Authentication by Email Reception

Don Libes
National Institute of Standards and Technology
libes@nist.gov

Abstract

This paper describes the use of email addresses as an authentication mechanism for public access servers. Intended for untrusted and low-risk environments, this mechanism provides reasonable security at very low cost to both user and server administrator. In particular, the initial and subsequent registrations are totally automated, and problem detection/resolution is highly automated.

Keywords: security, authentication, email reception, email address

Introduction

In this paper, I describe the use of email reception as an authentication mechanism for public access servers, such as email- and Web-based servers in untrusted and low-risk environments [DoD]. Even the simplest implementation provides security that is significantly better than trust and requires significant power to crack. Despite its security limitations, this type of authentication should be attractive for a large percentage of servers that are now currently trust-based. In particular, the system administration cost is near zero because initial and subsequent reregistrations (e.g., dealing with forgotten passwords) is totally automated. Problem detection and resolution is also automated to a high degree. It is this high degree of automation and corresponding low cost that is the motivating factor for this work.

The underlying premise of this work is that the ability to receive email at a particular email address provides a useful type of authentication. I do not claim this technique is new. However, it is rarely used, in part because its implementation and ramifications are not obvious, and in part because it is not well described in the literature. This paper remedies that lack. In addition, I will expand on the idea, show its applicability, and describe several nonobvious benefits.

I will use a simple example to introduce the basic idea. I will note additional refinements later.

Example: Our site (the NIST campus, with approximately 3000 staff) wants to augment our already-existing online phone directory with URLs. Ideally, visitors can then reference our directory and find the home page for any NIST staff member.

The obvious difficulty is: How are we to populate a directory with thousands of URLs from hundreds of servers and in which the URLs follow no consistent naming scheme? There is no way to automate this procedure without staff involvement.

Reprinted from: ***Proceedings of the Fifth System Administration, Networking, and Security Conference (SANS 96)*****, Washington, DC, May 12-18, 1996.**

One solution is to provide a mechanism by which staff could register their own URLs. Since each staff member knows their own URL, they can register themselves. Staff could also use the service to change their registered URL.

The traditional implementation of this solution would use an email alias which staff could fill out with the URL information. In a trust-based environment, this alias would forward email to a program that would immediately update the online directory. In an untrusted environment, the information would be sent to a human administrator who would verify that the request to set a URL came from someone who was authorized to do so. This task is a time-consuming procedure and checking authorization is particularly hard since there is no electronic authentication mechanism established at NIST. The administrator is reduced to using his or her own judgement based on factors such as whether the return address *looks* authentic, the URL *looks* authentic, and so on. Clearly, a secure, automated procedure is more desirable than this manual, trust-based procedure.

Using Email Addresses for Local Authentication

User authentication means that users must be able to convince others that they are who they say they are. Most email envelopes are easily spoofed and are therefore not sufficiently secure. Digital signatures provide a solution in theory, however they can present complications for many users in practice [Rivest]. For example, users must obtain and install whatever digital signature software is required by the server. And the issue of key storage is problematic for many sites [EPIC]. While some people use finger servers to advertise public keys, sites which are not directly on the Internet can not make use of the finger protocol and must provide some other alternative.

In contrast, servers can *offer* passwords securely. Upon request, servers can generate and send email passwords to a specific address. The recipient provides authentication by returning the password. Figure 1 shows an example of this in the URL directory scenario above:.

