Part I

Discrete Logarithm Problem

1. Discrete Logarithm Problem

1. Discrete Logarithm Problem

Groups

A group is a pair $(\mathbb{G}, \cdot)$ of a set $\mathbb{G}$ and an operator $\cdot$. The group has the following properties:

closure $\forall a, b \in \mathbb{G}, a \cdot b \in \mathbb{G}$.

associativity $\forall a, b \in \mathbb{G}, (a \cdot b) \cdot c = a \cdot (b \cdot c)$.

identity element there exists an element $e \in \mathbb{G}$, such that for every $a \in \mathbb{G}$ the equation $e \cdot a = a \cdot e = a$. The element is unique and is generally denoted as 1.

inverse element for each $a \in \mathbb{G}$, there exists an element $b \in \mathbb{G}$ such that $a \cdot b = b \cdot a = 1$.

If there is also commutativity the group is called abelian.

Finite Groups

Let $\mathbb{G}$ be a finite group or order m. For arbitrary $g \in \mathbb{G}$ consider the set:

$$\langle g \rangle = \{g^0, g^1, \dots\}$$

By Lagrange Theorem, $g^m = 1$. Let $i \leq m$ be the smallest positive integer for which $g^i = 1$. Then, the set $\langle g \rangle$ is finite and has i elements.
It can be verified that, for any g, $\langle g \rangle$ is a subgroup of $\mathbb{G}$ and has order i. We also say that i is the order of g.
Useful properties:

- for any integer x, $g^x = g^{x \bmod i}$
- $g^x = g^y$ if and only if $x = y \bmod i$
- i divides m

Cyclic Groups

If there exists g that has order m, then $\langle g \rangle = \mathbb{G}$. In this case the group is cyclic and g is a generator of $\mathbb{G}$.
If $\mathbb{G}$ is a group of prime order p, then $\mathbb{G}$ is cyclic. Furthermore, all elements of $\mathbb{G}$ are generators, except the identity.
If p is prime, then $(\mathbb{Z}_p^*, \cdot)$ is cyclic.

1. Discrete Logarithm Problem

The Discrete Logarithm

If $\mathbb{G}$ is cyclic group of order q and g is a generator, then for every $h \in \mathbb{G}$ there is a unique $x \in \mathbb{Z}_q$ such that $g^x = h$.
x is the discrete logarithm of h with respect to g, and we write $x = \log_g h$.
Some properties of logarithms hold, for example:

- $\log 1 = 0$
- $\log_g(h1 \cdot h2) = [(\log h1 + \log h2) \bmod q]$

The Discrete Logarithm Problem (DLP)

The groups of cryptographic interest have:

- a PPT algorithm $\mathcal{G}(n)$ that outputs a cyclic group $\mathbb{G}$, its order q (an n-bits integer), and a generator g.
- the group operation can be computed in polynomial time in n.
- the Discrete Logarithm Problem is hard

The Discrete Logarithm Assumption

The Discrete Logarithm Experiment, $\mathrm{DLog}_{\mathrm{Adv},\mathcal{G}}(n)$

1. Run $\mathcal{G}(n)$ and obtain $(\mathbb{G}, q, g)$
2. Choose $h \leftarrow \mathbb{G}$
3. Adv is given $\mathbb{G}, q, g, h$ and outputs $x \in \mathbb{Z}_q$
4. The experiment succeeds if $g^x = h$.

We say that the DLP is hard relative to $\mathcal{G}$ if for every PPT algorithm Adv, there is a negligible function negl such that

$$\Pr[\mathrm{DLog}_{\mathrm{Adv},\mathcal{G}}(n)\text{succeds}] \leq \mathrm{negl}(n)$$

The Computational Diffie-Hellman (CDH) Problem

This problem is related to the DLP. Given $h_1 = g^x$ and $h_2 = g^y$, define

$$\mathrm{DH}_g(h_1, h_2) = g^{xy} = h_1^y = h_2^x$$

The CDH problem is to compute $\mathrm{DH}_g(h_1, h_2)$ given randomly chosen h_1, h_2.
If DLP is easy for $\mathcal{G}$, then CDH is also easy. It is not know whether the hardness of DLP implies the hardness of CDH.

