Open Network Handles—Abstract Protocol Specification *(DRAFT 0.1)*

Michael J. O'Donnell

22 August, 2002

Abstract

I propose a system of *Open Network Handles* to provide permanent primitive network handles promiscuously to all who request them. Handles provide an intermediate level of service between IP numbers and domain names. While assignment of IP numbers is constrained by routing considerations, the owner of a handle may reassign it to different addresses over time for mobility or changes in configuration of resources. Unlike domain names, handles carry no significance in natural language, so they should not have high commercial value, nor should they attract disputes based on assertions of rights in significant names.

This document describes a protocol for an open network handle system at the abstract level—what information needs to be in each type of message—but no syntax or other implementation detail.

0.1 STUB

1 Purpose of the Protocol

I propose a system of open network handles, similar to domain names, but deliberately free of humanly meaningful semantics. A handle is intended to provide permanent global consistent reference to a particular agent, and nothing else.

This article describes a protocol for an Open Network Handle System (ONHS) at an abstract level. It only describes the messages sent and received by users of the system, not the internal messages, data structures, and algorithms needed to implement such a system. It describes messages abstractly in terms of the information that they carry. All questions of format, syntax, and other implementation details are left to a later work.

Most if not all of the difficult problems that must be solved to implement ONHS have already been addressed in the implementation of the Domain Name System, so there is little uncertainty regarding the feasibility of the system. In the short run, the service provided by ONHS is a subset of the service provided by DNS. By cutting back service to a logical minimum, ONHS hopes to provide stronger confidence in the permanence of handle resolution than DNS provides for domain name resolution, mainly because by avoiding humanly meaningful semantics, we can avoid the administrative disputes surrounding name resolution.

I have discussed elsewhere the nature of handles, and their relations to routes, addresses, and names [**?**]. For present purposes, it suffices that an agent participating in the global Internet may own a handle, and may bind and rebind it at will to addresses at which the agent may be reached. Any agent may query ONHS to discover the address currently bound to a given handle.

1.1 ONHS Responsibilities

ONHS deals with communications between *handle owners*, *queriers*, and *handle servers* who constitute a highly distributed *global system*. Many or even most handle owners and queriers will also participate in the global system as handle servers, but it is convenient to separate their actions according to the roles they serve.

ONHS is only intended to provide a sort of continuity in communication with an agent whose address changes from time to time. It is *not* intended to provide any sort of assurance regarding the identity nor the good behavior of that agent, including the agent's good faith and competence in managing his handle.

ONHS provides mechanisms for an agent to acquire a handle, and to announce bindings of that handle to different addresses over time. When another agent queries the system, it tries to return a hint, giving the address most recently bound to that handle by the owner.

The global system focuses on putting each querier in contact with the correct handle owner. As much as possible, authentication of the handle owner, and all other assurance of quality in communication, are left as the responsibilities of the querier and handle owner. The ONHS global system takes responsibility for responding to each query with the correct resolvent address promptly with high probability. The precise promptness of response and highness of probability are determined by the needs and capabilities of the global network at any time. The global system is only responsible for overall productivity—owners and queriers are responsible for the reliability of individual transactions. Responses to queries are called *hints*, because the global system does not try to make them individually highly reliable.

Although the global system does not take direct responsibility for the correctness of individual hints, to be useful it must operate in a manner that allows owners and queriers to authenticate their communications effectively. To this end, the global system also takes responsibility for maintaining a low rate of false hints. In a distributed system, we usually cannot enforce a one-to-one correspondence between queries and responses, so the low rate of false hints is a separate goal from the high probability of prompt correct response.

With current public-key authentication techniques, and presumably with other authentication techniques invented in the future, individual agents have the tools to insure the correctness of their communications, as long as their authentication techniques remain secure. But improvements in computing speed and mathematical advances tend naturally to break the security of any particular technique, so as a secondary goal, the global system should help handle owners through transitions in authentication techniques. This seems to be best done by allowing a combination of self-authentication and third-party authentication, along with support for irrevocable announcements by handle owners transferring the meaning of an old handle to a new one. Such transfer can be used to reassign a handle to a new agent, as well as to a new authentication technique.

