DIVISION OF COMPUTER SCIENCE

Self Authenticating Proxies

Marie Rose Low
Bruce Christianson

Technical Report No.201

June 1994

Self Authenticating Proxies

Marie Rose Low and Bruce Christianson

commrnrl@herts.ac.uk comqbc@herts.ac.uk

Computer Science Division, Hatfield Campus,
University of Hertfordshire,
Hatfield, Herts.

Abstract

Authentication and access control are usually implemented as two separate protection mechanisms because they are logically separate functions. A consistent approach to both of these functions is proposed in this paper. In this new approach, resource management, another aspect of protection, can also be included. By combining the properties of public key encryption with cascading proxies, a single mechanism is devised to provide these three aspects of protection. The mechanism provides independence from the system infrastructure and from any particular security domain, control policy or authentication server enabling principals to define and enforce their own protection requirements.

1. Introduction

As the use of computer systems becomes more widespread, there is a growing need for the ability to work with and on remote systems in an open and distributed environment. This requires more than just a network connection between systems. There is also a need for system services and users to be able to determine for themselves how they use and share their resources within these distributed systems. Whilst there has to be a strong mechanism for services to protect themselves from malicious use and users from malicious services, the mechanism must be flexible and independent from as much of the systems' infrastructure as possible so as to operate successfully over heterogeneous environments and autonomous resource management domains. The properties of public key encryption and cascading proxies are combined in this paper, to develop a new mechanism which can provide a solution to some of these requirements.

In the past, authentication and access control have frequently been seen as the only two sides to protection that are required to ensure proper use of and access to services. These two functions have also usually been implemented as separate mechanisms. This paper will suggest that the two mechanisms can be approached in a unified manner and that resource management is a third and important aspect of protection which can be easily incorporated within this new approach.

Most service providers provide a service only to those principals who have been granted authority to use it. Access control is the means for checking whether or not a principal has authority to request an operation. Authentication is the process by which principals obtain a high level of assurance that the principal with whom they are communicating is who they suppose it to be. Accounting is one of the main tasks for resource managers. These managers must ensure that use of a resource is not given to someone who has not the funds, even if he has the right to use it. The recipient of a service needs to be charged once the service has been provided.

With the exception of covert channel analysis, there has not been very much research that views resource control as a part of protection and, consequently, the most commonly used protection methods deal only with authentication of the principals involved and access control.

2. Existing Authentication and Access Control Methods

Authentication of principals is usually implemented using a shared key encryption algorithm e.g. Kerberos [Steiner et al. 1988]. There is typically an authentication server for each management domain that shares a secret key with each of the principals in that domain. The authentication server is relied upon to authenticate these principals and their requests. This is done by checking that they have knowledge of the shared key and requires at least one on-line transaction with the authentication server per request.

Access control is usually provided either by Access Control Lists (ACL) or by capabilities [Lampson 1971]. A service provider may have an ACL that identifies which clients have rights to use the service and what those rights are. A capability is a 'token' held by a client that identifies a service and the rights to that service. The mechanism proposed in this paper is developed from capabilities; the next section describes how capabilities have been used and extended.

2.1. Capabilities and Authentication

The traditional or 'pure' **capability** is a token which consists of the identifier of the service(s) and the access rights for that service. Possession of a capability suffices to gain access to the service. Problems arise with modification, propagation, revocation and theft of capabilities. Capability systems have been implemented using either hardware support or system software to help prevent these problems. This proves to be very restrictive and problematic in a distributed heterogeneous environment with mutually suspicious domains. Modification can be prevented if the capability is protected by an encrypted checksum, e.g. Amoeba [Mullender 1985]. Capabilities can be allowed to propagate freely and then checked against the user's access rights at the point of access, as suggested in the SCAP architecture [Karger 1988]. Propagation and theft can be prevented by making capabilities identity dependent as in ICAP [Gong 1990]. The user of the capability has to be authenticated to prove ownership when the capability is presented to gain access to a resource.

Schemes using such capabilities are dependent on authentication servers e.g. Kerberos [Steiner et al. 1988], which uses a trusted third party authentication service based on the Needham, Schroeder model [Needham, Schroeder 1978]. A development of the notion of a capability is the 'proxy', wherein a principal can act using another principal's authority. The authority has to be deliberately and verifiably delegated.

