Using Digital Certificate to Prevent Man-in-the Middle Attack in TLS Protocol

Zheng Wu
1613261

***Abstract*—In the large field of computer system security, protecting communication is a very significant topic. With the rapid development of modern cryptography, many excellent symmetric encryption algorithms have proposed, but the distribution of secret keys has always been a problem. There are asymmetric encryption algorithms and key exchange algorithms to solve the distribution problem, and even digital signature solutions to verify identity. However, when it comes to transferring public keys, these methods may still be attacked by man-in-the-middle. In the report, an AKE protocol about TSL1.3 based on the ECDSA and ECDH was implemented, then a scheme of digital certificate was designed in the protocol to prevent the man-in-the-middle attack (MitM).**

Keywords—TLS, ECDH, ECDSA, MitM, Digital certificate

I. Introduction

. With the development of computer technology and the popularity of the Internet, information communication takes place every moment, and most of them involve sensitive information. In order to protect the security of information communication, people use various cryptographic protocols. Among them, the AKE protocol combines authentication and key exchange protocols, which is the most commonly used security protocol in network communications. Several international standards applying the AKE protocol play significant role in different field, including communication standards (e.g. WBAN and Bluetooth Specification), security standards (e.g. Transport Layer Security, TLS). These international standards are established by the underlying security protocols and different encryption algorithms.

With the rapid development of modern cryptography, many excellent symmetric encryption algorithms have proposed, but the distribution of secret keys has always been a problem. There are asymmetric encryption algorithms and key exchange algorithms to solve the distribution problem, and even digital signature solutions to verify identity. However, when it comes to transferring public keys, these methods may still be attacked by man-in-the-middle.

In this report, we firstly focus on stimulating the implementation of TLS handshake protocol based on ECDH and ECDSA, of which the final version has been published in August 2018 [1], then a man-in-the-middle attack (MitM) was used to attack the initial implementation. In order to prevent the attack, a scheme of digital certificate was designed in the protocol to prevent the MitM attack.

II. Literature Survey

A. TLS protocol

TLS stands for Transport Layer Security and is the successor of SSL (Secure Socket Layer)[2]. TLS provides a secure channel between two communicating peers, such as web browsers and servers. This protocol is divided into two layers: the TLS recording protocol and the TLS handshake protocol. The recording protocol is used to encapsulate high-level protocols, such as the TCP handshake protocol. The TLS handshake protocol enables clients and servers to authenticate each other and negotiate cryptographic modes and parameters. In this report, we only implement what related to the TLS handshake protocol, and set the digital signature algorithm as ECDSA and the key exchange algorithm as ECDH.

B. ECDH

.ECDH stands for 'Elliptic Curve Diffie–Hellman Key Exchange', which is a anonymous key agreement scheme. It allows two parties to negotiate a safe shared secrete even in an insecure channel. ECDH is similar to the classical Diffie-Hellman Key Exchange algorithms, but it is based on the ECDLP problem from ECC(Elliptic Curve Cryptosystem) instead of modular exponentiations. The process of the algorithm is as follows:

1. **Alice** generates a **random** ECC key pair: {**alicePrivKey**, **alicePubKey** = alicePrivKey * G}
2. **Bob** generates a **random** ECC key pair: {**bobPrivKey**, **bobPubKey** = bobPrivKey * G}
3. Alice and Bob **exchange their public keys** through the insecure channel (e.g. over Internet)
4. **Alice** calculates **sharedKey** = bobPubKey * alicePrivKey
5. **Bob** calculates **sharedKey** = alicePubKey * bobPrivKey
6. Now both **Alice** and **Bob** have the same **sharedKey** == bobPubKey * alicePrivKey == alicePubKey * bobPrivKey

C. ECDSA

ECDSA stands for 'Elliptic Curve Digital Signature Algorithms'[4], which is used to create a digital signature of data in order to verify the identity of data sender. ECDSA is very secure because of the trap door function, for point multiplication **R**=K***P,** if someone knew **R** and **P**, there is no possibility to figure out the value of K, since there is no 'point division'. Therefore, it is no possible to find the private keys and there is no way of faking a signature without getting the private key.

D. MitM Attack

A man-in-the-middle attack means that an attacker intercepts the communication between the two parties and secretly eavesdrops or tampers with the communication data between the two [5], so that the two parties of the communication think that they are talking directly with each other through a private connection, but in fact the entire session of two parties are fully controlled by the attackers. There are several kinds of MitM attacks, such as DNS spoofing, Https spoofing and SSL hijacking [6].