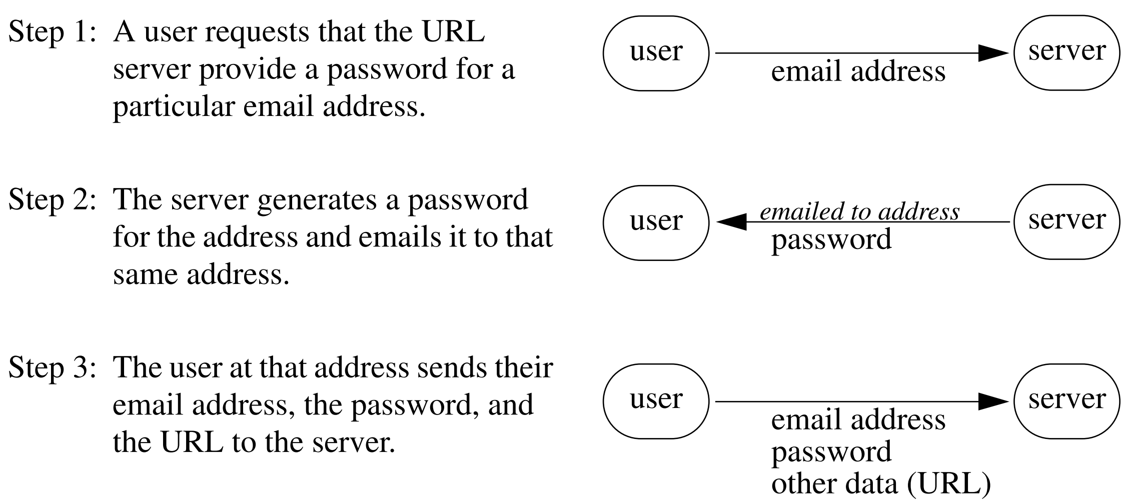

Figure 1: Authentication by email in URL scenario.

Typically the user's email address can be derived from the delivery envelope, but explicitly providing it obviates the possibility of the address being rewritten in different ways depending upon different delivery mechanisms being used for steps 1 and 3. The precise delivery mechanism used in those steps is irrelevant. However, the delivery mechanism in step 2 must be email.

This scenario prevents one user from masquerading as another by simply claiming to be them. If rogue user A asks for a password for user B, the server sends the password to B. Rogue user A does not get the password and thus cannot return it to the server.

This works similarly whether users are on the same host or different hosts. If rogue user A@host1 asks for a password for user B@host2, the server sends the password to B@host2. A does not get the password and thus cannot return it to the server. If rogue user A@host1 asks for a password for user B@host1 (and B is really at host2), the directory rejects this request outright. As a directory service, it will have the list of known email addresses and B@host1 will not be in its directory so the request cannot be valid.

This scenario also prevents a user from masquerading as the server. Rogue user A could behave like a server and send a password to user B, but user B would then send the password back to the true server which would reject the password, since it was not generated by the server in the first place. Conceivably, rogue user A could ask the server for a password and then pass it on to user B. However, user B would need to supply the email address of user A which user B would have no reason to do. Instead, user B would supply their own address and the server would reject the password because it does not correspond to the given address.

Intercepted email can be used to obtain the password and masquerade as another user. In the low-security environment that I assume in this paper, intercepting email is too hard and does not arise. For example, at NIST, tapping into our routers or networks requires specialized equipment that would be so expensive to obtain that nobody would bother – at least just to change a URL. Nonetheless, interception can be solved by augmenting the request in step 1 (above) with a user's public key. The server would save the user's public key and encrypt the generated password with the public key before sending it to the address in step 2. Finally, the user would return the password now encoded with their private key. Subsequent requests from the user requiring authentication would continue to be encoded with the user's private key. Again, this solves the problem of dealing with a trusted key registry or hosts which cannot offer finger service for whatever reason.

The scheme is ineffective if a rogue user has physical access to another user's files or processes. For example, user A can masquerade as user B simply by walking over to their workstation and accessing their mail file to find the password that the server has returned. Not even a digital signature stops this attack and the mechanism described in this paper offers no solution. However, it should be evident that moderately stronger authentication is of no advantage. In our scenario, even if the rogue user could not convince the server to change the URL, the rogue user could still use physical access to directly change the page to which the URL points.

To repeat, the underlying premise of this work is that the ability to receive email at a particular email address provides a useful type of authentication. This type of authentication is obviously not appropriate for all applications but it works well in untrusted, low-risk environments.

