POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRONIKI I TECHNIK INFORMACYJNYCH
INSTYTUT INFORMATYKI

Rok akademicki 2008/2009

**PRACA DYPLOMOWA INŻYNIERSKA**

Kacper Bąk

# Certificateless Cryptography

Opiekun pracy
dr inż. Artur Krystosik

Ocena .....

.....
Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

## ABSTRACT

This paper presents project and implementation of certificateless cryptography, and it's relationship with public-key cryptography. The scheme comes from identity-based cryptography and improves some of its weaknesses. Moreover, it enables sender to encrypt a message when the only information she knowns is recipient's identity (e.g. email address). Additionally, users do not need certificates to bind identity with specific public key. Certificateless cryptography may be employed to provide transparent email encryption, which is desirable in real-world security applications.

Software created for the purpose of this work provides both infrastructure (application servers) and clients who wish to communicate confidentially. Users simulate email by sending encrypted messages to each other. The paper describes software architecture, modules, application-level protocols and tests.



**Keywords:** *cryptography, certificate, CL-PKE, PKI, ID-PKC, IBE, ECC, PBC*


---


## KRYPTOGRAFIA BEZCERTYFIKATOWA

Praca przedstawia projekt i implementację kryptografii bezcertyfikatowej oraz jej miejsce w kryptografii klucza publicznego. Opisywany schemat stanowi rozwinięcie kryptografii opartej na tożsamościach. Cechą charakterystyczną jest możliwość zaszyfrowania wiadomości znając tylko daną tożsamość (np. adres email) odbiorcy, a także brak certyfikatów dla użytkowników kryptosystemu. Kryptografia bezcertyfikatowa umożliwia realizację przezroczystego szyfrowania poczty elektronicznej, co ma znaczenie w praktycznych realizacjach usług bezpieczeństwa.

Zaprojektowane oprogramowanie składa się zarówno z infrastruktury (odpowiednich serwerów) jak i z uczestników, którzy pragną zachować poufność komunikacji. Wiadomości wymieniane między użytkownikami symulują pocztę elektroniczną. W pracy została opisana architektura systemu, moduły, zastosowane protokoły i testy.



**Słowa kluczowe:** *kryptografia, certyfikat, CL-PKE, PKI, ID-PKC, IBE, ECC, PBC*

Serdeczne podziękowania
za okazaną pomoc i życzliwość
składam mojemu promotorowi
dr inż. Arturowi Krystosikowi

# Contents

# Chapter 1

# Introduction

There is no doubt that cryptography plays significant role in modern world. Wide use of cryptography gives people access to confidential communication, electronic commerce and secure data storage. While history of secrets goes back to Ancient Egypt and Rome, it had strong impact on The Second World War and, perhaps, is even more important now, in the era of globalisation.

The classical aim of cryptography was to make encrypted content unreadable to anyone but the two parties who agreed to use some specific scheme. Nowadays, cryptography provides more sophisticated services, such as message integrity, authentication, time stamping etc. If we consider communication secrecy, then we can divide cryptography into two areas: private-key (symmetric-key) and public-key (asymmetric-key) techniques. The first has been in use since the ancient times, while the second appeared in the second half of the 20th century.

Public-key cryptography (PKC) allows parties to set up secure transmission channel with no prior exchange of secret keys. Each user generates a pair of keys called public and private key. The former is used for encryption and the latter for decryption. This groundbreaking idea solves the issue of key distribution and reduces the number of required crypto-keys. In fact it shifts the problem of key distribution to the problem of binding user with his key pair. This binding is really at the core of PKC security. Fortunately, in practice, it is much easier to certify particular binding than to deliver the keys themselves. There are several methods of which public-key infrastructure is the best known.

Public-key infrastructure (PKI) proves authenticity of users' keys by means of certificates. On the organizational level it is a set of authorities which handle digital certificates. Although, PKI is often the choice, it has significant drawbacks. Firstly, the infrastructure is heavy-weight and rather expensive. Moreover, certificates must be verified by users (whether they match correct identity), but non-technical users usually have problems with that. Another drawback is revocation of old/compromised keys. So called Certificate Revocation Lists (CRL's) might grow rapidly and become awkward to manage. This is usually the case in big deployments of PKI.

Certificateless cryptography (CL-PKE) is an interesting alternative to traditional PKI. It makes use of identities, which are users' public keys formed of arbitrary strings, in place of

certificates. Besides, it's infrastructure is lightweight and can be deployed at much lower cost. Moreover, it offers transparent encryption, so that non-technical users could easily secure their data.

The main aim of this work was implementation of certificateless cryptography in the context of transparent email encryption. At the time of writing dissertation, there are still no widely-available implementations of this scheme. As it will become apparent, certificateless cryptography has got interesting properties and can provide flexibility not found in currently employed methods.

# Chapter 2

# Background

The chapter introduces the field of public-key cryptography and outlines various problems specific to this area. Furthermore, it discusses several methods of guaranteeing authenticity of public-keys and indicates how certificateless cryptography relates to them. Besides problem analysis, it also presents underlying mathematics of elliptic curve cryptography (ECC) and pairing-based cryptography, which are crucial for understanding CL-PKE scheme.

## 2.1 Problem Analysis

Providing authenticity of public keys is one of the hardest problems within public-key cryptography. There are no obvious solutions for this issue, because all of them have some drawbacks and limitations. Fortunately, there are several approaches to the stated problem, and all of them concentrate around one idea: trust. Public-key cryptography is used when two parties are not able to meet and exchange crypto-keys directly; which is almost always the case in computer networks. So, if people do not know each other, how can they know that certain key belongs to the person who claims so? The answer is that, in general case, they cannot know this fact just from looking at the key. There must be some other parties who know the two users and who are trusted by them. This crucial observation leads to several different realizations of the trusted third party idea.

The two general approaches can be described as decentralized and centralized trusted third parties. The former is somewhat similar to peer-to-peer networks, where every node has equal capabilities and rights. The latter resembles traditional client-server model, where server plays central role and provides services to clients.

There are universal questions concerning both models of trust. Even though, decentralized trust might seem more fair and reliable, it may be vulnerable to manipulation and can discriminate against new users. On the other hand, centralized trust relies on single points of failure (servers) and might raise doubts whether particular party should be trusted, to what extent, and finally, why. To conclude, one needs to decide which model of trust fits better her needs and should be aware of far-reaching consequences of the decision.

### 2.1.1 Web of Trust

Web of trust is the most common implementation of decentralized trusted third party. Its infrastructure is very simple, since there are no additional nodes besides users. Users collect public keys of others and certify them if they are sure that particular key belongs to particular person. When two parties do not know each other, they must know other people who belong to trusted chain and can provide a link between the two parties. Of course, every person in the chain must be trusted by the participants.

An example of web of trust can be found in Fig. 2.1. When Alice obtains Bob's key, she can verify that it really belongs to Bob. Alice trusts Isabel and Isabel trusts Bob, therefore Alice can trust Bob. On the other hand, if Alice receives Andrew's key, there is no way in which she can bind Andrew with the key, since there is no one who trusts Andrew, although Andrew trusts Alice, Bob and Adam.

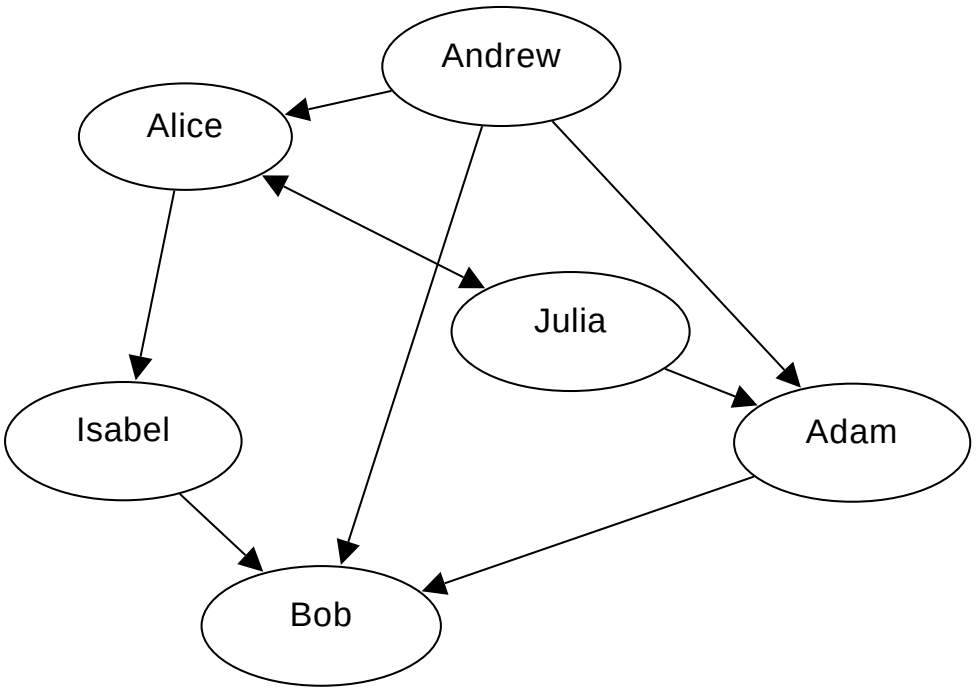


Figure 2.1: Sample web of trust.

Due to lack of single certificate authority users are made to collect public-keys and certificates on their own machines. The concept of web of trust is employed by OpenPGP systems. This model of trust is very flexible and gives users much freedom when it comes to certification. Every user decides himself whether to accept or reject certain key. OpenPGP was initially meant to protect personal emails, but nowadays it is used to encrypt/sign files and even the whole drives. It is especially popular among more advanced users coming from academia or open source movement. Companies usually prefer centralized models, and for this reason do not incorporate web of trust.

#### Advantages

Decentralization of trust and certificate management facilities make web of trust a very reliable mechanism. It does not contain a single point of failure and does not require expensive infrastructure since all operations are performed on users' machines. Moreover, flexibility of web of trust is beyond the scope of centralized systems. Highly appreciated by freedom-fighters, provides strong cryptography for individuals with almost no cost in terms of money.

#### Drawbacks

Web of trust requires a lot of trust between pairs of users, and consequently, might be vulnerable to various manipulations. For example, let's say there is a relation of trust between Andrew and Isabel as in Fig. 2.1, and Alice did not verify Isabel's key but intentionally accepted a flawed one. Now the chain of trust is susceptible to attack since someone can read Andrew's message addressed to Isabel without their permission. This problem could be solved by voting if there were more chains of trust between Andrew and Isabel (bypassing Alice) or by using a separate channel so that Andrew can check Isabel's key.

What is more, users must spend their time and gain some knowledge on certification, which often causes problems to newcomers, because it demands some experience and a lot of care. Ease of use might be very problematic to non-technical people because users have to perform the same tasks as servers in centralized schemes. According to a very informative paper [14] only four out of twelve motivated and experienced email users managed to correctly send encrypted email using PGP 5.0. Although the paper concerns rather old version of software, the results are likely to stand as handling basic tasks (i.e. encryption, publishing/getting keys, key verification) did not change much over time.

Finally, people who live in remote areas and do not have many friends, might seem less credible, and thus discriminated. This can be problematic, because it can be hard for a person to be trusted right away and fully benefit from using security software unless there are other users who can introduce the person.

In conclusion, if cryptography is to be used by masses, it should be transparent and easy to use. All of the mentioned issues can be, more or less, solved by centralized schemes, such as public-key infrastructure.

### 2.1.2 Public-Key Infrastructure

Public-key infrastructure is the most popular solution for proving authenticity of public keys. Similarly to web of trust, it applies certificates to confirm relation between user and his public key. On the other hand, PKI's model is centralized and hierarchical, composed of special nodes called Registration Authority (RA) and Certificate Authority (CA), who make up the infrastructure. The authorities are trusted third parties, which are not run by ordinary users.

The role of RA and CA is as follows. Registration Authority collects requests from users to issue digital certificate for their public keys. However, before CA can sign the key, Registration Authority must verify credentials of the client. Upon successful verification, Certificate Authority generates certificate, which contains user's key, identity and CA's signature.

The fundamental question is: who granted CA the right to authorize users' keys? Usually it is another Certificate Authority, who is even more trusted. PKI forms hierarchical structure in which CA's keys are further signed by other CA's. However, root Certificate Authorities sign their keys themselves and for this reason those certificates are usually deployed with software.

