symmetry

MDPI

Article

An Implementation Suite for a Hybrid Public Key Infrastructure

Jason Chia [1,*], Swee-Huay Heng [2,*], Ji-Jian Chin [3], Syh-Yuan Tan [4] and Wei-Chuen Yau [5]

1 Faculty of Engineering, Multimedia University, Selangor 63100, Malaysia
2 Faculty of Information Science and Technology, Multimedia University, Melaka 75450, Malaysia
3 Faculty of Computing and Informatics, Multimedia University, Selangor 63100, Malaysia; jjchin@mmu.edu.my
4 School of Computing, Newcastle University, Newcastle upon Tyne NE4 5TG, UK; syh-yuan.tan@newcastle.ac.uk
5 School of Electrical and Computer Engineering, Xiamen University Malaysia, Sepang 43900, Malaysia; wcyau@xmu.edu.my
* Correspondence: 1161300548@student.mmu.edu.my (J.C.); shheng@mmu.edu.my (S.-H.H.)

Abstract: Public key infrastructure (PKI) plays a fundamental role in securing the infrastructure of the Internet through the certification of public keys used in asymmetric encryption. It is an industry standard used by both public and private entities that costs a lot of resources to maintain and secure. On the other hand, identity-based cryptography removes the need for certificates, which in turn lowers the cost. In this work, we present a practical implementation of a hybrid PKI that can issue new identity-based cryptographic keys for authentication purposes while bootstrapping trust with existing certificate authorities. We provide a set of utilities to generate and use such keys within the context of an identity-based environment as well as an external environment (i.e., without root trust to the private key generator). Key revocation is solved through our custom naming design which currently supports a few scenarios (e.g., expire by date, expire by year and valid for year). Our implementation offers a high degree of interoperability by incorporating X.509 standards into identity-based cryptography (IBC) compared to existing works on hybrid PKI–IBC systems. The utilities provided are minimalist and can be integrated with existing tools such as the Enterprise Java Bean Certified Authority (EJBCA).

Keywords: digital certificates; identity-based cryptography; public key infrastructure; X.509

Citation: Chia, J.; Heng, S.-H.; Chin, J.-J.; Tan S.-Y.; Yau W.-C. An Implementation Suite for a Hybrid Public Key Infrastructure. *Symmetry* **2021**, *13*, 1535. https://doi.org/10.3390/sym13081535

Academic Editors: José Carlos R. Alcantud and Mihai Postolache

Received: 22 June 2021
Accepted: 17 August 2021
Published: 20 August 2021

1. Public Key Infrastructure

Public key infrastructure (PKI) provide a means to verify authenticity and encrypt messages sent over insecure channels [1,2]. It has become an industry standard because it secures the use of asymmetric cryptography against man-in-the-middle attacks, which solves the key distribution problem [3]. Briefly, a PKI is used to generate and distribute digital certificates which bind random-looking public keys to a public name of the owner. The PKI used in the industry today typically consist of a chain of trust, depicted in Figure 1. A chain of trust enables users who trust a root trust to then further trust certificates which are issued by the root. The intermediary CA is "trusted" by users because they trust the root, which vouches for the intermediary by issuing them a certificate; users will then trust certificates issued by the intermediary. This hierarchical nature enables businesses to setup their own private PKI, which provide a multitude of authentication services for their day-to-day operations (e.g., code signing, file server authentication, etc.) [4].

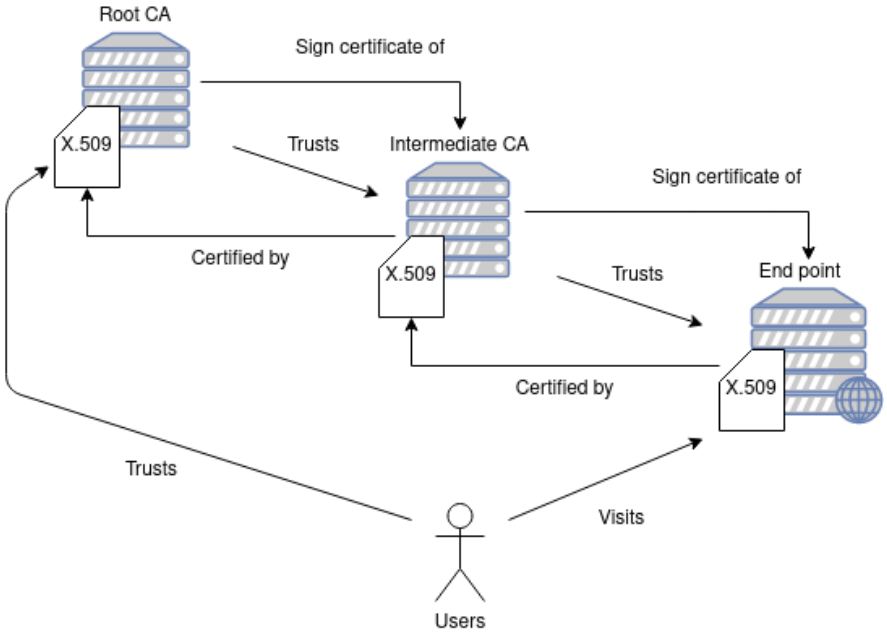

Figure 1. Chain of trust. The root CA is self-certified.

Figure 2 illustrates the process of obtaining and using a digital certificate for to authenticate the communication between an Alice and Bob. To obtain a digital certificate, Bob first presents identifying documents and a certificate signing request (CSR) to a registration authority (RA). Upon validation of the documents, the RA forwards the CSR to a CA for signing, which then generates the digital certificate for Bob. When Alice wants to communicate with Bob, she first requests Bob's certificate from a public directory and checks whether the certificate is authentically signed by a trusted CA. This process establishes the authenticity of Bob's public key which Alice then uses for communication. To skip this process would make Alice and Bob susceptible to a man-in-the-middle attack. Notably, the process is time consuming and poses an additional overhead on Alice's side (i.e., the client side) [5].

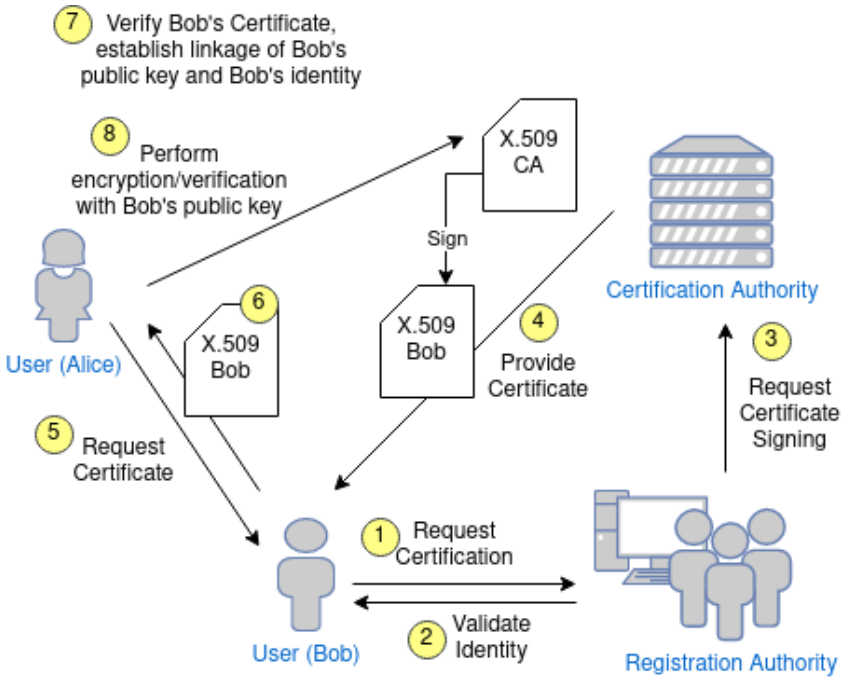

Figure 2. Public key infrastructure.

In contrast to the conventional PKI, identity-based cryptography (IBC) eliminates the need for certificates altogether by creating public keys which are not random looking. Essentially, in identity-based cryptosystems, the public key is a publicly verifiable string which can be used for encryption and verification [6]. Figure 3 depicts an identity-based cryptosystem, where the process of certificate validation is no longer necessary. This is because Bob's public key is now their identity, through which Alice can easily verify this fact without the need for a digital certificate. In the case of an IBC, a private key generator (PKG) replaces the role of a CA by generating a user key corresponding to the identity of the user.

A study by Bai has also found other advantages of an IBC environment against a PKI environment, primarily on lesser bandwidth, lower computational requirements, lower storage space demand and the ability to perform decryption/verification offline [7]. However, revocation becomes an issue if the private key for Bob is leaked. The identity "bob@mmu.edu.my" can never be used again unless the PKG regenerates its master keys. As mentioned by Bai, the delivery of user private keys (UPK) also poses an issue, since in a PKI setting, users may opt to generate the private key privately and only transmit the CSR across to the RA for certification.

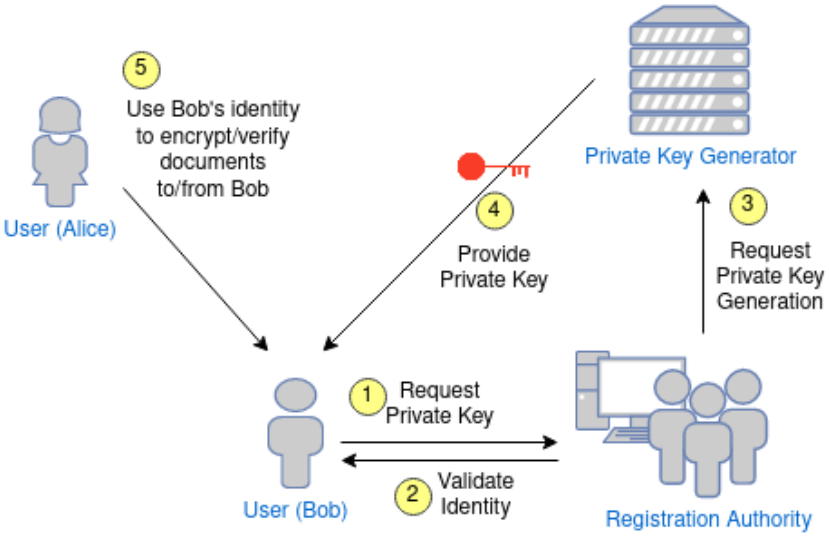

Figure 3. Identity-based cryptosystem.

1.1. Motivation

PKI and IBC are suitable for different scenarios. PKI systems are favorable in systems where key leakages are common (e.g., Internet) while IBC systems are suitable for closed systems (e.g., corporate intranets). Due to the hierarchical nature of PKI with chain of trust, companies may establish their own PKI and maintain certificates of internal services and users that can also be used externally. This would be very costly for companies with a lot of employees and internal services, as most commercial PKI charge for their services based on the number of users. Recently, a sizeable workforce has switched over to working remotely due to the COVID-19 pandemic. This has spurred organizations to begin using PKI to secure their employee's remote VPN connection (a VPN or virtual private network connection allows employees to access an organization's network securely from their home through a "network tunnel". Traditionally, users would logon to a VPN using a username–password combination, which is risky because bad password hygiene could easily [8] expose the VPN to seasoned password crackers) [9]. It is reasonable to expect that in order to address such a surge in demand, the development of a hybrid PKI system built as a hierarchical extension to include IBC would be useful for the IT infrastructure of various organizations. In addition to maintaining trust in PKI established by the IT industry, a hybrid PKI–IBC system would enjoy the following benefits:

1. Cost effectiveness: an organization can communicate securely both internally and externally with the cost of only one digital certificate, where the cost is independent of the number of staff in the organization. This benefit can never be achieved using conventional PKI in which each staff is required to possess an individual certificate to authenticate their own public key;
2. Simple key management: inherited from the distinctive certificate-less feature of IBC, PKG can go offline upon issuing the user private key for users. Users can then communicate in a peer-to-peer manner in contrast to PKI users who need to communicate with CA to verify the authenticity of their peer's digital certificates.
3. Database-less: Certificates are not necessary in IBC, thus no certificate database storage is required.

1.2. Problem Statement

Operating a private PKI to secure internal communications is a costly endeavor due to the use of digital certificates [5]. IBC is a lightweight alternative that alleviates much of the cost because it does not require digital certificates. However, IBC is not suitable for external communication because users from external domains do not inherently trust the PKG. In this work, we aim to address the incompatibility of PKI and IBC through what we call a hybrid PKI system. Such a system allows the external communication of the organization to derive their security from the low-cost IBC system by piggybacking on the trust established by a PKI.