The larger the user community, the better this authentication mechanism pays off over the manual approach. As another example, many sites employ staff to maintain a database of email aliases. For example, NIST has aliases for many people who want to be available as *some-nickname*@nist.gov. Similarly, aliases exists for hundreds of projects. This enables mail to, for example, des-bugs@nist.gov to go directly to whomever is responsible for the DES software without the sender first having to find out who that might be today. Maintaining these aliases manually is expensive and has all the same authentication concerns as the URL problem. Unlike URLs, it is not trivial to make sure that aliases are unique. Lag time between user submission and a human administrator responding that an alias is already in use can be very frustrating. Fortunately, the authentication mechanism already described, easily addresses the alias database scenario. Turnaround time and personnel cost is zeroed; reliability and security is increased.

A bootstrap problem – How does the local server know about the user in the first place?

In the previous section, I described a scenario involving a server providing authenticated access to a known set of users. In that scenario, the server was provided with a directory of known email addresses a priori. For this reason, the server can immediately detect attempts to change non-existent URLs. If the incoming email address is not in the directory, the request is rejected.

A bootstrap problem exists here. In particular, how did the users get their email addresses listed in the first place? This obviously requires some mechanism outside that provided by the URL registration mechanism. I will not address this except to say that, presumably, a user could be issued a password to the server at the same time that their email address is entered in the directory. In this case, the user would not have to request a password in the first place. This provides an entirely different solution to the URL authentication problem. However, later I will show some advantages to the dynamic authentication mechanism.

Using email for public access servers

In many contexts, access is truly public. Service is offered to anyone with access to the Internet. This is quite different from the prior scenario in which service is offered to a set of users all known beforehand.

Many public servers operate without any per-user authentication. Anonymous ftp servers are examples of this. In contrast, some servers do require registration albeit to anyone. The registration is occasionally used merely to collect statistics. But more often, registration provides the server with a means to maintain an association of per-user data between transactions of the same user. The real name of the user may not be available and in most cases is totally irrelevant. For example, there may well be multiple people with the same name. The important point is that information entered by a specific user can be distinguished from that entered by another person. User names may be ambiguous, but since email addresses are unique, associating the user's data with their email address suffices. The only problem is how to prove that a user is indeed the one that owns a given email address.

To solve this problem, no a priori directory is necessary. The previous three-step procedure can be used, with the addition that the server creates an entry for the user immediately upon receiving the request for a password.

Most of the analysis is identical to the local server case. The only significant difference is that the server cannot know before responding whether the email is bogus or not. Fortunately, this is irrelevant. The server responds without such knowledge. If email addresses bounce back to the server, the bounce messages are ignored. If email addresses disappear into the void (are never heard from again for whatever reason), the server takes no action – none is necessary. The user will eventually notice the lack of a response and follow up, perhaps by sending another request.

In summary, a pre-existing directory does not actually simplify the authentication problem substantively.

What a pre-existing directory *can* do is offer a built-in association between email address and other data such as user names. Without a directory, users can claim their name is anything including ridiculous names (Santa Claus) or real names that are not likely theirs (Bill Clinton). In many cases, other users can tell immediately whether a name is bogus or not. For example, a Bill Clinton associated with a .edu email address, is likely not the same Bill Clinton at whitehouse.gov. Of course, this detection cannot be done in all cases, but it can be substantially addressed by allowing users access to the known email address and associated user names. If multiple users with the same name are shown, the users can surmise that they cannot trust all of them but one. In the general sense, authenticating user names requires a totally different mechanism than is described in this paper.

Applying this technique to Web access

Most of the techniques described earlier can be provided via Web pages. For example, the user request for a password can be accomplished by having the server provide a form on a page which the user fills out and submits via HTTP. Similarly, once the user has a password, the user can then enter it into other Web pages as authentication. Naturally, the email address must be included at all times since HTTP provides no implicit means of obtaining this information. Variables of type HIDDEN may be used to pass both the password and email address between pages so that they need not be entered repeatedly for each user transaction [BLee].

