Fast Re-Authentication Protocols for Mobile and Wireless Networks

Cheng Lai Low, Peter Bertok
School of Computer Science & IT, RMIT University, Melbourne, Australia
Contact: pbertok@cs.rmit.edu.au

Abstract : **Authentication in mobile environments has to be performed frequently. Full authentication is time consuming, and repeated execution adversely effects network performance. A solution to accelerate the process is the use of an initial full authentication followed by less expensive re-authentications. This paper proposes two authentication protocols with fast re-authentication: one for users who re-connect to the same network access point after temporary disconnections and another protocol that accelerates handoff for users changing access points.**

1. Introduction

User mobility, in wireless environments in particular, poses many security related challenges. Firstly, wireless networks are easy to penetrate. Authenticating users at every connection attempt is the first line of defence, but the number of connection requests can be very high, as wireless users switch between network access points (handoff) and the connection can be intermittent due to weak signals. This means that authentication can be very expensive, since comprehensive protocols are computationally intensive and often use complex cryptographic algorithms. Secondly, limited bandwidth and noisy channels with frequent re-transmission of messages result in reduced performance. Mobile users, however, expect services comparable to those available in wired environments, and security also needs to be on a similar level. At present, secret-key security protocols are predominant in wireless networks because of their lower computational complexity, but the degree of security they offer may not always be sufficient.

This paper proposes an efficient solution for reliable authentication of users to address these problems.

2. Authentication

2.1. Authentication in mobile environments

Early authentication methods focused on connecting roaming mobile phone users to networks. The network needed to ensure that only valid users have access to its services and the users access a secure facility, as security lapses in a network can lead to permanent damage to a visiting user. The main aims were establishing a session key for confidential communication, mutual authentication and non-repudiation [1]. Solutions also looked at maintaining privileges while moving to another network [10].

Computer networks face similar problems, but additional security challenges are presented by the tools available for intruders on the wired side of the network. For performance reasons, the solutions used secret key cryptography [8], or the RSA algorithm with short keys [1], which resulted in lower level of security. Notwithstanding this disadvantage, it was a necessary compromise.

To improve security, a re-authentication approach was proposed in [8]. After sending a certain number of data packets, the client had to be re-authenticated via a faster re-authentication method. The method had low computational complexity and provided limited security due to the use of a secret key algorithm. The timing of re-authentication was also problematic, long established uninterrupted connections incurred a re-authentication overhead unnecessarily, and sessions with low traffic volume were not re-authenticated for a considerable amount of time, if ever.

Another approach was using thorough initial authentication followed by a cheaper re-authentication e.g. when a client was temporarily disconnected [6]. The original re-authentication method was improved later [7]. The protocol was based on secret key cryptography, and in the protocol the key between the communicating partners was re-used in the re-authentication process until it expired after a given time. This was a security risk, as frequent re-authentication can provide large amount of data for analysis by an adversary. More importantly, however, the protocol did not facilitate fast handoff for users moving from network access point to the next.

Recent proposals include schemes where the user authenticates to several neighbouring nodes before actual handoff [13]. This method improves response time, but also increases network traffic and puts greater computational load on the client.

2.2. Main issues

The main constraints on mobile applications are imposed by limited battery power and limited computing resources. The largest consumer of battery power is transmission, therefore the number of messages sent by the mobile node should be kept low, and the messages should be short. Limited processing power is becoming less of an issue, as (i) hardware keeps improving and (ii) mobile applications that connect to a fixed network can shift part of the computational load to the more powerful fixed nodes. While providing lower level of security, secret key encryption algorithms are faster than public-key methods. However, newer and faster algorithms, such as elliptic curve cryptography, have extended the use of public key cryptography [4], and provide increased security with lower computational load. A recent performance analysis of a public key enabled Kerberos reported its usage to be viable in mobile applications [11].

A remaining problem is that many cryptographic protocols rely on the use of timestamps and require

clock synchronisation, a difficult task in mobile environments.

