Trustworthy Access Control with Untrustworthy Web Servers

Tim Wilkinson, Dave Hearn and Simon Wiseman
Defence Evaluation and Research Agency
Malvern, England

{t.wilkinson,d.hearn,s.wiseman}@eris.dera.gov.uk

Abstract

If sensitive information is to be included in a shared web, access controls will be required. However, the complex software needed to provide a web service is prone to failure. To provide access control without relying on such software, encryption can be used. Bob is a prototype system that supports complex access control expressions through the transparent use of encryption.

1. Introduction

The business benefit of an Intranet web is that information is available to those that need it in a timely fashion. However, most large organisations have some information that is considered sensitive and is not needed by all users. For example, Human Resources data might need sharing amongst members of the HR department, while other people are prevented from accessing it.

Existing solutions to this problem, for example [1], rely on complex web server software working correctly and being configured correctly, which means there is considerable risk that the controls will fail. In many commercial organisations, as long as the information remains on the company Intranet, the risks involved will be worth taking, given the relatively limited damage that would be caused if the controls fail. However, an organisation which handles particularly sensitive data, such as health care records or defence intelligence information, may find the risk unacceptable.

With increased use being made of electronic commerce to make trading more efficient, the boundaries of an Intranet are fast being eroded. Increasingly, an organisation will host some proprietary information belonging to its trading partners on its Intranet and these partners may need some access to the Intranet in order to conduct business. Typically, the partners will be in competition with each other and the host organisation would need to ensure that the information belonging to one partner is not revealed to another (either accidentally or deliberately). Should an access control failure occur, damage to the host organisation's reputation might lead to lost business and even claims for damages. In these circumstances, a commercial organisation may find the risk of complex access control software failing hard to justify to the shareholders or potential customers.

One way of controlling access to information in a web without relying on the web server software is to use separate servers for information of different sensitivities. Unfortunately this solution does not scale when many combinations of information sensitivity and user trustworthiness are required.

The only way a single untrusted web server can be used to handle information of different sensitivities is to remove responsibility for access control and separation from the server software. This can be achieved by encrypting documents, in a key not available to the server, before they are given to the server. This removes access control responsibilities from complex web server software and becomes a matter of distributing data decryption keys appropriately. Unfortunately, the general problem of key distribution is by no means a simple task, but some options are described in section 3.

This paper describes a prototype system called Bob that uses an encryption based approach to provide trustworthy access control in a web based on untrustworthy web servers. Bob was initially developed by the Defence Evaluation and Research Agency (DERA) for the UK Ministry of Defence. The concept, however, is a generic one and additional work

has shown how Bob can be used to protect Electronic Patient Records in a medical environment.

In the next section the basic access control scheme is described, then the problem of key distribution is discussed. The question of how the release of information into the web is controlled is then tackled as well as possible ways of reconciling the conflicting requirements of data discovery and confidentiality. Finally, techniques for handling dynamically created content are considered.

2. Access Control

2.1 Access Control Expressions

The access control scheme can be described in terms of groups, each containing a number of users. These groups will usually represent a particular business function, project or trading partner. Each file accessed through the web server is labelled with an Access Control Expression (ACE), which indicates those users who are permitted to observe the file.

An ACE is a formula defined in terms of groups combined with operators "&" and "|", which are 'and' and 'or' respectively. Files with an ACE of the form "X & Y" can be observed by any user who is in group X and group Y, while files with ACEs of the form "X | Y" can be observed by any user who is either in group X or group Y.

Complex ACE formulae can be used, and some examples are shown below:

ACE	required groups
X & Y & Z	X Y Z
X & (Y | Z)	X Y or X Z
(W | X) & (Y | Z)	W Y or W Z or X Y or X Z

Suppose an organisation had a number of departments that handle sensitive information, including Engineering (ENG) and Finance (FIN). In addition, the organisation handles sensitive information belonging to its customers, who include ACME and DERA. A group would be created for each department and for each customer, and staff would be placed in these groups according to the departments for which they work and the customers that they serve.

Now sensitive engineering data about work for ACME would be labelled "ENG & ACME". An engineer who was not working on the ACME project would not be in the ACME group and so would be unable to see this data. Similarly, sensitive financial data about work for ACME would be labelled "FIN & ACME".

