Lab 4 Solutions

Cody Lewis

Part 1

1

Key transport involves the parties transferring a symmetric key between each other, for example, with two parties, one could use public key cryptography to encrypt and transfer a symmetric key to the other,

$$A \rightarrow B : E_{Pu_B}(K). \tag{1}$$

Key agreement involves the parties collaboratively generating a symmetric key, for example, the Diffie-Hellman key exchange,

$$\begin{aligned} A \rightarrow B :& g^a \bmod p && (2)\\ B \rightarrow A :& g^b \bmod p && (3)\\ \Rightarrow K =& g^{ab} \bmod p. && (4) \end{aligned}$$

2

The Diffie Hellman key exchange protocol starts by publicly announcing a prime, p, and a generator, g. Next the parties each choose a secret key, x_i, and send to each other $g^{x_i} \bmod p$. From that, they may calculate the symmetric key, $g^{\prod_{i=1}^{n} x_i} \bmod p$.

This works by taking the common value, g, adding a unique value, x_i then sending them mixed together. This allows each party to mix their unique values with the common ones, generating a symmetric key among all of the parties.

3

A public key certificate is a certificate that states that a particular public key belongs to an identity. This allows a user to verify that a public key actually belongs to who they think it does.

4

Public Key Infrastructure is the structure that manages and allows for the usage of public key certificates.

5

Cross-certification allows multiple parties each holding public key, to find a common ancestor among their public key certificate authorities. This establishes that they both trust a common authority further up in the hierarchy and thus can trust each other.

Part 2

6

a

Item	Value
Version	3
Certificate Serial Number	00:8F:93:94:05:B8:B4:1A:0F:B3:67:B2:5E:73:96:61:B1
Signature Algorithm	PKCS #1 SHA-256 With RSA Encryption
Issuer Name	COMODO RSA Organization Validation Secure Server CA
Period of Validity	January 15, 2018, 12:00:00 AM GMT to April 11, 2021, 11:59:59 PM GMT
Subject Name	*.newcastle.edu.au

b

The public key is 2 048 bits long, hence, it is acceptable for both data encryption and verification,

c

The modulus N is 2 048 bits long.

7

a

B is provided with a fresh key on a per session basis.

b

Step 3 would be suceptable to a replay attack and since B is not provided key freshness, such an attack where the session key is comprimised will be sucessful without B being able to detect any issues. One way to fix this would be by the addition of a time stamp in the form of a nonce to this step,

$$TS \to B : E_{K_B}(N_{TS}, K_{AB}, A, N_A). \tag{5}$$

8

b

The man-in-the-middle attack on Diffie-Hellman involves an adversary modifying the communication between the 2 parties during agreement. The adversary makes it so each party unknowingly agrees on a symmetric key with the adversary rather than the other party.

c

Since they know each other's public keys, A and B would notice if an adversary was performing a man-in-the-middle attack, hence, it will not work. However, if the adversary continually performs the attack, they could cause the communication between A and B to be effectively interrupted, as they will not be able to inverse the additions made during the attack.

d

This method is convenient as most users do not own a certificate, although, it also means that trusted secure communication can be perform towards the web server, rather than the client.

9

Below is a python implementation of the Diffe-Hellman key exchange, and the man-in-the-middle attack on it,

```python
#!/usr/bin/env python3

import random

'''
A simulation of the Diffie-Hellman Key exchange, and MITM attack

Author: Cody Lewis
Date: 2019-08-25
'''

def KeyGen(p, g, mitm=False, verbose=False):
    '''Generate the public and private keys.'''
    gen = random.SystemRandom()  # Cryptographic rng
    pub_pri_keys = dict()
    if mitm:
        pr_E = gen.randrange(1, p + 1)
        pub_pri_keys.update({"E": {"pu": pow(g, pr_E, p), "pr": pr_E}})
    pr_A = gen.randrange(1, p + 1)
    pr_B = gen.randrange(1, p + 1)
    pub_pri_keys.update(
        {
            "A": {"pu": pow(g, pr_A, p), "pr": pr_A},
            "B": {"pu": pow(g, pr_B, p), "pr": pr_B}
        }
    )
    pub_pri_keys.update({"p": p, "g": g})
    if verbose:
        print("Generated public/private keys")
    return pub_pri_keys


def DHex(keys, mitm=False, verbose=False):
    '''Compute the shared key.'''
    result = None
    if mitm:
        result = {
            "K_AE": pow(keys["A"]["pu"], keys["E"]["pr"], keys["p"]),
            "K_EA": pow(keys["E"]["pu"], keys["A"]["pr"], keys["p"]),
            "K_BE": pow(keys["B"]["pu"], keys["E"]["pr"], keys["p"]),
            "K_EB": pow(keys["E"]["pu"], keys["B"]["pr"], keys["p"])
        }
    else:
        result = {
            "K_AB": pow(keys["A"]["pu"], keys["B"]["pr"], keys["p"]),
            "K_BA": pow(keys["B"]["pu"], keys["A"]["pr"], keys["p"]),
        }
    if verbose:
        print("Generated session keys")
    return result
```

```python
def print_keys(keys):
    '''Print out a dictionary in a new line format.'''
    for k, v in keys.items():
        print(f"{k}: {v}")

if __name__ == '__main__':
    P = 223
    G = 79
    print("Performing standard Diffie-Hellman")
    KEYS = KeyGen(223, 79, verbose=True)
    print_keys(KEYS)
    print_keys(DHex(KEYS, verbose=True))
    print()
    print("Performing Man-in-the-Middle")
    KEYS = KeyGen(223, 79, mitm=True, verbose=True)
    print_keys(KEYS)
    print_keys(DHex(KEYS, mitm=True, verbose=True))
```

Below is an output,

```
Performing standard Diffie-Hellman
Generated public/private keys
A: {'pu': 108, 'pr': 171}
B: {'pu': 79, 'pr': 223}
p: 223
g: 79
Generated session keys
K_AB: 108
K_BA: 108

Performing Man-in-the-Middle
Generated public/private keys
E: {'pu': 100, 'pr': 188}
A: {'pu': 124, 'pr': 218}
B: {'pu': 93, 'pr': 55}
p: 223
g: 79
Generated session keys
K_AE: 148
K_EA: 148
K_BE: 156
K_EB: 156
```