Our aim is fast and secure authentication in mobile and wireless networks. Users with frequent disconnections e.g. due to a weak signal should not go through expensive authentication procedures. Likewise, authenticated users moving form one network access point to the next should experience a fast and smooth handoff. In this paper we present two authentication/re-authentication protocols, one for fast reconnection to the same server, and a variant of it with single-sign-on to help fast handoff. They show similarities to existing protocols, such as the Neuman-Stubblebine protocol [6], but use different message structure, and employ public key encryption and additional message authentication codes to avoid the pitfalls.

3. Proposed Solution

3.1. Protocol Features

The two protocols proposed here provide high level of security by using public-key cryptography. In the protocols, messages are sent directly to client *A* by servers other than *B* (the one *A* wants to communicate with). In practice, these messages can be relayed by server *B*, but *B* will not process them. The central authentication server, denoted here by *KDC* for historical reasons [3], is a single entity for PKNBP, while it has two components, an initial authentication server (*AS*) and a ticket granting server (*TGS*) linked together, for PKNBS.

3.2. Public key nonce-based protocol (PKNBP)

The protocol proposed by Neuman and Stubblebine [6] does not depend on the presence of synchronized clocks. Their work is an extension of the nonce-based mutual authentication protocol proposed by Kehne et al. [5]. We have improved the nonce-based protocol and combined it with public key cryptography. We refer to this enhancement as Public Key Nonce-Based Protocol (PKNBP). The formal representation of our protocol is illustrated in Figure 1.

Notation:

A	Principal A
B	Principal B
KDC	Key Distribution Center
$\mathbf{N_A}$	Nonce of A
$\mathbf{N_B}$	Nonce of B
$\mathbf{K_{Rand}}$	Randomly generated session key
$\mathbf{K_{PrivA}}$	Private key of A
$\mathbf{K_{PubB}}$	Public key of B
$\mathbf{K_{PrivB}}$	Private key of B
$\mathbf{K_B}$	Secret key of B
$\mathbf{K_{PubK}}$	Public key of KDC
$\mathbf{K_{AB}}$	Secret session key between A and B
$\mathbf{T_B}$	Expiration time specified by B
HMAC	Hashed Message Authentication Code

Participants	Message	Notes
1. A ⇨ B	A, N_A, $\{\{K_{Rand}\}K_{PrivA}\}K_{PubB}$	A generates a random key and signs it with its private key. The message is encrypted with B's public key. A then sends to B the message and its identity together with a nonce generated by A.
2. B ⇨ KDC	B, $\{\{\{K_{Rand}\}K_{PrivA}$, A, N_A, T_B, K_{AB}, HMAC(A, B, K_{AB}, T_B, K_B) $\}K_{PrivB}$ $\}K_{PubK}$, N_B	B sends to the KDC its identity, a nonce generated by B and instructions to the KDC to issue credentials to A signed with its own private key and encrypted with KDC's public key.
3. K ⇨ A	$\{B, N_A, K_{AB}, T_B, N_B\}K_{Rand}$, {A, K_{AB}, T_B, HMAC(A, B, K_{AB}, T_B, K_B) $\}K_{PubB}$	KDC encrypts the identity of B, A's nonce, B's nonce, the session key and the expiration time specified by B with the random secret key generated by A. It sends this to A together with the ticket.
4. A ⇨ B	{A, K_{AB}, T_B, HMAC(A, B, K_{AB}, T_B, K_B) $\}K_{PubB}$, $\{N_B\}K_{AB}$	A forwards the ticket to B together with B's nonce encrypted in the session key.

