

110105

20427

U.S.

Express Mail Label No. EQ 270432 330

Approved for use through 07/31/2006. OMB 0651-0039

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PROVISIONAL APPLICATION FOR PATENT COVER SHEET - Page 1 of 2

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

PTO/SB/16 (10-05)
112960 U.S. PTO
60/732025
110105

INVENTOR(S)		
Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
ARI	JUELS	Brookline, MA
BJORN MARKUS	JAKOBSSON	Bloomington, IN

Additional inventors are being named on the _____ separately numbered sheets attached hereto

TITLE OF THE INVENTION (500 characters max):

Methods and Apparatus for Processing of Client-Side Data Caches to Enable User Tracking and Security Functionality

Direct all correspondence to: CORRESPONDENCE ADDRESS

The address corresponding to Customer Number:

OR

Firm or Individual Name Ari Juels

Address 106 Addington Road, Apt. 2

City Brookline	State MA	Zip 02445
Country United States of America	Telephone 617-566-4936	Email AJUELS2@GMAIL.COM

ENCLOSED APPLICATION PARTS (check all that apply)

- Application Data Sheet. See 37 CFR 1.76
- Specification Number of Pages 14
- Drawing(s) Number of Sheets _____
- CD(s), Number of CDs _____
- Other (specify) _____

Fees Due: Filing Fee of \$200 (\$100 for small entity). If the specification and drawings exceed 100 sheets of paper, an application size fee is also due, which is \$250 (\$125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).

METHOD OF PAYMENT OF THE FILING FEE AND APPLICATION SIZE FEE FOR THIS PROVISIONAL APPLICATION FOR PATENT

- Applicant claims small entity status. See 37 CFR 1.27.
 - A check or money order is enclosed to cover the filing fee and application size fee (if applicable).
 - Payment by credit card. Form PTO-2038 is attached
 - The Director is hereby authorized to charge the filing fee and application size fee (if applicable) or credit any overpayment to Deposit
- TOTAL FEE AMOUNT (\$) 100
- Account Number: _____ A duplicative copy of this form is enclosed for fee processing.

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information is required by 37 CFR 1.51. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

PROVISIONAL APPLICATION COVER SHEET
Page 2 of 2

PTO/SB/16 (10-05)

Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.



No.



Yes, the name of the U.S. Government agency and the Government contract number are: _____

WARNING:

Petitioner/applicant is cautioned to avoid submitting personal information in documents filed in a patent application that may contribute to identity theft. Personal information such as social security numbers, bank account numbers, or credit card numbers (other than a check or credit card authorization form PTO-2038 submitted for payment purposes) is never required by the USPTO to support a petition or an application. If this type of personal information is included in documents submitted to the USPTO, petitioners/applicants should consider redacting such personal information from the documents before submitting them to the USPTO. Petitioner/applicant is advised that the record of a patent application is available to the public after publication of the application (unless a non-publication request in compliance with 37 CFR 1.213(a) is made in the application) or issuance of a patent. Furthermore, the record from an abandoned application may also be available to the public if the application is referenced in a published application or an issued patent (see 37 CFR 1.14). Checks and credit card authorization forms PTO-2038 submitted for payment purposes are not retained in the application file and therefore are not publicly available.

SIGNATURE _____



Date 1 November 2005

TYPED or PRINTED NAME Ari Juels

REGISTRATION NO. _____
(if appropriate)

TELEPHONE 617-566-4936

Docket Number: _____

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (*i.e.*, GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

1 November 2005

Methods and Apparatus for Processing of Client-Side Data Caches to Enable User Tracking and Security Functionality

Markus Jakobsson¹ and Ari Juels²

¹ 5631 East Kerr Creek Rd.
Bloomington, IN 47408
markus@indiana.edu

² 106 Addington Road, Apt. 2
Brookline, MA 02445
ajuels2@gmail.com

Abstract. Like conventional cookies, *cache-cookies* are data objects that servers store in Web browsers. Cache-cookies, however, are essentially unintentional byproducts of protocol design for browser caches. Web browsers do not permit users to determine access-control policies for cache-cookies as they do for conventional cookies. Consequently, cache-cookies permit Web servers to perform highly invasive tracking of users on the Internet. The security community has identified cache-cookies as a threat to user privacy and security, and proposed browser modifications to eliminate them.

In this paper, we argue that cache-cookies have a positive face. We propose privacy-sensitive protocols that use cache-cookies for identification and authentication of users. To protect their privacy, many users today block conventional cookies in their browsers. The cache-cookie tools we propose can help restore lost usability and convenience to such users while maintaining good standards for privacy. As we show, our techniques can also help combat online security threats as phishing and pharming.

Keywords: cache-cookies, personalization, malware, pharming, phishing, privacy, Web browser

1 Introduction

A *cookie* is a piece of information stored in a browser state. Some cookies are intended only to be possible to be read by the parties that wrote them; these are referred to as *first-party* cookies. Other cookies can be read by separate entities; these are referred to as *third-party* cookies. Today's browsers allow users to select whether to accept first-party and third-party cookies, giving the appearance of allowing the user to opt out from the planting of cookies within their browsers. As we will describe, however, there are alternatives to cookies that do not easily allow users to opt out; these are referred to as *cache-cookies* since they use browser caches as repositories for cookie data.

