International Journal of Cryptology Research 1 (1): 43-53 (2009)

A Method of SSO IN P2P Networks with the Capability of Adaptation at Network Topology Change

[1]Mohammad Reza Sohizadeh Abyaneh, [2]S.Amir Hossein A.E.Tabatabaei and [1]Mahmoud Salmasizadeh
[1]Sharif University of Technology
[2]Zaeim Electronic Industry
Email:sohizadeh@gmail.com

ABSTRACT

In this paper, we propose a method to provide distributed authentication in p2p networks. Our method has the advantages of reducing the computational burden by supporting SSO as well as avoiding unnecessary repetition while the network topology varies. The main contribution of this paper which is adding a phase (share update) to previous works in order to increases the speed of the method. This is accomplished by an increase of about 30%.

Keywords: Authentication, Single Sign On (SSO), Peer-to-Peer (p2p) Networks

INTRODUCTION

"Peer-to-peer (P2P) computing is the sharing of computer resources and services by direct exchange between systems. These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files."[1] A broad definition of P2P includes the client server mode of computing, as well as exchange directly amongst clients or amongst servers.

Peer-to-peer systems, beginning with Napster [2], Gnutella [3], and several other related systems, became immensely popular in the past few years, primarily because they offered a way for people to get music without paying for it. However, under the hood, these systems represent a paradigm shift from the usual web client/server model, where there are no "servers;" every system acts as a peer, and by virtue of the huge number of peers, objects can be widely replicated, providing the opportunity for high availability and scalability, despite the lack of centralized infrastructure.

Peer-to-peer networks are formed in a "haphazard" manner and do not rely on an established infrastructure [4]. Security (i.e. authentication and trust management) is based on the notion of trust and it is not well defined for these networks. Centralized security solutions for them could be significantly vulnerable [5]. So, recently, a lot of effort has been put in providing security infrastructure for P2P networks [6, 7, 8, 9]. These efforts

aim at distributing CA functionality (Certificate Authority) to a set of nodes in the network and concentrate on two major fields:

1) Designing useable authentication methods for infrastructureless networks.
2) Exploiting an authentication method in a Single Sign On (SSO) process to let the users get their desired services by being authenticated once.

In these methods the main concept is sharing a secret (CA's private key) among some of nodes and their partial signature are aggregated by a node to form its valid certificate. However, these methods have some drawbacks in common:

1) For each different service, the nodes must get new partial certificates from the whole distributed CA nodes.
2) If the network topology varies (one of the distributed CA member nodes leave the network or/and other nodes contribute the network and probably become one of the distributed CA member), the whole certificates must be renewed and the distributed CA members must share another secret as well.

There are some methods of providing p2p networks with SSO [10]. They have exempted us from the first drawback, but the second one has still remained.

In this paper, we propose a method of Single Sign On, based on a distributed CA concept as well as being capable of prompt adaptation at the time of network topology change. This is achieved by contributing an *update phase* to prior SSO methods. By using our proposed methods, the nodes not only can exploit SSO facilities, but also should not be worried about certification renewal whenever the network topology varies. In this way, the computational overhead reduces about 30%. In our proposed method, whenever one node enters or leaves the network, the shares among the CA members are updated so that the whole secret remains intact. The rest of the paper is organized as follows: Section 2 provides some general concepts used in the paper, section 3 overview on the previous works have boon done in this area of research, section 4 elaborates the proposed method, in section 5, we will evaluate the proposed method and section 5 concludes the paper.

BASIC CONCEPTS

Notation

This paper is mainly concerned with authenticating users, programs and services referred to as *client*. Each public key PK_X and corresponding private key SK_X is associated with a principal X, private key SK_X is then said to *speak for* X.A message m encrypted under the key K_X is denoted $|m|_{Kx}$.

Terms Definition

There are three distinguished entities or roles used in this paper which are naturally an ordinary node like other nodes but might have different responsibilities and functionalities as following:

1) *Application Servers(S):* Entities which provide the clients with their desired services. Each application server has a local name space $N(S)$. The access control list at S associates privileges with names from $N(S)$, and clients of S may refer to other clients of S using names from $N(S)$.
2) *Authentication Servers* (A)*:* Entities which authenticate the clients to verify whether they are allowed to use their intended service from a server. Each authentication server has a local name space $N(A)$. A will implement one or more means to check whether a principal had previously registered with some given name from $N(A)$.
3) *Clients(C)*: Normal nodes which request to be served by Servers. Global name space N^* is defined so that if $c_1 \in N(A_1)$, $c_2 \in N(A_2), \ldots, c_r \in N(A_r)$ hold then $c_1 @ A_1 \mid c_2 @ A_2 \mid \ldots \mid c_n @ A_r \in N^*$ holds.

