

Cryptography

Diffie–Hellman  
Key Exchange

Diffie–Hellman  
Key Exchange  
Algorithm

Analysis of DHKE

Man-in-the-Middle  
Attack on DHKE

Implementations of  
DHKE

Diffie–Hellman in  
OpenSSL

DHKE in Python

# Diffie–Hellman Key Exchange

Cryptography

School of Engineering and Technology  
CQUniversity Australia

Prepared by Steven Gordon on 20 Feb 2020,  
dh.tex, r1798

# Contents

## Diffie–Hellman Key Exchange Algorithm

### Analysis of DHKE

### Man-in-the-Middle Attack on DHKE

### Implementations of DHKE

### Diffie–Hellman in OpenSSL

### DHKE in Python

# Diffie–Hellman Key Exchange

- ▶ Diffie and Hellman proposed public key cryptosystem in 1976
  - ▶ Motivation: solve the problem of how to exchange secret keys for symmetric key crypto
  - ▶ Proposed protocol for exchanging secrets using public keys
  - ▶ Merkle also contributed to the idea; sometimes called Diffie–Hellman-Merkle key exchange
- ▶ DHKE is algorithm for exchanging secret key (not for secrecy of data)
  - ▶ E.g. two users want to use symmetric key crypto, but need to first exchange a secret key
- ▶ Based on discrete logarithms
  - ▶ Easy to calculate exponential modulo a prime
  - ▶ Infeasible to calculate inverse, i.e. discrete logarithm

3

It is important to note that DHKE is a “key exchange” protocol. The purpose is for two users to exchange a secret key. Once a secret key has been exchanged with DHKE, the two users can then use that secret key for other purposes (e.g. for encrypting data using AES).

If you do not know what a discrete logarithm is, it is worth refreshing your knowledge in number theory from Chapter ??.

# Diffie–Hellman Key Exchange (algorithm)

*One-time setup.* A and B agree upon public values prime  $p$  and generator  $g$ , where  $g < p$  and  $g$  is a primitive root of  $p$ .

*Protocol.*

1. A: select private  $PR_A < p$
2. A: calculate public  $PU_A = g^{PR_A} \bmod p$
3. A  $\rightarrow$  B: send  $PU_A$
4. B: select private  $PR_B < p$
5. B: calculate public  $PU_B = g^{PR_B} \bmod p$
6. B: calculate secret  $K_B = PU_A^{PR_B} \bmod p$
7. B  $\rightarrow$  A: send  $PU_B$
8. A: calculate secret  $K_A = PU_B^{PR_A} \bmod p$

*Result.*  $K_A = K_B$  is the shared secret value

The values  $p$  and  $g$  are either agreed upon in advance, or selected by one user and sent to the other in the first message. Both values are public; the attacker is assumed to know them.

When two users need to exchange a shared secret, one of them initiates the protocol. User A and B actually perform the same steps, but just with different values. First a private value  $PR$  is randomly selected. Then a public value  $PU$  is calculated. Both users exchange their public  $PU$  values (and the attacker may learn them). Finally, both users calculate their private values  $K$  based on their own  $PR$  and received  $PU$ . The values and calculations are designed such that the  $K$  calculated by each user will be the same.  $K$  is the shared secret key.

# Diffie–Hellman Key Exchange (exercise)

Assume two users, A and B, have agreed to use DHKE with prime  $p = 19$  and generator  $g = 10$ . Assuming A randomly chose private  $PR_A = 7$  and B randomly chose private  $PR_B = 8$ , find the shared secret key.

# Contents

## Diffie–Hellman Key Exchange Algorithm

## Analysis of DHKE

## Man-in-the-Middle Attack on DHKE

## Implementations of DHKE

## Diffie–Hellman in OpenSSL

## DHKE in Python

# Requirements of DHKE

1. Same shared secret:  $K_A$  and  $K_B$  must be identical
2. Computational efficiency: Easy to calculate  $PU$  and  $K$
3. Secure: Infeasible to determine  $PR$  or  $K$  from known values
  - ▶ Attacker knows 3 public values in  $PU_A = g^{PR_A} \bmod p$
  - ▶ Must be practically impossible to find the 4th value  $PR_A$

While we don't show it here, it can easily be proved that DHKE will produce the same value of  $K$  for both users.

Modular exponentiation, while slow with big numbers, is easy to calculate, i.e. can be achieved in less than seconds.

The inverse operation of modular exponentiation, referred to as a discrete logarithm, is hard to calculate. With large enough values, it is considered impossible to calculate.

