Identity Based Encryption

Giulia Mauri

Politecnico di Milano

email: *giulia.mauri@polimi.it*
website: *http://home.deib.polimi.it/gmauri*

June 10, 2015

Overview

References:

[1] Dan Boneh and Matthew Franklin, "Identity-Based Encryption from the Weil Pairing", *Advances in Cryptology*, 2001
[2] "The Boneh-Franklin BF Cryptosystem",*RFC 5091*, 2007
[3] Matthew Green and Giuseppe Ateniese, "Identity-Based Proxy Re-Encryption", *Proceedings of the 5th International Conference on Applied Cryptography and Network Security*, 2007

Introduction

The main motivation for an identity-based encryption was to simplify the certificate management in e-mail systems.

Figure : Public Key Infrastructure

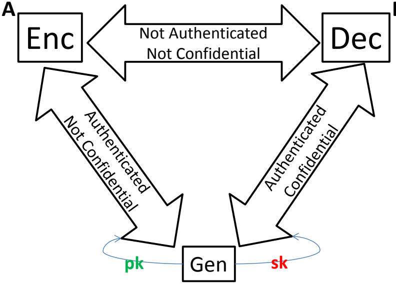
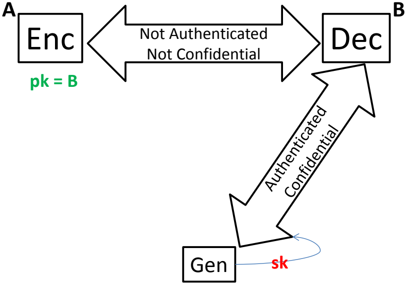


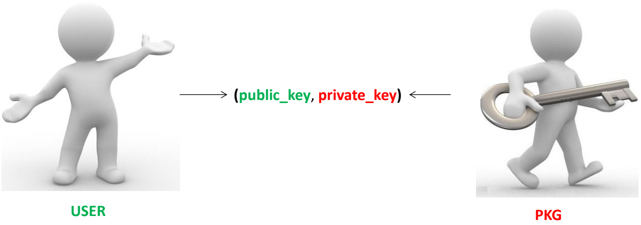
Figure : Identity Based Encryption



Since Shamir posed the problem in 1984, there have been several proposals for IBE schemes. Here, we consider the scheme proposed by Dan Boneh and Matthew Franklin in [1].

Identity Based Encryption

An identity-based encryption is a public key scheme, where the public key is derived directly from the user identity, while a trusted third party, called the Private Key Generator (PKG), generates the corresponding private key.



When Alice sends an email to Bob, she can simply encrypt the message using the public key string. There is no need for Alice to obtain Bob's public key certificate. When Bob receives the encrypted email he contacts a third party, the PKG. Bob authenticates himself to the PKG and obtains his private key from the PKG. Bob can then read his email.

Bilinear Map - ref[3]

We say a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ is a bilinear map if:

1. $\mathbb{G}_1, \mathbb{G}_T$ are groups of the same prime order q.
2. For all $a, b \in \mathbb{Z}_q^*, g \in \mathbb{G}_1, e(g^a, g^b) = e(g, g)^{ab}$.
3. The map is non-degenerate (i.e., if $\mathbb{G}_1 = \langle g \rangle$, then $\mathbb{G}_T = \langle e(g, g) \rangle$).
4. e is efficiently computable.

Note: $\mathbb{G}_1$ and $\mathbb{G}_T$ are elliptic curves. We will see more details in the following.

Decisional Bilinear Diffie Hellman Assumption - ref[3]

Let $(\mathbb{G}_1, \mathbb{G}_T)$ be a pair of bilinear groups with an efficiently computable pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, and let g be a random generator of $\mathbb{G}_1$.
The DBDH problem is to decide, given a tuple of values (g, g^a, g^b, g^c, T), where a, b, c are randomly selected from $\mathbb{Z}_q^*$), whether $T = e(g,g)^{abc}$ or if T is a random element of $\mathbb{G}_T$.

