lecture Notes in Computer Science*, 498–507. Springer. ISBN 3-540-54620-0.

K.G. PATERSON (2002a). Cryptography from pairings: a snapshot of current research. *Information Security Technical Report* **7**(3), 41–54.

K.G. PATERSON (2002b). ID-based signatures from pairings on elliptic curves. *Electronics Letters* **38**(18), 1025–1026.

HOLGER PETERSEN, PATRICK HORSTER & DELTA PATRICK HORSTER (1997). Self-certified keys - Concepts and Applications. In *In Proc. Communications and Multimedia Security'97*, 102–116. Chapman & Hall.

S. SAEEDNIA (2003). A note on Girault's self-certified model. *Information Processing Letters* **86**(6), 323–327.

SHAHROKH SAEEDNIA (1997). Identity-Based and Self-Certified Key-Exchange Protocols. In *ACISP*, VIJAY VARADHARAJAN, JOSEF PIEPRZYK & YI MU, editors, volume 1270 of *Lecture Notes in Computer Science*, 303–313. Springer. ISBN 3-540-63232-8.

AMIT SAHAI & BRENT WATERS (2005). Fuzzy Identity-Based Encryption. In *EUROCRYPT*, RONALD CRAMER, editor, volume 3494 of *Lecture Notes in Computer Science*, 457–473. Springer. ISBN 3-540-25910-4.

R. SAKAI, K. OHGISHI & M. KASAHARA (2000). Cryptosystems based on pairing. In *The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan*, 135–148.

ADI SHAMIR (1984). Identity-Based Cryptosystems and Signature Schemes. In *CRYPTO*, G. R. BLAKLEY & DAVID CHAUM, editors, volume 196 of *Lecture Notes in Computer Science*, 47–53. Springer. ISBN 3-540-15658-5.

NIGEL P. SMART (2003). Access control using pairing based cryptography. In *Proceedings of the 2003 RSA conference on The cryptographers' track*, CT-RSA'03, 111–121. Springer-Verlag, Berlin, Heidelberg. ISBN 3-540-00847-0. URL http://dl.acm.org/citation.cfm?id=1767011.1767023.

NP SMART (2002). Identity-based authenticated key agreement protocol based on Weil pairing. *Electronics Letters* **38**(13), 630–632.

HATSUKAZU TANAKA (1987). A Realization Scheme for the Identity-Based Cryptosystem. In *CRYPTO*, CARL POMERANCE, editor, volume 293 of *Lecture Notes in Computer Science*, 340–349. Springer. ISBN 3-540-18796-0.

S. TSUJII & T. ITOH (1989). An ID-based cryptosystem based on the discrete logarithm problem. *Selected Areas in Communications, IEEE Journal on* **7**(4), 467–473.

B. WATERS (2005). Efficient identity-based encryption without random oracles. *Advances in Cryptology–EUROCRYPT 2005* 557–557.

DAE HYUN YUM & PIL JOONG LEE (2004a). Generic Construction of Certificateless Encryption. In *ICCSA (1)*, ANTONIO LAGANÀ, MARINA L. GAVRILOVA, VIPIN KUMAR, YOUNGSONG MUN, CHIH JENG KENNETH TAN & OSVALDO GERVASI, editors, volume 3043 of *Lecture Notes in Computer Science*, 802–811. Springer. ISBN 3-540-22054-2.

DAE HYUN YUM & PIL JOONG LEE (2004b). Identity-Based Cryptography in Public Key Management. In *EuroPKI*, SOKRATIS K. KATSIKAS, STEFANOS GRITZALIS & JAVIER LOPEZ, editors, volume 3093 of *Lecture Notes in Computer Science*, 71–84. Springer. ISBN 3-540-22216-2.

A. Source Code for the Original Construction

A.1. CertificatelessEncAppActivity.java

```
// This is the main class of the application and implements
    the main screen

package certificateless.encryption.app;

import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Toast;

public class CertificatelessEncAppActivity extends Activity
    implements
   OnClickListener {
 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main_layout);

  View composeButton =
     findViewById(R.id.compose_message_button);
  composeButton.setOnClickListener(this);

  View readButton = findViewById(R.id.read_message_button);
  readButton.setOnClickListener(this);

  View setupButton = findViewById(R.id.setup_button);
  setupButton.setOnClickListener(this);

  View helpButton = findViewById(R.id.help_button);
  helpButton.setOnClickListener(this);
 }

 // Declaring the buttons
 public void onClick(View v) {

  switch (v.getId()) {
  case R.id.help_button:
   Intent i1 = new Intent(this, Help.class);
   startActivity(i1);
   break;
```

```
    case R.id.compose_message_button:
      Intent i2 = new Intent(this, ComposeMessage.class);
      startActivity(i2);
      break;

    case R.id.read_message_button:
      Intent i3 = new Intent(this, ReadMessage.class);
      startActivity(i3);
      break;

    case R.id.setup_button:
      CurveParams curveParams = new
          CurveParams().load(getResources()
          .openRawResource(R.raw.a_181_603));

      KeyGenerationCenter.setup(v.getContext(), curveParams);
      Receiver.setSecPub(curveParams);

      Toast.makeText(CertificatelessEncAppActivity.this,
          "Setup Complete", Toast.LENGTH_LONG).show();
      break;
    }
  }
}
```

A.2. ComposeMessage.java

```
// This class implements the compose mail interface and
   provides the functionality to compose, encrypt and send
   messages

package certificateless.encryption.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.EditText;
import android.widget.Toast;

public class ComposeMessage extends Activity implements
    OnClickListener {

  public static List<Element[]> enc_message_list = new
     ArrayList<Element[]>();
  public static String enc_text = "";
  public static String email = "";

  public static final int BIT_LENGTH = 100;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.compose_message);

    View encButton = findViewById(R.id.compose_message_enc);
    encButton.setOnClickListener(this);

    View sendButton =
        findViewById(R.id.compose_message_send);
    sendButton.setOnClickListener(this);
  }

  public void onClick(View v) {

    CurveParams curveParams = new
        CurveParams().load(getResources()
        .openRawResource(R.raw.a_181_603));
    Pairing pairing = PairingFactory.getPairing(curveParams);

    final EditText compose_message_to = (EditText)
        findViewById(R.id.compose_message_to);
    final EditText compose_message_subject = (EditText)
        findViewById(R.id.compose_message_subject);
    final EditText compose_message_text = (EditText)
        findViewById(R.id.compose_message_text);

    switch (v.getId()) {
    // Encrypting the message
    case R.id.compose_message_enc:
      int message_len = 76;
      int pad_len = 0;

      email = compose_message_to.getText().toString();
      String encoding = "UTF-16BE";
```

```
String message =
    compose_message_text.getText().toString();
String message_final = message;
String value = "00";
String str_add = String.format(
   String.format("%%0%dd", message_len - 2),
       0).replace("0",
   "*");

if (message.length() % message_len != 0) {
  int message_len_quo = message.length() / message_len;
  pad_len = (message_len_quo + 1) * message_len;

  int pad_count = pad_len - message.length();
  String pad = String.format(String.format("%%0%dd",
      pad_count),
     0).replace("0", "~");
  message_final = message + pad;

  if (pad_count < 10)
    value = "0" + pad_count;
  else
    value = new Integer(pad_count).toString();
}

message_final = message_final + str_add + value;

try {
  byte[] message_final_bytes =
      message_final.getBytes(encoding);

  enc_message_list.clear();

  for (int i = 0; i < message_final.length() /
      message_len; i++) {
    byte[] newarr = new byte[2 * message_len];

    System.arraycopy(message_final_bytes, i * 2 *
        message_len,
       newarr, 0, 2 * message_len);

    Element element_temp = pairing.getGT().newElement();
    element_temp.setFromBytes(newarr);

    Element enc_message[] = new Element[4];
    Sender sender = new Sender();

    Methods method = new Methods();
```

```
        String ID =
            method.stringtoBinary(email).substring(0,
            BIT_LENGTH);

        enc_message = sender.enCrypt(ID, curveParams,
            element_temp.duplicate());

        enc_message_list.add(enc_message.clone());
      }

    } catch (UnsupportedEncodingException e) {
      System.out.println("Encoding Error");
    }

    for (int i = 0; i < enc_message_list.size(); i++) {

      try {
        String enc_text_inter = "";
        enc_text_inter = new String(
            enc_message_list.get(i)[0].toBytes(), "UTF-8")
            + new
                String(enc_message_list.get(i)[1].toBytes(),
                "UTF-8")
            + new
                String(enc_message_list.get(i)[2].toBytes(),
                "UTF-8")
            + new
                String(enc_message_list.get(i)[3].toBytes(),
                "UTF-8");
        enc_text = enc_text + enc_text_inter;
      } catch (UnsupportedEncodingException e) {
        System.out.println("Encoding Error");
      }
    }
    compose_message_text.setText(enc_text);
    Toast.makeText(ComposeMessage.this, "Message
        Encrypted",
        Toast.LENGTH_LONG).show();

    break;

  // Sending the mail on tapping send button
  case R.id.compose_message_send:
    Mail m = new Mail("certificateless.enc@googlemail.com",
        "thesis1234");

    String[] toArr = {
        compose_message_to.getText().toString() };
    m.setTo(toArr);
```

```
    m.setFrom("certificateless.enc@googlemail.com");
    m.setSubject(compose_message_subject.getText().toString());
    m.setBody("***BEGIN ENCRYPTED MESSAGE***\n\n\n" +
       enc_text
       + "\n\n\n***END OF ENCRYPTED MESSAGE***");

    try {
      if (m.send()) {
        Toast.makeText(ComposeMessage.this,
           "Email was sent successfully.",
              Toast.LENGTH_LONG)
           .show();

        Intent i5 = new Intent(this,
           CertificatelessEncAppActivity.class);
        startActivity(i5);
      } else {
        Toast.makeText(ComposeMessage.this, "Email was not
           sent.",
           Toast.LENGTH_LONG).show();
      }
    } catch (Exception e) {
      Log.e("Email", "Could not send email", e);
    }
    break;
   }
  }
}
```

