A Joint Authorisation Scheme

Marie Rose Low, James A. Malcolm

University of Hertfordshire
(M.R.Low, J.A.Malcolm)@herts.ac.uk

Abstract

There are many situations where more than one principal needs to give authorisation so that a single function can take place. Self-Authenticating Proxies provide a mechanism which may be suitably employed to support the requirements of joint authorisation.

1. Introduction

In Computer Supported Co-operative Working (CSCW) a single task usually involves more than one principal. Where several parties are involved in a task it is important that the participants have the authority to carry out their responsibilities and that each party's activities are restricted to those they are authorised to perform. CSCW does not imply only that a task is divided into distinct jobs each carried out by a different principal: there are many situations where a single activity requires that two or more principals agree jointly that it should occur. Authorisation policies in such situations can become very complex, as can the implementation of access control mechanisms to effect these policies.

The objective of this paper is to show that the use of Self-Authenticating Proxies (SAProxies) [4, 5] would satisfy many of the requirements for joint authorisation as identified by Varadharajan and Allen in [8].

In the following section we discuss situations where joint authorisation is necessary. In section 3 we describe the requirements placed on a scheme that supports joint authorisation. This is followed by a brief description of the SAProxy scheme and illustrations of how this scheme may be applied. Section 6 concludes the paper with a discussion on how the SAProxy scheme satisfies the requirements identified.

2 The Need for Joint Authorisation

An authorisation policy defines the rights given to those governed by that policy. There are situations where a policy states that the authority to perform a single action must come from more than one party e.g. two keys must be held by two different parties who come together to open a safe. This means that the authority from both parties must be joined together to make a single request in order to satisfy the policy.

The following situations are instances where joint authorisation is required.

- joint authors submitting a paper for publication.
- presenting two forms of identification to prove who you are.

- a principal having authority to perform an action because one party had granted him authority to act in a role and another has granted the role authority to perform an action.
- a customer giving someone a cheque and the bank guaranteeing his cheque.
- a subscriber to a journal authorised to read a journal by an administrator who is authorised to act as such by a publisher.

Sometimes joint authorisation can be represented as serial delegation of authority, but other cases require a different representation. Although delegation is involved, in that two parties grant authority, there has also to be a means of matching and combining this authorisation to satisfy the policy for a particular action. Joint authorisation usually implies active participation of several principals in an action. In such a case e.g. when two authors submit a paper, evidence of authorisation must be presented in parallel in order to fully express the joint action. This requires that two tokens of authorisation be bound together in some logical, unforgeable manner to form the authorisation necessary to perform a single action. In some cases e.g. the guaranteed cheque, the joint authorisation from the account holder and the bank may be represented as delegation of authority in series. Serial authorisation is easier to represent with typical delegation schemes such as in [3] and [9] than parallel authorisation. As will be shown later both of these are possible using the SAProxy scheme.

3. Requirements for Joint Authorisation

In [8], Varadharajan and Allen identify the following attributes necessary in a scheme which allows joint authorisation. It is important that any such scheme provides a suitable level of abstraction for an implementation and that the range of applicability of the scheme is wide. It must be able to support a wide range of applications and must take into account the availability of the underlying technology. It is also important that there is a logical separation between an authorisation policy and the implementation of that policy. In particular, a scheme must be able to enforce different forms of authorisation policies and not just a specific form. Further considerations they identify are:

- that not all participants may be available at the same time to perform the joint function, thus an action which is jointly authorised may require grouping of the different authorisations.
- that authority granted must be tied to a specific action so that this authority is not misused.
- that the authorisation from different principals must obviously apply to the one function i.e. they must all agree to the same objective. Even the set of participants may have to be specified and agreed.
- that the parties involved need to be notified of any action they are involved in and to be informed of any error handling.