In contrast, the server must still return the password by email. As before, this is the step that allows the user to prove he or she can receive email at the claimed address.

The following figure shows an example of a commercial service that provides authentication using email reception. The Four11 service offers a white pages directory [SLED]. Figure 2 shows the beginning of the Four11 Web page in which new users can enter their email address. A password is generated and emailed back to the claimed address.

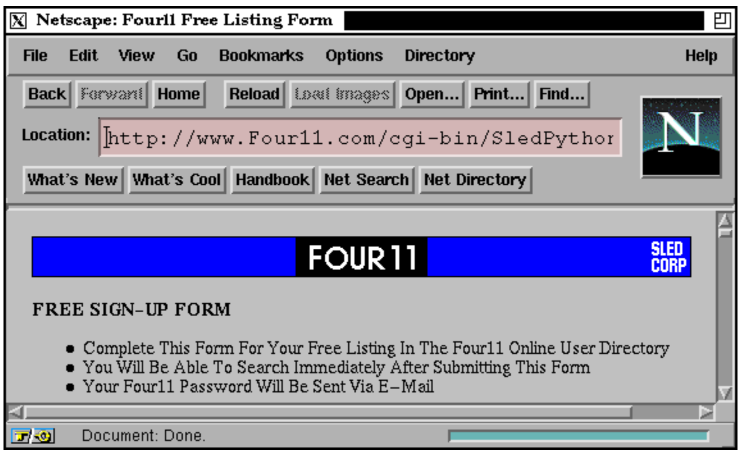

Figure 2: Four11 Web page querying user for email address.

Pathfinder is a commercial news service which also requires registration [TW]. Pathfinder uses a slight variation on the theme. The user is permitted to choose the initial password while filling out the registration form. The server generates a unique URL and emails that back to the address. By visiting the unique URL, the user is registered. An excerpt of the server response is shown below:

```
Dear Don Libes,

Thank you for joining Pathfinder.  Please find below the
instructions for validating your account.  Validating your
account will give you unlimited access to all the great stuff
on Pathfinder - free!

To validate online:

  1) Go to the following URL to be automatically validated:

     http://auth.pathfinder.com/auth/
validate_member.cgi?member=libes&validcode=744170240

To validate via e-mail:
```

```
  1) Cut and paste the following 2 lines of text into an email
message, making sure you don't alter them in any way:

          Member Name: libes
          Validation:  744170240

  2) Send the e-mail message containing the 2 lines to us at:
path-validation@pathfinder.com
```

The approach used by Pathfinder is conceptually the same approach. The only difference is that the password changes upon registration. Initially, the server generates one and provides it embedded in the URL. Once authenticated, the server switches to using the password originally supplied by the user and now known to be authentic.

The extra complexity provided by the Pathfinder approach may appear to simplify the user's responsibility. But for all practical purposes, the user's responsibilities are identical. All public access servers should provide a mechanism for changing the password after initial registration and authentication.

Resolving problems automatically

Automatic problem resolution is a significant attribute of the approach described. Indeed, this benefit is the motivating factor for this work. I have already noted how initial registration is automated. In the remainder of this paper, I will describe other aspects of this type of authentication, how problems may arise in them, and how they can be automated.

Bounced email

Mail sent by the server has the potential to bounce. The most likely reason for this to occur is that the given address is incorrect. However, even email to valid addresses can bounce. For example, this can happen if critical gateways are unavailable for a sufficiently long period of time (typically three days).

Resending bounced email is a fruitless effort. If algorithms existed to examine the address, adjust it, and resend it, the email would never have bounced in the first place. So there is no point in having the server try. Thus, the traditional action taken by most public access servers upon receiving bounced email is simply to discard it. The user who cares enough will eventually retry, perhaps providing a correct address or path the second time around.

