Diffie-Hellman Key Exchange (Finite Field)

The Diffie-Hellman Key Exchange is a method of securely exchanging cryptographic keys over a public channel.

Overview

The Diffie-Hellman protocol allows two parties to establish a shared secret key without ever having to transmit the key itself. This is achieved through the use of modular arithmetic and the difficulty of the discrete logarithm problem.

Key Components

1. A large prime number p
2. A generator g (usually a primitive root modulo p)
3. Private keys a and b (chosen by Alice and Bob respectively)
4. Public keys A and B (computed and shared by Alice and Bob)

The Protocol

1. Alice and Bob agree on public parameters p and g.
2. Alice chooses a private key a and computes her public key: $A = g^a \bmod p$
3. Bob chooses a private key b and computes his public key: $B = g^b \bmod p$
4. Alice and Bob exchange their public keys.
5. Alice computes the shared secret: $s = B^a \bmod p$
6. Bob computes the shared secret: $s = A^b \bmod p$
7. Both Alice and Bob now have the same shared secret s.

Mathematical Proof

The Diffie-Hellman Key Exchange results in the same shared secret for both parties.

The reason this works is due to the properties of modular exponentiation:

$s = B^a \bmod p = \left(g^b\right)^a \bmod p = g^{ab} \bmod p$

$s = A^b \bmod p = \left(g^a\right)^b \bmod p = g^{ab} \bmod p$

As we can see, both computations result in the same value: $g^{ab} \bmod p$.

Listing 1: diffehellman_a.py

```
from random import getrandbits

class DiffeHellmanKeyExchangeA:
    # 512-bit pre-shared g and p values
    g =
0x678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e2935630e1c2062354d0da20a6c416e50be794ca4
    p =
0xfca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899bcd132acd50d99151bdc43ee737592e17

    def __init__(self):
        self.private_key = getrandbits(512)
        self.public_key = pow(self.g, self.private_key, self.p)

    def recv_public_key(self, other_combination: int):
        self.shared_secret = pow(other_combination, self.private_key, self.p)
```

Listing 2: diffehellman_b.py

```
from random import getrandbits

class DiffeHellmanKeyExchangeB:
    # 512-bit pre-shared g and p values
    g =
0x678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e2935630e1c2062354d0da20a6c416e50be794ca4
    p =
0xfca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899bcd132acd50d99151bdc43ee737592e17
    def __init__(self):
        self.private_key = getrandbits(512)
        self.public_key = pow(self.g, self.private_key, self.p)

    def recv_public_key(self, other_combination: int):
        self.shared_secret = pow(other_combination, self.private_key, self.p)
```

Listing 3: diffehellman.py

```
from diffehellman_a import DiffeHellmanKeyExchangeA
from diffehellman_b import DiffeHellmanKeyExchangeB

# This file represents the Public Channel
# over which A and B communicate
print(f"g={hex(DiffeHellmanKeyExchangeA.g)}")
print(f"p={hex(DiffeHellmanKeyExchangeA.p)}")

# Agree on g and p, stored in implementation
A = DiffeHellmanKeyExchangeA()
print(f"A.public_key={hex(A.public_key)}")
# generate g^a mod p, store in A.public_key
B = DiffeHellmanKeyExchangeB()
print(f"B.public_key={hex(B.public_key)}")
# generate g_b mod p, store in B.public_key

# Exchange
A.recv_public_key(B.public_key)
B.recv_public_key(A.public_key)

# Assertion that the secret is same
assert A.shared_secret == B.shared_secret

print(f"A.shared_secret={hex(A.shared_secret)}")
```

Figure 1: Output