APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

FollowMe

Mobile Objects Security

Will Harwood

Abstract

This report provides an analysis of some of the issues involved in producing secure mobile objects and offers solutions to the problems encountered. The central problem in producing usable, secure mobile object systems is to find a means by which objects may have limited or partial trust in their execution hosts yet still be able to carryout useful tasks on behalf of their users as the objects move from host to host.

Approved 3 September, 1998
Architecture Report

Distribution:
Supersedes:
Superseded by:

Table of Contents

1 Introduction

1.1 Mobility as a Programming Metaphor

The design of software is driven by metaphor. Mobile objects (or agents) comes from the extension of Alan Kay's "little people" metaphor of object based computing, in which objects are "little people" that talk to one another forming a community of friends. Adding mobility grants the little people the possibility of movement and of dynamically forming new groupings based on physical co-location. Choosing to design software from such a perspective has advantages compared to remote communication, but in doing so it becomes necessary to re-address a number of issues, not the least of which is what does secure computation mean in a world of mobile objects?

Consider the simple example of gathering information from a number of different information providers. In the mobile object paradigm we may decide to do this by two different approaches:

- Sending an object to each provider and communicating the results of this objects search back to the user (this could be done either serially with one object visiting each information provider site in turn or concurrently with each provider being sent it's own copy of a search object).
- Creating a search object that visits each information provider in turn, gathering information which it stores within itself and eventually disgorges when it returns home to the user.

Each of these approaches has advantages and disadvantages. Considering the second approach, we may imagine the object sent out by a user to tour a network of information providers. The user may trust the information providers to act as hosts for the information gatherer but possibly only to a limited extent. For example the information providers may covertly discriminate in the service that they offer depending on the ownership of the gatherer or on which other information providers have been visited. Information providers may attempt to read the information supplied by other providers and base their own contribution on what has already been supplied or they may attempt to alter the previously supplied information. Generally information providers can only be trusted to act within the

limits set by their own self interest, beyond this limit the gather must arrange security for itself.[1]

The anonymity conditions implied above are often difficult to achieve in practice and often conflict with other security requirements. The needs for confidentiality and authenticity however usually can be met by an appropriate cryptographic scheme. The challenge is to produce an approach to the security of mobile objects which is sufficiently generic to be useful in a number across applications rather than producing a tailored made solution for each application.

There are a number of objectives that we wish to meet with a mobile object security model.

1. Mobile objects should be able to carry secrets from place to place. These secrets may either represent credentials for obtaining services at various places or may represent confidential communications between specific places. Places should be able to convince themselves that mobile objects are acceptable with respect to the place's security requirements.

2. Mobile objects should be able to convince themselves that particular places are acceptable, with respect to the mobile object's security requirements, as hosts.

3. Mobile objects should be able to carry commitments between places.

4. Places should be able to convince themselves of the freshness of commitments and the uniqueness of the mobile objects carrying them.

All access control security requirements could be met by locating objects on hosts trusted to maintain the object's encapsulation because all objectives are trivially met. Objects on untrusted hosts would then interact with an object on the trusted host via an access control monitor implemented in the object on the trusted host. The access control monitor will permit access to the object's methods provided that its access control preconditions are met. These preconditions can involve any restrictions on method access based on local state (including invocation history, time of day, etc.), which host is performing the invocation and the parameters to the method invocation. Such a centralised solution however is not always desirable. Our main concern in this document is to understand under what circumstances it is possible to allow the protected state to exist on partially trusted hosts. Generally:-

- A host is called "partially trusted" when it is trusted for some actions but not others. Often the partial trust amounts to the belief that a host will follow a path of "enlightened self interest" in which

[1] Pedantic footnote: Providers and gathers act on behalf of principals. It is these that have self interest and ensure that adequate security measures are taken.

it will participate in actions that ultimately serve the interests of the host's owner.

- Mobile objects should be able to execute on partially trusted hosts in such a way that it is possible to guarantee that the object is correctly execute or that execution violations are detectable;
- Mobile objects should not require interactive protocols with their host of origin to produce these guarantees (if this is not possible for an application we should carefully examine whether or not mobile objects are a suitable technology for the application);
- Protection should, at least in principle, be provably correct.

The main concern of this report is with what kind of security can be provided for mobile objects when they move between partially trusted hosts. However before pursuing this topic the next two sections quickly note the main issues associated with secure communications between objects and what can be achieved using trusted hosts.

2 Communications Between Mobile Objects

2.1 Overview of Communications

Mobile objects may communicate with one another or with fixed services either when all parties reside on the same host or when the parties are split across a number of hosts. Generally a mobile object cannot keep the contents of it s communication secret from the host since we must assume a host is capable of looking inside the object and extracting the plain text of any communication from whatever internal buffers the object uses. This also means that any keys that a mobile object uses at a host to ensure secure communication are also available to a host and that any keys used by an object for authentication of itself to a service or any remote capability that it wishes to use at a host are revealed to the host (and may be used later by the host).

Objects can only have secure local communication if they trust the host. If two objects communicate remotely then the execution host of both objects in the communication must be trustworthy.

Remote services cannot trust the identity claims made by callers unless the identity claims include claims about the remote host identity and the service trust the behaviour of the calling host. This means that object identity claims are claims of the form 'I am object X at Place A'. If the service is denied on the grounds that A is untrusted, the service may choose to deny access from X in the future, because the untrusted place A could create a new object that can claim to be X to the service and the new object could contact the service from trusted host B. Therefor either:

- an object has to be sure of the identity of the host before it reveals its secret. Such identity proof can only be done before the object moves to the host from a suitably trustworthy host, or
- the object, or its originator, must ensure that the secret can only be revealed to specific trustworthy hosts.

