ADDING PERFECT FORWARD SECRECY TO KERBEROS

Wameedh Nazar Flayyih
University of Baghdad – College of Engineering – Computer Engineering Department

ABSTARCT
Kerberos system is a powerful and widely implemented authentication system. Despite this fact it has several problems such as the vulnerability to dictionary attacks which is solved with the use of public key cryptography. Also an important security feature that is not found in Kerberos is perfect forward secrecy. In this work the lack of this feature is investigated in Kerberos in its original version. Also a public key based modification to Kerberos is presented and it is shown that it lacks the prefect forward secrecy too. Then some extensions are proposed to achieve this feature. The extensions are based on public key concepts (Diffie-Hellman) with the condition of keeping the password based authentication; this requires little modifications to the original Kerberos. Four extensions are proposed; two of them modify the (Client-Authentication Server) exchange achieving *conditional perfect forward secrecy*, while the remaining two modify the Client-Server exchange achieving *perfect forward secrecy* but with increased overhead and delay.

الخلاصة
نظام Kerberos هو نظام قوي و فعال و مستخدم بصورة واسعة. لكن هناك عدة مشاكل يعاني منها هذا النظام، إحداها ضعفه أمام هجمات القاموس و قد تم حل هذه المشكلة بالاعتماد على التشفير بالمفتاح العام (public key). لكن هناك خاصية أمنية مهمة غير متوفرة في هذا النظام وهي السرية الأمامية التامة (أو السرية المستقبلية التامة). هذا البحث يقوم بدراسة النظام من ناحية وجود هذه الخاصية أم لا بالإضافة إلى دراسة احد الأنظمة المحورة عن النظام الأصلي باستخدام التشفير بالمفتاح العام (public key)، ويظهر البحث افتقار النظامين الأصلي و المحسن لهذه الخاصية. ثم يقدم البحث مقترحات تتضمن بعض التعديلات على نظام (Kerberos) من اجل تحقيق السرية الأمامية التامة. التعديلات تستند على مبادئ التشفير بالمفتاح العام (public key) طريقة (ديفي–هلمان) بشرط المحافظة على خاصية التعريف بالاعتماد على كلمة السر و هذا يضمن كون التعديلات قليلة على النظام الأصلي و سهلة التطبيق. يعرض البحث أربعة تعديلات، اثنتان منها تغير الحوار بين العميل و خادم التعريف (Client-Authentication Server) محققة السرية الأمامية الشرطية، بينما تغير الاثنتان الأخريان الحوار بين العميل و الخادم (Client-Server) محققة السرية الأمامية التامة لكن مع زيادة التأخير.

KERBEROS, Public Key, Forward Secrecy, Authentication, Diffie-Hellman, Security

INTRODUCTION

Kerberos is an authentication system developed at Massachusetts Institute of Technology (MIT) as part of Project Athena [Miller et al,1987]. There are two versions of this system: Kerberos v4 and Kerberos v5. The later solved some of the problems found in v4. It is based on symmetric encryption and trusted third party. Several problems are found in Kerberos, but dictionary attacks is the most known attack and the one that has got a lot of attention in trying to reduce its risk. One of the proposed extensions to solve this attack, known as Public Key Cryptography for Initial Authentication in Kerberos (PKINIT), modifies the basic protocol to allow public-key authentication; in the process, it adds a fair amount of complexity to the protocol [Cervesato et al,2008]. Other solutions are the use of the Secure Remote Password (SRP) protocol [Wu,1999], and the Public key based Kerberos for Distributed Authentication (PKDA) [Sirbu and Chuang,1997]. Other directions that are taken by some researchers is to use Elliptic Curve Cryptography (ECC) and Elliptic Curve Diffie Hellman (ECDH) as the public key mechanism in Kerberos [Zhu et al, 2008][Ozkan, 2003] and the same mechanisms can be used in the extensions that will be proposed in this paper.

A security feature that is not found in Kerberos is *"perfect forward secrecy"*, which means that the compromise of long term secret keys leads to the compromise of all past session keys [Menezes et al,1996], a problem that is considered serious in some environments, and it is the main focus of this work. The reason is that it was not a feature in the original design, so it uses long term symmetric keys to encrypt the session keys. The details of the protocol will be shown in section 2. In section 3, one of the extensions presented in PKINIT will be shown for comparison with the extensions that will be proposed in this paper in section 4 to add perfect forward secrecy to Kerberos. A comparison is made between the proposed extensions in section 5.

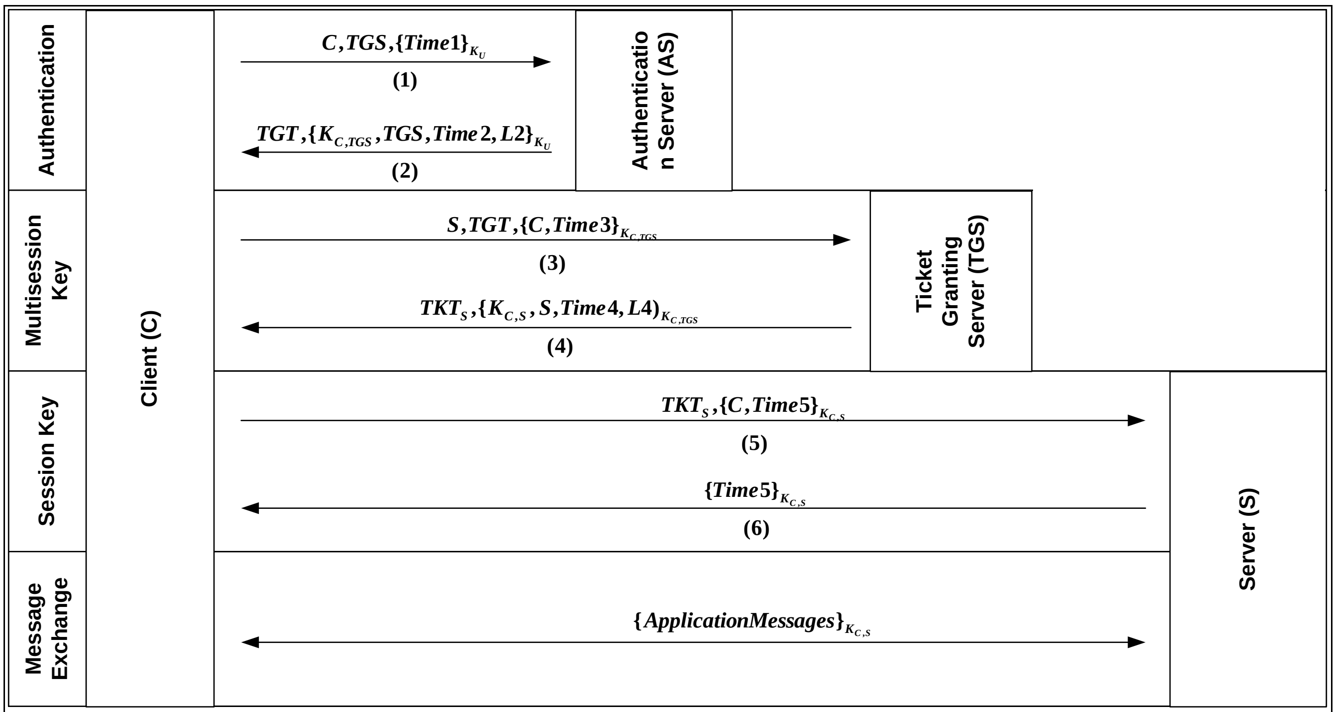
KERBEROS V5

It is well known that Kerberos is based on symmetric encryption and trusted third party. Figure (1) shows the exchanges of the original Kerberos v5. The client (C) shares with the authentication server (AS) a long term key; and truly speaking the secret is a password (K_U shared between the "User" on the client machine and AS).