In this report, a MitM (Mallet) wanted to deceive the client(Alice). He obtained the Alice's authority and replaced the public key sent by the server(Bob) with his own public key. At this time, Alice possessed the Mallet's public key , but thought it was Bob's public key. Therefore, Mallet can pretend to be Bob, using his private key to

make a digital signature, and then Alice will use the fake public key to verify.

III. Design

A. The initial implementation and Attack

The initial implementation of TLS handshake protocol is based on ECDSA and ECDH, the protocol description is shown below:

1. Before the TCP connection, client (Alice) could take advantages the free time to generate and calculate the public, private keys and signatures, then encapsulating the string and public keys as M1, signatures as M3.
2. Establishing TCP connection.
3. Client send M1 to server(Bob); server receives M1 and send M2(string and public keys from server).
4. Client receives M2 and send M3; servers receives M3 and verifies the signatures, and send M4.
5. Client receives M4 and verifies the signature from Server, if it is true, then the two sides will negotiate and a new shared secret by the exchange information after closing TCP

For digital signatures, when Alice receives the signature with data from Bob, she will use the public key sent by Bob before to decrypt signature, and made a comparison between data's hash and decrypted digest of the signature, if the two are the same, it can be proved that data and signature have not been tampered and the sender is Bob, seeing Figure1.

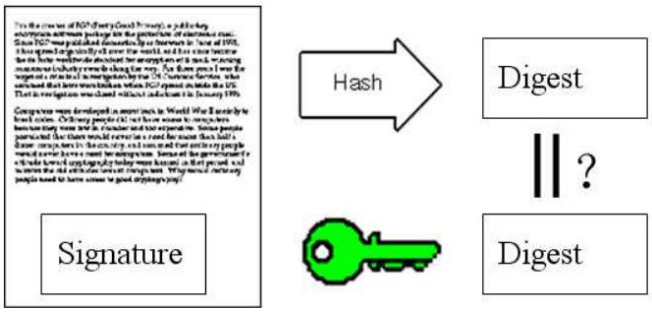


Figure 1

The main reason is showed blow:

1. If someone changes the data, after Alice decrypts the signature, she will find that the two are inconsistent

2. If someone replaces the signature, Alice can only solve a string of garbled characters using Bob's public key, which is obviously inconsistent with the data.

3. If someone attempts to modify the data, and then make the modified data into a signature, so that Alice's comparison cannot find inconsistencies; but once the signature is unlocked, it is impossible to regenerate Bob's signature because there is no Bob's private key.

As mentioned above, digital signatures can verify the source of data to a certain extent. The reason is to a certain extent, because this method may still be attacked by a MitM. Once the release of the public key is involved, the receiver may receive the fake public key from the MitM and perform wrong authentication. Therefore, to determine the identity of the other party, there must be a source of trust, otherwise, no matter how many processes are just transferring the problem, rather than really solving the problem.

B. Proposed solution.

In the part, a digital certificate could be used to prevent MitM attacks". The digital certificate is actually a public key + signature, issued by a third-party CA (certification authority) organization. It is used to establish secure communication between two parties who are unknown to each other or have lack of trust. Referring a trusted third party is a feasible solution to end the trust cycle. The creation and authentication processes of certificate are shown in Figure2 and Figure3.

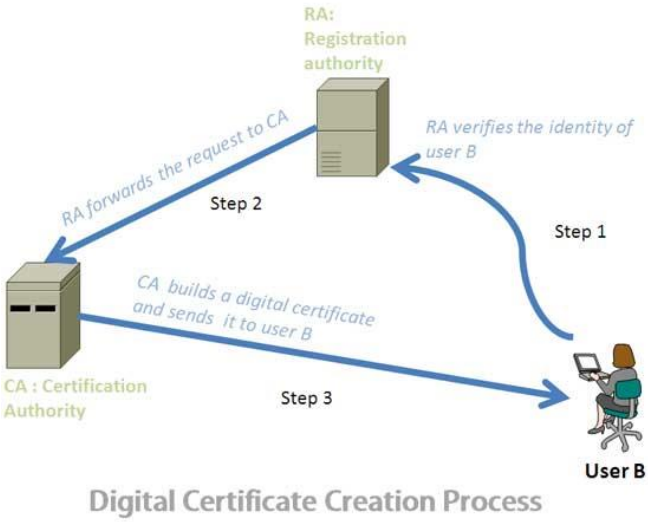


Figure 2 [7]