In practice this simplifies our view of communications security in a mobile object system. All communication security is treated as primarily host to host. Host have authentication keys for one another and hosts perform any encryption and decryption required as a secure service offered to objects. Objects may provide encryption/decryption and authentication keys to hosts but these are only of value in contexts where there are reasons to

trust the host not to masquerade as objects at a later date (e.g. one time keys that cannot be re-used).

3 Trusted Hosts

Trusted hosts are secure hosts in that a mobile object trusts to maintain its encapsulation and not to disclose its secrets. When a mobile object is at a secure host the object may offer interfaces to objects at that place in the knowledge that these interfaces will be respected.

Trusted hosts may offer trusted services that provide guarantees of behaviour that may not be obtained at other places, for example they may offer a negotiation facility that acts as a trusted third party. Objects may negotiate via the negotiation service by placing “bond” with the negotiation service and the service offering guarantees to the other party based on the bonds. For example in a purchase model a purchaser wishes to confirm the ability of the seller to deliver if an order is placed but the seller does not want to reveal the actual quantity of goods he has in stack for fear of affecting the final price. Likewise the seller would like to know that the purchaser is not a time waster and has sufficient funds available before proceeding with negotiation, naturally the purchaser does not wish to reveal how much he has at his disposal. The purchaser proves the ability to pay by placing credit details with the third party negotiation service and the seller confirms the ability to meet the purchaser’s requirements by placing stock details with the negotiation services. These details are not released to the other party in the negotiation rather the negotiation service guarantees that the “bonds” have be placed with it and the negotiation may proceed on that basis.

Because trusted hosts respect interfaces it is possible to define the behaviour of an object at a secure place by an access control matrix that defines which principals may call which methods, the object can impose the required access control via a security manager that conceptually sits between the method proper and the caller. Objects carry secrets that allow them to sign messages on behalf of their principals supporting authentication for access control.

Movement between trusted hosts can be achieved securely using standard approaches to secure message transmission based on such technology as SSL [1].

Given a network of hosts in which some hosts are trusted if one has a technique which allows mobile objects to keep some part of its state secret from some hosts and expose it at others, then it is possible to use trusted hosts as secure ‘phone boxes. That is, a mobile object that wishes to participate in a secure transaction with a service can go to a trusted place

and expose its authentication and encryption keys and set up a secure communication with a remote service. Indeed, it is possible for the mobile object to use its own encryption algorithm under these circumstances but there seems little point since one still trusts the host to maintain encapsulation.

4 A Model of Security

4.1 Security in a World of Mobile Objects

4.1.1 Overview of Security Model

To understand the security requirements for mobile objects in which hosts are partially trusted we consider a simple abstraction of mobile objects in which an object is split into the object's code, an unprotected part of the objects state and a protected part of the objects. In The code part of the mobile object can be protected against modification by a host by signing, but there is no mechanism available that can guarantee that a host will execute the code part. The unprotected part of the mobile object models the fact that there may be some scratch variables or public communication channels between hosts that of no interest from the point of view of security. The protected part of state constitutes a collection of secure storage channels that can be used for communication between the mobile objects execution hosts. Conceptually the protected part can be thought of as living on a secure server that implements an access control monitor for the protected state. Hosts may read and write variables in the protected state and may sign variables or groups of variables in the protected state. The access control monitor can expresses access control preconditions in terms of the access requested, the host requesting the operation, state of the variables in the protected state and the history of invocations on the protected state.

As well as providing access control the secure server for the protected variables can ensure correct sequencing of host interactions and provide a defence against replay attacks.

From the point of view of security the code and the unprotected parts of state are irrelevant. It is not possible to guarantee that an untrusted host executes the code in the mobile part of the object or uses the unprotected state in the mobile part. An untrusted host may simply attempt to change the state of the protected part by using whatever privilege it has been granted.

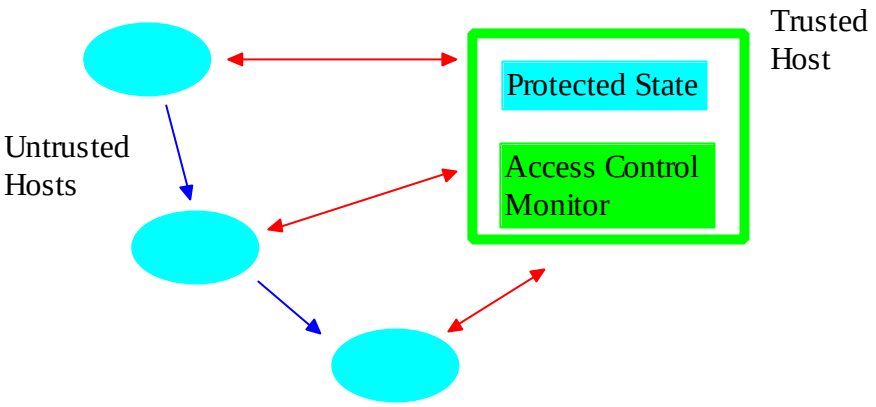

Figure 1: Split Object Abstraction

Generally the problem to be faced in distributing the protected state is that it is not possible to ensure that a partially trusted host obeys the access control preconditions if that host is given the protected state. However, in some cases, it possible to translate preconditions into a combination of **cryptographically enforceable preconditions** and **integrity post conditions** that can be checked after the state has been altered. This means that other partially trusted hosts will be in a position to check that hosts that have manipulated the state to date have done so in a way compatible with the security requirement. This means we can allow the object to move between partially trusted hosts and check whether their integrity has been maintained.