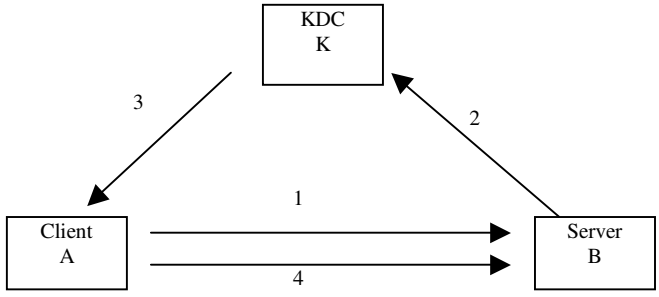

Figure 1: Public key nonce-based protocol (PKNBP)

The protocol begins with principal *A* initiating authentication by sending a message to *B* containing *A*'s identity together with a nonce generated by *A* and a randomly generated key signed using *A*'s private key, encrypted with *B*'s public key. Upon receiving the request, *B* decrypts the part of the message encrypted with its public key. *B* then sends a message containing instructions, its own identity, and a nonce generated by *B* to the *KDC*. The instructions specify *A* as the intended recipient, suggest an expiration time, and include the nonce received from *A*, the session key K_{AB} and the randomly generated session key signed with *A*'s private key. The instructions are signed with *B*'s private key and encrypted with *KDC*'s public key. The *KDC*, upon receipt of the message, decrypts the instructions with its private key and *B*'s public key, thus verifying that it is indeed from *B*. The *KDC* also needs to retrieve the randomly generated key from *A* by decrypting it with *A*'s public key. The *KDC* then issues a ticket to A together with a message containing the identity of *B*, *A*'s nonce, the session key, the expiration time suggested by *B* and *B*'s nonce encrypted in *A*'s randomly generated secret key. *A* decrypts that part of the message from the *KDC* encrypted with its randomly generated secret key and verifies the nonce is the same as the one included in message 1. If the nonce is correct, *A* is assured that the *KDC* received the nonce from *B*. Furthermore, *A* knows *B*'s nonce and can use it to prove its identity to *B*. Finally, *A* forwards the ticket to *B* together with *B*'s nonce encrypted with the session key. *B* decrypts the ticket to retrieve the session key and verifies that the nonce it received is indeed the one it sent out in message 2.

Note that the ticket contains a Hashed Message Authentication Code (*HMAC*) of *A*, *B*, K_{AB} and T_B encrypted with the secret key of *B*. The purpose of the *HMAC* is to enable fast re-authentication for a client who already has a valid ticket. The ticket contains a secret that only *B* can possibly know; a *HMAC* of *A, B,* K_{AB} and T_B calculated with a secret key of *B*. Hence, there is no way to forge a valid ticket as the attacker will not know *B*'s secret key. So when *A*, who has a valid ticket from *B*, sends it to *B*, *B* is able to verify the validity of the ticket. It then issues a new ticket to *A* together with a new session key and a newly generated nonce encrypted in the old session key. *A* then verifies that the nonce it receives is correct and proves its authenticity by sending back the new nonce generated by *B* encrypted with the new session key.

The formal representation of the fast re-authentication process is illustrated in Figure 2.

Notation:

A	Principal A
B	Principal B
N_A'	Nonce of A (N_A' ? N_A)
N_B'	Nonce of B (N_B' ? N_B)
K_{PubB}	Public key of B
K_B	Secret key of B
K_{NEW}	Randomly generated session key (New)
K_{AB}	Secret session key between A and B
T_B	Expiration time specified by B
HMAC	Hashed Message Authentication Code

Participants	Message	Notes
1. A ⇨ B	N_A', {A, K_{AB}, T_B, HMAC(A, B, K_{AB}, T_B, K_B) }K_{PubB}	A sends to B a newly generated nonce together with a valid ticket.
2. B ⇨ A	N_B', {A, K_{NEW}, T_B, HMAC(A, B, K_{NEW}, T_B, K_B) }K_{PubB}, {N_A', K_{NEW}}K_{AB}	B sends to A a newly generated nonce together, a new ticket and encrypts the new session key and A's nonce with the old session key.
3. A ⇨ B	{N_B'}K_{NEW}	A sends B' nonce encrypted with the new session key back to B.