1.3. Organization

We introduce PKI and state the problem we aim to tackle in Section 1. In Section 2, we present the existing literature on this topic and state our contribution in Section 3. Our methods and plan for implementation is presented in Section 4. In Section 5, we present the architecture of a hybrid PKI. In Section 6, we briefly discuss and state the algorithms which are used in our implementation. We show screen captures of the functionality and the features of our work in Section 7. A comparison with existing works is performed in Section 8, followed by our conclusion of this work in Section 9.

2. Existing Works

The earliest idea of a hybridized PKI dates back to 2005. Price and Mitchell [10] proposed a total integration of IBC with PKI. One of their findings indicates that backward compatibility issues are due to the certification policy standard for X.509. Their proposed solution includes stating policy statements in a cross-domain certificate issued by the trust authority (TA) of an IBC or just generating new identity-based keys for the conventional certificate user which eliminates the benefits of a lightweight and low-cost trust infrastructure. We note that these concerns are moot if the IBC end-users will not use their identity-based keys for further certificate issuance. Chen et al. extended Price and Mitchell's work in 2007 with a design specific to Active Networks only [11].

Lee proposed a different idea of having both PKI and IBC together instead of complete integration, known as Unified PKI [12]. Unified PKI uses certificate-less cryptography instead of IBC and assumes the availability of key privacy services. Tan et al. introduced a true PKI–IBC hybrid, whereby the trust authority contains both a private key generator (PKG) for identity-based keys and a conventional CA [13]. Their work integrates with the popular open source PKI software Enterprise Java Bean Certificate Authority (EJBCA), with their IBC modules built-in as command line extensions to the existing EJBCA software. However, their approach binds the CA wishes to have a hybrid trust infrastructure to use EJBCA instead of other software (i.e., OpenCA, NSS).

In 2016, Reimair et al. presented a solution for PKI on a multi-device user, known as the Cryptographic Service Interoperability Layer or CrySIL [14]. While the solution was intended to solve a different problem, namely to address the storage of cryptographic keys across various devices, they proposed that it could be used to emulate IBE/ABE

(ABE stands for attribute-based encryption, which is a more general form of identity-based encryption, in that the public key of a user is a set of attributes instead of an identity string) systems [15] using X.509 standard.

The current trend for PKI is to move towards a decentralized block-chain-based system to mitigate the effects of the single point of failure on the CA [16], in addition to privacy preservation concerns [17]. For instance, a trust enhancement scheme by Chen et al. [18] attempts to transfer some authority of the CA to a decentralized block-chain, while Chiu et al. proposed a decentralized PKI-based on the Ethereum block-chain to better alleviate privacy concerns for users of the PKI [19]. While these studies are progressive for PKI development, there is a gap in addressing the integration of PKI and IBC systems.

3. Our Contribution

We propose a design and implementation for a hybrid PKI framework that has the following features:

1. Independent of the underlying software used by the CA.
2. Users from domains issued by the CA can decrypt/verify identity-based encrypted/signed messages from users on the hybrid PKI's domain.

Essentially, we introduced an extension to existing PKI infrastructures instead of a replacement, allowing our design to be easily adapted into available PKIs. This is key because PKI has been an industrial standard and is key to security and networking infrastructures in many organizations, thus replacing a PKI would thus be very costly. We note that our extension should be protected the same way that a PKI is protected: a compromise in the PKG of the extension should warrant revocations of relevant keys from the issuing CA, similarly to the compromise of a PKI warranting the revocation of its issued certificate by a CA further up the chain of trust. Ideally, the management of the hybrid PKI should follow the same policy as the management of a conventional PKI.

The hybrid PKI that we propose will be especially beneficial in the following scenarios:

- An organization wants to have an IBC security environment in their organization without an existing conventional PKI.
- An organization wants to reduce the cost incurred by end-user certificates generated by conventional PKI by replacing end-user certificates in the organization with IBC solutions.

In each of the scenarios above, we make a basic assumption that communication within the organization will be using IBC, while communication with external entities not in the organization would employ conventional PKI. Figure 4 visualizes that in any case, an internal user should be able to verify communication from another internal user as well as to verify communication from an external user which could either be using the IBC from their own organization or conventional PKI to communicate, respectively. Our communication model ensures backward compatibility with existing PKI.

In addition, we also implement additional plugins for Web browsers and the popular email client Mozilla Thunderbird to facilitate the use of the hybrid PKI framework.

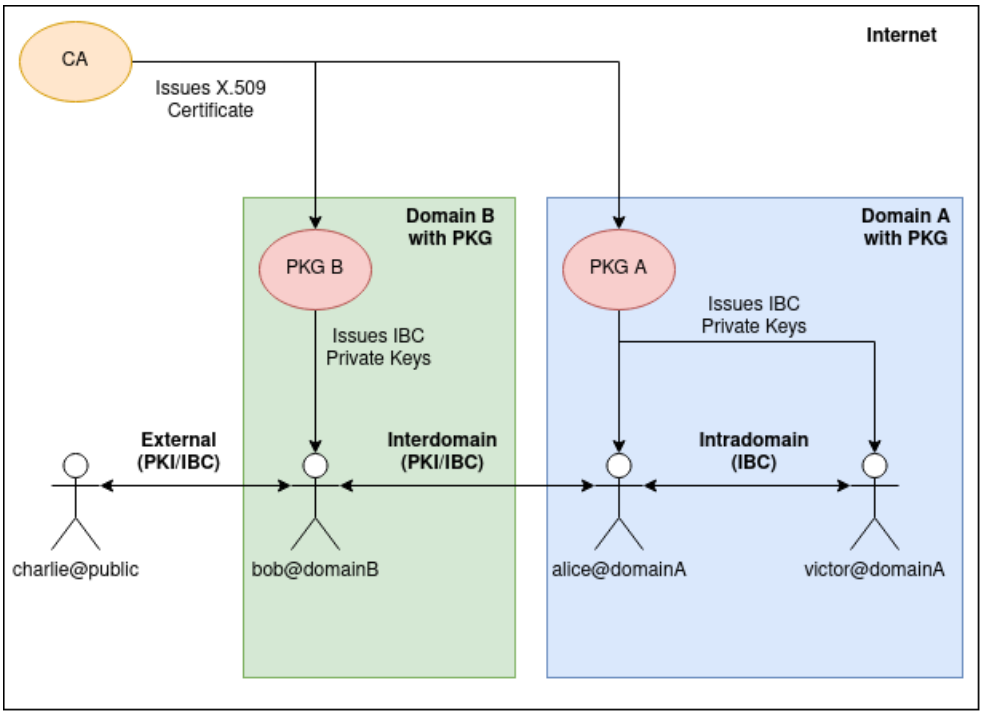

Figure 4. Communication model: the intra-domain communication may rely directly on IBC, but the inter-domain channel requires a combination of PKI/IBC methods to function correctly.

4. Methodology

As it stands, the best way to design such a system is to design a modular component for a PKG back-end service. This allows other software to easily integrate with the PKG service and thus achieve PKI software independence. In addition, the PKG service can also be deployed on different hardware, allowing hardware units to be created and deployed separately.

Another benefit of designing from a purely back-end module is that security developers can easily extend its features to incorporate their methods of access control and other front-end utilities. We stress again that our design merely provides an extension to existing PKI infrastructure and leaves access control up to the policies of the existing CAs.

The design shall employ platform-independent programming languages such as Python or Java. Similarly to Tan's existing work [13] on EJBCA, we chose the Java programming language as it also comes with plethora of choices in terms of cryptographic libraries. Two cryptographic libraries were chosen and used, namely Bouncy Castle [20] (BC) and Apache Milagro Crypto Library [21] (AMCL). We chose BC due to its capability on X.509 certificate manipulation and AMCL due to some elliptic curve cryptographic (ECC) primitives that were missing on BC.

5. Architecture

Figure 5 refers to the back-end system architecture. The CA is modeled as a black box because we make no assumptions on the underlying PKI software. The PKG back-end service is able to interact with Web interfaces and the CA directly. The back-end module is interfaced by a socket server (i.e., transmission control protocol (TCP) or user datagram protocol (UDP) sockets) which listens only on the local network interfaces of the machine (i.e., localhost). Security developers would then build access control or Web application software which connects to the back-end module via network sockets and interacts with the PKG.

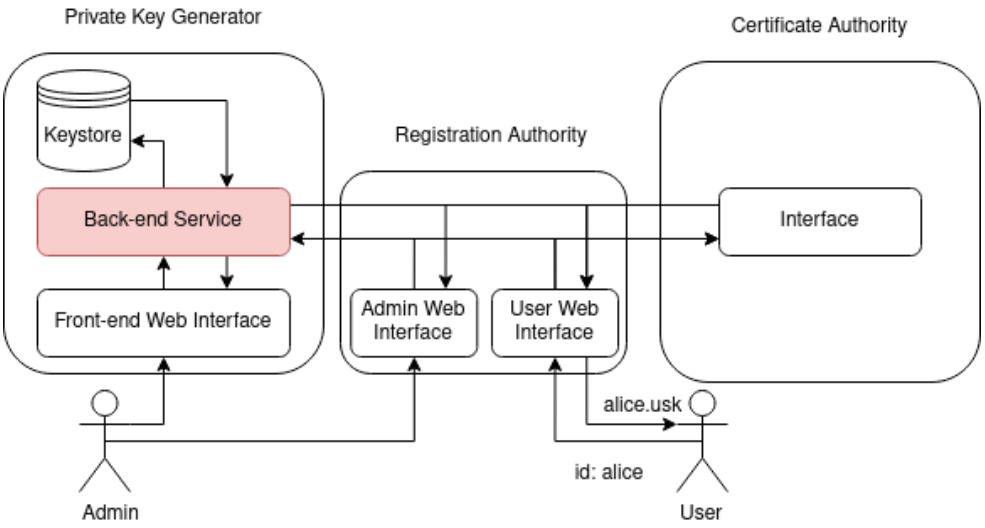

Figure 5. Back-end system architecture. The front-end, admin and user Web interfaces can be custom built to interact with the service.

The back-end service holds a set of master private keys for user private key generation as well as a conventional cryptographic key (i.e., RSA, ECDSA) shown in Figure 6: The conventional key (ckey) is used to generate the appropriate cryptographic message syntax or CMS for its master public keys (MPK). The CMS proves to external verifiers that the master public key is legitimate and is sanctioned by a root trust (i.e., a mutually trusted CA). In other words, external/inter-domain users would first validate a CMS against a root trust, obtain the master public key from it if it is found to be valid, and then only perform identity-based verification/encryption. To interact with the back-end, a client (i.e., Web server or an access control middle-ware) first initiates a socket connection to the back-end, then sends in the payload containing a JSON (JSON or Javascript Object Notation is a popular format for communicating semi-structured data) formatted service request as well as the parameters. Table 1 shows a list of functionalities provided by the back-end module.

5.1. Intra-Domain Communication

Suppose domain A contains a PKG as well as users Alice and Victor. Alice would like to sign a document for Victor to verify. Alice and Victor are both organizational members of domain A and thus received their private keys from the domain A's PKG. Alice and Victor intrinsically trust domain A's PKG, as they are using private keys derived from it.

Alice would sign the document with her identity-based private key and send the document along with the signature to Victor, which can then verify her signature along with domain A's PKG. Since Victor trusts domain A's PKG, verification is straightforward as described by the underlying identity-based signature protocol. This interaction is shown in Figure 7.

5.2. Inter-Domain Communication

Suppose domain A contains a PKG and user Alice, while domain B contains another PKG and user Bob. Alice would like to sign a document for Bob to verify. Alice and Bob are from different organizations and do not intrinsically trust the each other's PKG, as either could be impersonators. A naive approach would be to simply let the CA of domain A issue a X.509 certificate for Alice, which Bob could then verify that Alice is indeed from domain A. This is the current solution for the organization with a private PKI. However, this would mean that the organization would need a linear amount of certificates corresponding to its employees who wish to communicate externally.