4.1.2 Cryptographically enforceable preconditions

Cryptographically enforceable preconditions on a host access to the protected state correspond to access control conditions which say that a host only requires read access to certain variable and write access to certain variables in order to execute the methods that it is permitted to use. That is, if a host has access to a method of the object it is only granted read and write access for those variables which are necessary for the execution of that method.

4.1.3 Integrity Post Conditions

To understand the possibilities for integrity post conditions we need to consider the nature of access control specifications. Access control specifications define a safe state S_0 and a set of transitions. A state is safe if and only if it is a state that is reached by a sequence of transition of the system starting at state S_0. An access control policy based on preconditions can ensure this inductive definition of state safety. To replace the precondition check by a post condition check there must be enough information in the state to guarantee the inductive condition i.e. enough information to guarantee that the state arose by a sequence of transitions

starting at S_0. In general this is not the case. There are two alternatives at this point. Firstly, for some applications, it is possible to define an absolute notion of the acceptability of a state, i.e. it is for practical purposes irrelevant how the state arose. Secondly the state could be redefined to contain an adequate trace of the transitions. For certain applications a combination of these two techniques can be used in which different parts of the state contain some guarantee about part of the path followed in their construction.

Naturally it is desirable whenever possible to be able to follow the first approach of finding an absolute notion of state acceptability that is independent of the interaction history of the object. Such a condition is an invariant of the state in that it is true before and after every method invocation of the object. Clearly, however, invarients that capture the required notion of security do not always exist. Under such circumstances we must adopt the second approach which can be viewed as finding an invariant over state and history together. This is more complicated in that the history must be securely recorded so that it can be used together with the state to test the invariant.

These techniques can be used on an applications specific basis. Although this is possible it is preferable to provide a generic infrastructure that can be used across a range applications.

The next two subsections discuss two generic mechanisms, cryptographically enforced access control and history based integrity post conditions.

4.2 Cryptographic Enforceable Access Control

Cryptographically enforceable access control is the name we give to a technique of achieving cryptographically enforceable preconditions and a generic absolute notion of acceptable state.

In order to distribute the protected state we start by making a simplifying assumption. The assumption is that read and write privileges to variables in the protected state is statically allocated. That is it is possible to represent the read and write access privileges via a fixed access control matrix. Each host is required to obey the access control matrix.

To ensure that hosts obey the access control matrix protected variables are treated as asymmetrically encrypted channels between hosts. Hosts with the read privilege in the access control matrix are given a read key for the variable and hosts with a write privilege are given a write key. Hosts with no privilege for a given variable are given no keys for that variable and hosts with both privileges are given both keys for the variable. Key distribution is performed using a public key infrastructure for hosts. When a host has a privilege for a variable it is given the key for that privilege

encrypted under the host's public encryption key. Key distribution is performed by distributing an access control matrix with the mobile object where the matrix entries are the encrypted keys for each privilege that is granted.

With just the above arrangement hosts with read privilege for a variable can check the validity of the binding to a variable. In general, however, this is not enough, any host should be able to check the validity of a binding to a variable. To achieve this each host with a write privilege is also given an asymmetric signing key and is required to sign its changes to the binding. The public signature check key for the variable is given to every host (i.e. distributed with the object in plain text form) so that each host can check that the variable is consistent with having been altered by a host with write privilege (although not being able to check which one or the contents of the variable).

The access control matrix is created by the "owner" of the object and signed by the owner.

The combined effect of the encryption control and signing is to produce a notion of state in which the encryption imposes preconditions on access and the signing defines a notion of acceptability of state.

4.3 History Based Integrity Post Conditions

4.3.1 Secure Histories

The problem to be addressed in any history based approach to defining the notion of acceptable state lays with the need to trust histories. The difficulties in creating trustable histories are illustrated considering a group of attacks on mobile objects which we generically refer to as sequencing attacks.

Firstly consider an itinerant object collecting data, then we may imagine the following unwinding attack on the object Figure 2:-

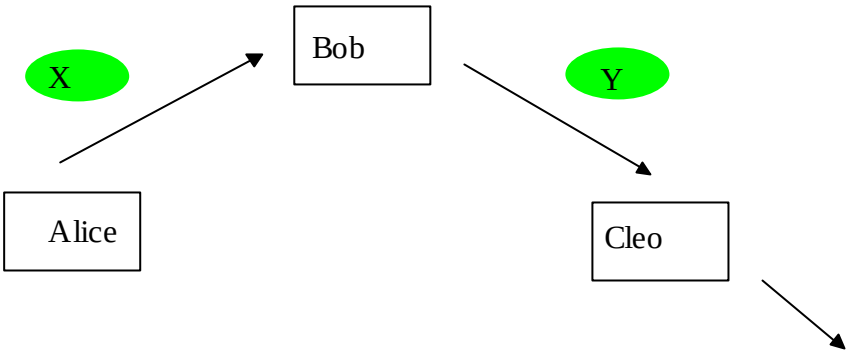

Figure 2: Unwinding Attack

The object moves from Alice to Bob to Cleo. Cleo inspects the objects state Y and decides that it prefers to remove Bob contribution, so Cleo unwinds the state of the object to X, the state prior to Bobs actions. Now it certainly depends on the nature of the state whether such unwinding is possible but it is quite easy to construct "shopping Agent" examples where such attacks are possible. The attack is countered by one of two approaches. Either it must be impossible for Cleo to unwind the state Y to X, or it must be known that Bob was an intended host for the object (which means Bob can use nested signing of his contribution so that it cannot be removed or altered).

