Towards Security in an Open Systems Federation

John A Bull[1] Li Gong[2] Karen R Sollins[3]

This paper argues that security design for Open Distributed Processing (ODP) would benefit from a shift of focus from the infrastructure to individual servers as the owners and enforcers of security policy. It debates the policy nuances, mechanisms, and protocol design consequences, that would follow from such a change of emphasis. In ODP, physically separate systems federate into heterogeneous networks of unlimited scale, so there can be no central authority, nor ubiquitous security infrastructure. Servers that offer, trade, supply and consume services must maintain their own security policies and defend themselves. For servers to take security policy and enforcement decisions, design is concerned with how they might seek advice and guidance from higher authority. This contrasts with an administrator imposed policy on a closed homogeneous network, where an infrastructure enforces administrator declared access rights to potential clients, including rights to delegate rights.

1 Introduction

Computer system security originated in the context of single multi-user systems, to be later refined in the context of closed homogeneous networks. This history has led to a focus on users: the consumers of service. As we move towards open system federations, this paper argues that focus should be re-directed towards servers: the suppliers of service. This amounts to a shift of emphasis, from the demand side, to the supply side of the computer service economy.

Systems participating in ODP will wish to trade services with other systems, yet defend themselves against attack from them. This is manifest as systems participate in open trade of services, with dynamic binding between supplier and consumer. Ultimately, each system will define its own security policy and control invocations of itself by the outside world. The logical conclusion from reducing granularity implies individual servers setting and enforcing their own security policies.

Most emerging ODP frameworks are object-based [2, 13, 14]. Encapsulation is enforced by the infrastructure, but beyond this: *you don't manage objects, objects manage themselves* [4, 25]. Each object is an instance of an abstract type with internal state. Each maintains its own integrity, and each might also maintain its own security policy, thus creating *secure enclaves*. This offers the prospect of dynamically variable security policies tailored for individual objects. It also offers scope for local policy enforcement, including immediate access revocation.

Although different approaches might meet the same ultimate goal, this paper shows how a subtle change of perspective from user to server can lead to different design solutions and eventually to different styles of implementation.

---

1 Architecture Projects Management Ltd, (APM/ANSA),
Poseidon House, Castle Park, Cambridge CB3 0RD, UK (jab@ansa.co.uk)

2 ORA Corporation,
301A Dates Drive, Ithaca, New York 14850, USA (li@oracorp.com)

3 M.I.T. Laboratory for Computer Science,
545 Technology Square, Cambridge, MA 02139, USA (sollins@lcs.mit.edu)

2 The Object Model

Activities within objects may invoke operations in interfaces of other objects, but for later convenience we may refer loosely to objects invoking objects. Object state may only be changed through invocation of an operation of an interface. Object encapsulation, necessarily enforced by a host platform, prevents any other form of access. For any given invocation, the object hosting the invoking activity is a client, and that offering the interface is a server. Concurrently, an object may respond to multiple clients and initiate multiple requests for service.

A client requests service by invoking a server Interface Reference (IR) through an infrastructure call. IRs are created by servers for their own services, and may be passed as parameters, or returned as results, in IR invocations. Thus, an IR may be created by a server, returned as a result, passed from object to object along a chain of service, and finally invoked, making the server the source and ultimate target of IRs.

3 Security Models

When an IR is invoked, the server must check the client's authenticity and authority to obtain service. These will be decided, respectively, by the server, or a trusted agent of the server, checking a shared secret presented by the client, and by checking a record that authority had been granted to the client. The client may have acquired authority by delegation, in which case the same checks must apply to the delegator. Following various checks, the server must decide, based on some security policy, whether to grant or deny access. In all cases, although the server ultimately has control, it may seek advice and guidance on any aspect of authenticity, authority, or security policy, from trusted third parties.

A possible model to support this requirement is for a common ancestor of the client and server in a bootstrapping situation, or a shared authority in inter-domain co-operation, to retain an access control list (ACL) for the services offered by the server. An ACL entry would take the form of client identity versus services available. The ancestor would distribute secrets (usually keys) to the offspring for purposes of authentication. The server would verify client identity by checking a supplied key. Either this would be done by keys shared directly between client and server, or by the server consulting a trusted authentication service that stores keys against identities. In summary, given a service request by way of a client invocation of an IR, the server action would be to check client identity using a supplied key and to check client authority by reference to the ACL. In some cases, such as where no ACL entry exists, the server may need to check the identity and authority of any delegator, and apply a security policy partially based on delegation rights.

In an alternative model, the server would create a sealed and signed certificate for whatever it chooses to regard as a unit of security, and return this with an IR. Each transfer of the certificate, which would probably, though not necessarily, accompany transfer of the IR, would similarly be signed. When the IR is invoked, the certificate, with any accumulated signatures, would be presented as an authority to obtain service. This certificate behaves like a capability created and issued by the server but with some essential differences: that, on presentation, the server could check itself as the originator; that, in the first instance, the certificate is assigned to a specified recipient; and that nested signatures can be used subsequently to trace any transfers. The server can check accumulated delegation information against a security policy.

These models are not mutually exclusive. A certificate need not originate from the server, provided the server has the means to check the authority of the originator with an ACL entry. Where a certificate is transferred to a recipient that has no ACL entry, that delegation would have to be confirmed with a signed transfer. Where a certificate is transferred to a recipient known to have an ACL entry, there would be no need to authorise the delegation. In all circumstances, the server controls access, even though reference to higher authority may be needed to make a decision.

This paper will concentrate on the second of the above two models. In effect, the model allows ACL entries, which might otherwise be maintained by the server, to migrate through the distributed system. In other words, each server issues identity-based capabilities for services offered. This neatly reconciles the capability versus ACL controversy, with both being represented in the same architecture. But it will be argued that the capability view leads to favourable scaling characteristics, and that the server-based, rather than infrastructure-based view, produces solutions more suited to servers defending themselves in an ODP scenario.