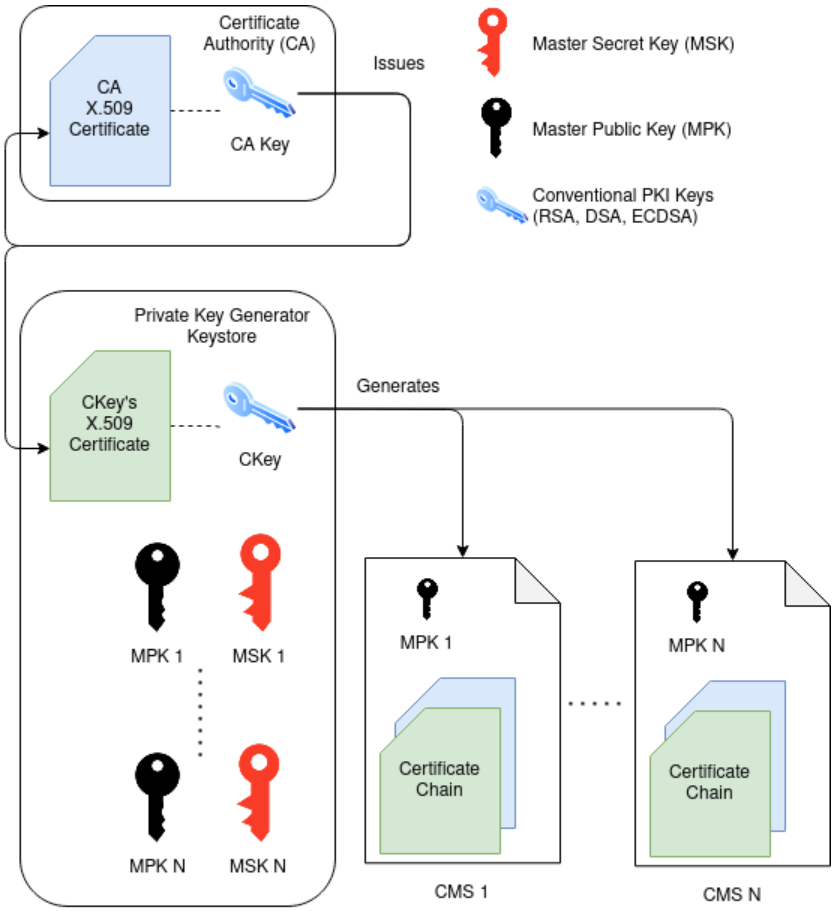

Figure 6. Keys on the back-end key-store. Note that the CMS is for external users to validate the authenticity of the PKG (see Section 5.2); internal users would simply use the MPK without needing to obtain the CMS from the PKG.

Table 1. Back-end TCP service functions for key-store management.

Function	Description	Service
Create key-store	Initialize a new key-store. Key-stores are used to contain master secret keys which are used to derive user keys from user identities. Each key-store is protected with a master password and consists of a conventional key (known as the ckey).	`kmnew`
Create CSR	Creates a certificate signing request for the ckey in the key-store. The ckey is used to sign master public keys.	`kmcsr`
Add certificate	Register a certificate of ckey signed by a certificate authority. This certificate is presented along with the signed master public key in the form of a cryptographic message syntax (CMS).	`kmncr`
View certificate	Obtain the certificate of the ckey in PEM format.	`kmvcr`
List key-alias	List out the master key aliases available in the key-store.	`kaget`
Load key-store	This service must be called ONCE on every startup to load the key-store into memory.	`kmlod`
Create master key	Create a new master key pair, each master key is protected with a master key password independently from the key-store password	`mskini`
Delete master key	Delete a master key	`mskdel`

Table 1. *Cont.*

Function	Description	Service
Derive user private key	Derive a user private key given a user ID. An available option to encrypt the user private key is a password which is supplied.	`uskder`
Get master public key	Obtain a master public key of a corresponding key alias.	`mpkget`
Get signed master public key	Obtain a signed master public key in the form of a CMS.	`mpkcms`

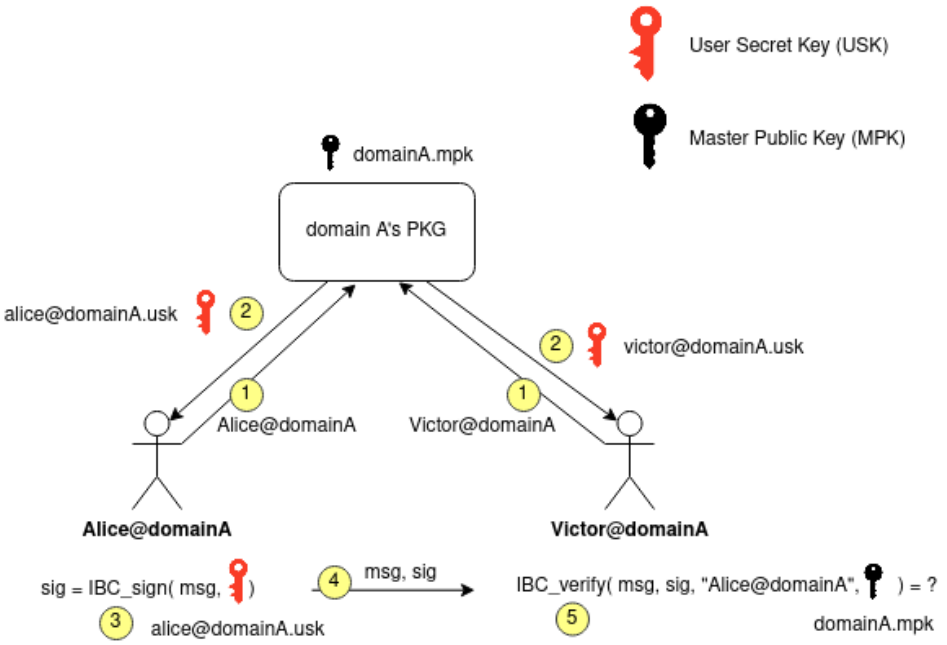

Figure 7. Intra-domain communication. Example shown here is for signing/verification, but it works with encryption/decryption as well. In the case of encryption/decryption, Alice would encrypt with Victor's identity "Victor@domainA" and Victor would decrypt with their user key victor@domainA.usk.

To overcome this, the CA of domain A only issues a X.509 certificate for a key-pair known as the certification key or "CKey". Figure 6 shows that the CKey is issued by the CA, which in turn is responsible for generating the CMS statements for the MPK in the key-store. Note that based on X.509 standards, the CMS contains a certificate chain up to the root CA. In other words, if Bob receives the CMS, he would be able to achieve two things:

1. Verify that domain A is a legal domain;
2. Obtain that domain A's authentic MPK.

Thus, our inter-domain communication flow shown in Figure 8 works as follows: Alice would first request a CMS statement from domain A' PKG and then send it to Bob. Bob from domain B would then verify the validity of the CMS statement using a certificate from a CA that he trusts. If domain A's PKG is legitimate and the CKey certificate is obtained from a mutually trusted CA, Bob would successfully verify the CMS. Upon verification, Bob can unpack the CMS, obtaining domain A's MPK from it. The rest of the communication follows that of intra-domain communication, as the successful verification of the CMS established trust to domain A's PKG and its master public keys.

In the case that domain A and domain B do not share the same CA which issued their CKey, as long as either PKG is issued by a globally trusted CA (e.g., DigiCert Global Root CA), users from either domain can verify the CMS statements and establish trust with the

opposite PKG. In the case that an external user does not belong to any organization (e.g., public user), that user can also verify the CMS just as one would verify certificates when browsing the Web.

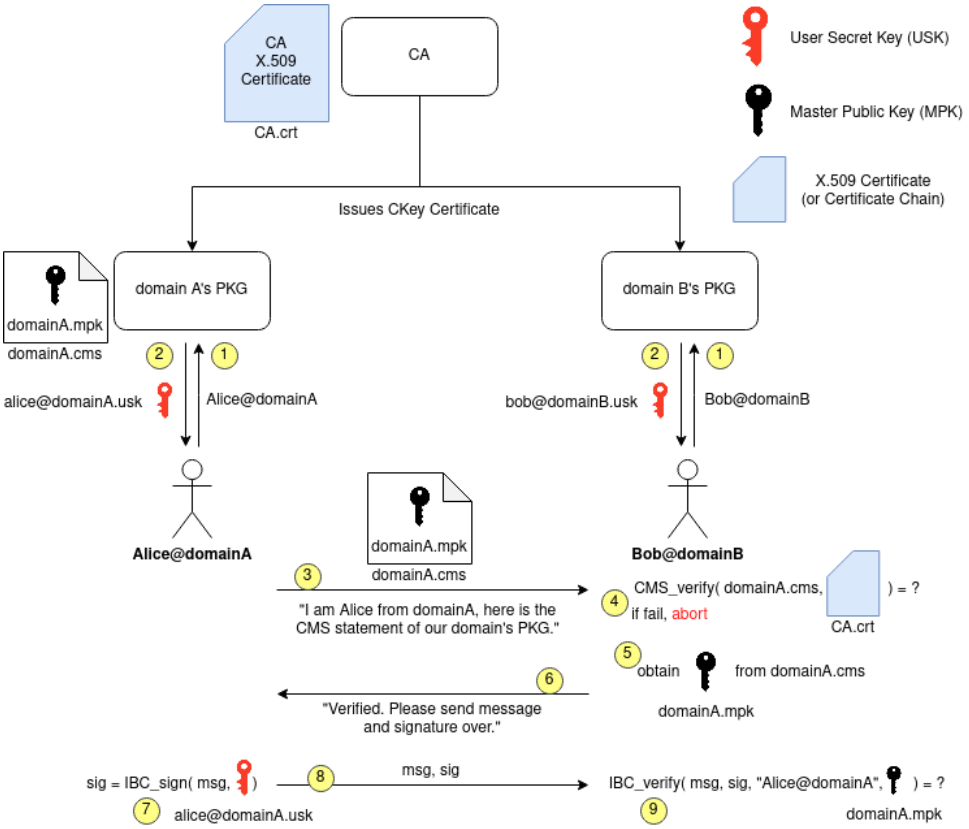

Figure 8. Inter-domain communication. Steps 3–5 are needed for Bob to establish trust of domain A and the MPK it receives.

6. Identity-Based Cryptography

In this work, we consider multiple identity-based cryptographic schemes into our hybrid PKI implementation suite. In total, we incorporate the same IBS schemes that were supported by Tan et al.'s work in 2014 and include an additional IBE scheme to provide message encryption capabilities to our system. We provide brief reasons as to why the schemes were chosen to be incorporated in the following sections.

6.1. RSA Identity-Based Signatures

The RSA identity-based signature (IBS) scheme proposed by Kiltz et al. [22] was incorporated into our work. The scheme was proven secure against existential forgery under chosen message attack in the random oracle model based on the difficulty of the RSA problem. The RSA IBS scheme is included because of the widespread use of the RSA-based key pairs in PKI; existing formats such as PKCS#1 can be re-used to store the MPK. The four algorithms of RSA IBS are shown in Algorithms 1–4.

6.2. Schnorr Identity-Based Signatures (DSA IBS)

Schnorr IBS, proposed by Galindo and Garcia in 2009 [23], is another scheme incorporated into our work. This IBS was proven secure against existential forgery under the chosen message attack in the random oracle model based on the difficulty of the discrete logarithm problem. We chose to include DSA IBS because DSA-based key pairs, similarly to RSA-based ones, are also industry standard and have undergone much standardization. For instance, the format for storing DSA public keys, ANSI X9.57, can be re-used to store the MPK. The four algorithms of Schnorr IBS are shown in Algorithms 5–8.

6.3. Elliptic Curve Schnorr Identity-Based Signatures (ECDSA IBS)

Similarly to Tan et al. [13], we convert Schnorr IBS to be usable with elliptic curves. The main reason for doing so is to provide an additional option of having shorter key sizes, in addition to ECDSA being an industry standard. The X9.62 standard for ECDSA signatures is re-used to store the MPK. The algorithms mainly follow Algorithms 5–8 in Section 6.2. A notable difference is that instead of doing arithmetic over an integer, ECSchnorr performs its operations over an elliptic curve. This means that for a finite group with prime order, q is now an elliptic curve $\mathbb{G}$ with a prime order q and arithmetic operations are performed on an additive group instead of a multiplicative one.

6.4. Boneh–Franklin Identity-Based Encryption

Our work also includes the first practical IBE scheme that was designed by Boneh and Franklin in 2001 [24]. We chose Boneh and Franklin's IBE scheme as it is an efficient and a pioneering IBE scheme, aside from providing confidentiality for user messages in contrast to the previously chosen IBS schemes that provides authenticity. The implemented scheme `FullIdent` due to Boneh and Franklin [24] is chosen-cipher-text secure (CCA) based on the difficulty of the bilinear Diffie-Hellman problem. The algorithms of the scheme are shown in Algorithms 9–12.

