A Multi-Party User Authentication and Key Agreement Protocol Based on Public Key Cryptosystem

Sushma Yalamanchili
Professor & Head, Department of CSE, V.R.Siddhartha Engineering College, Vijayawada.

M.Kameswara Rao
Lecturer, P.B.Siddhartha College, P.G.Centre, Vijayawada.

Ch.Smitha
Lecturer, P.B.Siddhartha College, P.G.Centre, Vijayawada.

***Abstract*—In this paper, we address the problem of multi-party user authentication and key agreement whereby a party gains assurance of the identity of other parties involved in a protocol for preventing impersonation and unauthorized access. Our scheme utilizes the idea of Buttyan, Nagy proposed multi-party challenge -response protocol. The proposed protocol allows each participant to directly verify all the other parties that were alive during the protocol run. Our protocol uses minimum number of messages required to solve the multi-party entity authentication problem. Specifically, we construct a multiparty secret key generation scheme which employs simple XOR operations. The authenticity of the protocol is assured by a digital signature scheme. Security attributes of our protocol are presented and analyzed as well.**

***Keywords*—authentication, public key cryptography, nonce, key agreement.**

I. INTRODUCTION

There are numerous occasions on the Internet where authentication protocols are required to identify participants. Several protocols have been proposed for mutual authentication [1, 2]. Of these, the best known is Needham-Schroeder-Lowe(NSL-public key protocol [3, 4]. The NSL protocol satisfies even the strongest forms of authentication, and has been studied extensively [5] .The operation of the three-message base protocol is as follows

1) $A \rightarrow B$: $E_{KUB}[n_A, A]$
2) $B \rightarrow A$: $E_{KUA}[n_A, n_B, B]$
3) $A \rightarrow B$: $E_{KUB}[n_B]$

Figure 1 The Needham-Schroeder-Lowe Protocol With Public Keys.

In the first step, the initiator 'A' of the protocol, generates a nonce n_A and encrypts this value along with his identity using the public key (KUB) of the responder 'B'. In the second step the responder, after receiving the message in step 1, generates his own nonce value n_B . He encrypts both nonce's n_A, n_B along with his identity using the public key of the initiator. In the third step, the initiator sends back the random value n_B to the responder, encrypted by the public key of the responder. Similar protocols, such as the Needham-Schroeder (NS for short) private-key protocol and the Bilateral Key Exchange protocol, have the same underlying structure as the NSL protocols were designed for two parties who want to authenticate each other, which is often referred to as bilateral authentication. In Modern communication there are three or more parties that need to authenticate each other. In such a setting we could naively instantiate multiple bilateral authentication protocols to mutually authenticate all partners. For n parties, such mutual authentication would require $(n \times (n-1))/2$ instantiations of the protocol, and three times as many messages. In practice, when multi-party authentication protocols are needed, protocol designers instead opt to design new protocols that require fewer handshakes. Any n-party challenge-response protocol for entity authentication, in which each party authenticates every other party, uses at least $2n - 1$ messages[6].Our work presents novel multi-party authentication protocols, in which a party authenticates all the other parties during the protocol run. For n parties, the proposed communication structure consists of $2n - 1$ messages, which turns out to be the optimal message complexity.

Message passing structure

Before going into the details of our protocols, we describe the message passing structure. If *A* authenticates every other party *B,C, . . .* in the protocol, then each of these parties, or more precisely vertices that are labeled with their names, must be traversed by a directed path starting from and ending in a vertex that is labeled with *A* as shown in figure 2.

Figure 2 Directed Path

Manuscript received October 5, 2009
Manuscript revised October 20, 2009

Similarly If *B,C,* . . . also authenticate every other party in the protocol, then there is a similar path for *B,C,* . . . as well .We obtain the protocol with the least number of messages by maximally overlapping these paths[6]. The resulting protocol graph has exactly $2n - 1$ edges as shown in figure 3

