Multimed Tools Appl
DOI 10.1007/s11042-014-2119-7

An implementation of enhanced public key infrastructure

Syh-Yuan Tan · Wei-Chuen Yau · Boon-Hock Lim

Received: 12 March 2014 / Revised: 18 May 2014 / Accepted: 20 May 2014

Abstract In this paper, we present the implementation of an enhanced public key infrastructure (PKI) which supports not only conventional public key cryptography (PKC) but also identity-based cryptography (IBC). In addition, we discuss the possible way of placing together IBC and PKI as well as solving the problems of user secret key revocation of PKI and IBC. As a proof of concept, an IBC framework is incorporated into Enterprise Java Bean Certified Authority (EJBCA) and the performance is reported.

Keywords PKI · Identity-based · Cryptography · EJBCA

1 Introduction

It is very crucial to ensure the confidentiality and integrity of multimedia contents while being stored on remote servers or during transmission between client systems and multimedia content providers [7, 26]. Encryption and digital signatures are two commonly used cryptographic primitives that could provide the security services, such as confidentiality, integrity, and authenticity of messages. Public key infrastructure (PKI) is an infrastructure that supports the operation of these cryptographic applications [2, 23]. It is used to generate, store and distribute digital certificates which can substantiate that a particular public key belongs to a certain entity. As shown in Fig. 1, Bob applies a digital certificate for his public key through a registration authority (RA). The RA verifies his identification information (e.g., driver's license) and sends his certification request to a certificate authority (CA). The

S.-Y. Tan (✉) · B.-H. Lim
Faculty of Information Science and Technology, Multimedia University, Melaka, Malaysia
e-mail: sytan@mmu.edu.my

B.-H. Lim
e-mail: boonhock0204@gmail.com

W.-C. Yau
Faculty of Engineering, Multimedia University, Cyberjaya, Malaysia
e-mail: wcyau@mmu.edu.my

Published online: 13 June 2014

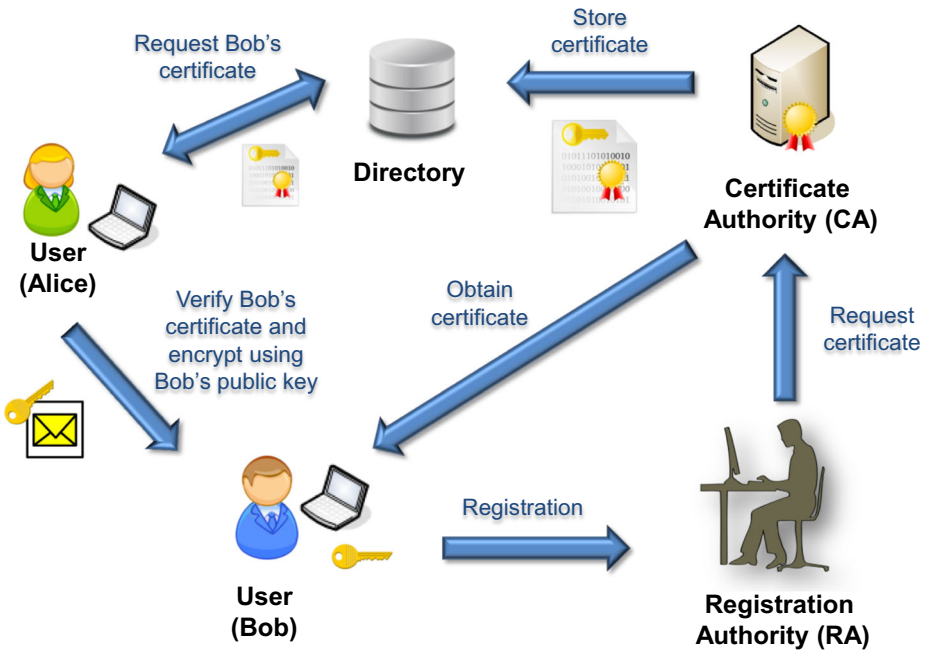

Fig. 1 Public key infrastructure

CA is a trusted third party who creates and signs a digital certificate that binds Bob's identity and his public key. The certificate is then sent to Bob as well as uploaded to a directory or repository. The CA also maintains a certification revocation list (CRL) containing information of invalid or revoked certificates. Alice downloads the certificate from the directory and verifies the validity of the certificate before extracting Bob's public key. She can then use the public key to encrypt messages for Bob. The PKI enables the use of public key in a secure manner and prevents man-in-the-middle attack. However, one of the main limitations of PKI is that the certificate management becomes inefficient when the directory contains huge amount of public keys.

In 1984, Shamir proposed the concept of identity-based cryptography (IBC) [22] that solves the certificate management issue. Any string that represents a user's identity (e.g. ID number, email address, phone number, etc.) can be used as a public key in IBC. This rules out the requirement of certificate authority as there IBC does not issue and store certificates. On the other hand, IBC requires a trusted third party, private key generator (PKG), which uses a master secret key to generate private keys for each user in the system. Therefore, an identity-based cryptosystem has the implicit key escrow issue. Figure 2 illustrates the concept of IBC.