Since no mathematical technique can make authentication foolproof, the global system should also provide handle owners and queriers with hooks by which they may audit security.

The *global system* operates in a highly distributed fashion on a loose coalition of more and less trusted hosts. A good system should allow individual queriers and handle owners, as well as coalitions, to contract with vendors who participate in the global system on their behalf to improve performance

and reliability for particular handles in which they take a special interest. A good system also allows essentially unlimited subspaces of the handle space to be administered locally by any authority who cares to do so.

2 Data Types in the ONHS Protocol

- An *authentication type* (<*autype*>) is a token assigned by general agreement (under current practice, probably by IANA) to represent a particular authentication technique.

- An *authentication specification* (<*authentication*>) is a sequence of the form

  $$[<\textit{autype}>, <\textit{datum-1}>, \ldots, <\textit{datum-n}>]$$

  where <*datum-1*>, . . . , <*datum-n*> are parameters appropriate to the authentication technique associated with <*autype*>. For example, if <*autype*> specifies a certain public-key technique, then we might have $n = 1$ and <*datum-1*> a public key for a certifier. Or, if <*autype*> specifies a certain public-key technique with a particular trusted certifier known by general agreement, then we might have $n = 0$, leaving the key to be determined externally.

- A *handle item* (<*hitem*>) is a pair of the form

  $$[<\textit{authentication}>, <\textit{datum}>]$$

  where <*datum*> is a (possibly empty) subsidiary datum added to <*authentication*> to make the handle unique. For self-certified handles based on public-key authentication, the public key contained in <*authentication*> suffices for uniqueness. But if a single trusted agent with a single public key wants to certify a set of handles, then <*datum*> must be sufficient to distinguish an individual handle in that set.

  <*authentication*> describes the authentication used for announcements of handle bindings. Queriers and handle owners are at liberty to use the method specified by <*authentication*> for end-to-end authentication, or to agree on another method.

- A *handle* (<*handle*>) is a finite nonempty sequence

$$[<\textit{hitem-n}>, \ldots, <\textit{hitem-0}>]$$

of handle items. To make the correspondence with hierarchical domain names easier to read off, I write such a sequence from right to left. Handles as sequences form a natural hierarchy, with essentially the same structure as the hierarchy of domain names.

When $m \leq n$ and

$$<\textit{handle-1}> = [<\textit{hitem-m}>, \ldots, <\textit{hitem-0}>]$$

is a suffix of

$$<\textit{handle-2}> = [<\textit{hitem-n}>, \ldots, <\textit{hitem-0}>]$$

then <*handle-1*> is an *ancestor* of <*handle-2*>, and conversely <*handle-2*> is a *descendant* of <*handle-1*>. When $m < n$, so <*handle-1*> $\neq$ <*handle-2*>, they are *proper* ancestors and descendants, respectively.

If $m < n$, $p = m + 1$, and

$$<\textit{handle}> = [<\textit{hitem-m}>, \ldots, <\textit{hitem-0}>]$$

is a handle, and

$$<\textit{prefix}> = [<\textit{hitem-n}>, \ldots, <\textit{hitem-p}>]$$

is a handle prefix, then the concatenation of <*prefix*> with <*handle*> is

$$<\textit{prefix}><\textit{handle}> = [<\textit{hitem-n}>, \ldots, <\textit{hitem-p}>, <\textit{hitem-m}>, \ldots, <\textit{hitem-0}>]$$

Notice that <*prefix*><*handle*> is a descendant of <*handle*>.