However, if the organisation were working on a joint project for ACME and DERA, the engineering details might be labelled "ENG & (ACME | DERA)", in which case any engineer working on an ACME (or DERA) project will be able to see details of the joint project as well. Alternatively, the data might be labelled "ENG & ACME & DERA", in which case only engineers who work on both ACME and DERA projects would be able to see the data.

2.2 Access Mediation

The ACE applied to a file accessed through the web server is not in itself used to mediate access. Instead, when the file is released into the server its ACE is used to determine the way the file's data is encrypted. The scheme uses a mixture of symmetric and asymmetric cryptography as follows.

When a file is released, a new symmetric key is generated and this is used to encrypt the file. This key is called the file's data key. The resulting encrypted data is prepended with a header before being released to the web server. The header contains the information that allows legitimate recipients to decrypt the encrypted data.

An asymmetric key pair is generated for each group in the access control scheme. This key pair is used to distribute a file's data key to those who are permitted to observe the file. One key of the pair is a key encrypting key and the other is a key decrypting key. The encrypting key is used to release information to the group, and the decrypting key is used by members of the group to observe data released to them.

In the simple case where the ACE is just a single group, the file's data key is encrypted using the group's encryption key. The result is placed in the header along with the file's ACE, as shown in figure 1. The way in which the data key is encrypted in general is explained in Annex A.

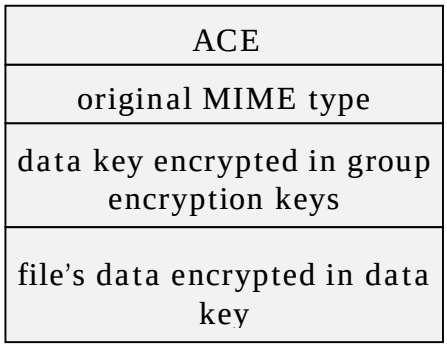

Figure 1: Format of protected file

To observe a file, the ACE in the header is examined to determine how the encrypted data key should be recovered. In the simple case, where the label is just a single group, the group's decryption key is used to recover the file's data key from the header. Once the data key is obtained, the file's data can be decrypted. If the group's decryption key is not available, because the user is not a member of the group that is permitted to observe the file's data, there is no way the file's data can be accessed.

When HTTP is used to retrieve a file from a web server, the reply includes information about the type of the file. This information is included in the HTTP Content-Type reply header field, whose format is a MIME type. Standard web servers use so-called 'mailcap' files to determine, on the basis of file extension, which MIME type is to be associated with each file they deliver. In Bob, all encrypted files are given an extension of ".bob" and a MIME type of "application/x-bob" is associated with this.

When Bob format data is decrypted, in a manner that is described later, the type of the result is changed to the original type taken from the header. This means the browser knows how to handle the data in the normal way.

2.3 Protecting the Group Keys

Most applications of public key cryptography assume that a user's application software can be trusted to protect keys from disclosure and to use them only in accordance with the user's wishes. Here, however, the assumption is that complex web server software cannot be trusted, and so the same level of distrust must be levelled at the workstation applications. Thus a group's decryption key must not be made available to a user's ordinary application software, as this could pass the key to other users who are not part of the group.

The solution is shown in Figure 2. An HTTP decryption proxy is installed on the user's workstation and access controls provided by the workstation's operating system are set so that the proxy has access to a file containing the user's group decryption keys, but the user's application software is denied any access to this file. The access controls are also used to protect the proxy's binary image and configuration data from modification. The need for operating system access controls to protect the use of cryptographic mechanisms is discussed fully in [2].

The job of the decryption proxy is to transparently decrypt any encrypted data retrieved from a web server and to restore the original MIME type of the data. The proxy is trusted to keep the group decryption keys and all document keys private, regardless of what data it handles (for example, it defends against buffer overrun problems).

The user's web-enabled applications, including their browser, would be configured with the local decryption proxy as their web proxy, while the decryption proxy would be configured to chain-on to the network's real web proxy if one is required.