Varadharajan and Allen then go on to describe and discuss three basic joint action schemes based on a medical example where two physicians are required to admit one patient to a hospital. In the first scheme, one physician, designated the primary physician, passes a 'ticket' to the other who then requests that the patient is admitted and a new patient object is created. This is viewed as delegation, where the first physician passes his authority to the second physician. In the second scheme, both physicians send their requests to a central joint action authority which is trusted to act only in legitimate circumstances. The joint action authority must be able to match disparate requests into groups before it dispatches the joint requests to admit the patient. In the third example both physicians send their individual requests to the authority that admits patients, whose responsibility it is to match the requests and put together the final action. In this last case there is really no support for joint based authorisation i.e. there is no merging of the requests to form a single action.

4. The SAProxy Scheme

Let us now consider the SAProxy authorisation scheme [4, 5] as a means of providing joint authorisation. The SAProxy scheme is a mechanism, based on tokens of authorisation and public key encryption [7]. An SAProxy token is an extended capability naming the principal to whom it is issued, the object in question and the access rights. It is digitally signed [2] by the issuer of the SAProxy. This scheme allows principals to define and delegate access rights, for objects over which they have authority, to a particular party. This is done in such a way that all authority granted to access information, a service or resource, is attributable to the party that granted it. All requests for access using SAProxies is also attributable to the requester.

SAProxies may be bound together in order to express further delegation of authority and combined authorisation. Thus, a SAProxy field may refer to another SAProxy instead of holding the name of a principal, object or access rights. SAProxies are referred to by their digital signatures which provide a unique and verifiable pointer mechanism.

The authority is unforgeable as the SAProxy tokens are digitally signed and can be produced and verified locally by each principal. Only the principal to whom the token is issued can use it, so there is no problem with propagation. Requests, and the authorising SAProxies, can be recorded to provide an audit trail of the actions of all principals. These features make SAProxies particularly suitable for expressing joint authorisation in tasks where the participants are working independently in a distributed environment. The following is a brief description of the mechanism. For a full description see [6].

4.1 The SAProxy Mechanism

SAProxies are based on public key encryption, so all principals involved must have a public key pair. Public keys are certified and signed by certification authorities.

A principal can then be identified and referred to by his public key certificate (PKC), because these are unique, and the PKC itself can be referred to by its signature block, e.g. a user A can be referred to by his PKC signature, $sigC_A$ [1].

SAProxies, represent the access rights to an object as delegated by the issuer of the SAProxy, to a party named in the SAProxy. SAProxies have the following fields: the identity of the principal it is issued to (the delegate), the identity of the issuer (the delegator), the object (resource/ information) and the access rights that are being granted to that object. The life-span of the SAProxy is not shown for simplicity. Each SAProxy is signed by the issuer and the signature block forms part of the token.

To illustrate the use of SAProxies, consider a principal A, with PKC C_A, who has authority over an object and wishes to grant write and read access to principal B with PKC C_B. The SAProxy, $D_{B:A}$, would be as follows:

$D_{B:A}$: $sigC_B$ $sigC_A$ *object* *write/ read*

The SAProxy is signed with the delegator's private key. Thus the SAProxy signature block is

$sigD_{B:A}$: $\{sigC_B$ $sigC_A$ *object* *write/ read*$\}K_A^-$

Anyone can obtain A's PKC and use A's public key to verify that A signed the SAProxy and that it has not been tampered with. This can be done in a domain that is remote from the one in which the SAProxy was generated.