The hierarchy of handles in ONHS is intended to be purely a hierarchy of authority over handle bindings, not a hierarchy of meaningful descriptions of the uses of the handles. An outermost handle comprising a single item is assigned according to generally agreed rules, but those rules should allow the most promiscuous possible assignment of handles. The owner of each handle determines the rules for ownership of all descendant handles.

- An *address type* (<*adtype*>) is a token assigned by general agreement to represent a particular type of network address.

- An *address* (<*address*>) is a pair of the form

  $$[<adtype>, <addrdatum>]$$

  where <*addrdatum*> is an address of the given type.

  In general, all that we require of an address is that it allows delivery to a consistently determined agent no matter where in the network it is invoked.

- A *Sequencer* (<*sequencer*>) is a value from some ordered domain with a minimum element $-\infty$ and a maximum element $+\infty$.

  The meaning of a sequencer is local to a particular type of value associated with a particular handle. Sequencers may be set from a clock, or an incremented register, or by any method the handle owner chooses. They are used only to compare the recentness of conflicting announcements about a single handle.

  The minimum sequencer $-\infty$ is used in queries, but not in announcements. The maximum sequencer $+\infty$ makes an announcement irrevocable.

- A *time* (<*time*>) is a value representing absolute time, according to some generally accepted system of time notation.

- A *performance specification* (<*performance*>) is a set of parameters affecting the performance of the system. Performance specifications may be used to suggest the use of such parameters, and also to report that certain parameters were used in an attempt to perform some service.

The exact scopes of authentication types, address types, time stamps, and performance parameters is open, both for the initial implementation of ONHS, and for future improvements of the system. Here are some particularly important members to include from the beginning, and some discussion of possible future developments.

- Authentication types should include at least the following.

  - *Unauthenticated* (probably not supported at the outermost level of the handle hierarchy).
  - At least one *public-key* method, with the public key provided as a parameter.
  - At least one *hashed public-key* method, with the hash code provided as a parameter.
  - At least one *password* method, with the handle of the password manager provided as a parameter.
  - *Inherited* authentication (not supported at the outermost level)—authentication taken from the nearest ancestor with a type other than inherited.
  - It is crucial for the continuing utility of the ONHS that stronger authentication techniques are added as they become available, and long before the currently included techniques become insecure.

- Address types should include at least the following.
  - *IP addresses*

  As soon as possible, if not initially, we should probably add address types like the following.

  - *UDP addresses*—pairs of the form $\langle A, P \rangle$, where A is an IP address and P is a port number.
  - *email addresses*
  - *URL*s

  In general, addresses should be able to refer to any reasonable sorts of *virtual agents*, not just to entire hosts. The sorts of extended addresses described above refer to particular applications, and even behavioral domains within applications, on a given host. In general, it makes sense to allow addresses that specify a host, an application running on that host, and some parameters to the application.

  Virtual agents may also be distributed among multiple hosts, so as soon as possible, if not initially, we should probably add address types like the following.

  - *Hunt lists* of addresses to be tried until receiving a satisfactory response.
  - *Location-dependent addresses* that direct communications along more efficient routes. Akamai already simulates such addresses within the DNS.
  - *Multicast addresses* that direct communications simultaneously to multiple subagents, sometimes with different data directed to different subagents.

  The ability to rebind handles deals with relatively slow, unpredictable mobility of agents. But for fine-grained and/or predictable mobility, as well as for intermittently connected agents, and predictable or measurable rapid variations in network traffic, we might choose to add the following.

  - *Time-dependent addresses.*

  Once we start thinking of extensions to the list of address types, it's hard to stop. To control such extensions in the future, we should apply two principles: (1) add an address type only when it supports an interesting new concept of virtual agent; (2) add an address type only when it allows efficient techniques that could not be implemented end-to-end between the querier and owner. For example, hunt lists look like a good type of extension because: (1) they allow multiple redundant agents to compose a single virtual agent; (2) the querier needs the hunt list precisely when he cannot reach the agent at the first address. Of course, distributed virtual agents may be queriers as well as handle owners, and I haven't yet thought about the implications of that generalization.