### Advantages

Public-key infrastructure is well-known for its scalability and high level of security if used correctly. PKI is able to guarantee privacy, authentication, integrity and non-repudiation services to its users. It was deployed on large scale in many organizations and is the most common cryptography-related feature on the Internet. Therefore, many issues were precisely identified and addressed, such as:

- securing Certificate Authorities, which are PKI's single points of failure,
- careful identity checking of the certificate holder to create valid certificate,
- moving from Certificate Revocation Lists to on-line status query mechanisms to avoid computational and bandwidth overhead,
- naming semantics in certificates by moving from global to more local structures to avoid name ambiguity,

Moreover, many people find public-key infrastructure relatively easy to use because they do not have to perform tedious tasks, such as finding a link of trust or manual key certification. In contrary to web of trust, a person who receives a newly created certificate can forthwith fully benefit from using strong cryptography.

### Drawbacks

Although, PKI works in practice, it has significant drawbacks. Evidently, PKI does not solve many problems as it was expected to do. Firstly, the infrastructure is heavy-weight and rather expensive, because requires trusted authorities to obey strict security policies. These rules concern both digital and physical security measures to protect Certificate Authority from compromise.

Moreover, certificates must be verified by users (whether they match correct identity), but non-technical users usually fail to do it right. Even though the checking process is not that complicated (e.g comparing web address with holder specified in certificate) many people

do not perform it. One solution is to make software do the checks but this approach has its limitations in real-world settings. Perhaps another solution, yet more radical, would be to completely abandon certificates as it happens in identity-based encryption.

Another drawback is certificate management and revocation of old/compromised keys. The main problem of key validation is how to quickly check whether particular key is up-to-date. Much of computational overhead comes from validation of full certification paths. So called Certificate Revocation Lists are original mechanism utilized by Public-Key Infrastructure. In practice, they might grow rapidly and become awkward to manage, which is usually the case in big deployments of PKI. Alternatively, there exists an on-line approach which moves much of the validation overhead from clients to dedicated servers that constitute to additional infrastructure.

Finally, if certificate is to match particular person, it must contain personal details such as name, surname, but in global world this still might not be precise enough, because names can repeat. The problem can be resolved by leaning towards local namespaces and providing more identity details. Unfortunately, the latter can lead to privacy compromise and might not be accepted by individual users. A different approach might be to use simple identities (e.g. email address) in place of proper names. More issues related with PKI are described in [7].

A striking example of improper PKI deployment are SSL certificates, which are synonym of secure e-commerce. First of all, in contrary to popular belief concerning web security, they only guarantee that public key matches specified URL address. Companies often outsource financial matters to other companies, and when user connects with the original web-site, she is later redirected to SSL protected web-page whose holder field in certificate does not match the original site. Users and web browsers accept this fact, even if it directly contradicts the idea of SSL certificates. The next point in SSL discussion is the method of certificate deployment. As mentioned earlier root certificates come with software, because they cannot be reliably fetched over unsecured protocol. Therefore, web browsers usually come with a set of certificates. The problem is, by default one downloads the browser over unsecured HTTP protocol, which can result in man-in-the-middle attack both on the browser and enclosed certificates. Producers of web browsers could easily resolve the problem but they tend to ignore it. To sum up, issues with SSL do not lie in certificates but rather in implementation of the PKI idea.

To conclude, public-key infrastructure automates many tasks which must be handled manually by web of trust users. The best solution would be to make security-related features totally transparent, so that people could use them without any particular knowledge or extra checks. Schemes which offer this kind of functionality can be built upon identity-based cryptography (ID-PKC).

### 2.1.3 Identity-Based Encryption

Aims of public key infrastructure and identity-based encryption (IBE) are quite similar but the way they approach certain problems are slightly different. First of all, users of IBE can set public key to be an arbitrary, yet unique, string. Therefore, the key can be

something easily memorable like an email address or a phone number. Secondly, there are no certificates binding user with his public key; Bob's unique ID string guarantees that no user besides him should be able to decrypt the content.

The above cryptosystem was firstly proposed by Shamir [13] in 1984 primarily to simplify certificate management in email systems. Besides, IBE allows to implement transparent data encryption in various communication systems, such as email or cellular telephony. Although transparent encryption is not important from theoretical point of view, it is a significant factor in real-world implementations. Usually non-technical users have no knowledge on computer security and for this reason misuse security-related software. Phishing, perhaps, is the best example of how to deceive average user and make him reveal sensitive data even if the connection seems protected properly.

IBE relies on trusted third party, often called Private Key Generator (PKG), which is responsible for generating secret keys corresponding to users' public keys. PKG has its own key pair called master public key and master private key (aka master key). The latter is involved in the process of creating user's private key from given identity.

The first practical implementation of IBE appeared in 2001 and was presented by Boneh and Franklin (BF) [5]. The BF scheme makes use of pairings (namely the Weil pairing) over elliptic curves and finite fields. Security of the scheme is based on elliptic curve analogue of the computational Diffie-Hellman assumption, so the underlying mathematical problem is hardness of finding discrete logarithms in finite cyclic groups.

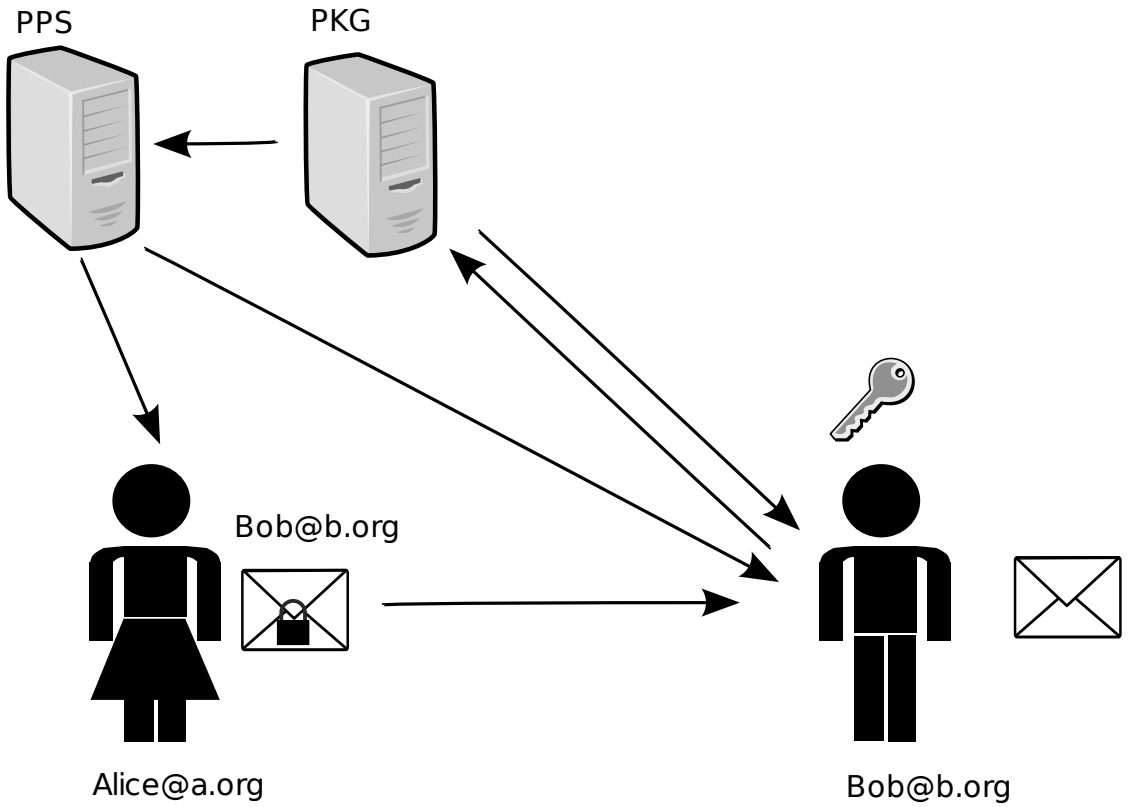


Figure 2.2: Scheme of Identity-Based Encryption

### Algorithms

Implementation of IBE cryptosystem relies on four randomized algorithms:

`Setup` Usually run by the PKG to create global public parameters and master key pair. The algorithm takes security parameter $k$ which determines strength of cryptosystem. Public parameters $\mathcal{P}$ include a description of finite message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Public parameters $\mathcal{P}$ are uploaded onto Public Parameters Server (PPS) and published. Master key $s$ is kept secret and is known only to PKG.

`Extract` The algorithm is run by the PKG when one requests for her private key. Input parameters are public parameters $\mathcal{P}$, master secret $s$ and identity string $\mathrm{ID} \in \{0,1\}^*$. The function returns user's private key $d$ corresponding to given identity.

`Encrypt` The procedure is invoked by the sender to encrypt message using specified identity. As input parameters takes public parameters $\mathcal{P}$, identity ID and message $M \in \mathcal{M}$. As output returns ciphertext $C \in \mathcal{C}$.

`Decrypt` Executed by the receiver to decrypt message using corresponding private key. Takes private key $d$, public parameters $\mathcal{P}$, identity ID, ciphertext $C$ as input and outputs message $M$.

### Advantages

IBE has several features not present in traditional PKI. First of all, users' public keys have very attractive form, which is much more user-friendly than numerical keys. Thanks to identities, there is no need for certificates and heavy-weight authorities. In principle, one could provide encryption services to every email user without any vetting. Moreover, it is possible to encipher message even if the recipient has not generated key pair yet. This approach reduces much of bandwidth and organizational overhead. In comparison to PKI, revocation of keys is much easier because when recipient's private key becomes obsolete, the sender does not need to get a new certificate from Bob. Even if the private key changes, corresponding identity stays the same, so the sender can be completely unaware of revocation of receiver's key.

### Drawbacks

On the other hand, when Bob wants to decrypt the message, he firstly contacts PKG, authenticates himself and then receives the corresponding private key. Trust placed in PKG is very high since it works as a key escrow and is capable of decrypting all the traffic. This property eliminates original IBE scheme from wide applications in big hostile networks like the Internet. However, it is still usable in closed commercial environments. Moreover, several modifications of IBE have appeared so far and fixed the above weakness. The most notable are certificate-based encryption (CBE) and certificateless cryptography (CL-PKE) schemes.

### 2.1.4 Certificate-Based Encryption

The primary goal of certificate-based encryption was to solve certificate revocation problem existing in traditional PKI. Certificate-based encryption was described by Gentry in his paper [10]. The scheme is based on IBE, but eliminates some of its deficiencies, such as presence of key escrow or compulsory confidential channel between user and Private Key Generator. In fact, CBE combines the best features of PKI and identity-based cryptography.

The role of certificates in CBE is twofold, on one hand they authenticate public keys, but on the other hand they act as decryption keys. The scheme provides both implicit and explicit certifications, the former comes from certificates and the latter from identities. Message decryption can be performed only if the recipient has got private key and valid certificate from her Certificate Authority, because messages are doubly encrypted.

Users of certificate-based encryption generate own private/public keys and request certificates from CA's. Certificate Authorities compute them from users' identities, as it happens in IBE, and deliver up-to-date certificates to clients. Since certificates can be publicly known, they do not have to be transfered over protected connections. Additionally, double encryption guarantees that neither client nor CA alone is capable of decrypting the ciphertext. Similarly to IBE, users' public keys can be arbitrary strings unique within CA.

#### Algorithms

The four randomized algorithms making up CBE are as follows:

`Setup` Run by CA to create global public parameters and master key pair. The algorithm takes security parameter $k$ which determines strength of cryptosystem. The public parameters $\mathcal{P}$ include a description of finite message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Master key $s$ is kept secret and is known only to CA.

`Certify` The algorithm is run by CA when client requests certificate for his identity. Input arguments are public parameters $\mathcal{P}$, master secret $s$, period $i$ in which the certificate is valid and identity string ID that contains user's public key and any necessary additional identifying information. The function returns certificate $Cert_B$ valid for given period.

`Encrypt` The procedure is invoked by the sender to encrypt message using specified identity. As input parameters takes public parameters $\mathcal{P}$, identity ID, validity period $i$, and message $M \in \mathcal{M}$. As output returns ciphertext $C \in \mathcal{C}$.

`Decrypt` Executed by the receiver to decrypt message using corresponding private key. Takes private key $d$, public parameters $\mathcal{P}$, identity ID, certificate $Cert_B$ and ciphertext $C$ as input and outputs message $M$.

#### Advantages

Certificate-based encryption is a serious alternative to traditional PKI model. It can supply lightweight infrastructure for public-key cryptography. CBE eliminates the need for third-party queries on certificate status and hence can greatly reduce demand on extra servers. CBE preserves remarkable features of both traditional public-key infrastructure and identity-based encryption, i.e. no key escrow property, reasonable trust to trusted third party, no confidential connection with CA and two modes of certification. Computational cost of key validation can be greatly reduced by application of *subset cores*, which are described in the original CBE paper. Thanks to this technique, a single CA server can handle all its clients.