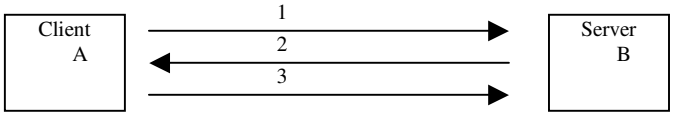

Figure 2: Fast re-authentication with PKNBP

Public key cryptography helped to authenticate the messages in the initial authentication phase. In step 1 only *A* can produce the message and only *B* can decrypt it; in step 2 only *B* can produce the message and only *K* can decrypt it. The key K_{AB} used for communication between *A* and *B* is also safely forwarded; only a successful attack on the *KDC* can compromise it. But even in that case, only the secret keys are compromised, *A* and *B* can keep using their own public-private key pairs.

Replay attacks do not work here. The message in step 1 of the initial authentication phase will not yield anything to an attacker, as the information coming back in step 3 is encrypted with a random key that only *A*, *B* and the *KDC* know. In the re-authentication phase of

the protocol *B* protects itself by calculating a *HMAC* that includes the value of the new key.

The computational load on *A* is low, it encrypts with public-key cryptography only in the very first message; the ticket it forwards in step 4 was encrypted with a public key by the *KDC*. The generation of secret keys for communication is performed by *B*, which not only lessens the load on *A*, but also provides additional protection for *B*. Using public-private keys of *A* and *B* instead of the previous secret key K_{AB} in step 2 of the re-authentication phase of the protocol could provide additional security, but it would increase the computational load. The messages transmitted by *A* are also kept short.

3.3. Public key nonce-based protocol with single sign-on (PKNBPS)

Our second protocol resolves the issue of single sign-on by using tighter coupling between participants. We shall refer to the second protocol as Public Key Nonce-Based Protocol with Single Sign-On (PKNBPS). The formal representation of our protocol is illustrated in Figure 3.

Notation:

A	Principal A
B	Principal B
AS	Authentication Server
TGS	Ticket Granting Server
$\mathbf{N_A}$	Nonce of A
$\mathbf{N_B}$	Nonce of B
$\mathbf{K_{PubA}}$	Public key of A
$\mathbf{K_{PubB}}$	Public key of B
$\mathbf{K_{PrivB}}$	Private key of B
$\mathbf{K_B}$	Secret key of B
$\mathbf{K_{PrivAS}}$	Private key of AS
$\mathbf{K_{PubTGS}}$	Public key of TGS
$\mathbf{K_{C,TGS}}$	Secret session key between client and TGS
$\mathbf{K_{AB}}$	Secret session key between A and B
$\mathbf{TKT_{C,TGS}}$	Ticket (client to TGS)
$\mathbf{T_B}$	Expiration time specified by B
HMAC	Hashed Message Authentication Code

Participants	Message	Notes
1. A ⇨ AS	A, TGS request	A request ticket from AS to communicate with Ticket Granting Service (TGS).
2. AS ⇨ A	$\{\{K_{C,TGS}\}K_{PrivAS}\}K_{PubA}$, $\{TKT_{C,TGS}\}K_{PubTGS}$	Key to communicate with TGS encrypted with the A's public key and ticket to communicate with TGS encrypted with TGS's public key. ($TKT_{C,TGS}$ includes $K_{C,TGS}$, the key the client uses to communicate with TGS).
3. A ⇨ B	A, N_A, $\{TKT_{C,TGS}\}K_{PubTGS}$	A sends the TGS ticket to B together with its identity and nonce.
4. B ⇨ TGS	B, N_B, $\{TKT_{C,TGS}\}K_{PubTGS}$, $\{\{A, N_A, T_B, K_{AB}, HMAC(A, B, K_{AB}, T_B, K_B)\}K_{PrivB}\}K_{PubTGS}$	B sends to the TGS its identity, a nonce generated by B, the TGS ticket from A and instructions to the TGS to issue credentials to A signed with its own private key and encrypted with KDC's public key.
5. TGS ⇨ A	$\{B, N_A, K_{AB}, T_B, N_B\}K_{C,TGS}$, $\{A, K_{AB}, T_B, HMAC(A, B, K_{AB}, T_B, K_B)\}K_{PubB}$	TGS encrypts the identity of B, A's nonce, B's nonce, the session key and the expiration time specified by B with the client-TGS session key. It sends this to A together with B's ticket.
6. A ⇨ B	$\{A, K_{AB}, T_B, HMAC(A, B, K_{AB}, T_B, K_B)\}K_{PubB}$, $\{N_B\}K_{AB}$	A forwards the ticket to B together with B's nonce encrypted in the session key.