The Decisional Diffie-Hellman (DDH) Problem

The problem is to decide whether $h' = \mathrm{DH}_g(h_1, h_2)$ or h' is chosen randomly.
The DDH is hard if, for any PPT, the probability of distinguishing h' from a randomly chosen element is negligible.
If CDH is easy, then DDH is also easy. The converse is not, because there are groups in which CDH is (believed) hard and DDH is easy.

1. Discrete Logarithm Problem

Key Establishment (also Key Exchange)

Umbrella term to indicate

Key Distribution: the key is generated by one of the parties and given to the other(s)

Key Agreement: the key is the result of a distributed computation including input from all the parties

1. Discrete Logarithm Problem

Diffie-Hellman Key Exchange (DHKE)

The basic DHKE is an anonymous key agreement procotol, which is secure against passive adversaries.

The parties do not share any keys before the protocol execution. It is a fundamental building block for several, richer protocols.

General Setting of a Key Exchange Protocol

- Two entities (Alice and Bob) agree on a security parameter n.
- Alice and Bob run the protocol.
- Alice learns k_A and Bob learns k_B, where $k = k_A = k_B$ is a n-bit shared secret.

The protocol is secure if any eavesdropping adversary cannot distinguish k from any random string of length n.

Note that this is a stronger requirement than saying that Adv cannot guess k.

Since k seems like random, DHKE can be used to generate symmetric keys.

Security of a Key Exchange Protocol

The Key Exchange Experiment

- Two parties agree on n and execute the protocol. The execution results in a sequence of messages ("protocol transcript") and a key k.
- A random bit b is chosen. If $b = 0$, then $\hat{k} \leftarrow \{0,1\}^n$. If $b = 1$, then $\hat{k} := k$.
- Adv is given the protocol transcript and $\hat{k}$. It outputs b'.
- The experiment succeeds if $b = b'$.

The protocol is secure if, for any PPT adversary, the experiment succeeds with probability at most $1/2$ plus a negligible function.

1. Discrete Logarithm Problem

DHKE – Generation of Parameters

Let $\mathcal{G}(n)$ a PPT algorithm that outputs

- a cyclic group $\mathbb{G}$
- its order q such that q is an n-bit number
- a generator g of $\mathbb{G}$

The group operations in $\mathbb{G}$ can be computed in polynomial time. In practice the group and the generator are chosen from a set of standardized precomputed parameters. Two families of groups are used:

- multiplication over integers modulus a prime
- addition over elliptic curves

DHKE Protocol

DHKE Protocol

Common Input: n

1. Alice runs $\mathcal{G}(n)$ and obtains $(\mathbb{G}, q, g)$
2. Alice chooses $x \leftarrow \mathbb{Z}_q$ uniformly at random and computes $h_1 := g^x$
3. Alice sends $(\mathbb{G}, q, g, h_1)$ to Bob
4. Bob receives $(\mathbb{G}, q, g, h_1)$. He chooses $y \leftarrow \mathbb{Z}_q$ uniformly at random and computes $h_2 := g^y$. Bob sends h_2 to Alice and outputs the key $k_B := h_1^y$.
5. Alice receives h_2 and outputs the key $k_A := h_2^x$.

DHKE yields a random group element. To obtain a random string of bits, the secret k must be postprocessed by a suitable key generation function (KGF).

Security of DHKE

Let $\hat{k} = KGF(k)$. We said that a key exchange scheme is secure if the adversary cannot distinguish $\hat{k}$ from a random string.

If the Decisional Diffie-Hellman assumption is true, the secret k is indistinguishable from a random group element.

Ift KGF transforms a random group element k into a random string $\hat{k}$, then DHKE is secure. This property is difficult to prove and for most hash functions it is false.

There are some KGF that are believed to have this property (e.g. HMAC with fixed key and SHA-3).

Security Issues of DHKE

Note that DHKE is anonymous and completely insecure against active attacks. In particular two attacks:

impersonation in which one of the parties is honest and the other is not

man-in-the-middle in which a third party intercepts and modifies the messages

Therefore DHKE is used as a building block in other protocols that include authentication of the parties.

Station-to-Station (STS) Protocol

A popular protocol implementing a key exchange secure against active attackers. Exploits Digital Signatures to provide mutual authentication.

Basic STS requires that Alice and Bob know in advance each other's public (verification) key.

Full STS Alice and Bob send to each other their certificates.