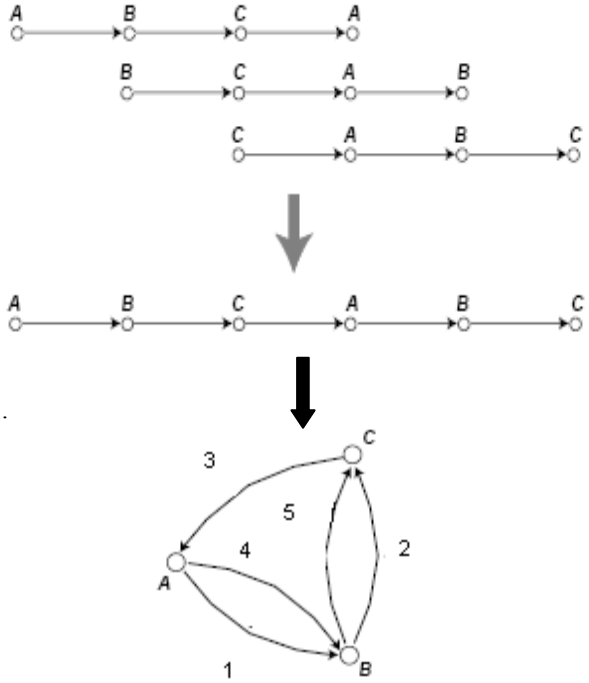

Figure 3 Message passing structure

II. OUR PROPOSED PROTOCOLS

Two multiparty authentication and key agreement protocols were proposed using the public key cryptosystem where each of the *n (n* ≥ 2*)* participating parties proves its identity to each of the other parties and securely exchange a shared secret key with simple XOR operations. First we describe the three-party versions of our protocols in detail and then extend it to the n-party version.

Notations

n_a -nonce /random value generated by user 'a'
ID_a -Identity of user 'a'
KUa -Public Key of user 'a'
KRa -Private Key of user 'a'
E_{KUa} [x] -Encryption of message x by public key of user 'a'-KUa
E_{KRa} [y] -Encryption of message y by private key of user 'a'-KRa
P_a -Participant /Party 'a'

Protocol 1:

Principle: Each participant generates an unpredictable random number, called nonce. Nonce values are passed around among the protocol participants in a circulating message. Each participant that receives the message and sees the challenges that the message contains includes its identity in the message before passing it further to the next participant. When a nonce gets back to the participant that generated it, the message contains the list of those participants that saw the nonce and forwarded the message. These forwarding participants have been alive during the protocol run.

Assumptions: We assume that each participant in the system share their public key with other participants. We also assume that participants trust each other for executing the protocol honestly. In particular, each participant must be trusted for correctly attributing a received message to its sender and faithfully copying all the relevant fields of the received message into the message that is passed further.

Messages of the three-party version:

1)A → B :E_{KUB} [E_{KRA} [n_A , ID_A]]
2)B → C :E_{KUC} [E_{KRB} [n_B , ID_B , n_A, ID_A]]
3)C → A :E_{KUA} [E_{KRC} [n_C,ID_C,n_B, ID_B, n_A]]
4)A → B :E_{KUB} [E_{KRA} [n_C, IDc, n_B, n_A]]
5)B → C :E_{KUC} [E_{kRB} [n_C, n_B]]

Description of the three-party version :User A generates an unpredictable nonce value n_A, encrypts it along with his identity (ID_A) using his private key for authentication and sends it to B by encrypting with B's public key for confidentiality in message 1. Upon reception of message 1, B decrypts the message and authenticates A and generates an unpredictable nonce value n_B , encrypts it along with his own identity ((ID_B),user A's nonce value n_A, and Identity ((ID_A) with his private key and sends it to C by encrypting with C's public key in message 2.Upon reception of message 2, C decrypts the encrypted part, and verifies that it was indeed generated by B by checking the identifier in the message , then C generates an unpredictable nonce value n_C , encrypts it along with its own identifier IDc, user B's nonce value n_B, and Identity ((ID_B), user A's nonce value n_A, with his private key and sends it to A by encrypting with A's public key. When A receives message 3, it decrypts the encrypted part of it, and verifies that it was indeed generated by C by checking the identifier .Furthermore, it checks if it received back its nonce value n_A, and the identifier of B too. If these verifications are successful, then A authenticated B and C. After successful verification it continues encrypting its nonce value n_A, nonce value of B (n_B) ,the identifier of C and its nonce value n_C , with his private key and sends the result to B by encrypting with B 's public key in message 4. When B receives message 4, it decrypts it, and verifies that it was indeed generated by A by checking the nonce value n_A. Furthermore, it checks if it received back its nonce value n_B and if the message contains the identifier of C too. If these verifications are successful, then B authenticated A and C, and it continues encrypting

its own nonce value $n_{B,}$ nonce value of C (n_C) with his private key and sends the result of the encryption to C by encrypting with C 's pubic key in message 5. Finally, when C receives message 5, it decrypts it, and verifies that it was indeed generated by B by checking the nonce value $n_{B.}$ It also checks if it received back its nonce value $n_{C,}$ If these verifications are successful, then C authenticated A and B and the protocol terminates.