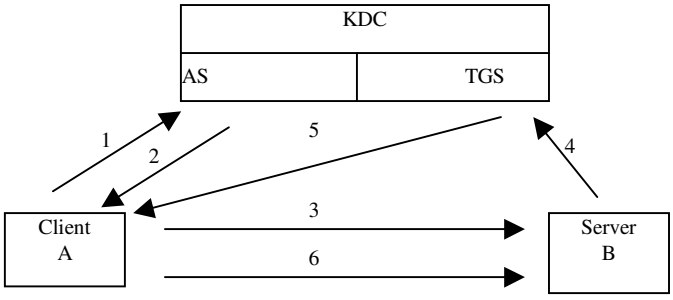

Figure 3: Public key nonce-based protocol with single sign-on (PKNBPS)

Step 1 of the protocol is the same as Kerberos [3]; *A* sends a *TGS* request to the *AS*. The *AS* returns the *TGT* to *A*, together with a session key with the *TGS* signed with the *AS*'s private key and encrypted with *A*'s public key. This authenticates *A*, as its private key is needed for decryption. From step 3 onwards, the flow of the protocol is similar to PKNBP with only some slight variations. In step 3, *A* sends his identity, nonce and *TGS* ticket to **B**. *B* upon receiving the request constructs a message containing the identity of *B*, *B*'s

nonce, the session key K_{AB}, the *TGS* ticket from *A* and instructions to the *TGS* to issue credentials to *A*. In PKNBP, *B* has the autonomy to choose the *KDC* to which it wants to send the message. However, in PKNBPS, *B* has to send the message to the *TGS* for which *A* was issued a ticket in step 2. Hence, there is a greater coupling in the protocol. After processing the message from *B*, the *TGS* encrypts the identity of *B*, *A*'s nonce, *B*'s nonce, the session key and the expiration time specified by *B* with the client-*TGS* session key. It sends this to *A* together with *B*'s ticket. *A* completes the authentication process by sending *B*'s ticket to *B* together with *B*'s nonce encrypted with the session key.

Authentication to a different server means repeating steps 3 - 6. A re-authentication of *A* to *B* after a disconnection uses the same steps as the PKNBP protocol described in the previous section. To avoid unauthorised access via an intruder generating a bogus message 3, the ticket to *TGS* ($TKT_{C,TGS}$) must not be forgeable. Replaying an old message in step 3 would help an intruder only if it can get hold of the corresponding key for communication between *TGS* and *A* ($K_{C,TGS}$).

In the protocol client *A* performs the expensive public key operations only in the second step of the protocol when it communicates with the *AS*. The subsequent authentication of *A* to *B* via *TGS* imposes no heavy load on *A*, as *TGS* and *B* are performing the public key encryptions/decryptions.

When connecting to another server, the protocol follows the Kerberos method, and client *A* has to present the *TGS* ticket to the new server. No public key encryptions / decryptions are performed by *A* at this time; these operations have been shifted to the new server and to the *TGS*.

3.4. Major advantages of PKNBP and PKNBPS

The major advantages of our protocols are as follows.