2.2. Proxies

The notion of delegating authority was used by Sollins [Sollins 1988] and then developed further by Neuman [Neuman1991] and Bull [Bull et al 1993] The basic principles of these schemes is that a 'token' (e.g. proxy) identifies its originator and a delegate principal as well as the object and access rights to be delegated. The principals each share a secret key with an authentication server. The token is digitally signed by the delegator using this secret key, making the token tamper proof. The validity of the token is determined by the authentication server. The authority delegated is constrained by elements held in the token.

Bull et al take a server-based view where each service, rather than the system infrastructure, takes on responsibility for issuing authority to its customers. The server always controls access even though reference to another authority may be needed to make a decision. Their model allows a high degree of autonomy so that, for instance, servers themselves may migrate freely without concern about the system infrastructure. It does however imply that a client has to have complete trust in the behaviour of the server. This shift of emphasis places the onus of responsibility for the use of services on the entities involved in a client server transaction. Authentication is based on shared key encryption and relies on an on-line authentication server. Although all three schemes can operate using public (rather than shared) key encryption, previous descriptions of how this could work do not show how the full benefits of using public keys can be exploited. The scheme proposed in this paper differs from the above in our use of public key encryption. The benefits of our scheme are: verifiable attribution of all actions; a reduction in the areas of trust needed at the time of user verification to those local to each principal; local policy decision making; confinement of honoured requests, and therefore damage, to those requests that are verifiably authorised; and an unforgeable audit trail of all operations performed. In particular, no on-line authentication service is required at the time of a transaction and services cannot masquerade as their clients.

3. Extending Proxies to SAProxies

Proxies enable authority to be passed from one principal to another. Public key encryption

[Diffie, Hellman 1976] provides each principal with the means of producing a unique signature. By combining the properties of public key encryption with proxies, we shall develop a mechanism which enables a principal possessing authority to pass on some (or all) of his authority to any other principal and to prove his authenticity without on-line access to an authentication server. Furthermore, the second party can prove beyond reasonable doubt to a third party, that this delegation of authority was intended by the delegators.

A public key 'signature' is a cryptographic checksum which is dependent both on the data it signs and on the private key used, and which can be checked using the matching public key. Thus if each principal (user, service) S has its own private key, K^{-}_{S}, then each signature is unique to that entity, i.e. unforgeable. A statement signed with a private key, can be shown to have been signed by a particular principal if the statement can be verified with that principal's public key, K^{+}_{S}.

There is a Certification Authority (CA), available to each principal, whose sole purpose is to bind a principal's public key with that principal's name. The public key, K^{+}_{S}, for S is bound to S's name in a Public Key Certificate (PKC) by the CA. The CA also has a public and private key pair. A PKC, C_S , for principal S is S's name and his public key 'signed' by the CA's private key, K^{-}_{CA},

Token	Owner	Issuer	Public Key	Life-span	Signature
C_S	S	CA_{id}	K^{+}_{S}	dates	$sig^{C}_{S}\,(K^{-}_{CA})$

where CA_{id} is the identity of the CA, 'dates' is the lifetime of the PKC and sig^{C}_{S} is the data {S, CA_{id}, K^{+}_{S}, dates} signed with the CA's private key K^{-}_{CA}.

Any principal, A, has a PKC, C_A, in the same format as above and certified by some CA. The CA's public key, K^{+}_{CA}, is made globally public so that it is both easy to obtain and difficult for an impostor to change. Once K^{+}_{CA} been correctly obtained by a principal then that principal, whether local or remote to A, can verify A's PKC by using K^{+}_{CA} to check the CA's signature. This can be done locally to the principal without on-line communication with the CA. The principal, S, can generate a proxy, P_A, for principal A granting A access to S's services. The proxy refers (points) to both A's and S's PKCs, C_A and C_S, which in turn contain A's and S's names. These references (pointers) are C_A's and C_S's signature blocks, sig^{C}_{A} and sig^{C}_{S}, which are unique and verifiable. The proxy is signed by S's private key, K^{-}_{S}. As the proxy refers to C_S which has S's public key, the proxy becomes 'self-authenticating'. We call P_A a SAProxy [Low, Christianson 1994]. The format of this token is:

Token	Delegatee PKC sig	Delegator PKC sig.	Object	Access Rights	Life-span	Token Signature
P_A	sig^{C}_{A}	sig^{C}_{S}	S	ar	dates	$sig^{P}_{A}\,(K^{-}_{S})$

The SAProxy, P_A, can be verified by any principal because it refers to all the information needed to do this. The benefits are twofold in that there is no need to call an authentication server either to generate the proxy or to verify it as both can be done 'locally'.