Not all security concerns are addressed explicitly in this paper, although the suggested model would not preclude development of other services, such as audit, non-repudiation, etc. In particular, confidentiality can be obtained through encryption once keys are distributed. Denial of service is not addressed: this being a particularly difficult subject area for most security models. Recovery, either from a breach of security or network failure is not considered, although decentralisation to servers should lead to designs resilient to such events.

The next section compares and contrasts these models in greater detail. Later sections concentrate on the merits and effects of changing emphasis to the server-based view, and discuss implementation in some object-based architectures, such as ANSA [2, 4]. In the next section, we contrast the consequences of viewing security from each of the infrastructure and server perspectives. In a later section, we show the flexibility and ease with which each object controls its own security policy. We then, mainly as proof of concept, describe protocols which illustrate how taking the server view leads to a different style of implementation, and to significant economies of mechanism. The paper concludes with suggestions for further study.

4 Different Views of Security

In most traditional systems, service access is controlled by a management authority that applies a central security policy. An example might be a certification authority that hands out certificates to potential clients as permitted by the policy. The service provider is expected to supply service on presentation of a valid certificate. For example, in Kerberos [27], it is the Kerberos server that issues a ticket for a client to use a service, not the service provider itself. In traditional capability-based systems, such as in CAP and Hydra (see [18]), and to some extent in Flex [30], a capability for accessing an object is not issued by the object itself, but by a separate authority, such as the operating system. In such an infrastructure-based view of security, service access is granted when a certificate is created. Non-distributed systems, and homogeneous networks, were built on notions of central control consistent with this view, which still has strong influence in many frameworks (eg [7/8, 14]).

For systems co-operating within distributed federations, many presumptions in favour of centralisation have to be reversed. A server-based view is needed, where a service provider retains control of security policy and decides whether to grant access at the place and time of supply. The service may seek assistance from other services (eg authentication, authorisation). In turn, these services may request assistance from other services, such as to contact similar services in other security domains.

The remainder of this section compares and contrasts the infrastructure-based and server-based views.

4.1 The Server-based View

In a typical object-based system, a server offers (or advertises) its service through an IR which is either returned as a result, or passed as a parameter in some other service invocation. In particular, an IR may be passed to a service broker or trader [3]. A potential client holding an IR, including a trader, may pass it to other potential clients for later dynamic binding to the server. IRs may pass along chains of service until finally invoked, thus leading to a return to the server. An IR may be distributed widely, and copied, including into different security domains.

This matches closely a model where, for some server chosen unit of security, a server issues a certificate, to a potential client, which is delegated from client to client until finally presented as a service request authority. It behaves as a capability that could be passed with an IR, except for the addition of some semantics that would allow the server to confirm source, content, and trace delegations.

The server can always decide whether to, and how to, respond to an interface invocation at the time of invocation. Assuming the server can validate the service chain to the final client, it can own and apply a security policy. The server does not have to make the decision completely unaided but may seek help from authentication and authorisation services. But always the server, and not some infrastructure service, is in control, and the server decides the assistance to seek and the variations to apply.

Security requirements can be declared by the application designer as part of the computational object. This may take the form of attributes, declared in an interface, which cause invocation of further authorisation, authentication, and policy services. This is consistent with an object model that hides function and data behind interfaces. The decision coded into the server may be based on many criteria, such as the time of day, or the physical path of a service chain (assuming such data can also be obtained with validity guarantees). Special context dependent security constraints, based on object state, could be added explicitly. Alternatively, a null security policy could be declared which would imply no additional overhead.

This style of server-based model allows a high degree of autonomy so that servers may migrate, and servers may be imported into systems, without concern about, or the server having to conform to, an imposed security infrastructure.

4.2 The Infrastructure-based View

Where security is assumed to be imposed by the infrastructure, a client (including the user) acquires rights by applying to an authority who decides, based on some security policy, whether to hand out certificates. Once rights are assigned, the server is expected to honour any service request for which a valid authority is presented. In this case, the certificate does not originate from the server, but from an ancestor of, or mentor of, the server. Certificates may be delegated, but, to decide the validity of delegation, the server must appeal to a higher authority for access control arbitration.

This view becomes less reasonable as systems participate in larger federations. In particular, when systems are not controlled by a single authority, but equal peers co-operate, this view alone may not be implementable. There is always likely to be a fallback to a server-based view, even though the server in this case may be of rather macroscopic proportions, such as an entire homogeneous network.

4.3 Infrastructure and Server Views Compared and Contrasted

A chain of service is always cyclic. The loop may be viewed either as starting from a client and passing through a sequence of servers before returning to the client, or as starting from a server and passing through a sequence of potential clients before returning to the server. The difference of emphasis arises from where the loop is broken for purposes of system design. In both cases, the security manager may decide what goes into a policy server and how policy servers relate to one another. Workable solutions will emerge either way, but both views should be considered. The distinction is analogous to managing the economy: there is only one economy, but for different purposes it is convenient to view it from either the supply or demand side.

An infrastructure-based solution leads to policies imposed on the server through a hierarchy of greater and greater authority. A server-based solution leads to servers submitting themselves to security policies by seeking advice and guidance from wiser and wiser mentors. It is analogous to a company attempting to impose security through management decree, but, in the limit, security will only be achieved by individuals choosing to submit to the policy. In devising solutions, it is necessary to recognise the power of the individual, or the server. Taking a server-based view can lead to solutions of greater simplicity, elegance and efficiency.

