

Protocols

In this section we will discuss Discrete Logarithm based protocols:

- ElGamal
- DSA Signatures

Discrete Logarithm Problem

Let (G, \times) be an abelian group.

Discrete Logarithm Problem

Given $g, h \in G$, find an x (if it exists) such that

$$g^x = h.$$

The difficulty of this problem depends on the group G :

- **Very easy**: polynomial time algorithm, e.g. $(\mathbb{Z}/N\mathbb{Z}, +)$.
- **Rather hard**: sub-exponential time algorithm, e.g. $(GF(p), \times)$.
- **Very hard**: exponential time algorithm, e.g. **Elliptic Curve groups**.

List of Hard Problems II

Given an abelian group (G, \times) and $g \in G$.

- **DLP** : Given $h \in G$ such that $h = g^x$ find x .
- **DHP** : Given $a = g^x$ and $b = g^y$ find $c = g^{xy}$.
- **DDH** : Given $a = g^x$, $b = g^y$ and $c = g^z$, determine if $z = xy$.

Later we will prove that:

- If we can solve DLP then we can solve DHP.
- If we can solve DHP then we can solve DDH.

$$\text{DDH} \leq_P \text{DHP} \leq_P \text{DLP}$$

Reductions

We will reduce one hard problem to another, which will allow us to compare the relative difficulty of the two problems, i.e. we can say

Problem A is **no harder** than Problem B

Let A and B be two computational problems. Then A is said to **polytime reduce** to B , written $A \leq_P B$ if

- There is an algorithm which solves A using an algorithm which solves B .
- This algorithm runs in polynomial time if the algorithm for B does.

Assume we have an **oracle** (or efficient algorithm) to solve **problem B** . We then use this oracle to give an efficient algorithm for **problem A** .

Reductions - $\text{DHP} \leq_P \text{DLP}$

Here we show how to reduce DHP to DLP, i.e. we give an efficient algorithm for solving the DHP given an oracle for the DLP.

Given g^x and g^y we wish to find g^{xy} .

First compute $y = \text{DLP}(g^y)$ using the oracle.

Then compute $(g^x)^y = g^{xy}$.

So DHP is no harder than DLP, i.e. $\text{DHP} \leq_P \text{DLP}$.

Remark: in some groups you can show that DHP is equivalent to DLP.

Reductions - $\text{DDH} \leq_P \text{DHP}$

Here we show how to reduce DDH to DHP, i.e. we give an efficient algorithm for solving the DDH given an oracle for the DHP.

Given elements g^x , g^y and g^z , determine if $z = x \cdot y$.

Using the oracle to solve DHP, compute

$$g^{xy} = \text{DHP}(g^x, g^y).$$

Then check whether $g^{xy} = g^z$.

So DDH is no harder than DHP, i.e. $\text{DDH} \leq_P \text{DHP}$.

Remark: in some groups you can show that DDH is probably easier than DHP.

ElGamal Encryption - Key Generation

ElGamal (1985): *A public key cryptosystem and a signature scheme based on discrete logarithms.*

Domain Parameter Generation:

- Generate a “large prime” p (> 512 bits) such that $p - 1$ is divisible by another “large prime” q (> 160 bits).
- Compute a generator g of the multiplicative group of order q in \mathbb{F}_p^* , via (for some random r)

$$g = r^{(p-1)/q} \pmod{p}.$$

until $g \neq 1$.

ElGamal Encryption - Key Generation

ElGamal (1985): *A public key cryptosystem and a signature scheme based on discrete logarithms.*

Key Generation:

- Select a random integer a , $1 \leq a \leq q - 1$ and compute

$$h \equiv g^a \pmod{p}.$$

Public key = (p, g, h) which can be published.

Private key = a which needs to be kept secret.

ElGamal - Encryption / Decryption

Bob encrypts a message for Alice as follows:

- Obtain Alice's authentic public key (p, g, h) .
- Represent the message as an integer m in $\{0, 1, \dots, p-1\}$.
- Generate a random ephemeral key k , with $1 \leq k \leq q-1$.
- Compute $c_1 = g^k$ and $c_2 = m \cdot h^k$.
- Send the ciphertext $c = (c_1, c_2)$ to Alice.