A request to S from A refers to both P_A and C_A, and is signed by A's private key, K^{-}_{A}. A asks to use his authority for S as granted to him in P_A.

The signed request has the following format,

Token	Service	Requester PKC sig.	Object	Access Rights	Life-span	Token Signature
R_A	S	sig^C_A	sig^P_A	*request*	dates	$sig^R_A (K^-_A)$

A then sends S the following tokens:

$$C_A \; P_A \; R_A$$

S verifies C_A using the CA's public key, K^+_{CA}, which is public knowledge. S can verify sig^R_A with A's public key, K^+_A which is in C_A. S then uses K^+_S, its own public key, to verify the signature, sig^P_A, of the SAProxy, P_A. S is now assured that A's request has been made with S's authority and can go ahead to grant it. C_S need not be included in the stream of tokens because it is S's own PKC. However, all the tokens which are referred to in the transaction must be recorded with the transaction log, so that the authority with which a transaction was granted can be verified and attributed to those involved if there is a subsequent dispute[1].

4. Using SAProxies

Services can become responsible for their own authentication and access control and can become the SAProxy grantors, controlling to whom they give access and what permissions they grant. They may maintain their own security policy and check this when they issue SAProxies or they may refer to a separate policy checker, either way they are responsible for any SAProxies they grant.

SAProxies can also be used as a means for managing resources. As well as including access rights, SAProxies may include resource funds and so resource managers may use SAProxies to control the use of their resources. As SAProxies can always be attributed to one principal, they can also be used for accounting purposes.

With public key encryption and SAProxies, the server can show the authority under which it performed all transactions by keeping an audit trail of these actions and the SAProxies which authorised them, and this authority cannot be forged. The user of the service can then check that all operations performed have been properly authorised and can hold the service accountable for any violation of trust, and likewise the service can defend itself against accusations of misbehaviour. Existing schemes that are dependent on an authentication server require that the authentication server is trusted by all the principals it serves, as it holds each principals' secret key. Should such a server become malicious or be compromised then all principals who share their secret keys with it are compromised as the authentication server can masquerade as its clients. This is important even for a trustworthy server which then has no way to defend its reputation against a malicious charge of malfeasance by a dishonest client who repudiates his own (fraudulent) act and falsely attributes it to the server.

In contrast, if a CA is violated and its private key compromised, the worst that can happen is that 'false' PKCs can be generated for genuine users. However, a principal's private key is not actually compromised when the CA is violated, and a dishonest CA cannot generate a correct transaction on behalf of a principal because the CA does not know the principal's private key. Furthermore, since the signature of a delegatee's PKC is used in a SAProxy to point to his identity instead of that delegatee's name, then, even if a CA's private key is compromised and bogus PKCs generated at a later date, the authorising SAProxies cannot be used with the false PKCs. Also, since a CA issues PKCs signed with the CA's own private key it is easy to determine which CA is behaving maliciously and which attempted transactions are bogus. This level of protection is not possible with shared key authentication.

[1] Note that S can not forge the request R_A because it does not know A's private key.

Where shared key encryption is used, there is a need for a trusted path, via a hierarchy, between authentication servers in different domains both to authenticate principals and to verify any cryptographic signatures generated for delegation tokens as proposed by Sollins, Neuman and Bull. Within the scheme described here, each principal determines for himself how much he trusts a remote CA. A principal will usually obtain knowledge of and confidence in a remote CA's public key by getting this key from one or more trusted third parties. It may be that the CA's are arranged in a hierarchy, with each CA acting as a trusted third party for those lower down, but this is a matter of choice, not a requirement. There is, in fact, no need for any CA to have knowledge of or to trust any other CA that is involved in a transaction.

It is only the principals themselves who need to have confidence in a CA's public key, and they may obtain this in any way which their security policy permits: from a trusted colleague who knows the key, from a local agent whose business it is to obtain these keys and which is then liable for their correctness, or by a direct physical approach. A highly secure procedure is feasible, since it need be done only once per remote CA. Once a remote CA's public key has been obtained, a PKC issued by that CA can be verified locally without the need for secure on-line communications between domains. A principal may even generate his own personal certificate for the remote CA's key so that it may be used with assurance in future transactions.