Algorithm 1 RSA IBS setup.

procedure SHIBS.SETUP(1^k)
  $(N, e, d) \xleftarrow{\$} \mathcal{K}_{rsa}(1^k)$
  Select $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$
  Select $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_e^*$
  Return $mpk \leftarrow (N, e, H_1, H_2)$; $msk \leftarrow d$

Algorithm 2 RSA IBS user private key generation.

procedure SHIBS.USKGEN(`ID`, msk)
  $H_1(\texttt{ID})^d \bmod N \rightarrow s$
  Return $usk \leftarrow s$

Algorithm 3 RSA IBS signing.

procedure SHIBS.SIGN(`msg`, usk)
  $r \xleftarrow{\$} \mathbb{Z}_N^*$ and $r^e \bmod N \rightarrow R$
  $H_2(R \parallel \texttt{msg}) \bmod N \rightarrow x$
  $sr^x \bmod N \rightarrow y$
  Return $\sigma \leftarrow (R, y)$

Algorithm 4 RSA IBS verification.

procedure SHIBS.VERIFY(`msg`, σ, `ID`, mpk)
  $H_2(R \parallel \texttt{msg}) \bmod N \rightarrow x'$
  Return $y^e \stackrel{?}{=} H_1(\texttt{ID}) \cdot R^{x'} \bmod N$

Algorithm 5 DSA IBS setup.

procedure SCIBS.SETUP(1^k)
  Select primes p, q such that $q|p-1$
  $g \xleftarrow{\$} \mathbb{Z}_p^*; a \xleftarrow{\$} \mathbb{Z}_q^*$ and $g^a \bmod p \rightarrow A$
  Select $H : \mathbb{Z}_p^* \times \{0,1\}^* \rightarrow \mathbb{Z}_q^*$
  Return $mpk \leftarrow (g, A, H)$; $msk \leftarrow a$

Algorithm 6 DSA IBS user private key generation.

procedure SCIBS.USKGEN(ID, msk)
  $r \xleftarrow{\$} \mathbb{Z}_q^*$ and $g^r \bmod p \rightarrow R$
  $r + a \cdot H(R, \mathtt{ID}) \bmod q \rightarrow s$
  Return $usk \leftarrow R, s$

Algorithm 7 DSA IBS signing.

procedure SCIBS.SIGN(msg, usk)
  $t \xleftarrow{\$} \mathbb{Z}_q^*$ and $g^t \bmod p \rightarrow T$
  $t + s \cdot H(T, \mathtt{ID} \parallel \mathtt{msg}) \bmod q \rightarrow u$
  Return $\sigma \leftarrow (T, u, R)$

Algorithm 8 DSA IBS verification.

procedure SCIBS.VERIFY(msg, σ, ID, mpk)
  $H(R, \mathtt{ID}) \rightarrow c_1$ and $H(T, \mathtt{ID} \parallel \mathtt{msg}) \rightarrow c_2$
  Return $g^u \stackrel{?}{=} T(RA^{c_1})^{c_2} \bmod p$

Algorithm 9 Boneh–Franklin IBE setup.

procedure BFIBE.SETUP(1^k)
  Select $P \in \mathbb{G}_1; e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$
  $a \xleftarrow{\$} \mathbb{Z}_q^*$ and $aP \rightarrow A$
  Select $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$
  Select $H_3 : \{0,1\}^n \times \{0,1\}^n \rightarrow \mathbb{Z}_q^*$
  Select $H_4 : \{0,1\}^n \rightarrow \{0,1\}^n$
  Return $mpk \leftarrow (P, A, H_1, H_2)$; $msk \leftarrow a$

Algorithm 10 Boneh–Franklin IBE user private key generation.

procedure BFIBE.USKGEN(ID, msk)
  $H_1(\mathtt{ID}) \rightarrow C$
  $aC \rightarrow S$
  Return $usk \leftarrow S$

Algorithm 11 Boneh–Franklin IBE encryption.

procedure BFIBE.ENCRYPT(msg, ID, mpk)
  $H_1(\mathtt{ID}) \rightarrow C$ and $\sigma \xleftarrow{\$} \{0,1\}^n$
  $r \xleftarrow{\$} H_3(\sigma, \mathtt{msg})$ and $E \leftarrow e(C, A)$
  Return $c \leftarrow (rP, \sigma \oplus H_2(rE), \mathtt{msg} \oplus H_4(\sigma))$

Algorithm 12 Boneh–Franklin IBE decryption.

procedure BFIBE.DECRYPT(c, usk)
  Parse $c \rightarrow (U, V, W)$
  If $U \notin \mathbb{G}_1^*$ Then Return $\perp$
  $E \leftarrow e(S, U)$ and $\sigma \leftarrow V \oplus H_2(E)$
  $M \leftarrow W \oplus H_4(\sigma)$ and $r \leftarrow H_3(\sigma, M)$
  If $U \neq rP$ Then Return $\perp$
  Return $\mathtt{msg} \leftarrow M$

7. Implementation

The hybrid PKI back-end module is implemented as a Java socket server which listens exclusively on local network interfaces. The reason is to decouple from access control mechanisms and allow certificate authorities to implement and extend their own with their own access and certification policies.

7.1. Web Interface Example

To demonstrate the capabilities of the back-end module, we coded a minimalist front-end using PHP which allows interaction with the back-end, with some of the screen-shots shown in Figures 9–11. The advantage of modular decoupling is that certificate authorities can have their own business logic in their front-end and only issue service calls to the back-end based on their requirements. To interact with the module, an application from the local machine opens a network socket to `localhost` and sends in a service request payload, shown by a snippet of our code in Figure 12.

In fact, the user who wishes to only use a PKG without any front-end from the command line may also opt to use a popular network utility tool known as netcat to open a simple network socket (i.e., TCP socket) and issue out a service call. An example is shown in Figure 13.

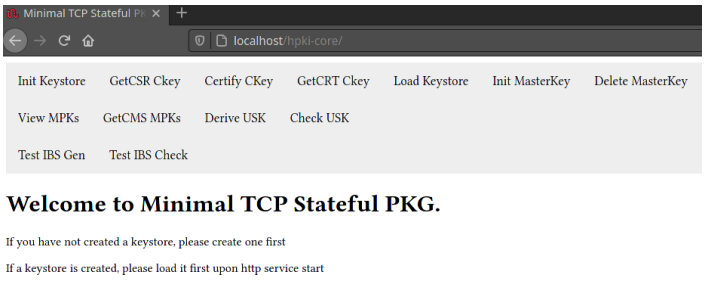

Figure 9. Minimalist front-end to the hybrid PKI module. This is a demo to illustrate the use-cases of the hybrid PKI back-end.

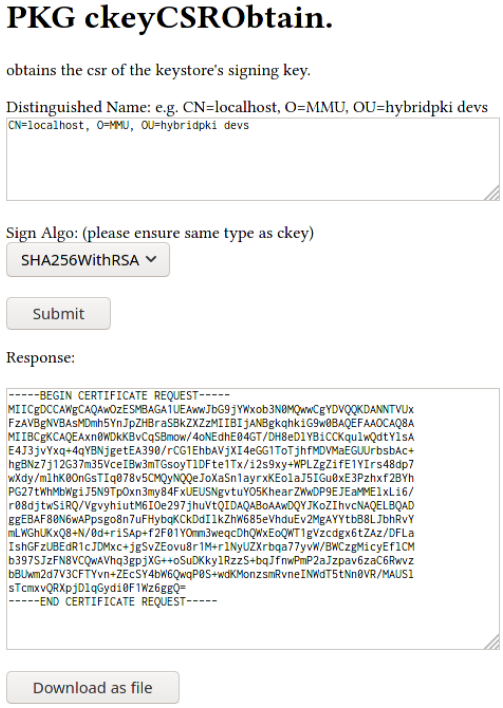

Figure 10. CSR generation. The generated CSR is sent to a CA for certification: it is used for inter-domain CMS generation by the PKG.

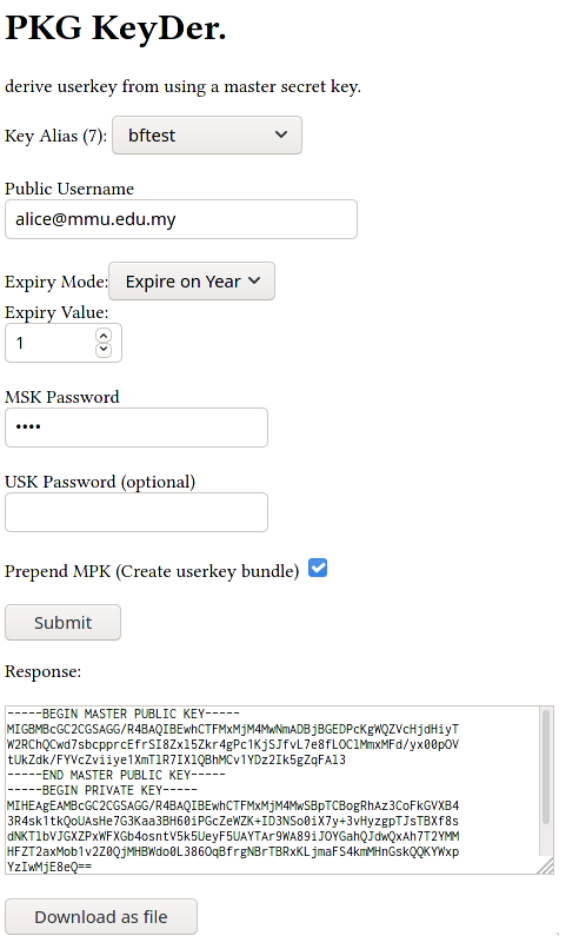

Figure 11. User key derivation. In this demo interface, an admin specifies which master key to be used to derive the user key for `alice@mmu.edu.my` along with an expiry mode (see Section 7.2). A user password may also be supplied for user key encryption using PKCS#8.

```
$ipaddr = '127.0.0.1';
$port = 6666;

$sock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
if ($sock === false) {
        die("failed: " . socket_strerror(socket_last_error()) . "\n");
}

//echo "Attempting to connect to '$ipaddr' on port '$port'...";
$res = socket_connect($sock, $ipaddr, $port);
if ($res === false) {
        die("failed: ($res) " . socket_strerror(socket_last_error($sock)) . "\n");
}

$in = '{';
$in .= '"dn":"'.trim($_POST["dn"],"\r\n").'",';
$in .= '"salgo":"'.trim($_POST["salgo"],"\r\n").'",';
$in .= '"service":"kmcsr"';
$in .= '}'."\r\n";

$out = '';

socket_write($sock, $in, strlen($in));
```

Figure 12. Example of issuing a service request in PHP to obtain a certificate signing request. A TCP socket connection is established to the back-end server and sends the service request in JSON format.

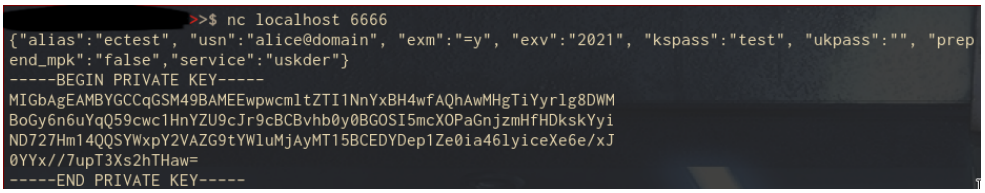

Figure 13. Using the back-end module with existing tools. The popular networking utility tool `netcat` is used to establish a connection to the back-end server and perform a `uskder` or user key derivation service request with the arguments passed in JSON format.

7.2. User Key Expiry

While key revocation is an ongoing problem for IBC, an idea has been put forwarded by Boneh and Franklin that users periodically renew their keys by appending the current time period to the user's key, forcing the user to request a new key every period (e.g., monthly, yearly). Bolyreva et al. proposed using fuzzy identity-based systems [25] to overcome this problem [26]; however, their solutions are more fitting for fuzzy IBC, requiring multiple user keys instead of conventional identity-based cryptography. Previously, in the hybrid PKI work by Tan et al., they employed a similar idea to Boneh and Franklin's method of appending a time period to the public identity; however, they treat the time period as an expiry date instead [13].

We adopted Tan et al.'s idea and extended it to include Boneh and Franklin's version. We introduce two additional arguments when generating a user's private key, namely an expiry mode and an expiry value. The following is an example of a public identity generated by our hybrid PKI system.