Challenger, $\mathcal{C}h$ | Adversary, $\mathcal{A}$

$\boxed{a, b, c \leftarrow \mathbb{Z}_q^*}$

$\xrightarrow{g, g^a, g^b, g^c}$

$\boxed{k \leftarrow \{0,1\}}$

IF $k = 0 : T \leftarrow \mathbb{G}_T$

IF $k = 1 : T = e(g,g)^{abc}$ $\xrightarrow{T}$

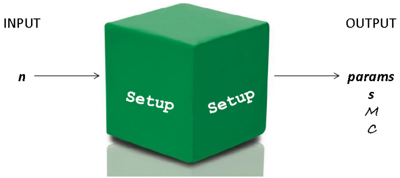
$k' = Exp(k)$ $\longrightarrow$

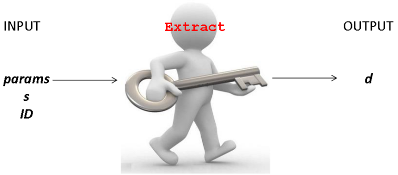
$$|Pr(Exp(0) = 1) - Pr(Exp(1) = 1)| \leq \epsilon$$

Definitions - ref[1]

An identity-based encryption scheme E is specified by four randomized algorithms:

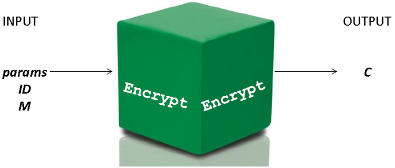
1. `Setup`: takes a security parameter n and returns $params(p, q, P, P_{pub}, hashfcn)$ and master-key s. The system parameters include a description of a finite message space $\mathcal{M}$, and a description of a finite ciphertext space $\mathcal{C}$.
2. `Extract`: takes as input params, master-key, and an arbitrary $ID \in \{0,1\}^*$, and returns a private key d. Here ID is an arbitrary string that will be used as a public key and d is the corresponding private decryption key. The `Extract` algorithm extracts a private key from the given public key.

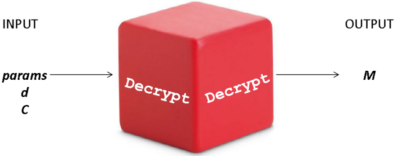




Definitions

3. $\texttt{Encrypt}$: takes as input params, ID, and $M \in \mathcal{M}$. It returns a ciphertext $C \in \mathcal{C}$.
4. $\texttt{Decrypt}$: takes as input params, $C \in \mathcal{C}$, and a private key d. It returns $M \in \mathcal{M}$.



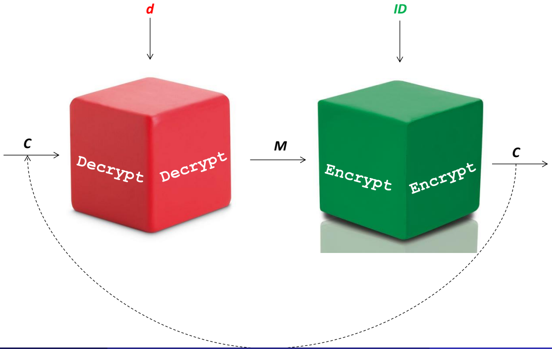


Definitions

The algorithm must satisfy the standard consistency constraint, namely when d is the private key generated by the algorithm $\texttt{Extract}$ when it is given ID as the public key, then:

$$\forall M \in \mathcal{M} : \texttt{Decrypt}(params, C, d) = M$$

with C=$\texttt{Encrypt}$(params,ID,M).



Implementing IBE in Sage

In the following, we describe a particular implementation of the Identity Based Encryption scheme in SAGE.
We consider that:

- $\mathbb{G}_1$ is a subgroup of the elliptic curve E/F_p defined by the equation $y^2 = x^3 + 1 \pmod p$, such that the point P defined over the curve E is of order q.
- $\mathbb{G}_T$ is a subgroup of the elliptic curve $E2/F_{p^2}$ defined by the equation $y^2 = x^3 + 1 \pmod{p^2}$.

IBE - Setup I - ref[2]

BFsetup establishes a master secret and public parameters.

