# Using the Web to Provide Private Information
# -or-
# A Short Paper About Password Protection Without Client Modifications

Bjorn N. Freeman-Benson
School of Computer Science, Carleton University
514 Herzberg Building, 1125 Colonel By Drive
Ottawa, Ontario, K1S 5B6 CANADA
bnfb@scs.carleton.ca

## 1. The Problem

The World Wide Web offers simple, unified access to terabytes of diverse public databases ranging from high energy physics and computer science through molecular biology and Scandinavian literature. However, the key word is *public*: the Web's simple and well-defined protocols (e.g., URLs and HTTP ), data formats (e.g., GIF and HTML), and browsers (e.g., Mosaic) were designed for easy access to free, public databases. An obvious and useful extension of the existing Web is to provide access to *private* databases.

For the sake of this paper, the term *private database* refers to an information source which restricts access to certain individuals. The reasons for restricting access can vary, and may be because the database contains confidential information, or because the database is commercial and only supports paying customers. The private databases discussed in this paper are the confidential information type, restricted to individuals (such as banking records) or to groups (design documents for XYZ Corporation's unreleased product). Restricted access can be implemented in a wide variety of ways including verifying the identity of the user for each session, hiding the information such that only the authorized user knows its location, sending encrypted documents to which only the authorized user has the key, etc. Further, the authorization authority (e.g., password database) can be maintained by the database server, by a group central authority, or be some form of public key encryption. All of these options have been discussed in detail in the protection/encryption literature.

The current CERN httpd server includes the ability to restrict access based on the login name and IP address of the requesting browser. Other efforts are underway to augment the protocols and browsers with authentication mechanisms and privacy enhanced mail. However, neither of these solutions is satisfactory for supplying private information to uncontrolled PCs using existing browsers: the CERN server requires a trusted group central authentication mechanism (which does not exist for uncontrolled PCs) and the authentication mechanisms require new (not yet implemented) servers and browsers. Further, the authentication mechanisms violate the (implied) rule that a URL is a complete address of an information page. With the existing Web, URLs can be sent via e-mail, placed in files and hotlink lists, and subsequently be used by anyone to read the specified information. Obviously, with an authentication-based mechanism both the URL and the password/decryption key would have to be sent, and thus a URL would no longer be a complete specification for an information page.

### 1.1 Example Problem

My motivation for this work is my conversion of my previously paper-based university course to a Web-based learning environment. In the paper-based version, the students had access to their marks and estimated grades through a special `inquire.grade` program. This program used the fact that UNIX login names are unique across our site and that the `-rwx--x--x` file protections allowed students to execute, but not read, a file. This program has been very popular with the students, as it is more private than the typical North American "post the grades on the professor's door" technique, and it allows the students to check or review their grades at any time. Obviously, this popular feature should be retained in the Web-based version of the course.

One technique for implementing the `inquire.grade` functionality on the Web would be to use a `telnet` URL which gated the students to a UNIX system where they would login and then run the UNIX `inquire.grade` program. This would be undesirable for many reasons, not the least of which is that it would be complex and error-prone — the anti-thesis of simple, unified, Web access.

## 2. A Solution

This paper describes an elegant technique for providing private information on the existing Web. In other words, providing confidential information to authorized people using the existing URL and HTTP specifications and *all* existing clients and browsers.

### 2.1 Special URLs

The solution is based on hiding the private information at special URL addresses. Obviously, this solution will not work if intruders are able to guess or obtain URLs, and thus URLs created from publicly available information alone are inadequate. For example, information "hidden" at `http://machine/private/loginname` is not private.

However, information "hidden" at http://machine/private/loginname/password *is* private (if the password itself is not derived from public information). The drawback of these URLs, however, is that many Web browsers display the current URL in a prominent place. Thus a "shoulder surfer" could glance at the screen and obtain the URL. (Note that applications such as the WWW Sokoban game use the new forms feature and queries like `init?pname=NAME&pword=PASSWORD&email=EMAIL` instead of `init/name/password/email`, but the sholder surfer problem remains.)

The solution is for the special URLs to contain access keys which are encrypted version of the login name and password. The encryption includes randomization and checksums to reduce the chance of forgery or the use of "cracker" programs. The encrypted keys are long enough to be difficult for a shoulder surfer to remember, and furthermore they are long enough that they are not fully visible in the standard Mosaic URL display.

### 2.2 Key Generation

The second half of the solution described in this paper is the mechanism for producing these encrypted keys. Obviously, if the users were required to produce these 40-60 character encrypted keys using a published algorithm, not only would the system be incredibly inconvenient and entirely unusable, but the publication of the algorithm compromise the system's security. Thus the encrypted keys are generated using searchable `ISINDEX` pages.

The user brings up the searchable login page and enters her username and password combination in the search box. The "search" is sent, as usual, to the server with the "?" URL syntax. If the password is correct, the server returns a Web page containing a link to the private information. The URL for this link includes the encrypted access key. (See Section 2.4 for a complete example.)

The URL which includes the encrypted access key can be stored in a hotlist, sent to another user via email, or even added as a link in another Web page. The access key is verified by the server upon every retrieval request, thus allowing future versions of this mechanism to include limited duration keys or even keys that are tailored to the requestor's IP address. Furthermore, because the keys are re-verified, password changes cause the immediate disqualification of existing keys regardless of how widely the keys have been propagated.

### 2.3 Password Maintenance

The third aspect of this solution is remove password maintenance without losing the capability to propagate keys. This feature is enabled by the fact that although the encrypted access key includes the password, it does not comprise the password. Thus the server can distinguish between three levels of users:

1. *Fully-Authorized Users*: those who search the login page with the correct password.
2. *Partially-Authorized Users*:: those who request private information using a valid encrypted key.
3. *Unauthorized Users*: those who "search" the login page with incorrect password or who request private information using an invalid encrypted key.