- The password is used to encrypt a timestamp and sent by the client to AS which decrypts and checks for time synchronization, this step is used to authenticate the user.
- AS sends a key ($K_{C,TGS}$) to be used between the client (C) and ticket granting server (TGS), this key is sent encrypted using K_U. AS also sends the client a ticket granting ticket (TGT) which contains the same key, ($K_{C,TGS}$), encrypted using a long term key shared between AS and TGS. Steps 1 and 2 represent the authentication phase exchange.
- The next phase, steps 3 and 4, are carried between the client (C) and TGS. In this phase the client authenticates itself to TGS by sending a timestamp encrypted using $K_{C,TGS}$. The client also sends the ticket TGT granted by AS, also it sends the name of the server S that it wants to communicate with.
- TGS decrypts the ticket TGT using the key shared with AS ($K_{AS,TGS}$), and it extracts the key $K_{C,TGS}$. Then it decrypts the message sent by the client and checks the timestamp to see the authenticity of client. Then TGS creates a session key ($K_{C,S}$) to be used between C and S. In this exchange the key ($K_{C,TGS}$) is used to authenticate C to TGS in step 3 and used in step 4 to encrypt the session key $K_{C,S}$. TGS also sends the client a ticket (TKT_S) to be used with the server (S). This ticket TKT_S contains the same key, ($K_{C,S}$), encrypted using a long term key shared between TGS and S.
- Steps 5 and 6 represent the last phase at which the exchange occurs between the client and server. In step 5 the client authenticates itself to the server by sending a timestamp encrypted under the session key ($K_{C,S}$). The client also sends TKT_S to the server.

- The server decrypts TKT and extracts the key ($K_{C,S}$), and use this key to check the authenticity of client message. Then the server authenticates itself to client by sending a timestamp encrypted under the same session key.

Since the client passes through steps 3 and 4 for each server it wants to contact, the key ($K_{C,TGS}$) is sometimes called a multisession key, because $K_{C,TGS}$ is used to encrypt many session keys. The concept of perfect forward secrecy is lost by encrypting the session keys using the multisession keys, and by encrypting the multisession keys using the password derived keys. So the compromise of multisession key causes the compromise of all session keys encrypted under it. Also the compromise of the password derived key (K_U) causes the compromise of the multisession key and consequently the session keys.


Authentication
Multisession Key
Session Key
Message Exchange
Client (C)
$C, TGS, \{Time1\}_{K_U}$
(1)
$TGT, \{K_{C,TGS}, TGS, Time2, L2\}_{K_U}$
(2)
Authentication Server (AS)
$S, TGT, \{C, Time3\}_{K_{C,TGS}}$
(3)
$TKT_S, \{K_{C,S}, S, Time4, L4\}_{K_{C,TGS}}$
(4)
Ticket Granting Server (TGS)
$TKT_S, \{C, Time5\}_{K_{C,S}}$
(5)
$\{Time5\}_{K_{C,S}}$
(6)
$\{ApplicationMessages\}_{K_{C,S}}$
Server (S)

$\{message\}_K$: means "message" is encrypted using key K

$TGT = \{K_{C,TGS}, C, TGS, Time2, L2\}_{K_{AS,TGS}}$

$TKT_S = \{K_{C,S}, C, S, Time4, L4\}_{K_{S,TGS}}$

L : Lifetime

$Time_i$: Timestamp at step i

K_U : Password derived key shared between the User and AS

Fig. 1 Original Kerberos v5 Exchange

PKINIT

Public-Key Kerberos PKINIT [Zhu and Tung, 2006] is an extension to Kerberos v5 that uses public key cryptography to avoid shared secrets between a client and AS; it modifies the authentication exchange but not other parts of the basic Kerberos v5 protocol. The long-term shared key (K_U) in the traditional exchange is typically derived from a password, which limits the strength of the authentication to the user's ability to choose and remember good passwords; PKINIT does not use K_U and thus avoids this problem.

In PKINIT, the client (C) and AS each has a pair of public-private key (PK_C, SK_C) and (PK_{AS}, SK_{AS}). $Cert_C$ and $Cert_{AS}$ are certificates that prove the authenticity of PK_C and PK_{AS} because they are signed by a trusted Certificate Authority. PKINIT can operate in two modes namely public key encryption mode and Diffie-Hellman mode. The keys are used during the authentication phase only. But they are used in different ways according to the mode. The Diffie-Hellman mode is investigated here and shown in Figure (2) because it achieves conditional perfect forward secrecy.

Diffie-Hellman [Diffie and Hellman,1976] key agreement provided the first solution to allow two parties, without shared keys, to establish a secret key by exchanging messages over unsecured channel. But it does not achieve authentication. Authentication is achieved here using the certificates. The key ($K_{C,TGS} = g^{ab}$) cannot be derived from Diffie-Hellman (D-H) exponents g^a and g^b which are sent publicly. Which means that AS will not send the key to the client encrypted under any key. So the compromise of the client's long term key (SK_C) will not help to get the key ($K_{C,TGS}$). This provides perfect forward secrecy. But ($K_{C,TGS}$) is also found in TGT encrypted under the long term key ($K_{AS,TGS}$) shared between AS and TGS. This yields the fact that the compromise of ($K_{AS,TGS}$) causes ($K_{C,TGS}$) to be compromised, meaning that perfect forward secrecy is not achieved. That's why we call it conditional perfect forward secrecy, because under the assumption that ($K_{AS,TGS}$) is shared between two well protected servers, the possibility of compromise is small but not impossible.

This extension involves several time consuming operations at both the client side and AS side because the goal was to deal with dictionary attacks on passwords.

