Assignment 1 - Protocol Security Lab

Josiah Plett - s241748

Ethan Andrew Misa - s242124

Chad Lantz - s241783

02239 - Data Security

October 9th, 2024

Task 1

1 - Purpose of Protocol

This protocol is trying to achieve a secure send of a payload from **B** to **A**, where **A** has verified who **B** is and **B** has verified who **A** is. The protocol begins with **A** — which is already verified with **s** — trying to verify itself to **B**, which gives us a hint for a typical real-world scenario where participants have this initial knowledge and want to achieve the goal.

Suppose **A** is a normal WhatsApp user. They've logged into WhatsApp before, so they and whatsapp's server **s** have verified each other; they have a shared secret. Now, a new user **B** who has never used WhatsApp would like to send **A** a file **Payload** through WhatsApp, without WhatsApp having to verify **B**. We want **A** and **B** to verify each other before the file is securely sent, and then to finish it off, we want to securely send that file from **B** to **A** without WhatsApp seeing the data.

2 - Describing the Attack

First we establish what the attack accomplished. It broke the goal of *Weak Authentication;* that is, when **B** thinks it's verified **A** (or vice versa), it actually hasn't. Specifically, **I** poses as **B** to **A** without any prior knowledge, and successfully not only convinces **A** that it is **B**, but it also convinces both **B** and **s** that it's **A**.

Now to discuss how it accomplished this. **I** first received **NA** and **KAB(A)** from **A**, encrypted using **I**'s own public key, so it could then read these values. It then impersonates **A** to **B** and sends **NA** plus *its own KAB,* **KAB(i)**, to **B**. From here the rest is history, because **I** can successfully impersonate **A** and **B** by acting as a middleman, just translating between the two **KAB**s that it knows, while **A** only knows **KAB(A)**, and **B** and **s** only know **KAB(i)**. In summary,

I poses as a middleman very early in the process, to intercept and replace the **KAB** that **A** and **B** are trying to exchange.

3 - Preventing the Attack

The first step in preventing this attack is realizing **KAB** should be sent from **s** to **B**, to fix the *Weak Authentication* issue. That's the only way to verify to **B** that the **KAB** it got from **A** is legit, since **A** isn't yet trusted while **s** is. We can't send **KAB** directly to **s** from **A** though, since **KAB** will be leaked publicly when **s** sends it. So, **A** encrypts **KAB** with **B**'s private key when it sends it to **s**. Voila, no more **KAB** impersonation, and **s** can't read **KAB**.

However, now **I** can initiate a replay attack on **B**, tricking a second **B** into sending a payload with the same data as the first, breaking *Strong Authentication*. Thus, the second and last step to preventing this attack is simply by having **B** generate a nonce **NB** when first sending a message to **A**, which **B** can use to verify it hasn't been duplicated when it receives the secure message from **s**.

In short, adding an encrypted **KAB** to the **s** message and giving **B** a nonce **NB** fixed all issues, makes the protocol secure.

See *Task 1 Solution* in the *Code Index* to reference our protocol definition changes.

4 - Improving Guessable Secrets

The guessable **sk(A,s)** poses a risk to the original program because **I** can intercept an encrypted message from **A** to **s**, and replicate it by trying many values for **sk(A,s)** until it finds a value that matches, and boom, **I** knows **sk(A,s)** which allows it to directly impersonate **A** instead of playing the middleman.

There is *not* another attack against our fix, described above in part 3. To understand why, we first need to note how someone with **sk(A,s)** (which they guessed), would try to use that knowledge to decrypt the **Payload**. Firstly, they need to get **KAB** to decrypt the **Payload**. With our updates, found in the *Code Index* of this assignment, you can see that **KAB** is sent during three messages, but all three of them are encrypted by **B**'s private key **pk(B)**, and thus off limits. So, since our introduction of **NB** prevents impersonation attacks, the intruder is forbidden from accessing **KAB** even if it does have **sk(A,s)**.

Task 2

1 - Purpose of Protocol

The protocol's purpose is to create a secret key **M** that is shared between the Agents **A** and **B** while authenticating the connection between themselves. For this protocol, **M** will be encrypted by a composite secret key built on **g**, a global static variable, by raising it to various powers. The protocol starts with **A** sending **B** the first component of their future shared key that's built on **X**, a private number only **A** knows: **exp(g,X)**. After receiving the message from **A**, **B** forwards this component of the shared key and generates its own side of the shared key, **Y**, and sends **exp(g,Y)** to the honest server **hm**. **hm** then takes both sides of the shared key and encrypts them first using the shared key between **hm** and **A,** then sends this safely to **B** using **hm** and **B**'s shared key. Next, **B** decrypts the outside of the message then sends the encrypted contents **exp(g,X)** and **exp(g,Y)** to **A**. Based on the dual encryption with the server's shared keys, **A** is able to ascertain that the server believes **B** is not an intruder. Now that **A** has access to **exp(g,Y)** by the use of exponent laws both **A** and **B** are able to generate the same key with **A** applying **exp(exp(g,Y) ,X)** and **B** applying **exp(exp(g,X) ,Y)**. Finally **A** uses the key to send the secret **M**, completing the secure message exchange.