Upon repeated failure, a user may give up or try and contact a human administrator associated with the server. A human that can examine the logs at the server can usually diagnose and report (verbally) the problem to the user, but this is a very expensive solution. It is also inconvenient. When the user and administrator are in significantly separated time zones, each phone mail message can take many hours before it is heard. And network delays between the user and server may turn simple test cases into multiple-day ordeals.

A solution to this problem is to record the headers of bounced messages on a Web page. Using this approach, users may browse through bounced messages even if email access to the server is not working. If the user emailed the original password request (rather than submitted it via the web), the user may find out that the request was never even received by the server. By removing the server administrator from the loop, the cost of email problem diagnosis is significantly reduced. Figure 3 shows the page seen by users after registering with the NICS, an experimental service currently being prototyped at NIST [Libes].

Notice that although a phone number is offered, it is intended for use only as a last resort, after the user has viewed the bounced messages and worked with their own local email administrator. If the user ever does

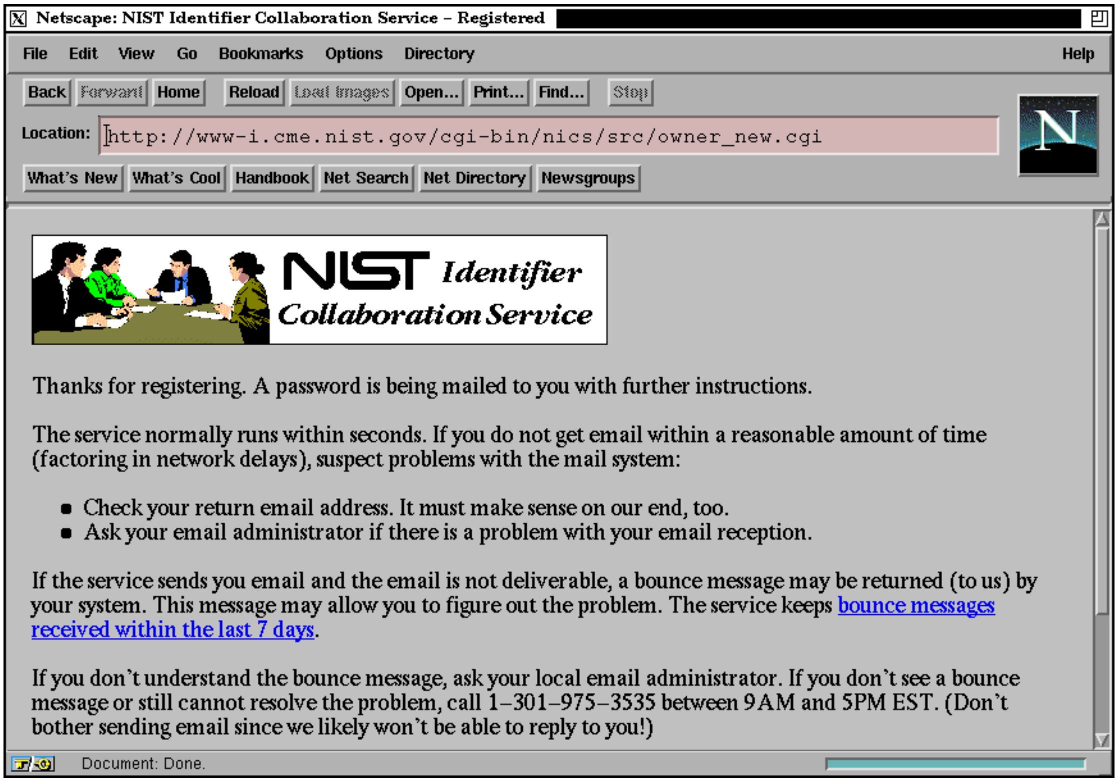

Figure 3: NICS Web page telling users to wait for password in the mail.

reach that point, the human interaction will likely be very efficient since analysis of the bounce records will already have ironed out a very large percentage of the usual possible problems.