A.3. Help.java

```
// This class implements the Help interface

package certificateless.encryption.app;

import android.app.Activity;
import android.os.Bundle;

public class Help extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.help);
  }
}
```

A.4. KeyGenerationCenter.java

```
// This class implements the algorithms run by the KGC

package certificateless.encryption.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.Context;

public class KeyGenerationCenter extends Activity {

  public static Element g, g1, g2, Fu_ID;
  private static Element msk, gamma;
  public static List<Element> u = new ArrayList<Element>();
  public static List<Element> v = new ArrayList<Element>();
  public static String ID;

  public static final int BIT_LENGTH = 100;

  // ~~~~~~~~ Step 1 - Setup: Performed by KGC called at
     startup ~~~~~~~~
  public static void setup(Context context, CurveParams
     curveParams) {

    Pairing pairing = PairingFactory.getPairing(curveParams);

    g = pairing.getG1().newRandomElement();
    g2 = pairing.getG1().newRandomElement();
    gamma = pairing.getZr().newRandomElement();
    g1 = g.duplicate().powZn(gamma.duplicate());
    msk = g2.duplicate().powZn(gamma.duplicate());

    for (int i = 0; i <= BIT_LENGTH; i++) {
      Element u_temp = pairing.getG1().newRandomElement();
      u.add(u_temp.duplicate());
    }

    for (int i = 0; i <= BIT_LENGTH; i++) {
      Element v_temp = pairing.getG1().newRandomElement();
      v.add(v_temp.duplicate());
```

```
    }
  }

  // ~~~~~~~~ Step 2 - Extract: Performed by KGC called by
     Receiver ~~~~~~~~
  public static Element[] extract(String email, CurveParams
     curveParams) {

    Pairing pairing = PairingFactory.getPairing(curveParams);

    Methods method = new Methods();
    ID = method.stringtoBinary(email).substring(0,
       BIT_LENGTH);
    Fu_ID = method.newHash(u, ID, curveParams);

    Element r = pairing.getZr().newRandomElement();

    Element d_ID[] = new Element[2];
    d_ID[0] =
       msk.duplicate().mul(Fu_ID.duplicate().powZn(r.duplicate()));
    d_ID[1] = g.duplicate().powZn(r.duplicate());

    return d_ID.clone();
  }
}
```

A.5. Mail.java

```
/* This class provides the functionality for sending mails.
 * The author of this class is John Simon and the code is
    available
 * at http://www.jondev.net/
 * */

package certificateless.encryption.app;

import java.util.Date;
import java.util.Properties;
import javax.activation.CommandMap;
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.activation.MailcapCommandMap;
import javax.mail.BodyPart;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
```

```
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

public class Mail extends javax.mail.Authenticator {
  private String _user;
  private String _pass;

  private String[] _to;
  private String _from;

  private String _port;
  private String _sport;

  private String _host;

  private String _subject;
  private String _body;

  private boolean _auth;

  private boolean _debuggable;

  private Multipart _multipart;

  public Mail() {
    _host = "smtp.gmail.com"; // default smtp server
    _port = "465"; // default smtp port
    _sport = "465"; // default socketfactory port

    _user = ""; // username
    _pass = ""; // password
    _from = ""; // email sent from
    _subject = ""; // email subject
    _body = ""; // email body

    _debuggable = false; // debug mode on or off - default
       off
    _auth = true; // smtp authentication - default on

    _multipart = new MimeMultipart();

    MailcapCommandMap mc = (MailcapCommandMap) CommandMap
       .getDefaultCommandMap();
    mc.addMailcap("text/html;;
       x-java-content-handler=com.sun.mail.handlers.text_html");
    mc.addMailcap("text/xml;;
```

```
      x-java-content-handler=com.sun.mail.handlers.text_xml");
  mc.addMailcap("text/plain;;
      x-java-content-handler=com.sun.mail.handlers.text_plain");
  mc.addMailcap("multipart/*;;
      x-java-content-handler=com.sun.mail.handlers.multipart_mixed");
  mc.addMailcap("message/rfc822;;
      x-java-content-handler=com.sun.mail.handlers.message_rfc822");
  CommandMap.setDefaultCommandMap(mc);
}

public Mail(String user, String pass) {
  this();

  _user = user;
  _pass = pass;
}

public boolean send() throws Exception {
  Properties props = _setProperties();

  if (!_user.equals("") && !_pass.equals("") && _to.length
      > 0
     && !_from.equals("") && !_subject.equals("")
     && !_body.equals("")) {
    Session session = Session.getInstance(props, this);

    MimeMessage msg = new MimeMessage(session);

    msg.setFrom(new InternetAddress(_from));

    InternetAddress[] addressTo = new
       InternetAddress[_to.length];
    for (int i = 0; i < _to.length; i++) {
     addressTo[i] = new InternetAddress(_to[i]);
    }
    msg.setRecipients(MimeMessage.RecipientType.TO,
       addressTo);

    msg.setSubject(_subject);
    msg.setSentDate(new Date());

    // setup message body
    BodyPart messageBodyPart = new MimeBodyPart();
    messageBodyPart.setText(_body);
    _multipart.addBodyPart(messageBodyPart);

    // Put parts in message
    msg.setContent(_multipart);
```

```
    // send email
    Transport.send(msg);

    return true;
  } else {
    return false;
  }
}

public void addAttachment(String filename) throws
    Exception {
  BodyPart messageBodyPart = new MimeBodyPart();
  DataSource source = new FileDataSource(filename);
  messageBodyPart.setDataHandler(new DataHandler(source));
  messageBodyPart.setFileName(filename);

  _multipart.addBodyPart(messageBodyPart);
}

@Override
public PasswordAuthentication getPasswordAuthentication()
    {
  return new PasswordAuthentication(_user, _pass);
}

private Properties _setProperties() {
  Properties props = new Properties();

  props.put("mail.smtp.host", _host);

  if (_debuggable) {
    props.put("mail.debug", "true");
  }

  if (_auth) {
    props.put("mail.smtp.auth", "true");
  }

  props.put("mail.smtp.port", _port);
  props.put("mail.smtp.socketFactory.port", _sport);
  props.put("mail.smtp.socketFactory.class",
      "javax.net.ssl.SSLSocketFactory");
  props.put("mail.smtp.socketFactory.fallback", "false");

  return props;
}

// the getters and setters
public String getBody() {
```

```
    return _body;
  }

  public void setBody(String _body) {
    this._body = _body;
  }

  public void setTo(String[] toArr) {
    this._to = toArr;
  }

  public void setFrom(String string) {
    this._from = string;
  }

  public void setSubject(String string) {
    this._subject = string;
  }
}
```