Here’s a real-life example of a use case for this protocol. Suppose **A** and **B** are at a cafe together, and they want to play a direct private chess match against each other on chess.com, **hm**, over a public network. Since they’re in the same place they want a direct connection, but they don’t want people peeking at their chess moves on the public network. So, they need to establish a shared key **M** which will serve as a symmetric key letting them talk directly, securely. This protocol supports this, using **A** and **B**’s existing relationships with **hm**.

2 - Describing the Attack

In this attack, the intruder used two sessions to circumvent an authentication between A1 and B1, which is why the initial goal failed was *Weak Authentication*. In its first session, it steps in between the first message from **A1** to **B1** containing **exp(g,X)**, which is evidence of A’s secret and unique value **X**. **I** can then start talking with **B1** (in session 1) after creating its own **X**, so moving forward **B1** thinks **I** is **A1**, breaking *Weak Authentication*. **I** can exploit this by starting a new session as a middleman with **B2**, until **hm** provides a signed, encrypted message using **sk(B1,hm)** as a final proof to what it thinks is **B2**, which **I** intercepts. Now that **I** has this message that only **hm** could’ve created that looks like it’s meant for **B1** to pass to **A1**, it can forward this to **B1**, establishing a secure connection between **I** and **B1** using **I**’s fraudulent **X**.

In short, **I** intercepts the first communications between **A1** and **B1**, replacing **A1**’s true **X** with its own; it then runs through a full new session until it gets a signed message from **hm** to use to fully convince the original **B1** that its fake **X** is legit.

3 - Preventing the Attack

The main problem of the attack was that the intruder, posing as **A**, was able to intercept a message from **hm** in session 2 meant for **B2**, that it then forwarded to **B1** within session 1. To fix this, we require that **hm** also sends an encrypted version of **exp(g,X)** using **sk(B,hm)**, which **B** can use to cross reference with its previous **exp(g,X)** ensuring that the new **X** matches the one it received from **A** earlier.

This is a simple change, but it's powerful because it prevents multiple different sessions from being strung together by an intruder, since there's a unique secret key, **X**, passed around and cross-referenced throughout the process. No more middle-manning possible.

See *Task 2 Solution* in the *Code Index* to reference our protocol definition changes.

4 - Dealing with Dishonest Parties

The first version of the protocol relies on the server being trustworthy, of course. In fact, 100% of the private, static starting knowledge of all parties in the protocol involves **hm**. If we replace **hm** by **Hm**, this means both **A** and **B** have absolutely no safe private knowledge at the beginning of the protocol, making an impersonation attack clear and easy.

This means all private knowledge in the problem must be constructed from the ground up, massively increasing the complexity of the protocol. In essence, this means not only that protocol is nullified, but establishing a secure connection through **Hm** through security methods taught in this class is impossible.

However, the answer to the question "Are there *any* protocols with the same initial knowledge and goals that are secure?" is "Yes, there are." Modern cryptographic tools like zero-knowledge proofs (ZKPs) can allow one party to prove knowledge of a secret without revealing said secret, which enables **A** and **B** to establish a secure connection directly.

Code Index

Task 1 Solution

```
Types: Agent A,B,s;
       Number NA,NB,Payload;
       Symmetric_key KAB;
       Function sk,pk

A->B: {A,B,NA,KAB}pk(B)
B->A: {|A,B,NA,NB|}KAB
A->s: {|A,{KAB}pk(B),NA,NB|}sk(A,s)
s->B: {A,{KAB}pk(B),NA,NB,s}inv(pk(s))
B->A: {|A,B,NA,NB,Payload|}KAB
```

Task 2 Solution

```
A->B: A,B,exp(g,X),{|A,B,exp(g,X)|}ck(A,hm)
B->hm: {|A,B,exp(g,X),exp(g,Y),
         {|A,B,exp(g,X)|}ck(A,hm)|}ck(B,hm)
hm->B: {|B,A,exp(g,X),{|B,A,exp(g,Y)|}sk(A,hm)|}sk(B,hm)
B->A: exp(g,Y),{|B,A,exp(g,X),exp(g,Y)|}sk(A,hm)
A->B: {|M|}exp(exp(g,X),Y)
```