1. Determine the subordinate security parameters n_p and n_q as follows. If $n = 2048$, then let $n_p = 1024$, $n_q = 224$, *hashfcn*=SHA-224.

Solution

```
sage: n = 2048, n_p = 1024, n_q = 224
sage: hashfcn = hashlib.sha224() #Note: import hashlib
```

IBE - Setup I - ref[2]

BFsetup establishes a master secret and public parameters.

1. Determine the subordinate security parameters n_p and n_q as follows. If $n = 2048$, then let $n_p = 1024$, $n_q = 224$, $hashfcn$=SHA-224.

Solution

```
sage: n = 2048, n_p = 1024, n_q = 224
sage: hashfcn = hashlib.sha224() #Note: import hashlib
```

2. Construct the elliptic curve and its subgroup of interest, as follows:

(a) Select an arbitrary n_q-bit Solinas prime q.

(b) Select a random integer r such that $p = 12rq - 1$ is an n_p-bit prime.

Solution

```
sage: q = 2^224 - 2^96 + 1 #n_q bit Solinas prime
sage: l = n_p - n_q - 4 + 1 #r length
sage: while not p.is_prime(): #p=int(4)
.:      r = random.randint(2^(l-1),2^l); #Note:import random
.:      p = 12 * r * q - 1
```

IBE - Setup II - ref[2]

3.Select a point P of order q in $E(F_p)$, as follows:
(a) Select a random point P' of coordinates (x', y') on the curve
$E/F_p : y^2 = x^3 + 1 \pmod{p}$
(b) Let $P = [12 * r]P'$
(c) If P $= 0$, then start over in step 2a

Solution

```
sage: E = EllipticCurve (GF(p), [0,1])
sage: P_1 = E.random_element()
sage: r_1 = 12 * r
sage: P = r_1 * P_1 #Point of order q
sage: while P == 0: #if P = 0 repeat
.:        P_1 = E.random_element()
.:        r = random.randint(2^(l-1),2^l)
.:        P = 12 * r * P_1
```

IBE - `Setup III` - ref[2]

4. Determine the master secret and the public parameters as follows:
(a) Select a random integer s in the range 2 to $q-1$
(b) Let $P_{pub} = [s]P$

Solution

```
sage: s = random.randint(2,q-1)
sage: msk = s #Master Secret Key
sage: P_pub = s * P

sage: params = (E,p,q,P,P_pub,hashfcn) #Public params
```

$(E, p, q, P, P_{pub}, hashfcn)$ are the public parameters.
The integer s is the master key.

IBE - `Extract` - ref[2]

BFextract derives the private key corresponding to an identity string.
1. Let $Q_{id} = HashToPoint(E, p, q, id, hashfcn)$ using Algorithm 4.4.1 in RFC 5091.
2. Let $S_{id} = [s]Q_{id}$

Solution

```
sage: def KeyGen(params,msk,id):
...:      Q_id = HTP(E,p,q,id,hashfcn)
...:      sk_id = msk * Q_id
...:      sec = (Q_id,sk_id)
...:      return sec
```

IBE - HashToPoint - ref[2]

HashToPoint(p, q, id, hashfcn) cryptographically hashes strings to points.
1. Let $y = HashToRange(id, p, hashfcn)$, using Algorithm 4.1.1 in RFC 5091, an element of F_p.
2. Let $x = (y^2 - 1)^{((2*p-1)/3)}$ mod p, an element of F_p.
3. Let $Q' = (x, y)$, a non-zero point in $E(F_p)$.
4. Let $Q = [(p+1)/q]Q'$, a point of order q in $E(F_p)$.

Solution

```
sage: def HTP(E,p,q,id,hashfcn):
.:        y = HTR(id,p) #Finding y
.:        x = pow((y^2-1),((2*p-1)/3),p) #Finding x
.:        Q_1 = E(x,y) #Finding (x,y) on E
.:        a_1 = int((p+1)/q)
.:        Q = a_1 * Q_1 #Point
.:        return Q
```

IBE - HashToRange

HashToRange(s,n,hashfcn) cryptographically hashes strings to integers in a range. It takes a string s, an integer n, and a cryptographic hash function hashfcn as input and returns an integer in the range 0 to n-1 by cryptographic hashing.
Note: the following implementation is not the RFC 5091 Algorithm 4.1.1.