#### Drawbacks

In comparison to IBE, user's public keys are longer, because they contain both identity and numerical part. Although, CBE removes third-party queries, it still requires online connection with clients, who regularly ask for new certificates. In practice, CBE servers would have to work non-stop to be able to provide certificates, which makes them vulnerable to DoS attacks. Certificateless cryptography is somewhat similar to CBE in terms of goals and implementation, but does not require continuous work.

### 2.1.5 Certificateless Cryptography

Certificateless cryptography (CL-PKC) was firstly presented by Al-Riyami and Paterson in their paper [2]. The work was highly influenced by BF scheme and as a result is an extension of the original IBE. CL-PKC eliminates the key escrow feature found in the Private Key Generator. Instead, creation of private key is split between a user and trusted third party called Key Generation Center (KGC). Consequently, user's public key is a pair composed of identity ID and public key $P_A$. The key is no longer easily memorable as in original IBE but the trust level placed on third party is much lower. It looks like functionality of CL-PKC is somewhere between traditional certified PKI and identity-based cryptography. Flexibility is one of the most significant attributes of certificateless cryptography; in fact, it can be transformed into traditional PKI or IBE. Similarly to IBE, mathematical foundations of CL-PKE come from elliptic curves and hardness of finding discrete logarithms in finite groups.

Main features of CL-PKC include the lack of key escrow property, no certificates to guarantee authenticity of public keys, the use of identities and existence of trusted third party which participates in key generation. To encrypt a message one needs public parameters, recipient's identity and public key. The use of identity in encryption prevents any other party from decrypting the content even if one tries to forge the second part of public key. Furthermore, the second part of public key (i.e. a point on elliptic curve) prevents KGC from deciphering the message.

Public key distribution works as in PKI, user publishes his public key in a public directory

or attaches to outgoing emails. As long as KGC does not try to forge the key, all the encrypted data are safe. This means that KGC must be as trusted as CA in traditional public key infrastructure. However, any forbidden activity of KGC can be detected by users. In practice no CA dares to publish fake keys since it puts the company out of profitable business, so it is reasonable to assume that KGC behaves honestly. Hence, it is possible to build secure transparent email encryption which allows non-technical users to communicate confidentially.

In contrast to PKI, certificateless scheme does not require expensive infrastructure composed of different kind of authorities. Similarly to IBE, only Key Generation Center and Public Parameters Server are needed. The optimal choice is to place them per name space such as DNS zone. These two servers shall cope with all the incoming traffic.

### Algorithms

Certificateless public key encryption with chosen ciphertext security (in [2] referred to as Full CL-PKE) is built upon seven randomized algorithms:

`Setup` Usually run by the KGC to create public parameters and master key pair. The algorithm's input is security parameter $k$ which determines strength of cryptosystem. The output consists of public parameters $\mathcal{P} = \langle \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, n, P, P_{pub}, H_1, H_2, H_3, H_4 \rangle$, description of finite message space $\mathcal{M} = \{0,1\}^n$ and ciphertext space $\mathcal{C} = \mathbb{G}_2 \times \{0,1\}^{2n}$ and master key $s$. Public parameters are uploaded onto PPS and published afterwards. Master key $s \in Z_q^*$ is kept secret and is known only to KGC.

`Partial-Private-Key-Extract` The algorithm is run by the KGC when one requests for her private key. Input arguments are public parameters $\mathcal{P}$, master secret $s$ and an identity string $\mathrm{ID}_A \in \{0,1\}^*$. The output is partial private key $D_A$ corresponding to given identity.

`Set-Secret-Value` Run by user to generate a secret value $x_A$. In general case public parameters $\mathcal{P}$ and appropriate identifier ID are inputs of the algorithm.

`Set-Private-Key` The algorithm computes user's private key $S_A \in \mathbb{G}_2^*$ from public parameters $\mathcal{P}$, partial private key $D_A$ and the secret value $x_A \in Z_q^*$.

`Set-Public-Key` The algorithm computes user's public key $P_A$ from public parameters $\mathcal{P}$ and secret value $x_A$.

`Encrypt` Invoked by the sender to encrypt message using specified identity. Takes public parameters $\mathcal{P}$, identity $\mathrm{ID}_A$, public key $P_A = \langle X_A, Y_A \rangle$, message $M \in \mathcal{M}$ and returns ciphertext $C \in \mathcal{C}$. Providing that the public key is corrupted, the algorithm returns $\perp$.

`Decrypt` Executed by the receiver to decrypt message using the corresponding private key. As input takes private key $S_A$, public parameters $\mathcal{P}$, identity $\mathrm{ID}_A$, ciphertext $C \in \mathcal{C}$ and outputs message $M$. Providing that the message is corrupted, the algorithm returns $\perp$.

**Advantages**

Certificatelesss cryptography can supply one of the most flexible infrastructures for public-key cryptography. It combines the best aspects of both traditional public-key infrastructure and identity-based encryption, such as lack of certificates, no key escrow property, reasonable trust to trusted third party and lightweight infrastructure. Applications of CL-PKE can be the same as for PKI and IBE, that is companies' networks, the Internet and consumer electronics devices. As with IBE, certificateless cryptography can be used as underlying mechanism for transparent email/sms encryption.

**Drawbacks**

Even thought public keys are not as simple as in identity-based encryption, it is still possible to fetch the numerical part of the key from given identity. To provide this kind of service, CL-PKE would have to incorporate a kind of public-key directory, present in web of trust. Furthermore, performed computations are rather complicated and expensive, so there is desire for faster algorithms before real-world systems can be implemented. Finally, original version of CL-PKE reduces certificates only for users, but preserves them on connections with PPS and KGC servers. To completely eliminate certificates, servers of the scheme would have to belong to hierarchical CL-PKE. Then, the public parameters and key of root server would be deployed together with software (as root CA's certificates in PKI).

## 2.2 Elliptic Curve Cryptography

Elliptic curves have numerous applications within public-key cryptography, and for this reason their own field is known as elliptic curve cryptography. They supply basic mechanisms for identity-based and certificateless public encryption schemes. Although elliptic curves are objects described by general equation:

$$Y^2 = X^3 + aX + b, \tag{2.1}$$

they are typically defined over finite fields when used within cryptography. Elliptic curves are attractive structures, because they allow to construct cryptosystems whose security relies upon hardness of finding discrete logarithms (ECDLP). Having points $P$ and $Q$, where $Q = kP$, there is no efficient algorithm for finding $k$. The most significant advantages of ECC include shorter keys and fewer computations than in other popular schemes (RSA, DH), while preserving the same level of security. An overall comparison may be found in Tab. 3.1, where $k$ is RSA public-key bit-length and $n_q$ is length of corresponding ECC key.

Elliptic curve over finite field is a set of points $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ satisfying (2.1) plus a point at infinity $\mathcal{O}$ being identity element. These points form a finite Abelian group $(E(\mathbb{F}_q), +)$ with negation of point $P = (x, y)$ being $-P = (x, -y)$ and operation of addition dependent on $q$, but holding the following properties:

- if $Q = \mathcal{O}$ then $P + Q = P$
- if $Q = -P$ then $P + Q = \mathcal{O}$
- if $Q \neq P$ then $P + Q = R \in E(\mathbb{F}_q)$.

The above facts allow us to construct operation of point multiplication, that is, for given $k$ it is straightforward to compute point $Q = kP$, since:

$$kP = \underbrace{P + P + \cdots + P}_{\text{k}}. \tag{2.2}$$

However, what is really utilized by elliptic curve cryptography is the cyclic subgroup of $E(\mathbb{F}_q)$, because for any point $P$ the set $\{\mathcal{O}, P, 2P, 3P, \ldots\}$ is a cyclic group. Of course the subgroup is much smaller (as comparison of $n_p$ and $n_q$ in Tab. 3.1 indicates). Elliptic curve discrete logarithm problem is built on top of the cyclic subgroup.

The area of elliptic curves is unbelievably wide and certainly out of scope of this dissertation. The above introduction shall be enough for understanding mathematical foundations of certificateless cryptography. Basic algorithms required by the project are described in book [4]. Furthermore, elliptic curves alone are not sufficient to construct identity-based schemes. These schemes incorporate pairings, which are mappings between groups.

## 2.3 Pairings

Pairing-based cryptography is a field in which cryptosystems are constructed upon pairings. Most of identity-based schemes, among which are IBE, CBE and CL-PKE, belong to this area. Generally speaking, pairing is a map between elements of two groups and a third group. In practice, it allows to solve certain problem in one group, even if the problem is said to be hard in another group. More detailed information about pairings and their applications in the context of elliptic curve cryptography can be found in [11].

Although pairing is a more general concept, its definition within cryptography is as follows. Let $\mathbb{G}_1, \mathbb{G}_T$ be cyclic groups of prime order $q$ and $\mathbb{G}_2$ group where each element has order dividing $q$. Moreover, groups $\mathbb{G}_1, \mathbb{G}_2$ are additive and $\mathbb{G}_T$ is multiplicative. Then, we say that map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an admissible pairing if satisfies the following properties:

1. Bilinear: $e(aP, bQ) = e(P, Q)^{ab}$ for all $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}$.
2. Non-degenerate: $e(P, Q) = 1_{\mathbb{G}_T}$ for all $Q \in \mathbb{G}_2$ if and only if $P = 1_{\mathbb{G}_1}$, and similarly $e(P, Q) = 1_{\mathbb{G}_T}$ for all $P \in \mathbb{G}_1$ if and only if $Q = 1_{\mathbb{G}_2}$
3. Computable: There is an efficient algorithm to compute $e(P, Q)$ for any $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$.

The above definition is sometimes called the asymmetric pairing. When groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are the same group, then we say that pairing is symmetric. Moreover, it may be hard

to find discrete logarithms in the three groups, but the Decision Diffie-Hellman problem (DDH) might be easy in $\mathbb{G}_1$ and $\mathbb{G}_2$. The DDH problem is important from theoretical point of view when it comes to security of pairing-based cryptosystems. The concrete implementations of pairings usually involve modified Weil or Tate pairing. More about the underlying problems and their precise mathematical discussions may be found in [5, 2, 10, 11].

## 2.4 Certificateless Cryptography Algorithms

After basic introduction to elliptic curve cryptography and pairings one is ready to understand precise definitions of CL-PKE algorithms:

`Setup` The algorithm comprises the following steps:

1. Generate tuple $\langle \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e \rangle$ where $\mathbb{G}_1$ and $\mathbb{G}_T$ are groups of some prime order $q$, order of each element belonging to $\mathbb{G}_2$ group divides $q$ and $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is pairing. Preferably $q$ is a Solinas prime, i.e. one of the form $q = 2^{e_2} \pm 2^{e_1} \pm 1$, where $e_1$ and $e_2$ are exponents.
2. Choose random generator $P$ of $\mathbb{G}_1$ group.
3. Choose random master key $s$ from $Z_q^*$ and compute public key $P_{pub} = sP$
4. Choose cryptographic hash functions $H_1\colon \{0,1\}^* \to \mathbb{G}_2^*$, $H_2\colon \mathbb{G}_T \to \{0,1\}^n$, $H_3\colon \{0,1\}^n \times \{0,1\}^n \to Z_q^*$ and $H_4\colon \{0,1\}^n \to \{0,1\}^n$, where $n$ is length of plaintexts.

`Partial-Private-Key-Extract` The following steps are performed to compute the partial private key:

1. Map identity to point on elliptic curve by computing $Q_A = H_1(\mathrm{ID}_A) \in \mathbb{G}_2^*$.
2. Compute the partial private key $D_A = sQ_A \in \mathbb{G}_2^*$.

`Set-Secret-Value` The algorithm chooses random $x_A \in Z_q^*$.

`Set-Private-Key` Outputs user's private key by computing $S_A = x_A D_A = x_A s Q_A$.

`Set-Public-Key` The algorithm returns $P_A = \langle X_A, Y_A \rangle = \langle x_A P, x_A P_{pub} \rangle = \langle x_A P, x_A s P \rangle$.

`Encrypt` Message encryption runs as follows:

1. If the conditions $X_A, Y_A \in \mathbb{G}_1^*$ and $e(X_A, P_{pub}) = e(Y_A, P)$ are satisfied, move to the next step. Otherwise, return $\perp$ and stop.
2. Compute $Q_A = H_1(\mathrm{ID}_A) \in \mathbb{G}_2^*$.
3. Choose random $\sigma \in \{0,1\}^n$.
4. Compute $r = H_3(\sigma, M)$.
5. Compute the ciphertext $C = \langle rP, \sigma \oplus H_2(e(Q_A, Y_A)^r), M \oplus H_4(\sigma) \rangle$. Note that the most complex operation is pairing $e(Q_A, Y_A)$, but its value is constant for given identity and public key, therefore it does not have be recomputed every time.