As described above, this solution requires that the user have HTTP access to the net. However, this can be provided in proxy form. For example, if the user is not directly on the Internet, email to a proxy HTTP server suffices to obtain Web access to the problematic server. If the user cannot send email to a proxy HTTP server, they almost certainly have substantial email problems that will show up in many other email scenarios to the point that they should be able to diagnose these problems immediately.

Mail from the server that is never acknowledged

Mail from the server to the user may go unacknowledged for a variety of reasons. For example, the user may have lost interest in using the service. In this case, no matter how long the server waits, the password will never be returned. There is no point in the server maintaining unauthenticated accounts indefinitely. It suffices to wait a finite amount of time and after that, erase any record of the account and password.

The specific amount of time to wait depends upon the maximum amount of time it could take to email a message from the server to a user. To that, it is reasonable to add several days to account for the possibility of the password arriving at the beginning of a three day weekend. Since it is impossible to provide precise figures on email delivery bounds, we have intuitively chosen a seven day period as the timeout for servers that we run.

Lack of acknowledgments may also be due to technical reasons. I addressed bounced email earlier. Another interesting possibility to consider is that the user may have accidentally supplied the address of a different user. I will describe this in the next section.

Unexpected email from the server

A user can unexpectedly receive email from a server if someone else supplies the user's email address when requesting an account. Assuming the email address is provided mistakenly, the user will not be expecting the server email. For this reason, the server should reasonably attempt to explain to the user why the email arrived. The following example was generated by the NICS service at NIST mentioned earlier. This message demonstrates a reasonable attempt at explaining what went wrong while allaying concerns of a security problem. (Presumably, the user will not be familiar with the concept of authentication by email reception.)

```
Subject: NIST Identifier Collaboration Service Registration
You are now registered with the NIST Identifier Collaboration
Service.
Your owner name is Bill Clinton
Your email address is libes@nist.gov
Your password is Vo2M

You should now further customize your account by going to:
<URL: http://www-i.cme.nist.gov/cgi-bin/nics/src/
owner_list_with_attributes.cgi?OwnerNames=18+Bill+Clinton>
Note: If you do not visit this URL in the next 3 days, you will
be deregistered from the service.

--
This message was generated by an automatic service at NIST.
If you have no idea what this message is all about, it is
possible that someone tried (but failed) to masquerade as you.
If this is the case, simply ignore this message.

For more info, see:
   http://www-i.cme.nist.gov/cgi-bin/nics/src/welcome.cgi
```

As a reminder, even if the user does return the password, the user is authenticated as themself and cannot masquerade as anyone else. For instance, the example above might have been generated by a person named "Bill Clinton" but an email address of libes@nist.gov was provided. The user at this email address can return the password to complete the authentication only with the stated email address of libes@nist.gov. The name "Bill Clinton" remains unauthenticated – at least by itself. Other users will recognize that the name itself is not necessarily authentic; only that the name was authentically set by the user with the email address libes@nist.gov.

In most cases, unexpected server email will not be acknowledged to the server but will be discarded by the user.

Forgotten passwords

Traditionally, users who forget their passwords must contact a human administrator for assistance. This shares all of the unfortunate characteristics as that of handling bounced email – it is expensive and inconvenient. Even worse, the human must somehow be convinced that the user is not masquerading. This is very difficult without a password and all the more so in a public situation in which the administrator has no idea of who the user is, including what they should sound like or look like.

Fortunately, it is possible to address forgotten passwords without human involvement. The solution is to have users send a request for a new password similarly to their original request. In fact, the same exact request can be supported. Users do not have to show any proof of who they are. As before, authentication is provided by the server emailing the password back to the specified email address. If one-way encrypted password are used, the server merely generates a new password and sends it to the associated email address.

To avoid denial-of-server attacks, the server must provide a means to allow the user to continue authentication even if a new password has been requested by a rogue user. This is achieved by the server maintaining both old and new passwords. Only after the new password has been returned to the server with the correct email address is the old password discarded.