Cryptography

Diffie–Hellman  
Key Exchange

Diffie–Hellman  
Key Exchange  
Algorithm

Analysis of DHKE

Man-in-the-Middle  
Attack on DHKE

Implementations of  
DHKE

Diffie–Hellman in  
OpenSSL

DHKE in Python

# Prove Identical Keys in DHKE (question)

Prove that user A and user B will always calculate the same shared secret key in DHKE. That is, prove that  $K_A = K_B$ .



# Brute Force Attack on PR in DHKE (question)

Assuming you have intercepted  $PU_A = 15$  from the DHKE exercise, how would you perform a brute force attack to find  $PR_A$ ? How could such a successful brute force attack be prevented in practice?

# Discrete Logarithm Attack in DHKE (exercise)

Assuming a brute force attack is not possible, write an equation that the attacker would have to solve to find  $PR_A$ .

# Discrete Logarithm is Computationally Hard Problem

- ▶ Discrete Logarithm Problem:

given  $g, p$  and  $g^x \bmod p$ , find  $x$

- ▶ For certain values of  $p$ , considered computationally hard
  - ▶  $p$  is a safe prime, i.e.  $p = 2q + 1$  where  $q$  is a large prime
  - ▶  $p$  is very large, usually at least 1024 bits
- ▶ 2016: Discrete logarithm with 768 bit prime  $p$  was solved within 5300 core years on 2.2GHz Xeon E5-2660 processor
- ▶ Considered harder to solve than equivalent integer factorisation
  - ▶ 768 bit integer factored in 2000 core years

# Contents

Diffie–Hellman  
Key Exchange  
Algorithm

Diffie–Hellman Key Exchange Algorithm

Analysis of DHKE

Analysis of DHKE

Man-in-the-Middle  
Attack on DHKE

**Man-in-the-Middle Attack on DHKE**

Implementations of  
DHKE

Implementations of DHKE

Diffie–Hellman in  
OpenSSL

Diffie–Hellman in OpenSSL

DHKE in Python

DHKE in Python

## MITM Attack on DHKE (exercise)

Consider the “Diffie–Hellman Key Exchange” exercise where user A chooses  $PR_A = 7$  and B chooses  $PR_B = 8$ . Show how a MITM can be performed such that an attacker Q can decrypt any communications between A and B that use the secret shared between A and B.

# Contents

## Diffie–Hellman Key Exchange Algorithm

## Analysis of DHKE

## Man-in-the-Middle Attack on DHKE

## Implementations of DHKE

## Diffie–Hellman in OpenSSL

## DHKE in Python

# Selecting Public Parameters $p$ and $g$

- ▶ Some (older) communication protocols defined a fixed value of  $p$  and  $g$ 
  - ▶ All clients and servers use the same values
- ▶ Newer protocols allow for an exchange of values (e.g. a Group Exchange protocol)
- ▶ Example fixed value in older versions of SSH (diffie-hellman-group1-sha1 using Oakley Group 2)

$$p = 2^{1024} - 2^{960} - 1 + 2^{64} \times (2^{894} \times \pi + 129093)$$

$$g = 2$$

$p$  is 1024 bits in length

As  $p$  and  $q$  are public and known to the attacker, using the same values all the time should not be a problem. Exchanging values involves extra communication overhead and also processing overhead. However following the principle of changing keys frequently to give an attacker less chance to compromise them, many protocols now support the ability to change the public parameters.

# Contents

## Diffie–Hellman Key Exchange Algorithm

## Analysis of DHKE

## Man-in-the-Middle Attack on DHKE

## Implementations of DHKE

## **Diffie–Hellman in OpenSSL**

## DHKE in Python



Cryptography

Diffie–Hellman  
Key Exchange

Diffie–Hellman  
Key Exchange  
Algorithm

Analysis of DHKE

Man-in-the-Middle  
Attack on DHKE

Implementations of  
DHKE

Diffie–Hellman in  
OpenSSL

DHKE in Python

# Contents

Diffie–Hellman Key Exchange Algorithm

Analysis of DHKE

Man-in-the-Middle Attack on DHKE

Implementations of DHKE

Diffie–Hellman in OpenSSL

**DHKE in Python**

Cryptography

# DHKE in Python Cryptography Library

Diffie-Hellman  
Key Exchange

▶ <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/>

Diffie-Hellman  
Key Exchange  
Algorithm

Analysis of DHKE

Man-in-the-Middle  
Attack on DHKE

Implementations of  
DHKE

Diffie-Hellman in  
OpenSSL

DHKE in Python