An important consequence of starting from a server view is that access control is applied by the supplier at the time and place of potential supply. Starting from a client view, with an ACL entry presumed to be extant in a policy server, leads to access being granted by management authority prior to request for supply. Hence, choosing where to break the service chain loop for purposes of design can lead to different solutions, with rather different characteristics. For example, where a certificate is granted by a certification authority, it is difficult (but not impossible) to revoke or vary its provisions while it is extant (see [8, 15] for various solutions). It is easier to see how to solve this problem where the server is always the focus of control. In fact, in the general case, other dynamic changes of security policy may be permitted, since policy is not applied until service is sought.

A further difference emerges when using the traditional access matrix [16] to examine the position. In the access matrix, a *subject* is an active entity accessing a passive *object*[4], such as a file. In an object-based model, the *object* is an active interface with implied semantics. Problems arise when trying to decide the *subject*. Access rights represent the rights of a *subject* to invoke an object interface. From an infrastructure-based view, the most appropriate choice of *subject* would appear to be the activity extant on behalf of the user (ie, a user's delegation) [17]. From a server-based view, the *subject* is, or is chosen from, the chain of delegation; it is the security policy encapsulated in the server that determines the *subject*. A *subject* may be one of a number of clients, including end users, or many clients in association, where each *object* is also a possible *subject*. This view fits better with object-based architectures.

---

[4] *Subject* and *object* in italics are used in their historical context of the access matrix, as per the reference. *Object* should not be confused with object as in object-based technologies.

5 Implementation from the Server Perspective

This section discusses some system design issues that arise when taking a server-based view of security. In particular, it discusses how a server-based implementation could be realised in object-based architectures evolving in a direction consistent with this paper, ANSA [2, 4] being one example. Protocols are described later.

5.1 Service Traders - Offers and Inquiries

A creator of a server will hold an IR to at least one server interface. The precise engineering for this is not of concern to this paper. The creator will also be returned an access certificate (AC) to authorise access to a service represented by the IR. This AC need not necessarily authorise the creator, but may authorise a client for whom the creator is acting. When the IR is invoked, other IRs and ACs may be returned to authorise access to other clients, probably for different services. To do this, the server may well act as a client, seeking guidance from other servers. Servers may register IRs and ACs with, and advertise through, a trader, which may, such as in ANSA [3], allow federation over multiple domains. A trader acts as a broker between service offers and inquiries. The trader itself is a server, and will have issued IRs to allow invocation of its trading services.

5.2 Propagation of Service Offers

From a server-based view, all service offers start from, and end at, a server. For the moment, ignore issues of object birth and object death, and assume that a client has requested a service (for which it must have an IR to invoke and an AC to present). The server wishes to create and return an IR, and an AC, as part of its response.

An AC associated with a returned IR authorises service to a client whose identity may not be that of the immediately invoking client. In this context, an identity is a name for an authentication key that may be retrieved by an authentication service (if not the client itself) when the IR is invoked.

The server associates with the AC a particular security policy to apply when the IR is invoked. This is done by embedding a policy identifier in the AC. Associated with this identifier will be a private policy key stored by the server. The server then computes a cryptographic seal of the AC with the key and returns the AC together with the seal. In effect, what is returned is a ticket that allows a request to be made for access to the interface, but which does not guarantee access. The ticket indicates an assignee, associates its issue with any special restrictions (ie policy), and is sealed in an unforgeable way with the private policy key of the server.

The client to whom the AC is issued can present it when requesting service. The client can also propagate it to another potential client by adding a signature of transfer (signed with its private key). A transferred AC can be transferred further; each transfer being very specific about the identity of the intended recipient. This creates a chain of delegation that can be traced and checked on IR invocation. It does not matter what route a transfer takes to reach a final client. An intermediate object through which an AC passes will only be of interest if that object could have had authority to invoke the IR with a traceable AC delegation. To delegate authority, a delegator must add a signature, otherwise the AC would be unusable by any further recipients. Possible protocols, and a format for the signature of transfer, are given in a later protocol section.

It would be possible to embed rights (such as access or transfer rights) in an AC in a more specific way than the server identifying a policy to apply on IR invocation. As suggested in [9, 26], it would then be possible for delegators of an AC to restrict or amplify rights. This may have merit in multi-level access control, where a delegator could request that a delegatee have a specific clearance before being granted access. However, when taking a strict server-based view, it may be better to maintain policy entirely within the server. The server is the final arbiter of policy, so any restrictions placed on route could only be advisory. The issue is open to debate.

5.3 A Mechanism for Transferring Authority

The integrity of a transfer of authority can be guaranteed with well-known techniques to sign and/or seal information using cryptographic methods, such as encryption (both conventional and public-key) and one-way (hash) functions [29]. The communicating objects may use a third party authentication service to arrange key sharing [21].

A basic sign and seal mechanism can be used to underpin all transfer of authority. It can be illustrated with a pseudo-random function f with the property that given f and f(m) it is computationally not feasible to compute m [23]. Thus, given a family of such functions f_k, indexed by key k from a key space, two parties who share k can authenticate to each other the origin and integrity of m by supplying m and $f_k(m)$ as a signature. A pseudo-random function can be approximated with a one-way hash function H, or an encryption E. The signature is H(k, m), or $E_k(m)$ (m encrypted with key k). Sometimes, $E_k(H(m))$ may suffice. Note that a pair of (m, H(k, m)) is self-identifying in that a server who knows k can verify the signature. To show that a signature is recent (ie that it is fresh), m could include a time-stamp or a nonce [10]. Also, if m has traversed a chain of services, the identities of the services on the chain can be appended to m, and signing done, by nesting several applications of f. For example, a signature signed with key k1, and then signed with k2, looks like:

(id2, id1, m, H(k1, id1, m), H(k2, id2, id1, m, H(k1, id1, m))).

The protocols are described later in more detail.