Many international enterprises such as Microsoft, TrendMicro, IBM, Canon etc. are not stopped by the key escrow problem and adopted this technology instead. The main reason behind this is the cost saving attraction of IBC which is approximately one third of PKI [24]. Nowadays, the research in new cryptographic primitives in IBC is very active but many of the proposed primitives cannot be implemented due to the reason that IBC is not backward compatible with PKI. This is caused by the fundamental difference between CA and PKG where the latter cannot generate certificate, it generates user private key.

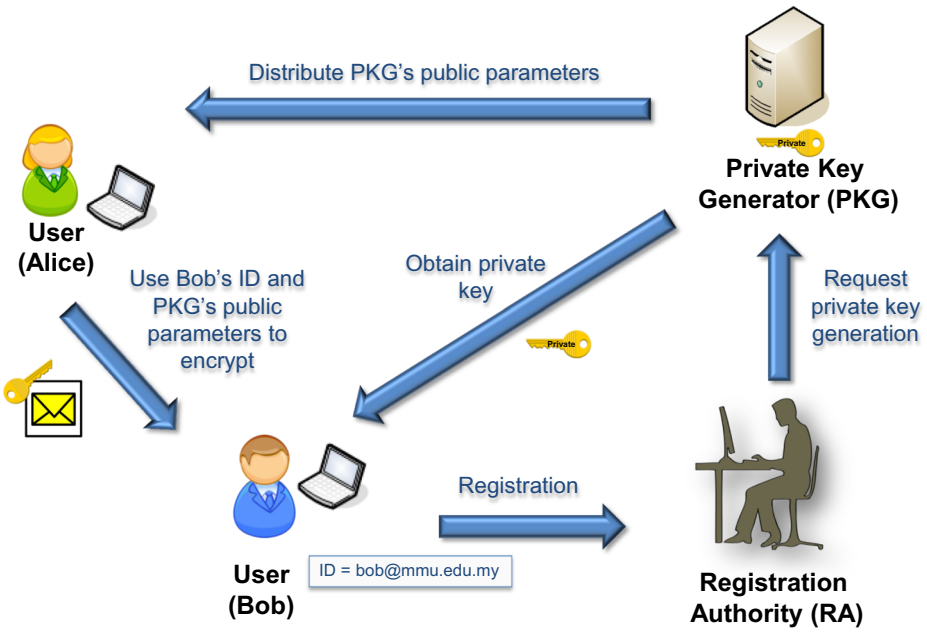

Fig. 2 Identity-based cryptography

In general, IBC is suitable for closed organizations (i.e., within a company) due to the key escrow property where the management level which controls the PKG can decrypt and monitor the information flow through the organization's gateway. Another important advantage of using IBC in an enterprise which consists of a huge number of employees is to reduce the cost and time of managing digital certificates. A company can setup their own PKG which is managed by their own system administrator. The company can treat the employees' email address as their public keys. The employees' private keys are then derived from the email address and master secret key of the PKG. It would then be convenient for the user to verify digital signatures of emails or electronic documents by only using the sender's email address (i.e., the public key in IBC).

Apart from the traditional applications where enterprises adopting PKI and purchasing digital certificates from CA or adopting IBC solution alone, there are various application scenarios where we could combine the use of PKI and IBC to satisfy different application requirements for enterprises. For example, two companies can adopt PKI and use a CA to verify the master public keys of their PKGs respectively. While, the users private keys that used within the company are derived by the respective company's PKG. In this case, each company needs to maintain only the digital certificate for the corresponding company rather than the digital certificates of every employee. For such implementation, it would require cryptographic packages that could support services provided by both conventional PKI as well as IBC. However, the current cryptographic libraries [6, 15, 16, 18] lack the support of IBC for practical applications. It is therefore crucial to come up with an implementation that could integrate cryptographic services offered by both PKI and IBC.

1.1 Related works

The earliest discussion on deploying IBC in the real world is dated back to 2002 by Chen et al. [3]. They proposed to replace PKI completely by using IBC and mechanism of managing user private keys as well as authenticating user public keys were demonstrated. However, their work does not catch the eye balls of the industries as PKI has been running since 80s. Existing technologies such as operating systems, software, embedded systems, mobile applications, to name a few, already have PKI embedded in them and none of the industries are willing to change. However, if the world takes a revolutionary step by replacing PKI with IBC, a lot of security problems can be solved efficiently. In view of this, Dalton [5] discussed the conveniences that IBC can offer to the National Health Service compared to PKI, and listed down the challenges to be solved if IBC is to be adopted.

In 2005, Price and Mitchell [19] proposed to integrate IBC with PKI, instead of replacing as suggested by Chen et al.. They found out that the root cause for backward compatibility issues of IBC is the certification policy standard in X.509. [4, 20] later expanded the idea in Price and Mitchell's idea but no concrete example was given. Furthermore, the description of cross certification of CA and PKG (called Trusted Authority (TA) in their work) is flawed. Although [4, 20] assumed the PKG of IBC already hold a certificate from a CA, the initialization process of cross certification for PKG and CA is ambiguous. The authors proposed to start the initialization process by encrypting to each other, a symmetric key plus a signature which authenticates the recipient, using the respective public keys. However, it is not possible to perform encryption and signing using the same set of public and private key pair. The only way of doing this is using signcryption scheme or using two sets of public and private key pair. If the latter is the case, the proposed cross certification model is having the same idea as in [13]. In 2010, [13] proposed another direction, which is having IBC and PKI working side by side instead of integrating them, namely, Unified PKI. Unified PKI can offer the benefit of both IBC and PKI but it is not efficient. Besides, its mechanism is very similar to that of Certificateless Cryptography [21] and defeated the purpose of IBC: user's public identity is the public key. Thus, in order to efficiently maintain the advantage of IBC, solving the cross-certification as suggested in [19] is the most feasible way to date.