- Performance parameters should include at least the following.

  - *Time to live* (TTL)

  TTL is taken directly from the current DNS. It is *not* the same as a time-dependent address. It gives a time-out for a given announcement, rather than a time-out for the address itself.

3 Essential Messages, Useful Messages, and Policy

In the following sections, I describe the different sorts of messages that may be sent by queriers, the global system, and handle owners. I do not describe messages sent between participants in the global system—I leave those to the implementation. To allow the greatest possible flexibility in distributed implementation, I distinguish messages that are essential to system utility, and messages that may improve system performance and/or utility but are not essential. I describe messages in a self-contained form, so that each message has a clear meaning entirely by itself without any conversational context. I also describe messages containing the smallest increments of information that seem to make sense. A particular implementation may allow some of the information to be determined by conversational context, and/or it may bundle messages for efficiency.

4 Announcements from Handle Owners

4.1 Essential Messages

- A *binding announcement* is a message of the form

  AnnounceBinding(<*handle*>, <*address*>, <*sequencer*>, <*certificate*>, <*performance*>)

  where <*certificate*> is a certificate authenticating the announcement by the technique appropriate to <*handle*>, and $-\infty <$ <*sequencer*>. Such a message indicates that the handle owner binds <*handle*> to <*address*>, taking precedence over all announcements with sequencers less than <*sequencer*>.

- A *delegation announcement* is a message of the form

  AnnounceDelegation(<*handle-1*>, <*handle-2*>, <*sequencer*>, <*certificate*>, <*performanc*

  where <*certificate*> is a certificate authenticating the announcement by the technique appropriate to <*handle-1*>, and $-\infty <$ <*sequencer*> $< +\infty$. Such a message indicates that the handle owner delegates binding authority over <*handle-1*> to <*handle-2*>, taking precedence over all announcements with sequencers less than <*sequencer*>.

- A *transfer announcement* is a message of the form

  AnnounceTransfer(<*handle-1*>, <*handle-2*>, <*certificate*>, <*time*>)

  where <*certificate*> is a certificate authenticating the announcement by the technique appropriate to <*handle-1*>. Such a message indicates that the handle owner transfers <*handle-1*> permanently and irrevocably to <*handle-2*>, advising all future queriers of <*handle-1*> to use <*handle-2*> instead. A transfer is essentially an irrevocable delegation—the sequencer is implicitly $+\infty$.

- A *compromise announcement* is a message of the form

  AnnounceCompromise(<*handle*>, <*certificate*>, <*time*>)

  where <*certificate*> is a certificate authenticating the announcement by the technique appropriate to <*handle*>. Such a message indicates that the handle owner believes that the security of <*handle-1*> was compromised at the given <*time*>. A compromise announcement is irrevocable—it implicitly carries the sequencer value $+\infty$.

4.2 Useful Messages

- A *cancel announcement* is a message of the form

  AnnounceCancel(<*handle*>, <*certificate*>, <*time*>)

  Such a message indicates that the handle owner repudiates the use of the handle from the given <*time*> on. Cancellation is irrevocable—it implicitly carries the sequencer value $+\infty$.

- Each announcement is an announcement *for* the handle in its first argument position.

4.3 Policy

A handle owner's policy depends on the importance of her handle.

To avoid the risk of differential delays reordering announcements, a handle owner should use increasing <*sequencer*> values in announcements. Without increasing <*sequencer*> values in her announcements, a handle owner is vulnerable to resubmission of a saved announcement by an adversary.

A handle owner should verify the effect of an important announcement with a test query.