Client Side

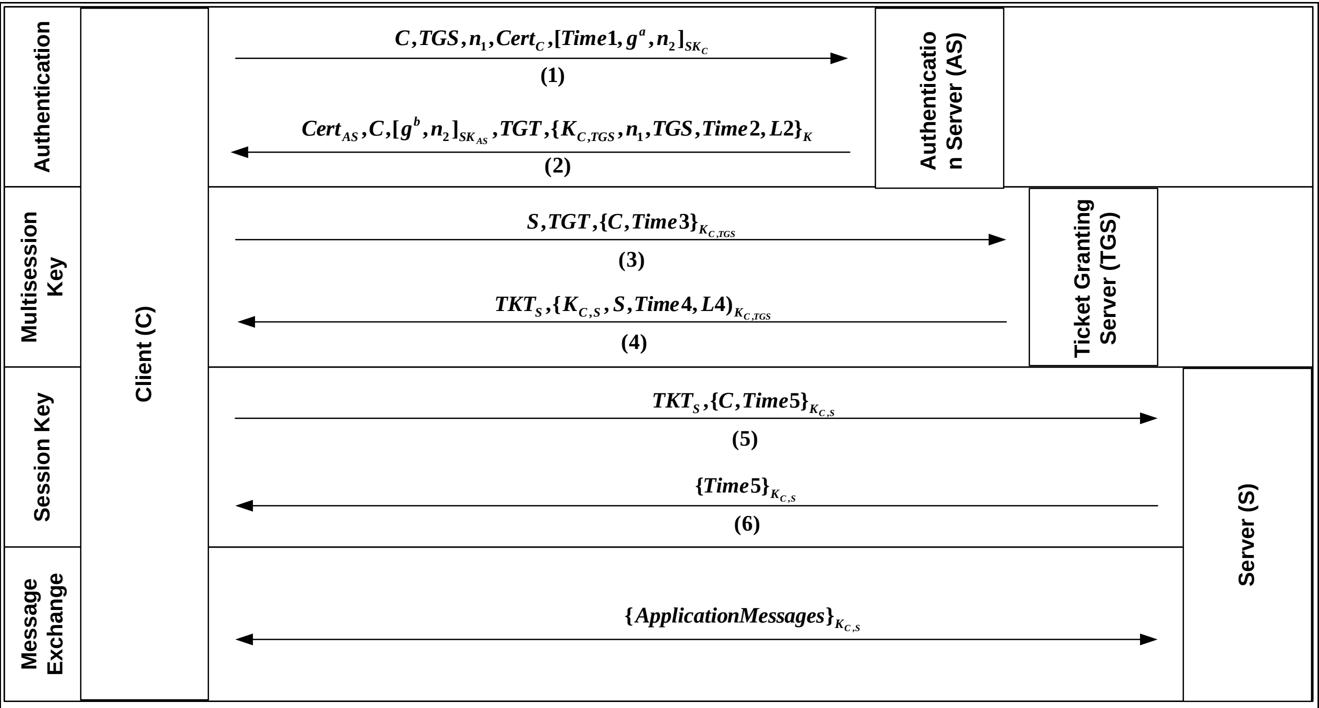
- Evaluating g^a.
- Digital signature creation using SK_C.
- Digital signature verification of $Cert_{AS}$.
- Digital signature verification using PK_{AS}.
- Evaluating g^{ab}.

AS Side

- Digital signature verification of $Cert_C$.
- Digital signature verification using PK_C.
- Evaluating g^b.
- Evaluating g^{ab}.
- Digital signature creation using SK_{AS}.

PROPOSED EXTENSIONS

The PKINIT using Diffie-Hellamn achieves conditional perfect forward secrecy, but it involves many public key operations and the requirement of certificate authority. This adds many changes to the original Kerberos. In the following sections some extensions are proposed to achieve perfect forward secrecy or conditional perfect forward secrecy with as ***little changes to Kerberos*** as possible. The extensions also consider ***the reduction of public key operations*** as possible. Four extensions are proposed, two of them modify the Client-AS exchange achieving *conditional perfect forward secrecy*, while the remaining two modify the Client-Server exchange achieving *perfect forward secrecy*. The solutions ***assume the password based authentication*** even that it has some weaknesses, but it stills the main authentication approach in most systems, so ***certificates are avoided***. By that we minimize the changes made to Kerberos which simplifies its implementation in the future.


Authentication
Multisession Key
Session Key
Message Exchange
Client (C)
Authentication Server (AS)
Ticket Granting Server (TGS)
Server (S)
$C, TGS, n_1, Cert_C, [Time1, g^a, n_2]_{SK_C}$
(1)
$Cert_{AS}, C, [g^b, n_2]_{SK_{AS}}, TGT, \{K_{C,TGS}, n_1, TGS, Time2, L2\}_K$
(2)
$S, TGT, \{C, Time3\}_{K_{C,TGS}}$
(3)
$TKT_S, \{K_{C,S}, S, Time4, L4)_{K_{C,TGS}}$
(4)
$TKT_S, \{C, Time5\}_{K_{C,S}}$
(5)
$\{Time5\}_{K_{C,S}}$
(6)
$\{ApplicationMessages\}_{K_{C,S}}$

$TGT = \{K_{C,TGS}, C, TGS, Time2, L2\}_{K_{AS,TGS}}$

$TKT_S = \{K_{C,S}, C, S, Time4, L4\}_{K_{S,TGS}}$

L : *Lifetime*

n : **Nonce (random number)**

$k = g^{ab}$

$Cert_X$: **Certificate of party X**

$[m]_{SK}$: **Signature of message m with secret key SK**

Fig. 2 Diffie Hellman based PKINIT exchange

<u>Changing The Client-AS Exchange (First Extension Group)</u>

The first extension group changes the exchange between the client and the authentication server (AS); this group has two extensions which are presented in part A and B below.

A) The extension uses the same concept of PKINIT with Diffie-Hellman but without the use of certificates. Figure (3) shows this extension, where only steps 1 and 2 are modified. The idea is to get ($K_{C,TGS}$) without being sent encrypted under a long term key. To accomplish this, the client generates private information (a) and sends public information (g^a). AS generates (b) and sends (g^b). Then both calculate g^{ab}, while no one other than the two can find this value because the publicly known info is only g^a and g^b, this achieves perfect forward secrecy. h(g^a) represents the hash of g^a, and is sent encrypted under K_U such that AS can be sure of the authenticity of g^a. On the other hand h(g^a ,g^b) is sent encrypted under K_U to prove the authenticity of g^b to the client, and to acknowledge the reception of g^a. The hash is sent instead of the information itself because it is smaller in size. The size of the hash output depends on the algorithm used and not on the size of input information. Some hash functions that can be used are MD5, SHA-1, and SHA-2. The recommended one is the SHA-2 in its various sizes. Also the National Institute of Standards and Technology (NIST) has launched a public competition to develop a new cryptographic secure hash algorithm, which NIST is naming "SHA-3" which is expected to be finally presented in year 2012 [William, 2008].