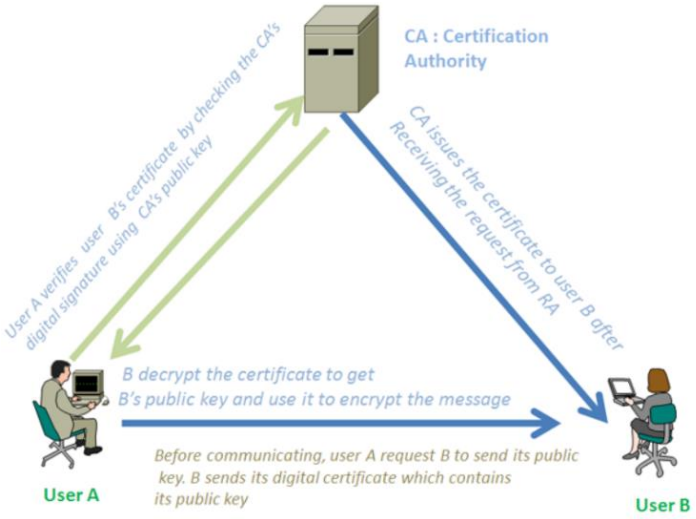


Figure 3 Secure communication with digital certificate [7]

In the process of my design, I use a **.py** file to represent the RA and CA, and in order to highlight the relationship between the client, the server and the CA organization, I omitted the interaction process between CA and RA, the protocol process that I designed is as follows:

- Bob (Server) sends a request to CA, then CA verify Bob's identity.
- Once the identity information is satisfied, CA creates the digital certificate using B's public key and other important information and send to Bob, it can be noted

CA signs the certificate with itself private key to ensure authenticity.

- Bob receives the digital certificate, when Bob send the digital signature to Alice, the certificate was attached to the signature. (seeing Figure 4)

- Alice receives the signature and certificate and verifies the digital certificate using CA's public key, it also can be noted the CA's public key is embedded in client (Alice).
- If the authentication is successful, it proves the public key of Bob would be true, and they will continue the next function.

IV. Implementation and Testing

In this part, some implemented codes about design, experimental platform and some results for testing will be shown.

A. *Some significant implementations*

a. Generate Digital Certificate

```
def generateDC(info):
    hstringca = info
    md5b = md5()
    md5b.update(hstringca.encode())
    hCA = md5b.hexdigest()[8:-8]
    hCA = int(hCA, 16)
    dc = sign(hCA, (256, int(SKca)))
    return dc
```

b. Verify identity and send DC

```
if (getArequest&Severidentity):
    print('Vertify identity completed, satisfied ')
```

```
        while (token==0):
            print(' DC will be sent to Server')
            sock.send(DC.encode())
            token = 1
            sock.close()
```

c. CA certificate in Client (the public key of CA*)*

```
# CA certificate, mainly is the public key of CA
PKCAx= '105656888354198245866251604935511391667930046043257259
PKCAy= '365827483828138892417842319696894271099650969816764121
```

d. Client verify certificate and signature

```
if verify(hdc,dc,(256,(int(PKCAx),int(PKCAy))))==True:
    print('The digital certificate is qualified')
    print('The public key of Server is valid ')
    if verify(hb,sb,(256,(int(PKbx),int(PKby))))==True:

        print('signature of server is valid')
        # 5   caculating the shared secret key
        Ka = mul(c_p, c_q, c_n, PKb, SKa)
        print('the shared secret is', Ka)
    else:
        print('signature of server is invalid, protocol fails')

else:
    print('Digital certificate of server is invalid')
    print('The public key of Server is insecure, protocol fails')
```

B. *Experimntal platform and testing results*

The experimental platform is a windows platform with python 3.6, in addition, the python IDE Pycharm was recommended to use.