`Decrypt` Ciphertext $C = \langle U, V, W \rangle \in \mathcal{C}$ decryption runs as follows:

1. Compute $\sigma = V \oplus H_2(e(S_A, U))$.
2. Compute $M = W \oplus H_4(\sigma)$.
3. Compute $r = H_3(\sigma, M)$ and continue if $U = rP$. Otherwise, return $\perp$ and stop.
4. Return $M$.

# Chapter 3

# Project

The following chapter elaborates each step of software engineering with regard to certificateless cryptography project. The first stage contains assumptions, limitations and intended scope. Later, requirements are gathered so that functionality of the system is well-defined. When it is already known what to do, one thinks how to do, i.e. designs software by outlining its architecture, main modules and data flow between programs. The core stage of product development involves implementation and tests, which guarantee that system is built and works as expected. The final step covers software deployment and configuration.

## 3.1 Assumptions

Every new theoretical idea requires a real-world prototype in order to assess practical aspects of the solution. Therefore, implementation of certificateless cryptography presented in this work can be classified as a proof-of-concept project. The project provides email-like platform but for the sake of simplicity the sender and the receiver work in client-server architecture with no proper email implementation.

Moreover, provided cryptosystem is open to anyone, so each user has the right to choose identity himself. This is very similar to the way public email servers work. Consequently, the solution does not require users to authenticate to trusted third party before obtaining identities. These crucial assumptions make KGC less trusted and cryptosystem as a whole can be more credible.

Implementation of the cryptosystem shall be configurable to adapt to various working environments. Editable options must include elliptic curve parameters, servers' ports, addresses, logins and passwords and perhaps other technically important features.

Heavy mathematics involved in the scheme has its reflection in underlying libraries such as GMP (GNU Multiple Precision Arithmetic Library) [9], PBC (Pairing-Based Cryptography Library) [12] and OpenSSL (Open Secure Sockets Layer Library) [1]. These essential components are assumed to work properly and with no significant bugs in the applied

context. Unfortunately earlier releases of GMP and PBC did not fulfil the requirement and were carefully used in the project.

The last but perhaps the most important assumptions are security considerations. The KGC server behaves honestly, that is, does not mount attacks against users. In practice it means that KGC is forbidden to actively propagate false public keys. The nature of this assumption is similar to one placed on Certificate Authority in public key infrastructure. In addition, encrypted message sent by Alice to Bob is not altered in any way, so no integrity techniques are applied to protect the data against this type of attacks.

## 3.2 Requirements

Most requirements for the project are basic prerequisites which make up workable model without more advanced features or extensions. Of course, extra functionality is tempting but usually is beyond the scope of a core problem. The gathered requirements come from fundamental features of scalable and configurable IBE/CL-PKC cryptosystems described in [2, 6, 3].

### 3.2.1 Global

General requirements refer to technical features which are valid across different software parts (PPS, KGC, The Sender, The Receiver).

**Connection with PPS** Connection with PPS is always secured by Transport Layer Security (TLS). Moreover, the subject name in the server certificate matches the URL of PPS.

**Secure Sockets** Secure Sockets are provided by OpenSSL and Sockets libraries.

**Certificate Format** SSL certificates for servers are stored in PEM format.

**Third-Party Libraries** GMP and PBC libraries support computing mathematical formulas defined in certificateless cryptography.

**Code Portability** Software can be compiled at least on the following platforms: MS Windows, Linux, *BSD.

**Security Parameter** Cryptographic strength of the CL-PKE scheme depends on security parameter $k$ which corresponds to the modulus bit-size of comparable security in Diffie-Hellman or RSA public-key cryptosystems. Value of security parameter has influence on the choice of basic hash function and elliptic curve parameters: $n_p$ (bit-length of prime $p$ determining the order of the base finite field $\mathbb{F}_p$ over which the elliptic curve $E(\mathbb{F}_p)$ is defined) and $n_q$ (bit-length of prime $q$ determining the order of the cyclic subgroup in $E(\mathbb{F}_p)$. Table 3.1 shows valid possibilities. Although the second column is not utilized by CL-PKE, it presents how security level determined by $k$ relates to strength of symmetric key cryptography.

| $k$ | Bits of security | $n_p$ | $n_q$ | Hash function | Hash function OID |
|---|---|---|---|---|---|
| 1024 | 80 | 512 | 160 | SHA-1 | `1.3.14.3.2.26` |
| 2048 | 112 | 1024 | 224 | SHA-224 | `2.16.840.1.101.3.4.2.4` |
| 3072 | 128 | 1536 | 256 | SHA-256 | `2.16.840.1.101.3.4.2.1` |
| 7680 | 192 | 3840 | 384 | SHA-384 | `2.16.840.1.101.3.4.2.2` |
| 15360 | 256 | 7680 | 512 | SHA-512 | `2.16.840.1.101.3.4.2.3` |

Table 3.1: Mapping between security parameter $k$ and corresponding values

**Solinas Prime** Order of the cyclic subgroup in $E(\mathbb{F}_p)$ is a Solinas prime $q$, that is one of the form $q = 2^{e_2} \pm 2^{e_1} \pm 1$, where $e_1$ and $e_2$ are exponents.

**Groups** Certificateless cryptography depends on specific groups to perform computations. Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are the same, and are in fact a group of points belonging to elliptic curve $E(\mathbb{F}_p)$.

**Elliptic Curve** Elliptic curve over field $\mathbb{F}_p$ is given by equation $Y^2 = X^3 + X$.

**Pairing** For the used Type A elliptic curve modified Tate pairing shall be applied. The pairing is supplied by PBC library.

**Basic Hash Function** Basic hash function is used by CL-PKE specific $H_3, H_4$ functions. Choice of the function depends on security parameter $k$ and can be found in Tab. 3.1.

**Identities** Identities can be arbitrary ASCII strings composed of printable characters.

**Format of Public Paramaters** Data encoded in ASCII text format contain the following lines:

$q$
$exp_2$
$sign_1$
$exp_1$
$sign_0$
$H$
$[P_x, P_y]$
$[P_{pubx}, P_{puby}]$
$KGC_{Addr}$
$KGC_{Port}$

where:

| | |
|---|---|
| $q$ | Solinas Prime, that is prime of the form: $q = 2^{exp_2} + sign_1 * 2^{exp_1} + sign_0$ |
| $H$ | OID of basic hash function as specified in Tab. 3.1 |
| $P_x$, $P_y$ | Coordinates of point $P$ which is random generator of $\mathbb{G}_1$ group |
| $P_{pubx}$, $P_{puby}$ | Coordinates of point $P_{pub}$ which is system-wide public key |
| $KGC_{Addr}$ | IP/URL address of KGC server |
| $KGC_{Port}$ | Port on which KGC server is listening |

**File Storage of Public Parameters** Public parameters are stored in ASCII text file.

### 3.2.2 Public Parameters Server

PPS stores publicly sharable cryptographic material which is called public parameters. For every user of the system PPS is the first place to look for data describing the system.

#### Functional

**Stores Public Parameters** PPS is the only place with valid up-to-date public parameters. Public parameters include all of the information needed to encrypt/decrypt message except for identities and users' key pairs.

**Server is Open to Anyone** Like a public web server, provides content to any connected client. Location of PPS is publicly known and available (preferably PPS per name space such as a DNS zone).

**PPS is Updatable by Key Generation Center** Public parameters can be changed by KGC. Any update of this type makes users recompute their keys. KGC authenticates itself before providing new set of public parameters.

**PPS Supplies Verbose Output** Text output from PPS makes it possible to trace changes in public parameters.

**Configuration** It is possible to set at least the following options: public parameters, file with certificate in PEM format, corresponding password, login and password for public parameters update, address and port of KGC.

#### Non-functional

**Threads** PPS runs as a single-threaded server, because sending public parameters is probably more efficient than starting separate thread and handling request.

**Server Configuration** Basic options are stored in ASCII text file in the following format:

$N_{Threads}$
$PPS_{Port}$
$Certfile$
$Certpass$

where:

| | |
|---|---|
| $N_{Threads}$ | Number of threads in server's threadpool. In present version this value is always 0. |
| $PPS_{Port}$ | Port on which PPS server is waiting |
| $Certfile$ | Path to PEM certificate |
| $Certpass$ | Password to certificate |

**KGC Login and Password** Both strings can be composed of any set of printable ASCII characters. There is no limit on size of the strings. The pair is stored in ASCII text file in the following format:

*login*
*password*

### 3.2.3 Key Generation Center

KGC is at the heart of CL-PKC cryptosystem. Its responsibilities include issuing keys for users, generation and update of public parameters.

#### Functional

**Issues (Partial) Private Keys** Main task of KGC is derivation of (partial) private keys from users' identities. Here identity is any, unique within KGC, string of characters. The string is preferably, but not limited to, email address. The distinction between partial private key and private key is that the former is used in CL-PKC system, while the latter in identity-based encryption. Thus, KGC is a universal unit capable of providing services for various cryptosystems.

**Generates System Parameters** KGC generates system parameters, that is public parameters together with master secret. Strength of the parameters depends on security parameter $k$ which corresponds to RSA modulus bit-length of comparable level of security.

**Updates Public Parameters Server** PPS must always store the most up-to-date public parameters. These parameters are supplied by KGC upon successful authentication to PPS.

**Stores Master Secret** KGC generates master secret along with public parameters. Master secret is kept in confidence by Key Generation Center. It is used for computing (partial) private keys from identity material.

**Server is Open to Anyone** Like a public email server, allows any user to register new account and start using it immediately. KGC does not require user credentials authentication which means that clients can choose ID's of their preference (just like a mail login). It would be preferable to have KGC per name space such as a DNS zone.

**KGC Supplies Verbose Output** Text output from KGC makes it possible to trace changes in public parameters and users obtaining (partial) private keys.

**Configuration** It is possible to set at least the following options: public parameters, master key, file with certificate in PEM format, corresponding password, login and password for public parameters update, address and port of PPS.

#### Non-functional

**Encrypted and Authenticated Connection** Connection with KGC is always secured by Transport Layer Security (TLS). Moreover, the subject name in the server certificate matches the URL of the KGC.

**Threads** KGC runs nominally as a thread-pooled server with one governing thread and two worker-threads. However, it is possible to increase/reduce the number of worker-threads.

**Server configuration** Basic options are stored in ASCII text file in the following fashion:

$N_{threads}$
$KGC_{Port}$
$Certfile$
$Certpass$

where:

| | |
|---|---|
| $N_{threads}$ | Number of threads in server's threadpool |
| $KGC_{Port}$ | Port on which KGC server is waiting |
| $Certfile$ | Path to PEM certificate |
| $Certpass$ | Password to certificate |

**File Storage of Master Secret** Numerical value of Master secret is stored in ASCII text file in the following format:

$s$

**PPS Address and Port** The pair is stored in text file in the following format:

$PPS_{Addr}$
$PPS_{Port}$

where:

| | |
|---|---|
| $PPS_{Addr}$ | IP/URL of PPS server |
| $PPS_{Port}$ | Port on which PPS server is waiting |

### 3.2.4 The Sender

The sender (aka Alice) acts as a person who wishes to send an encrypted message to a friend (aka Bob). To send such a message Alice asks Bob for his public key, downloads public parameters from PPS, encrypts the message and sends it to Bob. To avoid unnecessary overhead related to POP3/SMTP protocols Alice connects to Bob directly over unsecured channel.

#### Functional

**Downloads Public Parameters** Alice downloads public parameters before encryption takes place. The parameters come from PPS which means that the sender must know location of the server.

**Retrieves Public Key from Bob** The sender asks Bob for his public key (but not identity). The key is *de facto* a point on elliptic curve defined in public parameters.

**Encrypts Message** Message encryption is the core operation of the project, and also the most complicated. It is a well-known fact that symmetric cryptography is much faster than public key cryptography in practical implementations. Therefore, it is reasonable to employ hybrid encryption which combines strengths of each encryption method. Symmetric encryption is used to cipher the message with a (pseudo)random secret key called Content Encryption Key (CEK). Later, the CEK is protected by public key cryptography. Effectively CL-PKC protects only CEK which is usually much shorter than the message, and so it takes less time to encrypt the data. Alice ciphers CEK with key computed from public parameters, Bob's identity and corresponding public key. Along with the encrypted message enciphered CEK, Public Parameters Server address and used identity are sent. These are needed in case Bob possesses several identities.