To recover the message, Alice does the following:

- Use the private key a to compute $c_1^{p-1-a} \pmod{p} \equiv c_1^{-a} \equiv g^{-ak}$.
- Recover m by computing $(c_1^{-a}) \cdot c_2 \equiv m \pmod{p}$.

Proof that decryption works:

$$(c_1^{-a}) \cdot c_2 \equiv g^{-ak} m g^{ak} \equiv m \pmod{p}.$$

ElGamal Encryption - Example

Select prime $p = 809$, then $809 - 1 = 808$ is divisible by $q = 101$.

Compute a generator $g = 3$ of the multiplicative group \mathbb{F}_p^* .

Alice chooses the private key $a = 68$ and computes

$$g^a \pmod{p} \equiv 3^{68} \pmod{p} \equiv 65.$$

Alice's **public key** then is $(p = 809, g = 3, h = 65)$, which can be published.

Alice's **private key** is $a = 68$ which she keeps secret.

ElGamal Encryption - Example

To encrypt the message $m = 100$, Bob selects a random integer $k = 89$ and computes

$$c_1 = g^k = 345 \quad \text{and} \quad c_2 = m \cdot h^k = 517.$$

Bob then sends the ciphertext (c_1, c_2) to Alice.

To decrypt, Alice first computes

$$c_1^{p-1-a} \pmod{p} \equiv 345^{740} \pmod{809} \equiv 720 \pmod{809},$$

and recovers m by computing

$$m \equiv 720 \cdot 517 \pmod{809} \equiv 100 \pmod{809}.$$

ElGamal Encryption - Security

Later we shall see that ElGamal encryption as it stands is not secure against **chosen ciphertext attacks**.

⇒ So usually a small modification is used.

However ElGamal is secure against chosen plaintext attacks, assuming the **Diffie-Hellman Problem** is **hard**.

To prove that ElGamal is secure against chosen plaintext attacks assuming the Diffie-Hellman Problem is hard:

- We give an algorithm which solves the DHP.
- Using an algorithm to break ElGamal encryption as an oracle.

ElGamal - Chosen Plaintext Attack

Recall that in the Diffie-Hellman problem we are given a group (G, \cdot) , g^x and g^y and are asked to compute g^{xy} .

Using the algorithm which breaks ElGamal encryption as an oracle, we show how to solve the DHP.

Consider the ElGamal cryptosystem with following parameters:

- Let public key be $h = g^x$.
- To construct a “ciphertext” $c = (c_1, c_2)$ we set
 $c_1 = g^y$ and c_2 is a random element of \mathbb{F}_p^* .
- Using the oracle to break ElGamal, decrypt c to obtain m .
- Finally, compute $c_2/m = g^{xy}$.

Conclusion: ElGamal is secure under chosen plaintext attack.

ElGamal - Chosen Plaintext Attack

Recall that in ElGamal encryption, we had

$$c_1 = g^k \quad \text{and} \quad c_2 = m \cdot h^k.$$

Now we have chosen $h = g^x$ and $c_1 = g^y$.

So in our proof we have taken $k = y$.

Now c_2 is random but gives a valid encryption of m , and therefore

$$c_2 = m \cdot h^y.$$

Substituting this into the expression c_2/m gives

$$\frac{c_2}{m} = \frac{m \cdot h^y}{m} = h^y = g^{xy}.$$

Perfect Security

A scheme is **perfectly secure** if a passive adversary, with **infinite** computing power can learn nothing about the plaintext given the cipher text.

This essentially means the key is as long as the message.

⇒ One time pads are perfectly secure

This is the major theorem of Shannon in the Information Theoretic area of cryptography.

Often called **Information Theoretically Secure**

Semantic Security

This is like perfect security but we only allow an adversary with **polynomially bounds** computing power.