Unfortunately making it impossible to unwind the state does not defend against hosts colluding to cheat other hosts. Figure 3 illustrates a form of cheating by cloning which we call the "piggy in the middle" attack. Alice passes an object in state X onto Bob but she also makes a copy of X that she passes to Cleo. Bob processes X to produce Y which is he passes to Cleo. Cleo now chooses which she prefers X or Y.

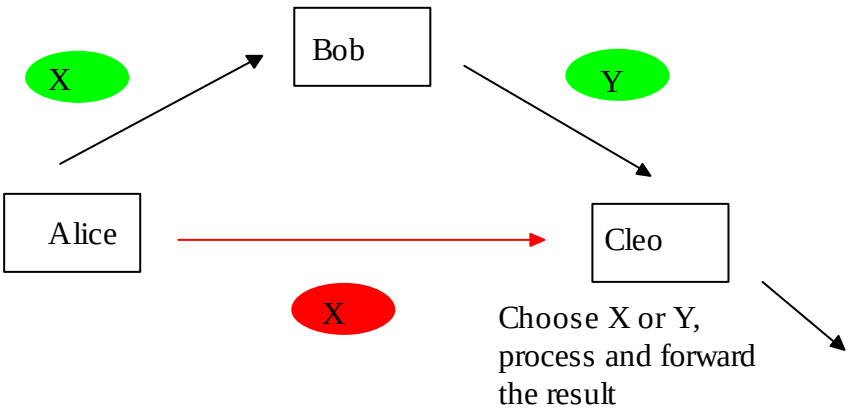

Figure 3: Piggy in the Middle

This attack can be defeated trivially if we know that a valid path must include Bob. However, if the valid paths include Alice, Bob, Cleo Alice, Cleo (no Bob) then nothing that can be done by Bob to the mobile object can defend Bob against this attack. The only solution is that information about the path actually followed by the object must be lodged elsewhere e.g. with an audit server using a secure communications path.

Generally it is necessary to defend against both the "unwinding attack" and the "piggy in the middle" attack. In effect this means that we must either ensure that the path is known in advance and has no dangerous choices or there is an audit server.

A final form of attack is the "best branch" attack. This is essentially the same as "piggy in the middle" where there are two "middle piggies" and two colluding hosts.

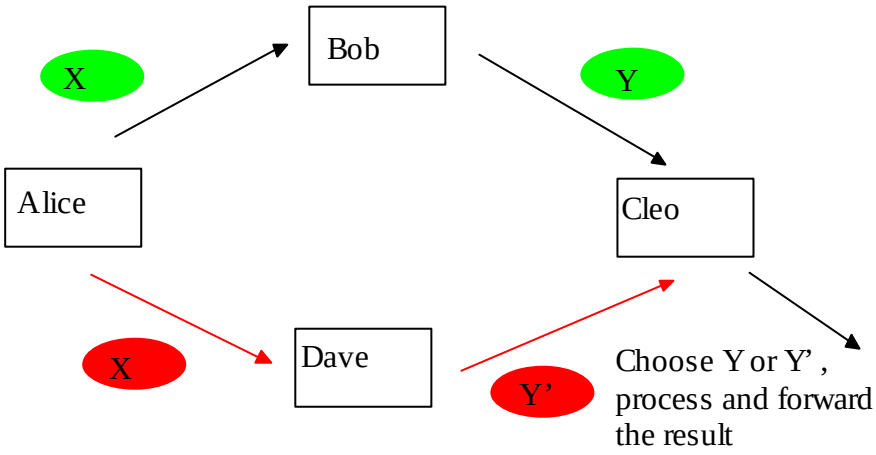

Figure 4 : Best Branch

In the "best branch" attack Alice colludes with Cleo to get the best out of a choice by cloning an object and sending clones to Bob and Dave. Cleo then chooses between the two results. Since the choice must be present this attack can only be avoided by use of a "sequence and audit server" for the choice point that fixes the choice made at Alice and records the fact. To simplify terminology we will simply call this an audit server.

In the next two sections we consider the specific solutions to these attacks. The first approach is the restriction of an objects itinerary to a **choice free path**. This restriction permits the implementation of a history as a nested signed data structure that is carried in the mobile object. The following section discusses the use of an audit server architecture to establish protected histories. Audit servers provide a means of protecting against

"piggy in the middle" and "best path" attacks for objects that make dynamic choices about their itinerary but at the cost of additional remote interaction.

4.3.2 Choice Free Paths

If a path is choice free the object can carry a signed path specification and a nested signed history i.e. each place adds its name to the history and signs the entire history. On receipt of an object a host checks the history against the path specification and rejects the object if they are incompatible. The history is protected from the unwinding attack because although a hostile host can unwind a history it cannot "rewind and sign" the history to make it compatible with the fixed itinerary of the object.

Potentially there is a replay attack when using this solution. A host may clone an object and replay it at later time down the same path. To defend against this attack hosts must keep a record of the objects which have visited them. The real issue here is to bound the memory requirement of the host. The simplest solution is for objects to have a fixed "use by" date allotted by their object factory and securely bound to the object. Hosts remember a secure digest of the object and its use by date for each object that visits. Objects that are expired are dropped by hosts on arrival and hosts forget expired object digests when the use by date has passed. Note that although there are problems with synchronising clocks in a distributed system they are of little concern to the current use of timestamps. Timestamps are used to bound the amount of information stored at a node. Clocks can be loosely synchronised and timestamps can allow generous use by periods without compromise of security.