**Sends Message to Bob** As stated previously, message is composed of encrypted content, CEK, PPS address and recipient's identity. It is transmitted to Bob over unsecured channel. Although in a real-world setting it could be email, here it is direct connection in the client-server model.

**Gets Input Data from User** There are two compulsory input data: message content and recipient's identity. Both can be any printable strings of ASCII characters.

**Supplies Verbose Output** Text output from the sender makes it possible to trace public parameters, Bob's public key and encrypted message.

#### Non-functional

**Connection with Bob** Connection with Bob is not secured in any way. It works as a public channel vulnerable to eavesdropping but no content manipulation.

**Symmetric Encryption** AES encryption algorithm is the choice for securing message with 256-bit (pseudo)random key.

### 3.2.5 The Receiver

The receiver (aka Bob) is recipient of CL-PKC encrypted message sent by Alice. Firstly Bob asks KGC for his partial private key, then generates private/public key pair and shares the latter with Alice. Upon receiving the message he decrypts it and prints out the content. Bob is not an email client and for simplicity acts a server waiting on incoming messages.

#### Functional

**Downloads Public Parameters** Bob downloads public parameters to get the address of KGC, and later to decrypt the message. The receiver must know location of the PPS server.

**Obtains (Partial) Private Key from KGC** The receiver connects to KGC, presents its identity and asks KGC to compute the (partial) private key. After successful generation the key is transmitted to Bob.

**Generates Key-Pair** Bob creates public/private key pair from partial private key and pseudo(random) value $x_A$. Both keys are points on elliptic curve.

**Shares Public Key on Demand** Bob shares his public key with any client requesting it. As it comes from certificateless cryptography, no certificate or authentication is required to confirm association between the key and the owner.

**Decrypts Message** Decryption process is somewhat similar to message encryption but includes one additional step. When Bob receives message, he looks for identity and proper public parameters. Secondly, he decrypts Content Encryption Key using public parameters, his identity and public key. The next step requires validation of the decrypted key. After that, the receiver deciphers the content by means of symmetric decryption algorithm.

**Receives Message from Alice** The message is composed of encrypted content, CEK, PPS address and recipient's identity. Bob receives the message over unsecured channel which is similar, when it comes to the level of trust, to POP3 email protocol. In this setting, the receiver acts as a server waiting for incoming messages.

**Supplies Verbose Output** Text output from the receiver makes it possible to trace public parameters, Bob's key pair and decrypted message.

**Configuration** It is possible to set at least the following options: public parameters, address and port of PPS, public/private key pair, (partial) private key.

**Non-functional**

**Connection with KGC** Connection with KGC is always secured by Transport Layer Security (TLS). Moreover, the subject name in the server certificate matches the URL of the KGC.

**Connection with Alice** Connection with Alice is not secured in any way. It works as a public channel vulnerable to eavesdropping but no content manipulation.

**Symmetric Encryption** AES encryption algorithm is the choice for securing message with 256-bit (pseudo)random key.

**PPS Address and Port** The pair is stored in text file in the following format:

$PPS_{Addr}$
$PPS_{Port}$

where:

| | |
|---|---|
| $PPS_{Addr}$ | IP/URL of PPS server |
| $PPS_{Port}$ | Port on which PPS server is waiting |

**(Partial) Private Key Storage** The key is stored in text file in the following format:

$[D_{Ax}, D_{Ay}]$

where:

| | |
|---|---|
| $D_{Ax}$, $D_{Ay}$ | Coordinates of $D_A$ partial private key |

**Key-Pair Storage** The private/public key pair is stored in the text file in the following format:

$[X_{Ax}, X_{Ay}]$
$[Y_{Ax}, Y_{Ay}]$

where:

| | |
|---|---|
| $X_{Ax}$, $X_{Ay}$ | Coordinates of $X_A$ public key |
| $Y_{Ax}$, $Y_{Ay}$ | Coordinates of $Y_A$ private key |

**Server Configuration** Basic options are stored in ASCII text file in the following fashion:

$ID$
$ID_{Pass}$
$Bob_{Port}$

where:

| | |
|---|---|
| $ID$ | Bob's identity of user's choice |
| $ID_{Pass}$ | Password for given identity (in case KGC required it) |
| $Bob_{Port}$ | Port on which Bob server is waiting |

## 3.3 Design

Software requirements and specification find their reflection in design of a project. The following section outlines proposed solution, i.e. software architecture, modularization, data flow between applications and specification of employed protocols. Due to natural software complexity, textual descriptions are supported by various diagrams which show how particular units interact with each other.

### 3.3.1 Architecture

Nature of the project makes it network-oriented and as such is similar to email or www services. The ultimate goal is to provide end-to-end data protection over networks, therefore it is obvious to follow client-server model to implement required services. The whole system is composed of four software parts which run as stand-alone applications: Public Parameters Server, Key Generation Center, The Sender and The Receiver. All the applications, excluding The Sender, run as servers. Architecture of proposed solution is depicted in Fig. 3.1.

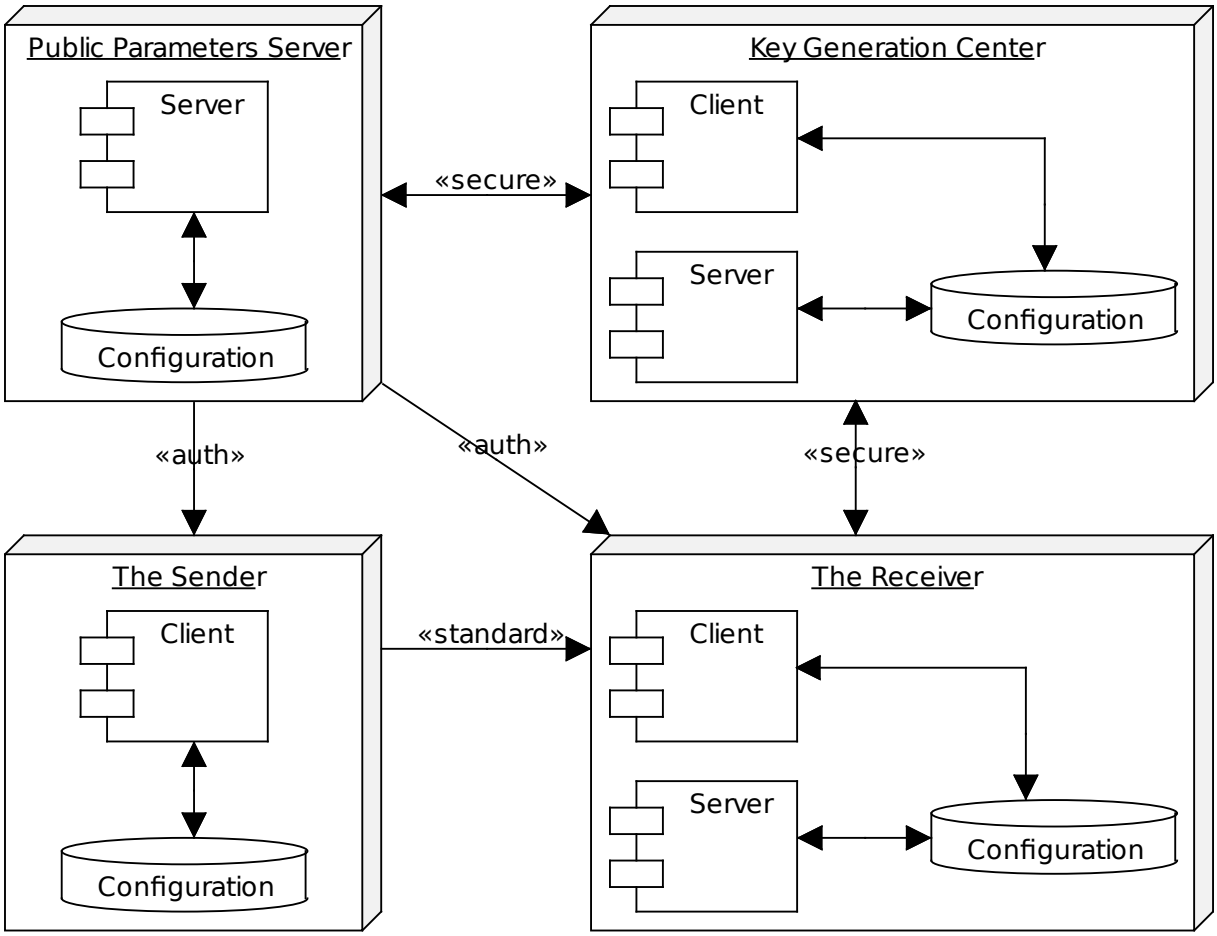


Figure 3.1: Deployment diagram of certificateless cryptography project

Basically, PPS stores public parameters which are used by every user of certificateless cryptography. It acts as a web server but uses different protocol to transfer data. Internally PPS server runs as a single-threaded application. As handling each request can be done very quickly, it is more efficient than starting separate thread to send public parameters. Occasionally, public parameters may be updated by Key Generation Center, which sends packet with new parameters to PPS. Parameters are updated after successful authorization, which is achieved by means of password. Besides logic, PPS contains module responsible for storing current configuration, i.e. public parameters and server's settings.

Key Generation Center application is more complicated than PPS, because it can work either as client or server. KGC switches to client mode during setup of cryptosystem. At this stage public parameters are uploaded onto PPS and master key is saved by KGC. Setup is done only when new parameters need to be generated (in practice very seldom). In nominal mode KGC is a server computing (partial) private keys from users' identities. Due to relatively expensive operation of point multiplication, which produces user's key, server sends these tasks to threadpool. The threadpool contains a constant number of threads, which are responsible for generating the key and sending it back to the user. This method of program execution allows to simultaneously accept requests and generate keys. Yet another module is configuration which manages server's settings.

Another server is implemented in The Receiver module (aka Bob). Its presence is justified

by simulation of electronic mail. In this nominal mode Bob receives messages directly from The Sender, decrypts them and prints out the output on screen. Additionally, The Receiver can run in client mode to generate own key pair. Again, client mode is used only when public parameters got updated or Bob needs new keys. As with previously described servers, The Receiver contains configuration unit responsible for managing application's settings.

The last software part is Alice application also called The Sender. It acts as a client who firstly receives public parameters from PPS, later obtains Bob's public key and finally sends encrypted data. Configuration module is relatively simple is not meant to supply any data persistence.

Most connections shall be secure to some extent. Properties of connections are marked in Fig. 3.1. Authenticated ones have `<<auth>>` writing. This kind of connections ensures that non-altered data come from certain source. When connection is described by `<<secure>>` label, then it has to preserve authenticity, integrity and confidentiality (because carries sensitive data, such as passwords or keys). If there is no need for additional security, then `<<standard>>` applies. Unsecured connection relies on TCP protocol to transfer data. In current implementation of certificateless cryptography certificate management is pushed upward, that is, certificates concern the whole domains instead of single users. If connection is not a standard one, then it is protected by Transport Layer Security (TLS) offered by OpenSSL library.

### 3.3.2 Data Flow

Network-orientation of the project can be confusing if described only in words. A well-known object method of software engineering is to present data flow in sequence diagrams. The diagrams not only show interactions between processes but also the order of operations. It is reasonable to depict only the core elements of the project which are not self-explanatory.

#### System Parameters Setup

Setup of system parameters is the first task to do to initialize certificateless cryptography platform. Administrator runs Public Parameters Server, which is waiting for packet with public parameters. Key Generation Center is responsible for generating public parameters and master key (altogether called system parameters). Strength of system parameters depends on security parameter $k$. When the parameters are ready, KGC sends proper packet to PPS. PPS sets new parameters after successful authentication of the packet and sends back confirmation (or error message). The whole process is depicted in Fig. 3.2.