Testing: First run the server.py,

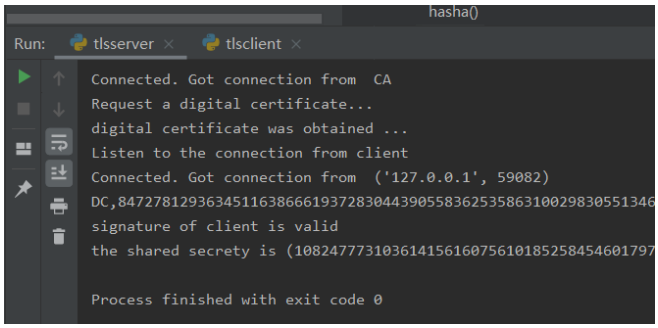
```
tlsserver
D:\anaconda\envs\pixel\python.exe C:/Users/007/
Begin
Listen to the connection from  CA.
```

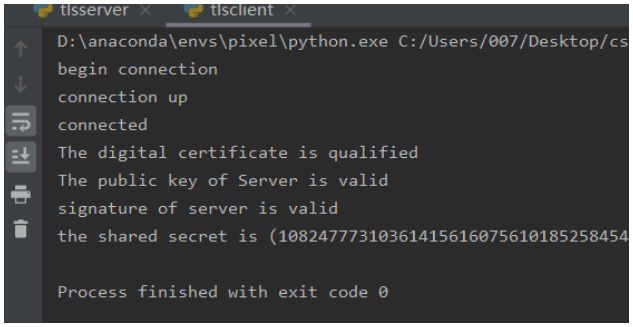
then run the CA.py to get the digital certificate. The results of CA and server are shown below.

```
tlsserver   CA
D:\anaconda\envs\pixel\python.exe C:/Users/007/Desktop/cse
Vertify identity completed, satisfied
DC, 847278129363451163866619372830443905583625358631002983
begin connection
 DC will be sent to Server
```

```
tlsserver   CA
D:\anaconda\envs\pixel\python.exe C:/Users/00
Begin
Listen to the connection from  CA.
Connected. Got connection from  CA
Request a digital certificate...
digital certificate was obtained ...
```

Finally, run the client.py to be interactive with server. The server and clients, that results in something like the figure below.





We can find both server and client could negotiate the shared secret in the end, and next the Symmetric encryption algorithm will play a significant role in communication process of two parties.

V. Learning Points

This part will introduce what I have learned in this project. First of all, we needed to find a topic to study by ourselves. Compared with a topic provided by teacher, we could develop our thinking ability and knowledge application ability, at the same time reduce the limitations and increase our research enthusiasm. Secondly, through this project, I understood and deepened my understanding of many knowledge points, and had some in-depth understanding of some network security protocols, especially the TLS protocol and the https protocol. In addition, for some ECC-based algorithms, such as the classic ECDH and ECDSA, through directly operating the code and looking into the original code of some functions. I felt the subtlety of these algorithms. By simulating the process of a digital certificate by myself, I understood the importance of identification in network communication. As a result, I knew that cryptography is only a small part of computer security. Even a site certified by a formal institution does not mean this can be trusted, it only shows that the data transmission is safe. Technology can never really protect users, the most important thing is to improve personal safety awareness.

VI. Conclusion

In conclusion, using digital certificate, issued by a trusted third party, could effectively prevent man-in-the-middle attacks. In fact, most formal websites use the HTTPS protocol now, which adds an TLS security layer between the HTTP protocol and the TCP protocol to ensure the secure transmission of data. As a result, traditional man-in-the-middle attacks have little room for survival, and the means of attack can only be changed from technical flaws to fraud. For example, many browsers on the Internet contain some irregular certification authority certificates, anyone can apply for a certificate, which are likely to cause a security risk.

References.

[1] Brian Jackson, “An overview of TLS1.3- faster and more secure,” June 2, 2020. [Online] Available: https://kinsta.com/blog/tls-1-3/

[2] Josh Fruhlinger, “How the SSL, TLS work “ Dec 4, 2018. [Online] Available: https://www.csoonline.com/article/3246212/what-is-ssl-tls-and-how-this-encryption-protocol-works.html

[3] .Svetlin Nacov. (2019, August) “EDCH Key Exchange”,[Online] Available: https://cryptobook.nakov.com/asymmetric-key-ciphers/ecdh-key-exchange

[4] Snifikino, “Understanding How ECDSA Protects Your Data,” (2015)[Online]Available:https://www.instructables.com/id/Understanding-how-ECDSA-protects-your-data/

[5] Dan Swinhoe,”What is a man-in-the-middle attack? How MitM attacks work and how to prevent them” (Feb 13, 2019). Available:. https://www.csoonline.com/article/3340117/what-is-a-man-in-the-middle-attack-how-mitm-attacks-work-and-how-to-prevent-them.html

[6] NortonLifeLock employee, “About man-in-the-middle attack,” Security Center, Norton . [Online] Available: https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle-attack.html

[7] Wing. “What is Digital Certificate”, [Online] Available: https://securitywing.com/digital-certificate-how-works/#respond