ELTE - Faculty of Informatics

Subject: Network and System Security

Lecturer: Mohammed B. M. Kamel


# Encryption Algorithm for TCP Session Hijacking

Presentation report

Date of submission: 2nd of May 2022

Ali Aqeel Zafar (AAFZDE)

Friedrich Ostertag (D7KQBF)

# Abstract

Today the Transmission Control Protocol (TCP) is widely in use as transport layer for transmission of information in computer networks. Unfortunately, multiple network attacks on the TCP protocol are possible due to flaws in its design. Main flaws of standard TCP are the lack of (continuous) authentication and the unencrypted data transmission.
In this report, we will go through the paper "Encryption Algorithm for TCP Session Hijacking" (Chen, M. et al. 2020) which proposes a way to verify integrity in TCP sessions and prevent session hijacking. This is achieved using RSA-based encryption, Diffie-Hellman key exchange. We will introduce the paper, explain the proposed model in detail with an example and finish with a conclusion.

# Introduction

In their paper, Chen et al. talk about the TCP's three-way handshake that establishes any connection between the source (client) and the destination (server). The three-way handshake provides synchronization in serial numbers, which are sequence and acknowledgement numbers. With the use of those numbers, TCP can provide a stateful and reliable end-to-end transmission stream for bytes and prevent packet loss.

While TCP has all the functionalities, the paper focuses on some problems. The main problem, which the paper tries to deal with, is session hijacking. As the verification is only executed once, at the beginning of a session, it is possible to hijack TCP sessions. Through TCP session hijacking, password sniffing and cookie hijacking could be done. Besides that, the data stream is transmitted in the form of plain text, meaning there is no encryption and authentication mechanism used. As a result, source identity and information cannot be reliably verified.

# Related work

The paper mentions a few concepts that other authors have tried to applied in the past to prevent session hijacking. These include using cookies for session authentication, configuring firewalls more strictly, using two factor authentication when establishing a connection or using a server-side reverse proxy. While cookies come with some design-related security flaws, firewalls can be difficult to set up, as most devices today have unlimited internet access. For the latter two points the paper doesn't list any disadvantages. Anyways, lets not go too much into detail here, as these approaches are not directly related to the new

model proposed in the paper. The authors use this point mainly to show the importance of securing TCP sessions.

## Contribution of the paper

The paper proposes an improved scheme of the TCP three-way handshake. RSA encryption and Diffie-Hellman Key exchange are used to provide identity verification and encryption in the TCP protocol. The authors claim to provide double-ended authentication to verify not only the client, but also the server. If they actually manage to do so will be discussed in the conclusion.

## Model Explanation

We came up with the following figure, to help explaining the model introduced in the paper. The red elements represent the addition of the authors to the standard TCP three-way-handshake. Also they introduce a trusted 3rd party (TTP), that distributes a pair of keys (S+, S-) to the client and server each. S+ will be later used for encryption, S- for decryption.