1.2 Contribution

While the correctness of development direction for using IBC and PKI together is explored, there is no PKI libraries which supports both PKI and IBC operations concurrently. In this work, we implement IBC into the Enterprise Java Bean Certificate Authority (EJBCA) [6] in order to provide flexible key management options. Besides acting as a research tool, the enhanced PKI is useful in the cases where:

1. An organization wants to setup a new security system with IBC.
2. An organization wants to setup IBC on top of existing PKI system.
3. IBC is used for internal communication; CA is used for external communication.

The remaining of the paper is organized as follow. In Section 2 we compare some well-known PKI libraries and justify the use of EJBCA followed by the integration of IBC and EJBCA in Section 3. In Section 4 we show the performance of the enhanced PKI and finally, Section 5 concludes this paper.

2 PKI libraries

We briefly introduce the strengths of some popular PKI libraries in terms of their supported cryptography algorithms, flexibility and potential of future modification.

2.1 Enterprise java bean certified authority

Enterprise Java Bean Certificate Authority (EJBCA) [6] is an enterprise class PKI Certificate Authority software, built using Java Enterprise Edition and this make EJBCA become a cross platform CA which can be set up on any operating system that supports Java. Besides, EJBCA can support unlimited number of root CAs and sub CAs as well as able to perform cross certification for CAs and bridge CAs. In addition, EJBCA supports 8192-bit RSA keys, 1024-bit DSA key and ECDSA key with named curves from NIST, SEC and X9.62 [6]. Besides, it supports multiple hash algorithms for signatures such as MD5, SHA-1, and SHA-2. Therefore, administrator may setup different CA with different algorithms.

In terms of performance and capacity, EJBCA can issue hundreds of certificate per second and store millions of certificate by using RDBMS for storage. The supported databases are Hypersoniq, MySQL, PostgreSQL, Oracle, DB2, MSSQL, Derby and so on. Furthermore, EJBCA provides Certificate Management Protocol (CMP), Simple Certificate Enrollment Protocol (SCEP) and Online Certificate Status Protocol (OCSP) to handle its X.509 certificates.

2.2 Network security services

Network Security Services (NSS) [15] is a set of libraries designed to support cross-platform development of security-enable client and server application. NSS is triple-licensed under GNU General Public License, GNU Lesser General Public License and Mozilla Public License. NSS is written in several languages such as Python, Java and C.

NSS supports common encryption and hashing algorithms such AES, DES, IDEA, Triple DES, MD5, SHA-1, SHA-2, Diffie-Hellman Key exchange, Elliptic curve and so on. NSS supports several databases such as PostgreSQL, MySQL, OpenLDAP and Virtuoso.

NSS also supports LDAP and OCSP for certificate status checking but the CRL have to be updated manually [15].

2.3 X Certificate and key management

X Certificate and key management (XCA) [25] is an application to create and manage X.509 certificate, certificate request, RSA, DSA and EC private keys, Smartcards and CRLs. XCA is under BSD License and written in C language which makes XCA become a cross-platform application.

XCA provides a very complete function in managing private keys and certificates. XCA can export certificate into X.509v3, PKCS #7/#11/#12 format. Besides, XCA also supports common cryptography algorithms such as RSA, DSA and ECDSA.

XCA can manage CRL as other libraries do but it needs manual update [25].

2.4 OpenCA

OpenCA [16] is under BSD License and written in Apache CGI plus Perl language. OpenCA support cryptography algorithms such as RSA, DSA, ECDSA, Diffie-Hellman,

EC Diffie-Hellman, DES, Triple DES, AES, RC2, RC4, MD2, MD5 , SHA-1, SHA-256, SHA-384, SHA-512 and HMAC. Certificate can export to format such as SSLv2/v3, TLSv1, and PKCS #5/#8/#12, S/MIME, X.509v3 and also support LDAP and OCSP for certificate status checking.

OpenCA only supports MySQL, Oracle and PostgreSQL databases. However, similar to NSS, manual administration is required to update CRL [16].

2.5 Candidate for IBC

The characteristics of the libraries are summarized in Table 1. We can see that the most flexible library is EJBCA as it is cross-platform and supports multiple databases. The main cryptography provider used in EJBCA is BouncyCastle [14] but it is always possible to add any Java cryptography provider such as Oracle, OpenJDK [17] or FlexiProvider [9]. Therefore, the cryptography algorithms suppported are not lesser than that of the other libraries. Most importantly, EJBCA is the only library which supports auto update for CRL and is also the only library which supports administration through command line interface (CLI).

