Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

SI

Laboratory work # 3

DiffieHellman key exchange

Author:
Petru Negrei

Supervisor:
A. Railean

November 2014

1 Introduction

1.1 Objective

Construct the DiffieHellman key exchange algorithm.

1.2 Definition

DiffieHellman key exchange (DH)is a specific method of exchanging cryptographic keys. It is one of the earliest practical examples of key exchange implemented within the field of cryptography. The DiffieHellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

2 Description

DiffieHellman establishes a shared secret that can be used for secret communications while exchanging data over a public network. The following diagram illustrates the general idea of the key exchange by using colors instead of a very large number. The crucial part of the process is that Alice and Bob exchange their secret colors in a mix only. Finally this generates an identical key that is computationally difficult to reverse for another party that might have been listening in on them. Alice and Bob now use this common secret to encrypt and decrypt their sent and received data.

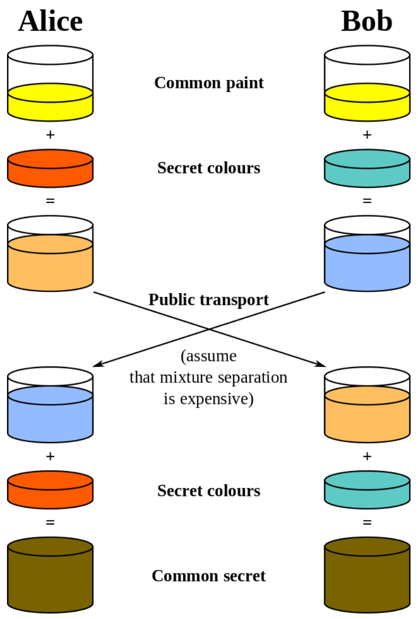

Illustration of the DiffieHellman Key Exchange

3 Implementation

Although there is an implementation of the Diffie-Hellman key exchange protocol in ruby OpenSSL module. I implemented the algorithm again using small numbers for testing.

```ruby
require 'prime'

# an implementation of the Diffie-Hellman key exchange protocol
module OpenSSL

   # p - prime. g- generator
   # pub_key - per-session public key matching the private key
   # this needs to be passed to #compute_key.
   # #priv_key - per-session private key
   attr_reader :p, :g, :priv_key

   class DH

      def initialize str="#{p}:#{g}"
         @p, @g = str.split(":").map(&:to_i)
      end
      def priv_key
         @priv_key ||= rand(1000) + 1
      end

      def to_der
         "#{p}:#{g}"
      end

      def generate_key!
         @pub_key ||= (g**priv_key) % p
      end

      def compute_key pub_key
         (pub_key ** priv_key) % p
      end

      def p
         @p ||= Prime.each(10**3).to_a.sample
      end

      def g
         5 # or something else
      end

      def pub_key
        @pub_key
      end

   end
end
```

Verification of working algorithm.

```
    dh1 = OpenSSL::DH.new
    dh1.generate_key!                        #generate the per-session key pair
    der = dh1.to_der                         #you may send this publicly to the participating party
    dh2 = OpenSSL::DH.new(der)
    dh2.generate_key!                        #generate the per-session key pair

    symm_key1 = dh1.compute_key(dh2.pub_key)
    symm_key2 = dh2.compute_key(dh1.pub_key)

    puts symm_key1 == symm_key2              # => true
```

4 Conclussion

After making this laboratory work I learn about DiffieHellman method of exchanging cryptographic keys. The protocol is considered secure against eavesdroppers if G and g are chosen properly, the order of G should have a large prime factor to prevent use of the PohligHellman algorithm. Also if parties use random number generators whose outputs are not completely random there is a possibility to predict to some extent.