Shared Secret Key generation of three-party version

The users A,B,C share a secret key which is obtained by XORing the nonce values of the participants in the protocol run.

$K= n_{A\Phi}\ n_{B\,\Phi}\ n_C$

Messages of the n-party version:

1) $P_1 \rightarrow P_2$:$E_{KU2}[E_{KR1}\ [n1,ID_1\ ]]$

2) $P_2 \rightarrow P_3$:$E_{KU3}[E_{KR2}\ [n_2,ID_2,n_1,ID_1]]$

3) $P_3 \rightarrow P_4$:$E_{KU4}[E_{KR3}\ [\ n_3,ID_3,n_2,ID_2,n_1,ID_1\ ]]$

4) $P_4 \rightarrow P_5$:$E_{KU5}[E_{KR4}[n_4,ID_4,n_3,ID_3,n_2,ID_2,N_1,ID_1]]$

. . . . . .

n-1)$P_{n-1} \rightarrow P_n$:$E_{KUn}[EKR_{n-1}[n_{n-1},ID_{n-1},n_{n-2},ID_{n-2},\ldots..$ … $n_2,ID_2,n_1,ID_1]]$

n)$P_n \rightarrow P_1$:$E_{KU1}[E_{KRn}\ [n_n,ID_n,n_{n-1},ID_{n-1},\ \ldots\ldots..$ …..,$n_2,ID_2,n_1]]$

n+1)$P_1 \rightarrow P_2$:$E_{KU2}[E_{KR1}[n_n,ID_n,n_{n-1},ID_{n-1},\ldots\ldots$ ….,$n_3,ID_3,n_2,n_1]]$

n+2)$P_2 \rightarrow P_3$:$E_{KU3}[E_{KR2}[n_n,ID_n,n_{n-1},ID_{n-1},\ldots..$ ….,$n_4,ID_4,n_3,n_2]]$

n+3)$P_3 \rightarrow P_4$:$E_{KU4}[E_{KR3}[n_n,ID_n,n_{n-1},ID_{n-1},\ldots\ldots$ …,$n_5,ID_5,n_4,n_3]]$

. . . . .

2n-1)$P_{n-1} \rightarrow P_n$:$E_{KUn}\ [E_{KRn-1}[n_n,n_{n-1}]\ ]$

Let us consider any of the encrypted messages of the protocol above. For a given nonce value in the message, the identifiers that stand before the nonce value correspond to those parties who have already seen and forwarded n. For instance, in message n, the identifiers before n_1 are ID_n ,ID_{n-1}, …,ID_2 and indeed, all the participants that have already seen n_1 when message n is sent. Therefore, when a party receives back its random number in a message, it must check if all the other parties are listed before its random number in the message.

Shared Secret Key generation of n-party version

The users $P_{1,}$ $P_{2,}$ $P_{3,}$ ……, P_n share a secret key which is obtained by XORing the nonce values of all the participants in the protocol run .

$K= P_{1\Phi}\ P_{2\ \Phi}\ P_{3\,\Phi}\ \ldots\ldots_{\Phi}\ P_n$

Protocol 2:

Principle: The main drawback of Protocol 1 is that it relies on the assumption that the protocol participants trust each other for honestly executing the protocol. In Protocol 2, we remove this assumption. The main idea of Protocol 2 is that we allow each protocol participant to directly verify the other parties who received its nonce value. Unlike in Protocol 1, the nonce value and the identifier of each participant is passed around among the other participants. A party verifies the other participant by decrypting with the public keys of the corresponding parties. If, after performing all the decryptions, it recovers its original nonce value, then it is convinced that all the other parties were alive during the protocol run.

Assumptions: We assume that each participant in the system share his public key with the other participants.