   Each application server simply stores a mapping between names in $N(S)$ and names in N^*. But each authentication server translates a request by a client C to be authenticated as global name to check whether it satisfies the identify requirements for every name [10].

Specifying Authentication Policies

An authentication policy P is a disjunction $P = P_1 \vee P_2 ... \vee P_n$ of *sub-policies*; P is *satisfied* for a principal P provided some sub-policy P_i is satisfied. Each sub-policy P_i specifies a set $\hat{P}_i$ $\left(P_i = \left\{ \hat{P}_i \right\} \right)$ of authentication servers $\{ A_i^1, A_i^2, ..., A_i^m \}$ and a threshold constraint t; P_i is *satisfied* by a principal P provided t of the authentication servers $\hat{P}_i$ each certify their identity requirements for P. For example, to implement what is known as 3-factor authentication, have every sub-policy specify a threshold constraint of 3 and include in exactly 3 servers that each use a different identity check.

PREVIOUS WORKS

Prior work on decomposing network-wide authentication services has focused on delegation—but not distribution—of trust. Kerberos [11] performs user authentication in wide-area networks, but ties user identity to a centralized authentication server. OASIS [12] and the Liberty Alliance Project [13] are recent industry efforts aimed at supporting a federated network identity. OASIS provides a standard framework for exchange of authentication and authorization information; Liberty uses this framework to delegate authentication decisions and to enable linking accounts at different authentication servers. The authentication policy in these systems corresponds to a disjunction of sub-policies, each specifying a single authentication server.

PolicyMaker [14] is a flexible system for securely expressing statements about principals in a networked setting. It supports a far broader class of authentication policies than our method does. Besides authentication, PolicyMaker also implements a rich class of authorization schemes. But with PolicyMaker, an application server must check each certificate involved in authentication or authorization decisions. In contrast, with our proposed method, the check at an application server is constant-time, because work has been factored-out and relocated to set-up protocols at the authentication servers and clients. Our proposed method which is based on [10] borrows from Gong's threshold implementation of Kerberos KDCs [15] and from COCA [16] the insights that proactive secret sharing and threshold cryptography can help defend distributed services against attacks by so-called mobile adversaries [17], which attack, compromise, and control a server for a limited period before moving on to the next.

PROPOSED METHOD

Our proposed method for authentication of client C to get service from application service S includes four phases: *Application Server Setup, Client Authentication, Client Access to Application Server and Share Update.* In this section these four phases are elaborated.

Application Server Setup

In this phase, as Fig.1 shows, the application server first determines a set of clients $A = \{A_i^1, A_i^2, ..., A_i^m\}$ for each sub- policies (1). Then the private key of this set (SK_i) is shared between tem (2) and the corresponding public key is saved in S.

1. S *Determines* $A = \{A_i^1, A_i^2, \ldots, A_i^m\}$ *for* $\hat{P}_i \quad 1 \le i \le n$
2. $S \rightarrow \forall A_i^j \in \hat{P}_i : SK_i^j \in (t, |\hat{P}_i|) = SK_i^1, SK_i^2, ..., SK_i^{|\hat{P}_i|}$
3. $S : SavesPK_i$

Figure 1: Application Server Setup

Secret Sharing Algorithm

In order to share the secret key among the determined clients as partial authentication servers in the first phase, we can use the algorithm proposed in [18]. This algorithm consists of two phases (as shown in Fig.2):

- The application server chooses a polynomial of degree $t-1$, a large prime number p and the secret key $SK_i = a_0$ (1).
- Each shared is computed (2).
- The secret key is recoverable having t appropriate shares (3).

1. $f(x) = (a_0 + a_1 x + ... + a_{t-1} x^{t-1}) \bmod p = \sum_{i=1}^{t} a_{i-1} x^{i-1} \bmod p$

2. $SK_i^j = f(j) = \sum_{j=1}^{t} a_{j-1} x^{j-1}$

3. $f(x) = \sum_{j=1}^{t} SK_i^j \times L_j(x)$, $L_j(x) = \frac{\prod_{k \neq j} (x - x_k)}{\prod_{k \neq j} (x_j - x_k)}$, $SK_i = f(0)$

Figure 2: Secret Sharing Algorithm

Share Verification Algorithm

To ensure the clients that they have received an appropriate share verification algorithm might be used as shown in Fig.3. In this algorithm *S* broadcasts some information (1, 2) by which the authentication servers can be assured of their share correctness (3).