Table 1 Libraries Summary

Element	Libraries			
	EJBCA	NSS	XCA	OpenCA
License	GNU Lesser General Public	GNU General Public, GNU Lesser General Public Mozilla Public	BSD	BSD
Certificates	PKCS #11/12, X.509v3, TLS, S/MIME, SSLv2	SSLv2/v3, TLS, PKCS #5/#8/#12, S/MIME, X.509v3	PKCS#7/#11/#12, S/MIME, X.509v3	SSLv2/v3, TLS, PKCS#5/#8/#12, S/MIME, X.509v3
Smart card Support	Yes	Yes	No	No
Language	JavaEE	C	C,C++	Perl
Database	Hypersoniq, MySQL, PostgreSQL, Oracle, DB2, MSSQL, Derby etc.	PostgreSQL, MySQL, OpenLDAP, Virtuoso	XCA database	PostgreSQL, MySQL, Oracle
Algorithms	AES, DES, IDEA, 3DES, RSA, DSA, ECDSA, SHA-1, SHA-2, Key Exchange	AES, DES, IDEA, 3DES, RSA, DSA, ECDSA, SHA-1, SHA-2 Key Exchange	AES, DES, IDEA, 3DES, RSA, DSA, ECDSA, SHA-1, SHA-2, Key Exchange	AES, DES, IDEA, 3DES, RSA, DSA, ECDSA, SHA-1, SHA-2, Key Exchange
CRL update	Automatic	Manual	Not supported	Manual
LDAP support	Yes	Yes	No	Yes
OCSP support	Yes	Yes	Yes	Yes
CLI support	Yes	No	Partial	No
Browsers support	Multiple	Multiple	Multiple	Multiple

The CLI support is the main reason EJBCA was chosen in this work as new functions can be easily added into the existing libraries. We will discuss in details the integration of IBC libraries and EJBCA libraries.

3 Integration of IBC and EJBCA

3.1 Architecture of IBC framework

We have implemented another two extra modes except the original CA mode for EJBCA, namely, CA-PKG and PKG. CA-PKG mode allows one to initialize both CA and PKG to setup their corresponding public and private keys; PKG mode only allows one to initialize the PKG to setup the master public key and secret key. After the CA or PKG is initialized, administrator can start to generate certificates and private keys for users in CA and

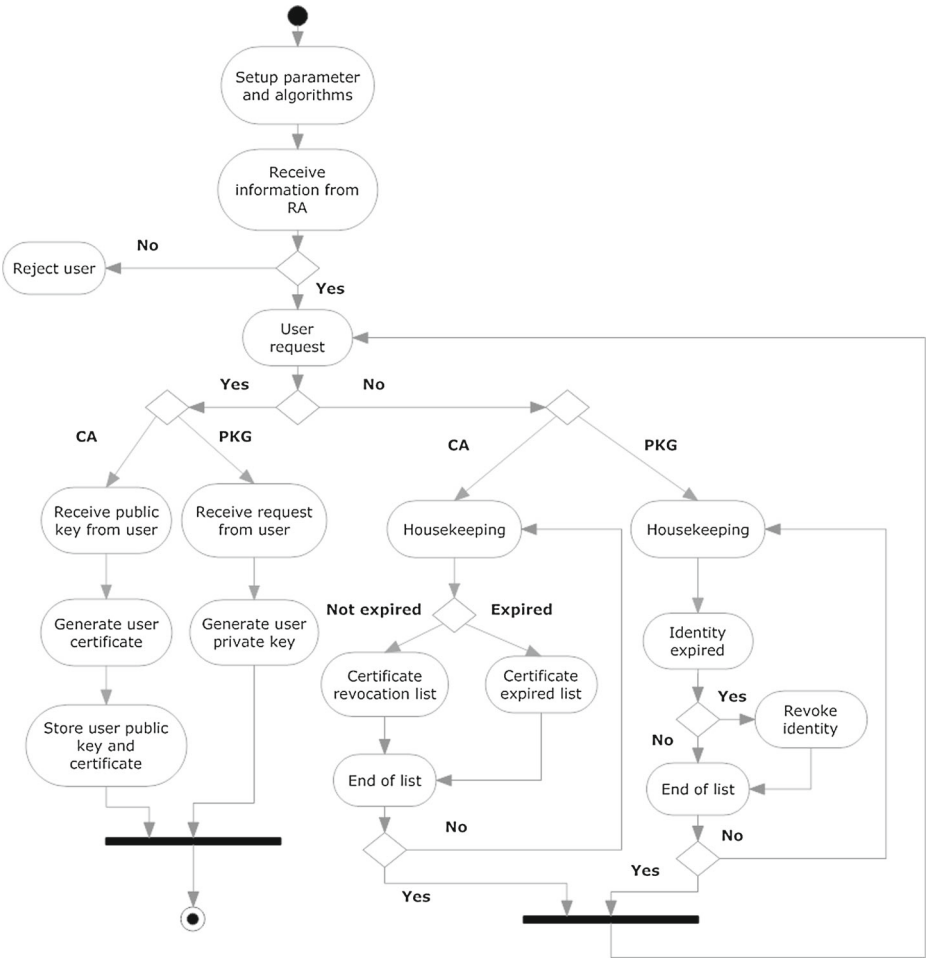

Fig. 3 Administrative activity diagram

PKG respectively. The system also checks the status of each certificate. If the certificate is expired, then it will put it into certificate expired list. If the certificate is not in good status, system will put it into certificate revocation list. On the other hand, PKG will check each of the identity. If the identity has been expired, it will revoke the identity. The administrative activity diagram is shown in Fig. 3.