- *Higher level of security* is achieved via the use public key cryptography.
- *Fast re-authentication* is supported after a temporary disconnection in a session.
- *Low computational load on mobile client,* as it performs a single public key encryption/decryption at the beginning of the protocol, and the more computationally intensive operations have been offloaded to the server.
- *Low number of messages* sent over the wireless network link.
- *Tolerance of loosely synchronized clocks* between interacting principals in our protocols. In PKNBS, however, the clocks of *AS* and *TGS* have to be synchronised.
- *Intrusion tolerance:* the *KDC* stores only the public keys of its principals. If the *KDC* is compromised, only the certificate of the *KDC* has to be revoked.
- *Fully distributed architecture:* each principal in the protocol can operate autonomously. The client has the freedom to choose which replicated server it wishes to connect to and the server can decide which *KDC* it would like to use. However, in PKNBPS, the server is required to connect to a specific *TGS* as the client holds a *TGT* specific to a particular *TGS*.

3.5. Formal verification of the proposed methods

Due to limited space, formal verification of the proposed protocols is not presented here; please see [12] for a verification that uses the Communicating Sequential Processes (CSP) framework [9].

4. Protocol Performance

4.1. Operations

The public/private key operations that require the most computing resources are summarized in Table 1.

Protocol	A	B	AS / KDC	TGS
PKNB authentication	Encryption: 2 Decryption: 0	Encryption: 2 Decryption: 2	Encryption: 1 Decryption: 3	N/A
PKNBS	Encryption: 0 Decryption: 2	Encryption: 2 Decryption: 1	Encryption: 3 Decryption: 0	Encryption: 1 Decryption: 3
PKNB, PKNBS Re-authentication to B	0	Encryption: 1 Decryption: 1	0	0
PKNBS Authentication to a different server	0	Encryption: 2 Decryption: 1	0	Encryption: 1 Decryption: 3

Table 1: Number of public/private key operations

Client *A* has to perform only two encryptions or decryptions in the initial authentication, and subsequent re-authentications to the same server or to another server do not require further public key operations from *A*. When authenticating to a different server in PKNBS, there is one less encryption than decryption operations, as *A* can re-use the ticket received from *AS* in step 2 of the protocol.

4.2. Measurements

The protocols were implemented and tested on a single Pentium workstation running the client, server and *KDC* concurrently. The conditions of the wireless environment such as bandwidth and high error rates were simulated in software. The key concept demonstrated by the performance measurements is the benefit of fast re-authentication in a wireless environment.

Figure 4 shows the average re-authentication response time of PKNBP and PKNBPS after network disconnections. In our simulation, we only considered errors which would lead to a disconnection in the wireless network environment; other errors are deemed

to be rectifiable by the network itself and hence, do not have a direct impact on the operation of the protocols. We ranged the probabilities of disconnections from ten to fifty percent. Fast re-authentication reduced response time to about one-tenth of a full authentication. PKNBP and PKNBPS scale well: fast re-authentication response time increases slower than full authentication when the probability of network disconnections increases. In a wireless environment that has poor connections and high error rates, the gentler gradient of fast re-authentication protocols is particularly advantageous.