A single <*handle*> value only lasts as long as its authentication technique remains secure. We hope that new and stronger authentication techniques will be invented months or years before old techniques become insecure, providing a substantial upgrade window. A handle owner who needs handle service beyond the lifetime of a particular authentication technique should create a new handle with a stronger new technique as soon as it becomes available, and announce an irrevocable transfer from the old handle to the new one. All queriers who use the old handle before it becomes insecure should redirect to the new one. Depending on the importance of the handle, the owner may provide authentication of identity out of band, and gather additional certificates of transfer from trusted third parties. A trusted time stamp, showing that the transfer was made before the old handle became insecure, is particularly valuable. The additional certificates are not included in the ONHS protocols: the handle owner may provide them directly to the querier.

A handle owner should not depend on ONHS to establish the authenticity of individual contacts made through the handle system. The handle owner has no defense against the credulity of a misdirected querier who consents to transactions without authentication. But the handle owner may present authentication policies to her correspondents as early as possible in the correspondence, to reduce the likelihood of loss of a correspondent through misdirection.

As soon as a handle owner learns that an adversary has gained the power to issue authentic-looking announcements, she may limit the damage by announcing a compromise, and using trusted third parties out of band to try to recapture lost correspondents. The existence of an unauthorized binding with a correct certificate often indicates that the handle authentication technique is compromised, but it may indicate another sort of compromise of the handle owner's computing and communication resources. If cancellation of handles is supported, the owner should cancel a compromised handle. By announcing compromise and cancellation, the owner limits the consequences of unauthorized bindings and delegations and directs attention to trusted third parties out of band.

Mention policy for transfers, including recipients of transfers.

5 Using Announcements to Resolve Handles

We are only interested in handles that appear to be in use.

- If a handle has been mentioned as the source in an authentic transfer announcement, then it is *in use*.
- If a handle has been mentioned as the target in an authentic transfer announcement, or as the source or target in a delegation announcement, or as the source in a binding announcement, but we are not aware of a cancel announcement for it, then it is *in use*.

The set of authentic announcements known to name servers determine partial resolutions of handles to other handles, and full resolutions of handles to addresses.

- Each handle *partially resolves* to itself.
- If <*handle-1*> partially resolves to <*handle-2*>, and there has been an authentic announcement of the form

  AnnounceDelegation(<*handle-2*>, <*handle-3*>, . . .)

  or

  AnnounceTransfer(<*handle-2*>, <*handle-3*>, . . .)

  then <*handle-1*> *partially resolves* to <*address*>.
- If <*handle-1*> partially resolves to <*handle-2*>, and if their corresponding descendants <*prefix*><*handle-1*> and <*prefix*><*handle-2*> are both in use, then <*prefix*><*handle-1*> partially resolves to <*prefix*><*handle-2*>.
- If <*handle-1*> partially resolves to <*handle-2*>, and there has been an authentic announcement of the form

  AnnounceBinding(<*handle-2*>, <*address*>, . . .)

  then <*handle-1*> *fully resolves* to <*address*>.

- A *resolution* of <*handle*> to <*address or handle*> is a sequence of announcements supporting the fact that <*handle*> partially or fully resolves to <*address or handle*> in the definition above.

After a long sequence of announcements, there are likely to be many ways to fully resolve a given handle, but all but one of them are usually obsolete.

- A binding/delegation announcement for <*handle*> is *obsolete* if there is another binding/delegation announcement for <*handle*> with a larger sequencer, or if there is a transfer or cancel announcement for <*handle*>.

- A resolution of <*handle*> to <*address or handle*> is *obsolete* if one or more of its announcements is obsolete.

The ONHS attempts to resolve handles by nonobsolete resolutions, but in a distributed system, there is no certain guard against obsolescence. The ONHS should provide resolutions that, with high probability, are not very obsolete. I am not sure what measure of obsolescence is appropriate, and I suspect that the measure will always be rather fuzzy. In general, the seriousness of obsolescence increases with the length of time since the posting of the announcement with a larger sequencer. It has little or nothing to do with the age of the obsolete announcement itself, nor with the increment in sequencers. The seriousness of obsolescence probably depends on the nature of the particular ancestor involved in the obsoleting announcement, but the nature of that dependence is not obvious, and probably varies according to the intentions of the agents involved.