Solution

```
sage: def HTR(id,p):
...:        h = int(hashlib.sha224(str(id)).hexdigest(),16)
...:        intTomod = mod (h, p)
...:        return intTomod
```

IBE - Encrypt ref[1]

BFencrypt encrypts a random message m for an identity string.

1. Compute $Q_{id} = HashToPoint(params, id, hashfcn)$
2. Choose a random $r \in Z_q^*$
3. Set the ciphertext to be $C = \langle rP, M \oplus HTR(g_{id}^r) \rangle$ where

$g_{id} = pairing(Q_{id}, P_{pub})$

Solution

```
sage: def Encrypt(params,id,m,Q_id):
...:        r = random.randint(1,q-1)
...:        C_1 = r * P
...:        F2 = GF(p^2,<a>)   #Field of polynomials in <a>
...:        E2 = EllipticCurve(F2,[0,1])
...:        P2 = E2(Q_id.xy()) #Same coordinates as Q_id in E
...:        z = F2.zeta(3)    #Third root of unity on F2
...:        qx,qy = Q_id.xy() #Point on E2
...:        phiQ =E2(qx*z,qy)
```

IBE - Encrypt ref[1]

Solution (cont)

```
...:        g_id = P2.weil_pairing(phiQ,p+1)
...:        g_id.integer_representation()
...:        g_id_r = pow(g_id,r)
...:        l = HTR (g_id_r,q)
...:        C_2 = (m).__xor__(int(l))
...:        C = (C_1,C_2)
...:        return C
```

BFdecrypt decrypts an encrypted session key using a private key.
Let $C = \langle C_1, C_2 \rangle$ be a ciphertext encrypted using the public key ID. To decrypt C using the private key S_{id} compute:
$C_2 \oplus HTR(pairing(S_id, C_1)) = M$.

Solution

```
sage: def Decrypt(params,C,S_id):
...:        (C_1,C_2) = C
...:        f_id = C_1.weil_pairing(S_id,q)
...:        n = HTR(f_id,q)
...:        m = int(C_2).__xor__(int(n))
...:        return m
```

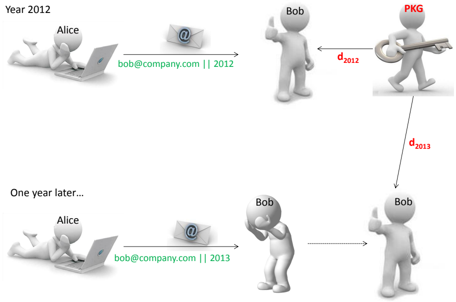
IBE - Results

Solution

```
sage: (Q_id,sk_id) = KeyGen(params,msk,id)
sage: m = random.randint(0,q-1)
sage: C = Encrypt(params,id,Q_id)
sage: M = Decrypt(params,C,sk_id)
sage: print m
>>184666388257605505968552799764283429691469241757744707484131
sage: print M
>>184666388257605505968552799764283429691469241757744707484131
```

NOTE: m and M must be equal!

1 Revocation of Public Keys:
Key expiration can be done by having Alice encrypt email sent to Bob using the public key `"bob@company.com || current-year"`. Bob can use his private key during the current year only. Once a year Bob needs to obtain a new private key from the PKG. While, Alice does not need to obtain a new certificate from Bob every time Bob refreshes his private key.



Application for IBE

2 Delegation of Decryption Keys:
Suppose Alice encrypts mail to Bob using the subject line as the IBE encryption key. Bob can decrypt mail using his master key. Now, suppose Bob has several assistants each responsible for different task. Bob gives one private key to each of his assistants corresponding to the assistant's responsibility. Each assistant can decrypt messages whose subject line falls within its responsibilities, but cannot decrypt messages intended for other assistants.