A special case arises when a source and destination coincide. This happens when an AC created by a server migrates through a chain of service, eventually to be presented as an authority for service. The same pattern of sign and seal above applies, except in this case the key is private to the server. This is significant, because the original key used to create the AC (k1 in the above example) need never be distributed outside the server. Thus, there is much less risk of disclosure. This is a cyclic version of cascaded authentication as discussed in [26]. This case merits special attention, because it is the normal case in a server-based view of security. It is also the most secure since it does not rely on a shared secret, only a private secret[5]. Each object maintains its own secrets and should one be compromised there should be no direct effect on any other. There should only ever be a need for a server to authenticate a chain of service that it originates.

5.4 Handling Service Requests

When an AC is presented on an IR invocation, the server validates the signatures and decides how to respond according to a policy. If the server knows the private keys of the transfer signatures - for example when the signatures are signed with a public-key

---

[5] "Three may keep a secret if two of them are dead". Benjamin Franklin (1706-1790)

system, and the public keys are stored in a public place - the server retrieves the appropriate keys and re-computes the seal to validate the chain of signatures. In some cases, an authentication service may be consulted. For example, when using a pseudo-random function with a private key, the signature of a client may be known only to an authentication service and not to other objects. Presenting the AC to the authentication server for validation is itself a delegation of an AC using the same sign and seal mechanism. Finally, the server validates the request using its own service specific private key originally used to create the AC.

If the validation is successful, the server knows the chain of delegation from the client identities embedded in the AC. It may now apply a security policy based, in part, on the policy identifier originally embedded in the AC. The policy can easily be server specific, since each server has final responsibility for security in its own domain. Moreover, there may be further levels of control imposed by the server, based on context dependent criteria, which may be special to an application, and not necessarily included in a general model. For example, there may be access policies which depend on time of day, or relationships between service requests and their originators.

Although a simple single chain of delegation has been illustrated, it is possible for multiple chains to be created, since any object may transfer authority to multiple destinations, and may do so with many chains concurrent. A chain of delegation is therefore not a simple linear structure but a hierarchy, with any object possessing authority being able to behave as a node in the hierarchy.

Assuming domain managers have exchanged keys, so that their authentication services may co-operate, trusted gateways may be created to allow security domains to integrate. When an authentication server is presented with a number of identities and a nested seal for verification, the authentication server may not hold all the necessary signatures or keys, in which case assistance from other authentication services may be needed. This process takes exactly the same pattern as any other requests to an authentication service. Authentication servers may be equal peers in a federated world. A request from one authentication service to another will appear the same as any other client-server request. Authentication may recurse through a chain of authentication servers, and through a number of security domains.

5.5 Revocation of Transferred Authority

By using an AC to invoke a revocation operation of a server interface, a client within a chain of transfers can request the server to revoke its subsequent transfers of the AC. A server (or its agent) would remember node points where a limb of the transfer hierarchy tree has to be severed. Since signatures are self identifying, the full path of an AC is known when it finally arrives. Given a request from any client in the path, the server, as the arbiter of policy, can agree (or indeed refuse) to deny access.

Revocation has been difficult in infrastructure-based models, especially if service is apparently guaranteed when a certificate is issued [15]. A server-based model supports revocation well, since access is granted at the point and time of service supply. As a result, refreshing the security policy cache (eg [8]) is not difficult. Revocation while a server is in process of supplying service (ie immediate revocation) is also possible. The implications of this on other aspects of an ODP model (outside a security context) would need to be considered, although it is unlikely to be worse than an apparently uncontrolled death of a server.

Compared with an ACL model or a capability model, a signed AC may be thought of either as an ACL entry that is allowed to migrate, or as a capability that embeds an identity. For distributed systems, this is an advance on both older models, since allowing an access list entry to migrate solves a significant scaling problem. A transfer of authority can happen without immediate reference to the server, since a security policy is applied only when a service is invoked. Also, all necessary authentication can be done at one time; again when the service is invoked. Use of identity-based capabilities (suggested in the ICAP architecture [9]) is useful for purposes of controlling the transfer and revocation of privilege because the chain of identities reveals the chain of delegation.

6 Control of Security Policy

The unit of security policy represented by an AC is chosen by the server when the AC is created. When the AC is presented (when an IR is invoked to cause a return to the server) a chain of delegation becomes available. This allows a range of policies:

- allow access always. This would apply to a service truly open to anyone. A trading service is an obvious case.
  Policy: open access; trust everyone.
- allow access only to the client to whom the AC was originally granted. In this case no delegation would be permitted by the client first specified.
  Policy: trust only those specifically designated.
- allow access only if all clients in a chain of delegation are authentic.
  Policy: trust others trusted by those you trust.
- allow access only if the final client in a chain of delegation is authentic, regardless of the authenticity of any other clients in the chain.
  Policy: trust anyone you know to be trustworthy, regardless of his sources.
- allow access only if clients in a chain of delegation are from some pre-defined set of the local security domain.
  Policy: trust a known trustworthy group, but avoid external threats.
- allow access only if clients in a chain of delegation are from some pre-defined set, where this set could include clients from other security domains.
  Policy: trust a known trustworthy group, and allow external threats.

Servers can apply a range of possible discretionary to mandatory policies against separately chosen units of security policy. The rules are, in effect, built into the server, although, as has been emphasised before, servers may call upon other servers to obtain them. Furthermore, these other servers comply with the same model, and set their own security policies. Hence, policy rules need not be static data structures, but can be modified dynamically. Other criteria might also apply, including application specific criteria.

In a server-based view, the society of servers as a whole has a characteristic behaviour that might be thought of as a system policy. For a system design, the default case may be to construct all servers to redirect access checks to other policy servers. Furthermore, the default may be to make this transparent to the application designer, who may relax this security transparency selectively. Thus, system policy is pervasive in the design structure, rather than being imposed through an infrastructure.