A group's decryption key is protected so that an application cannot pass it on to users who are not in the group, as this would give the recipient access to all files released to the group. Similarly, a file's data key is protected, otherwise this would give the recipient access to the particular file. However, once a file has been decrypted and given to an application, the cryptography does not stop the application passing the decrypted data to another user. This is part of the general problem of controlling the release of data while using untrustworthy application software, which is discussed in section 4.

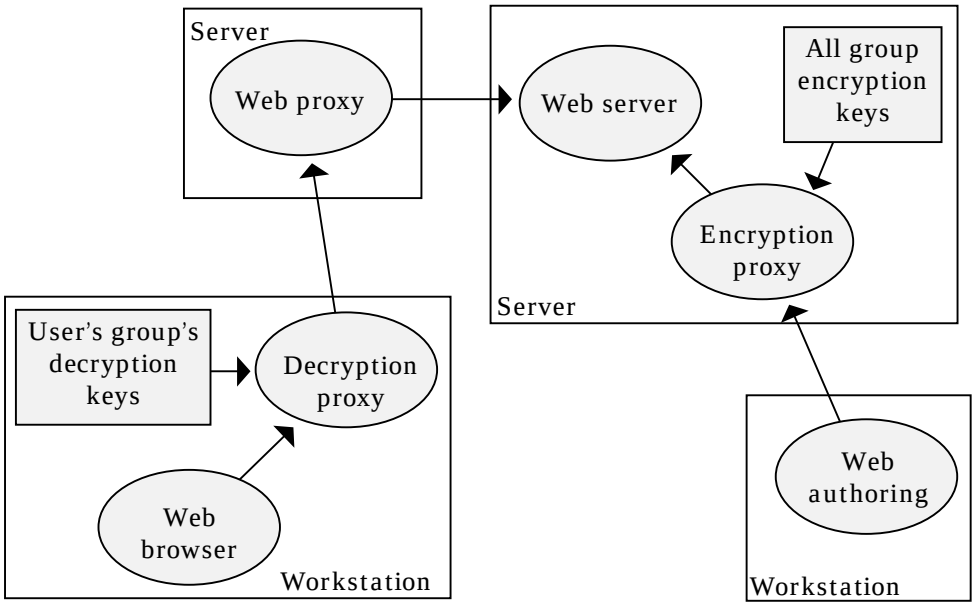

Figure 2: Overall architecture

Protecting a file's data key from disclosure also affords extra protection to the group decryption key. A user in possession of a document key, and the same key encrypted with a group encryption key, has the potential to mount a brute force attack to obtain the group decryption key. With a single document key, the user has only a small amount of information on which to base their attack, but if they can find several documents released to the same group, the brute force effort required will decrease.

Application software used by authors to create web content must be prevented from modifying group encryption keys. This is because the application, which must be considered untrustworthy, could gain access to all data subsequently released by replacing the group encryption key with one for which it knows the corresponding group decryption key. Since there is no need for anyone to know the encryption key, it seems prudent to keep it private as well as to prevent its modification.

Note that, having protected both the encrypting and decrypting keys from disclosure and modification, it would be possible to use symmetric cryptography for the group keys. The advantage of asymmetric cryptography, however, is that it gives extra protection in the event that proxies are compromised. For example, should a server's group encryption keys be divulged, no data is compromised if asymmetric cryptography is used.

Web content is typically created on a workstation and uploaded into the web server using FTP or HTTP. The process of releasing web content can be controlled by placing a proxy, for the appropriate protocol, between the web authoring application and the web server. This encryption proxy needs access to the all the group encryption keys, so it can encrypt a released file in accordance with its ACE. The encryption proxy is trusted to allow the group encryption keys to be modified only under strictly controlled circumstances, discussed in the next section. In addition, the proxy keeps the encryption keys private, though this is less important.

Figure 2 shows the placement of the encryption proxy in the current Bob implementation. As an alternative, the proxy could be placed on the user's workstation, which has the advantage of protecting the data's from eavesdropping as it passes from workstation to server. The disadvantage, however, is that the encryption keys need to be more widely distributed.

In order to know how the file's data should be encrypted, the encryption proxy needs to know the file's ACE. The way this is conveyed from the web authoring software running on the user's workstation to the proxy is discussed in section 4.