There are advantages in avoiding a fixed universal hierarchy in an open environment. Each CA can operate its own security policy, and the corresponding security domains do not need to map neatly onto management domains, as is required in the case of a hierarchy. It is the principals who finally decide which public keys they will accept. The amount of authority granted to principals served by a remote CA may be set according to the local principal's level of trust in the remote CA's procedures to generate only genuine PKCs.

There is also no problem with denial of service if a CA becomes unavailable because there is no requirement for on-line access to a CA even within a single system. CA's can therefore be made more secure by making them less available. The SAProxy scheme is particularly suited to distributed services where any one of the servers can verify another server's SAProxy and PKC. Servers providing the same service on different systems have no trouble honouring requests based on the authority of another server.

5. Delegation of Authority

SAProxies can be cascaded so that a principal that has been granted authority by a service or policy checker can delegate all or part of this authority to another principal by generating a SAProxy for that second principal. Thus, the grantee becomes a grantor, and passes his SAProxy (proof that he has authority) with the delegatee's SAProxy (proof of delegation) to the delegatee.

Consider the situation where user A, with authority to use service S, wants to delegate some of its work load to his junior, user B. A achieves this by giving B a SAProxy, P_B, which essentially authorises B to use A's SAProxy, P_A, within the constraints set by the access rights in P_B. A generates P_B, for B, and signs it with K^-_A.

Token	Delegatee PKC sig	Delegator PKC sig.	Object	Access Rights	Life-span	Token Signature
$\mathbf{P_B}$	sig^C_B	sig^C_A	sig^P_A	ar'	dates	sig^P_B (K^-_A).

A's SAProxy, P_A, and A's PKC, C_A, are bound into B's SAProxy by A's signature.

B generates his request, R_B, to use his authority, P_B, to access S and signs it with K^-_B.

Token	Receiver	Requester PKC sig.	Object	Access Rights	Life-span	Token Signature
R_B	S	sig^C_B	sig^P_B	*request*	dates	sig^R_B (K^-_B)

B then presents his request, with the set of tokens[2] needed to verify it to the server. These tokens are:

$$P_A \, C_A \, P_B \, C_B \, R_B$$

C_B and the request from B are verified as in the previous example for A. Then sig^P_B is checked using K^+_A. Assuming sig^P_B is valid and that B referred to in the SAProxy, P_B, is the same B that presented the request, R_B, then the second SAProxy, P_A, is verified. This is repeated for all SAProxies in the cascade.

B can delegate to another user C in the same way as A, augmenting the cascade of SAProxies from one user to another. Cascades may be restricted by including delegation rights as well as access rights in the SAProxies - then any user in the cascade may restrict this right to the next user and thereby stop the cascade. This allows for non-transferable SAProxies.

The notion of cascading SAProxies to delegate authority may also be applied to represent combined authority to request an operation. In this situation, authority is not 'cascaded' but 'chained' to indicate that several principals are taking joint responsibility for an action [Guan et al. 1991]. Thus if it takes two principals A and D, to authorise an action, this can be represented by a request from one that is endorsed by the other. Assume that A's request above (section 3) also needs D's authority before it is granted. Then the following request from D to S, signed by K^-_D, endorses A's request to S.

Token	Receiver	Requester PKC sig.	Object	Access Rights	Life-span	Token Signature
R_D	S	sig^C_D	sig^R_A	sig^P_D	dates	sig^R_D (K^-_D)

In this request statement D is saying to S to grant the request in R_A within the constraints in D's own SAProxy, P_D, as referred to by sig^P_D.

It is then up to the service provider to ensure that the SAProxies required by the security policy have been received.

The above examples show how SAProxies can be used to confine the operations that a server performs both by confining the amount of trust delegated to a principal and by not trusting the authority of just one principal.

6. User Authentication, Access Control and Resource Management

SAProxies may also represent a role holder identity. In particular, a group of users can be set up as service administrators. For example a service, S, may recognise more than one principal as its manager. S issues these principals with SAProxies. These managers may be distributed throughout the system and they become responsible for granting access to the users of the service. S leaves all dealings with the user's to its managers and is only interested in ensuring

[2] If the server keeps any of the frequently used tokens locally, then they need not be sent in the transaction, thus improving efficiency. However there are dangers in keeping tokens locally, and so the risks in doing this must be weighed against any savings made.

that each transaction is authorised by a manager with the appropriate SAProxy.