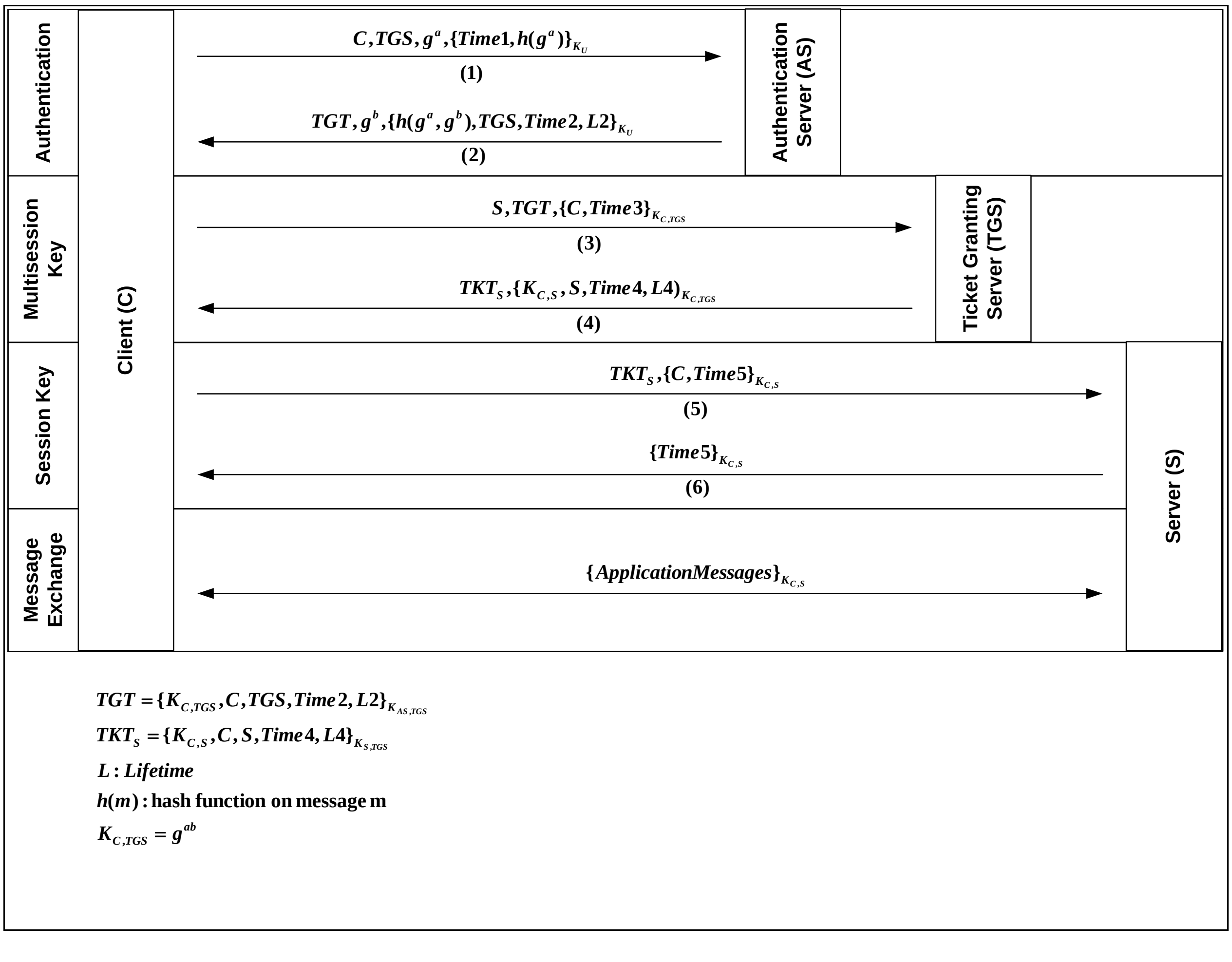
Like PKINIT in previous section, this approach achieves *conditional perfect forward secrecy*, since $K_{C,TGS}$ is sent encrypted using $K_{AS,TGS}$ inside the TGT. Since certificates are not required it gives some advantages concerning the computational load on the two parties, and the communication cost. The time consuming operations are given below:

Client Side

- Evaluating g^a.
- Evaluating g^{ab}.

AS Side

- Evaluating g^b.
- Evaluating g^{ab}.



Authentication
Multisession Key
Session Key
Message Exchange
Client (C)
Authentication Server (AS)
Ticket Granting Server (TGS)
Server (S)
$C, TGS, g^a, \{Time1, h(g^a)\}_{K_U}$
(1)
$TGT, g^b, \{h(g^a, g^b), TGS, Time2, L2\}_{K_U}$
(2)
$S, TGT, \{C, Time3\}_{K_{C,TGS}}$
(3)
$TKT_S, \{K_{C,S}, S, Time4, L4)_{K_{C,TGS}}$
(4)
$TKT_S, \{C, Time5\}_{K_{C,S}}$
(5)
$\{Time5\}_{K_{C,S}}$
(6)
$\{ApplicationMessages\}_{K_{C,S}}$
$TGT = \{K_{C,TGS}, C, TGS, Time2, L2\}_{K_{AS,TGS}}$
$TKT_S = \{K_{C,S}, C, S, Time4, L4\}_{K_{S,TGS}}$
L : Lifetime
$h(m)$: hash function on message m
$K_{C,TGS} = g^{ab}$

Fig. 3 Client-AS Modified Kerberos Proposed Exchange

B) A simple modification to the previous extension is to send the public D-H exponents (g^a) and (g^b) encrypted under K_U, as shown in figure (4). This adds additional security in the sense that D-H security is based on the concept that knowing (g^a) and (g^b) it is impossible to find g^{ab}. Here, (g^a) and (g^b) are not public so the problem becomes harder on the adversary. This modification keeps the same advantages of the previous solution with small increase in computational time caused by the encryption and decryption of (g^a) and (g^b). Since (g^a) is sent encrypted under K_U then AS will be sure of the authenticity of (g^a), so it is not necessary to calculate $h(g^a)$ and send it encrypted under the same key. In the same manner in the second step it is not required to send the hash of (g^b) encrypted under K_U. The message in the first step is smaller in size than in the previous extension by the size of $h(g^a)$. On the other hand the second step is of same size as previous extension because h(g^a,g^b) has the same size as h(g^a) by the definition of hash functions.