3.2 Identity-based signature schemes

We setup the IBC framework for EJBCA based on the Shamir's IBS scheme [11], Schnorr's IBS scheme [10] and the EC Schnorr's IBS. Before going into the details of each IBS, we briefly describe the model of IBS as follows:

Setup(1^k). The PKG takes a security parameter (1^k) as input. It generates master secret key msk and master public key mpk. Note that mpk is publicly known, but msk will be known only to the PKG.

Extract(msk, ID). PKG run this algorithm to generate user secret key (usk). It takes msk and user public identity (ID) as input and returns usk to user.

Sign(usk, ID, M). User takes in usk, ID and message M as parameter to generate the corresponding signature σ.

Verify(mpk, ID, M, σ). To verify the signature, verifier checks if mpk, signed message M and the user ID are related to the corresponding signature σ.

3.2.1 Shamir identity-based signature scheme

We adopt the Shamir IBS presented in [11] which is proven secure against existential forgery under chosen message attack in the random oracle model assuming the RSA problem is hard:

Setup(1^k). The PKG takes a security parameter (1^k) as input and generates master public key $mpk = \{N, e\}$ where N is the product of two random prime numbers p and q, while $e \in \phi(N)$ where $gcd(e, \phi(N)) = 1$. PKG then calculates $msk = \{d\}$ where $d = e^{-1}$ mod $\phi(N)$ and chooses two hash functions $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$ and $G : \{0, 1\}^* \rightarrow \mathbb{Z}_e^*$.

Extract(msk, ID). PKG computes $x = \{H(ID)\}^d$ mod N and returns to the user, the user private key $usk = \{x\}$.

Sign(usk, ID, M). User randomly chooses $t \in \mathbb{Z}_N^*$ to compute $T = t^e$ mod N and $c = G(T||M)$. It then calculates $s = xt^c$ mod N and sets the signature as $\sigma = \{T, s\}$.

Verify(mpk, ID, M, σ). Verifier checks whether the equations $s^e = H(ID)T^{G(T||M)}$ mod N holds and accepts the signature if it is true, rejects it otherwise.

3.2.2 Schnorr identity-based signature scheme

We adopt the Schnorr IBS from [10] which is proven secure against existential forgery under chosen message attack in the random oracle model assuming the discrete logarithm problem is hard:

Setup(1^k). The PKG takes a security parameter (1^k) as input and generates $g \in \mathbb{Z}_p^*$ where p, q are prime numbers and $q|p - 1$. PKG then randomly chooses $msk = \{z\}$ and computes $Y = g^z$ mod p where $z \in \mathbb{Z}_q^*$. PKG sets $mpk = \{p, q, g, Y\}$.

Extract(msk, ID). PKG randomly chooses $r \in \mathbb{Z}_q^*$ to compute $y = r + z \cdot H(g^r, ID)$ mod q and returns the private key $usk = \{y, g^r\}$ to the user.

Sign(usk, ID, M). User randomly chooses $a \in \mathbb{Z}_q^*$ to compute $b = a + y \cdot G(ID, g^a, M)$ mod q and outputs the signature $\sigma = \{g^a, b, g^r\}$.

Verify(mpk, ID, M, σ). Verifier checks whether the equation $g^b = g^a(g^r g^{zc})^d$ mod p holds where $c = H(g^r, ID), d = G(ID, g^a, M)$ and accepts the signature if it is true, rejects it otherwise.

3.2.3 EC Schnorr identity-based signature scheme

We convert the Schnorr IBS [10] into the elliptic curve version as follows:

Setup(1^k). The PKG takes a security parameter (1^k) as input to initialize an elliptic curve $\mathbb{G}$ with prime order q and generates $g \in \mathbb{G}$. PKG then randomly chooses $msk = \{z\}$ and computes $Y = g^z$ where $z \in \mathbb{Z}_q^*$. PKG sets $mpk = \{\mathbb{G}, g, Y\}$.

Extract(msk, ID). PKG randomly chooses $r \in \mathbb{Z}_q^*$ to compute $y = r + z \cdot H(g^r, ID)$ mod q and returns the private key $usk = \{y, g^r\}$ to the user.

Sign(usk, ID, M). User randomly chooses $a \in \mathbb{Z}_q^*$ to compute $b = a + y \cdot G(ID, g^a, M)$ mod q and outputs the signature $\sigma = \{g^a, b, g^r\}$.

Verify(mpk, ID, M, σ). Verifier checks whether the equation $g^b = g^a(g^r g^{zc})^d$ holds where $c = H(g^r, ID), d = G(ID, g^a, M)$ and accepts the signature if it is true, rejects it otherwise.

3.3 Class diagrams

The main interface of CLI in EJBCA is the class `EjbcaEjbCli` which reads and lists all the available main and sub commands. The main command of each entity in EJBCA is specified in the classes `Base<entity>Command`: `BaseCaAdminCommand`, `BaseRaCommand` and `BaseAdminsCommand`; while the sub commands of each entity is specified in the calsses `<action>Command` such as `CaInitCommand`, `CaGetCrlInfo`, `RaAddUserCommand`, `AdminsAddAdminCommand`, to name a few. Figure 4 summarises the class architecture of CA.