A certificate is a message that binds together an identifier and a public key and signed by a trusted third party (the Certification Authority). In order to use certificates, Alice and Bob must know the CA's public key.

STS Setup

It is common to Basic and Full STS.

Basic STS Setup

Alice and Bob agree on

- a digital signature algorithm (Sign,Vrfy).
- a symmetric encryption algorithm, along with padding and mode of operation, (Enc,Dec).
- a key derivation function $KGF(\cdot)$
- the STS protocol parameters $(n, \mathbb{G}, q, g)$

Alice and Bob generate their digital signature keypairs (pk_A, sk_A) and (pk_B, sk_B) respectively. The public keys are securely deliverered to each other (Basic STS) or to the Certification Authority (Full STS).

Basic STS Protocol

Basic STS Protocol

1. Alice chooses $x \leftarrow \mathbb{Z}_q$ and computes $h_1 := g^x$
2. Alice $\rightarrow$ Bob: h_1
3. Bob chooses $y \leftarrow \mathbb{Z}_q$, computes $h_2 := g^y$, and computes $\hat{k} := KGF(h_1^y)$
4. Bob $\rightarrow$ Alice: $h_2, \mathrm{Enc}(\hat{k}, \mathrm{Sign}(sk_B, h_2 \| h_1))$
5. Alice computes $\hat{k} := KGF(h_2^x)$, decrypts with $\hat{k}$ and verifies with pk_B. If verification fails, stop.
6. Alice $\rightarrow$ Bob: $\mathrm{Enc}(\hat{k}, \mathrm{Sign}(sk_A, h_1 \| h_2))$
7. Bob decrytps and verifies with pk_A. If verification fails, stop.

In Full STS Bob and Alice send their Certificates along with their signed messages.

1. Discrete Logarithm Problem

Computing the Discrete Logarithm

There are two categories:

generic algorithms work for arbitray groups (generic algorithms)

specific algorithms work for a specific groups

All known generic algorithms are exponential. Instead, specific algorithms can be very efficient, think of $(\mathbb{Z}_n, +)$. Therefore, the hardness of DLP for a group depends on the existence of an efficient specific algorithm.

1. Discrete Logarithm Problem

Baby Step, Giant Step

Computes the DL in time $O(\sqrt{q}\,\mathrm{polylog}(q))$.

We want to solve the equation $g^x = y$.
We choose $t = \lceil \sqrt{q} \rceil$ and write two lists:

baby steps $g^0, g^1, \ldots g^t$

giant steps $yg^0, yg^{-t}, yg^{-2t}, \ldots, yg^{-\lceil q/t \rceil t}$

Each list contains $O(\sqrt{q})$ elements. Suppose we find two elements that are the same in both lists, say $g^i = yg^{-kt}$.
Then we can solve and obtain $\log_g y = kt + i \bmod q$.
We are sure to find two equal elements because all the numbers between 0 and $t^2 - 1$ (and thus between 0 and q) can be written as $kt + i$ for some k and some i.

Baby Step, Giant Step

Pseudocode

Baby Step, Giant Step

$t := \lceil \sqrt{q} \rceil$
for $i = 0, \ldots, t$ **do**
    $\alpha_i := g^i$
end for
sort the pairs (i, α_i) by the second component
for $k = 0, \ldots, t$ **do**
    $\beta_k := yg^{-kt}$
    if $\alpha_i = \beta_k$ for some i **then return** $kt + i \bmod q$
    end if
end for

1. Discrete Logarithm Problem

Pohlig-Hellman

Effective when the factorization of q is known.
We need the following

Lemma

If $\text{ord}(g) = q$ *and* $m|q$, *then* $\text{ord}(g^m) = q/m$.

Suppose that we know a factorization of q

$$q = \prod_{i=1}^{k} q_i$$

It is not necessary that the q_i are primes, but they must be pairwise relatively prime.

Pohlig-Hellman

We can write k equations, each in a smaller group:

$$\left(g^{q/q_i}\right)^{x_i} = \left(g^x\right)^{q/q_i} = y^{q/q_i} \quad \forall 1 \leq i \leq k$$

The k solutions x_i form a system of modular equations

$$\begin{aligned} x &= x_1 \bmod q_1 \\ &\dots \\ x &= x_k \bmod q_k \end{aligned}$$

These can be combined using the Chinese Reminder Theorem.

