A Shared-Key Security Protocol and Its Flaw

Waseem Akhtar Mufti
wmufti@cs.aau.dk
Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, 9220 Aalborg, Denmark

Abstract:
Behaviour of cryptographic protocols is notoriously hard to understand because such a protocol is a concurrent process and can run with multiple instances in parallel. Such behaviour becomes more complicated because of malicious activities of attacker who is always assumed to be present in the environment of network. Being a public network (internet) it is very difficult to know who is who? Such threats need to be handled by a flawless security protocol which can guarantee to security of computer systems. In this paper we have analysed and compared the two versions (flawed and un-flawed) of Needham-Schroeder Shared-Key (NSSK) protocol and have provided clear and detailed description to understand its flaw and how an unflawed version defeats an attacker.

1. INTRODUCTION

Computer technology has been changing our daily lives. Activities of society and economic system rely on computer networks for communication, finance, energy distribution, transportation. The information age continues to evolve as well as the internet expands; electronic communication is becoming the preferred means of interaction for commercial, industrial and private use. Therefore, it is important to ensure that transmitted information is not compromised by malicious parties, especially in areas such as defence, medicine, and ecommerce, where leakage of information and corruption could be serious consequences. In fact, in order to protect against potential threats, communication messages are frequently secured by cryptographic applications and these secured exchange of messages are known as security protocols.

Security protocols are becoming more and more interesting issues, though much research on this topic exists. In fact, there are open issues that require more attention. It is not surprising that much work has been aimed at analysing of security protocols. In recent years, some available methods for analyzing security protocols using process algebra, such as CSP [1], Spi- calculus [2, 5]. These techniques allow us give precise description of essential properties of concurrent and communication programs. Moreover, those facilitate modelling and verifying authentication protocols. Those are the formal ways, which provide rigorous mathematical analyses of computer systems to evaluate cryptographic protocols and security mechanism.

In this paper we discuss the original Needham-Schroeder Shared-key Protocol (NSSK) [3], invented by Roger M. Needham and Michael D. Schroeder in 1978, and its attack which was first observed by D. Denning and G. Sacco [4] in 1981. It allows individuals communicating over a network to prove their identity to each other while also preventing eavesdropping or re-play attacks, and provides for detection of modification and the prevention of unauthorized reading. This protocol is few lines of computer program, since its instances on a network run as concurrent programs running in parallel therefore its behaviour becomes very complex. Because of its complex behaviour the attack was found after quite a long since it was being used by many organizations which caused huge financial loss.

In rest of the paper we define few cryptographic terms. In section 3 we describe how a NSSK protocol works. Section 4 and 5 describe the actual attack on protocol, few cryptographic terminologies and blocking of attack.

2. SHARED-KEY AND NONCE

Before going into details of protocol mechanism we define few concepts.

2.1 Shared-key

Shared-key is used by two or more communicating principals for both encryption and decryption purpose. It is generated by one of principals on network. In our example of NSSK protocol shared-key has been generated by a Server denoted by S which is a trusted third party. Before communication begins the principals request to a trusted third party which generates the key and distributes. A third party server, which distributes key, is considered to be an absolutely trusted principal on network. At this point we are not going into details of how encryption and decryption is done, which can be simply considered as an algorithm which is only to known to those principals involved in communication. A principal may request a separate shared-key for each principal on network to communicate with. Details of how the order of send and receive messages takes place becomes clear as we go along this article.

2.2 Nonce

Nonce is a fresh random number which can be generated by any principal and is never repeated, regenerated or copied by any principal on network. Since it is always a new random number therefore we call it a fresh number. Nonces are normally used to guarantee the freshness or originality of messages sent or received on network. Some

protocols use Time stamps [4] instead of nonces; NSSK protocol described here uses nonces.

3 HOW DOES IT WORK? PROTOCOL AS A CONCURRENT PROCESS

Here we will describe the mechanism of awed version of NSSK protocol. The original NSSK protocol allows server to generate a session key $\boldsymbol{K_{ab}}$ for principal $\boldsymbol{A}$ and $\boldsymbol{B}$. Through shared-key $\boldsymbol{K_{ab}}$ both principals $\boldsymbol{A}$ and $\boldsymbol{B}$ are able to communicate with each other. This protocol can be represented as follows. Figure 1 describes its pictorial representation.