A.6. Methods.java

```
// This class implements various methods used by the
    different classes

package certificateless.encryption.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.security.MessageDigest;
import java.util.List;

public class Methods {

  public static final int BIT_LENGTH = 100;

  public String stringtoBinary(String text) {

    byte[] bytes = text.getBytes();
    String binarystr = "";
    for (int i = 0; i < bytes.length; i++) {
      binarystr = binarystr + bytetoBinaryString(bytes[i]);
    }
    return binarystr;
```

```
}

// Computing the SHA-1 hash
public String sha1Hash(String input) {

 byte[] output;
 String binarystr = "";

 try {
  MessageDigest md = MessageDigest.getInstance("SHA1");
  md.update(input.getBytes());
  output = md.digest();

  for (int i = 0; i < output.length; i++) {
   binarystr = binarystr + bytetoBinaryString(output[i]);
  }

 } catch (Exception e) {
  System.out.println("Exception: " + e);
 }
 return binarystr;
}

/*
 * Converting bytes to hexadecimal. The code for this
    method has been taken
 * from http://www.herongyang.com
 */
public String bytesToHex(byte[] b) {
 char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6',
    '7', '8', '9',
    'A', 'B', 'C', 'D', 'E', 'F' };
 StringBuffer buf = new StringBuffer();
 for (int j = 0; j < b.length; j++) {
  buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
  buf.append(hexDigit[b[j] & 0x0f]);
 }
 return buf.toString();
}

/*
 * Converting a byte to binary string. The code for this
    method has been
 * taken from http://helpdesk.objects.com.au
 */
public String bytetoBinaryString(byte n) {
 StringBuilder sb = new StringBuilder("00000000");
 for (int bit = 0; bit < 8; bit++) {
  if (((n >> bit) & 1) > 0) {
```

```
      sb.setCharAt(7 - bit, '1');
     }
    }
    return sb.toString();
  }

  // Calculating the hash used by the scheme
  public Element newHash(List<Element> vector, String
     bitstr,
     CurveParams curveParams) {

    Pairing pairing = PairingFactory.getPairing(curveParams);
    Element hash_val = pairing.getG1().newOneElement();
    hash_val.mul(vector.get(0).duplicate());

    for (int i = 0; i < BIT_LENGTH; i++) {
     if (bitstr.charAt(i) == '1') {
      hash_val.mul(vector.get(i + 1).duplicate());
     }
    }
    return hash_val;
  }
}
```

A.7. ReadMessage.java

```
//This class implements the interface to read messages

package certificateless.encryption.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;

import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.os.Bundle;
import android.text.method.ScrollingMovementMethod;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;
import android.widget.Toast;

public class ReadMessage extends Activity implements
    OnClickListener {
```

```
TextView read_msg_txt;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.read_message);

  read_msg_txt = (TextView)
      findViewById(R.id.read_message_text);

  View decButton = findViewById(R.id.read_message_dec);
  decButton.setOnClickListener(this);

  View loadButton = findViewById(R.id.read_message_load);
  loadButton.setOnClickListener(this);
}

public void onClick(View v) {

  final TextView read_message_text = (TextView)
      findViewById(R.id.read_message_text);

  String encoding = "UTF-16BE";
  CurveParams curveParams = new
      CurveParams().load(getResources()
      .openRawResource(R.raw.a_181_603));
  int message_len = 76;

  switch (v.getId()) {

  case R.id.read_message_dec:
    List<Element[]> enc_message_list = new
        ArrayList<Element[]>();
    enc_message_list = ComposeMessage.enc_message_list;

    String dec_msg_padded = "";
    for (int i = 0; i < enc_message_list.size(); i++) {
      Element dec_message =
          Receiver.deCrypt(enc_message_list.get(i),
          curveParams);

      try {
        String msg_inter = new
            String(dec_message.duplicate()
            .toBytes(), encoding);
        dec_msg_padded = dec_msg_padded + msg_inter;

      } catch (UnsupportedEncodingException e) {
```

```
        System.out.println("Encoding Error");
       }
      }

      String msg_end_num = dec_msg_padded.substring(
         dec_msg_padded.length() - 2,
            dec_msg_padded.length());
      int dec_msg_pad_len = Integer.parseInt(msg_end_num);

      String dec_msg = dec_msg_padded.substring(0,
         dec_msg_padded.length() - dec_msg_pad_len -
            message_len);

      read_message_text.setText(dec_msg);

      Toast.makeText(ReadMessage.this, "Message Decrypted",
         Toast.LENGTH_LONG).show();

      break;

    case R.id.read_message_load:
      read_msg_txt.setMovementMethod(new
         ScrollingMovementMethod());
      read_message_text.setText(ComposeMessage.enc_text);

      break;
    }
  }
}
```

A.8. Receiver.java

```
// This class implements the algorithms executed by the
   Receiver

package certificateless.encryption.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

public class Receiver {

  private static Element x_ID;
  public static Element pk_ID[] = new Element[2];
  private static Element sk_ID[] = new Element[2];
```

```
public static final int BIT_LENGTH = 100;

// Step 3 & 4 - SetSec & SetPub: Performed and called by
   Receiver
public static void setSecPub(CurveParams curveParams) {

  Pairing pairing = PairingFactory.getPairing(curveParams);

  x_ID = pairing.getZr().newRandomElement();

  pk_ID[0] =
     KeyGenerationCenter.g.duplicate().powZn(x_ID.duplicate());
  pk_ID[1] =
     KeyGenerationCenter.g1.duplicate().powZn(x_ID.duplicate());
}

// ~~~~~~~~ Step 5 - SetPriv: Performed and called by
   Receiver ~~~~~~~~
private static void setPriv(CurveParams curveParams) {

  Pairing pairing = PairingFactory.getPairing(curveParams);

  Element r_prime = pairing.getZr().newRandomElement();

  Element d_ID[] =
     KeyGenerationCenter.extract(ComposeMessage.email,
     curveParams);

  sk_ID[0] = (d_ID[0].duplicate().powZn(x_ID.duplicate()))
     .mul(KeyGenerationCenter.Fu_ID.duplicate().powZn(
        r_prime.duplicate()));
  sk_ID[1] = (d_ID[1].duplicate().powZn(x_ID.duplicate()))
     .mul(KeyGenerationCenter.g.duplicate().powZn(
        r_prime.duplicate()));
}

// ~~~~~~~ Step 7 - Decrypt: Performed by Receiver ~~~~~
public static Element deCrypt(Element cipherText[],
   CurveParams curveParams) {

  Pairing pairing = PairingFactory.getPairing(curveParams);

  setPriv(curveParams);

  String str_to_hash = cipherText[0].duplicate().toString()
     + cipherText[1].duplicate().toString()
     + cipherText[2].duplicate().toString()
     + Receiver.pk_ID[0].duplicate().toString()
```

```
      + Receiver.pk_ID[1].duplicate().toString()
      + KeyGenerationCenter.ID;

   Methods method = new Methods();
   String w = method.sha1Hash(str_to_hash);

   Element Fv_W = method.newHash(KeyGenerationCenter.v,
      w.substring(0, BIT_LENGTH), curveParams);

   Element temp1 =
      KeyGenerationCenter.Fu_ID.duplicate().mul(
      Fv_W.duplicate());

   Element temp2 = cipherText[2].duplicate()
      .mul(cipherText[3].duplicate());

   if (pairing.pairing(cipherText[1].duplicate(),
      temp1.duplicate())
      .isEqual(
         pairing.pairing(KeyGenerationCenter.g.duplicate(),
            temp2.duplicate())) == false) {

     System.out.println("ABORT: problem matching ...");

     return pairing.getGT().newZeroElement();
   } else {

     Element dec_msg = cipherText[0].duplicate().mul(
        pairing.pairing(cipherText[2].duplicate(),
           sk_ID[1].duplicate()).div(
           pairing.pairing(cipherText[1].duplicate(),
              sk_ID[0].duplicate())));

     return dec_msg.duplicate();
   }
  }
}
```

