Cryptography and Network Security

CBER 704

Project Report

Diffie-Hellman Key Exchange

Submitted to: Sk Md Mizanur Rahman, Ph.D.

Student Name	ID Number
Rajan Boudel	301365245
Krisuv Bohara	301274636
Oscar Cordoba	301214625
Fasoranti Tioluwani	301335271

Contents

Table of Figures

Introduction

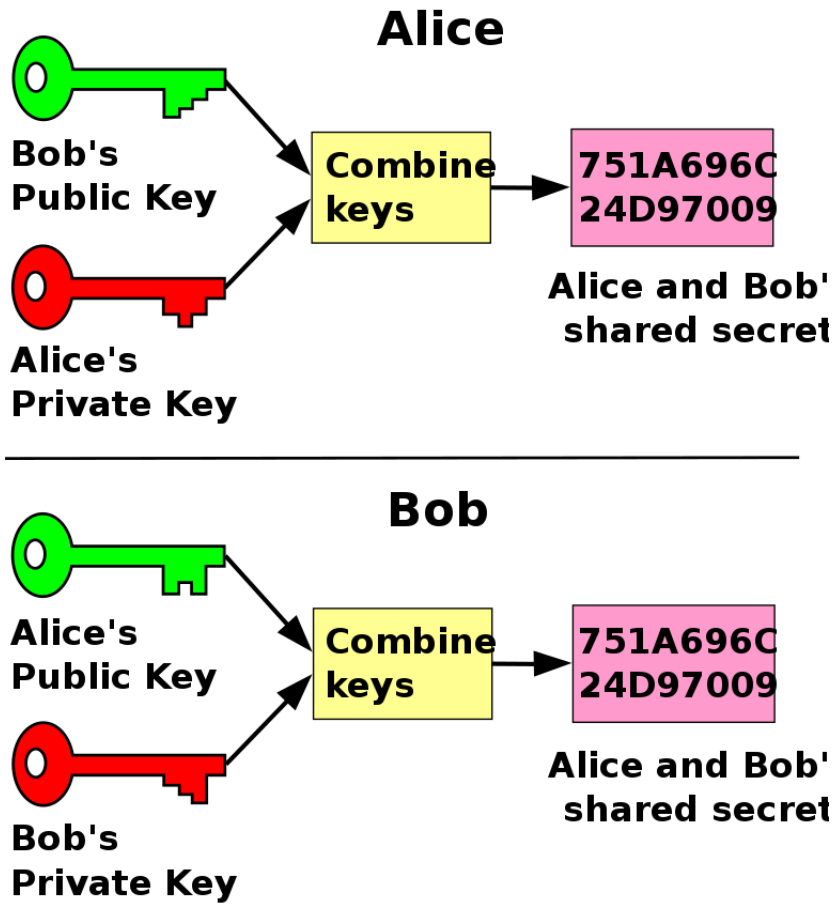

Figure 1 Diffie-Hellman Key Exchange

A cryptographic protocol called Diffie-Hellman key exchange (DHKE) allows two parties to create a shared secret key over an unsecure communication channel without having to know each other's private keys beforehand. This technique, which was independently proposed in 1976 by Whitfield Diffie and Martin Hellman, transformed the field of cryptography by presenting a novel method of key exchange that did not require a pre-shared secret. This implies that the shared secret key required for secure communication cannot be discovered by an eavesdropper, even if they manage to intercept every conversation between the two parties. (Davidgothberg, 2006)

Background

Many people held the view that it was impossible to exchange keys securely over an insecure channel prior to the development of the DHKE. This is due to the inherent insecurity of any technique that depends on transmitting the secret key over the channel—anyone could intercept and steal the key.

The discrete logarithm problem is a mathematical puzzle that the DHKE uses to get around this restriction. It would take an unworkable amount of time to break the DHKE, even with the most powerful computers, because this problem is so hard to solve.

Key Concept:

- **Public-Key Cryptography**

The foundation of Diffie-Hellman is public-key cryptography. Every participant in this paradigm possesses two keys: a private key that needs to be kept private and a public key that can be shared freely.

- **Shared Secret key Creation:**

Facilitating the cooperative generation of a shared secret key between two parties over an unreliable communication channel is the main objective of the Diffie-Hellman key exchange.