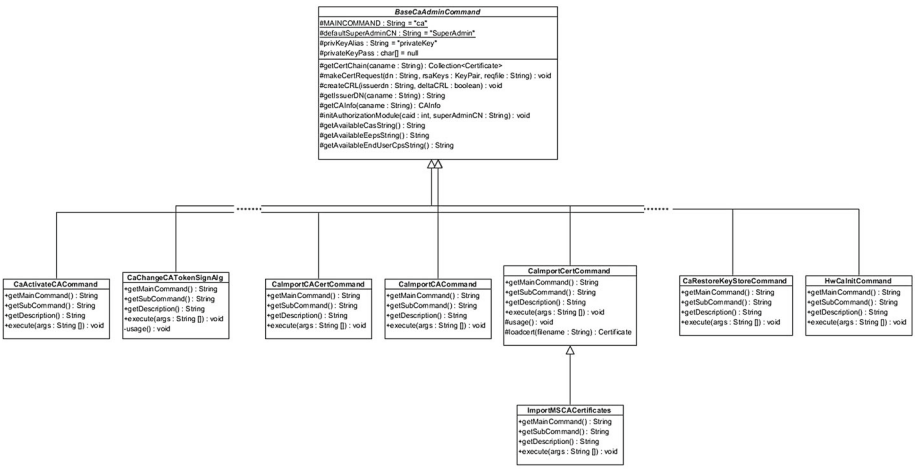

Fig. 4 Class diagrams for CA frameworks [6]

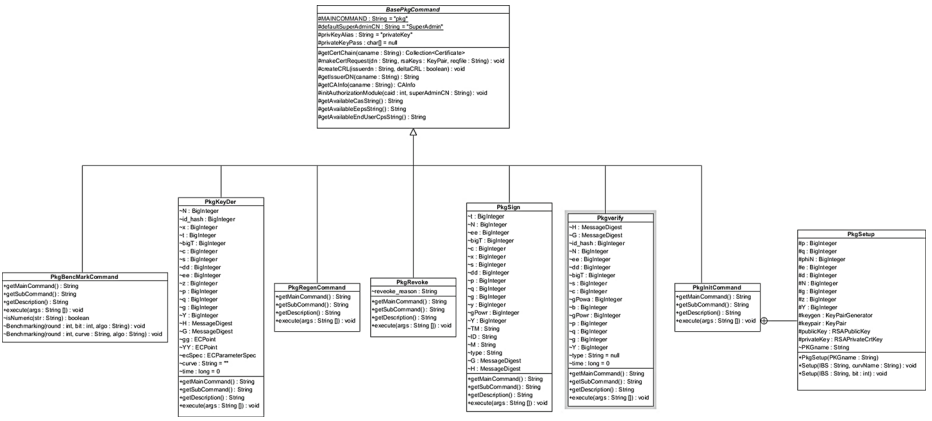

Fig. 5 Class diagrams for proposed IBC frameworks

In order to make sure IBC works fine with the existing EJBCA services, we follow the same class architecture of CA in implementing the IBC framework as shown in Fig. 5. `BaseCAPkgCommand` and `BasePkgCommand` specifies the main command for CA-PKG and PKG modes respectively. Revoke class is used to revoke user identity which is having invalid status while `<entity>InitCommand` class is used to initialize a CA-PKG or PKG. After a PKG is setup successfully, administrator can generate user private key by using the class `PkgKeyDer` where the algorithm will correspond to the algorithm chosen during initialisation. Although Sign and Veify algorithms of an IBS scheme are not run by the PKG but the users, in order to test the validity of the extracted user secret key, we prepare

```
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh capkg init CAPKG01 CN=AdminCAPKG,
O=EJBCA_Sample,C=SE soft null 1024 RSA 60 null SHA256WithRSA
Initializing CA
Generating rootCA keystore:
CA name: CAPKG01
SuperAdmin CN: SuperAdmin
DN: CN=AdminCAPKG,O=EJBCA_Sample,C=SE
CA token type: soft
CA token password: hidden
Keytype: RSA
Keyspec: 1024
Validity (days): 60
Policy ID: null
Signature alg: SHA256WithRSA
Certificate profile: ROOTCA
CA token properties: null
Signed by: self signed
Initalizing Temporary Authorization Module with caid=1728228641 and superadmin CN 'Supe
rAdmin'.
Creating CA...
CAId for created CA: 1728228641
-Created and published initial CRL.
CA initialized
RSA key type selected, PKG will be initialized with Shamir-IBS.
Creating PKG...
Time taken for Setup() is: 223108524ns
PKG name is : CAPKG01
PKG security parameter is : 1024
Setup sucessfull
```

Fig. 6 CA-PKG Initialization

```
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh pkg
Available sub commands for 'pkg':
  Revoke               Revoke identity which is in bad status
  benchmark            Benchmark PKG's operations.
  init                 Create a Pivate Key Generator (PKG).
  key                  Key derivation of end user private key.
  regen                Regenerate PKG's master private key
  sign                 Generate a signature using ID as Public Key. ID is email_addr||due_date
  verify               Verify a signature using ID as Public Key.
```

Fig. 7 Supported PKG operations

the `PkgSign` classes and `PKGVerify` classes as well. We do not show the class diagram of CA-PKG as one can view it as the combination of class diagrams of CA and PKG.