Just as a principal can be referred to by the digital signature of his PKC, a SAProxy can be referred to by its signature. Within a single SAProxy, the information shown above need not be explicitly stated but may be referred to by other SAProxies. If B now delegates his own authority to C, B does not refer to the object explicitly, but refers to $D_{B:A}$. In doing so, B is automatically binding into his delegation of authority, proof of his own authority. This is shown in B's token, $D_{C:B}$, generated for C:

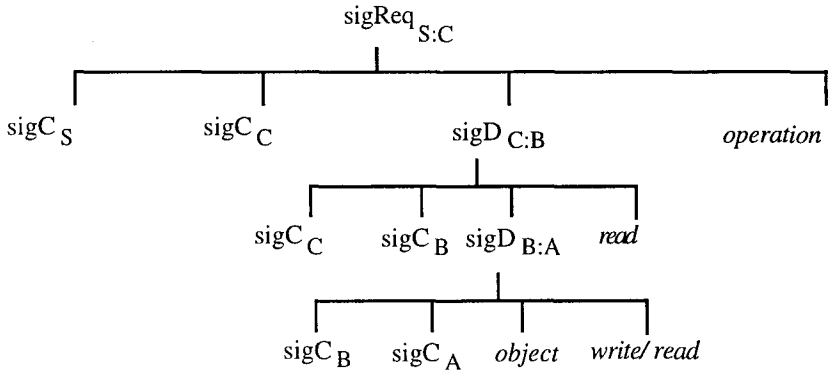
$D_{C:B}$: $sigC_C$ $sigC_B$ $sigD_{B:A}$ *read*

Thus these tokens of delegation can be bound together to show the grounds upon which access to an object is granted. It is important to note that because these SAProxies are signed by each issuer, they can be verified by any party, with the appropriate PKC, and their origin determined. Thus authority granted in a SAProxy cannot be repudiated by the grantor.

To read the object, C then sends his tokens of authorisation to the object server with his request. C has to send his SAProxy from B, $D_{C:B}$, and the authorisation SAProxy for B from A, $D_{B:A}$, together with the operation he wants to perform. He binds these together by signing them with his private key to show that he is the true issuer of the tokens. This request can therefore also be expressed in SAProxy format. The object server, S, is delegated authority by C to act on his behalf on the object referred to (indirectly in this case) within the constraints of the requested operation. The request from C to S signed with C's private key is:

$Req_{S:C}$: $sigC_S$ $sigC_C$ $sigD_{C:B}$ *operation*

The object server checks that the line of authority, through the SAProxies can be followed, that the access rights delegated are not enlarged and that the operation requested falls within those allowed to the requester (in this case only read is allowed). This is shown in the following tree:



This request is also attributable to its issuer and a principal cannot deny that he has made a request or that he has delegated authority.

4.2 Using Roles

The SAProxy scheme allows for the definition of roles. A role can be named and authority granted to that role instead of to a principal. Then a principal who is granted authority to act in a role can use the authority assigned to that role.

A principal G, with authority to assign roles to other principals, can issue a SAProxy as follows:

$R_{B:G}$: $sigC_B$ $sigC_G$ *null* W

In $R_{B:G}$, G authorises principal B to act in role W. The token is signed by G.

Delegation is applied not only to principals, but also to roles although these need to be bound to or invoked by a principal before the authority delegated can be exercised.

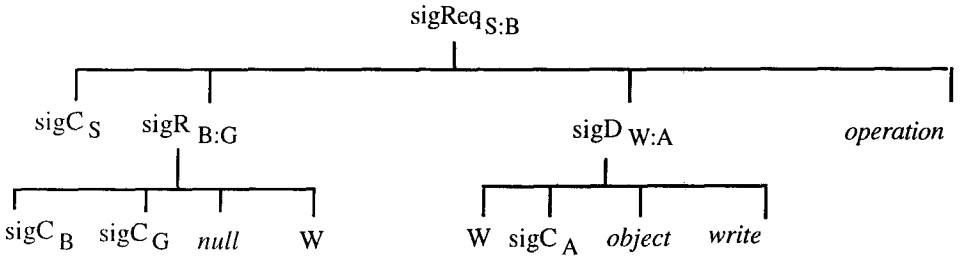
Now A can grant access to his object to the role W instead of directly to B.

