Assignment #3

CPEN 442

September 8

<alias of the student>

Department of Electrical and Computer Engineering

University of British Columbia

Vancouver, Canada

1. How to Install and Run the Program
2. Description of the Program

We wanted to develop a VPN that allows both the client and the server to communicate with each other without the need to wait for a reply. Each client therefore creates two socket connections: one for sending information and one for receiving information. In order for the two clients to talk to each other, one must agree to be the server and the other, the client. The server will instantiate the server connection first, while the client will connect to the server before instantiating its own server socket. When the connections are established, the clients will try to establish mutual authentication. If successful, the clients will share a new symmetric key for further communication.

The data is protected using SHA-256 cryptography. Each block size as suggested by the algorithm name is 256 bits. We generate a hash message authentication code to provide data integrity and authentication of the message. This is done by generating a new key, encrypting the hashes of the message and then adding the nonce to the hash. Once the hash has been finalized, we send the cipher plus the nonce and the hash. To decrypt the data, we first verify the MAC to ensure integrity. If successful, the cipher is then decrypted using the shared symmetric key and the nonce.

We want to implement mutual authentication that provides perfect forward secrecy. A shared secret key is provided at the beginning of the connection between the client and the server. We chose to use the ephemeral Diffie Hellman for key exchange. Diffie Hellman key exchange allows us to achieve perfect forward secrecy. If an attacker who does not know the initial shared symmetric key, they can still record all the data being transferred. If the symmetric key is revealed later on, the attacker may be able to decrypt the messages they kept. With the Diffie Hellman key exchange, each session creates a new symmetric key. So even if the

attacker knows the initial shared symmetric key, they are not able to decrypt the messages that were stored.

The mutual authentication protocol we implemented has 4 states. The step by step for this protocol is:

1. The client sends a nonce (Ra) to the server.
2. The server receives Ra
   1. Creates own nonce (Rb).
   2. Creates an identifier (“bob”).
   3. Creates a value.
   4. Encrypts the identifier, Ra, and value.
   5. Sends Rb + the cipher
3. Client receives the message
   1. Decrypts the cipher with the shared key.
   2. Extracts the identifier, Ra, and value.
   3. Verifies Ra is correct
   4. Creates session key.
   5. Creates identifier (“alice”)
   6. Computes a value
   7. Encrypts the identifier, Rb, and value.
   8. Sends cipher back.
4. Server receives the message
   1. Decrypts the cipher with the shared key.
   2. Verifies Rb.
   3. If successful, create session key.

During the mutual authentication step, the values of and are shared between the clients. The information was encrypted with the shared secret value, so attackers will not be able to guess what the values are. The client will then compute, and the server will compute . The reason that they both obtain s is because:

*Ab* mod *p* = (*ga* mod *p*)*b* mod *p* = (*ga*)*b* mod *p* = (*gb*)*a* mod *p* = (*gb* mod *p*)*a* mod *p* = *Ba*mod *p*[1].

This is secure because it is will take a long time for an attacker to find a and b to decrypt the key. Because these are session keys, each time the client connects, it will create a new key, and therefore the old key is practically useless.

The g and p numbers are hardcoded into the server and client. By using the number field sieve method, it can generate a lookup table for a given prime(size 512 bits) in a few years. Since the p is static, if given enough time, the number field sieve can find *a* for a given ga  mod p in a few days. A more secure way to implement g and p is to have the server regenerate p routinely. The prime number being used currently is 90 digits which equates to 300 bits give or take. For public distribution, it would be best to generate a bigger prime number.

# please talk about key sizes and more about part 4 encryption key size, and integrity key size