The attribution of security compromise is even more confusing than resolution. If a handle is compromised, then all of its descendants inheriting authentication from it are presumably compromised as well. Other handles with the same authentication method and the same key are probably compromised, but there probably won't be very many of those, and it seems to much to expect the ONHS to track them down.

- If there has been an authentic announcement of the form

  AnnounceCompromise(<*handle*>, . . .)

  then <*handle*> is *compromised*.

- If <*handle*> is compromised, then all of its descendants inheriting authentication from <*handle*> are also *compromised*.
- If <*handle*> is compromised, then every announcement for <*handle*> is *compromised*, and every resolution containing such an announcement is *compromised*. (Notice that a partial resolution *to* <*handle*> may not be compromised.)

What sort of compromise have I missed?

6 Queries and Checks to Handle Servers

Queries ask for hints derived from announcements sent previously by handle owners.

6.1 Essential Messages

- An *address query* is a message of the form

  QueryAddress(<*handle*>, <*address*>, <*performance*>)

  Such a message requests the latest address, if any, bound to the <*handle*> to be returned to the <*address*>.

6.2 Useful Messages

There are some additional messages that may be useful to a querier, but they are discussed under Section 7 on Audit Requests.

6.3 Policy

Mention trusting the ONHS, end-to-end authentication, selective auditing.

7 Audit Requests

Handle owners, queriers, and other interested parties may audit the operation of the ONHS system, by querying individual announcements and by requesting notification of all announcements for a given handle.

7.1 Essential Messages

- A *notification request* is a message of the form

  RequestNotification(<*handle*>, <*address or handle*>, <*certificate*>)

  where the optional <*certificate*> may come from any agent. Such a message requests that an audit message containing a copy of each authentic and inauthentic announcement for <*handle*>, with its certificate from the putative handle owner, be sent to <*address or handle*>. This message type may need to be refined in the future to allow richer specifications of what announcements to audit.

- A *notification cancellation* is a message of the form

  CancelNotification(<*handle*>, <*address or handle*>, <*certificate*>)

  where the optional <*certificate*> comes from the same agent who made a corresponding notification request. Such a message cancels the stream of audit messages to <*address or handle*>.

7.2 Useful Messages

- An *announcement query* is a message of the form

  QueryAnnounce(<*handle*>, <*address*>)

  Such a message requests an audit of the most current authentic binding, delegation, or transfer announcement for <*handle*>, with its certificate from the handle owner.

- An *announcement check* is a message of the form

  CheckAnnounce(<*handle*>, <*sequencer*>, <*address*>)

  Such a message requests reassurance from a handle server that it is not aware of any authentic announcement of a binding, delegation, or transfer for <*handle*> more recent than <*sequencer*> to be returned to <*address*>.

- A *security query* is a message of the form

  QuerySecurity(<*handle*>, <*address*>)

  Such a message requests notification of any authentic announcement that the security of <*handle*> has been compromised to be returned to the <*address*>, with its certificate from the handle owner.

- A *security check* is a message of the form

  CheckSecurity(<*handle*>, <*address*>)

  Such a message requests reassurance from a handle server that it is not aware of any authentic announcement that the security of <*handle*> has been compromised to be returned to the <*address*>.

8 Hints, Reassurances, Audits, and Declinations from Handle Servers

Responses by handle servers to queries, checks, and audit requests include:

- *Hints* and *warnings*, providing information from announcements, normally in response to queries.
- *Reassurances*, indicating that the handle server is not aware of any announcement of a certain form, possibly in response to checks.
- *Audits*, passing on information from announcements for auditing purposes.
- *Declinations*, indicating that the handle server cannot or will not provide a certain service.

8.1 Essential Messages

- An *address hint* is a message of the form

  HintAddress(<*handle*>, <*address*>)

  Such a message indicates that the handle server is aware of a chain of authentic announcements binding the given <*handle*> to the given <*address*>.