A, B, S: principal
N_a, N_b: nonce
K_{as}, K_{bs}, K_{ab}: key
dec : nonce → nonce

Msg 1 $A \rightarrow S : A, B, N_a$
Msg 2 $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{\text{Kbs}}\}_{\text{Kas}}$
Msg 3 $A \rightarrow B : \{K_{ab}, A\}_{\text{Kbs}}$
Msg 4 $B \rightarrow A : \{N_b\}_{\text{Kab}}$
Msg 5 $A \rightarrow B : \{dec(N_b)\}_{\text{Kab}}$

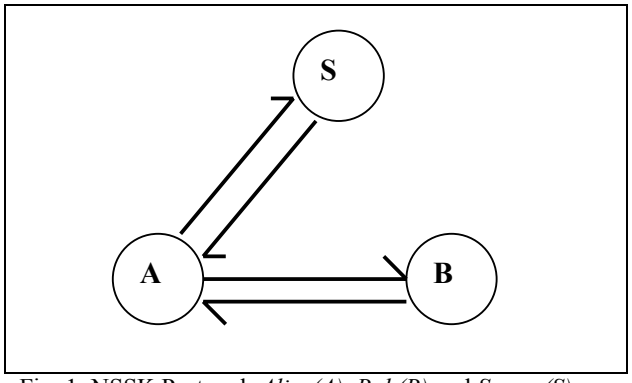

Fig. 1: NSSK Protocol: *Alice(A)*, *Bob(B)* and *Server(S)*

A, ***B*** and ***S*** are three principals ***Alice***, ***Bob*** and ***Server*** respectively. N_a and N_b are two nonces generated by *A* and *B* respectively. K_{as} and K_{bs} are two shared-keys used for communication between *Alice* and *Server*, and *Bob* and *Server* respectively. These two keys exist in advance to establish the trust with *Server* before the actual run of protocol. Key K_{ab} is generated by *Server* to establish trust between *Alice* and *Bob*. *dec* is a decrement function which is used to performs decrement operation on nonce.

In **message 1** *Alice* sends message to *Server* by saying '*I am Alice and want to communicate with Bob and here is my nonce* N_a'. In m**essage 2** *Server* sends a cipher (encrypted) text containing N_a, *B* and K_{ab} along with an other cipher text $\{K_{ab}, A\}_{\text{Kbs}}$ which is encrypted by key K_{bs}. Note that the whole **message 2** is encrypted by key K_{as} which can only be decrypted by *Alice* since he already shares this key with *Server*. Moreover, when *Alice* receives **message 2** it checks back the nonce N_a which guarantees the freshness of **message 2** means that it has only been generated by *Server*. *Alice* can not decrypt the cipher text $\{K_{ab}, A\}_{\text{Kbs}}$ because it does not have key K_{bs}. Here it is very important to understand that the network on which *Alice*, *Bob* and *Server* are communicating is a public network (internet) and everyone on network can record every message and this is the reason that three principals are using encryption and decryption to achieve secrecy and authenticity of messages.

In m**essage 3** *Alice* simply forwards cipher text $\{K_{ab}, A\}_{\text{Kbs}}$ to *Bob* which was embedded by *Server* in **message 2.** As mentioned earlier *Bob* has already access to key K_{bs} therefore it can decrypt and now has received the key K_{ab}. Figure 1 shows that there is no direct connection between *Server* and *Bob* where *Bob* can only receive messages through *Alice*. *Server* could also send K_{ab} directly to *Bob* in parallel to **message 2** because all principals are running separate instances of protocol concurrently. If it were so, then it was possible for *Bob* to receive a cipher text from *Alice* before receiving key K_{ab} from *Server*, therefore having no direct connection between *Serve* and *Bob* confirms the correct sequence of messages.

In **message 4** *Bob* sends its nonce to *Alice* encrypted by K_{ab} then *Alice* responds in **message 5** by decrementing *Bob*'s nonce and encrypts it with key K_{ab}. At this point trust has been established between *Alice* and *Bob* through shared-key K_{ab} therefore they can securely communicate.