`alice@mmu.edu.my2023<y`

The public identity above is for "alice@mmu.edu.my" which is valid for all years earlier than 2023. A user public key must take the form of:

`<user id><expiry value><expiry mode>`

The following are a list of supported expiry modes:

1. Expire on year. `<y`
   - Upon creation, specify how many years (integer) the user private key will be valid for, starting from the current year.
   - The key is considered expired if the current year is greater than or equal to the year specified on the public identity.
   - Examples:
     (a) `alice@mmu.edu.my2023<y`. If the current year is 2020, this key is created with expiry value y = 3 and is only valid for year 2020, 2021 and 2022.
2. Valid for year. `=y`
   - Upon creation, specify the year (integer from 1000 to 9999) in which the key is valid for.
   - The key is considered expired if the current year is not equal to the year specified on the public identity.
   - Examples:
     (a) `bob@mmu.edu.my2020=y`. The key is created with expiry value y = 2020 and is only valid for the year 2020.
3. Expire on day. `<d`
   - Upon creation, specify how many days (integer) the user private key will be valid for, starting from the current day.
   - The key is considered expired if the current day is greater than or equal to the date (format YYYYMMDD) on the public identity.

- Examples:
  (a) `bob@mmu.edu.my20210201<d`. If the current date is 20201203, the key is created with expiry value d = 60 and is only valid until 1 February 2021.

Apparently, support for other expiry mode can also be easily incorporated by adding to the application logic (i.e., expiry on month `YYYYMM<m` or valid on month `YYYYMM=m`). The verifier or encryptor would then simply parse the mode and interpret regardless of whether a key has expired.

7.3. Browser Plugin and Email Plugin

To supplement the use of the hybrid PKI system, a browser plugin and email plugin prototype were also developed. The browser plugin features several use-cases such as page verification over an insecure HTTP channel and Web log in using IBS, while the email plugin has IBE features, where a user can send an email to the corresponding recipient and encrypting the email with the recipient's email address.

The plugins provide user functionality to sign/decrypt and to verify/encrypt. It is unable to generate user keys nor setup the MPK. These applications only aim to enhance the hybrid PKI ecosystem and are non essential to the use of the back-end system.

7.3.1. Firefox Browser Plugin

Figures 14 and 15 show a screen-shot of the Firefox plugin menu bar and options page, respectively. The plugin allows users to generate identity-based signature as well as to verify one as shown in Figures 16 and 17, respectively. In addition, pages that are signed with IBS schemes can also be verified without HTTPs, shown as a proof of concept in Figure 18.

The plugin stores user private keys in their encrypted form, prompting for user password each time the user private key is required (e.g., signing or decryption). This works in synergy with the back-end module which generates password-encrypted user private keys based on PKCS#5. The plugin uses the Javascript library for AMCL [21] and jsrsasign [27]. It can be ported to other browsers (e.g., Chrome) with only modifications to the user interface application programming interface (API) function calls which differs between one browser and another (the MDN Web Docs introduced the browser extension API that is uniform across platforms with only minute differences, allowing easy cross-platform extension building).

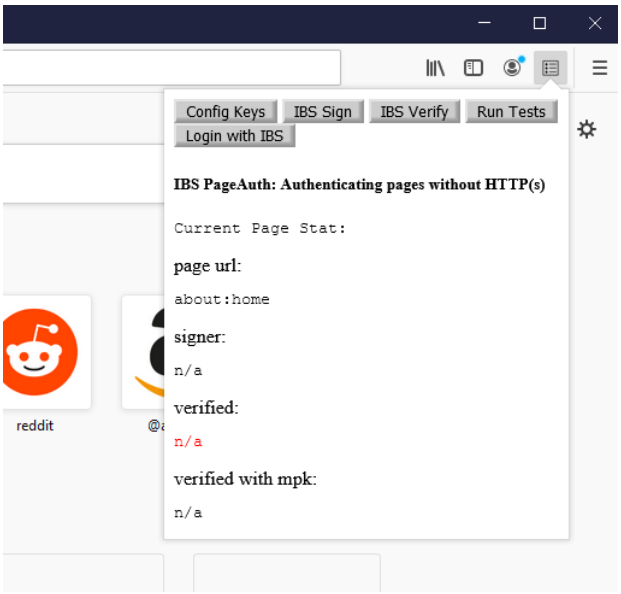

Figure 14. Browser plugin menu action. The user can perform IBS signing or verification through this plugin. If a website also uses the hybrid PKI suite, the user may also perform log in using the plugin with their user key.

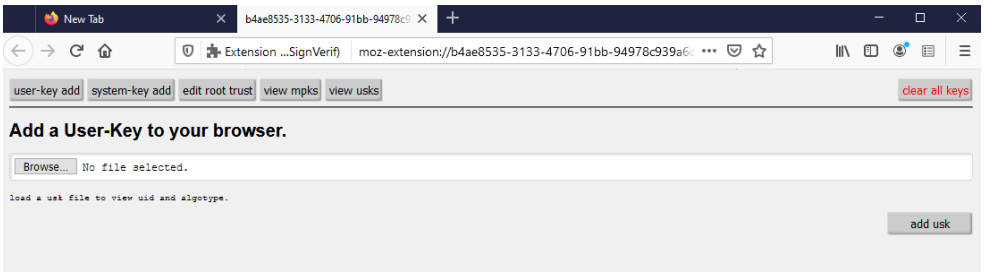

Figure 15. Browser plugin option page. The user can add user keys or MPKs to the plugin, which can then be used to perform IBS operations such as signing/verification and even log in onto websites that are using IBC as a means for entity authentication.

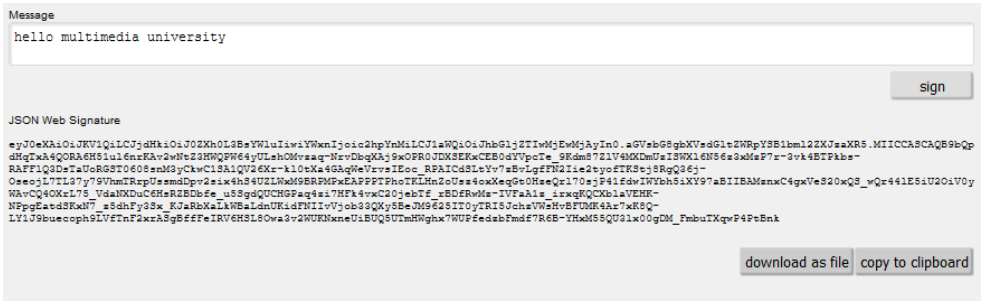

Figure 16. Browser plugin IBS generating an example. The user signs a plain-text message using the plugin. The signature is in the form of a JSON Web signature (JWS).

Figure 17. Browser plugin IBS verifying an example. The user verifies a JWS, which supports message recovery and obtains the signed message as well as the signer's public identity.

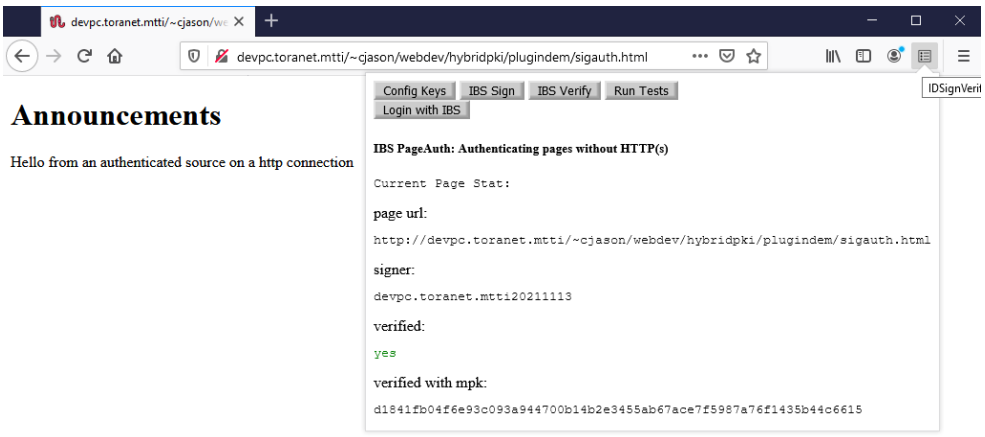

Figure 18. HTML page authentication over HTTP with IBS. If a website uses the hybrid-PKI suite, users may authenticate the HTML page using the browser plugin without using conventional PKI certificates.

7.3.2. Verify-Only Server

The hybrid PKI system also features a verify-only server (VOS) that can be deployed for servers running Websites that would like to have an IBS log in functionality. The VOS performs IBS verification and is interfaced the same way as the back-end module. This means that Web applications can be combined with the plugin to provide an IBS-based log in using only the user private key.

Figure 19 shows an example instance of logging in with IBS. The Web application samples a random challenge in which the user has to perform an IBS on. The plugin simplifies this process and allows the user to log in to company Web apps with their hybrid PKI system.

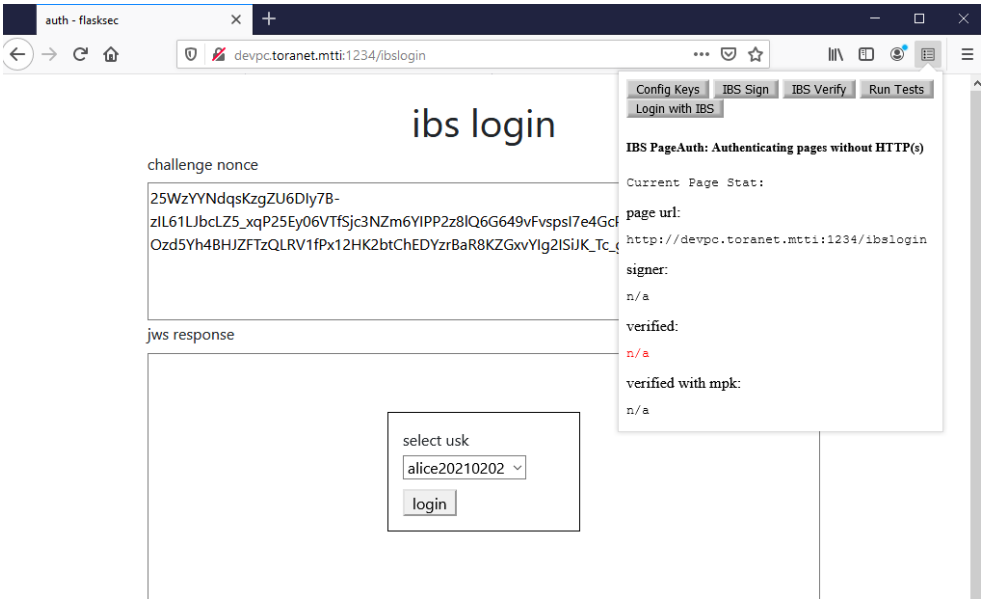

Figure 19. Example of website log in with browser plugin using IBS. An example website that uses the VOS can be logged into using the browser plugin. The server needs not store any passwords. The server only needs to store a list of identities (e.g., a white-list) that are allowed to log in as well as the MPK. When a user attempts to log in, the website can challenge with a random nonce for the user to perform IBS, which they then verify using the signer's identity and their white-list.

7.3.3. Thunderbird Mail IBE Plugin

We also show proof of concept for IBE on the popular email client Thunderbird. Figure 20 shows a user's email encrypted with IBE before they send it out. The plugin allows encryption and decryption linked to the recipient's email address.

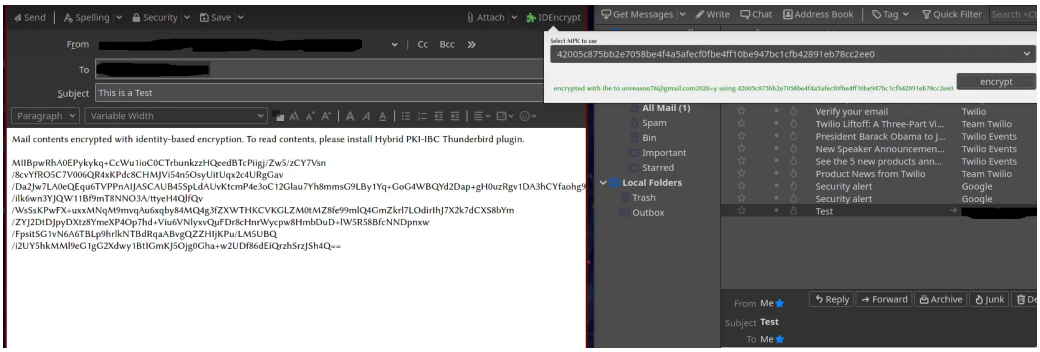

Figure 20. Email encryption with Thunderbird Hybrid PKI plugin. The user uses the plugin to encrypt the contents of the email using the email address of the recipient as the public key: the recipient can decrypt the email using their user key if their email address matches.