Messages of the three-party version:

1)A → B :$E_{KUB}\ [E_{KRA}\ [n_A\ ,\ ID_A]\ ]$

2)B → C :$E_{KUC}\ [\ E_{KRB}\ [n_B\ ,\ ID_B]\ ,\ E_{KRA}[n_A,\ ID_A]\ ]$

3)C → A:$E_{KUA}\ [\ E_{KRC}\ [n_C,ID_C],\ E_{KRB}\ [n_B,\ ID_B],$ $E_{KRA}\ [n_A,ID_A]]]$

4)A → B:$E_{KUB}\ [\ E_{KRA}\ [n_C,\ IDc],\ n_B,\ n_A]\ ]$

5)B → C:$E_{KUC}\ [E_{kRB}\ [n_C,n_B]\ ]$

Description of the three-party version:

Party A generates an unpredictable nonce value n_A, encrypts it along with his identity (ID_A) using his private key and sends it to B by encrypting with B's public key in message 1. Upon reception of message 1, B decrypts the message and authenticates A and generates an unpredictable nonce value n_B , encrypts it along with his own identity ((ID_B) with his private key and sends it to C along with the message sent by user A (message 1) by encrypting with C's public key in message 2.Upon reception of message 2, C decrypts the encrypted part, and verifies that it was indeed generated by A and B by checking the identifiers in the message, then C generates an unpredictable nonce value n_C , encrypts it along with its own identifier ID_C with his private key and sends it to A along with the message sent by user B in message 2 by encrypting with A's public key. When A receives message 3, it decrypts the encrypted part of it, and verifies that it was indeed generated by B and C by checking the identifiers .Furthermore, it checks if it received back its nonce value n_A and identifier ID_A If these verifications are successful, then A authenticated B and C. After

successful verification it continues encrypting its nonce value n_A, nonce value of B (n_B) ,the identifier of C and its nonce value n_C, with his private key and sends the result to B by encrypting with B 's public key in message 4. When B receives message 4, it decrypts it, and verifies that it was indeed generated by A by checking the nonce value n_A. Furthermore, it checks if it received back its nonce value n_B and if the message contains the identifier of C too. If these verifications are successful, then B authenticated A and C, and it continues encrypting its own nonce value n_B, nonce value of C (n_C) with his private key and sends the result of the encryption to C by encrypting with C 's pubic key in message 5. Finally, when C receives message 5, it decrypts it, and verifies that it was indeed generated by B by checking the nonce value n_B. It also checks if it received back its nonce value n_C. If these verifications are successful, then C authenticated A and B and the protocol terminates.

Shared Secret Key generation of three-party version

The users A,B,C share a secret key which is obtained by XORing the nonce values of the participants in the protocol run.

$K= n_A \Phi n_B \Phi n_C$

Messages of the n-party version:

1) $P_1 \rightarrow P_2 : E_{KU2} [E_{KR1} [n1, ID_1]]$

2) $P_2 \rightarrow P_3 : E_{KU3} [E_{KR2} [n_2, ID_2], E_{KR1} [n_1, ID_1]]$

3) $P_3 \rightarrow P_4 : E_{KU4} [E_{KR3} [n_3, ID_3], E_{KR2} [n_2, ID_2], E_{KR1}[n_1, ID_1]]$

4) $P_4 \rightarrow P_5 : E_{KU5}[E_{KR4} [n_4, ID_4], E_{KR3}[n_3, ID_3], E_{KR2}[n_2, ID_2], E_{KR1}[n_1, ID_1]]$

.
.
.
.

n-1) $P_{n-1} \rightarrow P_n : E_{KUn} [E_{KRn-1} [n_{n-1}, ID_{n-1}], E_{KRn-2}[n_{n-2}, ID_{n-2}], \ldots\ldots\ldots E_{KR2}[n_2, ID_2], E_{KR1}[n_1, ID_1]]$

n) $P_n \rightarrow P_1 : E_{KU1} [E_{KRn} [n_n, ID_n], E_{KRn-1}[n_{n-1}, ID_{n-1}], E_{KRn-2}[nn_{-2, IDn-2}], \ldots\ldots\ldots \ldots\ldots, E_{KR2}[n_2, ID_2], E_{KR1}[n_1, ID_1]]$

n+1) $P_1 \rightarrow P_2 : E_{KU2} [E_{KRn}[n_n, ID_n], E_{KRn-1}[n_{n-1}, ID_{n-1}], \ldots\ldots\ldots E_{KR3}[n_3, ID_3], n_2, n_1]]$

n+2) $P_2 \rightarrow P_3 : E_{KU3} [E_{KRn}[n_n, ID_n], E_{KRn-1}[n_{n-1}, ID_{n-1}], \ldots\ldots\ldots\ldots, E_{KR4}[n_4, ID_4], n_3, n_2]]$

.
.
.