All services can be viewed as resources and so users need to be allocated 'an amount' of the resource available to them and then must be charged for what they use. SAProxies can have the amount of the resource available to the user embedded in the SAProxy. The user entitled to use a service may have this right limited by the amount of funds in his SAProxy. The service provider can then ensure that a user request does not exceed this limit. A user that has access to a resource may delegate authority to a principal that is not totally trusted, secure in the knowledge that if he restricts the amount of resource available to that user then there is only a limited amount of damage the user can do. Repeated use of SAProxies can be prevented by recording the number of attempts to use a SAProxy within a period of time. Similar techniques to avoid double spending in NetCash are discussed in [Medvinsky, Neuman 1993].

A server may use a resource administrator and delegate to it the responsibility for sharing the resource fairly. All requests to the server must then have a SAProxy from this administrator in the cascade. Alternatively, a server can allocate the funds available between several resource administrators and then they become responsible for sharing their funds with the users they manage. All actions can be confidently attributed to those involved as all requests are signed by each principal's private key.

When an operation has been completed the service provider must charge the user (or the user's system) for the use of the service. In a distributed system this will require firm proof that the service was indeed requested and supplied as there may not be much trust between the systems.

A verifiable log of the request is available in the audit trail and can be sent to the user's system. This may be copied to all users in the cascade as well. This proves that the request was made and the audit trail can form a part of the invoice when charging for the service.

7. Summary

The benefits of SAProxies arise from the use of public key cryptography. The process of binding the PKC to a proxy gives it the self-authenticating property. The fact that each principal has his own private key, known only to himself and yet verifiable by the rest of the world is what gives SAProxies their independence from the system infrastructure and from any particular security domain or authentication server at the time of access. A SAProxy authorises a principal to access a service by having the access rights within the SAProxy. The user's actions, and hence any malicious damage by the user or an intruder in the domain, is confined to the rights in the token. The user presenting the request authorised by the SAProxy can be authenticated because the request is signed by that user's private key. Delegation of authority and joint authority can be represented within the mechanism. Attribution of actions performed on presenting SAProxies is possible because a principal's private key is never shared with any other principal and so any request signed by a principal's private key can be attributed to that principal. Theft of tokens is made unprofitable by nesting their signatures. Tampering with the SAProxy or requests can be detected because the signature is dependent on all the bits of the data signed. Where operations performed are logged with the SAProxies that authorised them, a verifiable audit trail of actions is available which makes both the servers and clients accountable. If SAProxies are extended to include allocation of funds, then they adapt easily to enable resource management.

The function of the CA is to generate a PKC for a principal; this is performed off-line and only once[3] per principal. Generation and verification of a SAProxy may be done locally by any principal so that principals become their own authentication servers. All that is needed is a local genuine copy of the public key of the CAs involved and a local trusted version of the public key algorithm.

The trustworthiness of the SAProxy scheme described is dependent on the following factors: the public algorithm used, the integrity of the CAs and the security of the private keys.

The public key algorithm used must be strong, as must the keys generated. This particularly

[3] A principal can of course generate a new public key pair whenever he wants, but within any given set of transactions a PKC need only be generated once.

applies to the CA's key pair. The longevity of the public keys (particularly the CAs') has to be determined i.e. how long before the private key can be determined from text and the public key.

The security of all the CAs' private keys must be maintained. If a CA's private key is compromised then anyone who has knowledge of it can certify any public key, including replacing a valid user's public key with a bogus key. Until the foul play is detected actions can be mistakenly attributed to that user. Consequently, the CA administrators should be trusted by the users in its domain and the certification procedures stringent. However, even if a CA is violated, this does not make users in any other domain vulnerable to attack because an impostor cannot acquire any more rights than would be granted to the principal he is impersonating. When the integrity of the CA is re-established (with a new public key pair) users in the violated CA's domain can have their public keys re-certified and new SAProxies (which refer to the new PKCs) issued. The private keys of users are not compromised by any attack on a CA and the validity of any transaction can still be correctly determined.