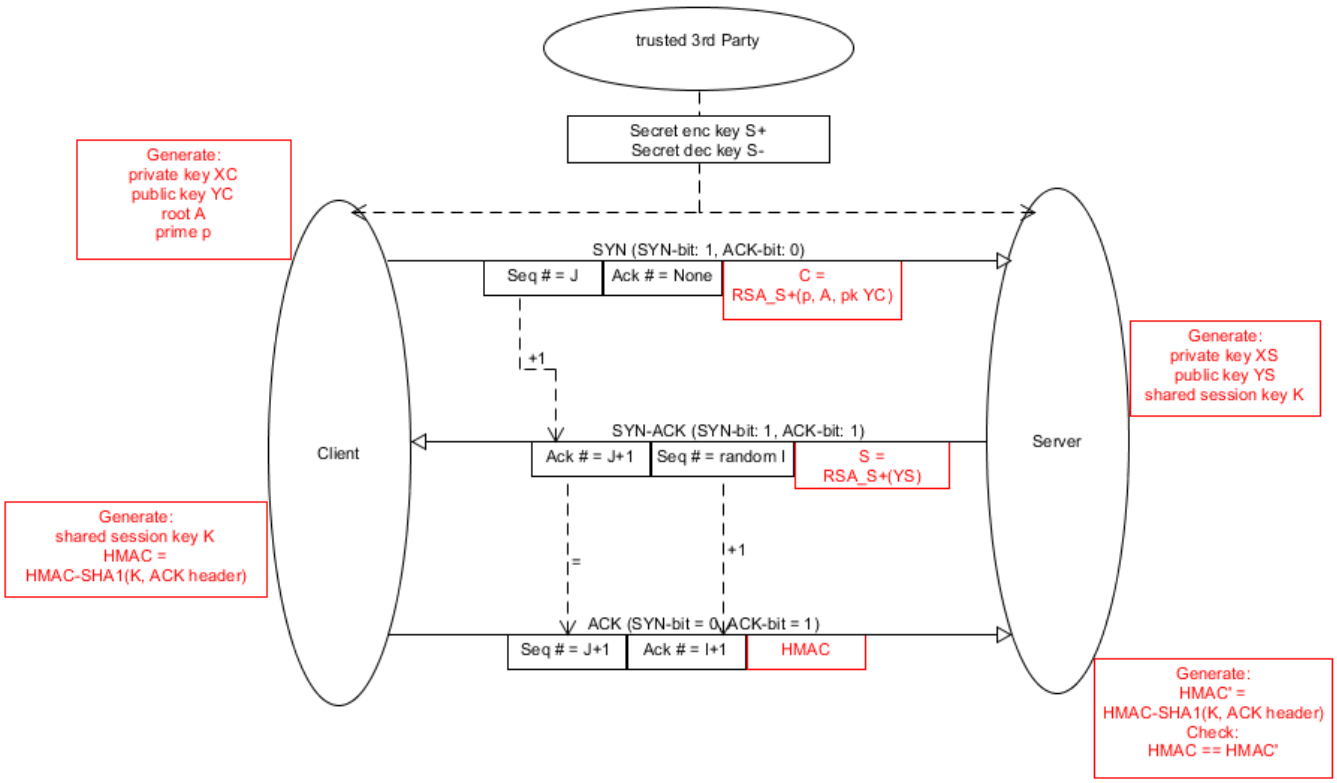


*Fig. 1: New TCP handshake model*

No we will go through the modification in each single phase of the handshake. For better understanding we use the same variable names as the paper.

## First Handshake-Packet:

Before the first packet (SYN) is sent from the client to the server, the client generates a large prime p, the root A and selects a private key XC (XC<P). According to DH, the client generates its public key YC = $A^{XC}$ (mod p). Then the client uses the RSA encryption with the key S+ to encrypt p, A and YC to the ciphertext. This ciphertext is included in the SYN-packet and sent to the server.

## Second Handshake-Packet:

The server receives the packet and by decrypting C by with S-, it gets p, A and YC. Then the server calculates its own private key XS and public key YS (YS = $A^{XS}$ (mod p)) and calculates the session key K = $YC^{XS}$ (mod p). Then the message S is generated by encrypting YS with RSA encryption using S+. The server then includes S in the ACK-SYN packet and sends it to the client.

## Third Handshake-Packet:

After receiving the packet, the client decrypts S using S- and retrieves YS. Then the client uses YS to calculate the session key K (K = $XC^{YS}$ (mod p)). Now the server and client have completed the DH-key-exchange and both possess the same key K.

The client now generates HMAC, which is a message authentication code (MAC) based on a key and a hashing algorithm. K is used as key, the header of the ACK-packet is used as message, SHA1 as hashing algorithm (HMAC = HMAC-SHA1 (K, ACK-header)). This HMAC value is added to the ACK-packet (the last packet of the three-way-handshake) and the packet is sent to the server.

Now the server calculates HMAC' by using HAMC-SHA1 the ACK-header and its own K. After that, the server can compare HAMC' and HAMC. If they are the same, the ACK-header AND K are the same and the connection is established. Since only the "real client" knows the correct K, the client's identity is verified by this. Every 100 (by authors proposed window value) data packets the client will repeat the computation of HMAC and send it to the server, who computes its own HMAC' and verifies their equality. This ensures a continuous authenticity. A large window size value will result in a good performance, but the security will be bad. While if the window size value is small, the performance will suffer, but the security will be high.