6.1 Locality of Control and Encapsulation of Security Policy

If an object interface is rigorously defined, then, subject to acceptable levels of reliability and efficiency, one is concerned with what functions the object performs, not with how it performs them. Similarly, subject to an audit of the object and its invoked security services, a client should be concerned with what security policy an object offers; not with how it offers it. Therefore, a server-based view of security is consistent with an object model that hides function and data behind interfaces. This leads to the following principle of locality of control of authority.

In an object-based model a principle of encapsulation applies whereby an object always controls its own destiny [2, 4, 25]. This may be loosely translated as: *you don't manage objects; objects manage themselves*. In the context of security, this would become a principle of locality (of control) of authority, whereby an object always controls transfer of authority across its encapsulation boundary. The object, acting as a server, decides when to create and pass an AC and, in doing so, decides when to pass a service authority to a potential client. It also decides when to accept an authority as a legitimate request for service and what policy rules to apply. The principle of locality of authority puts the server in control of creating tokens of authority, checking legitimacy of access to its services, and of maintaining its own security policy, although it is not obliged to do these entirely unaided. This principle of control of flow of authority out of and into an encapsulation boundary was proposed in [20], although there it was set in the context of a traditional non-object-based database model.

When an object is created it is formed from a template. An instantiation interface is invoked to fill in initial values from given parameters. At this point, a shared key for mutual authentication with the parent is passed to the child, and this, effectively, establishes the server's separate identity. An identity name may be created, but this only indicates an authentication key to retrieve.

It is possible for an object to be created such that it would only ever allow access to a pre-designated client (which need not necessarily be its parent). This implies a security policy defined when the system is conceived, with no provision for any subsequent discretionary delegation; hence, this implies a mandatory security policy. At the other extreme, possession of an AC could always imply a right to delegate. This is a fully discretionary situation.

There is a midway position where delegation of the AC may not be allowed, but where the server could interpret it as right for the client to be returned a different AC, to a different unit of security policy. This implies a limited discretionary policy[6]. It could be used to apply different security policies to different classes of service, such as where management facilities control access to basic service facilities.

Finally, a server could choose to control access to a number of services, and apply a range of policies to them. Mandatory policies could apply in some cases; fully discretionary in others. A possible application is to make management services mandatory but basic services discretionary. It is unlikely that such full generality would be contemplated in any system design, but this is not precluded.

---

[6] Other models with similar objectives (eg [28]) have called this an extended discretionary policy, but this seems to be a misnomer. An in-between position seems either to be a relaxed mandatory policy, or a restricted discretionary one.

6.2 The Role of User

The human user is assigned an initial responder object in the system. This object, when created, has a human identity embodied in it (through a password key in this case), and it is built to reflect the security policy the human wishes to apply. In effect, the object is a system clone of the user. Thus, the server-based model accords well with ordinary human society, where each individual is responsible for setting and enforcing his/her own security policy.

The issue is illustrated well in a (most interesting) study by Abadi et al [1], which considers what characteristics a smart card must have for a human possessing the card to authenticate mutually with a system, and then further be able to delegate privilege. It transpires that complete integration of the human would imply a smart card with functionality equivalent to an object that is fully part of the distributed system. For reverse authentication - that is, for the human to authenticate the system - the human (smart card) would need to issue the equivalent of ACs to the system. The study [1] also suggests how various limited forms of smart card could be used to advantage. Without such a card one is limited to a user object in the system conducting some form of password authentication protocol with the user, together with its contingent problems.

7 Protocols

This section presents some example protocols that embody the principles discussed in the previous sections. It focuses on the benefits of viewing AC authorisation from a server-based perspective. Also, at various times, usually when a service in invoked, there is a need to authenticate a chain of authority. A cascaded authentication protocol can be used for this purpose with the two significant advantages: that a whole chain can be presented to an authentication service in a single package; and that authorisation and authentication protocols can be integrated to achieve a significant economy of mechanism. An initial analysis of the protocols with the BAN/GNY logics [5, 11] shows that the protocols meet their goals.

7.1 Authentication

It is easier to show the cascaded authentication protocol first from a client point of view, since this is consistent with how most other papers present the subject. Later discussion will show how the protocol can be used by a server to authenticate a chain of delegation that it initiates, and finally satisfies. In both cases there is a loop back to the originator. Suppose a client C invokes servers S1, S2, etc, then the loop is:
$C \Rightarrow S1 \Rightarrow S2 \Rightarrow S3 \Rightarrow S \rightarrow S3 \rightarrow S2 \rightarrow S1 \rightarrow C$, where $\Rightarrow$ is invokes, and $\rightarrow$ is responds to.
From the view of a server S delegating to clients C1, C2, etc, the loop would be:
$S \equiv C1 \equiv C2 \equiv C3 \Rightarrow S$, where $\equiv$ implies delegates to, and $\Rightarrow$ again implies invokes.

Figure 1 presents a cascaded invocation example, in which client U invokes server C, which in turn invokes server S. The following is a generalised and modified version of the Otway-Rees protocol [22]. In all protocols presented here, H(I, J) indicates a one-way hash function (OWHF) as discussed earlier, where (I, J) represents a concatenation of all arguments. K is an authentication (strictly a validation) server; Uk is U's key known to U and K, and Un is a nonce chosen by U. The steps of the authentication protocol are:

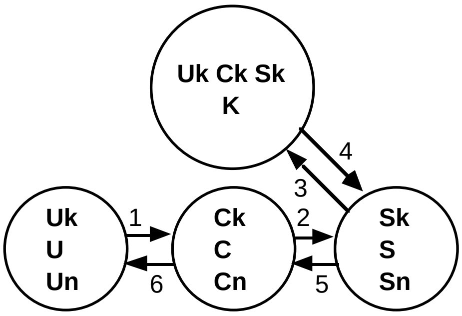

Fig 1. Asymmetric cascaded authentication

1. U → C : U, H(Uk, C), Un
2. C → S : U, C, H(Ck, S, U, H(Uk, C), Un), Un, Cn
3. S → K : U, C, S, H(Sk, K, U, C, H(Ck, S, U, H(Uk, C), Un), Un, Cn), Un, Cn, Sn
4. K → S : H(Sk, Sn, U), H(Ck, Cn, U), H(Uk, Un, C)
5. S → C : H(Ck, Cn, U), H(Uk, Un, C)
6. C → U : H(Uk, Un, C)

In this protocol C and S act as carriers between U and K. At step 2, K validates that the message could only have come jointly from both U and C, since only U could create H(Uk, C), and only C could nest it in H(Ck, S, U, H(Uk, C), Un). Similar logic can be applied to step 3. At step 6, the message to U still validates that C must have been the original recipient of the message at step 1 since, after being satisfied about C's identity, only K could have inserted C's identity and U's nonce in H(Uk, Un, C).