$D_{W:A}$: W $sigC_A$ *object* *write*

To write to the object, B then has to send all his tokens of authorisation to the object server, S, with his request. B has to send his role SAProxy from G, $R_{B:G}$, and the authorisation SAProxy for W from A, $D_{W:A}$, together with the operation he wants to perform. He binds these together by signing them with his private key to show that he is the true owner of the tokens. The request from B to S signed with C's private key is:

$Req_{S:B}$: $sigC_S$ $sigC_B$ $sigD_{W:A}$ *operation*

The object server checks that the role, W, in $D_{W:A}$ is the same role, W, granted to B in $R_{B:G}$. The line of authority can be seen in the following tree:



The only operation that W is authorised to request is write.

The role, W, may be assigned to different principals without having to redefine W's authority. The SAProxy scheme allows the following; more than one party can give rights to a role; these rights can be dynamically assigned to the role; the rights required to perform an object operation need not be set in an object's access control list and finally, a consistent approach is used at different levels of authorisation which makes the enforcement and auditability of this authorisation scheme easy to implement and verify.

5. Using SAProxies for Joint Authorisation

In this section we illustrate how SAProxies may be used to satisfy the requirements for joint authorisation identified in [8]. One of the examples we use is the same medical example used in that paper, where two physicians must act together to admit a patient. We use the concept of delegation, as applied in SAProxies, for both long-lived and short-lived associations. Delegation implies the granting of a subset of authority, whatever the result i.e. the fact that accretion of rights may take place when more than one party delegates a subset of its authority, does not deny the fact that each party has in fact delegated its authority.

Let us consider the first example used in [8] and described in section 3 above. The physicians, S and J, are assigned the role of primary physician, PPH, and physician, PH, respectively by the hospital administrator, H, in a role SAProxy.

$R_{S:H}$: $sigC_S$ $sigC_H$ *null* PPH

$R_{J:H}$: $sigC_J$ $sigC_H$ *null* PH

These role SAProxies are then used in all further tokens to indicate that any actions performed by S and J are authorised in their capacity as physicians. It is a matter of hospital policy whether the authority granted to a role, such as PH, is implied, whether it is explicitly stated in another SAProxy or whether it is a combination of both. For the moment, for the sake of simplicity, we shall assume that the authority of a physician to admit a patient is implied in the role PH. In order to follow the example accurately, S, the primary physician, issues a SAProxy to J granting his authority to admit patient, Fred:

$D_{J:S}$: $sigR_{J:H}$ $sigR_{S:H}$ *Fred* *admit*

J then forms a request SAProxy to the server, F, responsible for the creation of patient objects and their registration. This request includes S's SAProxy authorising Fred to be admitted. J signs the request in his capacity as a physician, thus endorsing S's SAProxy and providing authorisation from both the physicians.

$Req_{F:J}$: $sigC_F$ $sigR_{J:H}$ $sigD_{J:S}$ *admit*

This way of passing authority from one physician to another does not actually represent the full semantics of joint authorisation, where each physician has equal authority. It is the equivalent of having two keys to a safe, and one key holder giving his key to the other. Physician J does not in fact explicitly state his authority, but instead implicitly gives it by endorsing S's authority with his signature.

SAProxies can be used to give a better expression of joint authorisation. There is no need to distinguish between the roles of the physicians other than one physician must take responsibility for actually putting together the request to admit a patient. Thus S's role SAProxy now assigns to S the role of PH.

$R_{S:H}$: $sigC_S$ $sigC_H$ *null* PH

Each physician gives his authority to admit the patient, Fred, in a SAProxy to the factory, F.