4.3.3 Audit Server

The audit server model requires hosts to interact with an audit server when an object is about to move and when it arrives at a host. When an object departs a host the host informs the audit server of its destination, when it arrives at the new host the new host informs the audit server of its arrival. The two communications include the digest of the object to ensure that the same object leaves one host and arrives at the other. Both these communications are over cryptographically secure channels with host authentication.

Assuming that there are many audit servers in the system the particular audit server used for a mobile object is fixed when the object is created and bound to the object by the creators signature (but different objects, even from the same source, may use different audit servers).

The audit server keeps a secure log of where the object has been and acts as a sequencing server ensuring that there is only one copy of the object recognised as legal at any time. The concept was first discussed in [2].

4.4 Replay of Bindings to Variables

So far the discussion has been about whole objects. Mobile objects are also open to more piecemeal attacks in which individual protected variables are attacked. Although the measure discussed so far address the problem of ensuring that the appropriate hosts generates a signed value they do not address the scenario in which a binding signed by a host is plucked out of one object and substituted into another or, indeed, is taken out of an object after a visit to the host and is substituted into the object after another, later, visit of the object to the same host.

Such replay attacks on the values bound to variables represents a significant problem for mobile objects. The problem arises because of the essential one way nature of variables as channels between hosts. Unlike the situation in more usual cryptographic protocols host do not always have the opportunity to exchange nonces to stop foil attacks.

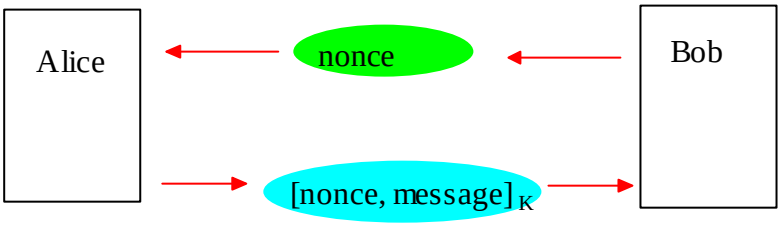

Figure 5 : Nonce Exchange

In a protocol where the principals, Alice and Bob, have two way interaction Bob can guarantee the freshness of a message from Alice using the following strategy : Bob sends Alice a random number, called a nonce, Alice creates the message she intends to send Bob and binds this together with the nonce using cryptographic signing or encode. Bob on receipt of the message checks that the nonce in the message is the nonce that he sent to Alice.

An alternative for the one way transmission of mobile objects is to provide a measure of protection by ensuring that mobile objects are created with a unique identity and that signing of variable bindings is based on that identity, the static content of the object and the point in the history where the signing took place. This means that each signing of a binding is tied to the object and to a point in the history of the object.

This interacts with the anonymity of writers signing variable updates. If writers sign anonymously then any writer can fake a claim that another writer updated the variable at an earlier point in the history.

4.5 Non-Anonymous Writers and Signing

Earlier it was indicated that the integrity requirement for a given host may only depend on the history of change of some of the variables in the protected part of an objects state. In this section we illustrate a framework within which non-anonymous host signing is used to pass promises or "commitments" between hosts. This provides a framework within which hosts may check perform more specific integrity checks on specific parts of the protected state thus providing a limited guarantee about how specific parts of the state where constructed.

If it is acceptable for writer not to be anonymous then writers can sign the variable updates with their own signature. Hosts receiving an object can extend the integrity check they perform to check that the last update to a variable is consistent with the objects path through the system.

More generally hosts wish to pass guarantees to one another to the effect that they are committed to the value of a variable or group of variables. Generally hosts wish to do this conditionally in that their commitments are dependent on the commitments made by other hosts to other variables. Such a pattern of commitment behaviour is an abstraction of the behaviour of the overall behaviour of an object. We can think of an object as having a commitment type which describes the consistency conditions on commitments made by hosts. An object is well behaved if the commitments made on the object state conform to its commitment type.

The commitment type of an object is the set of the acceptable commitment histories or traces of hosts. A commitment trace is a pair (P,C) consisting of a path specification P which says where the object has been and a commitment state C a set of dependant commitments. Given a commitment type T for an object, a host accepts the object if there is a pair (P,C) in T such that the object has the history P and has commitments C.

From the point of view of commitments to bindings the behaviour of an object can be described by a non-deterministic machine involving the basic actions:-

!!v	assign to v and sign the binding
!v	sign the binding leaving other signatures intact
p?x,... ,q?y→ !(!)v, ... ,!(!)u	conditional sign the bindings v,... ,u with a dependency on the condition part

A commitment behaviour for an object can then be described by the grammar:-

Behaviour := Host Name : Action (& Behaviour)*

Action = Name | Stop | Basic action . Action | Action + Action

where the action name means enable the host with name "Name" as the next host to visit, Stop means stop, Basic action "." Action means do basic action followed by Action and (Action + Action) means choose between the two actions.