There are two practical differences between this and the Otway-Rees protocol. First, this protocol uses OWHFs rather than encryption. Second, U's message to K is nested in C's message to K. In the Otway-Rees protocol a *common challenge* was used to tie the two participants' messages together for presentation to K.

It is possible that nesting could introduce a weakness with reversible encryption when there is predictability in the input data (such as with English text). Many uses of the same encryption function might help a dictionary attack. This is not a serious concern with OWHFs. A cryptographic signature could originate from an infinity of possible sources, so finding the exact inverse is difficult. On the other hand, since an infinity of possible sources could generate the same signature, a valid seal does not 100% guarantee the originality of the attached message. But given a sufficiently large signature field, the chance of erroneous validation would be small. The precise size of field to choose is a subject of much further debate, and maybe further research, in information theory [eg 23]. A 128 bit cryptographic signature is generally thought to be sufficient for most practical purposes.

The protocol of Otway-Rees introduced an asymmetry from earlier protocols to reduce the total number of messages sent[7], and the cascaded version presented here has similar merit. The introduction of nesting, to replace the *common challenge*, allows the protocol to be cascaded easily, as suggested in [26], and, as will be shown, leads to an economy of mechanism when overlaid with the delegation proposal.

---

[7] For an example, compare the Otway-Rees [21] with the Needham-Schroeder protocol [22].

7.2 Delegation

This sub-section presents a server-based view of delegation. An AC originated by a server is signed and sealed by passing it through a OWHF. At each stage, a signature may be added, and the package sealed, by nesting again through a OWHF.

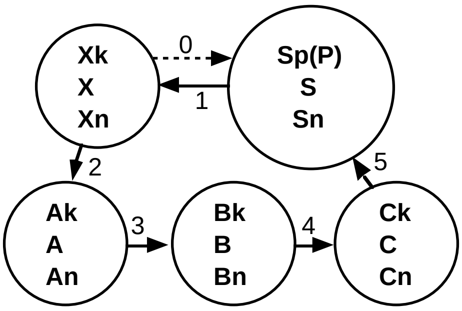

Fig 2: Transfer of authority

Figure 2 shows a server S creating an AC to return to X. The chosen unit of security policy P of S is indicated by a private key Sp. If X is providing some service to A, X may request that A's identity be embedded in the AC. The signed and sealed AC would then take the form P, H(Sp, A).

This AC is now transferred from A to B to C through a chain of delegation for final presentation by S. At each step a signature is formed by taking the accumulated data, adding a client authentication key, passing this through a OWHF to produce a cryptographic seal, and then adding this seal and the clear-text client identity to the accumulated data before sending. In full, the steps are:

```
1. S →X : A, P, H(Sp, A)
2. X →A : A, P, H(Sp, A)
3. A →B : B, A, P, H(Sp, A), H(Ak, B, P, H(Sp, A))
4. B →C : C, B, A, P, H(Sp, A), H(Bk, C, H(Ak, B, P, H(Sp, A)))
5. C →S : S, C, B, A, P, H(Sp, A), H(Ck, S, H(Bk, C, H(Ak, B, P, H(Sp, A))))
```

When the AC, with all attached signatures, finally returns to S, S can present the whole package for authentication to a server, K. K uses the clear-text client identities to retrieve its copies of the relevant shared keys. A transfer from S to K, and possibly from K to K' to check signatures from another domain, uses the same sign and seal protocol, with signatures of S and K added at the relevant steps.

7.3 Combining Authentication and Delegation

It may be observed that the mechanisms for authentication and delegation are very similar, and may be integrated. To overlay the cascaded authentication protocol, only a nonce need be added at each delegation step. For example, where {.} is the sequence representing the original AC (given in full in step 1; abbreviated in step 2) the original delegation from S to X, and then to A, would take the form:

```
1. S →X : S, {A, P, H(Sp, A)}, H(Sk, X, {A, P, H(Sp, A)}), Sn
2. X →A : S, X, {.}, H(Xk, A, S, H(Sk, X, {.}), Sn), Sn, Xn
```

A basic transfer of an AC then takes the form:

Source id, Destination id, Data, H(Source key, Destination id, Data), nonce
where Data includes the original P, H(Sp, A).

A delegator of an AC must identify its target client (or possibly client group). When the service is finally invoked, the server must be able to authenticate the chain of delegation, from itself, back to itself, and thereby have the identities of potential clients for applying a security policy. It is this control of export and import of authority out of and into a server that is significant in considering the design from the server perspective. However, as was pointed out very early in this paper, it is possible to integrate this with an ACL-based scheme, where a signature may not necessarily be needed at every step. Also, for particular system designs, security policies could be fixed so that the full delegation generality might not always be needed.

8 Key Management

To complete the picture, this section offers a brief discussion of two issues in key distribution. The first is distribution of a new key to be used by an object A, and known only to A and the authentication server K. The second is distribution by K of a conversation key to be used by objects A and B. Both are based on use of the *exclusive or* ($\oplus$) operation.

