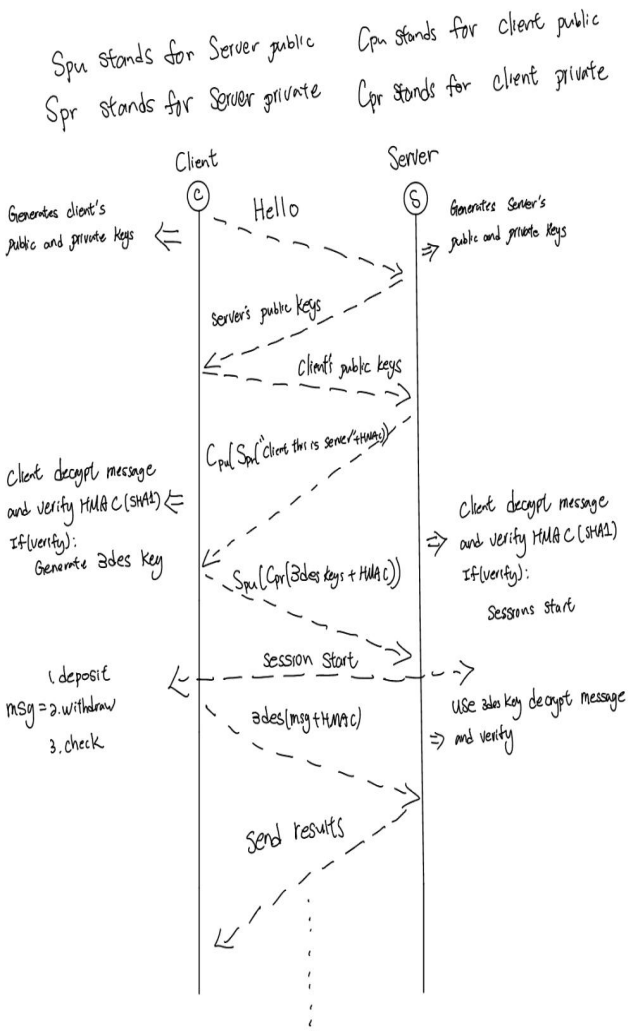
Nicholas Pardave

Cryptography and Network Security

8/8/2022

White Hat Report

For my report, I will explain the implementation of our server/client exchange and the component that I was responsible for, which was the HMAC implementation along with the SHA-1 implementation to correctly hash the given inputs.. My team consisted of Sean Hung and Samyuth Sagi.

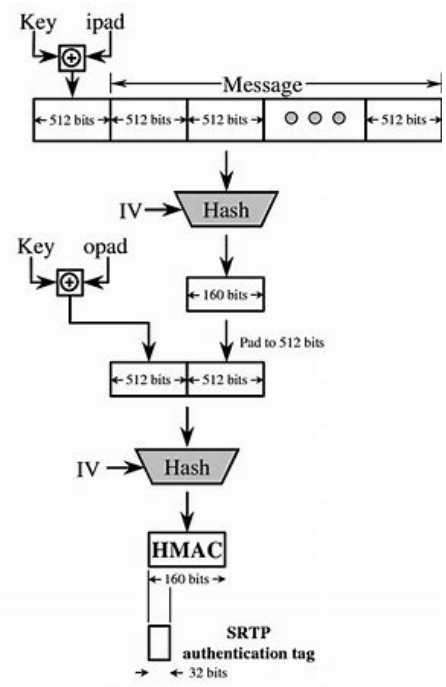


As an overview of what my team and I decided to do for an implementation of the project, we have provided an image that corresponds to the activity between the client and the server. To begin, both the server and the client generate their public and private keys. Next, the server and client open a TCP server in which the server and client will communicate after authentication. Then, the client sends “Hello” to the server. Once the server receives the message, the server sends

its public keys to the client. As a result, the client also sends their public key. Then, the server encrypts “Client this is server” and the HMAC value with their private key and then once more with the clients public key and sends the ciphertext to the client. The client then decrypts the message using their private key and verifies if the current message has been corrupted using HMAC with SHA-1 implemented. If the verification is successful, then the client generates a 3DES key. The client then sends over the 3DES key along with the HMAC and encrypts it with their private key and then with the server’s public key. Once the server receives the message, the server decrypts it with their private key and then with the public key of the client. Then, the server checks the HMAC the client sent for verification. If the verification is successful, then the session begins. Throughout the session, the client will encrypt their messages with 3DES and send it to the server where the server will decrypt the message with the 3DES key the server had previously received. At the same time, the server verifies if the message has been corrupted using HMAC and sends the results if the verification is successful. Thus, the server and client are able to communicate within a secure network between one another.