# Example / Experiment

In the chapter “Experiment”, the authors don’t really run any experiment. Rather they give an example of how their model works with numbers. We will go through their example and show problems, that occured in the paper.

The client generates p = 97, A = 5 and selects a private key XC = 36 (XC<P). The client generates its public key YC = $A^{XC}$ (mod p) = $5^{36}$ (mod 97) = 50. Then the client sends C = Enc_S+ (p, a, YC) to the server.

The server receives the packet and by decrypting C by with S-, it gets p, A and YC. Then the server calculates its own private key XS = 58 and public key YS = $A^{XS}$ (mod p) = $5^{58}$ (mod 97) = 44 and calculates the session key K = $YC^{XS}$ (mod p) = $50^{58}$ (mod 97) = 75. Then the server sends S = Enc_S+ (YS) to the client.

Now the client gets YS by decrypting S with the key S- and computes K = $XC^{YS}$ (mod p) = $36^{44}$ (mod 97) = 35. Then the client computes HMAC = HMAC-SHA1(K, ACK-header) and sends it to the server. The server computes HMAC’ = HMAC-SHA1(K, ACK-header) and checks, if HMAC and HMAC’ are the same, which is only the case for the same key K.

We noticed that the secret session key K, which is supposed to be used by the client and the server, is not being calculated to the same value. This means the HMAC will differ from HMAC' and the verification will fail. The reason of this issue was that the paper was not using the Diffie-Hellman Algorithm properly to calculate the session key K. We can only guess, what the authors were actually doing, as their result is correct, while the calculation is wrong and should not lead to a correct result.

The server’s K is calculated in the following way: K = $YC^{XS}$ = $A^{XC^{XS}}$ = $5^{36^{58}}$= 75

The client’s K should be calculated correctly in the following way (according to DH): $\mathrm{K} = \mathrm{YS}^{\mathrm{XC}} = \mathrm{A}^{\mathrm{XS}^{\mathrm{XC}}}$= $5^{44^{36}}$= 75. This way it would be K_client = $\mathrm{A}^{\mathrm{XS}^{\mathrm{XC}}}$ = 75 = $A^{XC^{XS}}$ = K_server. Both keys would have the same value.

However with the calculation of the paper K = $XC^{YS}$(mod p) the key would be 35, which is wrong. In the paper’s number example the calculation is done with K = $XC^{YS}$ = $50^{58}$ which leads to the correct result 75 but does not match the actual values for XC (= 36) and YS (= 44). Maybe the authors used the same calculation as for the server’s K again ($YC^{XS}$ $(mod\ p)$), which is not practically feasible, as the client does not possess XS.

# Conclusion

The authors conclude that the model they propose works fine and provides integrity to prevent session hijacking and allow identification of malicious packets. They propose further experiments and proofs to find the window size value, which balances security and runtime best.

In the end we would say that there were many flaws in the paper that we encountered. Variable names were changed and differently used, which resulted in confusion. The calculations provided in the paper contained mistakes and some parts were hard to understand due to sentences not being written in understandable English.

Still, the presented idea could provide value, when implemented correctly. Although it might not be as ideal, as the authors tell. TCP Session Hijacking is limited, but still possible between the verification of HMAC values and critical data can be retrieved within the small time bracket. Besides that a TTP server is required, which could lead to more problems when it comes to using this approach practically. Furthermore, the authors do not give any reason for why the window size should be 100. It is not clear, how much this approach will influence the performance, or if there even exists a window value which provides a practical trade-off and allows for fast enough AND secure transmission. Throughout the paper it was also not pointed out, how double authentication is being achieved. If the server was compromised, it doesn't need to proof the possession of K, as the verification step (HMAC == HMAC'?) is only executed by the server. A malicious server would simply accept any HMAC value from the client and establish the connection.

To conclude this report, we would say that yes, the idea could indeed provide value and improve the security of TCP. But before that, far more research and also practical experiments need to be undertaken and answers to the above questions need to be found.