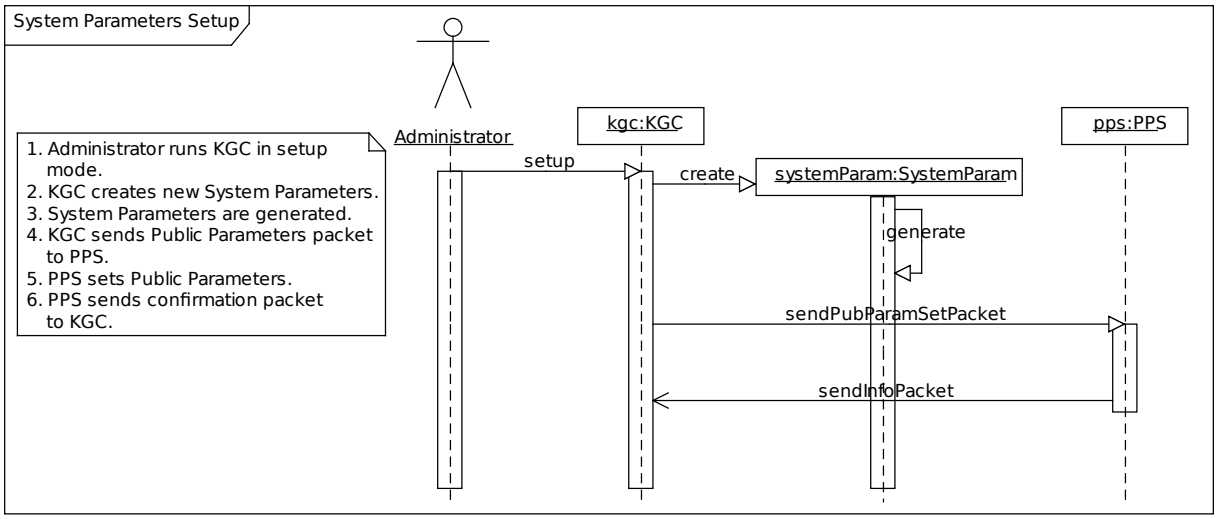


Figure 3.2: Sequence diagram of system parameters setup

### User's Key Pair Setup

To become a member of CL-PKE scheme each user must generate own key pair: public and private key. The Receiver (aka Bob) application is run by user in setup mode to compute the keys. Firstly, Bob contacts Public Parameters Server and obtains public parameters which include KGC address. In the next step he sends key-request with his identity to KGC. After that, KGC generates partial private key and sends back to Bob. The Receiver is now ready to choose random secret value and compute corresponding private and public keys. Figure 3.3 presents steps involved in the described process.

Similarly to original IBE scheme, there is a confidential channel with KGC to obtain partial private key. If the communication was not secure, the attacker could wiretap the partial private key and mount an attack against identity owner. The attack might be as follows: setup false key pair and publish public key in a directory. Any sender using the given key and identity would encrypt message only readable to the attacker, but not the owner.

### Confidential Communication

Confidential communication is the main service provided by the project. There are two users who want to send secure messages. Application named Alice sends message to Bob, who acts as recipient. Bob prints out decrypted message so that user can read the output. The whole activity starts with user typing the message and identity she intends to use. Next, Alice fetches Bob's public key and encrypts the message using Content Encryption Key and symmetric algorithm. When she is done, encrypted CEK is attached to encrypted data. She sends the packet to Bob, who reverses the activities. Firstly, he decrypts CEK and after that, decrypts the message. The interaction between user and applications is shown in Fig. 3.4.

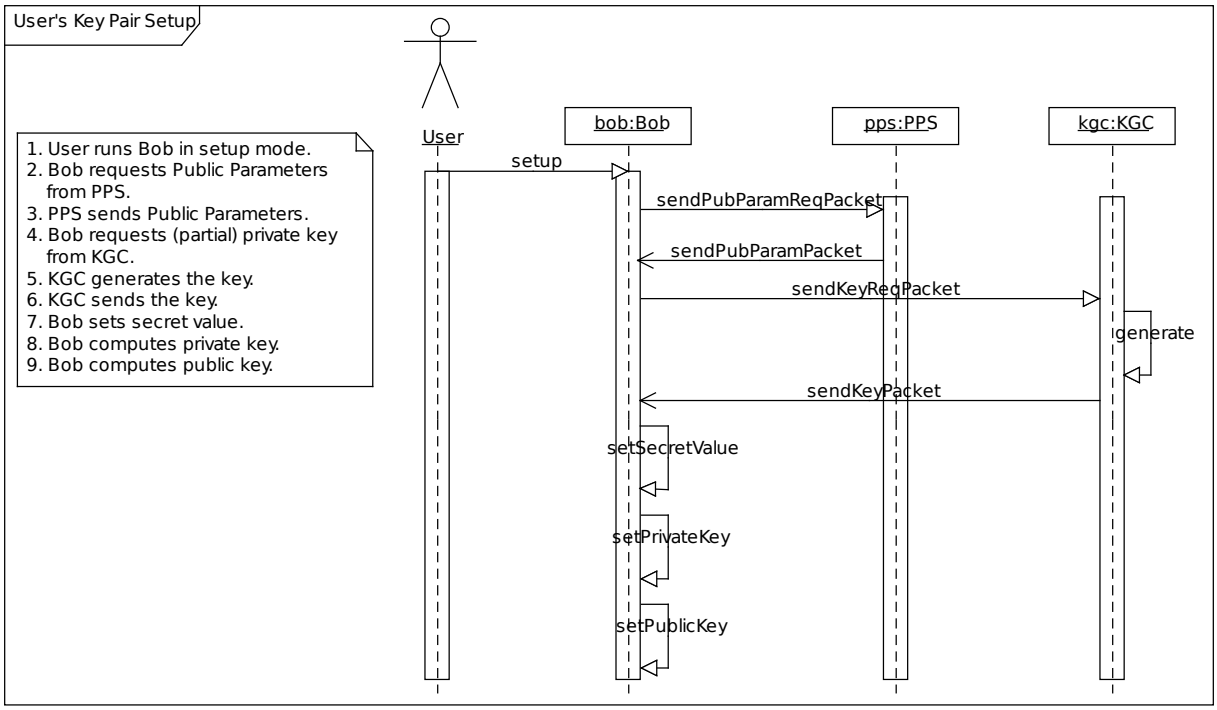


Figure 3.3: Sequence diagram of user's key pair setup

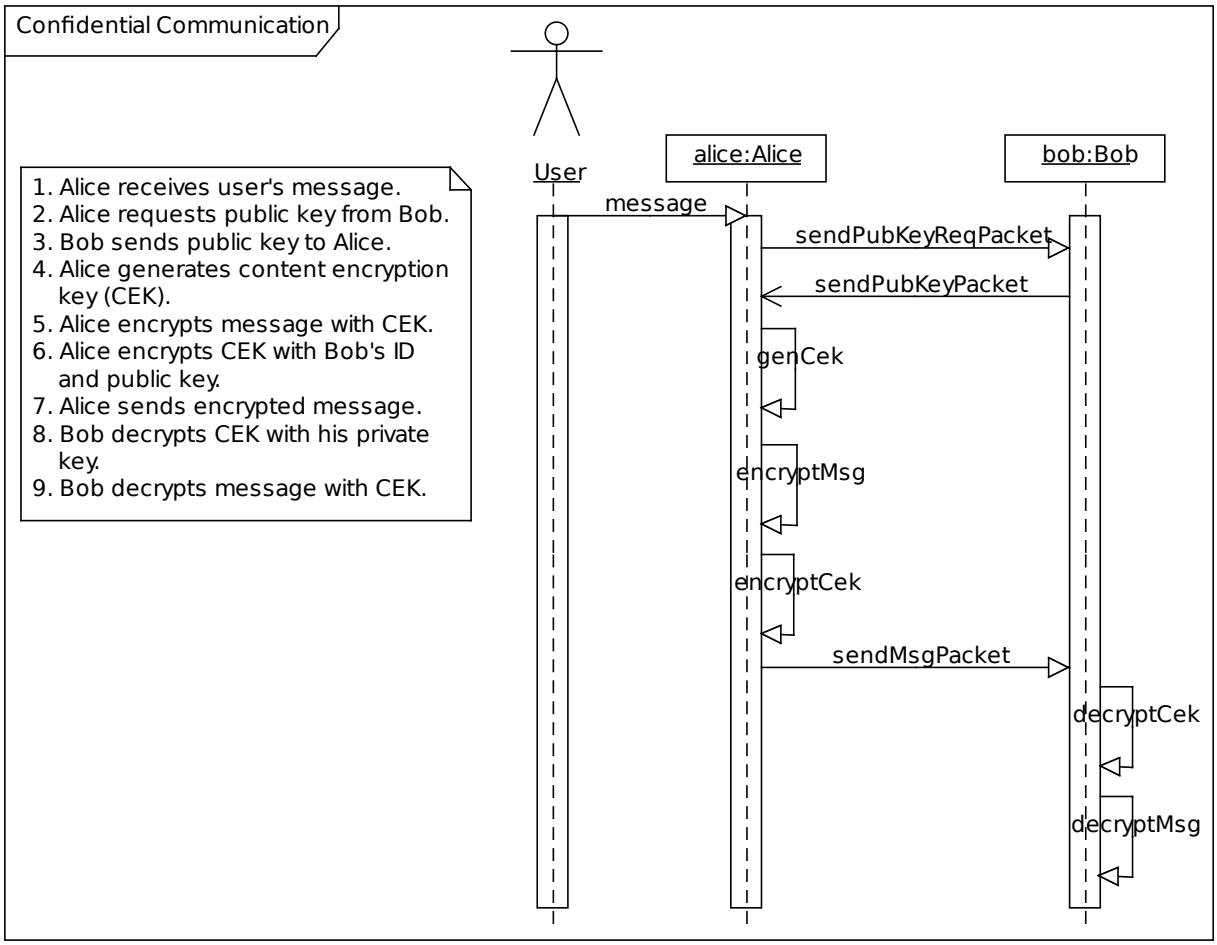


Figure 3.4: Sequence diagram of confidential communication

### 3.3.3 Protocols

TCP is the underlying protocol of the platform. However, TCP does not provide any level of data security in the sense of confidentiality or authenticity. Therefore, some links make use of TLS protocol to protect sensitive data. Transport layer is medium used for carrying applications' data but it is the application, that defines format of the data. Organized chunks of data are called packets. In general case the two types of packets are: binary or textual. The project makes use of textual packets, because they are more universal, work well with GMP and PBC libraries, can be easily modified and are more scalable. Certificateless cryptography utilizes the following application-level packets:

**Public Parameters**

| | | |
|---|---|---|
| Name | : | `PubParamPacket` |
| Direction | : | PPS → {Alice,Bob} |
| Content | : | Public Parameters |
| Description | : | Sent by PPS on client's request. |

**Public Parameters Request**

| | | |
|---|---|---|
| Name | : | `PubParamReqPacket` |
| Direction | : | {Alice,Bob} → PPS |
| Content | : | |
| Description | : | Client requests public parameters packet. |

**Set Public Parameters**

| | | |
|---|---|---|
| Name | : | `PubParamSetPacket` |
| Direction | : | KGC → PPS |
| Content | : | Public Parameters, KGC login and password |
| Description | : | Contains up-to-date public parameters. PPS requires valid login and password to set up new public parameters. |

**(Partial) Private Key**

| | | |
|---|---|---|
| Name | : | `KeyPacket` |
| Direction | : | KGC → Bob |
| Content | : | (Partial) Private Key |
| Description | : | KGC sends generated (partial) private key. |

**(Partial) Private Key Request**

| | | |
|---|---|---|
| Name | : | `KeyReqPacket` |
| Direction | : | Bob → KGC |
| Content | : | Identity and optional password |
| Description | : | User requests (partial) private key for his identity. |

**Public Key**

| | | |
|---|---|---|
| Name | : | `PubKeyPacket` |
| Direction | : | Bob → Alice |
| Content | : | Public Key |
| Description | : | Sent by Bob when Alice requests public key. |

**Public Key Request**

Name : `PubKeyReqPacket`
Direction : Alice → Bob
Content :
Description : Alice requests public key packet.

**Message**

Name : `MsgPacket`
Direction : Alice → Bob
Content : Encrypted message
Description : Alice sends CL-PKE encrypted message to Bob.

**Information**

Name : `InfoPacket`
Direction : PPS → {KGC,Alice,Bob}
Content : Result of operation execution
Description : Alice requests public key packet.

All the packets and the way they are interchanged between processes are depicted in data flow diagram in Fig. 3.5.