Pohlig-Hellman

Additional Considerations

The lowest complexity is achieved when we know the full factorization of q. Since the equations require that the moduli are pairwise relatively prime, when a prime q_i appears r_i times we must write e single equation $\bmod q_i^{r_i}$.
Instead of a complexity $O(\sqrt{q_i^{r_i}})$ we can achieve a complexity $O(\sqrt{q_i})$ with additional considerations.

Pohlig-Hellman

Additional Considerations

Call x the solution of the equation $x \equiv \log_g y \bmod q_i^{r_i}$. We can write:

$$x = x_0 + x_1 q_i + x_2 q_i^2 + \cdots + x_k q_i^k + \cdots + x_{r_i} q_i^{r_i} \quad 0 \leq x_k < q_i$$

Multiply both sides by q/q_i

$$x\frac{q}{q_i} = x_0\frac{q}{q_i} + qn$$

with $n = x_1 + x_2 q_i + \ldots$.
Substitute in the original equation $y = g^x$

$$y^{q/q_i} = g^{xq/q_i} = g^{x_0 q/q_i}\left(g^q\right)^n = g^{x_0 q/q_i}$$

Solve and find x_0, e.g. using BSGS.

Pohlig-Hellman

Additional Considerations

Write

$$y_1 = yg^{-x_0} = g^{x-x_0} = g^n$$

Raise by q/q_i^2

$$y_1^{q/q_i^2} = g^{x_1 q/q_i}\left(g^q\right)^{x_2+qx_3+\dots} = g^{xq/q_i}$$

Solve and find x_1. Then proceed similarly until you find all the x_{r_i}

Attacks to the Discrete Logarithm Problems

- generic algorithms

  Baby Step, Giant Step $O(\sqrt{q}\,\mathrm{polylog}(q))$

  Pohlig-Hellman $O(\mathrm{polylog}(q)\max_i \sqrt{q_i})$

- specific algorithms for $(\mathbb{Z}_p^*, \cdot)$ with p an n-bit prime

  index calculus $2^{O(\sqrt{n \log n})}$

  general number field sieve $2^{O(\sqrt[3]{n \log^2 n})}$

The existence of subexponential algorithms for $\mathbb{Z}_p^*$, makes that group less appealing than other groups such as the Elliptic Curves.

1. Discrete Logarithm Problem

Discrete Logarithm Integrated Encryption Scheme (DLIES)

Encryption based on DLP over modular arithmetic ($\mathbb{Z}_p$) are not used much. DLIES is a hybrid encryption scheme providing semantic security and message integrity under assumption of the DDH problem.

To send a message to Bob, Alice needs the following:

- a key derivation function (KDF), a Message Authentication algorithm (Mac/Vrfy), a symmetric encryption scheme (Enc/Dec)
- a strong prime p and a generator g of $\mathbb{Z}_p$
- Bob's public key pk_B

Bob private key sk_B is a random element of $\mathbb{Z}_p$. The public key is $pk_B = g^{sk_B}$.

Discrete Logarithm Integrated Encryption Scheme (DLIES)

Encryption

To send message m, Alice executes the following algorithm.

1. Generate a random number $r \leftarrow \mathbb{Z}$ and calculate a shared secret $s = (pk_B)^r \bmod p$.
2. Calculate two symmetric keys $k_1 \| k_2 = KGF(s)$
3. Encrypt the message $c = \mathrm{Enc}(k_1, m)$
4. Compute the MAC $t = \mathrm{Mac}(k_2, c)$
5. Send $g^r \| c \| t$

DLIES Decryption

Bob receives $r||c||t$ and executes the following.

1. Calculate the shared secret $s = (g^r)^{sk_B} \bmod p$
2. Calculate the two symmetric keys $k_1||k_2 = KGF(s)$
3. Verify the tag $\text{Vrfy}(k_2, t, c)$
4. Decrypt the message $m = \text{Dec}(k_1, c)$

1. Discrete Logarithm Problem

The Digital Signature Standard (DSS)

The DSS, known also as Digital Signature Algorithm (DSA) is widely used and has resisted many years with no serious attacks. Unfortunately there is also no proof of security.