If encryption is to be used for access control, the problem of key revocation and replacement must be addressed. Key replacement will be needed occasionally to defray the risk of key compromise, and may also be performed on demand when keys are

known to have been compromised. Keys might be compromised because proxies fail to hold the keys securely, the Public Key Infrastructure (PKI) might fail to deliver them securely, or a brute force attack might be successful.

A group key can be replaced by creating a new group identity and associated key pair. All users in the affected group can then be put in the new group, and the new decrypting key distributed accordingly. Then, all files with ACEs that mention the affected group are found and their ACEs are updated to replace the affected group with the new group. In addition, the data key is recovered and re-encrypted in accordance with the new ACE. Once all files have been processed, the original group is redundant and users can be removed from it. Performed in this way, group key replacement need not be completed as an atomic action and may even be carried out as a background task, depending upon urgency.

An individual document key can be changed easily. It is simply a matter of recovering the original file data key, using the decrypting key of some group which can access it, decrypting the data, and replaying the normal process associated with publishing.

3. Key Distribution

3.1 Public Key Infrastructure

The decrypting group keys of the groups to which a user belongs, need to be distributed privately to the decryption proxy on the user's workstation. One way of achieving this is to make use of public key technology. Each proxy would be identifiable by a distinguished name and associated public key, most likely wrapped together into an identity certificate. The proxy would hold the complementary private key in private local storage. An administrator wishing to place a consumer group decryption key into a proxy would obtain the identity certificate corresponding to the proxy. After verifying the certificate, the public key contained within it can be used to encrypt a group key for forwarding to the proxy. Only a holder of the proxies' private key can unwrap the group key.

At this point the message containing the hidden group key can be presented to the user of the system by, for example, electronic messaging. Once the message has been inserted into the proxy, the proxy can unwrap the message to reveal the group key and place it in private storage. Additional fields could be associated with the key, such as a time after which the key is invalid.

There is still an issue of how the proxy's private key is made available to the proxy initially. In organisations that prefer central key generation, the private key could be physically or electronically delivered to the proxy in a secure manner, and then imported through a trusted import function. Alternatively, the proxy could generate its own private key at installation time, and export the corresponding public key for signature by a certification authority.

3.2 Key distribution with NT security

While the ultimate solution is to distribute keys through a public key infrastructure, as discussed above, a lighter-weight alternative is possible using the security mechanisms of a networked operating system. These mechanisms only work in well-managed closed networks, so the technique will not always be applicable, but where the operating system's environmental assumptions hold it is perfectly adequate.

A key distribution scheme for Bob has been implemented using the security functionality provided by Windows NT. The relevant features are Services and Named Pipes.

A Named Pipe is a communications pipe mechanism whose use is subject to NT security in much the same way as files. A server process on one machine can create a named pipe and set its access control list so that only processes running under certain user accounts can connect to it. When a client process does connect to the pipe, the server process can establish the identity of the client account.

A Service is a process that is started when a machine boots and generally runs under a special system account, rather than one associated with a particular user. Ordinary users may subsequently log-in to the machine and the service continues to run.

More details about these features and a description of how they can be used to build systems for handling sensitive information is given in [3].

Figure 3 shows the general arrangement of processes and services used to distribute group keys. A simple database of group decryption keys is stored on a key server host. The decryption proxy on each workstation is installed as a service and this runs under a special system account. These proxies obtain the user's group keys from a process, the Key Server, using a Named Pipe. The Key Server could reside on the web server host, though it would be better to install it on a more tightly controlled machine.