- A *delegation hint* is a message of the form

  HintDelegate(<*handle-1*>, <*handle-2*>)

  Such a message indicates that the handle server is aware of a chain of authentic announcements delegating the binding of <*handle-1*> to <*handle-2*>.

- A *transfer hint* is a message of the form

  HintTransfer(<*handle-1*>, <*handle-2*>)

  Such a message indicates that the handle server is aware of a direct irrevocable transfer of <*handle-1*> to <*handle-2*>.

- A *compromise warning* is a message of the form

  WarnCompromise(<*handle*>, <*time*>)

  Such a message indicates that the handle server is aware of an authentic announcement that the security of <*handle*> has been compromised.

- For each sort of announcement, a corresponding *audit* message copies the announcement to a third party.

8.2 Useful Messages

- A *no-compromise reassurance* is a message of the form

  AssureNoCompromise(<*handle*>, <*certificate*>)

  Where <*certificate*> is a certificate from a handle server. Such a message indicates that the handle server is not aware of any authentic announcement of a compromise to <*handle*>. The <*certificate*> is only intended to demonstrate that the message came from an appropriate handle server, not that the server has complete information.

- A *no-announce reassurance* is a message of the form

  AssureNoBinding(<*handle*>, <*sequencer*>, <*certificate*>)

Where <*certificate*> is a certificate from a handle server. Such a message indicates that the handle server is not aware of any authentic announcement of a binding, delegation, or transfer for <*handle*> more recent than <*sequencer*>. The <*certificate*> is only intended to demonstrate that the message came from an appropriate handle server, not that the server has complete information.

- For each sort of request, a corresponding *declination* indicates that the handle server cannot or will not provide the requested service, and may provide a reason.

8.3 Policy

A handle server should respond to each **QueryAddress** message as promptly as possible with an address or handle bound to the nearest/lowest possible ancestor of the given handle. If the server is aware of more than one binding to the same ancestor, it should prefer the one with the highest sequencer. If sequencers are the same, the server may apply any sensible local policy.

The global system should make a best effort to respond promptly to every address query with the irrevocable transfer of the handle if there is one, and the authentic binding of that handle carrying the latest time stamp if there is no reassignment. The system is useful only if the probability of such a prompt response is very high. The system may guide and/or limit its effort based on performance parameters provided by the querier. If two or more authentic bindings to the same handle tie for the latest time stamp value, the highest priority is to return at least one of them. If the cost is reasonable, the system should return as many of them as possible, or at least announce the existence of the multiplicity.

The system should keep the probability of a false hint very low. If there is a conflict, this takes precedence over prompt correct response. A correct response has little value when buried in incorrect responses. On the other hand, avoiding incorrect responses has some value even when correct responses are absent.

The real technical point of authentication *within the global system* is to keep the rate of false hints low, and to avoid masking a correct announcement behind an inauthentic one. Once a querier is in contact with the correct handle owner, they may perform all sorts of end-to-end authentication completely independently of the ONHS, using other sorts of trusted registries if

they like.

The global system should make a best effort to report a security compromise of a given handle promptly in response to every query of every sort involving the same handle. If a binding is discoverable at a sensible cost, it should provide that binding also. But compromise announcements should be stored in such a way that cost is never or at most very seldom an issue in discovering an authentic announcement of a security compromise.

The global system may find it valuable to give higher priority to discovery of an irrevocable reassignment than to discovery of a binding to an address, so that failure to find a binding will be even rarer in the case of reassignment than in the case of a normal binding.

As long as the cost is moderate, the system should provide certificates authenticating announcements even when they are not requested, and especially in the case of reassignment.

The system should provide the promptest and fullest possible notification and explanation of negative results that is feasible at a sensible cost.

Mention audit policies, priority to handle owner.

9 Examples

References

[1] STUB