4 Performance evaluation

We test the performance of the IBC framework in Ubuntu Server 13.10 64-bit installed on server with Xeon E5-2407 2.20Ghz, 8GB RDIMM.

4.1 Screenshots

Figure 6 shows the command to initialize CA+PKG. To initialize a new CA, administrator has to specify certain parameter such as CA name, key length, chosen signature. Subsequently, the PKG will be initialized according to the algorithm chosen for CA, i.e. Shamir IBS for RSA, Schnorr IBS for DSA or EC Schnorr IBS for ECDSA. Since the focus of this work is IBC, we will show only the PKG operations in the remaining of this section.

```
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh pkg init PKG01 prime256v1 ECDSA
ECDSA key type selected, PKG will be initialized with EC-Schnorr-IBS.
Creating PKG...
Time taken for Setup() is: 52682114ns
PKG name is : PKG01
PKG security parameter is : prime256v1
Setup sucessfull
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh pkg key tan@mail.com 365 PKG01
Generating EC-Schnorr-IBS user private key...
Time taken for KeyDer is: 77022919ns
Your public key is: tan@mail.com20150311
Please renew before: 20150311
User secret key has been store: /home/tan/Desktop/ejbca_4_0_16/PKG/KeyDer/tan@mail.com2
0150311.txt
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh pkg sign tan@mail.com20150311 "Th
is is message to be signed." PKG01
Time taken for Signing is: 70098296ns
Signature stored in: /home/tan/Desktop/ejbca_4_0_16/PKG/Sign/tan@mail.com20150311.txt
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh pkg verify tan@mail.com20150311 "
This is message to be signed." PKG01
Time taken for Verify is: 111551887ns
verify
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh pkg verify tan@mail.com20150311 "
This is XXX message to be signed." PKG01
Time taken for Verify is: 115621677ns
rejected
```

Fig. 8 Demonstration of PKG operations

```
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh pkg Revoke
Description: Revoke identity which is in bad status
Usage: pkg Revoke <Public Key> <PKGname> <reason>
unspecified(0)
keyCompromise(1)
CACompromise(2)
affiliationChanged(3)
superseded(4)
cessationOfOperation(5)
certificateHold(6)
removeFromCRL(8)
privilegeWithdrawn(9)
AACompromise(10)
tan@James:~/Desktop/ejbca_4_0_16/bin$ sudo ./ejbca.sh pkg Revoke tan@mail20150311 PKG01 9
tan@mail20150311 has been added to corresponding CRL.
```

Fig. 9 User private key revocation

Figure 7 shows the operations available for the proposed PKG. Figure 8 shows the major operations of a PKG. It shows how to generate a user secret key for an end user. Administrator will have to key in the public identity of end user, the identity validity period and under which PKG the user belongs to. When the identity is expired, i.e. the date shown in email address is invalid, email sender will understand that this email address is no longer secure and thus achieve the purpose of user revocation.

In implementation, a system can be configured such that as long as the date on the email address is expired, then it is an invalid recipient. On the other hand, if a user noticed that his email address is expired, he will need to contact the PKG to update a new one. PKG also can revoke user private key as shown in Fig. 9.

4.2 Performance

To test the performance for the PKG framework, we run each service for 100 rounds to get an average time for Shamir IBS, Schnorr IBS and EC Schnorr IBS. We noticed that the first few rounds will require longer time due the the fact that the required codes have not been placed in processor's cache yet. In view of this, we run each service for 110 rounds but only the 10th to 110th rounds are used in calculating the average timing. Table 2 shows the required time for each service in second.

The Setup of each IBS scheme follows strictly the requirement stated in FIPS 186-3 [8] and thus resulted in the rather lengthy time. For the EC-based IBS, the setup is significantly faster than that of RSA-based and DSA-based schemes because we use the curves prime192v1, prime224v1 and prime256v1 of X9.62[1] [1] from BouncyCastle which has the curve parameters fixed, instead of generated randomly. However, we argue that *Setup* needed to be done only once during the initialization of PKG and the *Extract* also needed to be run only once for each user. *Sign* and *Verify* on the other hand will be run multiple times but the experiments showed that their performance are very practical where by the total of these two services in a single round of 3072 bits keys does not even exceed 0.05, 0.06 and 0.1 second for Shamir IBS, Schnorr IBS and EC Schnorr IBS respectively.

[1]These three elliptic curves are having the similar security level corresponding to 1024, 2048 and 3072 in RSA and DSA.