Formally:

For all probability distributions on the message space whatever a passive adversary can compute in polynomial time about the plain text given the cipher text, they could also compute without the cipher text.

i.e. Having the ciphertext does not help finding what the message was.

Hard to understand but luckily this is equivalent to **polynomial security** which is easy to understand....

Polynomial Security

Suppose you are given

- ⇒ an encryption function f
- ⇒ two plain text messages m_1 and m_2
- ⇒ a cipher text c such that

$$c = f(m_1) \text{ or } c = f(m_2)$$

A scheme is polynomially secure if in polynomial time you cannot decide which message c is the encryption of, with probability significantly greater than 0.5.

Note: This means a polynomially secure encryption function must be non-deterministic.

ElGamal and Semantic Security

ElGamal is secure against passive attacks, under a definition of semantic security, if the [Decision](#) Diffie–Hellman problem is hard.

We leave this as an exercise

Non-Malleability

A scheme is non-malleable if given a plaintext/ciphertext pair

$$(M, C)$$

it is impossible to determine a valid ciphertext on a related message.

Note related is vaguely defined here on purpose.

ElGamal - Malleability

Recall the ElGamal ciphertext is of the form

- $(g^k, m \cdot h^k)$

where

- k is an ephemeral per message secret
- h is the public key

This is clearly malleable since on seeing this ciphertext Eve can create a valid ciphertext of the message $2m$ without ever knowing m

- $(g^k, 2 \cdot m \cdot h^k)$

Chosen CipherText Security

Malleability implies that the scheme is not secure against chosen ciphertext attacks (CCA).

Suppose Eve wants to decrypt a target ciphertext message c

She is allowed access to a decryption oracle

- But is not allowed to ask the oracle to decrypt c

She “alters” the ciphertext to c'

Obtains the decryption m' of c'

Recovers m from m' .

CCA - ElGamal

The message Eve wants to break is

- $c = (c_1, c_2) = (g^k, m \cdot h^k)$

Eve creates

- $c' = (c_1, 2c_2)$

Eve asks her oracle to decrypt c' to give m'

Then Eve computes

$$\begin{aligned}\frac{m'}{2} &= \frac{2c_2c_1^{-x}}{2} = \frac{2mh^k g^{-xk}}{2} \\ &= \frac{2mg^{xk} g^{-xk}}{2} = \frac{2m}{2} \\ &= m\end{aligned}$$

Digital Signature Algorithm

We have one digital signature scheme, RSA, why do we need another one ?

⇒ What if someone breaks the RSA algorithm ?

⇒ What if factoring is easy ?

DSA = Digital Signature Algorithm

- Sometimes called DSS = Digital Signature Standard

The **elliptic curve** variants of DSA run very fast and have smaller footprints and key sizes.

DSA is based on the difficulty of the DLOG problem in the group $GF(p)^* = \mathbb{F}_p^*$.

DSA

A DSA signature consists of two 160-bit blocks r and s .

r is a function of a 160-bit random number k

⇒ which is different for every message (like a session key).

s is a function of

⇒ the message,

⇒ the signers private key x

⇒ r .

The signature has a 1 in 2^{160} probability of being forged.

DSA : Domain Parameters

All are public:

Choose a 160 bit prime q , and a **large** prime p

$\Rightarrow p$ has 512-1024 bits.

$\Rightarrow q$ divides $p - 1$.

Calculate

$$g = h^{(p-1)/q}$$

where h is a random integer less than p and $g > 1$.

g is an element of order q in $GF(p)^*$.

$$g^q = 1 \pmod{p}.$$

DSA : Key Setup

Each user generates a secret signing key, x ,

$$\Rightarrow 0 < x < q.$$

Public key is y where

$$y = g^x \pmod{p}.$$

y is associated with the user in a certified directory

\Rightarrow e.g. via an X.509 certificate.

DSA : Signatures

To sign a message m .

⇒ User computes one-way hash $h = H(m)$.

⇒ User chooses a random **ephemeral** key, $0 < k < q$.