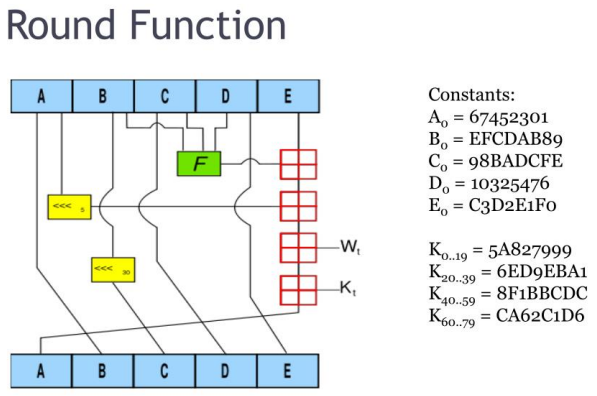
One of the major components that was necessary to get a secure network between the server and the client was the use of verification and ensuring that the message was not corrupted during the send process. The process to ensure that a message does not get corrupted is a Message Authentication Code(MAC) on the message. There are many varieties of MAC algorithms, but one of the algorithms we have much more confidence and knowledge on is Hash-based Message Authentication Code(HMAC). The HMAC algorithm requires a hashing function, which will be used to essentially shrink the message into a smaller chunk, which requires multiple steps in order to result with the HMAC value.

For the implementation of the HMAC, it followed the diagram shown on our right. Since we decided to use SHA-1 as our Hashing function, the size for each block of message must be 512-bits. Before we can make any changes to the original message, we must first pad the key to the size of 512 bits. We accomplish this by essentially adding bits of 0 to the front of the binary representation of the key. Once we have a key of size 512 bits, we will XOR the value with ipad, which is initially set and fixed. We then take the XOR result and add it to the front of our message block. The newly edited message gets passed through the hash function, or SHA-1 in our scenario, and returns a 160 bit MAC code. We then pad the MAC code by adding 0 to the front of the binary representation of the MAC code. Lastly, we XOR the padded key from earlier with opad, another initially set and fixed value. The XOR result will be added to the front of the padded MAC code and passed through SHA-1 to result with our HMAC value.

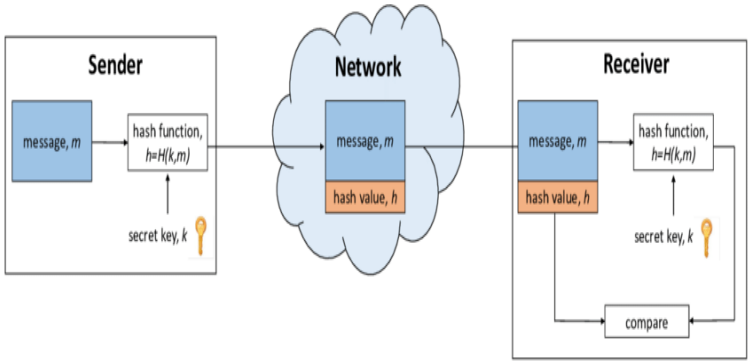


For the implementation of SHA-1, we followed closely to a round based function. Our first step was to loop over the characters of the message, turn them into ASCII values and turn

their values as 8-bit representations. Afterwards, we add 0s to the end of the bits until the length is 448 in mod 512 since we need 64 bits of space for the next step. Then, you must take the binary representation of the original length of the message and represent it as 64-bits. Then, the 64-bits is added to the end of the message. Next, I break down the message into 512-bit chunks and for each of the chunks, I break them down into 16 32-bit chunks. Now, we will extend the chunks to get 80 32-bit chunks by using the original 16 32-bit chunks and using the expression (chunk[i-3] XOR chunk[i-8] XOR chunk[i-14] XOR[i - 16]) and then rotate left 1 bit where $15 < i < 80$. Once we have the chunks that we need, we can proceed with the implementation of the round function, which is the foundation of creating the Hash value. The implementation of the round function is shown on the image above. Essentially for each chunk, we will set values A,B,C,D,E as H0,H1,H2,H3,H4, which are given as fixed hexadecimals and continue with the round function. There will be 80 rounds since there are 80 32-bit pieces per chunk. For every 20 rounds, there will be different F functions and K values. At the same time, we constantly change the values of A,B,C,D,E. The new values of B will be A, the new value of C will be B shifted 30 bits to the left, D will be C, and E will be D. A will be E XOR F XOR K XOR A shifted 5 bits to the left XOR W where W is the 32-bit chunk we had calculated previously. At the same time, after the end of 80 rounds, we will add the current values of A,B,C,D,E and add them to H0,H1,H2,H3,H4 respectively. Once we have gone over every chunk, we will then join all of the H values together and return its hex value as the 160-bit Hash value for SHA-1.



With the implementation of HMAC and SHA-1, we were able to create a functioning and well-secured MAC algorithm. What this means between the communication between the server and the client is that when one sends a message to the other through the network, we will now have an extra level of security to ensure that the message we receive is correct. When the server or client sends a message over the network to the client, we will also send the Hash value along with the message. This implies that when the receiver gets the message with the hash value, the receiver will plug the message into the hash function and then compare the result with the hash value that came along with the message. If the hash value is the same as the result, then this implies that the message wasn't corrupted and now the receiver has access to the message. However, if the result does not equal the hash value that came along with the message, this implies that the message was corrupted and that communication must be terminated.



Overall, the implementation of the HMAC provides another level of security on the network between the server and the client, meaning we are much less likely to be intercepted by an attacker without noticing the corruption on the message being sent through. Thus, the implementation of HMAC and SHA-1 is vital to the overall security of the network between the server and client to prevent any sort of attacks.