The commitment behaviour of an object is a collection of actions labelled by the host they can occur at. For example the commitment behaviour of a somewhat simple payment scheme might be:

Purchaser : !!<order, amount>.Merchant &

Merchant: Purchaser? <order, amount> → !<order, amount> . Bank + Stop &

Bank : Stop

i.e. the purchaser writes and commits to an order and an amount in payment. The merchant either accepts this order for the payment and commits to it and forwards the object to the bank or he rejects the order and in this example simple stops. The bank on receipt of the object has no commitments to make and does whatever it does with the order.

At each step the commitment behaviour defines the acceptable commitment states. As seen by the merchant the object state variables order and amount are jointly committed to by the Purchaser, as seen at the bank there is a nested signing of these variables by the purchaser and the merchant.

The commitment type that can be built from the above form of description for each host is the complete commitment type. The projection of this at each host is the complete local commitment type for each host. In practice hosts often do not have to concern themselves with the complete local commitment type but can be content with a subtype i.e. a weaker set of assertions implied by the complete type at the host.

The complete local commit type for a host is a subset of the commitment type that contains just the traces with paths up to the current host. A subtype of a commitment type (or a local commitment type) is obtained by either taking a subset of the commitments in each trace or by taking a subsequence of the path. That is if T is a (local or not) commitment type then $T' \leq T$ (T' is a subtype of T) if and only if

$T' = \{ (P',C') \mid (P,C) \in T \text{ and } (P' \propto P \text{ and } C \subseteq C') \}$

where "$\propto$" is the subsequence relation.

A host integrity check is a check that an object conforms to a subtype of the complete local commitment type.

4.6 Information Flow

A natural question to ask is a given cryptographic access control matrix adequate? Adequacy means that only the intended information is revealed to a host and that only the intended control over variables is given to a host. Clearly state variables are linked by the code method code of an object. A host may be able to deduce the value of variables that it cannot read from values of variables that it can read using the relationships defined by the code. Likewise a host may be able to gain some control over variables it cannot write by careful use of variables that it can write using the knowledge of the relationships in the code and the likely behaviour at other hosts.

Generally the access control matrix should be consistent with the local information flow seen at a host. This means that if a host can deduce from the program that an information flow exists between two variables, x and y, from x to y, at an earlier host on the objects path and that information flow is undisturbed by the action of intervening hosts, then either both variables should be readable at the host or neither variable should be readable at the host. A stronger variant of this condition is obtained by ignoring the path and simply requiring that if there is an information flow between two variables at a host then either they are both readable at the host or both unreadable at the host. A similar condition holds for writable variables. The simplest solution is to adopt a no information flow policy based on Denning[3] or its later type based variants [4].

Sometimes, however, such information flow conditions are too strong and require relaxation. The simplest case of this is when there is an information flow between two variables but there is a significant loss of information in the flow. Generally we need to say that an information flow is acceptable when there is no useful information that can be extracted from the flow. If there is an information flow x to y where x is of type X and y is of type Y, we define the information obtained from y about x as a guessing function, *f*, obtained by analysing the program, from the value of y to the set of possible values of x. A policy, P, for x is a partition of X into values indistinguishable by looking at y. A program meets the policy for x if the range of the guessing function, *f*, is a coarser partition of X than the policy P. Note we choose the requirement that range *f* be a coarser partition than P rather than just a cover of P for composibilty. Given a policy P, with information flows to variables x and y, with guessing functions *f* and *g* respectively then if the condition were that range f and range g simply covers P then combining the predictions could lead to more information than obtainable from *f* or *g* and, indeed, could violate P.

4.7 A Practicality of Peripatetic Objects

When objects dynamically choose which host to visit next, as may happen with a simple information gathering object the simple view of variables associated with hosts becomes cumbersome. It is often more convenient to

consider hosts writing information into a shared data structure. Such a structure may be thought of as a variable with a different access property available, that of extension. If a variable has the extension property it implements a stack[2]. A host with extension access is given the right to push a new value onto the stack. This requires that it possess the write key for the variable and a signing key. The stack is sealed by performing nested signing. That is, the added values, the previous values and the previous signatures are all signed before the object departs the host.

The structure of the stack (in a SML like definition) is:-

Stack X= signature of (Signature * Stack) | push of (X * Stack) | Empty

Signing, at host h with key k, is performed by:-

sign(S) = signature(Sign(h, k, S),S)

Since movement choices are being made dynamically a choice and audit server is involved in the moves. The integrity check of a stack variable is performed by checking the stack signing against the actual path obtained from the audit server.

[2] Actually it needn't be a stack. It just needs to be an appropriately nestable structure, such as labelled trees.

5 Life Cycle

The life cycle model for mobile objects is that mobile objects are created by a factory that creates objects of particular types. Objects are created at the request of a principal that becomes the owner of the object. When the object is created the ownership information and the object id are built into the object by the factory. The factory is a trusted entity that acts on behalf of the owner and on behalf of the potential hosts of the object. Each object has a commitment type that defines the objects possible paths through the network and the commitments that are expected to its protected state at each host. Conceptually an object's commitment type represents the structure of contracts that are possible between hosts on its potential path. The object also has a purpose which indicates the purpose of the object e.g. payment protocol object. Finally the factory fixes a timeliness policy.

Object: object id, owner, factory, commitment type, kind, timeliness policy, code, factory signature

The owner of the object may provide additional data to the object. Firstly it provides a path restriction so that rather than being able to follow all paths it may only follow some paths. Secondly if the paths involve choices the owner fixes an audit/choice server for the object and finally the owner timestamps the object so that its ready for use.

Object : path restriction, audit/choice point server, timestamp

The object follows its path through the network moving from host to host. On arrival at a host the host checks the:-

- factory signature on the object id, owner, commitment type, kind, timeliness policy and code;
- owner signature on the path restriction and audit/choice server and timestamp;
- the commitments made to date and how they correspond to the history.