- **Problem with Discrete Logarithm:**

The discrete logarithm problem's difficulty determines Diffie-Hellman's security. Even if an eavesdropper were to notice the exchanged public keys, it would be difficult for them to deduce the shared secret due to the computationally demanding nature of the challenge.

Core Principles of DHKE Protocol

To achieve secure key establishment over an insecure communication channel, the Diffie-Hellman Key Exchange (DHKE) is based on several fundamental principles.

Public Key Calculation: Using their private key and a mutually agreed-upon mathematical function, each participant publishes their public key. Through the unsecure channel, this public key is freely shared.

Selecting Private Keys: A private key is a secret number that is generated at random for each participant. Throughout the key exchange procedure, this private key is kept private.

Creating Shared Secret: Every participant computes a shared secret key using their own private key and the public key of the other party. This shared secret key cannot be discovered by an eavesdropper and is the same for both parties.

Working Of DHKE Protocol

DHKE operates through the discrete logarithm problem, a mathematical function. It is thought that even the most powerful computers will have great difficulty solving this problem. Consequently, even in the event a third party was to intercept the two parties' conversation, they would be unable to determine the shared secret key. (Yao & Zhao, 03 December 2013)

- An outline of the DHKE protocol is provided below:
- A public group (g) and a prime number (p) are agreed upon by Alice and Bob.
- Alice chooses a random private key (a) and computes public key (A = g^a mod p)
- Bob chooses a random private key (b) and computes public key (B = g^b mod p)
- Alice sends her public key (A) to Bob.
- Bob sends his public key (B) to Alice.
- Alice calculates the shared secret key by using K = B^a mod p
- Bob calculates the shared secret key by using K = A^b mod p
- With their shared secret key (K), Alice and Bob can now encrypt and decrypt messages.

Applications

There are numerous applications for the adaptable DHKE protocol. Among the most popular applications are:

- The cryptographic protocol called TLS (Transport Layer Security) is used to protect web traffic. The data that is being transmitted between the client and the server is encrypted and decrypted using a shared secret key that is established using the DHKE.
- Secure Shell (SSH): To establish a connection with a distant computer, utilize the secure remote access protocol, or SSH. To authenticate users and encrypt data being transmitted between the client and the server, it uses the DHKE to create a shared secret key.
- IPsec, or Internet Protocol Security, is a security protocol used to protect IP communications. To encrypt and decrypt data being transferred between two hosts, it can use the DHKE to create a shared secret key.

Security Considerations:

- **Small Prime Numbers**: DHKE vulnerabilities may arise from the use of small prime numbers (p). For increased security, using big prime numbers is advised.
- **Participants with malicious intent**: An active attacker may attempt to pass for one of the participants to obstruct the exchange of keys. Implementations of DHKE ought to include safeguards against these kinds of attacks.
- **Key Compromise**: All communication using a participant's private key becomes insecure if the shared secret key is also compromised. To avoid key compromises, proper key management procedures are essential. (Zhao, 2010)

Benefits of DHKE

A cryptographic technique for safely transferring cryptographic keys over a public channel is Diffie-Hellman Key Exchange (DHKE). It is a commonly used protocol in many different applications, such as secure file sharing, secure authentication, and secure internet communication. These are a few of the main advantages of DHKE:

- **Efficiency:** Two parties can quickly establish a shared secret key with the help of the DHKE protocol, which is comparatively efficient. For real-time applications that demand quick key exchange, this is essential.
- **Security:** Because DHKE relies on the discrete logarithm problem, which is thought to be very challenging for even the most powerful computers to solve, it is regarded as one of the most reliable cryptographic protocols. Because of this, it is extremely impervious to man-in-the-middle and eavesdropping attacks.
- **Key Agility**: DHKE makes it possible for new shared secret keys to be exchanged for every session, providing ongoing defense against potential hacks. This improves security overall by preventing the use of compromised keys for additional communication.
- **Scalability**: DHKE is appropriate for applications involving multiple parties because it can accommodate many participants without experiencing appreciable performance degradation.
- **Standardization**: Because DHKE has been around for a while and is well-established, it is compatible with a variety of platforms and systems and is widely adopted.

Drawbacks of DHKE

A popular cryptographic protocol for safely transferring cryptographic keys over a public channel is Diffie-Hellman key exchange (DHKE). DHKE has certain disadvantages that should be considered when implementing it in different applications, despite its widespread adoption.