8. Comparison with Previous Similar Works

We compare our system with state of the art PKI–IBC hybrid frameworks which supports the use cases mentioned under Section 3 in terms of features. We consider the efficiency comparison separately due to the heterogeneity of the very few existing PKI–IBC hybrid frameworks that exists in the literature. It is difficult to reproduce the claimed efficiency of the various PKI–IBC hybrid frameworks, some of which do not have any implementation [10,12].

We compare our system with Price and Mitchell's work [10], Lee's Unified PKI (UPKI) [12], Tan et al.'s original enhanced PKI [13] and IBE/ABE services using X.509 presented by Reimair et al. [15]. We note several key differences and improvement over their work in Table 2. The new hybrid PKI module is a realization of past frameworks and an improvement over Tan et al.'s work in terms of features and conforms to more PKCS standards than its predecessor. This work also provides a means to allow external communication which follows the method as described by Lee's UPKI framework, utilizing a CMS containing the MPK of a domain A to bootstrap trust on users of domain B should they share the same ancestor in their trust hierarchy. Another big advantage provided by this work is the detachment process from existing PKI software, whereby companies may freely choose whichever PKI software they desire to easily integrate with the hybrid PKI module. The main difference between the solution proposed by Reimair et al. and ours is the storage of user keys: their idea requires the user key to be centrally stored on a security module (SM) which manages the key database. To sign/decrypt documents, users would perform the service request to the SM in order to obtain the signature or plain-text.

Table 2. Comparison with previous similar works.

Aspect	Price and Mitchell [10]	UPKI [12]	Tan et al. [13]	Reimair et al. [15]	This Work
Software/service dependency	Ind.	Ind.	EJBCA	CrySIL [14]	Ind.
IBS supported	N/A	N/A	Yes	No	Yes
IBE supported	N/A	N/A	No	Yes	Yes
ABE supported	No	No	No	Yes	No
User key format	N/A	N/A	ASCII	Unavailable to users, stored on CrySIL's security module	PKCS#8
User key encryption supported	N/A	N/A	No	Yes	Yes, PKCS#5
Master public key format	N/A	N/A	ASCII	ASN.1 DER encoded	ASN.1 DER encoded
Master key storage	N/A	N/A	plain-text	CrySIL's security module key-store	JKS with password encryption
External domain communication	PKI (Price and Mitchell proposed cross-certification by the domain's CA)	PKI	No	PKI	PKI
Interface	N/A	N/A	Command line	HTTP	Network sockets
Key revocation policy	Re-issue	Re-issue	Re-issue	User key not stored by user	Re-issue
Key expiry mode	No	No	Expire on date	No	Expire on date, year and valid for year
Key escrow problem	CL	CL	Present	Present	Present

N/A—not available/unspecified as it is only a framework without actual implementation; Ind.—independent; CL—solved using certificate-less crypto; JKS—Java key-store; ASN.1—Abstract Sytax One notation; DER—distinguished encoding rules.

8.1. Security Analysis

The PKI–IBC hybrid implementation should be treated the same way one would treat a PKI, because it is an extension of it. Following this logic, we consider the following common 3 attacks scenarios on a PKI for the implementations considered under Table 2. We found by ad hoc reasoning, that in 3 of the scenarios, our implementation is comparably secure as the implementation presented by Reimair et al. [15].

8.1.1. Scenario 1: Compromised PKG Host

In the scenario depicted in Figure 21, we assume an attacker has obtained root access onto the host machine running the PKI–IBC software. For the implementation from Tan et al. [13], this scenario causes the entire IBC to be compromised because the master key is in plain-text. For the implementation presented by Reimair et al. [15], the attacker would gain access to the master keys loaded into memory when a user request for the signing/decryption because the user keys are generated on-demand. For the same reason, our implementation would cause the master key to be compromised if a user request for key generation. If no such requests were to happen while the attacker has root access, the security of the master keys for both would then rely on the underlying key protection mechanism.

Admittedly, this is the most powerful practical attack that an attacker can launch, analogous to an attacker compromising a certificate's authority.

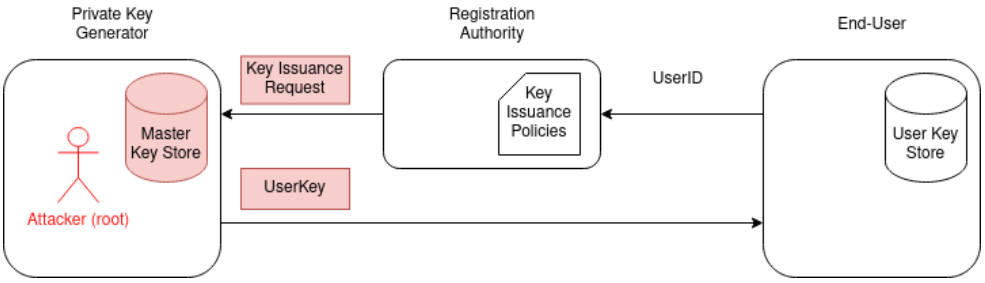

Figure 21. A scenario where the PKG is compromised.

8.1.2. Scenario 2: Compromised Key-Issuance Channel

In the scenario depicted in Figure 22, we assume that an attacker has obtained root access onto the servers of the registration authority (RA), who is responsible for validating user information and subsequently request key issuance. This is exactly the security model defined for identity-based cryptographic schemes in the literature; an attacker can corrupt any number of user identities to obtain the corresponding user keys, but still must be unable to generate new valid user keys of identities which were not previously corrupted to perform IBC operations correctly (e.g., decryption/signing) [24,28]. It is difficult to qualitatively analyze the security of the hybrid PKI–IBC system in this scenario because it depends on the cryptographic scheme in question. In conventional PKI, this is analogous to an attacker that has compromised a RA and would be able to request a certificate on any public key of their liking.

This scenario is not applicable for the works of Reimar et al. because user keys are never issued. While this is advantageous for scenarios as such, the maintenance cost of having an always present central security module to do decryption/signing for the user is very costly.

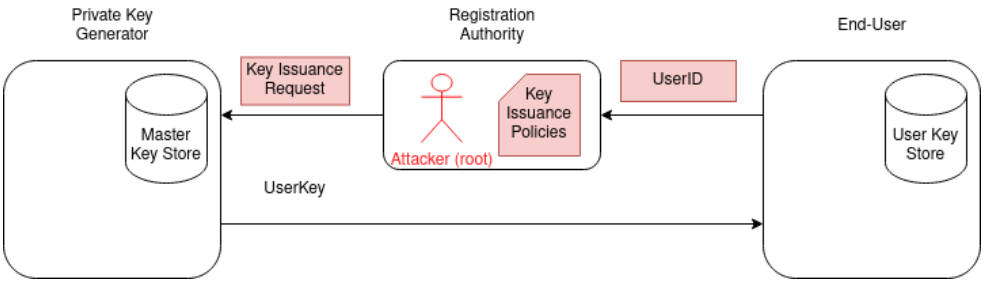

Figure 22. A scenario where the registration authority is compromised.

8.1.3. Scenario 3: Compromised User Device

In the scenario depicted in Figure 23, we assume an attacker has obtained root access to the user's device. We observe that for our implementation and that of Reimair et al.: if the user does not attempt to perform IBC operations, their keys can still be secure; if the user performs decryption/signing, the attacker would gain access to the password used to encrypt the user key, which allows it to obtain the user key. Likewise, for Reimair et al., the user would supply identifying information to the central CrySIL module, which is leaked to the attacker, thereby allowing it to impersonate the user and illegally perform decryption/signing on behalf of the user.

In conventional PKI, this scenario likely leads to the user key being exposed if it is not protected; if the user has knowledge of this occurring, they would report a key compromise to the CA and request for revocation.

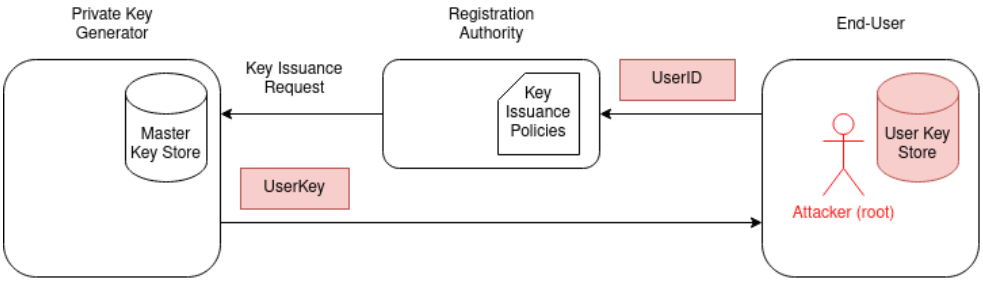

Figure 23. A scenario where the user device is compromised.

8.2. Efficiency Analysis

We would like to reiterate that we do not claim to introduce a more efficient PKI–IBC hybrid framework, but rather a much more inter-operable and feature rich implementation. However, it is still in our best interest to present the efficiency results of our scheme which show its practicality. For the security levels used in our analysis, we select our parameters for 80-bit, 112-bit, 128-bit and 192-bit security per NIST recommendations [29]. For elliptic curve-based schemes such as ECDSA IBS, the curves secp160r1, secp224r1, secp256r1 and secp384r1 were chosen to correspond to the security levels.

We briefly discuss the efficiency of the implementation in comparison with existing implementations (i.e., Tan et al. [13]). As both software were developed on the same programming language and use the same cryptographic library, we expect the difference in run-time and memory usage to be minimum. We argue that our implementation is much more lightweight in terms of memory footprint as it is decoupled with EJBCA's core library (While this may not be an issue if an organization is using EJBCA as their PKI software, an organization that does not use EJBCA would need to install and run EJBCA regardless of whether they are using it).

8.3. Private Key Generator

Speed is not a main concern as the role of a PKG is to issue keys post identity validation. As both implementations run on the Java virtual machine and use the same cryptographic

library (i.e., Bouncy Castle), the difference in run-times between our work and Tan et al.'s work is negligible. Furthermore, even if there is improvement over theirs, it is insignificant as user private keys do not need to be issued in real-time (or for the matter of security, should not be issued in real time upon request).

Table 3 shows the generated key sizes according to the various security levels. We use the same public ID (e.g., alice@mmu.edu.my) and expiry mode =y to generate all the keys. We note that IBS based on DSA primitives require larger key sizes because parameters such as the discrete log parameters prime p and order of group q need to be stored. However, we can see later, in Section 8.4, that the IBS with DSA primitives has the shortest run-time. The shortest keys are those of elliptic curve keys due to the fact that the parameter sizes are shorter and the fact that only the name of the curves are stored (e.g., secp224r1) instead of storing the actual curve parameters.

While the implementation by Reimair does not need the user to store the keys, the maintenance of their CrySIL security module (SM) bears a huge cost given that it needs to be accessible to the user at all times for decryption/signing to occur. The bandwidth required for their IBE/ABE solution using CrySIL would also be large given that the user would need to communicate with the SM when it needs to decrypt a message. In contrast, our solution works even if the PKG were to be removed to form a closed system, in addition to being more bandwidth efficient as the user need not send a message in for decryption/signing.

Table 3. Generated key sizes (bytes). Results split into encrypted/non-encrypted sizes.

Security Level	Scheme	UPK Size (Bytes)	Encrypted UPK Size (Bytes)	MPK Size (Bytes)
80-bit (Legacy)	RSA IBS (1024)	509	659	286
	DSA IBS (1024)	895	1037	664
	ECDSA IBS (secp160r1)	221	363	140
112-bit	RSA IBS (2048)	859	1005	465
	DSA IBS (2048)	1627	1764	1204
	ECDSA IBS (secp224r1)	258	395	152
	Boneh–Franklin IBE (BLS12383)	339	485	245
128-bit	RSA IBS (3072)	1208	1350	639
	DSA IBS (3072)	2317	2455	1724
	ECDSA IBS (secp256r1)	274	416	156
	Boneh–Franklin IBE (BLS12461)	379	525	274
192-bit	RSA IBS (7680)	2768	2910	1419
	DSA IBS (7680)	5405	5543	4048
	ECDSA IBS (secp384r1)	339	485	176
	Boneh–Franklin IBE (BLS24479)	558	700	448

8.4. Plugins

Signing and verification as well as encryption will be repeatedly used on the plugins by users of the system, thus we compare the performance run-time of the plugins in terms of signing and verification. Our test machine is an Intel(R) Core(TM) i7-8750H CPU @ 2.2 GHz running on a 64-bit Linux OS with 6 cores and 12 threads. A 64-bit Firefox browser on version 84.0.1 was used for testing. We recorded the average run-times of the schemes from 100 runs with increasing message sizes. Figures 24 and 25 show the run-time for RSA-based IBS in terms of signing and verification, respectively. We see that although message sizes increase, the run-time is mainly constant as the run-time footprint of the hashing of messages is negligible compared to the much longer exponentiation run-times. This trend is consistent throughout the IBS schemes built with other primitives, as shown in Figures 26–29 for DSA and ECDSA IBS.