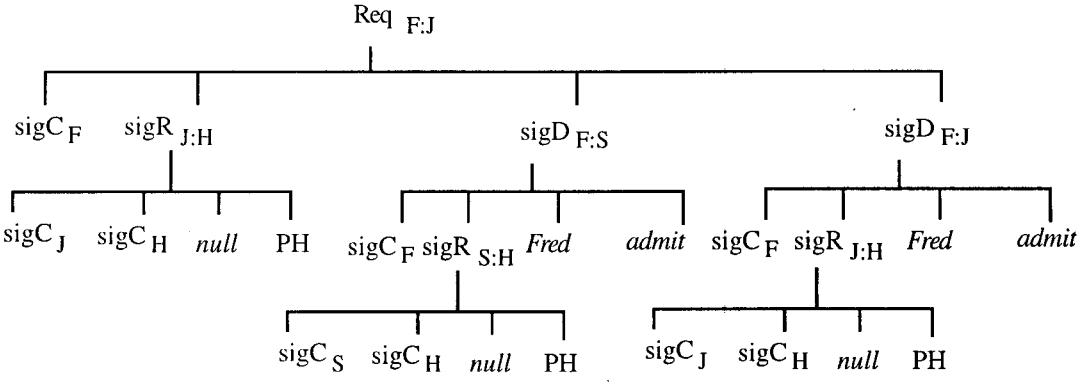
$D_{F:S}$: $sigC_F$ $sigR_{S:H}$ *Fred* *admit*

$D_{F:J}$: $sigC_F$ $sigR_{J:H}$ *Fred* *admit*

Let us then assume that J puts together the request to the factory, thus making it a true joint action. This is also done in SAProxy format and is signed with J's private key. In this case both the object to be operated on and the operation to be performed are referenced indirectly via the SAProxies.

$Req_{F:J}$: $sigC_F$ $sigR_{J:H}$ $sigD_{F:J}$ $sigD_{F:S}$

The following tree of SAProxies shows the line of authority.



In this situation neither physicians delegates to the other - each gives his consent that the patient is admitted to the factory. As the SAProxies are unforgeable, neither physician can put in a request without the genuine consent of the other.

It is easy to see there is no significant difference if the policy requires that a joint action authority (JAA) is to be used to form the joint request. Both physicians may use exactly the same SAProxies and the JAA forms the request in the same way as physician J did. As each party is identified by their PKC and each SAProxy is attributable to its issuer the authority cannot be misused. It is, however, necessary in this case to check against the current policy when verifying the SAProxies. There is no delegation of authority to the JAA itself and so the line of authority through the SAProxy tree is not obvious. It is only by checking that the JAA is authorised to put together joint requests that the tokens can be validated. It is, of course, possible for the physicians to delegate their authority to the JAA itself and not to the factory. Then the JAA acts as a delegate and puts in the request on behalf of the physicians e.g.

$D_{JAA:S}$: $sigC_{JAA}$ $sigR_{S:H}$ *patient* *admit*

$D_{JAA:J}$: $sigC_{JAA}$ $sigR_{J:H}$ *patient* *admit*

The two physicians grant the JAA authority to admit the patient and the JAA puts in the request to the factory.

$Req_{F:JAA}$: $sigC_F$ $sigC_{JAA}$ $sigD_{JAA:J}$ $sigD_{JAA:S}$

Thus SAProxies may be employed to suit many forms of policies. Obviously, for any request to be accepted, the tree of SAProxies must be checked to ensure that each party is acting in a role it has been given, that all SAProxies refer to the same patient and that the current policy is being enforced. As the tree of SAProxies is available to all, this is no problem.

5.1 Further uses of SAProxies

SAProxies may be used to represent joint authorisation in the situations described in section 2.

Joint authors submitting a paper for publication would take much the same form as two physicians admitting a patient. We have shown in section 4.2 how two distinct authorities can grant a role the rights to an object and a principal the right to act in that role.

Financial systems is another area where the use of SAProxies is suitable for representing joint actions. We now illustrate this by using SAProxies to guarantee an electronic cheque.

Consider SAProxies that have in the access rights field the 'amount of funds' that the SAProxy owner is authorised to use. This SAProxy is like a physical cheque and can be used in much the same way that we use cheques and bank cards. The use of such SAProxies is dependent on the policies followed by the service provider and its clients. A cheque SAProxy, $Q_{S:C}$, from a client, C, to a service provider, S, to pay for a service would have the following information.