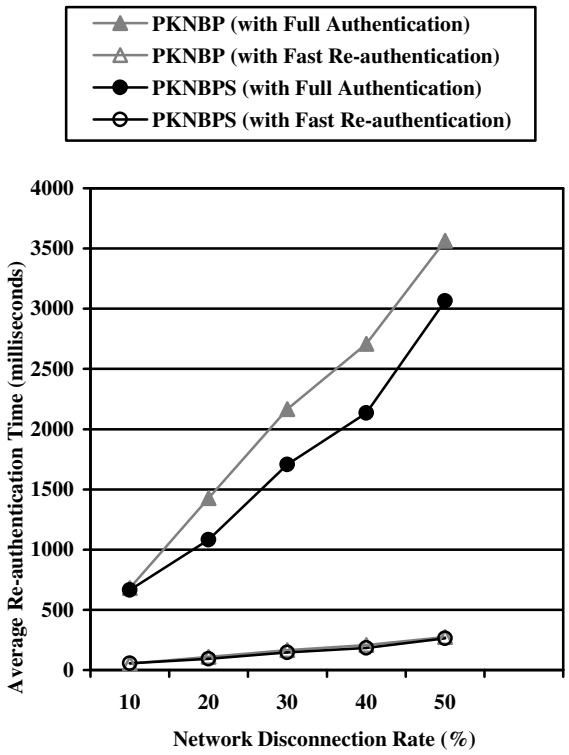

Figure 4: Full authentication times and fast re-authentication times

Without fast re-authentication, PKNBPS performs slightly better than PKNBP at higher disconnection rates. However, with fast re-authentication, the performance of PKNBPS and PKNBP are the same. This is not surprising, as both protocols use the same re-authentication mechanism.

5. Summary

In this paper, we presented two protocols for the fast re-authentication of mobile clients. The protocols use public key encryption for higher security, while keeping the mobile client's computational load low by shifting some of its load to the servers. The protocols use a low number of authentication messages between user and network and the client has to transmit only short messages, which saves battery power. The protocols can tolerate loose synchronisation of clocks. They also operate in a distributed fashion, the client can choose to connect to any of the servers.

Performance measurements indicated that the proposed protocols offer good scalability and performance, and are suitable for practical applications.

Acknowledgements

This research was partially supported by Sun Microsystems grant no. 7832-030217-AUS.

References

[1] H.Y.Lin, L.Harn: Authentication Protocols for Personal Communication Systems, ACM SIGCOMM Computer Communication Review, Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication October 1995, Volume 25 Issue 4

[2] M. A. Sirbu, J. Chuang. Distributed Authentication in Kerberos Using Public Key Cryptography. Internet Society 1997 Symposium on Network and Distributed System Security, February 1997.

[3] J. Kohl, C. Neuman. The Kerberos Authentication Service (v5). Internet RFC 1510, September 1993.

[4] B.Schneier: Applied Cryptography, Wiley 1996

[5] A. Kehne, J. Schonwalder and H. Langendorfer. Multiple Authentications with a Nonce-Based Protocol Using Generalised Timestamps. Operating systems Review, 26(4):84-89, October 1992.

[6] B. C. Neuman, S. Stubblebine. A Note on the Use of Timestamps as Nonces. Operating Systems Review, 27(2):10-14, April 1993.

[7] T.Hwang, N.Y.Lee, C.M.Li, M.Y.Ko, Y.H.Chen: Two Attacks on Neuman-Stubblebine Authentication Protocols, Information Processing Letters, 53:103-107, 1995

[8] A.Ganz, S.H.Park, Z.Ganz: Robust Re-authentication Method and Key Exchange Protocol for IEEE802.11 Wireless LANs, IEEE Military Communications Conference, October 1998

[9] P. Ryan, S. Schneider. The Modelling and Analysis of Security Protocols: the CSP Approach. First Ed. December 2000. Addison-Wesley.

[10] R.Molva, D.Samfat, G.Tsudik: An Authentication Protocol for Mobile Users, IEE Colloquium on Security and Cryptography Applications to Radio Systems, 1994

[11] A. Harbitter, D. A. Menascé. The Performance of Public Key-enabled Kerberos Authentication in Mobile Computing Applications. (ACM) Conference on Computer and Communications Security:78-85, 2001.

[12] C.L.Low: Client Server Authentication in Mobile Environment, Masters thesis, December 2002, RMIT University, Melbourne, Australia

[13] Sangheon Pack and Yanghee Choi, "Pre-Authenticated Fast Handoff in a Public Wireless

LAN based on IEEE 802.1x Model, IFIP TC6 Personal Wireless Communications 2002, Singapore, pp.175-182, October 2002