Category 2 users are those with a valid URL given to them by a category 1 user. The server discriminates between categories 1 and 2 by augmenting the "password-search" results page. This page has two outgoing links: one to the private information pages via an encrypted URL described previously, and the other to a password-changing page via a second encrypted URL. The latter page is a searchable page, and when a new password is entered and "searched for," the server updates its password database (and thus invalidates any existing keys for that login name).

### 2.4 An Example

Consider the following example use of my Web based inquire.grade replacement: Julie, student number 123456, would like to examine her marks for assignment 3. She goes to the login page at URL `http://ursaminor.scs.carleton.ca:3000/95.207`. At this searchable login page, she enters her student number and password, i.e., `123456,guest`, and presses return. The result of this "search" is the you-have-logged-in-correctly page with two outgoing links. The first link contains the URL `<tt>http://ursaminor.scs.carleton.ca:3000/95.207/_BDNKndm4orl4pb94qplKrq84sWVG/marks.html` which refers to the page containing a summary of her marks. That page, in turn, has links to detailed explanations of her marks for each assignment including what test cases were used, which ones her programs passed, etc. Julie will probably store this URL in her Mosaic hotlist so that she can jump directly to her marks-summary page without logging in again.

The second link on the you-have-logged-in-correctly page contains a URL which refers to the password-change page. (Note that this page is not available for the `123456,guest` login on the sample server as I cannot allow the guest password to be changed.)

Julie has a good friend, Dave, whom she would like to allow access to her marks. All she has to do is to tell him the URL (`http://ursaminor.scs.carleton.ca:3000/95.207/_BDNKndm4orl4pb94qplKrq84sWVG/marks.html`) and he can browse her marks. However, if they part company acrimoniously, she can deny him access by merely changing her password using the password-change page.

## 3. An Implementation

Implementations of this privacy technique require two features in their server:

1. The ability to special case process certain searchable pages, and
2. The ability to process one word of a URL as an encrypted key.

### 3.1 The Existing Implementation

The implementation used by Section 2.4 is an HTTP server written in Smalltalk-80 at Carleton University. The server receives requests and dispatches the URL to an internal tree of `URLProcessorNode` objects. Each `URLProcessorNode` is responsible for processing one word of a URL, either by returning a document/page, or by passing the rest of the URL on to another node. Simple `URLProcessorNode` subclasses implement the default behavior of returning HTML and ASCII files as well as directory listings. Special subclasses, however, implement the "searching" and encrypted key verification.

The `PasswordProcessorNode` checks the search request against the login name-password database. If the password is correct, the node builds and returns an HTML page with the encrypted keys. If the password is not correct, the node builds and returns an error-message HTML page.

The `KeyVerifierNode` verifies an encrypted key. If the key is valid when checked against the password database, the node passes the remainder of the URL on to the remaining nodes as specified by the URL. If the key is invalid, the node returns nothing (which is interpreted as a server error by WWW browsers). The result is that encrypted keys in the URL are treated as go/no-go filters and an arbitrary hierarchical structure can be placed "underneath" the key.

### 3.2 A Potential Implementation

Another possible implementation would use existing servers which call-out to separate searching programs. The password "search" program would create a subdirectory named with the encrypted key. This would allow the standard file/directory server to process the encrypted key as if it were a normal URL component. Naturally, the directory containing all the encrypted key subdirectories would not itself be browsable.

The disadvantage of this approach is that the password information for each user would be duplicated: once in the password database and again in the name of the encrypted key subdirectory. Duplication of this sort can result to an inconsistency where the password has changed but the old encrypted key subdirectory was not correctly renamed.

Additionally, if the encrypted key includes a randomizing element (as suggested above) there will be a problem with key clutter, i.e., multiple valid, but unused, keys due to multiple logins. Unfortunately, there is no way to search all of Web-space and thus there is no good way to garbage collect these keys. One possible mechanism would be to require the user to change the password on a regular basis — all old keys can be flushed when the password is changed.[1] Another mechanism would be to time limit all keys, although this would seriously reduce their usefulness, as they could no longer be stored in other documents or in hotlink lists.

## 4. Some Analysis

The related work on authentication protocols and privacy enhanced mail (PEM) is orthogonal to, rather than in competition with, the scheme described in this paper.

- This scheme, with encrypted keys in the URLs, protects hyperlinks by hiding them behind complex names. However, anyone who knows their names can follow them and access the data.
- Authentication protocols use publicly visible hyperlinks and provide security by verifying the browsing person's identity before allowing certain hyperlinks to be traversed.
- PEM protects documents, not hyperlinks, thus effectively verifying the final recipient of the information (the recipient is not necessarily the same person who used the browser to retrieve the document).

All protection schemes have advantages and disadvantages. One disadvantage of password-only and hidden-information protection schemes (such as the one described in this paper) is their vulnerablity to “cracking” or “guessing”. However, these schemes have worked well to protect private information for many years and there is every reason to believe that they will continue to do so. The scheme described in this paper has reasonable security in addition to its major advantages:

1. it is extremely simple,
2. it works today,
3. it works with *all* existing WWW browsers,
4. it allows the controlled release of private information via URLs containing encrypted keys.

## Acknowledgements

This work has been supported in part by the Canadian National Science and Engineering Resource Council under grant OGP-0121431, and by an Academic Equipment Grant from Sun Microsystems.

## References

This paper was written as a WWW hypertext document and all of its references are to other Web pages. Such references do not translate to the printed page. Thus the reader is directed to the on-line version of this paper at `http://ursaminor.scs.carleton.ca/Papers/www94-paper.html`.

1. Actually, even this is not true, as the old encrypted keys should be valid if the password were to be changed back to what it originally was.