Einsendeaufgabe 4

Sicherheit

Wintersemester 2025/2026
02.02.2026

EINGEREICHT BEI
FernUniversität in Hagen
Fakultät für Mathematik und Informatik
Kooperative Systeme
Verteilte Systeme
Prof. Dr. Christian Icking

EINGEREICHT DURCH
Emily Lucia Antosch
emilyluciaantosch@web.de
Hamburg

Contents

1. Aufgabe 4.1

Client:

```python
import socket
import random


def get_binary_representation(x: int) -> list[int]:
    if x <= 0:
        raise ValueError("x must be a positive integer")

    bits = []
    s = x
    while s > 0:
        xi = s % 2  # 0 or 1 for even or odd
        bits.append(xi)
        s = (s - xi) // 2  # integer division without remainder

    return bits  # bits[0] is LSB, bits[-1] is MSB


def square_and_multiply(g: int, x: int, n: int) -> int:
    # Get binary representation of x
    bits = get_binary_representation(x)
    l = len(bits)

    z = 1
    # Iterate from MSB (l-1) down to LSB (0)
    for i in range(l - 1, -1, -1):
        z = (z * z) % n  # z := z^2 mod n
        if bits[i] == 1:
            z = (z * g) % n  # z := (z * g) mod n

    return z


def main():
    # Diffie-Hellman parameters (publicly known, must match server)
    n = 27803
    g = 5

    # Client's private key (random, kept secret)
    x = 21131

    # Compute client's public value: A = g^a mod p
    A = square_and_multiply(g, x, n)
```

```
44
45      print("Diffie-Hellman Key Exchange - Client")
46      print(f"Generator g: {g}")
47      print(f"Prime p: {n}")
48      print(f"Client private key a: {x}")
49      print(f"Client public value A = g^a mod p: {A}")
50
51      # Connect to server via TCP
52      host = "localhost"
53      port = 12345
54
55      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
        client_socket:
56          print(f"\nConnecting to server at {host}:{port}...")
57          client_socket.connect((host, port))
58          print("Connected!")
59
60          # Send client's public value A to server
61          client_socket.sendall(str(A).encode("utf-8"))
62          print(f"Sent client's public value A: {A}")
63
64          # Receive server's public value B
65          data = client_socket.recv(4096)
66          B = int(data.decode("utf-8"))
67          print(f"\nReceived server's public value B: {B}")
68
69          # Compute shared session key: K = B^a mod p
70          shared_key = square_and_multiply(B, x, n)
71
72          print("\n")
73          print("KEY EXCHANGE COMPLETE")
74          print(f"Shared session key K = B^a mod p:")
75          print(f"{shared_key}")
76
77  if __name__ == "__main__":
78      main()
```

Server:

```
1   import socket                                                    Python
2   import random
3
4
5   def get_binary_representation(x: int) -> list[int]:
6       if x <= 0:
7           raise ValueError("x must be a positive integer")
8
```

```
9       bits = []
10      s = x
11      while s > 0:
12          xi = s % 2  # 0 or 1 for even or odd
13          bits.append(xi)
14          s = (s - xi) // 2  # integer division without remainder
15
16      return bits  # bits[0] is LSB, bits[-1] is MSB
17
18
19  def square_and_multiply(g: int, y: int, n: int) -> int:
20      # Get binary representation of x
21      bits = get_binary_representation(y)
22      l = len(bits)
23
24      z = 1
25      # Iterate from MSB (l-1) down to LSB (0)
26      for i in range(l - 1, -1, -1):
27          z = (z * z) % n  # z := z^2 mod n
28          if bits[i] == 1:
29              z = (z * g) % n  # z := (z * g) mod n
30
31      return z
32
33
34  def main():
35      n = 27803
36      g = 5
37
38      # Server's private key (random, kept secret)
39      y = 17555
40
41      # Compute server's public value: B = g^b mod p
42      B = square_and_multiply(g, y, n)
43
44      print("Diffie-Hellman Key Exchange - Server")
45      print(f"Generator g: {g}")
46      print(f"Prime p: {n}")
47      print(f"Server private key b: {y}")
48      print(f"Server public value B = g^b mod p: {B}")
49
50      # Set up TCP server
51      host = "localhost"
52      port = 12345
53
```

```
54      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
        server_socket:
55          server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
            1)
56          server_socket.bind((host, port))
57          server_socket.listen(1)
58
59          print(f"\nServer listening on {host}:{port}")
60          print("Waiting for client connection...")
61
62          conn, addr = server_socket.accept()
63          with conn:
64              print(f"Connected by {addr}")
65
66              # Receive client's public value A
67              data = conn.recv(4096)
68              A = int(data.decode("utf-8"))
69              print(f"\nReceived client's public value A: {A}")
70
71              # Send server's public value B to client
72              conn.sendall(str(B).encode("utf-8"))
73              print(f"Sent server's public value B: {B}")
74
75              # Compute shared session key: K = A^b mod p
76              shared_key = square_and_multiply(A, y, n)
77
78              print("\n")
79              print("KEY EXCHANGE COMPLETE")
80              print(f"Shared session key K = A^b mod p:")
81              print(f"{shared_key}")
82
83
84  if __name__ == "__main__":
85      main()
```