The protocol for distribution of a new key to A is based on a master key Am that is used only for this purpose. The new key is denoted by Ak. Kt, Kn, and An are nonces. The protocol is as follows:

```
1. A →K : An
2. K →A : Kt, Ak⊕H(Am, An, Kt), H(Ak, Kt), Kn
3. A →K : H(Ak, Kn)
4. K →A : H(Ak, An)
```

In step 2, K generates a new key Ak and uses the master key Am to pass it to A. Since A knows Am and An, and is given Kt, A can generate $x = H(Am, An, Kt)$ and use this to recover Ak from $x \oplus (Ak \oplus H(Am, An, Kt))$. A then uses H(Ak, Kt) to check that the message had not been intercepted, the components split, and an invalid (spoof) field substituted for $Ak \oplus H(Am, Kt)$; that is, to check that Ak was the key that K had intended to send. The nonce An protects against replay of a message 2.

The nonce Kt serves two purposes. First, without Kt the message would read $Ak \oplus H(Am)$, and since H(Am) could be used to recover later keys, discovering H(Am) would be just as useful as discovering Am. Discovering H(Am, Kt) is not so useful, since a different Kt each time makes H(Am, Kt) a once only value.

Second, Kt ties the message components together to protect against them being separated and substituted during transmission. This works the same way as the *common challenge* in the Otway-Rees protocol introduced in §7.1. Here, however, nesting components is not an alternative.

If Am is discovered, any keys distributed using Am are also potentially compromised. However, just knowing Am does not compromise other keys; Am has to be used in conjunction with recorded protocols. Conversely, if any Ak, or any sequence Ak1, Ak2, Ak3, etc, is discovered, this cannot compromise Am. This would only allow recovery of H(Am, Kt), rather than Am, which is of limited concern because of the once only use of Kt.

Strictly, message 1 is all that is necessary to distribute the new key. The two further messages, 2 and 3, provide some confirmation of receipt. They protect against earlier messages not having reached their destinations. Both A and K would have to use an earlier sub-key until each knows that they both have the new key.

The protocol is necessary and sufficient for the purpose described, although it is of interest to observe that it is not complete. A does not know what K knows about A's state of knowledge; that is, {A does not know that {K knows that {A knows that {K knows the new key}}}}. A complete state of common knowledge (requiring an infinity of acknowledge responses) is unattainable [see 12].

Slight protocol variations are possible. For example, if, before being prepared to use the new key, K and A insisted on receipt of messages 2 and 3, the component H(Ak, Kt) of message 1 could be omitted. Its function concerning validation of Ak would be inherent in the subsequent messages.

Most authentication protocols demonstrate how a key distribution server distributes a session key for use between two communicating objects. Although this is not strictly needed in the model outlined in the paper, it can be a basis for support of confidentiality. It is presented here using a OWHF [10]. Here Ks is the session or conversation key:

1. A → B : A, H(Ak, B), An
2. B → K : A, B, H(Bk, K, A, H(Ak, B), An), An, Bn
3. K → B : H(Bk, A, Bn)⊕Ks, H(Bk, A, Bn, Ks), H(Ak, B, An)⊕Ks, H(Ak, B, An, Ks)
4. B → A : H(Ak, B, An)⊕Ks, H(Ak, B, An, Ks)

After message 2, it is only K that can check the OWHF, since only K knows both Ak and Bk. The order of nesting specifies that it must have come by way of A to B to K. K now has to send a session key Ks back to A and B. The keys Ak and Bk ensure that the messages in 3 could only have been generated by K. The nonces An and Bn ensure no replay of earlier messages. Also, only A and B, respectively, could encode H(Ak, B, An) and H(Bk, A, Bm) to find Ks. H(Ak, B, An, Ks) and H(Bk, A, Bn, Ks) can then be checked by A and B to ensure that the messages were not interfered with, and that a valid Ks was recovered. Only A and B could recover the same Ks, so both are happy that they have a common session key. A further message would be needed if B wished to know that A really has received Ks, but this is a trivial addition using Ks under H: for example H(Ks, B).

Various protocols are possible, offering different characteristics, with either A or K initiating a sequence. Efficiencies may be achieved with protocol components used in parallel, even though needing more of them. Also there may be variations in the knowledge states of the communicating objects on protocol completion.

9 Conclusions

As ODP begins to emerge, this paper suggests that it may be more appropriate to consider security from the perspective of the server, rather than the infrastructure, as has been traditional. Although both aim at the same ultimate goal, taking a server-based view of the issues could lead to more appropriate system design solutions.

An ODP environment consists of physically separated systems, integrated into heterogeneous networks of any scale. In such an environment, the focus must be on the server defending itself against outside abuse, while still being prepared to trade services with other systems. Furthermore, as scale increases, it becomes essential for clients to locate and bind dynamically with outside services. Systems will wish to co-operate, yet remain autonomous, and subscribe to different security policy regimes. Looking at the problem from the service supply side, rather than the more traditional user demand side, suggests how possible solutions might evolve. This suggests prospects of dynamically variable security policies based on local and fine grain data relationships and semantics. It suggests scope for local policy enforcement and easy immediate revocation of privilege.

This paper also discusses how a server-based view could be implemented in an object-based architecture, such as ANSA [2, 4]. There is scope for further study in this area, particularly in the design of cryptographic algorithms, key management, and protocols based on a simple use of cryptographic seals and signatures, rather than reversible encryption. Such protocols are of interest where legal restrictions would prevent full encryption of data messages. They are of particular merit where a cycle allows a protocol based on private, rather than shared keys. Identity-based crypto-systems [24], and subliminal channels, could also be explored in the context of this paper. These offer the means to engineer signature schemes based purely on identity information without shared keys.