A.9. Sender.java

```
// This class implements the algorithms executed by the
   Sender

package certificateless.encryption.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
```

```
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

public class Sender {

  public static Element C[] = new Element[4];
  public static Element Fu_ID;

  public static final int BIT_LENGTH = 100;

  // ~~~~~~~~~~ Step 6 - Encrypt: Performed and called by
     Sender ~~~~~~~~~
  public Element[] enCrypt(String ID, CurveParams
     curveParams, Element m) {

    Pairing pairing = PairingFactory.getPairing(curveParams);

    if (pairing.pairing(Receiver.pk_ID[0].duplicate(),
       KeyGenerationCenter.g1.duplicate()).isEqual(
       pairing.pairing(Receiver.pk_ID[1].duplicate(),
          KeyGenerationCenter.g.duplicate())) == false) {

      System.out.println("ABORT: Incorrect Shape");

      return null;
    } else {

      Element s = pairing.getZr().newRandomElement();

      C[0] = m.duplicate().mul(
         (pairing.pairing(Receiver.pk_ID[1].duplicate(),
            KeyGenerationCenter.g2.duplicate())).powZn(s
            .duplicate()));

      C[1] =
         KeyGenerationCenter.g.duplicate().powZn(s.duplicate());

      Methods method = new Methods();

      Element Fu_ID = method.newHash(KeyGenerationCenter.u,
         ID,
         curveParams);

      C[2] = Fu_ID.duplicate().powZn(s.duplicate());

      String str_to_hash = C[0].duplicate().toString()
         + C[1].duplicate().toString() +
            C[2].duplicate().toString()
         + Receiver.pk_ID[0].duplicate().toString()
```

```
        + Receiver.pk_ID[1].duplicate().toString() + ID;

    String w = method.sha1Hash(str_to_hash);

    Element Fv_W = method.newHash(KeyGenerationCenter.v,
       w.substring(0, BIT_LENGTH), curveParams);

    C[3] = Fv_W.duplicate().powZn(s.duplicate());

    return C.clone();
   }
  }
}
```

B. Source Code for the Derived Construction

B.1. CertificatelessEncModAppActivity.java

```
// This is the main class of the application and implements
    the main screen

package certificateless.encryption.mod.app;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

public class CertificatelessEncModAppActivity extends
    Activity implements
   OnClickListener {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);

    View composeButton =
        findViewById(R.id.compose_message_button);
    composeButton.setOnClickListener(this);

    View readButton = findViewById(R.id.read_message_button);
    readButton.setOnClickListener(this);

    View setupButton = findViewById(R.id.setup_button);
    setupButton.setOnClickListener(this);

    View helpButton = findViewById(R.id.help_button);
    helpButton.setOnClickListener(this);
  }

  // Declaring the buttons
  public void onClick(View v) {

    switch (v.getId()) {
    case R.id.help_button:
      Intent i1 = new Intent(this, Help.class);
      startActivity(i1);
      break;

    case R.id.compose_message_button:
      Intent i2 = new Intent(this, ComposeMessage.class);
```

```
    startActivity(i2);
    break;

   case R.id.read_message_button:
    Intent i3 = new Intent(this, ReadMessage.class);
    startActivity(i3);
    break;

   case R.id.setup_button:
    Intent i4 = new Intent(this, Setup.class);
    startActivity(i4);

    break;
   }

  }
}
```

B.2. ComposeMessage.java

```
// This class implements the compose mail interface and
   provides the functionality to compose, encrypt and send
   messages

package certificateless.encryption.mod.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.EditText;
import android.widget.Toast;

public class ComposeMessage extends Activity implements
   OnClickListener {
```

```
public static List<Element[]> enc_message_list = new
    ArrayList<Element[]>();
public static String enc_text = "";
public static String email = "";

public static final int BIT_LENGTH = 100;
private static final int ACTIVITY_RESULT_QR_DRDROID = 0;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.compose_message);

  View encButton = findViewById(R.id.compose_message_enc);
  encButton.setOnClickListener(this);

  View sendButton =
      findViewById(R.id.compose_message_send);
  sendButton.setOnClickListener(this);
}

public void onClick(View v) {

  final EditText compose_message_to = (EditText)
      findViewById(R.id.compose_message_to);
  final EditText compose_message_subject = (EditText)
      findViewById(R.id.compose_message_subject);

  switch (v.getId()) {
  // Encrypting the message
  case R.id.compose_message_enc:

    Intent qrDroid = new Intent(Services.SCAN);

    try {
      startActivityForResult(qrDroid,
          ACTIVITY_RESULT_QR_DRDROID);
    } catch (ActivityNotFoundException activity) {
      Services.qrDroidRequired(ComposeMessage.this);
    }

    break;

  // Sending the mail on tapping send button
  case R.id.compose_message_send:
    Mail m = new Mail("certificateless.enc@googlemail.com",
       "thesis1234");
```

```
    String[] toArr = {
        compose_message_to.getText().toString() };
    m.setTo(toArr);
    m.setFrom("certificateless.enc@googlemail.com");
    m.setSubject(compose_message_subject.getText().toString());
    m.setBody("***BEGIN ENCRYPTED MESSAGE***\n\n\n" +
        enc_text
       + "\n\n\n***END OF ENCRYPTED MESSAGE***");

    try {
      if (m.send()) {
        Toast.makeText(ComposeMessage.this,
           "Email was sent successfully.",
              Toast.LENGTH_LONG)
           .show();

        Intent i5 = new Intent(this,
           CertificatelessEncModAppActivity.class);
        startActivity(i5);
      } else {
        Toast.makeText(ComposeMessage.this, "Email was not
            sent.",
           Toast.LENGTH_LONG).show();
      }
    } catch (Exception e) {
      Log.e("Email", "Could not send email", e);
    }
    break;
  }
}

// Reading the QR code
protected void onActivityResult(int requestCode, int
    resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

  if (ACTIVITY_RESULT_QR_DRDROID == requestCode && null !=
      data
      && data.getExtras() != null) {

    CurveParams curveParams = new
        CurveParams().load(getResources()
        .openRawResource(R.raw.a_181_603));
    Pairing pairing =
        PairingFactory.getPairing(curveParams);

    final EditText compose_message_text = (EditText)
        findViewById(R.id.compose_message_text);
    final EditText compose_message_to = (EditText)
```

```
    findViewById(R.id.compose_message_to);

// Read result from QR Droid (it's stored in
    la.droid.qr.result)
String biometrics =
    data.getExtras().getString(Services.RESULT);
Methods method = new Methods();

String BID =
    method.stringtoBinary(biometrics).substring(0,
   BIT_LENGTH);

Element Fh_BID = method.newHash(KeyGenerationCenter.h,
    BID,
   curveParams);

int message_len = 76;
int pad_len = 0;

email = compose_message_to.getText().toString();
String encoding = "UTF-16BE";
String message =
    compose_message_text.getText().toString();
String message_final = message;
String value = "00";
String str_add = String.format(
   String.format("%%0%dd", message_len - 2),
       0).replace("0",
   "*");

if (message.length() % message_len != 0) {
  int message_len_quo = message.length() / message_len;
  pad_len = (message_len_quo + 1) * message_len;

  int pad_count = pad_len - message.length();
  String pad = String.format(String.format("%%0%dd",
      pad_count),
     0).replace("0", "~");
  message_final = message + pad;

  if (pad_count < 10)
    value = "0" + pad_count;
  else
    value = new Integer(pad_count).toString();
}

message_final = message_final + str_add + value;

try {
```

```
    byte[] message_final_bytes =
       message_final.getBytes(encoding);

    enc_message_list.clear();

    for (int i = 0; i < message_final.length() /
       message_len; i++) {
     byte[] newarr = new byte[2 * message_len];

     System.arraycopy(message_final_bytes, i * 2 *
        message_len,
        newarr, 0, 2 * message_len);

     Element element_temp = pairing.getGT().newElement();
     element_temp.setFromBytes(newarr);

     Element enc_message[] = new Element[4];
     Sender sender = new Sender();

     String ID =
        method.stringtoBinary(email).substring(0,
        BIT_LENGTH);

     enc_message = sender.enCrypt(ID, curveParams,
        Fh_BID,
        element_temp.duplicate());

     enc_message_list.add(enc_message.clone());
    }

  } catch (UnsupportedEncodingException e) {
    System.out.println("Encoding Error");
  }

  for (int i = 0; i < enc_message_list.size(); i++) {

    try {
     String enc_text_inter = "";
     enc_text_inter = new String(
        enc_message_list.get(i)[0].toBytes(), "UTF-8")
        + new
           String(enc_message_list.get(i)[1].toBytes(),
           "UTF-8")
        + new
           String(enc_message_list.get(i)[2].toBytes(),
           "UTF-8")
        + new
           String(enc_message_list.get(i)[3].toBytes(),
           "UTF-8");
```

```
        enc_text = enc_text + enc_text_inter;
      } catch (UnsupportedEncodingException e) {
        System.out.println("Encoding Error");
      }
    }
    compose_message_text.setText(enc_text);
    Toast.makeText(ComposeMessage.this, "Message
       Encrypted",
       Toast.LENGTH_LONG).show();
  }
 }
}
```