The host then

- obtains any keys sent to it via the access control matrix;
- executes the appropriate methods;
- performs the final signing of the variables its altered;

- updates the history before the object moves.

6 Examples

6.1 A Little Language

In order to discuss the examples succinctly we introduce a little language. Within our framework presented above a mobile object can be viewed as a finite state machine. From the point of view of the security model a mobile objects state consists of where it is on it itinerary and the state of signing of its protected variables. We define a mobile objects security behaviour as an annotated non-deterministic finite state machine. The "states" of finite state machine correspond to being at particular hosts at particular points in the itinerary and the annotations correspond to the conditional commitments being made to variables. We will refer to the annotated finite state machine as an itinerary specification. As well as an itinerary specification a mobile object also has a cryptographic access control matrix (CACM) specifying which hosts can read and write which variables (or stack variables). When an object arrives at a host the state of the protected variables are checked against the CACM and the history is checked against the itinerary specification. If these are valid then the action associated with the particular visit to the host are performed. If it is not possible to perform the actions required t a host then an error is raised and the object discarded.

```
IS ::= Host (+ Host)* [[[Actions]]] | Name | Skip |
       IS + IS | IS ; IS |
       let Name = IS (and Name = IS)* in IS | (IS)

Actions ::= Check | Update |
            Check(,Check)* → Update (, Update)* |
            Action ; Action  | Action + Action|(Action)|
            Skip

Check ::= Host ? Var | Host ? <Var,…,Var>

Update ::= ! Var | !! Var | ^Var | <Update, … , Update>
```

A basic itinerary specification (IS) is a host name followed by an action (e.g. host[action]). Itinerary specifications may be combined by:-

- Skip - the itinerary specification that does nothing

- choice (IS_1 + IS_2), in which case exactly one itinerary specification is followed;
- sequencing (IS_1 ; IS_2) , in which case the itinerary specifications are followed one after the other.

A local definition may be introduced by the "let" construct and the local definition may used as a itinerary specification.

The form $host_1$ + … + $host_n$ [action] is an abbreviation for $host_1$[action] + … + $host_n$[action]. The itinerary specification "host" is an abbreviation for host[Skip].

Actions correspond to checking signatures, updating signatures and conditionally updating signatures as discussed earlier, that is:-

- Checks
  - host ? Var - test if host signed Var
  - host ? <Var_1,… , Var_n> - test if host jointly signed Var_1,… , Var_n
- Updates
  - !! Var - assign a value to Var and sign it
  - !Var - sign Var
  - ^Var - treat Var as a stack variable pushing a value onto Var and perform a nested signing
  - < $Update_1$,… , $Update_n$> - update and make mutually dependent
- Check*(*,Check*)** → Update (,Updates)* conditional signing i.e. Updates depend on variables in checks
- Action + Action - choice between actions. The host may perform any of the actions which succeed
- Action ; Action - sequencing of actions
- Skip - do nothing

If the action at a host fails then a failure is signalled and the object is dropped.

In effect the CACM acts a set of variable declarations for a distributed program which is defined by the itinerary.

6.2 Payment Protocol Objects

The payment protocol extends the earlier payment example by including a purchaser (P), a merchant (M), the purchaser's bank (PB) and the merchant's bank (MB).

The basic interaction is that the purchaser requests a payment object from an object factory. The object factory will provide a payment object suitable

for the purchasers bank and the merchant's bank. The purchaser then raises an order with the merchant for some specific materials (order) at a price (amount). The purchaser includes his account details with the order but does not disclose them to the merchant. The merchant reads the order and amount and creates an order number and forwards all this to the purchaser's bank (PB). The purchaser bank arranges a transfer of funds from the purchaser's account to the merchant's account at the merchant's bank. The merchants bank informs the merchant of the transfer. The possible paths of the object are

```
P[<!!order,!!amount,!!PurchaserAccount>];
M[P?<order,amount,PurchaserAccount> →
   <!order,!amount,!!OrderNumber,!!MerchantsAccount>
];
PB[M?<order,amount,ordernumber,MerchantsAccount>,
   P?<order,amount,PurchaserAccount> →
      <!amount,!OrderNumber >];
MB[M?<amount,OrderNumber,MerchantsAccount>,
      PB?<amount, OrderNumber> →
         <!amount, !OrderNumber>];
M[MB?<amount,OrderNumber>]
```

The CACM for the payment object is:

	order	**amount**	**order number**	**Purchaser account**	**Merchant account**
P	W	W		W	
M	R	R	R/W		W
PB		R	R	R	
MB		R	R		R

6.3 Information Gatherer

The information gatherer object is an peripatetic object that makes dynamic decisions about where to go next. The gatherer may be subject to sequencing attacks outlined above and needs to use an audit (and sequencing) server to defend against them.