But this is not end of story; some one can make fool to *Alice* pretending *Bob* which is described in next section.

4. ATTACK ON NSSK PROTOCOL – MAN IN THE MIDDLE

The vulnerability of NSSK protocol was first observed by D. Denning and G. Sacco [2] in 1981. It was observed that an attacker can easily take advantage if at any point in the protocol **key** $\boldsymbol{K_{ab}}$ is compromised. Before going into details of such attack we define few terms as follows.

4.1 Key compromisation by crypto-analysis

In cryptography the term Key compromisation means that while there is encryption service is running between two principals on the network one can succeed to guess about secret key, in our example it is K_{ab}, and hence key has been compromised.

Cryptanalysis is not so easy to get key which is being used in the current session of communication, rather it is a very exhausting programming effort based on complex mathematical operations to get it done successfully. A successful cryptanalysis may take one week, one month or one year to get the required key. As a matter of fact a communication session between two principals is not too long such that any attacker succeeds by doing cryptanalysis. Therefore it is believed for an attacker to be impossible to get required key during the current session of communication between two principals; but what

happens if an attacker uses an **old session key** into **new session key** which he was successful to get that key by spending a long time using cryptanalysis? This was the attack which was first observed in NSSK protocol which is known to be **key compromised attack.**

4.2 Secrecy

It is a basic property of any security protocol defined as the freshness or intactness of data being transferred between principals on network. An information is secret if no unauthorised person has read or manipulated it, although it can be recorded or copied as many times as an unauthorised person wants but it can not be read or understood. Every security protocol must satisfy its secrecy property.

4.3 Authenticity

It is also a basic property of security protocol which must be satisfied. *Alice* must be able to know whether the message transmitted by *Bob* has been successfully received from the **real** (Authorised) ***Bob***, after all on a computer network (internet) no one can differentiate between Monkey and Horse. Anyone in between can pretend and record messages and therefore can forward or **re-play** to any other principal. Although, forwarding or re-play of messages does not mean to read, change or understand it because messages are always in form of cipher text. This kind of malicious activity is also called a replay attack [4] which will help in describing the actual attack on NSSK protocol.

4.4 Claimed attack on NSSK protocol

Following is the attacked NSSK protocol description:

i.1 $A \rightarrow S$: A, B, N_a
i.2 $S \rightarrow A$: $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{Kbs}\}_{Kas}$
i.3 $A \rightarrow \boldsymbol{I}(B)$: $\{K_{ab}, A\}_{Kbs}$
ii.3 $\boldsymbol{I}(A) \rightarrow B$: $\{K_{ab}, A\}_{Kbs}$
ii.4 $B \rightarrow \boldsymbol{I}(A)$: $\{N_b\}_{Kab}$
ii.5 $\boldsymbol{I}(A) \rightarrow \boldsymbol{I}(A)$: $\{dec(N_b)\}_{Kab}$

Messages i.1 to i.3 are a snapshot of protocol from an old session. During the **old session** in **message i.3** an intruder (attacker) ***I*** pretend to be *Bob* and records every cipher text sent by *Alice*. Now assume that in addition of recording messages ***I***, by means of cryptanalysis, have also succeeded to get key K_{ab} which belongs to an **old session** some time ago. But ***I*** could not break the secrecy of messages during the previous session because just after that the **old session** was ended between *Alice* and *Bob*.

Messages ii.3 to ii.5 represent **new session** between *Alice* and *Bob*. In **message ii.3** ***I*** pretends to be *Alice* and replays (re-sends) an **old ticket** $\{K_{ab}, A\}_{Kbs}$ which was recorded by ***I*** during the old session which also contains an old session key K_{ab}. At this stage *Alice* and *Bob* believe that they are talking with authorised principals but actually they are talking with a **middle man (*I*)** who is not an authorised principal.

In **message ii.3** ***I*** sends an **old ticket** to *Bob*, now since *Bob* has key K_{bs} therefore it can decrypt the cipher text $\{K_{ab}, A\}_{Kbs}$. Now *Alice* and *Bob* are sharing and **old session key** with ***I***. In **message ii.4** ***I*** can decrypt $\{N_b\}_{Kab}$ by using old session key K_{ab} and similarly in **message ii.5**.