Note that the performance is somewhat sluggish as Javascript is not known for itd speed—especially in terms of a plugin. However, we stress that these run-times are practical and do not really hinder the workflow of a human using it for signing/verification purposes, which is the main goal of our work. The average run-time of all IBS schemes implemented in the plugin, as shown in Figures 30 and 31, indicates that the hybrid PKI

system is practical in terms of run-times. Notice that the 192-bit results in Figure 31 have an ECDSA with a shorter run-time than DSA and RSA. We believe that this is due to the disproportionate increase in parameter sizes for ECDSA versus DSA and RSA: for 128-bit to 192-bit security, ECDSA increases its parameter sizes from 256 bits to 384 bits—a 1.5 times increase—whereas DSA and RSA had roughly increased by 2.5 times: from 3072 bits to 7680 bits.

In the case of identity-based encryption, only the curves BLS12383 [30], BLS12461 [31] and BLS24479 [32] are currently used to support 112-bit, 128-bit and 192-bit security, respectively. We run the same run-time experiment on 64-bit Mozilla Thunderbird on version 78.6.0. We perform encryption/decryption with a random plain-text increasing in size and show our results in Figures 32 and 33. The results show the same roughly consistent run-times which are mostly dominated by a pairing operation during encryption and decryption that are computationally intensive. A higher encryption run-time is also observed as it requires an additional point multiplication operation in $\mathbb{G}_2$. The email plugin runs on Javascript similar to the browser plugin.

We aim to support more standardized pairing-friendly curves as well as different variants of the Boneh–Franklin cryptosystem when public interest increases in our prototype. However, as pointed out by Michael Scott, standardization efforts have stalled due to newer security insights into the true security of curves [33].

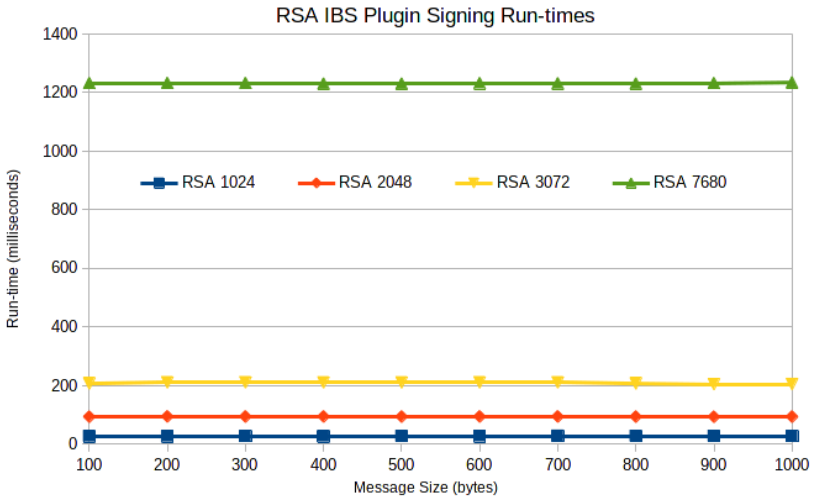

Figure 24. RSA IBS signing run-time vs. message size.

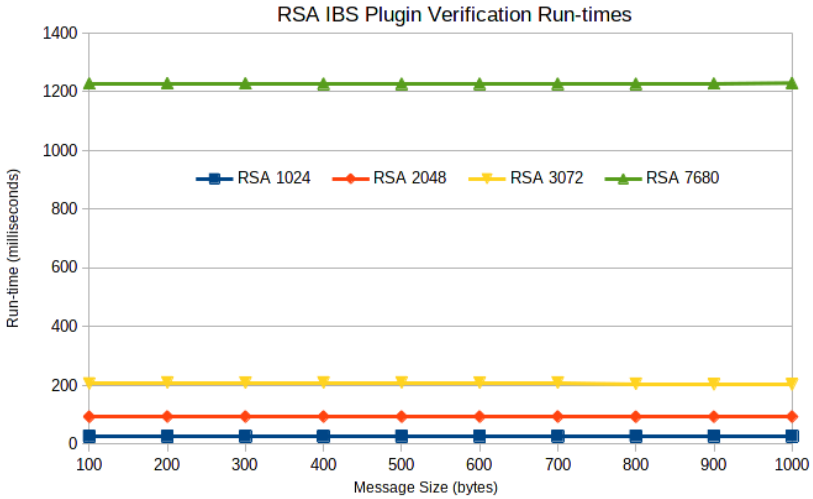

Figure 25. RSA IBS verification run-time vs. message size.

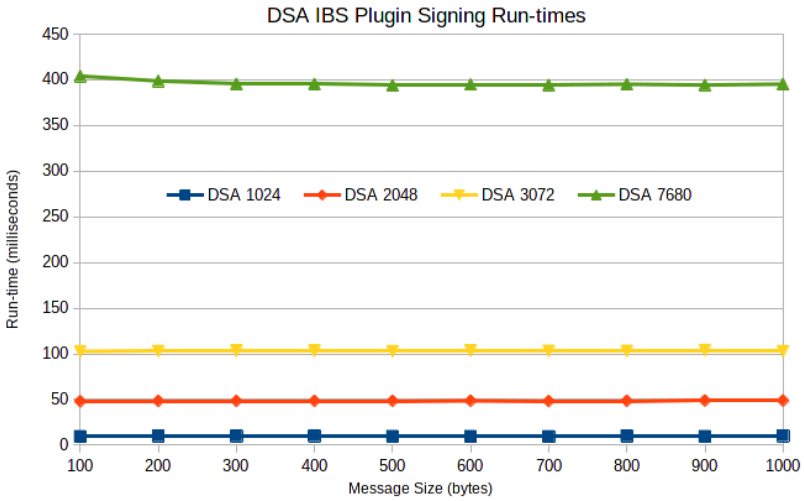

Figure 26. DSA IBS signing run-time vs. message size.

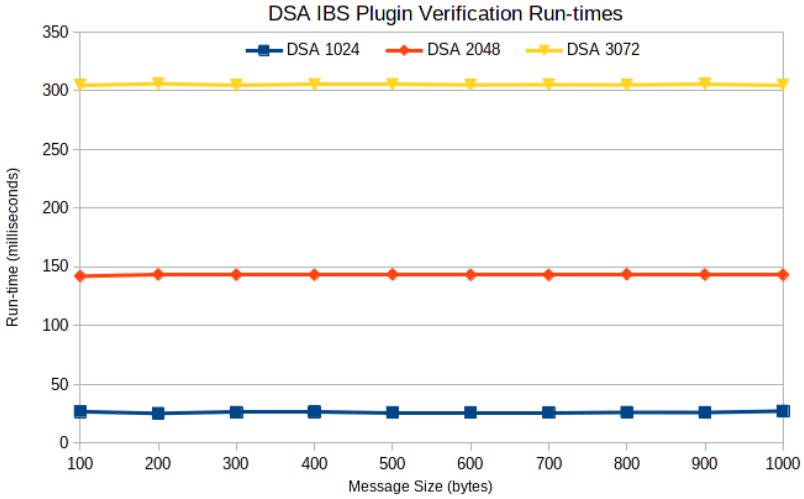

Figure 27. DSA IBS verification run-time vs. message size.

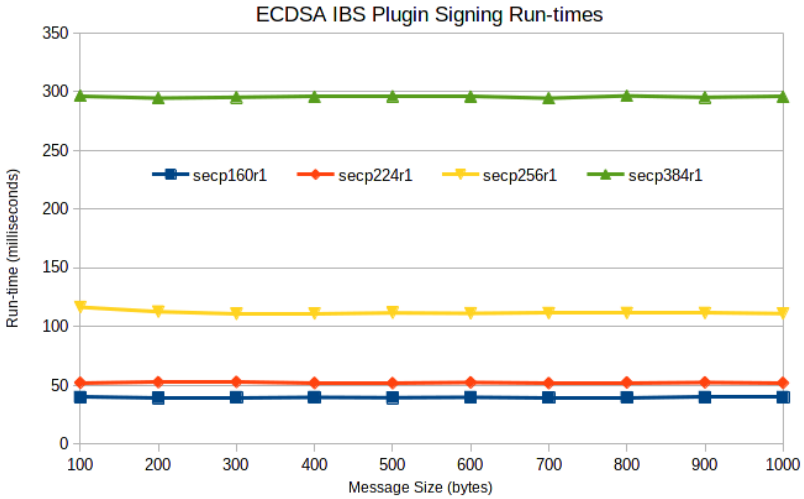

Figure 28. ECDSA IBS signing run-time vs. message size.

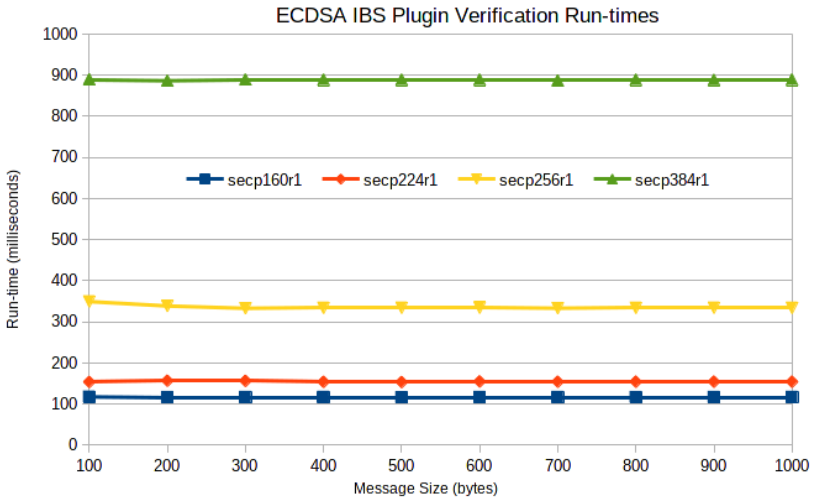

Figure 29. ECDSA IBS verification run-time vs. message size.

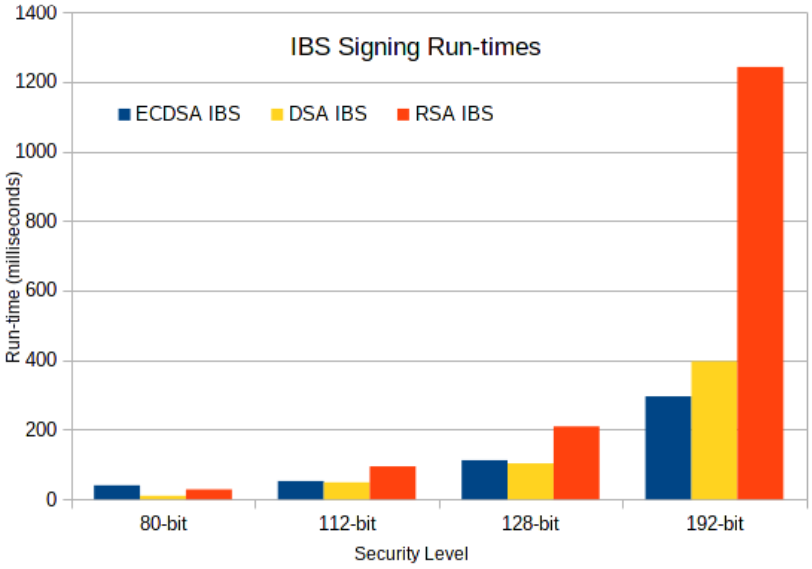

Figure 30. Average IBS signing run-time vs. security level.