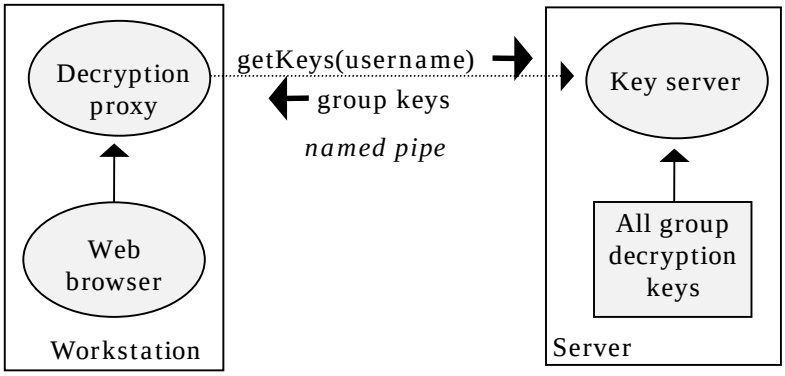

Figure 3: Distributing keys using NT security

The decryption proxy runs as soon as the workstation boots. Whenever it detects that a user has logged-on to the workstation, the proxy connects to the key server's Named Pipe and sends the account name of the user who has just logged on. The key server obtains a list of groups to which the user belongs and then returns a list of decryption keys for these groups to the decryption proxy.

Once the decryption proxy receives the list of group keys for the user, it can transparently decrypt any encrypted data returned from the web server. However, the proxy must ensure that any incoming connections are not from a remote workstation, in case a user in a different group is logged-on there.

The access control lists on the key server's Named Pipe are set so that only the service account used by the decryption proxies can access it. A user's application processes therefore cannot obtain any decryption keys from the web server.

4. Controlling Release

On a well-managed web site, files are not changed in an ad-hoc way. Subsets of web pages and links are updated or otherwise modified, and then uploaded to the server in a publishing operation. It is within this publishing function that access control requirements can be stated and release can be sanctioned.

The first step in publishing is for an author to create one or more documents for publication. Each document needs to have an ACE associated with it. The way this is done depends upon the application used and the environment in which it runs. A simple version might use Microsoft Word to create the documents, in which case the ACEs can be held as document properties. If the workstation provides support for labelled documents, the ACEs could be derived from the security labels of the documents. The Bob demonstration does this, using NT workstations augmented with Deep Purple [4] to provide the labelled documents.

Once the documents have been assembled, they must be released to the web server. Since the assumption is that application software is not sufficiently trustworthy to protect documents from disclosure, the release process must be controlled. In Bob, release is handled by a trustworthy service running on the user's NT workstation. Ordinary applications can request this service to release files to the web server. To defend against an application making inappropriate requests to release some data, the user is asked to confirm each request.

The release service obtains the user's sanction using a trusted path interface, to avoid the sanction being spoofed by an application. A trusted path interface is supported directly by Deep Purple, which uses NT's standard access controls on Desktops to implement it. As part of the release sanction, the user is asked to confirm the ACE for the product to be released. This prevents the application from changing the ACE after the user has set it and before the file is released.

The release service may also check the content of the files to be released to ensure that no data is hidden from the casual reader. This is important as an application may attempt to leak data by hiding it in files that are to be released. While checking for hidden text, the service may also generate a summary of the file's content. This can be presented to the user when they are asked to confirm the release, so that an application that attempts to change the data being released may be discovered.

For example, suppose an author prepares a web "page" comprising some HTML text and two images in GIF format. The release service can check that the application has not hidden sensitive data in comment tags in the HTML, and if any is found the user can be warned not to release the data. In the trusted path dialogue, which asks the user to confirm the release request, the "page" would be summarised, so the user can see the number of paragraphs of text and the number of images being released. If this summary is not as expected, the user has a chance of rejecting the confirmation request.

The release sanction can be obtained separately for each "page", or a single sanction for all the "pages" to be released as one "product" can be obtained. Whilst the former is in principle more secure it could also be seen as an inconvenience. It is common for users to take more care over a single operation than one they need repeat many times. Hence better overall security might be obtained by adopting a more relaxed approach in which one update involving several "pages" is sanctioned as a whole.

Assuming the user confirms the intention to release the data, the release service proceeds to upload the files to the encryption proxy on the server. It is important to prevent applications directly uploading data to the server's encryption proxy, as this would provide them with a way of leaking data. This protection can be achieved by using cryptographic techniques, but in a closed NT based environment named pipes can be used.

5. Searching and Dynamic Content

The usual approach to information discovery is to allow web sites to be indexed by a search engine, and to allow users browsing the site to search the indexes. The Bob approach is to protect sensitive data from complex web servers that are hard to trust. Search engines are also complex software, and at present there seems to be no reason to trust them more than a web server.