This attack was possible by re-playing the cipher text $\{K_{ab}, A\}_{Kbs}$ of an **old session**, but how to defeat this kind of attack ? The answer to this question has been given in next section.

5. FIXED VERSION OF NSSK PROTOCOL – DEFEATING THE ATTACKER

Following is the fixed version of NSSK protocol description:

1. $A \rightarrow B$: A
2. $B \rightarrow A$: $\{A, N_b\}_{Kbs}$
3. $A \rightarrow S$: $A, B, N_a, \{A, N_b\}_{Kbs}$
4. $S \rightarrow A$: $\{N_a, B, K_{ab}, \{K_{ab}, N_b, A\}_{Kbs}\}_{Kas}$
5. $A \rightarrow B$: $\{K_{ab}, N_b, A\}_{Kbs}$
6. $B \rightarrow A$: $\{N_b\}_{Kab}$
7. $A \rightarrow B$: $\{dec(N_b)\}_{Kab}$

In fixed version of NSSK protocol, before sharing of key K_{ab} *Bob* sends cipher text to *Alice* in **message 2** containing **nonce N_b** and name of *Alice* encrypted with key K_{bs}. As explained previously, nonce N_b will serve as an identifier to *Bob* during sharing of key K_{ab} in next messages. In **message 3** *Alice* also sends the cipher text $\{A, N_b\}_{Kbs}$ to *Server*. As a response in **message 4** *Server* embeds the cipher text $\{A, N_b\}_{Kbs}$ along with key K_{ab} encrypted with key K_{bs} and the whole message is encrypted by key K_{as}.

A clear difference can be observed in **messages 3** and **4** of NSSK flawed and fixed version protocols. In **message 5** when *Bob* receives cipher text it can decrypt using key K_{bs} and can check its own nonce N_b and receives the key K_{ab}. Since *Bob* had already sent nonce N_b to *Alice* in **messages 2** therefore nonce N_b guarantees the **freshness of key K_{ab}** which belongs to the current session of communication and since attacker ***I*** does not have access to nonce N_b therefore it can not attack on this protocol. Finally in **messages 6 and 7** *Alice* and *Bob* have established the trust using key K_{ab} and therefore can encrypt and decrypt messages without the fear of ***I***.

If, by any means, ***I*** had been successful to get an old session key K_{ab} even then it would not have access to nonce N_b. Since *Bob* always can check its nonce N_b before sharing the key K_{ab} therefore there is no chance of confusion between old and new session key (K_{ab}). Moreover, since nonce N_b guarantees the freshness of shared-key therefore we can say that it assigns the **freshness type** to key K_{ab} which can always be verified by checking the nonce N_b which is always a fresh number.

CONCLUSION

In order to understand a shared-key cryptographic protocol we have defined few related terms in very basic terminology. We have described both versions of original Needham-Schroeder Shared-key protocol with flaw and without flaw. We have compared both protocols in the light of already known attack. We have examined the flaw of NSSK protocol and found the possibility of **man in the middle attack** when key K_{ab} is compromised. Moreover, ***I*** exploits the **un- typed key** $\boldsymbol{K_{ab}}$ to use it from an old session by getting it confused with new session between *Alice* and *Bob*. We have also observed in fixed version that by assigning a **type** to K_{ab} by means of a nonce N_b attack can be defeated.

REFERENCES

[1] C.A.R. Hoare Communicating Sequential Processes. International Series in Computer Science, 1985. ISBN 0-13-153271-5 (0-13-153289-8 PBK) . Prentice Hall 1985.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi-calculus. In Fourth ACM Conference on Computer and Communications Security. ACM Press, 1997.

[3] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. Communications of the ACM, 21(12), December 1978.

[4] D. Denning and G. Sacco Timestamps in key distributed protocols. Communication of the ACM, 24(8):533–535, 1981.

[5] M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. CONCUR'97: Concurrency Theory, volume 1243 of Lecture Notes in Computer Science, pages 59-73. Springe-Verlag, 1997.