B.3. Decode.java

```
// This is part of "QRDroidServices", by DroidLa.

package certificateless.encryption.mod.app;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.content.res.Configuration;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Toast;

public class Decode extends Activity {

 private static final int ACTIVITY_RESULT_QR_DRDROID = 0;
 private ProgressDialog dialog;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  setContentView(R.layout.decode);

  // Get Spinner instance
  final Spinner spinner = (Spinner)
     findViewById(R.id.spin_complete);
```

```
// "Decode" button
final Button button = (Button)
    findViewById(R.id.button_decode);
// Set action to button
button.setOnClickListener(new OnClickListener() {
 @Override
 public void onClick(View v) {
  // Has the user entered image path?
  String path = ((EditText) findViewById(R.id.txt_path))
     .getText().toString();
  // TODO: This path should not be entered manually by
     the user!

  if (0 == path.trim().length()) {
   Toast.makeText(Decode.this,
      getString(R.string.enter_url),
      Toast.LENGTH_LONG).show();
   return;
  }

  // Create a new Intent to send to QR Droid
  Intent qrDroid = new Intent(Services.DECODE); // Set
     action
                    // "la.droid.qr.decode"

  qrDroid.putExtra(Services.IMAGE, path);

  // Check whether a complete or displayable result is
     needed
  if (spinner.getSelectedItemId() == 0) { // First item
     selected
                 // ("Complete content")
   // Notify we want complete results (default is
      FALSE)
   qrDroid.putExtra(Services.COMPLETE, true);
  }

  // Send intent and wait result
  try {
   startActivityForResult(qrDroid,
      ACTIVITY_RESULT_QR_DRDROID);

   // Wait for result
   if (null == dialog || !dialog.isShowing()) {
    dialog = ProgressDialog.show(Decode.this, "",
       getString(R.string.procesing), true);
    dialog.setCancelable(true);
    dialog.show();
   }
```

```
      } catch (ActivityNotFoundException activity) {
        Services.qrDroidRequired(Decode.this);
      }
    }
  });
 }

 @Override
 /**
  * Reads data decoded from image and returned by QR Droid
  */
 protected void onActivityResult(int requestCode, int
     resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

  // Close dialog
  if (null != dialog && dialog.isShowing()) {
   dialog.cancel();
  }

  if (ACTIVITY_RESULT_QR_DRDROID == requestCode && null !=
      data
     && data.getExtras() != null) {
   // Read result from QR Droid (it's stored in
      la.droid.qr.result)
   String result =
      data.getExtras().getString(Services.RESULT);

   if (resultCode != RESULT_OK || null == result
      || 0 == result.length()) {
    // Image could not been loaded or decoded
    Toast.makeText(Decode.this, R.string.not_decoded,
       Toast.LENGTH_LONG).show();
    return;
   }

   // Just set result to EditText to be able to view it
   ((EditText) findViewById(R.id.result)).setText(result);
  }
 }

 @Override
 public void onConfigurationChanged(Configuration
     newConfig) {
  super.onConfigurationChanged(newConfig);
  // Nothing
 }
}
```

B.4. Encode.java

```
// This is part of "QRDroidServices", by DroidLa.

package certificateless.encryption.mod.app;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.content.res.Configuration;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Spinner;
import android.widget.Toast;

public class Encode extends Activity {

  private static final int ACTIVITY_RESULT_QR_DRDROID = 0;
  private boolean image = false;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.encode);

    // Get Spinner instance
    final Spinner spinner = (Spinner)
        findViewById(R.id.spin_url);

    // "Encode" button
    final Button button = (Button)
        findViewById(R.id.button_encode);
    // Set action to button
    button.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        // Is there something to encode?
        String code = ((EditText) findViewById(R.id.txt_code))
            .getText().toString();
        if (0 == code.trim().length()) {
          Toast.makeText(Encode.this, R.string.enter_code,
              Toast.LENGTH_SHORT).show();
```

```
      return;
    }

    // Create a new Intent to send to QR Droid
    Intent qrDroid = new Intent(Services.ENCODE); // Set
       action
                    // "la.droid.qr.encode"

    // Set text to encode
    qrDroid.putExtra(Services.CODE, code);

    // Check whether an URL or an imge is required
    if (spinner.getSelectedItemId() == 0) { // First item
       selected
                 // ("Get Bitmap")
      // Notify we want complete results (default is
         FALSE)
      image = true;
      qrDroid.putExtra(Services.IMAGE, true);
      // Optionally, set requested image size. 0 means
      // "Fit Screen"
      qrDroid.putExtra(Services.SIZE, 0);
    } else {
      image = false;
    }

    // Send intent and wait result
    try {
      startActivityForResult(qrDroid,
         ACTIVITY_RESULT_QR_DRDROID);
    } catch (ActivityNotFoundException activity) {
      Services.qrDroidRequired(Encode.this);
    }
   }
  });
}

@Override
/**
 * Reads generated QR code
 */
protected void onActivityResult(int requestCode, int
    resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

  if (ACTIVITY_RESULT_QR_DRDROID == requestCode && null !=
     data
     && data.getExtras() != null) {
   // Read result from QR Droid (it's stored in
```

```
        la.droid.qr.result)
    // Result is a string or a bitmap, according what was
        requested
    ImageView imgResult = (ImageView)
        findViewById(R.id.img_result);
    EditText txtResult = (EditText)
        findViewById(R.id.txt_result);

    if (image) {
      String qrCode =
          data.getExtras().getString(Services.RESULT);

      // If image path was not returned, it could not be
          saved. Check
      // SD card is mounted and is writable
      if (null == qrCode || 0 == qrCode.trim().length()) {
        Toast.makeText(Encode.this, R.string.not_saved,
            Toast.LENGTH_LONG).show();
        return;
      }

      // Show success message
      Toast.makeText(Encode.this,
          getString(R.string.saved) + " " + qrCode,
          Toast.LENGTH_LONG).show();

      // Load QR code image from given path
      imgResult.setImageURI(Uri.parse(qrCode));

      imgResult.setVisibility(View.VISIBLE);
      txtResult.setVisibility(View.GONE);

    } else {
      String result =
          data.getExtras().getString(Services.RESULT);
      // Just set result to EditText to be able to view it
      txtResult.setText(result);
      txtResult.setVisibility(View.VISIBLE);
      imgResult.setVisibility(View.GONE);
    }
  }
}

@Override
public void onConfigurationChanged(Configuration
    newConfig) {
  super.onConfigurationChanged(newConfig);
  // Nothing
}
```

```
}
```

B.5. Help.java

```
// This class implements the Help interface

package certificateless.encryption.mod.app;

import android.app.Activity;
import android.os.Bundle;

public class Help extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.help);
  }
}
```

B.6. KeyGenerationCenter.java

```
// This class implements the algorithms run by the KGC

package certificateless.encryption.mod.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.Context;

public class KeyGenerationCenter extends Activity {

  public static Element g, g1, g2, Fu_ID, Fh_BID;
  public static Element d_ID[] = new Element[3];
  private static Element msk, gamma;
  public static List<Element> u = new ArrayList<Element>();
  public static List<Element> v = new ArrayList<Element>();
  public static List<Element> h = new ArrayList<Element>();
  public static String ID, BID;
```

```
public static final int BIT_LENGTH = 100;

// ~~~~~~~~ Step 1 - Setup: Performed by KGC called at
   startup ~~~~~~~~
public static void setup(Context context, CurveParams
   curveParams) {

 Pairing pairing = PairingFactory.getPairing(curveParams);

 g = pairing.getG1().newRandomElement();
 g2 = pairing.getG1().newRandomElement();
 gamma = pairing.getZr().newRandomElement();
 g1 = g.duplicate().powZn(gamma.duplicate());
 msk = g2.duplicate().powZn(gamma.duplicate());

 for (int i = 0; i <= BIT_LENGTH; i++) {
  Element u_temp = pairing.getG1().newRandomElement();
  u.add(u_temp.duplicate());
 }

 for (int i = 0; i <= BIT_LENGTH; i++) {
  Element v_temp = pairing.getG1().newRandomElement();
  v.add(v_temp.duplicate());
 }

 for (int i = 0; i <= BIT_LENGTH; i++) {
  Element h_temp = pairing.getG1().newRandomElement();
  h.add(h_temp.duplicate());
 }
}

// ~~~~~~~~ Step 2 - Extract: Performed by KGC called by
   Receiver ~~~~~~~~
public static Element[] extract(String email, String
   biometrics,
  CurveParams curveParams) {

 Pairing pairing = PairingFactory.getPairing(curveParams);

 Methods method = new Methods();
 ID = method.stringtoBinary(email).substring(0,
    BIT_LENGTH);
 BID = method.stringtoBinary(biometrics).substring(0,
    BIT_LENGTH);

 Fu_ID = method.newHash(u, ID, curveParams);
 Fh_BID = method.newHash(h, BID, curveParams);
```

```
   Element r = pairing.getZr().newRandomElement();

   d_ID[0] =
      msk.duplicate().mul(Fu_ID.duplicate().powZn(r.duplicate()));
   d_ID[1] =
      g.duplicate().mul(Fh_BID).duplicate().powZn(r.duplicate());
   d_ID[2] = Fh_BID.duplicate().powZn(gamma.duplicate());

   return d_ID.clone();
  }
}
```