Pre-computation Attacks: An attacker may pre-compute the shared secret key for a given set of public keys using DHKE, making it susceptible to such attacks. The use of the discrete logarithm problem, which can be effectively solved for sets of prime numbers, is the source of this vulnerability. It is important to use large prime numbers and generate new group parameters on a regular basis to reduce this risk.

Key Size Growth: The size of the prime numbers used in DHKE determines its security. The required key size grows along with the security requirements, which can limit performance and storage, especially on devices with limited resources.

Man-in-the-Middle Attacks: If the parties do not authenticate one another, DHKE is not intrinsically resistant to man-in-the-middle attacks. This implies that the attacker can eavesdrop or control the conversation by intercepting the communication and pretending to be one of the parties. DHKE is frequently used in conjunction with other authentication techniques, like digital signatures or out-of-band communication, to thwart these attacks.

Complexity: Especially for devices with limited resources, the implementation of DHKE can be computationally demanding and complex due to the mathematical calculations and group operations involved. This leads to overhead and potential security issues.

Problems with Huge Groups: DHKE implementation can be difficult when dealing with very large groups, even though it's necessary for improved security. This is a result of the calculations' increased computational complexity.

Coding Demonstration

```python
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def is_primitive_root(g, p):
    for i in range(2, int((p - 1)**0.5) + 1):
        if pow(g, (p - 1) // i, p) == 1:
            return False
    return True

def find_primitive_root(p):
    for g in range(2, p):
        if is_primitive_root(g, p):
            return g
    return None

def diffie_hellman_key_exchange(p, g, private_key):
    public_key = pow(g, private_key, p)
    return public_key

def xor_encrypt_decrypt(message, key):
    # Initialize an empty list to store the result
    encrypted_message = []
    # Iterate through each character in the message
    for char in message:
        # XOR the character with the key and append the result to the list
        encrypted_char = char ^ key
        encrypted_message.append(encrypted_char)
    # Return the list of XORed characters
    return encrypted_message
```

Figure 2 Code Part 1

- **def is_prime**

  Function that basically checks whether the input is a prime number. It will return true if it has no divisor other than 1 and false in case the statement is incorrect.

- **def if_primitive_root**

  Function basically checks if ‘g’ is a primitive root module to ‘p’ which is a prime number.

- **def find_primitive_root**

  Function is responsible for finding a primitive root module for a given prime number. It iterates through possible value of ‘g’ and returns the first value.

- **def diffie_hellman_key_exchange**

  Function computes the public key in Diffie-Hellman key exchange. It takes the prime number ‘p’ a base ‘g’ and a private key. This function will return the public key by using modular exponentiation.

- **def xor_encrypt_decrypt**

  This function is responsible for performing the XOR encryption or Decryption on the input message using the generated key. It passes through each character in the input message. The XORs will store the result in a list and at last return it.

Main Function

1. Part 1

```python
def main():
    print("Diffie-Hellman Key Exchange and XOR Encryption/Decryption Program")

    # User input for prime number
    p = int(input("Enter a prime number (p): "))

    # Check if p is prime
    if not is_prime(p):
        print(f"{p} is not a prime number. Please enter a prime number.")
        return

    # Find a primitive root modulo p
    primitive_root = find_primitive_root(p)

    if primitive_root is not None:
        print(f"A primitive root modulo {p} is: {primitive_root}")
    else:
        print(f"No primitive root found for {p}. Choose a different prime.")
        return

    # User input for private keys
    alice_private_key = int(input("Enter Alice's private key: "))
    bob_private_key = int(input("Enter Bob's private key: "))

    # Key exchange
    alice_public_key = diffie_hellman_key_exchange(p, primitive_root, alice_private_key)
    bob_public_key = diffie_hellman_key_exchange(p, primitive_root, bob_private_key)

    # Display public keys
    print("\nKey Exchange:")
    print(f"Alice's public key: {alice_public_key}")
    print(f"Bob's public key: {bob_public_key}")
```

Figure 3 Main Function-1

- First the program runs and asks the user for a prime number then, the prime number is validated. Programs proceed only if the input is a prime number.
- After the prime number is validated, the programs run to find a primitive root module. After we get a primitive root module we proceed further.
- The Program now asks for alice's and bob's private key.
- Then the **def diffie_hellman_key_exchange** function will help to share the key and returns the public key by using modular exponentiation.