$Q_{S:C}$: $sigC_S$ $sigC_C$ *account* *amount*

C's bank, K, may guarantee this cheque SAProxy by endorsing it with another SAProxy, $Q_{C:K}$, in order to guarantee payment to the service provider.

$Q_{C:K}$: $sigC_C$ $sigC_K$ $Q_{S:C}$ *null*

This guaranteed cheque may be sent as payment to a service provider when a transaction has taken place or it may also be bound in with the SAProxy that requests the service. The recipient is then authorised, both by the bank and the account holder, to withdraw the stated amount of funds from the bank.

6. SAProxies and the Requirements of Joint Authorisation

The SAProxy scheme, as has been shown, is widely applicable. It provides a flexible mechanism to implement many kinds of joint authorisation policies. It is not dependent on any specific technology, as all that is required is a public key algorithm and this may be implemented on any system. The flexibility of the scheme is such that any form of authorisation policy may be enforced by it. An authorisation policy is not part of the implementation of the scheme to enforce that policy.

Participants in a transaction do not need to be available at the same time to perform a joint function. Although the format of SAProxies that has been used has not included the life-time of each token, the full format of the token does allow for the time during which the token is valid to be included. As the tokens are signed and therefore unforgeable, there is no danger that they may be used outside the validity period. SAProxies can therefore be generated before they are required so that all participants do not have to be available at the same time. Only the principal putting in the request for the action, whether that is a single physician or a JAA, need be available at this stage.

The authority granted in a SAProxy is stated specifically and can be defined in detail as SAProxies may be bound together. The line of authority and the objectives of the SAProxies can be traced through a tree of SAProxies, so all that is needed is to ensure that joint authorisations refer to the same objective.

Varadharajan and Allen identify the need for notification to the parties involved that an action has been performed. As the SAProxy mechanism allows for logging of security tokens, this requirement has not been addressed specifically in this paper. We suggest that whether and how this is done is a matter of policy and not does have an impact on the scheme used for joint authorisation. Notification may be given to the requester of an action or to each participant involved in the action.

Thus, we have shown that the SAProxy scheme goes beyond the typical delegation scheme as in [3] and [9] in that it enables true joint authorisation to be represented precisely and that it is a flexible tool with which to provide joint authorisation in a wide range of applications.

References

[1] Christianson B., Low M.R. Key-spoofing attacks on nested signature blocks. *IEE Electronics Letters*, 31(13):1043-1044, June 1995.

[2] Diffie W., Hellman M.E. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644-654, November 1976.

[3] Gasser M., McDermott E. An Architecture for Practical Delegation in a Distributed System. In: *Proceedings of the IEEE Symposium on Security and Privacy,* pages 20-30, Oakland, CA, USA. 7-9 May 1990. IEEE Computer Society Press.

[4] Low M.R., Christianson B. A Technique for authentication, access control and resource management in open distributed systems *IEE Electronics Letters* 30(2):124-125 January 1994

[5] Low M.R., Christianson B. Self Authenticating Proxies *Computer Journal* 37(5):422-428 October 1994

[6] Low M.R. *Self Defence in Open Systems: Protecting and Sharing Resources in a Distributed Open Environment* Hatfield: University of Hertfordshire, Computer Science Division. Thesis (PhD), September 1994.

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

[8] Varadharajan V., Allen P. Joint Actions Based Authorisation Schemes. *ACM Operating Systems Review* 30(3):32-45, July 1996

[9] Varadharajan V., Allen P., Black S. An Analysis of the Proxy Problem in Distributed Systems *Proceedings of the 1991 Symposium on Security and Privacy*, Oakland, CA, USA IEEE Computer Society Press pp255-275, 20-22 May 1991