1. $S \rightarrow A : \{g\}$, g is a randomly chosen number
2. $S \rightarrow A$:
$\{ g^{SK_i} \bmod p, g^{a_1} \bmod p, g^{a_2} \bmod p, ..., g^{a_{k-1}} \bmod p \}$
3. $\forall A_i^j : g^{SK_i^j} \bmod p \stackrel{?}{=} g^{SK_i} \times \prod_{l=1}^{t-1} (g^{a_l})^l$

Figure 3: Share Verification Algorithm

Client Authentication

In this phase, as shown in Fig.4, client intends to get an authentication token to use it as a permission of being served by application server. Each token in correspondent to a sub-policy $\left(\left\{ P_i = \hat{P}_i \right\} \right)$ which should be determined by the application server at the client request (1,2). The token (3) implicitly proves that SK_C belongs to C. This token is signed by the secret key of each authentication servers, who have a share of the main secret key. Client receives all partial signed tokens and computes his main token to deliver it to the application server (4).

$$
\begin{aligned}
&1.\ C \rightarrow S : \textit{Request for Authentication Policy.}\\
&2.\ S \rightarrow C : P_i\\
&3.\ \forall A_i^j \in \hat{P}_i :\\
&\qquad C \rightarrow A_i^j : \{C, PK_C, P_i, T_s, T_f\}\\
&\qquad T_f = \textit{Starting Time}\\
&\qquad T_f = \textit{Finishing Time}\\
&\qquad A_i^j \rightarrow C : |\, C, PK_C, P_i, T_s, T_f \,|_{SK_i^j}\\
&4.\ C : \textit{Computes Authentication Token} =\\
&|\, C, PK_C, P_i, T_s, T_f \,|_{SK_i}
\end{aligned}
$$

Figure 4: Client Authentication

Client Access to Application Server

In this phase (as shown in Fig.5), client is authenticated to application server and it is allowed to use its desired service. This phase is a *challenge and response* protocol, in which client firstly requests for a challenge from application server (1). In response, application server produces a random number m, signs it by its secret key and sends it to client (2). Then, client decrypts the challenge, signs it by its secret key and appends it to the token received at the prior phase and sends it to application server (3). Consequently, application server is able to authenticate the client (4).

$$
\begin{aligned}
&1.\ C \rightarrow S : \textit{Request Authentication challenge.}\\
&2.\ S \rightarrow C : \textit{challenge} = |\, m \,|_{SK_S}\\
&3.\ C \rightarrow S : \left\{|\, m \,|_{SK_S}, |\, m \,|_{SK_C}, |\, C, PK_C, P_i, T_s, T_f \,|_{SK_i}\right\}\\
&4.\ S : \textit{Checks for validity of } m \textit{ and } C \textit{ having } SK_C
\end{aligned}
$$

Figure 5: Client Access to Application Server

It should be noted that these tokens can be used for other application servers provided that they have the same sub-policy. In this way, Single Sign On goal is achieved.

Share Update

In previous methods, no measurement is taken to support network topology or sub-policy variations. If this happens, phases of 2 and 3 must be repeated.

This phase, which is the main contribution of this paper, has been contributed to the other three phases to avoid computational overheads due to repetitive token requiring in the case of network topology or sub-policy variations. In this phase (as shown in Fig.6), each application server has a polynomial $\left(g_j(x)\right)$ of degree $t-1(4.1.1)$. It chooses a random number of g and sends signed $g^{\beta j,d}$ to its sub-policy authentication servers (2). Then, it updates their secret and sends the updated shares for them (3). The authentication servers then update their share (secret key) accordingly (4).

$$g_j(x) = (\beta_{j,1}x + \beta_{j,2}x^2 + ... + \beta_{j,t-1}x^{t-1}) \bmod p$$

$$1. = \sum_{d=1}^{t-1} \beta_{(j,d)} x^d \bmod p$$

$$g_j(0) = 0$$

$$2. \{ \{ g^{\beta_{j,d}}, \quad | h(g^{\beta_{j,d}}) |_{SK_i} : | 1 < d < t \}$$

$$3. SK_i^{j \to k} = g_j(k) \bmod p \quad 1 < k < t$$

$$A_i^j \to \{A_i^k\} : | SK_i^{j \to k} |_{PK_i^k} \quad 1 < k < t$$

$$4. SK_i^{k(new)} = SK_i^{k(old)} + \sum_{i=1}^{t} SK_{j \to k}$$

Figure 6**:** Share Update

In this way, if an authentication server is contributed to or left from the sub-policy, this phase can be run and all the shares can be updated without needing the authentication tokens to be changed.