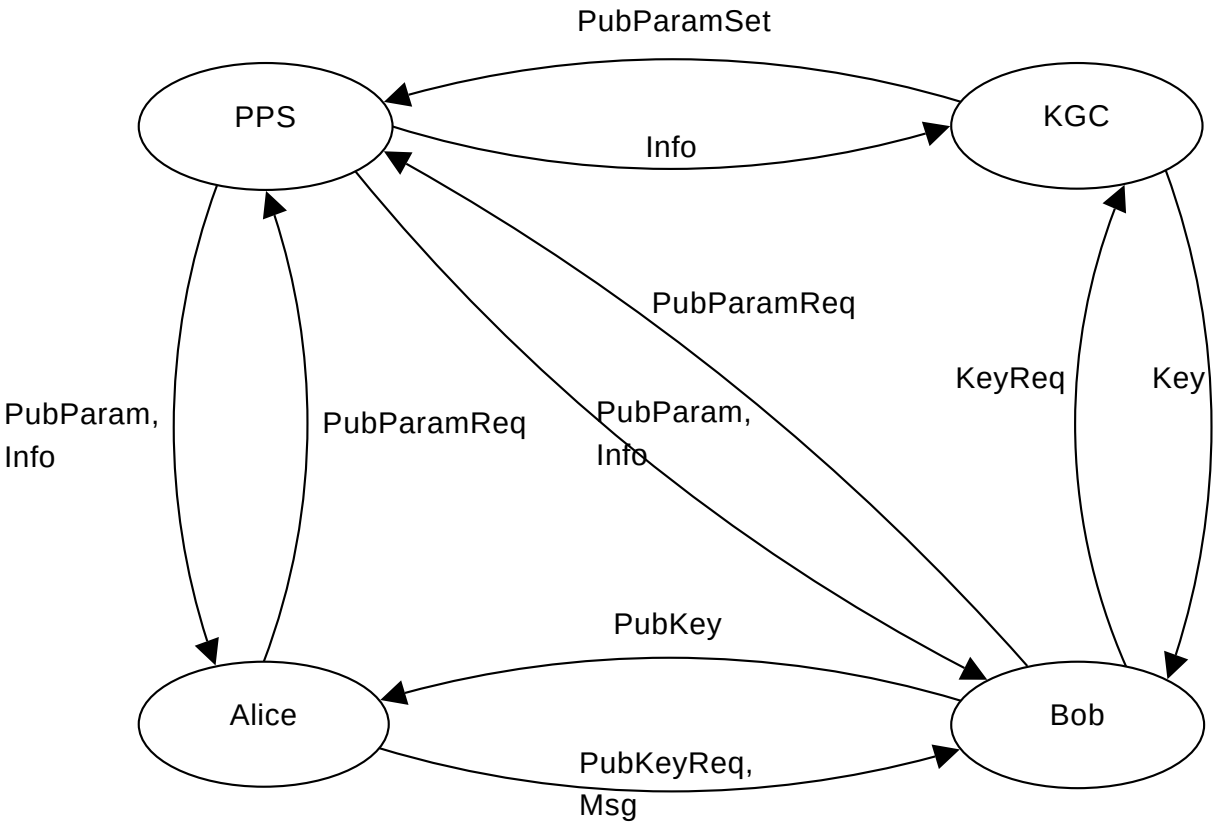


Figure 3.5: Data Flow Diagram of packets.

## 3.4 Implementation

The main aim of software engineering is to create application according to requirements. This stage focuses on specific tools such as programming languages, libraries and components which allow to quickly produce software of high quality.

Certificateless cryptography is implemented in conformity with object-oriented paradigm. This method focuses on data modelling and constructing data-centric software. Moreover, it hides technical details, so that large parts of code can be reused thanks to well-defined interfaces.

The project itself is written in C++ language, but prototypes of several functions were built in Haskell. The latter supported test-related activities, which are described in Sec. 3.5. C++ was chosen as the main language, because it provides object-oriented paradigm and generic programming which make it easy to develop software on relatively high level of abstraction. Besides, the language is known for its outstanding performance compared to other popular object-oriented languages, such as Java or Ruby. Moreover, C++ works well with underlying C libraries: GMP, PBC and OpenSSL. Another library, i.e. Sockets, supplied event-driven abstraction of the sockets mechanism.

On the other hand, programming in C++ itself is certainly slower than programming in Ruby or other modern language, because developer still needs to tackle with manual memory management or separating header files from implementation. Moreover, weak typing mechanism does not prevent programmer from making mistakes, which results in non-easily detectable bugs and long time spent on application debugging. Anyway, the cost of programming in C++ is perhaps still lower than the cost of creating/binding existing libraries to other languages when there is no reference code.

The whole project of certificateless cryptography is meant to use only free libraries and mechanisms, such as BSD sockets, GMP, PBC, OpenSSL and Sockets. This approach allows to create low-cost project relying on well-tested, universal and common components. Furthermore, thanks to rich documentation and great deal of resources/examples available on the Internet, many coding problems can be solved quickly and reliably.

### 3.4.1 Common

Even though architecture of certificateless cryptography defines four applications, there is a rich common base of classes, methods, functions which implement essential algorithms, hash functions, synchronization facilities and wrappers of low-level procedures. The Common module aggregates the following pieces of code:

- Certificateless cryptography algorithms and data structures.
- Basic and advanced hash functions.
- Standard and secure sockets both for clients and servers.
- Windows/POSIX mechanisms of synchronization and threading.

- Application-level packets (described in Sec. 3.3.3).
- Configuration modules of applications.
- Generic utilities for parsing application parameters, wrapping up libraries and data conversions.

Gathering these facilities in one place makes development of target applications easier and more reliable. It is a good practice of object-oriented paradigm to wrap up shared methods, so that that the code is more concise, parts of software fit well with each other and interfaces are clear. Besides this, final applications benefit from having similar structure and alike constructions. All of these support creating self-documenting code which is easy to understand and maintain.

### 3.4.2 Public Parameters Server

PPS contains the following components:

**Configuration** Manages server's configuration and is responsible for loading, storing and sharing setup data. One global instance of the class is valid across the PPS application. Besides basic parameters of SSL sockets it also keeps CL-PKE public parameters, login and password required for public parameters update.

**PPS Server Socket** The server socket module contains business logic of PPS by handling incoming packets among which the most important are Public Parameters Request and Set Public Parameters. In response to the first packet sends public parameters or information signaling that no parameters are set up. The second packet invokes validation of login and password upon which new parameters are saved. Whatever the result, PPS sends packet with feedback to KGC. The server socket uses Sockets library and overloads methods which handle certain events (reception of packets).

**PPS Main** The main module holds application's entry point and setup routines. It can be run either in setup or normal mode. The former takes input parameters and saves them in configuration files. The latter creates server socket and makes it wait for incoming events.

### 3.4.3 Key Generation Center

KGC contains the following components:

**Configuration** Manages server's configuration and is responsible for loading, storing and sharing setup data. One global instance of the class is valid across the KGC application. It keeps basic parameters of SSL sockets, address and port of PPS, login and password for public parameters update.

**KGC Server Socket** Its primary role is (partial) private key generation by means of `Partial-Private-Key-Extract` algorithm. The server socket handles only one type of packet, namely Key Request which later invokes the mentioned algorithm. Every request is sent to the Threadpool component, so that both keys and requests can be handled simultaneously. Similarly to PPS, the server socket uses event-driven Sockets library.

**KGC Client Socket** This module is used only in setup mode when KGC wishes to generate and publish public parameters. On successful connection to PPS, sends Set Public Parameters packet and in case of any error prints out message with details.

**Threadpool** Runs fixed number of threads which take key generation tasks and after serving them block on internal queue. A single task computes (partial) private key from user's identity and sends it back to client.

**KGC Main** As with PPS, the main module contains application's entry point and setup routines. It may run in setup or normal mode. The setup mode implements CL-PKE `Setup` algorithm, makes use of KGC Client Socket and saves configuration data in files. In the other mode, KGC runs server socket and handles incoming requests.

### 3.4.4 The Sender

The Sender contains the following components:

**Configuration** Manages client's configuration and is responsible for loading and sharing setup data. One global instance of the class is valid across The Sender application. The module keeps addresses and ports of PPS and Bob servers.

**Alice Client Socket** This core component performs `Encrypt` function specific to certificateless cryptography. Firstly sends Public Key Request to Bob in order to obtain recipient's public key. When the key is already received, client socket reads identity and message from the standard input, encrypts message and sends data to Bob afterwards. Having sent the packet, closes connection with The Receiver.

**Alice Main** The main component is the place where application's entry point is kept. The application firstly sets up configuration data, later fetches public parameters from PPS, and finally runs client socket module.

### 3.4.5 The Receiver

The Receiver contains the following components:

**Configuration** Manages configuration and is responsible for loading, storing and sharing setup data. One global instance of the class is valid across the Bob application. Besides basic parameters of SSL sockets it also keeps address and port of PPS, current public parameters and public/private key pair.

**Bob Server Socket** The server socket component handles packets coming from Alice, that is Public Key Request and Message. In fact, the latter implements CL-PKE `Decrypt` algorithm which is invoked whenever encrypted message arrives. After message decryption, its content is printed out so that user can read the text.

**Bob Client Socket** The module is responsible for communication with KGC and provides the following CL-PKE routines: `Set-Secret-Value`, `Set-Private-Key` and `Set-Public-Key`. On start, the component connects to KGC and sends Key Request to obtain (partial) private key. After reception of the Key packet, Bob generates key pair and closes connection. In case of error, shows message explaining the problem. The client socket makes use of the Sockets library to make up protected communication channel with KGC.

**Bob Main** Organizes global flow of control and contains application's entry point. Similarly to KGC, may be run in either setup or normal mode. The former reads input parameters, obtains public parameters, generates key pair and stores data in configuration files. The latter runs server socket and waits in infinite loop for incoming events.

## 3.5 Test

Tests are integral part of every serious software project; they usually come in two flavours: unit and functional. As mentioned earlier, some supporting code was written in Haskell. Haskell is a pure functional programming language offering very high level of abstraction and interactive environment (at least in GHC implementation). It works great as a prototyping and supporting tool, since the programmer does not need to concentrate on low-level details, but the algorithms themselves. Thanks to interactive environment it is possible to quickly generate valid public parameters or any other values useful for testing and debugging C++ code. Moreover, Haskell helped me with understanding implementation of ECC-related algorithms and detecting mistakes in `SignedWindowDecomposition` algorithm in [6].

The set of selected tests covers a great deal of code and verifies several aspects of final software, i.e. correctness of cryptographic algorithms, application of basic libraries, protocols, functionality of the system. Therefore, it is justified to assume that certificateless cryptography project works properly when all tests terminate with success.

### 3.5.1 Unit

Unit tests are meant to verify code of small units which are usually classes or sets of functions. Test vectors are one of the most reliable method used for verifying correctness of implemented cryptographic algorithms. They are set of input/output data characteristic to specific procedure. Usually they are provided by authors of the function or renowned organizations. Sometimes when there is no implementation of theoretical idea there might be no test vectors, which complicates process of testing. It is partly the case of certificateless cryptography. At the time of writing this dissertation there is still no widely-available

implementation of CL-PKE, and for this reason it is not possible to test the whole scheme. However, thanks to implementations of identity-based cryptography testing of common algorithms is feasible. Sample unit tests incorporating vectors are described below.

### Function `hashToRange`

Hash to range procedure hashes string $s$ onto element belonging to set $\{0, 1, \ldots, n-1\}$. The hash is computed by provided basic hash function $hashFcn$. The vector comes from [6] document.

**Input**

| | | |
|---|---|---|
| s | = | `54:68:69:73:20:41:53:43:49:49:20:73:74:72:69:6e:67:20:77:69:74:68:6f:75:74:20:6e:75:6c:6c:2d:74:65:72:6d:69:6e:61:74:6f:72` (“`This ASCII string without null-terminator`”) |
| n | = | `0xffffffffffffffffffffefffffffffffffffff` |
| hashFcn | = | `1.3.14.3.2.16` (SHA-1) |

**Output**

| | | |
|---|---|---|
| v | = | `0x79317c1610c1fc018e9c53d89d59c108cd518608` |

### AES-256 Algorithm

Text messages are encrypted with symmetric algorithm, namely 256-bit version of AES. Although, it is provided by well-tested OpenSSL library, the whole function is built upon several OpenSSL procedures. Therefore, it is desirable to verify whether they are used in CL-PKE project correctly. The test vector comes from [8] document.

**Input**

| | | |
|---|---|---|
| plaintext | = | `0x00112233445566778899aabbccddeeff` |
| key | = | `0x000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f` |

**Output**

| | | |
|---|---|---|
| ciphertext | = | `0x8ea2b7ca516745bfeafc49904b496089` |

### CL-PKE Implies ID-PKC

Certificateless cryptography is easily transformable into identity-based encryption, i.e. by setting secret value $x_A = 1$ and computing private/public keys accordingly. Following this idea, CL-PKE should accept any test vectors valid for identity-based cryptography.

#### CL-PKE Scheme

Another test checks if certificateless cryptography works, i.e. executes each of algorithms and verifies the final result. Firstly, generates system parameters for security parameter $k = 1024$. Later, creates recipient's key pair and encrypts fixed message $M$ under certain identity ID, its public key and global public parameters. The ciphertext $C$ is then decrypted using the same ID and corresponding private key. Finally the output $M'$ is compared with original message $M$. Although this kind of test is much weaker than test vectors, it is reasonable to build it when there are no known test vectors for given scheme.