DSS Key Generation (Gen)

On input n:

1. Find p, n-bit prime
2. Find q, n-bit prime such that $q|(p-1)$ and $q^2 \nmid (p-1)$
3. Find g, generator of a subroup of $\mathbb{Z}_p^*$ having order q
4. Choose $x \leftarrow \mathbb{Z}_q$ and set $y := g^x \bmod p$
5. Set $pk = (p, q, g, y)$ and $sk = p, q, g, x$.

Also, choose a cryptographic hash function $H : 0,1^* \rightarrow \mathbb{Z}_q$.

DSS Signature Creation

DSS Signature (Sign)

On input a message $m \in 0,1^*$

1. Choose a security nonce $k \leftarrow \mathbb{Z}_q^*$ and set
$$r := (g^k \bmod p) \bmod q$$
2. Compute
$$s := (H(m) + xr)k^{-1} \bmod q$$
3. Output (r, s)

In the unlikely case that r or s are equal to 0, the algorithm is run again with a different nonce.

DSS Signature Verification

DSS Signature (Vrfy)

On input a message m and a signature (r, s)

1. Compute

$$u_1 := H(m)s^{-1} \bmod q$$
$$u_2 := rs^{-1} \bmod q$$

2. Output 1 if

$$r = (g^{u_1} y^{u_2} \bmod p) \bmod q$$

DSS Dignature Correctness

The scheme is correct because:

$$\begin{aligned} g^{H(m)s^{-1}} y^{rs^{-1}} &= g^{H(m)(H(m)+xr)^{-1}k} g^{xr(H(m)+xr)^{-1}k} \bmod p \\ &= g^{(H(m)+xr)(H(m)+xr)^{-1}k} \bmod p \\ &= g^{k} \bmod p \end{aligned}$$

Since they are equal mod p, they are also equal mod q.

DSS Nonce Reuse Attack

DSS is subject to a nonce reuse attack if two distinct messages m, m' are signed using the same nonce k.

Signature of m

$$r := (g^k \bmod p) \bmod q$$
$$s := (H(m) + xr)k^{-1} \bmod q$$

Signature of m'

$$r' := (g^k \bmod p) \bmod q$$
$$s' := (H(m') + xr)k^{-1} \bmod q$$

We have that $r = r'$. Then, we can write:

$$s - s' \equiv (H(m) + xr)k^{-1} - (H(m') + xr)k^{-1} \pmod q$$
$$k(s - s') \equiv H(m) - H(m') \pmod q$$

which can be solved for k. Then x can be calculated from:

$$xr \equiv ks - H(m) \pmod q$$

Comments on DSA Signature

The current standard (FIPS 186-3), specifies the following pairs for the sizes of p (key size) and q (signature size): (1024,160), (2048,224), (2048,256), and (3072,256).

The hash function can be SHA-1 or any SHA-2 with an hash size longer than the size of q.

Key and signature sizes choice depends on the time horizon over which data security must be guaranteed.

1. Discrete Logarithm Problem

Schnorr Signature Scheme

The Schnorr scheme is provably secure in the oracle model (i.e. in the assumption that we can build a random function). It is used for entity authentication in challenge response protocols.

Schnorr Signature Scheme

Gen choose a cyclic group G and an element $g \in G$ of order q. The private key is an integer $x \in \mathbb{Z}_q$. The public key is $y = g^x$.

Schnorr Signature Scheme

Schnorr Signature Scheme

Sign

1. Choose an ephemeral key $k \leftarrow \mathbb{Z}_q$
2. Compute the ephemeral public key $r := g^k$
3. Compute $e := h(m \| r)$
4. Compute $s := k + xe \bmod q$

The signature is the pair (e, s).

Vrfy

1. Compute $r := g^s y^{-e}$
2. Iff $e = h(m \| r)$ return 1.

The scheme is correct, because

$$g^s y^{-e} = g^{k+xe} g^{-xe} = g^k$$

Schnorr Identification Scheme

Suppose a card reader wants to authenticate a card.

The card reader (R) knows the public key, y.

The card (C) holds the private key, x.

Schnorr Identification Scheme

1. (commitment) $C \to R$: $r = g^x$ (can be precomputed)
2. (challenge) $R \to C$: e
3. (response) $C \to R$: $s := k + xe \bmod q$

The point of the initial commitment is to stop either the challenge being concocted so as to reveal the private key, or the response being concocted so as to fool the reader.