EVALUATION OF PROPOSED METHOD

In this section an evaluation of our proposed method is explained in a list as follows:

- The conjunction implicit in the meaning of a sub-policy also allows an application server to defend against compromised authentication servers and specify independence assumptions about those severs. For a sub-policy P_i involving threshold parameter t, a set of t or more authentication servers in $\hat{P}_i$ must come under control of an adversary before that adversary can cause P to be satisfied.
- The disjunction used to form an authentication policy P from sub-policies and the threshold parameter in sub-policies supports fault-

tolerance, since the failure of one or more authentication servers then won't necessarily render *P* unsatisfiable.

- Note that the disjunction and sub-policy threshold constraints implement a distribution of trust, since these constructs allow an authentication policy to specify that more trust is being placed in an ensemble than in any of its members.
- The absence of negation in authentication policies is worth noting. Without negation, the inability of a principal to be certified by some authentication server can never lead to a successful authentication; with negation, it could. So, by omitting negation from our policy language, crashes and denial of service attacks cannot create bogus authentications.
- By using our proposed methods, the nodes not only can exploit SSO facilities, but also should not be worried about certification renewal whenever the network topology or sub-policies varies.
- In [10] the only comparable method, the performance-critical path consisting of the *Client Authentication* and *Client Access to Application Server* has been benchmarked on a 2.60GHz Pentium 4. The result is as follows:
  For RSA signatures using a (4; 5)-threshold sharing of a 1024-bit RSA key. The Client Authentication protocol took 430 msec and the Client Access to Application Server protocol took 947μsec. Our contributed phase (*share update*) has also been benchmarked on the same platform and took 306 msec.
- According to the results, at the time of sub-policy or network topology variation our method is about 28.8% faster. Because it is not needed to run phase 2 and 3 again.
- As mentioned in section 4.3, to take advantages of SSO, application servers can share their policies and then the clients are able to use their authentication tokens to get services from different application servers.

CONCLUSION

In this paper we proposed a method to provide distributed authentication in p2p networks. Our method has the advantages of reducing the computational burden by supporting SSO as well as avoiding unnecessary repetition phases while the network topology or sub-policies vary. The main contribution of this paper which is adding the forth phase (share update) increases the speed of the method about 30%.

REFERENCES

[1] http://www.peer-topeerwg.org/whatis/index.html

[2] Several websites have introductory information, e.g.www.bearshare.com, www.limewire.com.

[3] www.Napster.com

[4] Shardul Gokhale, Partha Dasgupta. 2004. Distributed Authentication for Peer-to-Peer Networks.

[6] Zhou, L., and Z. J. Haas. 1999. Securing Ad Hoc Networks., *IEEE Network*.

[7] Weimerskirch, André, and Gilles Thonet. 2002. A Distributed Light-Weight Authentication, *LNCS*.

[8] Luo, Zerfos, Kong, Lu and Zhang. 2002. Self-securing Ad Hoc Wireless Networks, *ISCC '02.*

[9] Botelho, Luis, Steven Willmott, Tianning Zhang and Jonathan Dale. 2002. Self Organized Public Key Management For Mobile Ad Hoc Networks, *Technical Reports in Computer and Communication Sciences*.

[10] William Josephson, Emin G¨un Sirer, Fred B. Schneider. 2005. Peer-to-Peer Authentication with a Distributed Single Sign-On Service.

[11] J. G. Steiner, B. C. Neuman, and J. I. Schiller.Kerberos. 1988. An authentication service for open network systems". *In Proceedings of the Winter 1988 Usenix Conference*, February 1988.

[12] Liberty Alliance Project. Introduction to the liberty alliance identity architecture, March 2003.

[13] Organization for the Advancement of Structured Information Standards. http://www.oasis-open.org, February 2004.

[14] Matt Blaze, Joan Feigenbaum, and Jack Lacy. 1996. Decentralized trust management". In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 164–173.

[15] L. Gong. 1993. Increasing availability and security of an uthentication service. *IEEE J. Select. Areas Commun.*, 11(5):657–662.

[16] L. Zhou, F. B. Schneider, and R. van Renesse. 2002. Coca: A secure distributed online certification authority. *ACM Transactions on Computing Systems*, 20(4):329–368.

[17] R. Ostrovsky and M. Yung. 1991How to withstand mobile virus attacks. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 51–59.

[18] A. Shamir. 1979. How to Share a Secret. *Communication of the ACM.*