Hence it is not appropriate to provide search engines with sensitive data in plain text, and yet it is pointless to provide them with the data in an encrypted form because they then cannot index it. Thus a balance needs to be struck between confidentiality of information and the ability to locate information.

Data that all users are permitted to view need not be encrypted. A search engine can then be used to index this unencrypted data and allow it to be searched, without the complex software being trusted. If, as part of the process of preparing sensitive documents for the server, the author creates an abstract which is visible to all, the abstract can be made available via the search engine and would provide a means for information discovery. The abstract can include a link to the encrypted document, so that it is readily available to any user that is a member of the intended audience.

Another issue is dynamic content, where web pages are generated on demand based on the data in a database. For example, when a user browses a dynamic page, a CGI script may access a database and create some HTML that is returned to the user.

With Bob, the script passes the appropriate request to the database, but the sensitive results are returned as encrypted Bob files. These are embedded indirectly into the generated web page by using HTML <OBJECT> tags. Each <OBJECT> tag is a link to an encrypted result, but the data referred to is displayed in place in the web page, rather than being shown as a hyperlink. Thus if the user's groups are such that they can access all the results, the page is completely filled in, while if they are not some fields will display an error message.

The Bob encryption process could be included in the database engine, by exploiting the Object Relational features of Oracle 8 or Informix IUS [5], or a separate trusted server process could be interposed between the scripts and the database. Alternatively, the sensitive data could be encrypted before it is placed in the database. This has the advantage that the database engine need not be trusted to handle the sensitive data properly, but the disadvantage is that the data cannot be searched or manipulated (e.g. projection) within the database.

Independently of when the data is encrypted, its release into the database must be sanctioned, as the producer is not involved when the data is served out to a requestor. Techniques for doing this using object-relational database engines are discussed in [5].

6. Performance

The use of encryption as an access control mechanism adds an overhead to the release of documents into the web server and to their retrieval by a browser.

Consider first the simple case, where an ACE contains just one group. The release of a document requires that a data key is generated, the file is encrypted in the data key and the data key is encrypted in the group key. To retrieve a document, the data key must be decrypted using the group key and the file must be decrypted

using the data key. Overall, the impact on performance is negligible and no noticeable delay is introduced.

In the general case, however, the overhead can be much higher. When a document is released, the data key must be encrypted using a group key many times. The number of encrypts using some group key is the number of elements in the ACE. This could be noticeable in terms of performance and latency, but any decrease would be negligible compared to the overhead of checking whether the document is suitable for release.

The overhead can sometimes be reduced as some ACEs can be re-written to have fewer terms while retaining the same meaning in terms of access control. For example, the ACE (A&B)|(A&C) can be re-written as A&(B|C), and this requires one less encrypt.

Generally, retrieving a document with a complex ACE is faster than releasing it. This is because there are many different collections of group keys that permit the data key to be recovered, and it is possible to choose the smallest such collection, to which the user belongs, by consulting the file's label. For example, suppose the ACE is (A&B)|(C&D&E)|(F&G) and the user is in groups C through G. By consulting the label in the file's header, it is possible to determine that only two decrypts are required, using the keys of groups F and G, whereas seven encrypts would have been needed to release the file.

Although there is a performance overhead introduced by the decryption of the file's data key, this can be performed in parallel with the transmission of the file's encrypted data. Thus this introduces no delay when retrieving a file with a long ACE.

7. Similar products

Although the use of encryption to protect access to resources is not a new idea, the technique employed in Bob has three significant features.

First, the encryption and decryption is not visible to the end-users. Contrast this with the Formlock[1] product from General Network Services, Here, when a user attempts to retrieve an encrypted document they are presented with a dialog box asking them what action should be taken.

Second, Bob's access control scheme supports complex access control expressions. Various cryptographic envelope schemes, for example IBM's Crytolopes[2] and DigiBox from InterTrust[3], also use encryption to control the distribution of data, but in contrast to Bob these control access simply by controlling the distribution of decryption keys to individuals.

Third, the encryption in Bob is used in a way that removes the need to trust complex, and hence untrustworthy, software. In common with other cryptographic access control solutions, Bob does not need to trust the web server, but unlike some other solutions it also does not need to place trust in the client's browser and associated plug-ins. Instead, Bob's implementation minimises the need to trust software hosted on the workstation, limiting it to just that which handles keys.