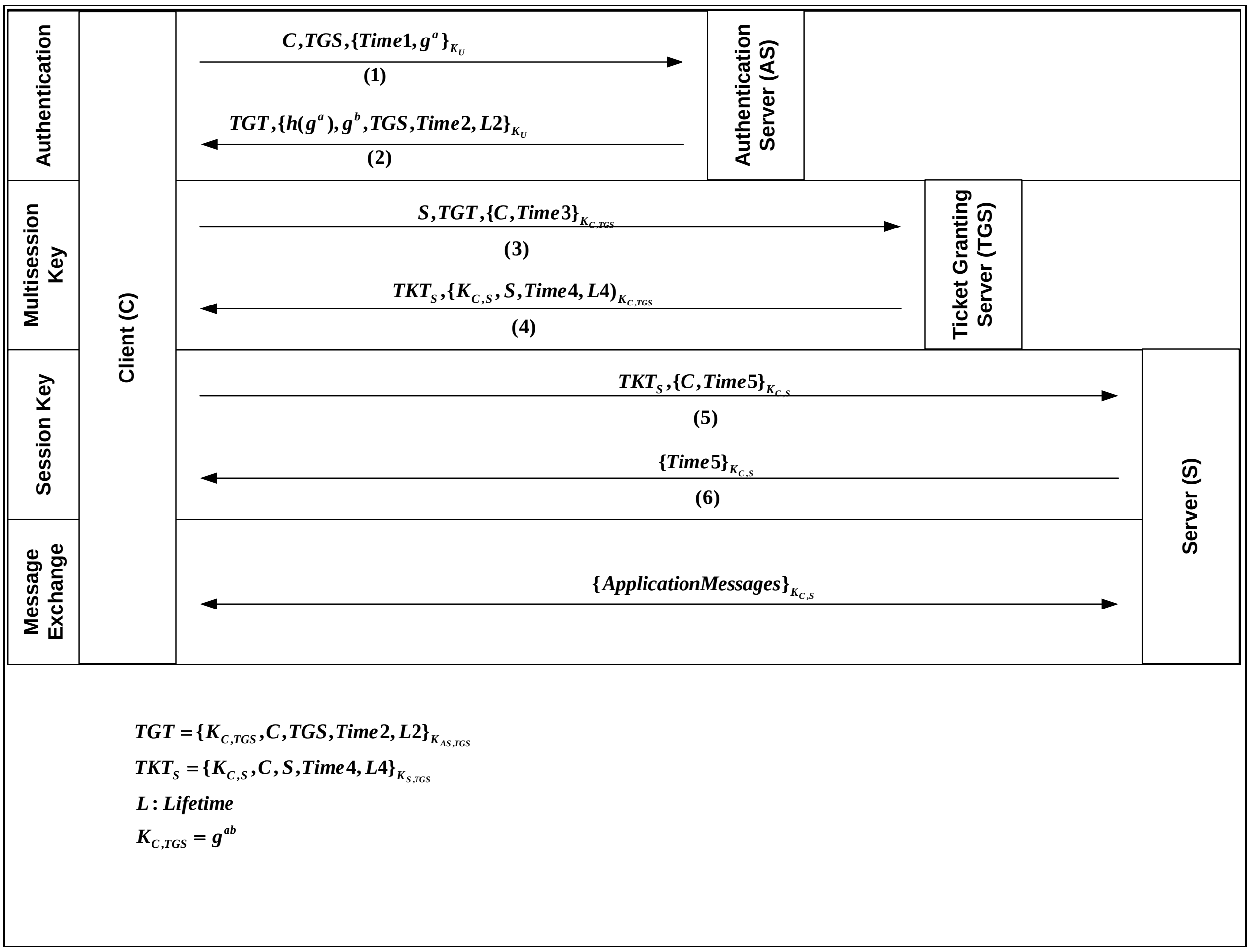

Authentication
Multisession Key
Session Key
Message Exchange
Client (C)
$C,TGS,\{Time1,g^a\}_{K_U}$
(1)
$TGT,\{h(g^a),g^b,TGS,Time2,L2\}_{K_U}$
(2)
Authentication Server (AS)
$S,TGT,\{C,Time3\}_{K_{C,TGS}}$
(3)
$TKT_S,\{K_{C,S},S,Time4,L4)_{K_{C,TGS}}$
(4)
Ticket Granting Server (TGS)
$TKT_S,\{C,Time5\}_{K_{C,S}}$
(5)
$\{Time5\}_{K_{C,S}}$
(6)
Server (S)
$\{ApplicationMessages\}_{K_{C,S}}$
$TGT=\{K_{C,TGS},C,TGS,Time2,L2\}_{K_{AS,TGS}}$
$TKT_S=\{K_{C,S},C,S,Time4,L4\}_{K_{S,TGS}}$
$L: Lifetime$
$K_{C,TGS}=g^{ab}$

Fig. 4 Client-AS Modified Kerberos Proposed Exchange With Encrypted Public Exponents

Changing The Client-Server Exchange (Second Extension Group)

This group changes the messages exchanged between the client and the server, it also has two possible algorithms presented in A and B below.

A) In this algorithm the steps 1 to 4 of the handshake are kept the same as the original Kerberos. The change occurs in the handshaking between the client and server to get the session key as shown in figure (5). Here the session key is derived using D-H which adds *perfect forward secrecy*, because the session key cannot be compromised when any one of (K_U , $K_{C,TGS}$, $K_{AS,TGS}$, $K_{TGS,S}$,or $K_{C,S}$) is compromised. This is more secure than the previous one due to perfect forward secrecy and due to the fact the session key is only known by the two communicating parties, client and server. Where the role of other parties (AS and TGS) is only authenticating the client. While in previous cases the AS and TGS are involved in the generation of the session keys.

The hash values are used as in subsection 4.1 to achieve message authenticity of g^a and g^b and to acknowledge the reception of g^a.

The computational load on the client is the same as in previous section. But the overhead in computation is now on the server and not on AS as above.

Client Side

- Evaluating g^a.
- Evaluating g^{ab}.

Server Side

- Evaluating g^b.
- Evaluating g^{ab}.