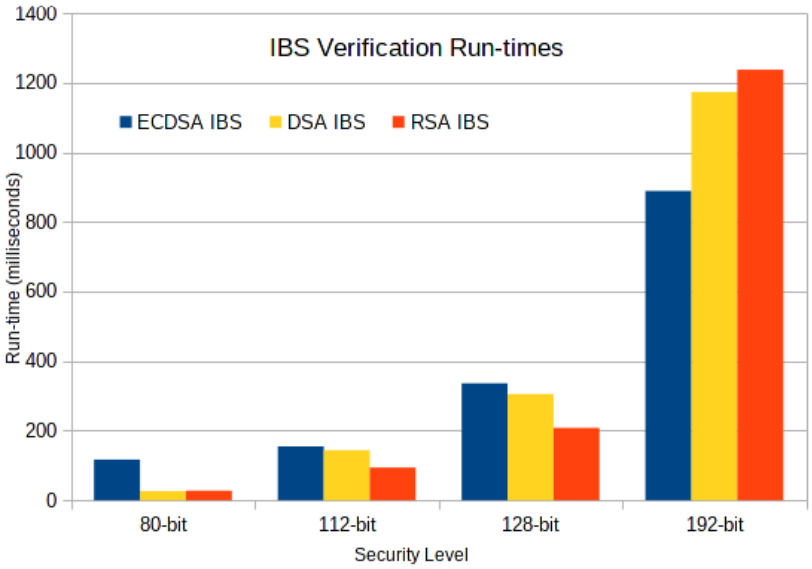

Figure 31. Average IBS verification run-time vs. security level.

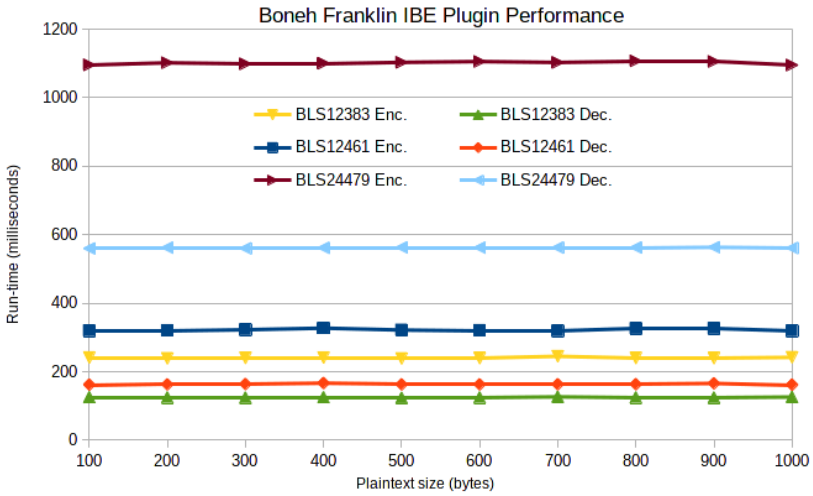

Figure 32. Email plugin run-time vs. plain-text size.

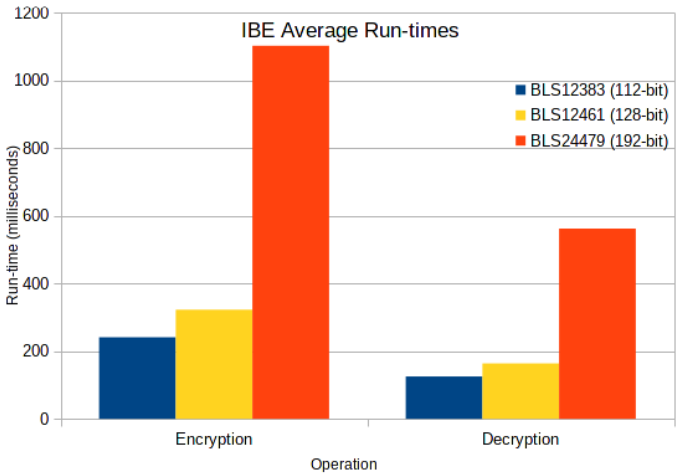

Figure 33. Email plugin average run-times.

9. Conclusions

In this work, we presented a hybrid PKI framework and showed an implementation of it along with utilities such as browser and email plugins. The new hybrid PKI system can be easily interfaced with existing PKI software through common methods to secure asymmetric cryptographic protocols. In addition, it can also provide identity-based cryptographic capabilities to the security environment. We showed a comparison with the existing literature and found it to be superior in terms of features provided. In addition, the hybrid PKI system easily allows secure intra-domain communication as well as external domain communication, where the authenticity of a PKG has yet to be established. In essence, this enables the framework to be easily deployed on existing corporations with or without their own trust infrastructures, replacing the majority of their user certificates and reducing their operating costs.

Author Contributions: Conceptualization, S.-H.H.; formal analysis, J.C., J.-J.C., S.-Y.T. and W.-C.Y.; funding acquisition, S.-H.H.; methodology, S.-H.H., J.-J.C., S.-Y.T. and W.-C.Y.; project administration, S.-H.H.; resources, S.-H.H.; software, J.C.; supervision, S.-H.H., J.-J.C., S.-Y.T. and W.-C.Y.; validation, J.C., J.-J.C., S.-Y.T. and W.-C.Y.; visualization, J.C.; writing—original draft, J.C.; writing—review and editing, S.-H.H., J.-J.C., S.-Y.T. and W.-C.Y. All authors have read and agreed to the draft version of the manuscript.

Funding: The authors acknowledge the Prototype Research Grant Scheme by the Ministry of Higher Education of Malaysia (PRGS/2/2019/ICT04/MMU/01/1) in providing financial support for this work.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: The authors would like to express gratitude to the anonymous reviewers for their helpful comments on a preliminary version of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Adams, C.; Lloyd, S. *Understanding PKI: Concepts, Standards, and Deployment Considerations*, 2nd ed.; Addison-Wesley Longman Publishing Co., Inc.: Boston MA, USA, 2002.
2. Stoianov, N.; Urueña, M.; Niemiec, M.; Machnik, P.; Maestro, G. Integrated security infrastructures for law enforcement agencies. *Multimed. Tools Appl.* **2013**, *74*, 4453–4468. [CrossRef]
3. Diffie, W.; Hellman, M. New Directions in Cryptography. *IEEE Trans. Inf. Theor.* **2006**, *22*, 644–654. [CrossRef]
4. Salek, A. Private PKI: Deployment, Automation and Management. Available online: https://www.securitymagazine.com/articles/93101-private-pki-deployment-automation-and-management (accessed on 2 May 2021).
5. Alrawais, A.; Alhothaily, A.; Cheng, X.; Hu, C.; Yu, J. SecureGuard: A Certificate Validation System in Public Key Infrastructure. *IEEE Trans. Veh. Technol.* **2018**, *67*, 5399–5408. [CrossRef]
6. Shamir, A. Identity-based Cryptosystems and Signature Schemes. In *Proceedings of the CRYPTO 84 on Advances in Cryptology*; Springer: New York, NY, USA, 1985; pp. 47–53.
7. Qing-hai, B. Comparative research on two kinds of certification systems of the public key infrastructure (PKI) and the identity based encryption (IBE). In Proceedings of the Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC), New Taipei, Taiwan, 23–27 July 2012; pp. 147–150.
8. Wang, C.; Jan, S.T.; Hu, H.; Bossart, D.; Wang, G. The Next Domino to Fall: Empirical Analysis of User Passwords across Online Services. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 196–203.
9. Trzupek, B. PKI is key to securing a post-Covid remote workforce. *Comput. Fraud Secur.* **2020**, *2020*, 11–13. [CrossRef]
10. Price, G.; Mitchell, C. Interoperation Between a Conventional PKI and an ID-Based Infrastructure. In *Public Key Infrastructure, Second European PKI Workshop: Research and Applications*; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3545, pp. 73–85.
11. Chen, Q.; Yu, S.; Li, Z. A Cross-Authentication Model for Heterogeneous Domains in Active Networks. In Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC 2007), Dalian, China, 18–21 September 2007; pp. 140–143.
12. Lee, B. *Unified Public Key Infrastructure Supporting Both Certificate-Based and ID-Based Cryptography*; IEEE Publications: Manhattan NY, USA, 2010; pp. 54–61.
13. Tan, S.; Yau, W.C.; Lim, B.H. An implementation of enhanced public key infrastructure. *Multimed. Tools Appl.* **2014**, *74*, 6481–6495. [CrossRef]
14. Reimair, F.; Teufl, P.; Zefferer, T. CrySIL: Bringing Crypto to the Modern User. In *Web Information Systems and Technologies*; Monfort, V., Krempels, K.H., Majchrzak, T.A., Turk, Ž., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 70–90.
15. Reimair, F.; Feichtner, J.; Teufl, P. Attribute-Based Encryption Goes X.509. In Proceedings of the 2015 IEEE 12th International Conference on e-Business Engineering, Beijing, China, 23–25 October 2015; pp. 393–400.
16. Rajendran, B. Evolution of PKI ecosystem. In Proceedings of the 2017 International Conference on Public Key Infrastructure and Its Applications (PKIA), Bangalore, India, 14–15 November 2017; pp. 9–10.
17. Orman, H. Blockchain: The Emperors New PKI? *IEEE Internet Comput.* **2018**, *22*, 23–28. [CrossRef]
18. Chen, Y.; Dong, G.; Bai, J.; Hao, Y.; Li, F.; Peng, H. Trust Enhancement Scheme for Cross Domain Authentication of PKI System. In Proceedings of the 2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Guilin, China, 17–19 October 2019; pp. 103–110.
19. Chiu, W.Y.; Meng, W.; Jensen, C.D. ChainPKI—Towards Ethash-based Decentralized PKI with Privacy Enhancement. In Proceedings of the 2021 IEEE Conference on Dependable and Secure Computing (DSC), Aizuwakamatsu, Japan, 30 January–2 February 2021; pp. 1–8.
20. The Legion of the Bouncy Castle Inc. Bouncy Castle. Available online: https://www.bouncycastle.org/ (accessed on 10 August 2020).
21. Whitney, J.M. Milagro Introduction. Available online: https://milagro.apache.org/docs/milagro-intro/ (accessed on 13 September 2020).
22. Kiltz, E.; Neven, G.; Joye, M. Identity-Based Signatures. *J. Cryptol. JOC* **2008**, *2*, 75.

23. Galindo, D.; Garcia, F.D. A Schnorr-Like Lightweight Identity-Based Signature Scheme. In *Progress in Cryptology—AFRICACRYPT 2009*; Preneel, B., Ed.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 135–148.
24. Boneh, D.; Franklin, M. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology—CRYPTO 2001*; Kilian, J., Ed.; Springer: Berlin/Heidelberg, Germany, 2001; pp. 213–229.
25. Sahai, A.; Waters, B. Fuzzy Identity Based Encryption. Cryptology ePrint Archive, Report 2004/086. 2004. Available online: https://eprint.iacr.org/2004/086 (accessed on 4 April 2020).
26. Boldyreva, A.; Goyal, V.; Kumar, V. Identity-based encryption with efficient revocation. In Proceedings of the ACM Conference on Computer and Communications Security, CCS, Alexandria VA, USA, 27–31 October 2008; pp. 417–426.
27. Urushima, K. Jsrsasign. 2012. Available online: https://kjur.github.io/jsrsasign/ (accessed on 12 August 2020).
28. Bellare, M.; Namprempre, C.; Neven, G. Security Proofs for Identity-Based Identification and Signature Schemes. In *Advances in Cryptology—EUROCRYPT 2004*; Cachin, C., Camenisch, J.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 268–286.
29. Elaine, B. *Recommendation for Key Management, Part 1: General*, 5th ed.; U.S. Department of Commerce, National Institute of Standards and Technology: Gaithersburg MD, USA, 2020.
30. Barreto, P.S.L.M.; Lynn, B.; Scott, M. Constructing Elliptic Curves with Prescribed Embedding Degrees. Cryptology ePrint Archive, Report 2002/088. 2002. Available online: https://eprint.iacr.org/2002/088 (accessed on 8 August 2020).
31. Barbulescu, R.; Duquesne, S. Updating Key Size Estimations for Pairings. Cryptology ePrint Archive, Report 2017/334. 2017. Available online: https://eprint.iacr.org/2017/334 (accessed on 3 July 2020).
32. Aranha, D.F.; Fuentes-Castañeda, L.; Knapp, E.; Menezes, A.; Rodríguez-Henríquez, F. Implementing Pairings at the 192-bit Security Level. Cryptology ePrint Archive, Report 2012/232. 2012. Available online: https://eprint.iacr.org/2012/232 (accessed on 7 August 2020).
33. Scott, M. Curves.txt. 2019. Available online: https://github.com/miracl/core/blob/master/curves.txt (accessed on 13 September 2020).