⇒ User computes $r = (g^k \pmod{p}) \pmod{q}$.

⇒ User computes

$$s = (h + xr) / k \pmod{q}.$$

The signature on m is the pair (r, s) .

DSA : Verification

Verifier

- ⇒ Message m
- ⇒ Signature (r, s) .
- ⇒ Signer's public key y

So verifier computes

- ⇒ $h = H(m)$
- ⇒ $a = h/s \pmod{q}$.
- ⇒ $b = r/s \pmod{q}$.
- ⇒ Accepts signature if and only if $v = r$ where
$$v = (g^a y^b \pmod{p}) \pmod{q}.$$

DSA : Baby Example

Domain Parameters $q = 13$, $p = 4q + 1 = 53$, $g = 16$.

Public/Private Key Pair $x = 3$, $y = g^x \pmod{p} = 15$.

Signature

\Rightarrow Hash of Message $h = H(m) = 5$.

\Rightarrow Ephemeral Key $k = 2$

$\Rightarrow r = (g^k \pmod{p}) \pmod{q} = 5$

$\Rightarrow s = (h + xr)/k \pmod{q} = 10$.

Verification

$\Rightarrow a = h/s \pmod{q} = 7$.

$\Rightarrow b = r/s \pmod{q} = 7$.

$\Rightarrow v = (g^a y^b \pmod{p}) \pmod{q} = 5$.

Note $v = r$ hence signature is verified.

DSA Security

The above algorithm uses the subgroup of \mathbb{F}_p^* of order q which is generated by g .

Hence the DLOG problem really is in the cyclic group $\langle g \rangle$ of order q .

For security we require

$\Rightarrow p > 2^{1024}$ to avoid attacks via the Number Field Sieve

$\Rightarrow q > 2^{160}$ to avoid attacks via BSGS (a time memory trade off).

DSA Security

Hence to achieve 80 bit of DES strength we need to operate on integers of 1024 bits in length.

This makes DSA even slower than RSA, since DSA operation is more complicated than RSA.

Would it not be good if we could use another group of size around 2^{160}

⇒ Which was not susceptible to NFS style attacks.

⇒ Which provided fast arithmetic.

DSA Generalisation

Can generalise DSA to an arbitrary finite abelian group in which the DLOG problem is hard.

We write $G = \langle g \rangle$ for a group generated by g ,

\Rightarrow Assume g has prime order $q > 2^{160}$.

\Rightarrow Assume DLOG with respect to g is hard.

\Rightarrow Assume a public function f

$$f : G \longrightarrow \mathbb{Z}/q\mathbb{Z}.$$

Each user generates a secret signing key, x ,

$\Rightarrow 0 < x < q$.

Public key is y where $y = g^x$.

DSA Generalisation

To sign a message m .

⇒ User computes one-way hash $h = H(m)$.

⇒ User chooses a random ephemeral key, $0 < k < q$.

⇒ User computes $r = f(g^k)$.

⇒ User computes

$$s = (h + xr) / k \pmod{q}.$$

The signature on m is the pair (r, s) .

DSA Generalisation

Verifier

- ⇒ Message m
- ⇒ Signature (r, s) .
- ⇒ Signer's public key y

So verifier computes

- ⇒ $h = H(m)$
- ⇒ $a = h/s \pmod{q}$.
- ⇒ $b = r/s \pmod{q}$.
- ⇒ Accepts signature if and only if $f(v) = r$ where
$$v = g^a y^b.$$

EC-DSA

This is basically how EC-DSA works.

As a group G it uses the points on some elliptic curve.

Since the EC-DLP is hard this gives a lot of security.

⇒ Hence keys sizes can be very small.

⇒ Makes the system much faster.

Clearly we can do the same for Diffie-Hellman key exchange.

⇒ EC-DH is the elliptic curve variant of Diffie-Hellman.

Will not go into gory details in this course.

⇒ ECC is becoming increasingly important in small wireless devices.

⇒ Uses less silicon/code/bandwidth/time etc etc.