Changing email addresses

Users occasionally change email addresses. Servers can support the ability for users to change this without involvement of a human administrator. To do this, the user submits (either via email or web) old and new email addresses and the password. This allows the server to authenticate the old address.

Rather than simply discarding the old address and switching to the new one, the server maintains both temporarily. A new password is generated and sent to the new address. The new email address is authenticated when the new password is returned, demonstrating the user can indeed receive email at the new address. Only at this point is the old email address and password discarded by the server.

This allows the user continued access to the server using the old email address and password while waiting for the email address change to be processed. This avoids the possibility of users work being delayed while waiting for something that is, in essence, administrative overhead.

User-changed passwords

Users can change passwords. There is nothing particularly interesting about this as far as its impact on the server, but I mention it here for completeness and to allay any concerns that this is not the case. In order to change a password, the user sends the email address and both old and new passwords. The server discards the old password and saves the new one. No further authentication is required.

Earlier, I mentioned the possibility of using one-way encrypted passwords. This is a good idea any time passwords are used as it reduces the risk of exposure due to an operating system flaw, system administrator error, and any other error not related to the design of the server itself. However, none of the algorithms described in this paper actually require the use of one-way encrypted passwords. For instance, plaintext passwords may well be sufficient on dedicated systems that do not allow normal user logins.

Other variations on a theme

Other variations of the authentication by email reception exist. For instance, earlier I mentioned that these techniques could be augmented with a private/public key mechanism to prevent eavesdropping.

Another possibility is that the server can issue passwords only for a single use. Much like a one-time pad, the server would throw away the password after a single use [Litterio]. Each time a user wanted to carry out a transaction, the user would request a new password. One possible application of this might be for a (different) server wanting to behave like a user. Whereas a user can remember a password (in relatively secure wetware), a server might have no place to safely store a password between runs. Instead of using a single password, the server could apply for a one-time password in order to carry out transactions as needs arose.

Summary

Authentication by demonstrating the ability to receive email at a specific address is very useful. While only applicable to a particular class of applications and in low-security environments, this characterizes a large number of public servers, performing both research and commercial services. One particularly valuable benefit of this type of authentication is that problem resolution can be automated to a very high degree with a corresponding low cost for human administration. In addition, there is no requirement to the user to obtain additional security software or hardware.

Acknowledgments

Thanks to Christine Piatko and Steve Ray for comments and corrections to this paper.

Disclaimer

Certain trade names and company products are mentioned in order to adequately specify procedures and equipment used. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

References

[BLee] T. Berners-Lee, D. Connolly, "Hypertext Markup Language – 2.0, RFC 1866, HTML Working Group, IETF, Corporation for National Research Initiatives, URL: http://www.w3.org/pub/WWW/MarkUp/html-spec/html-spec_toc.html, September 22, 1995.

[EPIC] "Commercial Key Escrow", URL: http://www.epic.org/crypto/CKE, Electronic Privacy Information Center, 1995.

[SLED] "Four11", URL: http://www.Four11.com, SLED Corp, 1995.

[Litterio] Francis Litterio, "Why are One-Time Pads Perfectly Secure?", URL: http://draco.centerline.com:8080/~franl/crypto/one-time-pad.html, Centerline Software, 1995.

[Libes] Don Libes, "NIST Identification Collaboration Service", URL: http://www-i.cme.nist.gov/cgi-bin/ns/src/welcome.cgi, National Institute of Standards and Technology, 1995.

[DoD] "Trusted Computer System Evaluation Criteria (The Orange Book)", DoD 5200.28-STD, URL: http://hightop.nrl.navy.mil/docs/orangebook.html, Department of Defense, December, 1985.

[TW] "Pathfinder", URL: http://pathfinder.com, Time Warner, Inc., 1995.

[Rivest] R.L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, 21(2):120--126, February 1978.