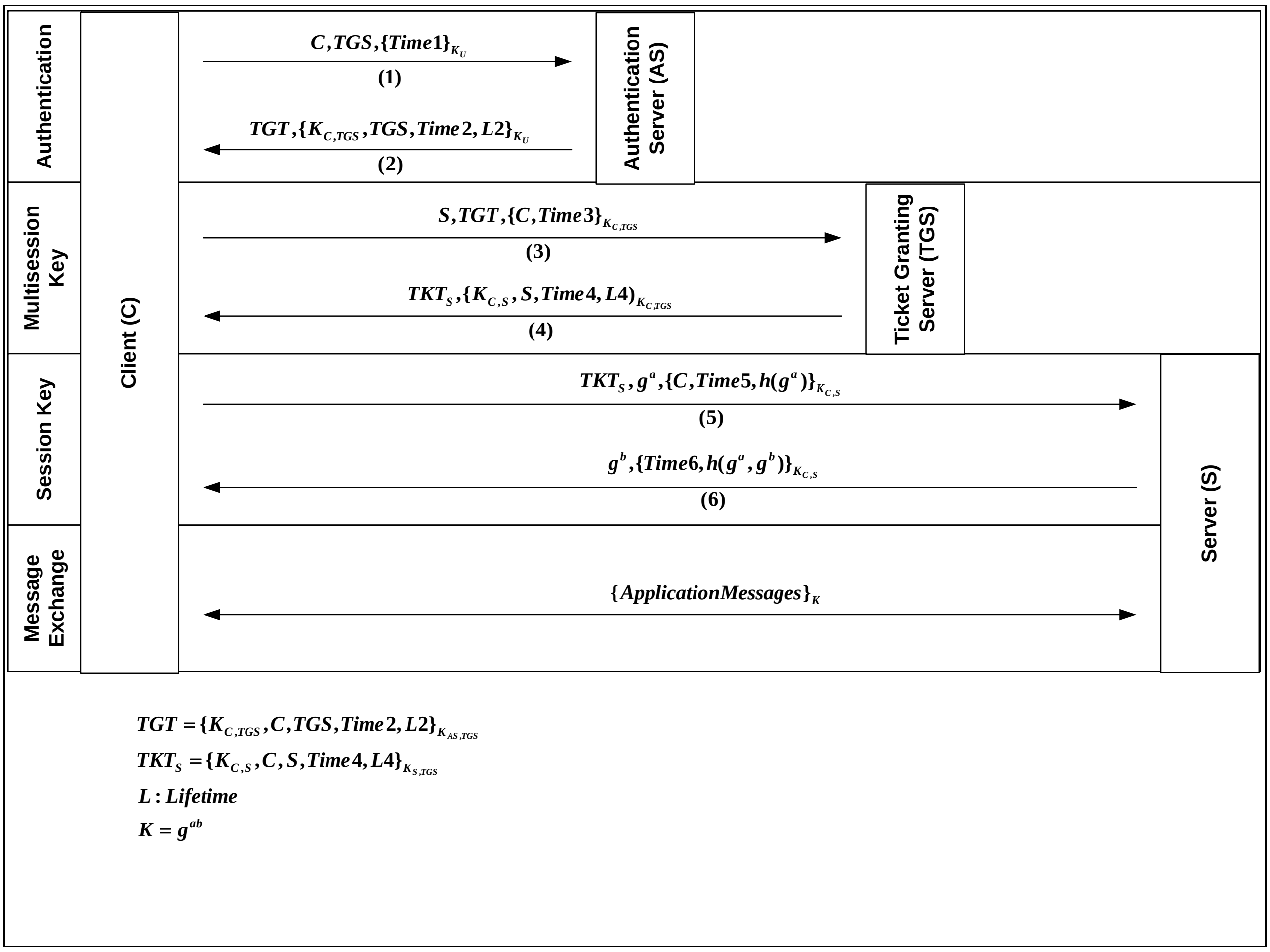
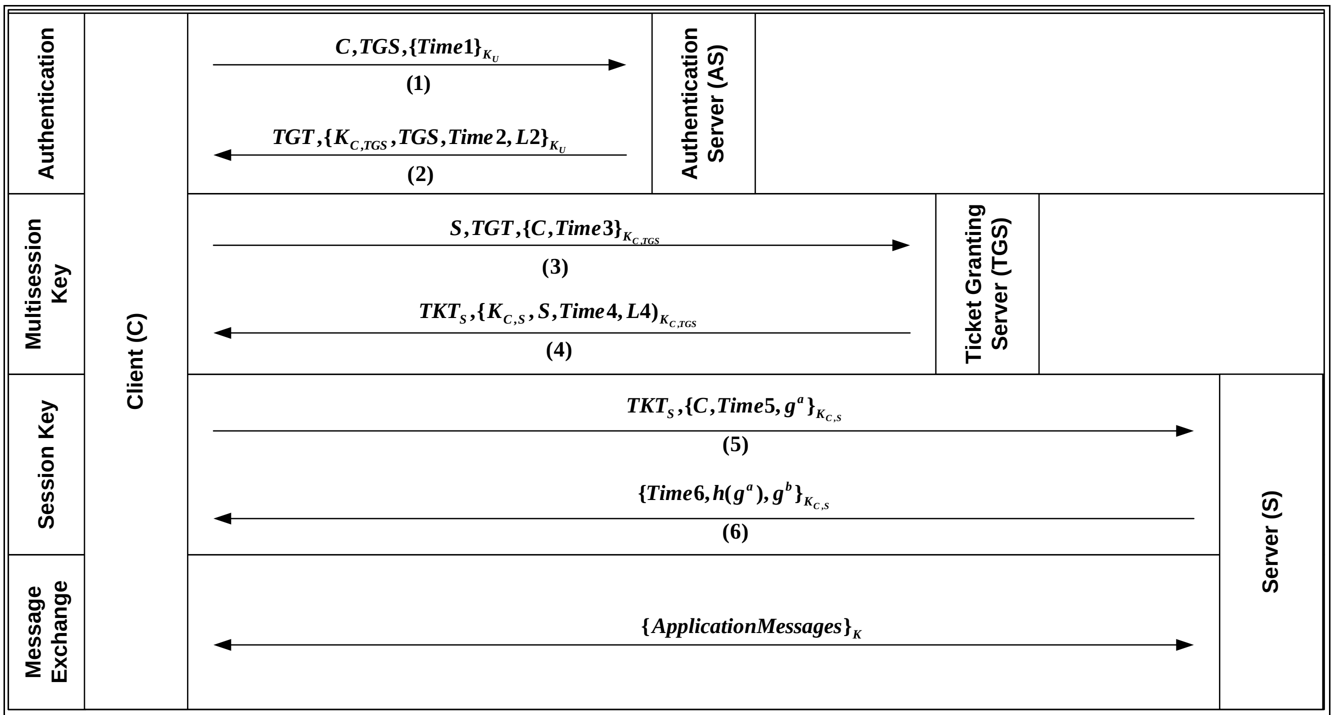

Authentication
Multisession Key
Session Key
Message Exchange
Client (C)
Authentication Server (AS)
Ticket Granting Server (TGS)
Server (S)
$C, TGS, \{Time1\}_{K_U}$
(1)
$TGT, \{K_{C,TGS}, TGS, Time2, L2\}_{K_U}$
(2)
$S, TGT, \{C, Time3\}_{K_{C,TGS}}$
(3)
$TKT_S, \{K_{C,S}, S, Time4, L4)_{K_{C,TGS}}$
(4)
$TKT_S, g^a, \{C, Time5, h(g^a)\}_{K_{C,S}}$
(5)
$g^b, \{Time6, h(g^a, g^b)\}_{K_{C,S}}$
(6)
$\{ApplicationMessages\}_K$
$TGT = \{K_{C,TGS}, C, TGS, Time2, L2\}_{K_{AS,TGS}}$
$TKT_S = \{K_{C,S}, C, S, Time4, L4\}_{K_{S,TGS}}$
L : Lifetime
$K = g^{ab}$

Fig. 5 Client-Server Modified Kerberos Proposed Exchange

B) Simple modifications can be made in a similar way to the modifications in part B of previous subsection 4.1. This is shown in figure (6). The Diffie-Hellman computations are the same as in part A, but the data to be symmetrically encrypted is larger in size which means more computational time. But this time increase is very small when compared with public key operations.