Lösung 3.1.2 Lösung 1

1. 11134
2. 6998
3. 10141
4. 27008
5. 16185

2. Aufgabe 3.2

Lösung 3.2 Lösung 2

An dem Punkt, an dem Bob seinen öffentlichen Schlüssel an Alice schickt, kann die dritte Person sitzen, um die man-in-the-middle attack durchzuführen. Dabei wird der echte öffentliche Schlüssel K_B+ abgefangen. Ein anderer öffentlicher Schlüssel der dritten Person K_m wird dann an Alice geschickt. Diese erstellt den gemeinsamen Schlüssel K_{ab} und schickt diesen zurück. Dieser wird von der dritten Person entschlüsselt mit ihrem privaten Schlüssel (da sie ja ihren öffentlichen Schlüssel an Alice geschickt hat). Danach schickt die dritte Person Schlüssel weiter, den Bob dann mit seinem privaten Schlüssel entschlüsseln kann, da die dritte Person ihn mit seinem öffentlichen Schlüssel verschlüsselt hat. Nun hat die dritte Person den privaten Schlüssel für die gemeinsame Kommunikation und weder Bob noch Alice wissen dies.

3. Aufgabe 3.3

Lösung 3.3 Lösung 3

Der KDC kennt in dem System alle öffentlichen Schlüssel aller Teilnehmer(P_A, P_B, ...). Jeder Teilnehmer kennt nur seinen eigenen privaten Schlüssel(S_A, S_B). Alle Teilnehmer kennen den öffentlichen Schlüssel der KDC.

Im folgenen wird eine Kommunikation zwischen Teilnehmer A und B dargestellt:

1. A fragt P_B von der KDC an. Die KDC signiert den Schlüssel mit seinem eigenen öffentlichen Schlüssel P_{KDC}.
2. B tut nun das selbe für den öffentlichen Schlüssel von A.
3. Nun führen A und B eine Challenge-Response aus:
   - A schickt eine Nonce N_A an B, die mit dem öffentlichen Schlüssel von B verschlüsselt worden ist.
   - B entschlüsselt N_A und verschlüsselt eine eigene Nonce, sowie die Nonce von A mit dem öffentlichen Schlüssel von A und schickt diesen an A.
   - Nun entschlüsselt A die Nonces, was A zeigt, dass die andere Seite tatsächlich B ist und schickt nun nur noch N_B an B, die mit dem öffentlichen Schlüssel von B verschlüsselt worden ist, um zu beweisen, dass A auch tatsächlich A ist.
4. (Optional) Um weitere Kommunikation weiterlaufen zu lassen, ohne die Authentification zu wiederholen, könnten sich A und B auf einen Sessionkey einigen (bspw. ein Hash aus den Nonces), die dann als gemeinsamer Schlüssel genutzt werden kann, um die Identität des anderne zu bestätigen.