B.7. Mail.java

```
/* This class provides the functionality for sending mails.
 * The author of this class is John Simon and the code is
    available
 * at http://www.jondev.net/
 * */

package certificateless.encryption.mod.app;

import java.util.Date;
import java.util.Properties;
import javax.activation.CommandMap;
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.activation.MailcapCommandMap;
import javax.mail.BodyPart;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

public class Mail extends javax.mail.Authenticator {
  private String _user;
  private String _pass;

  private String[] _to;
  private String _from;

  private String _port;
```

```
private String _sport;

private String _host;

private String _subject;
private String _body;

private boolean _auth;

private boolean _debuggable;

private Multipart _multipart;

public Mail() {
  _host = "smtp.gmail.com"; // default smtp server
  _port = "465"; // default smtp port
  _sport = "465"; // default socketfactory port

  _user = ""; // username
  _pass = ""; // password
  _from = ""; // email sent from
  _subject = ""; // email subject
  _body = ""; // email body

  _debuggable = false; // debug mode on or off - default
     off
  _auth = true; // smtp authentication - default on

  _multipart = new MimeMultipart();

  // There is something wrong with MailCap, javamail can
     not find a
  // handler for the multipart/mixed part, so this bit
     needs to be added.
  MailcapCommandMap mc = (MailcapCommandMap) CommandMap
     .getDefaultCommandMap();
  mc.addMailcap("text/html;;
     x-java-content-handler=com.sun.mail.handlers.text_html");
  mc.addMailcap("text/xml;;
     x-java-content-handler=com.sun.mail.handlers.text_xml");
  mc.addMailcap("text/plain;;
     x-java-content-handler=com.sun.mail.handlers.text_plain");
  mc.addMailcap("multipart/*;;
     x-java-content-handler=com.sun.mail.handlers.multipart_mixed");
  mc.addMailcap("message/rfc822;;
     x-java-content-handler=com.sun.mail.handlers.message_rfc822");
  CommandMap.setDefaultCommandMap(mc);
}
```

```
public Mail(String user, String pass) {
  this();

  _user = user;
  _pass = pass;
}

public boolean send() throws Exception {
  Properties props = _setProperties();

  if (!_user.equals("") && !_pass.equals("") && _to.length
      > 0
     && !_from.equals("") && !_subject.equals("")
     && !_body.equals("")) {
    Session session = Session.getInstance(props, this);

    MimeMessage msg = new MimeMessage(session);

    msg.setFrom(new InternetAddress(_from));

    InternetAddress[] addressTo = new
       InternetAddress[_to.length];
    for (int i = 0; i < _to.length; i++) {
     addressTo[i] = new InternetAddress(_to[i]);
    }
    msg.setRecipients(MimeMessage.RecipientType.TO,
       addressTo);

    msg.setSubject(_subject);
    msg.setSentDate(new Date());

    // setup message body
    BodyPart messageBodyPart = new MimeBodyPart();
    messageBodyPart.setText(_body);
    _multipart.addBodyPart(messageBodyPart);

    // Put parts in message
    msg.setContent(_multipart);

    // send email
    Transport.send(msg);

    return true;
  } else {
    return false;
  }
}

public void addAttachment(String filename) throws
```

```
    Exception {
  BodyPart messageBodyPart = new MimeBodyPart();
  DataSource source = new FileDataSource(filename);
  messageBodyPart.setDataHandler(new DataHandler(source));
  messageBodyPart.setFileName(filename);

  _multipart.addBodyPart(messageBodyPart);
}

@Override
public PasswordAuthentication getPasswordAuthentication()
    {
  return new PasswordAuthentication(_user, _pass);
}

private Properties _setProperties() {
  Properties props = new Properties();

  props.put("mail.smtp.host", _host);

  if (_debuggable) {
    props.put("mail.debug", "true");
  }

  if (_auth) {
    props.put("mail.smtp.auth", "true");
  }

  props.put("mail.smtp.port", _port);
  props.put("mail.smtp.socketFactory.port", _sport);
  props.put("mail.smtp.socketFactory.class",
     "javax.net.ssl.SSLSocketFactory");
  props.put("mail.smtp.socketFactory.fallback", "false");

  return props;
}

// the getters and setters
public String getBody() {
  return _body;
}

public void setBody(String _body) {
  this._body = _body;
}

public void setTo(String[] toArr) {
  this._to = toArr;
}
```

```
  public void setFrom(String string) {
    this._from = string;
  }

  public void setSubject(String string) {
    this._subject = string;
  }
}
```

B.8. Methods.java

```
// This class implements various methods used by the
    different classes

package certificateless.encryption.mod.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.security.MessageDigest;
import java.util.List;

public class Methods {

  public static final int BIT_LENGTH = 100;

  public String stringtoBinary(String text) {

    byte[] bytes = text.getBytes();
    String binarystr = "";
    for (int i = 0; i < bytes.length; i++) {
      binarystr = binarystr + bytetoBinaryString(bytes[i]);
    }
    return binarystr;
  }

  // Computing the SHA-1 hash
  public String sha1Hash(String input) {

    byte[] output;
    String binarystr = "";

    try {
      MessageDigest md = MessageDigest.getInstance("SHA1");
```

```
    md.update(input.getBytes());
    output = md.digest();

    for (int i = 0; i < output.length; i++) {
      binarystr = binarystr + bytetoBinaryString(output[i]);
    }

  } catch (Exception e) {
    System.out.println("Exception: " + e);
  }
  return binarystr;
}

/*
 * Converting bytes to hexadecimal. The code for this
     method has been taken
 * from http://www.herongyang.com
 */
public String bytesToHex(byte[] b) {
  char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6',
      '7', '8', '9',
      'A', 'B', 'C', 'D', 'E', 'F' };
  StringBuffer buf = new StringBuffer();
  for (int j = 0; j < b.length; j++) {
    buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
    buf.append(hexDigit[b[j] & 0x0f]);
  }
  return buf.toString();
}

/*
 * Converting a byte to binary string. The code for this
     method has been
 * taken from http://helpdesk.objects.com.au
 */
public String bytetoBinaryString(byte n) {
  StringBuilder sb = new StringBuilder("00000000");
  for (int bit = 0; bit < 8; bit++) {
    if (((n >> bit) & 1) > 0) {
      sb.setCharAt(7 - bit, '1');
    }
  }
  return sb.toString();
}

// Calculating the hash used by the scheme
public Element newHash(List<Element> vector, String
    bitstr,
    CurveParams curveParams) {
```

```
    Pairing pairing = PairingFactory.getPairing(curveParams);
    Element hash_val = pairing.getG1().newOneElement();
    hash_val.mul(vector.get(0).duplicate());

    for (int i = 0; i < BIT_LENGTH; i++) {
      if (bitstr.charAt(i) == '1') {
        hash_val.mul(vector.get(i + 1).duplicate());
      }
    }
    return hash_val;
  }
}
```