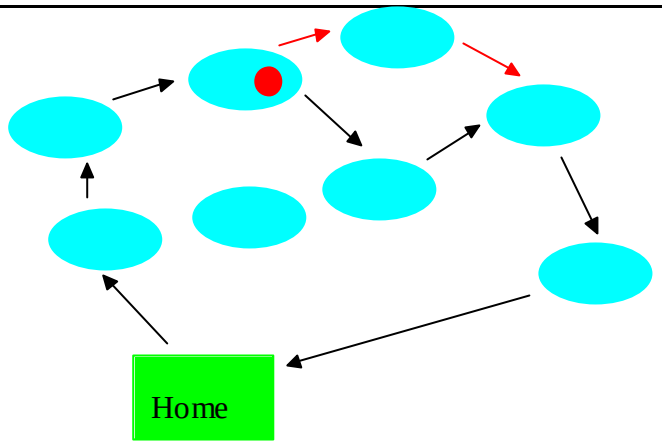

Figure 6 : Information Gatherer

The gatherer object starts from a home host and moves around a network of information providers according to some itinerary that involves choices. At each host it performs some interaction with the host and pushes the resultant data onto a stack variable.

The behaviour of a gatherer is then:-

```
let itinerary = (P1 + … + Pn[^stack]);itinerary + Skip
in home; itinerary ; home
```

The integrity check of the stack variable requires that the signing is compatible with its (dynamic) path history. The CACM for the gather is:

	Home	**P_i**
Stack	R/W	W

This has the effect that the provider processes cannot read each others contribution to the stack.

6.4 Lottery

In the lottery a client obtains a ticket from a ticket seller (TS), the client fills in his ticket and then sends it to the ticket registrary. The registrary records the ticket and then stamps the ticket to form a reciept and sends in back to the client. The security requirements are that the client is known to the ticket seller (e.g. in order to extract payment) but anonymous to the registrary (e.g. in order to stop bias). The registrary needs to know that the ticket has been supplied by the ticket seller and that it has not be altered by anyone other than the client. The client needs to know that the

reciept really did come from the ticket registrary. If there is a fraud perpetrated by the client by multiple use of a ticket, then the ticket registrary should be able to detect the fraud and the ticket registrary co-operating with the ticket seller should be able to unmask the perpetrator.

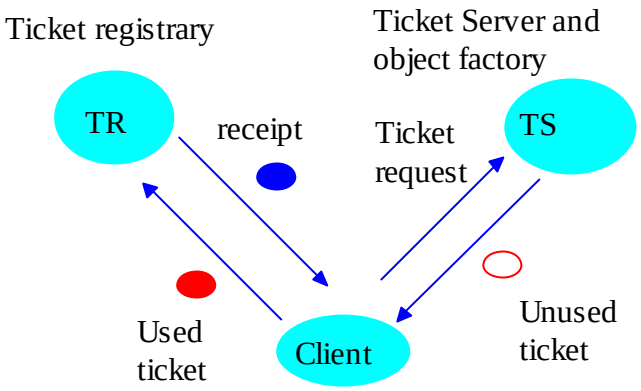

Figure 7 : Lottery

The objects behaviour is:-

TS . Client . TR[<!TN, !Bet>] . Client[TR?<TN,Bet>]

The cryptographic access control specifies which entities can read and write the variables TN (ticket number) and Bet. These variables are anonymously signed on writing and the signature is checked on arrival at a new host. The cryptographic access control matrix is:-

The anonymity achieved by this solution is questionable. The issue hinges

	TN	**Bet**
TS	W	
Client	R	W
TR	R	R

on the question of who is the client. The client host Client is known to the registrary because it is in the path description. However Client does not sign the object using its public key. If the hosts public key arises from the user of the Client then the users identity is never revealed. So the security requirements above would be met by an implementation in which the Client is a trustworthy host machine that may be used by several users. As each user uses the host the host takes on personality of the user (i.e. takes on the users digital identity). The host must be trusted to protect users from Trojan Hoarse attacks.

7 Conclusions

7.1 Communications

The possibility of secure communications between mobile objects has been shown to depend on placing trust in the execution host of the mobile objects communicating. When objects communicate it must be assumed that the hosts has access to both the plain text and the keys used by the host.

7.2 Trusted Hosts

For some applications it may be desirable to have a backbone of trusted hosts available within a partially trusted network. Trusted hosts can provide neutral ground from which a mobile object may provide some services or perform some acts, such as remote communication, in the knowledge that the integrity and confidentiality of mobile objects state, code and interfaces will be maintained.

7.3 Partial Trust

The key component of this report is the introduction of the notion of a partially trusted host, which is trusted for some secrets but not others. An analysis of the basis of the partial trust security model has been performed and specific mechanisms have been introduced which provide a means of realising partial trust in practice.

7.4 Future

This report is a collection of ideas about what is fundamental to mobile object security. It is clear at the heart of the report is there are two notions. Firstly that mobile objects act like collections of channels between hosts and that security can been seen in terms of an access control matrix controlling the use of these channels. Secondly that mobile objects must be able to act as carriers of promises or commitments to the values communicated over these channels. The hosts can be seen as authorised

and unauthorised users of these channels. So far we have only sketched the analysis and consequences of these ideas informally. One might hope that in a more formal setting the exact correspondence between what may, and may not, be achieved by mobile objects as channels versus simple communication channels between remotely located objects may be obtained.

Finally the suggestive notation use din the examples section above might be elaborated to provide a high level policy description language both policy writing and policy implementation (via a Java translation).

8 References

1. SSL 3.0 Specification, Netscape Inc, March 1996 1983 (published as an Internet Draft).

2. **A. Herbert and R. Needham**, A Registration and Sequencing Server, Proc 8th SIGOPS Symposium on Operating Systems Principles, ACM, Dec 1981.

3. **D. Denning**, Cryptography and Data Security Addison-Wesley 1983.

4. **D. Volpano, G. Smith and C. Irvine**, A Sound Type System for Secure Information Flow, Journal of Computer Security, 4(3):1-21 1996**.**