### 3.5.2 Functional

Functional tests help to assess software from user's point of view. In practice user runs executable version of software and performs various activities which are directly visible and verifiable. This kind of tests allows to validate behaviour of many functions which are coupled with each other and provide expected functionality. As the tests are performed by humans, they are well-defined and strict so that user with no prior experience is capable of checking results.

#### System Parameters Setup

1. Run PPS by executing `pps` command.
2. Run KGC by executing `kgc --setup=1024` command, which creates system parameters with security parameter $k = 1024$.
3. Check whether public parameters shown by PPS are the same as these of KGC.
4. If parameters are equal, then the test is passed.

#### User's Key Pair Setup

1. Run PPS by executing `pps` command.
2. Assuming that system parameters are generated, run KGC by executing `kgc` command.
3. Run Bob by executing `bob --setup`.
4. Check whether (partial) private key shown by KGC is the same as this of Bob.
5. Check whether Bob application prints out public and private key (two points).
6. If (partial) private keys are equal and Bob showed key pair, then the test is passed.

**Confidential Communication**

1. Run PPS by executing `pps` command.
2. Assuming that system parameters are generated, run KGC by executing `kgc` command.
3. Assuming that key pair is generated, run Bob by executing `bob` command.
4. Run Alice by executing `alice` command.
5. Check whether public parameters shown by Alice are the same as these of PPS.
6. Check whether public key shown by Alice is the same as this of Bob.
7. Type ID: `bob@domain.org`.
8. Type message: `Hello Bob!` and press enter twice.
9. Check whether message shown by Bob is the same as this of Alice.
10. If public parameters, public key, message are equal in mentioned applications, then the test is passed.

## 3.6 Setup

Setting up CL-PKE cryptosystem is rather a simple process. Firstly, administrator should generate two certificates in PEM format (using OpenSSL package). One certificate is for Public Parameters Server and another one for Key Generation Center. Note that certificateless cryptography is about lack of certificates for users, not for core servers. Generated PEM files shall be called respectively `pps.pem` and `kgc.pem`. Administrator should put these files into directories with `pps` and `kgc` applications. Of course the described settings might be overridden by updating certain configuration files (described in Sec. 3.7).

Having prepared the certificates, administrator runs `pps` application with `--setup` option to generate basic configuration. Next, the `pps` command should be run with no additional parameters. The PPS server is waiting for initial public parameters. Now operator starts `kgc` application with option `--setup=`$k$, where $k$ is valid security parameter described in Tab. 3.1. KGC generates system parameters, sets up the PPS server and terminates. After that, `kgc` application shall be run again, but without parameters. Both servers are ready and are waiting for incoming requests.

The two running servers form infrastructure of certificateless cryptography. One of the users, namely Bob, generates his public/private key pair. It is achieved by starting `bob` program with `--setup` option. The application terminates after successful creation of the keys. Because the receiver acts as a server, user shall start it by typing `bob` command.

At this point the cryptosystem is set up and user can encrypt and send some message by means of `alice` program.

## 3.7 Configuration

Configuration of different software parts allows user to adapt the software according to own preferences. All the data are stored in ASCII text files. This way of keeping data fits better to implemented system than external database. Basically, there is no need to use additional software just to save up to several kilobytes of data. Moreover, text files are platform independent and make it possible to update parameters by hand. Each application's configuration files are kept in the same directory as application.

### 3.7.1 Public Parameters Server

`.ppsppconfig` Contains public parameters stored according to **Format of Public Parameters** requirement.

`.ppsservconfig` Contains options of PPS server stored according to **Server Configuration** requirement.

`.ppsauthconfig` Contains authentication data used for remote public parameters update. The configuration is stored according to **KGC Login and Password** requirement.

Command line allows to set the following options when executing `pps`:

| Option | Description |
|---|---|
| `--setup` | Indicates setup mode |
| `--port=`$PPS_{Port}$ | Override PPS port |
| `--login=`*Login* | Override login for public parameters update |
| `--pass=`*password* | Override password for public parameters update |
| `--certfile=`*Certfile* | Override path to PEM certificate file |
| `--certpass=`*Certpass* | Override password to certificate |

### 3.7.2 Key Generation Center

`.kgcppconfig` Contains public parameters stored according to **Format of Public Parameters** requirement.

`.kgcspconfig` Contains master secret stored according to **File Storage of Master Secret** requirement.

`.kgcservconfig` Contains options of KGC server stored according to **Server Configuration** requirement.

`.kgcclientconfig` Contains parameters of connection with PPS stored according to **PPS Address and Port** requirement.

Command line allows to set the following options when executing `kgc` in normal mode:

| Option | Description |
|---|---|
| `--nthreads=`$N_{Threads}$ | Override number of threads in pool |
| `--certfile=`$Certfile$ | Override path to PEM certificate file |
| `--certpass=`$Certpass$ | Override password to certificate |

Additionally, in setup mode the following options can be set:

| Option | Description |
|---|---|
| `--setup=`$k$ | Generate system parameters of given security level |
| `--port=`$KGC_{Port}$ | Override KGC port |
| `--ppsport=`$PPS_{Port}$ | Override PPS port |
| `--ppsaddr=`$PPS_{Addr}$ | Override PPS address |
| `--ppslogin=`$login$ | Override login for public parameters update |
| `--ppspass=`$password$ | Override password for public parameters update |

### 3.7.3 The Sender

The Sender (`alice`) does not use any configuration files. However, command line allows to set the following options when executing `alice`:

| Option | Description |
|---|---|
| `--ppsport=`$PPS_{Port}$ | Override PPS port |
| `--ppsaddr=`$PPS_{Addr}$ | Override PPS address |
| `--bobport=`$Bob_{Port}$ | Override Bob port |
| `--bobaddr=`$Bob_{Addr}$ | Override Bob address |

### 3.7.4 The Receiver

`.bobppconfig` Contains public parameters stored according to **Format of Public Parameters** requirement.

`.bobqidconfig` Contains partial private key stored according to **(Partial) Private Key Storage** requirement.

`.bobpkconfig` Contains Bob's public and private key stored according to **Key-Pair Storage** requirement.

`.bobconfig` Contains options of Bob application stored according to **Server Configuration** requirement.

`.bobclientconfig` Contains parameters of connection with PPS stored according to **PPS Address and Port** requirement.

Command line allows to set the following options when executing `bob` in setup mode:

| Option | Description |
|---|---|
| `--setup` | Indicates setup mode |
| `--ppsport=`$PPS_{Port}$ | Override PPS port |
| `--ppsaddr=`$PPS_{Addr}$ | Override PPS address |

# Summary

## 3.8 Conclusion

Certificateless cryptography is a promising solution improving several weaknesses of public key infrastructure and identity-based encryption. Although the scheme is rather complex, it allows to hide many details so that end-users are not concerned with manual handling data security. Some applications of CL-PKE scheme may include email, cellular telephony, world wide web and corporate networks.

In comparison to other schemes, such as web of trust, traditional public key cryptography, identity-based encryption and certificate-based encryption, certificateless cryptography is a flexible and secure cryptosystem. The only practical problem might be relatively high computational cost, but it is improving as new mathematical propositions appear.

The final solution fulfills requirements, is a quality product and was delivered on time. These factors count as success according to goals of software engineering. The code is rather easy to extend and read, so it is possible to add new features and apply improvements. On the other hand, if I could start the project again and had the same knowledge as now, I would try to write every piece of code in Haskell instead of C++. The biggest disadvantage of C++ is that the language is very error-prone, and thus programmer has to spend considerable amount of time on debugging. It is very probable that creating library bindings or applying existing interfaces from C to Haskell would take less time than looking for bugs in C++ code.

All in all, the project provided me with rare opportunity of combining the most interesting fields of computer science: information security, software engineering and functional programming, which altogether create a bridge between theory and practice. Nevertheless, the project is not at all closed and it can be extended further.

## 3.9 Future Work

One of fundamental assumptions was that the project is a proof-of-concept, and consequently its functionality is highly limited. On the other hand, internal complexity and need for external libraries made the project quite a big undertaking. Due to limited time for the work, some features were not included into initial specification, but could be added in subsequent releases of software.

Perhaps one could expect certificateless cryptography to completely eliminate the need for certificates. In present version certificates are still required for KGC and PPS servers to provide secure connections with clients. To address this issue, the project would have to transform into hierarchical certificateless cryptography, where servers belong to tree-like structure. Then, a server of one name space (e.g pw.edu.pl) becomes a client of another name space (i.e. edu.pl). However, the root server (that is ".") does not have any parent and therefore its public parameters shall be built in software as it happens with SSL certificates.

Moreover, it would be a good idea to introduce certificateless cryptography into commonly used real-world applications, such as Mozilla Thunderbird email client. A simple plug-in might provide transparent encryption for millions of users all around the world. Besides, it could encourage other programmers to provide faster and more secure solutions related with electronic mail.

Furthermore, current version of the project does not provide any public directory server which would store users' public keys and send keys corresponding to given identities. This kind of application would be a fundamental factor of transparent data encryption.

Finally, created software still has some bottlenecks, especially when it comes to generation of system parameters for big value of security parameter $k$. Furthermore, many cryptographic operations shall be optimized so that average encryption/decryption time is similar to that of RSA algorithm. Besides, neither The Sender nor The Receiver caches data such as pairing values, and it is another point where optimizations are possible.

# Bibliography


[1] OpenSSL: The Open Source toolkit for SSL/TLS. URL: www.openssl.org.

[2] Sattam S. Al-riyami, Kenneth G. Paterson, and Royal Holloway. Certificateless public key cryptography. pages 452–473. Springer-Verlag, 2003.

[3] G. Appenzeller, L. Martin, and M. Schertler. Identity-based Encryption Architecture. Internet-Draft (Expired: May 2008), nov 2007. Available at `http://tools.ietf.org/id/draft-ietf-smime-ibearch-06.txt`.

[4] Ian F. Blake, G. Seroussi, and N. P. Smart. *Elliptic curves in cryptography*. Cambridge University Press, New York, NY, USA, 1999.

[5] Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 213–229, London, UK, 2001. Springer-Verlag.

[6] X. Boyen and L. Martin. Identity-Based Cryptography Standard (IBCS) #1: Supersingular Curve Implementations of the BF and BB1 Cryptosystems. RFC 5091 (Informational), dec 2007. Available at `http://www.ietf.org/rfc/rfc5091.txt`.

[7] Carl Ellison and B. Schneier. Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure, 2000. Available at `http://www.schneier.com/paper-pki.html`.

[8] FIPS. *Advanced Encryption Standard (AES)*. NIST, nov 2001. Available at `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[9] Free Software Foundation, Inc. *GMP: The GNU Multiple Precision Arithmetic Library*, 2006. Available at `http://gmplib.org/`.

[10] Craig Gentry. Certificate-based encryption and the certificate revocation problem. In *EUROCRYPT*, pages 272–293, 2003.

[11] Ben Lynn. *On the Implementation of Pairing-Based Cryptography*. PhD thesis, Stanford University, 2007.

[12] Ben Lynn. *PBC: Pairing-Based Cryptography Library*, 2008. Available at `http://crypto.stanford.edu/pbc/`.

[13] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[14] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999.

# Appendices

# Nomenclature

| | |
|---|---|
| BF | Identity-based encryption Boneh and Franklin scheme |
| CA | Certificate Authority |
| CBE | Certificate-Based Encryption |
| CEK | Content Encryption Key |
| CL-PKC | Certificateless Public-Key Cryptography |
| CL-PKE | Certificateless Public-Key Encryption |
| CRL | Certificate Revocation List |
| DDH | Decision Diffie-Hellman problem |
| DH | Diffie-Hellman |
| DoS | Denial of Service |
| ECC | Elliptic Curve Cryptography |
| ECDLP | Elliptic Curve Discrete Logarithm Problem |
| GMP | GNU Multiple Precision Arithmetic Library |
| IBE | Identity-Based Encryption |
| ID | Identity |
| ID-PKC | Identity-Based Public-Key Cryptography |
| KGC | Key Generation Center |
| MITM | Man-In-The-Middle |
| PBC | Pairing-Based Cryptography |
| PEM | Privacy-enhanced Electronic Mail |
| PGP | Pretty Good Privacy |
| PKC | Public-Key Cryptography |
| PKG | Private Key Generator |

| | |
|---|---|
| PKI | Public-Key Infrastructure |
| PPS | Public Parameters Server |
| RA | Registration Authority |
| SSL | Secure Sockets Layer |
| TLS | Transport Layer Security |