Blockchain

Nepal

Public Key Cryptography(2)

October 11, 2017

ElGamal Crypto Scheme (1/2)

Key-pair Generation

- domain parameters: p, q, g
  - p is a large prime (defines a multiplicative group over {1, 2, ..., p-1})
  - q is a prime divisor of p-1
  - g in [1, p-1] is an element of order q
    (the smallest positive t satisfying $g^t = 1 \bmod p$ is t = q)
- private key: uniformly randomly selected x from [1, q-1]
- public key: $y = g^x \bmod p$

ElGamal Crypto Scheme (2/2)

Encryption

- input: domain params p, q, g; public key y; message m in [0, p-1]
- choose uniformly random k from [1, q-1]
- compute $C_1 = g^k \bmod p$ and $C_2 = my^k \bmod p$
- output: (C_1, C_2)

Decryption

- input: domain params p, q, g; private key x; ciphertext (C_1, C_2)
- output: $C_2C_1^{-x} \bmod p = my^k.g^{-xk} \bmod p = mg^{xk}.g^{-xk} \bmod p = m$

ElGamal Security

Relation to hard problem

- Discrete logarithm problem in Zp^*
- No proof of equivalence yet though

Recovering m given p, q, g, y, C_1, C_2

is equivalent to

solving the Diffie- Hellman problem

ElGamal Crypto Scheme - Java Implementation

```
KeyPairGenerator generator = KeyPairGenerator.getInstance(" ElGamal","BC");

SecureRandom random = new SecureRandom();

generator.initialize(256, random);  // keep it at least 160 bit

KeyPair keypair = generator.genKeyPair();


// Public key
PublicKey pubKey = keyPair.getPublic();

// Private key
PrivateKey privateKey = keyPair.getPrivate();
```

ElGamal Crypto Scheme - Java Implementation

```
// Encryption
String ALGORITHM = "ElGamal/ECB/PKCS1PADDING";
Cipher cipher = Cipher.getInstance(ALGORITHM, "BC");
cipher.init(Cipher.ENCRYPT_MODE, publicKey);
byte[] cipherText = cipher.doFinal(message.getBytes());
```

```
// Decryption
String ALGORITHM = "ElGamal/ECB/PKCS1PADDING";
Cipher cipher = Cipher.getInstance(ALGORITHM, "BC");
cipher.init(Cipher.DECRYPT_MODE, privateKey);
byte[] plainText = cipher.doFinal(cipherText);
```

Key Exchange Problem

How to setup a shared key between two endpoints?

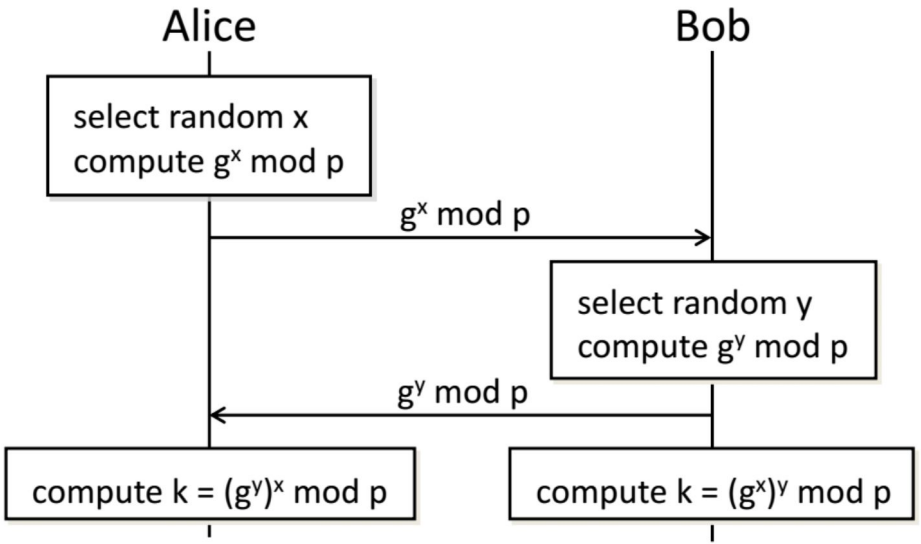
First proposed solution (1976)

Diffie-Hellman Key Exchange Protocol

Raplh Merkle, Martin Hellman, and Whitfield Diffie

Diffie-Hellman Key Exchange Protocol

Public Parameters: a large prime p, a generator element g of $Z_p^*=\{1, 2, ..., p-1\}$



Diffie-Hellman Key Exchange Problem

- Attacker (eavesdropper) can get $g^x \bmod p$ and $g^y \bmod p$

- Attacker will need x or y to compute the shared secret (g^{xy})

- It is hard to compute x from $g^x \bmod p$
  - “**discrete logarithm problem**”, no ppt algorithm exists
  - for large p, it is practically infeasible to compute x from g^x

Exercise

- Generate two Diffie-Hellman (DH) key pairs and compute the shared secret. Use this shared secret to encrypt the following message:

  “A quick brown fox jumps over the lazy dog”