The third factor is the ability and motivation of the principals to keep their private keys secret. This assumption sounds highly unrealistic but it is based on the reasoning that it benefits each principal to keep his private key secure and that if a principal's private key is compromised, then the impostor can do no more than that principal is entitled to do and no one else can be attributed with that action other than the user whose key was compromised.

There are also practical factors which need to be considered. The computational requirements and performance costs of using a public key algorithm such as RSA [Rivest et al. 1978] and the amount of extra data in a message generated by the extra tokens need to be assessed in any particular approach. These two factors need to be weighed against the flexibility of the scheme, the reduction in the number of messages because there is no continual need to access authentication servers (especially across domains) and the reduction in the amount of trust principals have to place in remote systems' authentication servers. Another practical problem is key management, the availability of a key generation mechanism for all the principals and the difficulties that arise when changing a CA's key pair. The scheme described in this paper does not solve the revocation problem but solutions that are application dependent may be found, for example by giving SAProxies a life-span and a unique number.

Mechanisms involving the use of SAProxies appear to solve many of the difficulties encountered by comparable schemes as well as providing a flexible and comprehensive means to protection and resource management.

8. Conclusion

In this paper, we have shown that by combining the properties of public key encryption with cascading proxies a single mechanism, SAProxies, can provide a unified and consistent approach to authentication, access control and resource management. SAProxies are more suited to an open and heterogeneous environment than are other comparable systems because they exploit the properties (independence from trusted server, localised trust, local verification) of public key encryption. Principals are authenticated locally by the service providers and not by an authentication server within the domain of the client principal, resulting in a reduction in the number of messages exchanged within a transaction. This is particularly significant in cross-domain transactions.

SAProxies show a way of putting together three elements of protection, authentication, access control and resource management, which up to now have always been viewed as separate functions. The flexibility of the mechanism and its independence from the system infrastructure enables the integrity of computer supported co-operative work to be maintained in an open and heterogeneous environment.

References

Bull J., Gong Li., Sollins K., (1992). Towards Security in an Open System Federation. In: *European Symposium for Research in Computer Security ESORICS '92*. Toulouse, France, 23 Nov 1992. Springer-Verlag Lecture Notes in Computer Science.

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

Gong, L., (1990), *Cryptographic Protocols for Distributed Systems.* Cambridge: Jesus College. Thesis (PhD).

Guan S., Abdel-Wahab H., Calingaert P., (1991), Jointly-Owned Objects for Collaboration: Operating SystemSupport and Protection Model. *Journal of Systems and Software.* Volume 16, No. 2, pp85-95.

Karger, P.A., (1988), *Improving Security and Performance for Capability Systems.* Cambridge: Computer Laboratory. Thesis (PhD).

Lampson B.W., (1971). Protection. In: *Proceedings of the Fifth Princeton Symposium on Information Sciences and Systems,*, Princeton University, March 1971, pp.437-443; reprinted in Operating systems Review, 8, 1, January 1974.

Low M.R., Christianson B., January 1994, A Technique for authentication, access control and resource management in open distributed systems. *IEE Electronics Letters. Vol.30* No.2 pp124-125.

Medvinsky G., Neuman B.C., 1993. NetCash: A design for practical electronic currency on the Internet. In: *Proceedings of the 1st Conference on Computer and Communications Security.* VA, USA. November 1993. ACM.

Mullender S.J., (1985), *Principles of Distributed Operating System Design.* Amsterdam: Vrije Universiteit te Amsterdam. Thesis (PhD).

Needham R.M., Schroeder M.D., (1978). Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM 21 (12)* pp. 993-999. December 1978.

Neuman N.C., (1991), *Proxy-Based Authorisation and Accounting for Distributed Systems.* Seattle: University of Washington, Department of Computer Science and Engineering. Technical Report 91-02-01.

Rivest R.L., Shamir A., Adleman L., February (1978). A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM. Volume 21,* Number 2, pp 120-126.

Sollins K.R., (1988), Cascaded Authentication. In: *Proceedings of the IEEE Symposium on Security and Privacy.* Oakland, CA, USA. April 1988. IEEE Computer Society Press, Los Alamitos, CA, USA pp156-163

Steiner J.G., Neuman C., Schiller J.I., (1988). Kerberos: An Authentication Service for Open Network Systems. In: *Proceedings of the USENIX Winter Conference 1988.* Dallas, Texas. Feb 9-12, 1988.