2n-1) $P_{n-1} \rightarrow P_n : E_{Kun-1} [E_{KRn-1}[n_{n-1}, ID_{n-1}]]$

Unlike in protocol 1 ,the identifiers and the nonce values encrypted by private keys value correspond to those parties who have already seen and forwarded the message . For instance, in message n, the identifiers and the nonce values are ID_n , n_n, ID_{n-1}, $n_{n-1}, \ldots, ID_2$, n_2, ID_1, n_1 .It Implies that all the participants that have already seen n_1 when message n is sent. Therefore, when a party receives back its random number in a message, it can check if all the other parties nonce values and identifiers are listed before its nonce value in the message.

Shared Secret Key generation of n-party version

The users $P_1, P_2, P_3, \ldots\ldots, P_n$ share a secret key which is obtained by XORing the nonce values of all the participants in the protocol run.

$K= P_1 \Phi P_2 \Phi P_3 \Phi \ldots\ldots \Phi P_n$

III. SECURITY ANALYSIS AND COMPARISON

The most obvious application of a public key encryption system is confidentiality where a message which a sender encrypts using the recipient's public key can be decrypted only by the recipient's paired private key .Presumably, this will be the owner of that key and the person associated with the public key used[9,10]. Another type of applications in public-key cryptography are digital signature schemes where a message signed with a sender's private key can be verified by anyone who has access to the sender's public key, thereby proving that the sender had access to the private key and therefore is likely to be the person associated with the public key used, and the part of the message that has not been tampered with. To verify that a message has been signed by a user and has not been modified the receiver only needs to know the corresponding public key.

To achieve authentication, *and* confidentiality, the sender could first sign the message using his private key, then encrypt the message and signature using the recipient's public key[9,10]. These characteristics can be used to construct many cryptographic protocols for multi-party key agreement and authentication.In protocol 1 we assumed that each participant trust one another for executing the protocol honestly. When a party receives a message it just verifies the authenticity of the party that had sent the message by decrypting the message with the corresponding party's public key and confidentiality is achieved as the message can only be decrypted by receivers private key .Protocol 2 executes with an assumption that no party trusts the other. So when a message is received by a party it verifies the authenticity of all the other parties who had already seen and forwarded the message. Encrypting the message with party's private key thus provide authentication and further

encryption using public key of the receiver introduces confidentiality.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we first reviewed mutual authentication protocols and then proposed a new multi party authentication schemes based on public key cryptosystem. The communication structure underlying the protocols is same as pattern for multiparty challenge-response mechanisms mostly used today. Analysis of our protocols showed that they can withstand the security flaws as strong authentication and confidentiality are part of public key cryptosystem. We extended our discussion in this paper to even more important concept of key-distribution, coupled with authentication. However, the work can be further extended to multi party contract signing protocols and multi party conferencing with key distribution.

REFERENCES

[1] J.A. Clark and J.L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, 1997. http://citeseer.ist.psu.edu/clark97survey.html.

[2] Security protocols open repository (SPORE). http://www.lsv.ens-cachan.fr/spore.

[3] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*,volume 1055, pages 147–166. Springer Verlag, 1996.

[4] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(2):120–126, February 1978.

[5] C.J.F. Cremers and S.Mauw. Generalizing Needham-Schroeder-Lowe for multi-party authentication. CS-Report 06/04,Department of Mathematics and Computing Science, Eindhoven University of Technology, 2006.

[6] Levente Buttyan, Attila Nagy and István Vajda,"Efficient Multi-party challenge response protocols for entity authnetication**"**, Periodica Polytechnica Ser. EL. Eng. Vol. 45, No. 1, PP. 43–64 (2001).

[7] J. Clark and J. Jacob. A survey of authentication protocol literature. http://www-users.cs.york.ac.uk/~ jac/papers/drareview.ps.gz

[8] C. Mitchell. Limitations of challenge-response entity authentication.IEE Electronics Letters, 25(17), 1989.

[9] PublicKey Protocols. In *Advances in Cryptology – CRYPTO'95*, pp. 236–247, 1995.

[10] William Stallings," Cryptography and Networksecurity", PHI, 4th edition, 2005.