B.9. ReadMessage.java

```
//This class implements the interface to read messages

package certificateless.encryption.mod.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;

import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.text.method.ScrollingMovementMethod;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;
import android.widget.Toast;

public class ReadMessage extends Activity implements
    OnClickListener {

  TextView read_msg_txt;
  private static final int ACTIVITY_RESULT_QR_DRDROID = 0;
  public static final int BIT_LENGTH = 100;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
  setContentView(R.layout.read_message);

  read_msg_txt = (TextView)
     findViewById(R.id.read_message_text);

  View decButton = findViewById(R.id.read_message_dec);
  decButton.setOnClickListener(this);

  View loadButton = findViewById(R.id.read_message_load);
  loadButton.setOnClickListener(this);
}

public void onClick(View v) {

  final TextView read_message_text = (TextView)
     findViewById(R.id.read_message_text);

  switch (v.getId()) {

  case R.id.read_message_dec:

    Intent qrDroid = new Intent(Services.SCAN);

    try {
      startActivityForResult(qrDroid,
         ACTIVITY_RESULT_QR_DRDROID);
    } catch (ActivityNotFoundException activity) {
      Services.qrDroidRequired(ReadMessage.this);
    }
    break;

  case R.id.read_message_load:
    read_msg_txt.setMovementMethod(new
       ScrollingMovementMethod());
    read_message_text.setText(ComposeMessage.enc_text);
    break;
  }
}

protected void onActivityResult(int requestCode, int
    resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

  if (ACTIVITY_RESULT_QR_DRDROID == requestCode && null !=
     data
     && data.getExtras() != null) {

    String encoding = "UTF-16BE";
    int message_len = 76;
```

```
CurveParams curveParams = new
    CurveParams().load(getResources()
    .openRawResource(R.raw.a_181_603));
List<Element[]> enc_message_list = new
    ArrayList<Element[]>();
enc_message_list = ComposeMessage.enc_message_list;

final TextView read_message_text = (TextView)
    findViewById(R.id.read_message_text);

// Read result from QR Droid (it's stored in
    la.droid.qr.result)
String biometrics =
    data.getExtras().getString(Services.RESULT);
Methods method = new Methods();

String BID =
    method.stringtoBinary(biometrics).substring(0,
    BIT_LENGTH);

Element Fh_BID = method.newHash(KeyGenerationCenter.h,
    BID,
    curveParams);

String dec_msg_padded = "";
for (int i = 0; i < enc_message_list.size(); i++) {
  Element dec_message = Receiver.deCrypt(Fh_BID,
     enc_message_list.get(i), curveParams);

  try {
    String msg_inter = new
       String(dec_message.duplicate()
       .toBytes(), encoding);
    dec_msg_padded = dec_msg_padded + msg_inter;

  } catch (UnsupportedEncodingException e) {
    System.out.println("Encoding Error");
  }
}

String msg_end_num = dec_msg_padded.substring(
    dec_msg_padded.length() - 2,
       dec_msg_padded.length());

String dec_msg = dec_msg_padded;

try {
  int dec_msg_pad_len = Integer.parseInt(msg_end_num);
  dec_msg = dec_msg_padded.substring(0,
```

```
          dec_msg_padded.length()
          - dec_msg_pad_len - message_len);
      }

      catch (NumberFormatException nFE) {
        System.out.println("Not an Integer");
        dec_msg = "***PROBLEM DECRYPTING***\n\n\n" + dec_msg;
      }

      read_message_text.setText(dec_msg);

      Toast.makeText(ReadMessage.this, "Message Decrypted",
         Toast.LENGTH_LONG).show();
    }
  }
}
```

B.10. Receiver.java

```
// This class implements the algorithms executed by the
    Receiver

package certificateless.encryption.mod.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

public class Receiver {

  private static Element x_ID;
  public static Element pk_ID[] = new Element[3];
  private static Element sk_ID[] = new Element[2];

  public static final int BIT_LENGTH = 100;

  // Step 3 & 4 - SetSec & SetPub: Performed and called by
     Receiver
  public static void setSecPub(CurveParams curveParams) {

    Pairing pairing = PairingFactory.getPairing(curveParams);

    x_ID = pairing.getZr().newRandomElement();

    pk_ID[0] =
       KeyGenerationCenter.g.duplicate().powZn(x_ID.duplicate());
```

```
  pk_ID[1] =
     KeyGenerationCenter.g1.duplicate().powZn(x_ID.duplicate());
  pk_ID[2] = KeyGenerationCenter.d_ID[2].duplicate().powZn(
     x_ID.duplicate());
}

// ~~~~~~~~ SetPriv: Performed and called by Receiver
    ~~~~~~~~
private static void setPriv(Element Fh_BID, CurveParams
    curveParams) {

  Pairing pairing = PairingFactory.getPairing(curveParams);

  Element r_prime = pairing.getZr().newRandomElement();

  sk_ID[0] =
     (KeyGenerationCenter.d_ID[0].duplicate().powZn(x_ID
     .duplicate())).mul(KeyGenerationCenter.Fu_ID.duplicate().powZn(
     r_prime.duplicate()));
  sk_ID[1] =
     (KeyGenerationCenter.d_ID[1].duplicate().powZn(x_ID
     .duplicate())).mul(KeyGenerationCenter.g.duplicate()
     .mul(Fh_BID.duplicate()).powZn(r_prime.duplicate()));
}

// ~~~~~~~ Step 6 - Decrypt: Performed by Receiver ~~~~~
public static Element deCrypt(Element Fh_BID, Element
    cipherText[],
    CurveParams curveParams) {

  Pairing pairing = PairingFactory.getPairing(curveParams);

  setPriv(Fh_BID, curveParams);

  String str_to_hash = cipherText[0].duplicate().toString()
     + cipherText[1].duplicate().toString()
     + cipherText[2].duplicate().toString()
     + Receiver.pk_ID[0].duplicate().toString()
     + Receiver.pk_ID[1].duplicate().toString()
     + Receiver.pk_ID[2].duplicate().toString()
     + KeyGenerationCenter.ID;

  Methods method = new Methods();
  String w = method.sha1Hash(str_to_hash);

  Element Fv_W = method.newHash(KeyGenerationCenter.v,
     w.substring(0, BIT_LENGTH), curveParams);

  Element temp1 =
```

```
      KeyGenerationCenter.Fu_ID.duplicate().mul(
      Fv_W.duplicate());

   Element temp2 = cipherText[2].duplicate()
      .mul(cipherText[3].duplicate());

   if (pairing
      .pairing(cipherText[1].duplicate(), temp1.duplicate())
      .isEqual(
         pairing.pairing(
            KeyGenerationCenter.g.duplicate().mul(
               KeyGenerationCenter.Fh_BID.duplicate()),
            temp2.duplicate())) == false) {

     System.out.println("ABORT: problem matching ...");

     return pairing.getGT().newZeroElement();
   } else {

     Element dec_msg = cipherText[0].duplicate().mul(
        pairing.pairing(cipherText[2].duplicate(),
           sk_ID[1].duplicate()).div(
           pairing.pairing(cipherText[1].duplicate(),
              sk_ID[0].duplicate())));

     return dec_msg.duplicate();
   }
  }
}
```

B.11. Scan.java

```
// This is part of "QRDroidServices", by DroidLa.

package certificateless.encryption.mod.app;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.content.res.Configuration;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
```

```
public class Scan extends Activity {

  private static final int ACTIVITY_RESULT_QR_DRDROID = 0;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.scan);

    // Get Spinner instance
    final Spinner spinner = (Spinner)
        findViewById(R.id.spin_complete);

    // "Scan" button
    final Button button = (Button)
        findViewById(R.id.button_scan);
    // Set action to button
    button.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        // Create a new Intent to send to QR Droid
        Intent qrDroid = new Intent(Services.SCAN); // Set
            action
                        // "la.droid.qr.scan"

        // Check whether a complete or displayable result is
            needed
        if (spinner.getSelectedItemId() == 0) { // First item
            selected
                    // ("Complete content")
          // Notify we want complete results (default is
              FALSE)
          qrDroid.putExtra(Services.COMPLETE, true);
        }

        // Send intent and wait result
        try {
          startActivityForResult(qrDroid,
              ACTIVITY_RESULT_QR_DRDROID);
        } catch (ActivityNotFoundException activity) {
          Services.qrDroidRequired(Scan.this);
        }
      }
    });
  }

  @Override
  /**
```

```
   * Reads data scanned by user and returned by QR Droid
   */
  protected void onActivityResult(int requestCode, int
      resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (ACTIVITY_RESULT_QR_DRDROID == requestCode && null !=
        data
       && data.getExtras() != null) {
      // Read result from QR Droid (it's stored in
         la.droid.qr.result)
      String result =
         data.getExtras().getString(Services.RESULT);
      // Just set result to EditText to be able to view it
      EditText resultTxt = (EditText)
         findViewById(R.id.result);
      resultTxt.setText(result);
      resultTxt.setVisibility(View.VISIBLE);
    }
  }

  @Override
  public void onConfigurationChanged(Configuration
      newConfig) {
    super.onConfigurationChanged(newConfig);
  }
}
```