2. Part 2

```
    # Shared secret calculation
    alice_shared_secret = pow(bob_public_key, alice_private_key, p)
    bob_shared_secret = pow(alice_public_key, bob_private_key, p)

    # Display shared secrets
    print("\nShared Secrets:")
    print(f"Alice's shared secret: {alice_shared_secret}")
    print(f"Bob's shared secret: {bob_shared_secret}")

    # Encryption and Decryption using XOR
    message = input("\nEnter the message to be encrypted: ")

    # Convert the message to a list of integers
    message_int = [ord(char) for char in message]

    # Alice encrypts the message
    encrypted_message = xor_encrypt_decrypt(message_int, alice_shared_secret)

    # Display encrypted message
    print("\nEncryption:")
    print(f"Original message: {message}")
    print(f"Encrypted message: {encrypted_message}")

    # Bob decrypts the message
    decrypted_message = xor_encrypt_decrypt(encrypted_message, bob_shared_secret)

    # Convert the decrypted message back to characters
    decrypted_message_str = ''.join([chr(char) for char in decrypted_message])

    # Display decrypted message
    print("\nDecryption:")
    print(f"Decrypted message: {decrypted_message_str}")

if __name__ == "__main__":
    main()
```

Figure 4 Main Function-2

- After the system provides the public key, the system then proceeds to ask for a message to be encrypted.
- The **def xor_encrypt_decrypt** function will be responsible for encryption of message.
- It performs the XOR encryption or Decryption on the input message using the generated key. It passes through each character in the input message. The XORs will store the result in a list and at last return it.
- The message is then encrypted in a list.
- The message is then decrypted by joining the string.