Acknowledgements

The research project at MIT was supported by The Defense Advanced Research Projects Agency (DARPA) through NASA Grant NAG 2-582 and through DARPA contract number DABT63-92-C-0002. APM is financially supported by all sponsors of ANSA, who also provide a valuable forum for technical debate.

References

1. M. Abadi, M. Burrows, C. Kaufman, and B. Lampson
Authentication and Delegation with Smart-Cards
DEC Systems Research Center, Report No.67, Oct 1990.

2. Architecture Projects Management Ltd (ANSA)
The Application Programmers' Introduction to the Architecture
Technical Reports TR 017.00, 1991.

3. R.J. van der Linden and J. Sventek
The ANSA Trading Service
IEEE Distributed Processing Committee Newsletter, Vol. 13, No. 4, 1991.

4. J.A. Bull
Object Request Broker
RFP Response OMG: 91.1.2, to the Object Management Group.
Architecture Projects Management Ltd, CO.059.00, 08 Jan 1991.

5. M. Burrows, M. Abadi, R.M. Needham
A Logic for Authentication
ACM Trans. on Computer Systems, Vol.8, No.1, Feb 1990, pp.18-36.

6. ECMA Security in Open Systems: A Security Framework.
ECMA TR 46, Jul 1988.

7. ECMA Security in Open Systems: Data Elements and Service Definitions
ECMA TR 138, Dec 1989.

8. M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson
The Digital Distributed System Security Architecture
Proc. of the 12th National Computer Security Conference
NIST/NCSC, Oct 1989, pages 305-319.

9. L. Gong
A Secure Identity-Based Capability System
Proc. of the IEEE Symp. on Security and Privacy
Oakland, California, May 1989, pp 56-63

10. L. Gong
Using One-Way Functions for Authentication
ACM Computer Comms Review, Vol.19, No.5, Oct 1989, pp.8-11.

11. L. Gong, R. Needham, and R. Yahalom
Reasoning about Belief in Cryptographic Protocols
Proc. of the IEEE 1990 Symp. on Security and Privacy
Oakland, California, May 1990, pp.234-248.

12. J.Y. Halpern and Y. Moses
Knowledge and Common Knowledge in a Distributed Environment
Proc. of the 3rd ACM Symp. on Principles of Distributed Computing
Vancouver, British Columbia, Aug 1984, pp.50-61.

13. ISO ODP Basic Reference Model of Open Distributed Processing
Part 2: Descriptive Model
ISO/IEC JTC1/SC21/WG7 N315, Mar 1991 (and later revisions) and
ISO/IEC JTC1/SC21 N6079 (draft), May 1991 (and later revisions)
Part 3: Prescriptive Model
ISO/IEC JTC1/SC21 N6080 (draft). May 1991 (and later revisions).

14. Working draft Security Frameworks Overview
ISO/IEC JTC1/SC21 N6166 and N6080, Jul 1991.

15. P.A. Karger
New Methods for Immediate Revocation
Proc. of the IEEE Symp. on Security and Privacy
Oakland, California, May 1989, pp.48-55.

16. B.W. Lampson
Protection
Proc. of the 5th Princeton Symp. on Info. Sciences and Systems, March, 1971.
Reprinted in ACM Operating. Systems Review, Vol.8, No.1, Jan 1974, pp.18-24.

17. B. Lampson, M. Abadi, M. Burrows, and E. Wobber
Authentication in Distributed Systems: Theory and Practice
Proc. of the 13th ACM Symp. on Operating Systems Principles, Oct 1991
Publ. as ACM Op. Systems Review, Vol 25, No.5, pp.165-182.
Also as, DEC Systems Research Center, Report No.83, Feb 1992.

18. H.M. Levy
Capability-Based Computer Systems
Digital Press, 1984.

19. R.C. Merkle
Protocols for Public Key Crypto-systems
Proc. of the IEEE Symp. on Security and Privacy
Oakland, California, May 1980, pp.122-134.

20. N.H. Minsky
Selective and Locally Controlled Transport of Privilege
ACM Trans on Prog Langs and Systs, Vol.6, No.4, Oct 1984, pp.573-602.

21. R.M. Needham and M.D. Schroeder
Using Encryption for Authentication in Large Networks of Computers
Communications of the ACM, Vol.21, No.12, Dec 1978, pp.993-999.

22. D.J. Otway and O. Rees
Efficient and Timely Mutual Authentication
ACM Op. System Review. Vol.21, No.1, Jan 1987, pp.8-10.

23. R.L. Rivest
Cryptography
In J. van Leeuwen, editor: Handbook of Theoretical Computer Science
Volume A, Algorithms and Complexity, chapter 13, pages 717-755.
Elsevier Science Publishers B. V., 1990.

24. A. Shamir
Identity Based Crypto-systems and Signature Schemes
Proc. of Crypto'84, Aug 1984, pp.47-53.

25. A. Snyder
Inheritance and the Development of Encapsulated Software Components
In B. Shriver and P. Wegner ed.:
Research Directions in Object-Oriented Programming
MIT Press, 1987, pp.165-188.

26. K. Sollins
Cascaded Authentication
Proc. of the IEEE Symp. on Security and Privacy
Oakland, California, Apr 1988, pp.156-163.

27. J.G. Steiner, C. Neuman, and J.I. Schiller
Kerberos: An Authentication Service for Open Network Systems
Proc. of the USENIX Winter Conference, Feb 1988, pp.191-202.

28. S.T. Vintner
Extended Discretionary Access Controls
Proc. of the IEEE Symp. on Security and Privacy
Oakland, California, Apr 1988, pp.39-49.

29. V.L. Vodyock and S.T. Kent
Security Mechanisms in High-Level Network Protocols
ACM Computing Surveys, Vol.15, No.2, Jun 1983, pp.135-171.

30. S. Wiseman
A Secure Capability Computer System
Proc. of the IEEE Symp. on Security and Privacy
Oakland, California, Apr 1986, pp.86-94.