8. Conclusions

Web server and browser software is complex and its security features are prone to failure or misconfiguration, and hence cannot be trusted to handle sensitive information appropriately. Bob avoids this problem by ensuring that the web server only handles encrypted data and that release of data from the browser is carefully controlled. With Bob access control in the web is reliant upon:

- operating system access controls in the workstations and servers;
- the correct operation of the special encryption and decryption proxies;
- ensuring released data is labelled with an appropriate access control expression;
- the generation and distribution of the group encryption and decryption keys.

Using encryption to protect information does not solve all the problems, because it is necessary to defend the cryptographic elements from misuse by the untrustworthy servers and applications [2]. The basic protection mechanisms needed in the workstations and servers are found in Windows NT and Unix, but initial key distribution remains a difficult problem to solve in general. Key distribution in a closed NT environment is, however, straightforward.

The Bob solution does not remove the need for trusted software, but it reduces the scale considerably. Rather than trusting web servers and browsers, including all their plug-ins, only the encryption and decryption proxies and the release server need to be trusted. These are quite easy to trust as they are small and simple.

[1] http://www.gns.ca/

[2] http://www.software.ibm.com/security/cryptolope/

[3] http://www.intertrust.com/

Controlling the release of data into the server is not a trivial problem, because to be effective the controls must be closely integrated with web authoring application software. Such software is relatively immature, but progress in standardising distributed web authoring and versioning extensions to HTTP[1] should simplify the design of the release service and make it more widely applicable.

Finally, the addition of access controls into a web conflicts with the natural intention of a web to be freely accessible. This creates considerable tension, as evinced by the problems associated with search engines. This aspect of the problem is worthy of more research.

9. References

[1] "Role Based Access Control for the World Wide Web", J.Barkley *et al*, Procs. 20th National Information Systems Security Conference, Baltimore, October 1997.

[2] "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments", P.Loscocco *et al*, Procs. 21st National Information Systems Security Conference, Crystal City, October 1998.

[3] "Adding Security Labelling to Windows NT", S.Wiseman, Information Security Technical Report, Vol 3, Num 3, Elsevier 1998.

[4] "Private Desktops and Shared Store", B.Pomeroy and S.Wiseman, Procs. 14th Annual Computer Security Applications Conference, Scottsdale, December 1998.

[5] "Securing an Object Relational Database", S.Lewis and S.Wiseman, Procs. 13th Annual Computer Security Applications Conference, San Diego, December 1997.

[1] http://www.ics.uci.edu/pub/ietf/webdav/

Annex A: Encrypting File Keys

A file's header contains the file's ACE and the file's data key encrypted in a way determined by the file's ACE. The function for encrypting the data key of a file D whose ACE is A is denoted H(D,A), and is defined as follows:

^ is the concatenate operator

$H(D, G) = e_G(D)$
$H(D, x \mid y) = H(D,x) \text{ ^ } H(D,y)$
$H(D, G \& x) = e_G(H(D,x))$
$H(D, (x|y) \& z) = H(D, (x \& z) \mid (y \& z))$

where

D is the file data key
G is a simple ACE of one group
x, y and z are arbitrary ACEs
$e_G(\alpha)$ is the result of encrypting α in the encrypting key associated with group G

To observe a file, it must be decrypted using its data key. This can be recovered from the file's header if certain group decrypting keys are known. The ACE determines which combinations of group decrypting keys permit the data key to be recovered.

The function that is used to recover a data key from the encrypted data E and ACE A in the header is denoted R(E,A). This either retrieves the decryption key or returns an error. It is defined as follows:

R(E, G) = If user in G then $d_G(E)$ else fail
R(Ex ^ Ey, x | y) = either R(Ex, x)
or R(Ey, y)
or fail if both fail
$R(E, G \& x) = R(d_G(E),x)$

where

E, Ex and Ey are encrypted key data from the header
G is a simple ACE of one group
x and y are arbitrary ACEs
$d_G(\alpha)$ is the result of decrypting α in the decrypting key associated with group G