Authentication
Multisession Key
Session Key
Message Exchange
Client (C)
Authentication Server (AS)
Ticket Granting Server (TGS)
Server (S)
$C, TGS, \{Time1\}_{K_U}$
(1)
$TGT, \{K_{C,TGS}, TGS, Time2, L2\}_{K_U}$
(2)
$S, TGT, \{C, Time3\}_{K_{C,TGS}}$
(3)
$TKT_S, \{K_{C,S}, S, Time4, L4)_{K_{C,TGS}}$
(4)
$TKT_S, \{C, Time5, g^a\}_{K_{C,S}}$
(5)
$\{Time6, h(g^a), g^b\}_{K_{C,S}}$
(6)
$\{ApplicationMessages\}_K$

$TGT = \{K_{C,TGS}, C, TGS, Time2, L2\}_{K_{AS,TGS}}$
$TKT_S = \{K_{C,S}, C, S, Time4, L4\}_{K_{S,TGS}}$
L : Lifetime
$K = g^{ab}$

Fig. 6 Client-Server Modified Kerberos Proposed Exchange With Encrypted Public Exponents

COMPARISON BETWEEN FIRST AND SECOND GROUP

To make the comparison the two groups presented in section 4.1 and 4.2, the following assumptions are considered: M clients, N servers, one AS, and one TGS. Figure (7) shows the keys generated for each client. The comparison will be taken from security and performance perspectives.

From A Security Perspective

For the first proposed extension the $K_{C,TGS}$ is generated using D-H, $K_{C,TGS} = g^{ab}$. This key is the base for all the keys for each client ($K_{C1,S1}$, $K_{C1,S2}$,…, $K_{C1,SN}$, and session keys), so the compromise of K_{U1} will not compromise the derived keys because $K_{C1,TGS}$ is not dependent on it. But this extension achieves conditional perfect forward secrecy because $K_{C1,TGS}$ is sent encrypted inside the TGT encrypted under $K_{AS,TGS}$ so the compromise of the later key compromises $K_{C1,TGS}$ and then all other keys ($K_{C1,S1}$, $K_{C1,S2}$,…, $K_{C1,SN}$, and session keys) are compromised. The compromise of $K_{AS,TGS}$ is not always considered a problem since it may be difficult to achieve because it is shared between AS and TGS which can be highly secured. And in many cases these two parties are implemented in the same server which reduces the risk of compromise. A possible enhancement is to periodically change this shared key, thus compromise of $K_{AS,TGS}$ will compromise only the keys created during the period of validity of this key (before it changes). This is not a difficult change since it involves two parties only, AS and TGS.

For the second extension, the session key is the one generated using D-H while all other keys remain as in the original Kerberos, K= g^{ab} . So the compromise of any key (K_U , $K_{C,TGS}$, $K_{AS,TGS}$, or $K_{C,S}$) will not compromise the session key K. This gives perfect forward secrecy.