Table 2 Performance of IBC Framework in EJBCA

Scheme	Service	Performance (s)		
		1024 bits	2048 bits	3072 bits
Shamir IBS [11]	Setup	0.152	0.914	3.720
	Extract	0.006	0.039	0.125
	Sign	0.004	0.011	0.025
	Verify	0.003	0.011	0.025
Schnorr IBS [10]	Setup	19.024	138.176	379.533
	Extract	0.001	0.006	0.014
	Sign	0.002	0.006	0.013
	Verify	0.003	0.016	0.038
EC Schnorr IBS	Setup	0.009	0.012	0.016
	Extract	0.020	0.023	0.032
	Sign	0.020	0.024	0.033
	Verify	0.036	0.046	0.063

5 Conclusion

We amended the architecture of EJBCA by incorporated an IBC framework into it. The enhanced EJBCA can now be deployed as PKI and/or IBC according to the needs of an organization. The IBC services were experimented and the performance showed that it is practical for real world usage. The future plan is to enrich the IBC services with more choices of IBS scheme and also other primitives such as identity-based encryption, key exchange as well as identification schemes.

Acknowledgments This research is partially supported by FRGS Grant (FRGS/1/2012/TK06/MMU/03/9) and TM R&D Grant (RDTC/130827).

References

1. ANSI X9.62-2005 (2005). Elliptic curve digital signature algorithm, ECDSA
2. Carlisle Adams SL (1999) Understanding the Public-key infrastructure: concepts, standards, and deployment considerations, 1st edn. Sams
3. Chen L, Harrison K, Moss A, Soldera D, Smart NP (2002) Certification of public keys within an identity based system. Information security, vol 2433. LNCS, pp 322–333
4. Chen Q, Li Z, Yu S (2007) A cross-authentication model for heterogeneous domains in active networks. IFIP Int Conf Netw Parallel Comput Workshops:140–143
5. Dalton CR (2003) The NHS as a proving ground for cryptosystems. Inf Sec Techn Report 8:73–88
6. EJBCA. http://www.ejbca.org/index.html
7. Eslami Z, Kazemnasabhaji M, Mirehi N (2013) Proxy signatures and buyer-seller watermarking protocols for the protection of multimedia content. Multimedia tools and applications. doi:10.1007/s11042-013-1555-0
8. (2009). 186-3, FIPS PUB, Digital Signature Standard (DSS)
9. FlexiProvider. http://www.flexiprovider.de/
10. Galindo D, Garcia FD A schnorr-like lightweight identity-based signature scheme. In: Proceedings of AfricaCrypt '09, LNCS, vol 5580. pp 135–148
11. Kiltz E, Neven G Identity-based signatures. In: Proceedings of the CISS '08, vol 2. pp 31–44

12. Krishnamurthy S (2008) Understanding the successes of identity-based encryption. NIST identity-based encryption workshop
13. Lee B (2010) Unified public key infrastructure supporting both certificate-based ID-based cryptography. IEEE Int Conf Availability, Reliab Secur:54–61
14. Legion of the Bouncy Castle. https://www.bouncycastle.org/
15. Network Security Services (NSS). https://developer.mozilla.org/en/docs/NSS
16. Open Source Certificate Authority (OpenCA) Labs. http://www.openca.org/
17. Open Java Development Kit (OpenJDK). http://openjdk.java.net/
18. Open Source Secure Socket Layer (OpenSSL) Project. https://www.openssl.org/
19. Price G, Mitchell CJ (2005) Interoperation between a conventional PKI and an ID-based infrastructure. Public key infrastructure, vol 3545. LNCS, pp 73–85
20. Rong R, Li Z, Jiang Y (2007) An authentication model for multi-type domains in active networks. IEEE international workshop on anti-counterfeiting, security, identification
21. Al-Riyami SS, Paterson KG Certificateless public key cryptography. In: Proceedings of the Asiacrypt '03, vol 2894. LNCS, pp 452–473
22. Shamir A Identity-based cryptosystems and signature schemes. In: Proceedings of the CRYPTO '84, vol 196. LNCS, pp 47–53
23. Stoianov N, Urueña M, Niemiec M, Machnik P, Maestro G (2013) Integrated security infrastructures for law enforcement agencies. Multimedia tools and applications. doi:10.1007/s11042-013-1532-7
24. Voltage Security Press Release (2006). Total cost of ownership of Voltage IBE 3X lower than PKI in Ferris research study. http://157.238.212.45/pressreleases/PR060530.htm
25. X Certificate and Key Management (XCA). http://xca.sourceforge.net/
26. Yi X, Zheng G, Li M, Ma H, Zheng C (2013) Efficient authentication of scalable media streams over wireless networks. Multimedia tools and applications. http://dx.doi.org/10.1007/s11042-012-1324-5

Syh-Yuan Tan received his Bachelor degree in I.T. and M.Sc. from Multimedia University (MMU), Malaysia in 2007 and 2010. He is currently attached to the Faculty of Information Science and Technology, MMU, Malaysia as a lecturer. His research interests include Cryptography and Network Security.

Wei-Chuen Yau is a Senior Lecturer in the Faculty of Engineering at Multimedia University, Malaysia. He received his B.S. and M.S. degrees from National Cheng Kung University, Taiwan, and his Ph.D. degree from Multimedia University. He also holds CISSP (Certified Information Systems Security Professional), GSEC (GIAC Security Essentials), and HCDA (Huawei Certified Datacom Associate) certifications. His research interests include cryptography, security protocols, intrusion detection, and network security.

Boon-Hock Lim is a Bachelor I.T. student of the Faculty of Information Science and Technology in Multimedia University, Malaysia. His research interests include cryptography and network security.