B.12. Sender.java

```
// This class implements the algorithms executed by the
    Sender

package certificateless.encryption.mod.app;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

public class Sender {

  public static Element C[] = new Element[4];
  public static Element Fu_ID;
  public static final int BIT_LENGTH = 100;

  // ~~~~~~~~~~ Step 5 - Encrypt: Performed and called by
```

```
    Sender ~~~~~~~~~~
public Element[] enCrypt(String ID, CurveParams
    curveParams,
    Element Fh_BID, Element m) {

  Pairing pairing = PairingFactory.getPairing(curveParams);

  if (pairing.pairing(Receiver.pk_ID[0].duplicate(),
     KeyGenerationCenter.g1.duplicate()).isEqual(
     pairing.pairing(Receiver.pk_ID[1].duplicate(),
        KeyGenerationCenter.g.duplicate())) == false) {

    System.out.println("ABORT: Incorrect Shape");

    return null;
  } else {

    Element s = pairing.getZr().newRandomElement();

    C[0] = m.duplicate().mul(
       (pairing.pairing(
          Receiver.pk_ID[1].duplicate().mul(
             Receiver.pk_ID[2].duplicate()),
          KeyGenerationCenter.g2.duplicate())).powZn(s
          .duplicate()));

    C[1] =
       KeyGenerationCenter.g.duplicate().mul(Fh_BID.duplicate())
       .powZn(s.duplicate());

    Methods method = new Methods();

    Element Fu_ID = method.newHash(KeyGenerationCenter.u,
       ID,
       curveParams);

    C[2] = Fu_ID.duplicate().powZn(s.duplicate());

    String str_to_hash = C[0].duplicate().toString()
       + C[1].duplicate().toString() +
          C[2].duplicate().toString()
       + Receiver.pk_ID[0].duplicate().toString()
       + Receiver.pk_ID[1].duplicate().toString()
       + Receiver.pk_ID[2].duplicate().toString() + ID;

    String w = method.sha1Hash(str_to_hash);

    Element Fv_W = method.newHash(KeyGenerationCenter.v,
       w.substring(0, BIT_LENGTH), curveParams);
```

```
    C[3] = Fv_W.duplicate().powZn(s.duplicate());

    return C.clone();
   }
  }
}
```

B.13. Services.java

```
package certificateless.encryption.mod.app;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.TabActivity;
import android.content.ActivityNotFoundException;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.res.Configuration;
import android.content.res.Resources;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TabHost;

/**
 * Shows three Tabs with options to Scan, Decode and Encode
    QR codes using
 * services provided by "QR Droid"
 *
 *
 * This is part of "QRDroidServices", by DroidLa. If you're
    creating an Android app
 * which uses one or more services provided by "QR Droid",
    you can use this code for
 * free, and modify it as you need, for personal and
    commercial use.
 *
 * Any other use of this code is forbidden.
 *
 * @author DroidLa
 * @version 1.0
 */
public class Services extends TabActivity {

  //Actions
  public static final String SCAN = "la.droid.qr.scan";
  public static final String ENCODE = "la.droid.qr.encode";
```

```
public static final String DECODE = "la.droid.qr.decode";

//Parameters
//SCAN / DECODE
public static final String COMPLETE =
    "la.droid.qr.complete"; //Default: false
//ENCODE
public static final String CODE = "la.droid.qr.code";
    //Required
public static final String SIZE = "la.droid.qr.size";
    //Default: Fit screen
//ENCODE / DECODE
public static final String IMAGE = "la.droid.qr.image";
    //Default for encode: false / Required for decode

//Result
public static final String RESULT = "la.droid.qr.result";

  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.tabs);

     //Recycled objects
   Resources res = getResources();
   TabHost tabHost = getTabHost();
   TabHost.TabSpec spec;
   Intent intent;

   //Scan Activity
   intent = new Intent().setClass(this, Scan.class);
   spec = tabHost.newTabSpec("Scan").setIndicator("",
      res.getDrawable(R.drawable.camera)).setContent(intent);
   tabHost.addTab(spec);

   //Encode Activity
   intent = new Intent().setClass(this, Encode.class);
   spec = tabHost.newTabSpec("Encode").setIndicator("",
      res.getDrawable(R.drawable.text)).setContent(intent);
   tabHost.addTab(spec);

   //Decode Activity
   intent = new Intent().setClass(this, Decode.class);
   spec = tabHost.newTabSpec("Decode").setIndicator("",
      res.getDrawable(R.drawable.image)).setContent(intent);
   tabHost.addTab(spec);

   //Show DEMO alert dialog
   AlertDialog.Builder builder = new
```

```
     AlertDialog.Builder(this);
builder.setMessage( getString(R.string.demo) )
      .setCancelable(true)
      .setNegativeButton( R.string.close, new
         DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int
            id) {
          dialog.cancel();
         }
      })
      .setNeutralButton( R.string.source, new
         DialogInterface.OnClickListener() {
     @Override
     public void onClick(DialogInterface dialog, int
        which) {
       startActivity( new Intent( Intent.ACTION_VIEW,
          Uri.parse( getString(R.string.url_source) ) )
          );
     }
   })
      .setPositiveButton( R.string.qrDroid, new
         DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int
           id) {
         try {
          startActivity( new Intent( Intent.ACTION_VIEW,
             Uri.parse( getString(R.string.url_market)
             ) ) );
          finish();
         } catch (ActivityNotFoundException e) {
          startActivity( new Intent( Intent.ACTION_VIEW,
             Uri.parse( getString(R.string.url_direct)
             ) ) );
          finish();
         }
         }
      });
builder.create().show();
}

/**
 * Display a message stating that QR Droid is requiered,
    and lets the user download it for free
 * @param activity
 */
public static void qrDroidRequired( final Activity
   activity ) {
 //Apparently, QR Droid is not installed, or it's
    previous to version 3.5
```

```
   AlertDialog.Builder builder = new
      AlertDialog.Builder(activity);
   builder.setMessage(
      activity.getString(R.string.qrdroid_missing) )
        .setCancelable(true)
        .setNegativeButton(
           activity.getString(R.string.cancel), new
           DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int
              id) {
            dialog.cancel();
           }
        })
        .setPositiveButton(
           activity.getString(R.string.from_market), new
           DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int
             id) {
           activity.startActivity( new Intent(
              Intent.ACTION_VIEW, Uri.parse(
              activity.getString(R.string.url_market) ) )
              );
           }
        })
          .setNeutralButton(activity.getString(R.string.direct),
             new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int
              id) {
            activity.startActivity( new Intent(
               Intent.ACTION_VIEW, Uri.parse(
               activity.getString(R.string.url_direct) )
               ) );
           }
         });
   builder.create().show();
   }

   @Override
   public void onConfigurationChanged(Configuration
      newConfig) {
     super.onConfigurationChanged(newConfig);
   }
}
```

B.14. Setup.java

```
// This class implements the interface to generate keys
```

```
   after entering email id

package certificateless.encryption.mod.app;

import it.unisa.dia.gas.plaf.jpbc.pairing.CurveParams;
import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.EditText;
import android.widget.Toast;

public class Setup extends Activity implements
   OnClickListener {

 private static final int ACTIVITY_RESULT_QR_DRDROID = 0;
 CurveParams curveParams;

 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.setup);

  View generateButton = findViewById(R.id.setup_generate);
  generateButton.setOnClickListener(this);
 }

 public void onClick(View v) {

  // Generating Keys
  switch (v.getId()) {
  case R.id.setup_generate:

    curveParams = new CurveParams().load(getResources()
       .openRawResource(R.raw.a_181_603));

    KeyGenerationCenter.setup(v.getContext(), curveParams);

    Intent qrDroid = new Intent(Services.SCAN);

    try {
     startActivityForResult(qrDroid,
        ACTIVITY_RESULT_QR_DRDROID);
    } catch (ActivityNotFoundException activity) {
     Services.qrDroidRequired(Setup.this);
    }
    break;
```

```
    }
  }

  @Override
  /**
   * Reads data scanned by user and returned by QR Droid
   */
  protected void onActivityResult(int requestCode, int
      resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (ACTIVITY_RESULT_QR_DRDROID == requestCode && null !=
        data
       && data.getExtras() != null) {
      // Read result from QR Droid (it's stored in
         la.droid.qr.result)
      String biometrics =
         data.getExtras().getString(Services.RESULT);

      final EditText setup_message_to = (EditText)
         findViewById(R.id.setup_message_to);

      KeyGenerationCenter.d_ID = KeyGenerationCenter.extract(
         setup_message_to.getText().toString(), biometrics,
         curveParams);
      Receiver.setSecPub(curveParams);

      Toast.makeText(Setup.this, "Setup Complete",
         Toast.LENGTH_LONG)
         .show();

      Intent i = new Intent(this,
         CertificatelessEncModAppActivity.class);
      startActivity(i);
    }
  }
}
```