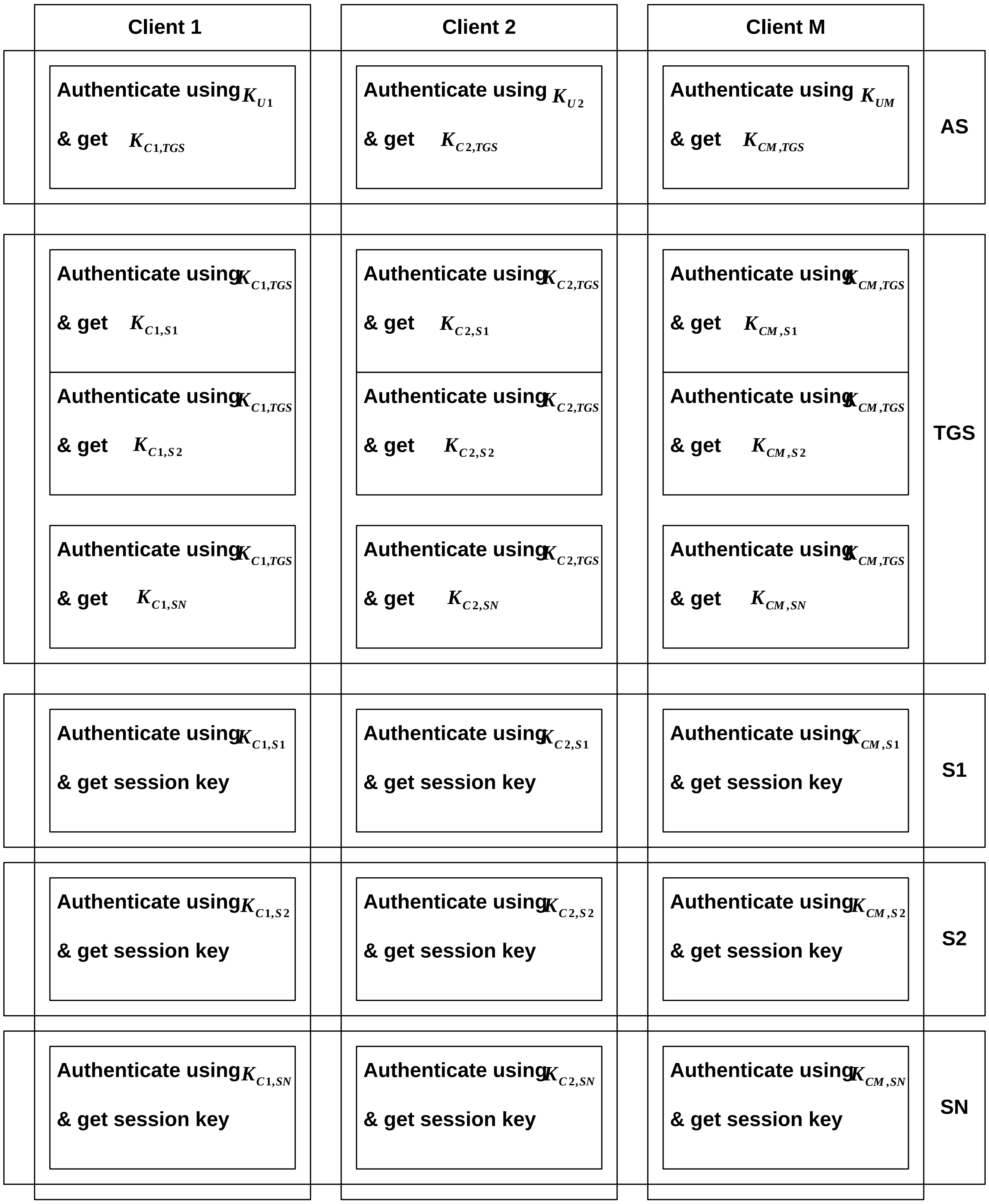

Client 1
Client 2
Client M
Authenticate using K_{U1} & get $K_{C1,TGS}$
Authenticate using K_{U2} & get $K_{C2,TGS}$
Authenticate using K_{UM} & get $K_{CM,TGS}$
AS
Authenticate using $K_{C1,TGS}$ & get $K_{C1,S1}$
Authenticate using $K_{C2,TGS}$ & get $K_{C2,S1}$
Authenticate using $K_{CM,TGS}$ & get $K_{CM,S1}$
Authenticate using $K_{C1,TGS}$ & get $K_{C1,S2}$
Authenticate using $K_{C2,TGS}$ & get $K_{C2,S2}$
Authenticate using $K_{CM,TGS}$ & get $K_{CM,S2}$
Authenticate using $K_{C1,TGS}$ & get $K_{C1,SN}$
Authenticate using $K_{C2,TGS}$ & get $K_{C2,SN}$
Authenticate using $K_{CM,TGS}$ & get $K_{CM,SN}$
TGS
Authenticate using $K_{C1,S1}$ & get session key
Authenticate using $K_{C2,S1}$ & get session key
Authenticate using $K_{CM,S1}$ & get session key
S1
Authenticate using $K_{C1,S2}$ & get session key
Authenticate using $K_{C2,S2}$ & get session key
Authenticate using $K_{CM,S2}$ & get session key
S2
Authenticate using $K_{C1,SN}$ & get session key
Authenticate using $K_{C2,SN}$ & get session key
Authenticate using $K_{CM,SN}$ & get session key
SN

Fig. 7 Key Generation Steps in Environment With M Clients, N Servers, one AS, and one TGS

From A Performance Perspective

The computational load on all parties can be found by locating the public key operations (exponentiations). For the first extension where $K_{C,TGS}= g^{ab}$ and according to the model shown in the figure:

- Each client will evaluate g^a and g^{ab} once for the entire login time, so it sums to 2 operations by each client.
- AS will evaluate g^b and g^{ab} once for each client which means 2M operations.
- All other parties have no comparable computational load.

For the second extension where the final session key between client and server is g^{ab}:

- Each client will evaluate g^a and g^{ab} once for each server, which gives 2N operations by each client.
- AS and TGS have no comparable computations.
- Each server will evaluate g^b and g^{ab} once for each client, then each server has to do 2M operations.

It is clear that the first extension has overall lower computations than the second extension. The first has less load on both the clients and servers, with higher load on the AS. So in general the first case outperforms the second one, keeping in mind that the former gives only conditional perfect forward secrecy, while the later gives perfect forward secrecy. But there are three main points that must be taken into account.

The first point that must be considered is that the first extension has a high load on the AS, but in the second extension the load is the same as in original Kerberos, and it is negligible when compared with public key operations. The high load on AS can be a serious problem in some situations, because AS may become a bottleneck. This case can happen on the system startup, where all clients try to authenticate themselves to AS at the same time.

The second point is the fact that not all clients will communicate with all servers. So the load in first extension is mainly on AS and is highly constant at 2M. While in second extension the load is mainly on clients and servers. The load on client depends on its communication demand, i.e. load increases linearly with the number of server it needs to connect. The load on server depends on the clients' demands, i.e. load increases linearly with the number of clients needing to communicate with that server. And in general it is not likely that these communication requests occur simultaneously, i.e. the requests are generally randomly distributed in time and among servers. As opposed to the requests to AS which may happen simultaneously. This distribution gives a reduced effect on the servers at any moment.

The last point is that the second group of extensions gives the clients and servers the flexibility to agree to use the extension or stay in the original Kerberos mode. This is a per session decision between client and server to achieve perfect forward secrecy or to achieve higher performance with lower security.

CONCLUSIONS AND FUTURE WORK

Throughout this work, analysis of Kerberos has been made in its original and PKINIT forms. The analysis focuses on the concept of perfect forward secrecy and on the performance of the algorithms. Two extensions have been proposed to achieve perfect forward secrecy without changing the authentication based on passwords, which differentiates them from PKINIT.

The two extensions proposed are similar in the basic idea that they use Diffie-Hellman to achieve the goal, but they differ in the steps where the change is done. The first extension makes the change at the Client-AS phase and achieves conditional perfect forward secrecy while the other extension achieves perfect forward secrecy by changing the last phase. In general, the first one outperforms the second, while the second has better security.

There are many implementation details that are out of the scope of this work, but are important to discuss. These details include the size of the keys used and the software implementation of the operations in addition to other optimizations in the cryptographic operations.

Another issue to be investigated is how to expand the Kerberos system to be used over larger networks that are separated by firewalls and proxies, a problem that original Kerberos suffers from.

An interesting issue is to develop a system that integrates several authentication systems based on different cryptographic operations, such as the certificate authorities, SSL, and Kerberos.

REFERENCES

- Cervesato, I., Jaggard, A. D., Scedrov, A., Tsay, J.-K. and Walstad, C., 2008, "*Breaking and Fixing Public-Key Kerberos*", Information and Computation, Volume 206, Issue 2-4, Pages 402-424.

- Diffie, W. and Hellman, M.E., November 1976, "*New Directions in Cryptography*". IEEE Transactions on Information Theory, Vol. IT-22, No. 6, pp. 644-654.

- Menezes, A., van Oorschot, P., and Vanstone, S., 1996, *"Handbook of Applied Cryptography"*, CRC Press.

- Miller, S., Neuman, C., Schiller, J., and Saltzer, J., December 21, 1987. *"Kerberos Authentication and Authorization System"*, M.I.T. Project Athena, Cambridge, Massachusetts.

- Ozkan, M., 2003, *"High-speed ECC based Kerberos Authentication Protocol for Wireless Applications"*, Global Telecommunications Conference. GLOBECOM '03. IEEE, vol. 3, pp. 1440-1444.

- Sirbu, M., and Chuang, J., 1997, "*Distributed Authentication in Kerberos Using Public Key Cryptography"*. Symposium on Network and Distributed System Security. San Diego, California: IEEE Computer Society Press..

- William E., 2008, *"A New Hash Competition"*, IEEE Security and Privacy, vol. 6, no. 3, pp. 60-62.

- Wu, T., February, 1999, "*A Real-World Analysis of Kerberos Password Security*", Proceedings of the 1999 Internet Society Network and Distributed System Security Symposium, San Diego, CA.

- Zhu, L., and Tung, B., 2006, "*Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)*". IETF RFC: 4556.

- Zhu, L., Jaganathan, K., and Lauter, K., 2008, *"Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)"*, IETF RFC: 5349.