Result

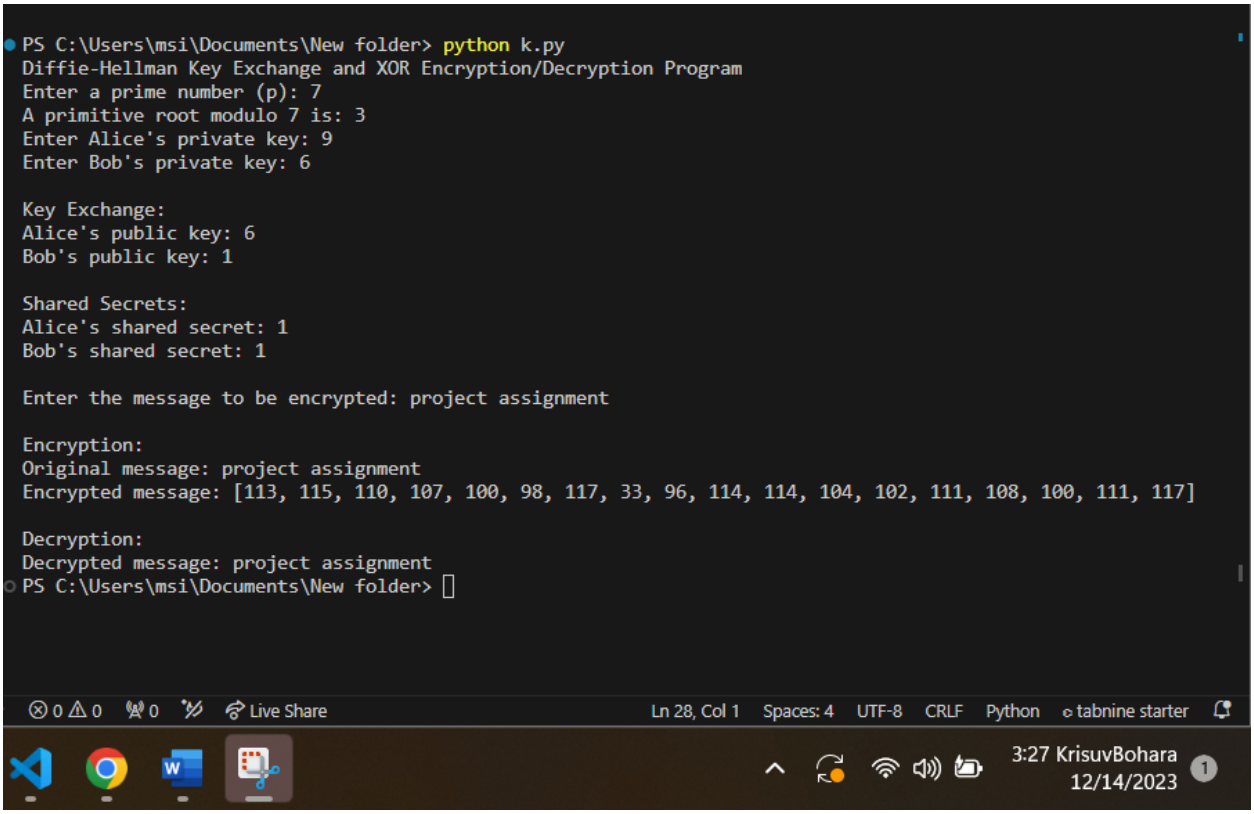

Figure 5 Program Result

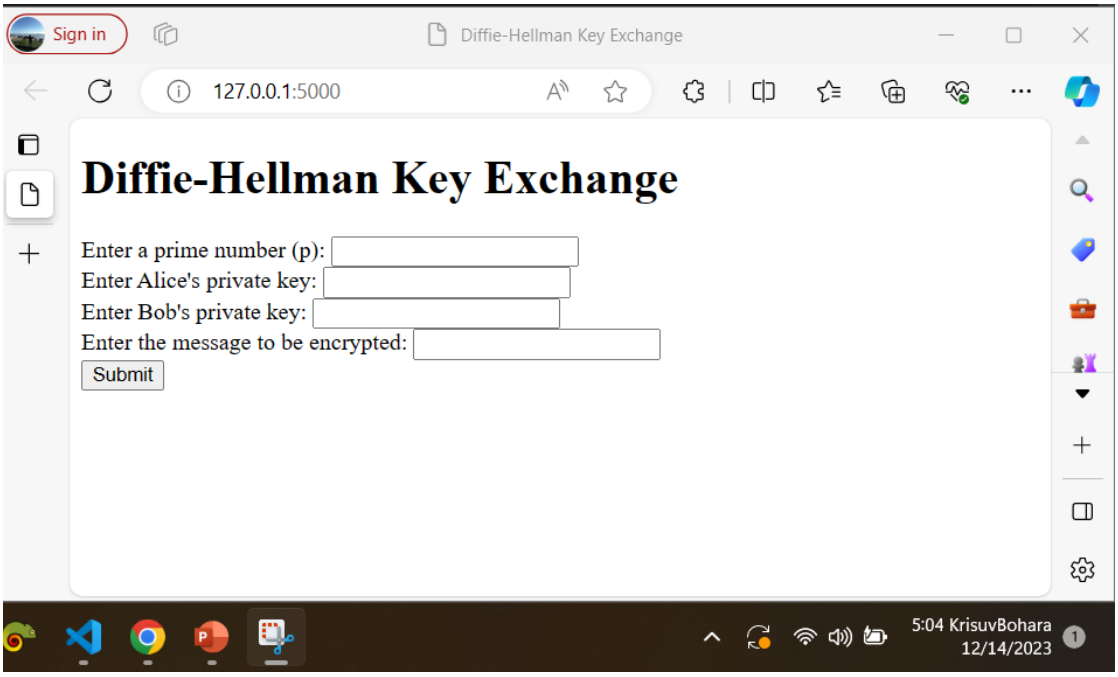

Figure 6 Web App-1

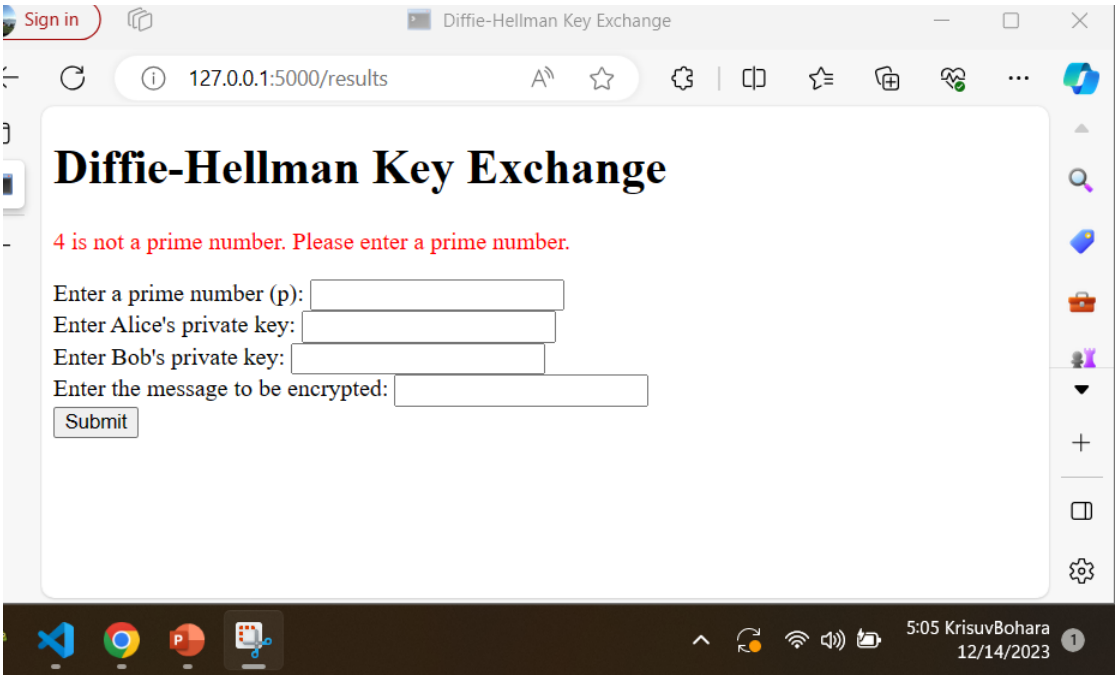

Figure 7 Web app 2

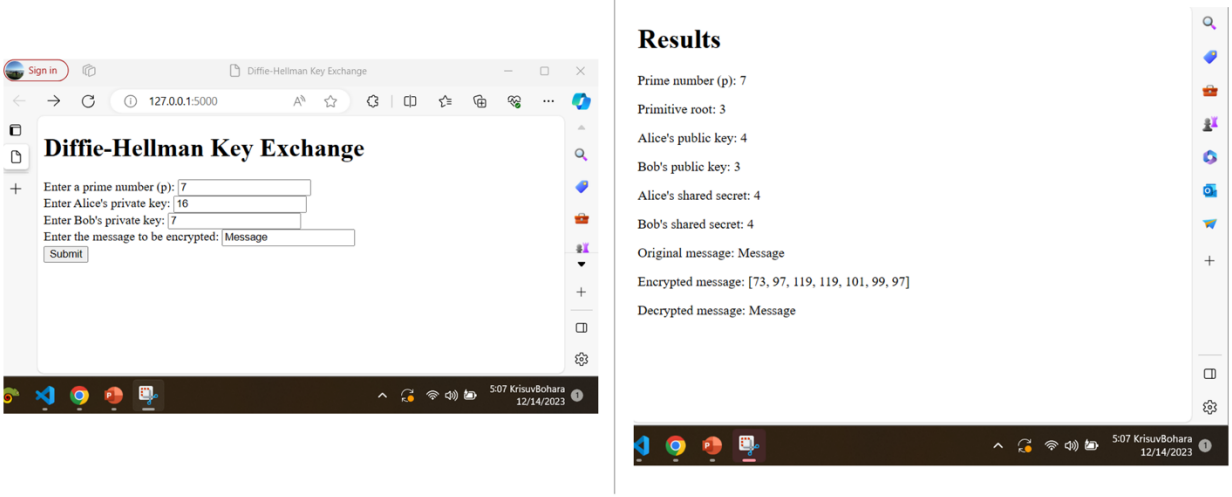

Figure 8 Web App 3

Conclusion

DHKE is a cornerstone of contemporary cryptography and necessary for safe communication and data protection in a variety of online activities because it combines efficiency, security, versatility, key agility, scalability, standardization, and PKI compatibility.

To conclude, the Diffie-Hellman Key Exchange (DHKE) is a cryptographic protocol that was the first to be widely used and revolutionized secure communication over untrusted networks. Its novel methods of public-key cryptography and key establishment have served as the basis for a wide range of safe communication protocols.

Bibliography

Davidgothberg. (2006, 10 12). *Davidgothberg*.

Yao, A. C.-C., & Zhao, Y. (03 December 2013). *Privacy-Preserving Authenticated Key-Exchange Over Internet.* IEEE.

Zhao, A. C. (2010). *Applied Cryptography and Network Security.* Beijing, China: ACNS.