In contrast to the common wisdom within the security community, we argue that cache-cookies are not purely a negative aspect of computing. Rather, we believe that they offer many useful features, some of which have a notably positive impact on user security. In particular, we describe novel applications of cache-cookies to user identification and authentication. We describe how such goals can be achieved in a manner respectful of user privacy.

A main contribution of the paper is a general read/write framework for cache-cookies, exhibiting some of the interesting resource constraints of devices like RFID tokens and on old-fashioned punchcards. Within this framework, we propose special data structures on cache-cookies that preserve their confidentiality against snooping servers. We also propose audit techniques to detect abusive usage of cookies, and a technique for creating cache-cookie identifiers whose privacy is linked to that of a Web server's private SSL key.

Another contribution in this paper is a new type of cache-cookie based on Temporary Internet files (TIFs). As we explain, these cache-cookies have the innate privacy features of conventional first-party cookies.

2 Cache-Cookie Management

Before proposing new applications of cache-cookies, we first describe some new ideas for their management. Previous work [4, 12] has described rudimentary planting and retrieval of cache-cookies essentially in the manner of conventional cookies. The aim has been to facilitate release of all cookies to the entity that has set them, or to some other, delegated entity (since cache-cookies do not contain the domain restrictions of conventional cookies). Again, this previous work has viewed cache-cookies strictly as a potential privacy violation.

We explore some refinements whereby a server can use cache-cookies to create a general read/write memory structure in a user's browser. As we demonstrate experimentally in section ??, it is possible for a server not merely to detect the presence of a particular cache-cookie, but to test quickly for the presence of any in a list of hundreds of cache-cookies. More interesting still, a server can unobtrusively mine cache-cookies in an *interactive* manner, meaning that it can refine its search on the fly, using preliminary information to guide its detection of additional cache-cookies. In consequence, the read/write memory structure useable by a server can be very large – so large as to resist brute-force search by an adversary. We show how exploit this characteristic in section 3 to support a range of privacy and security goals.

Thanks to our general view of cache-cookies as a storage mechanism, we are able to propose fruitful uses for a new type of cache-cookies based on what are called *cache files*, namely general-purpose data files downloaded by browsers.

We first present our framework for using cache-cookies to simulate a general read/write memory structure.

2.1 General read/write framework

Consider a particular browser cache, such as the history cache that contains recently visited URLs. A server can, of course, plant any of wide variety of cookies in this cache by redirecting the user to URLs within its domain space (or externally, for that matter). For example, a server operating the domain `www.ravenwhite.com` can redirect a browser to a URL of the form “`www.ravenwhite.com?Z`” for any desired value of Z , thereby inserting “`www.ravenwhite.com?Z`” into the history cache.

In this case, for any $Z \in \{0, 1\}^{l+1}$, it is natural to index the URL “`www.ravenwhite.com?Z`” by the $(l + 1)$ -bit string Z . In general, we shall index cache cookies by a set $R = \{0, 1\}^{l+1}$. In practice, of course, we can make l fairly large – on the order of several hundred or thousand bits – quite efficiently. We let the predicate $P_{i,t}[r]$ denote whether the URL corresponding to a given index $r \in R$ is present in the cache of user i at time t . If so, $P_{i,t}[R] = 0$; otherwise $P_{i,t}[R] = 1$.

Of course, a server interacting with user i can change $P_{i,t}[R]$ from 0 to 1, but not the reverse. We can, however, achieve a more flexible read/write structure.

Let $S = \{0, 1\}^l$. Let us define a predicate $Q_{i,t}[s]$ over S , for any $s \in S$. This predicate can assume a bit value, i.e., $Q_{i,t}[s] \in \{0, 1\}$; otherwise, it is “blank,” and we write $Q_{i,t}[s] = \phi$, or it is “erased,” and we write $Q_{i,t}[s] = \nu$. Let \parallel denote string concatenation. Let us define $Q_{i,t}$ as follows. Let $r_0 = P_{i,t}[s \parallel '0']$ and let $r_1 = P_{i,t}[s \parallel '1']$. If $r_0 = r_1 = 0$, then $Q_{i,t}[s] = \phi$; if $r_0 = r_1 = 1$, then $Q_{i,t}[s] = \nu$. Otherwise, $Q_{i,t}[s] = b$, where $r_b = 1$. Henceforth, for convenience, we drop the subscript t unless context requires it; likewise, we sometimes drop the subscript i for clarity.

This definition yields a simple write-once structure M with erasure for cache cookies over the set S . When $Q_i[s] = \phi$, a server interacting with user i can write an arbitrary bit value b to $Q_i[s]$: It merely has to set $P_{i,t}[s \parallel b] = 1$. The server can erase a stored bit b in $Q_i[s]$, by setting $P_{i,t}[s \parallel 1 - b] = 1$.

Within the structure M , we can define an m -bit memory structure M' capable of being written c times. We let M' consist of a sequence of $n = cm$ bits in M . Once the first block of m bits in M' has been written, the server re-writes M' by erasing this first block and writing to the second block; proceeding in the obvious way, it can write c times to M' . To read M' , the server performs a binary search, testing the leading bit of the memory blocks in M' until it locates the current write boundary; thus a read requires at most $\lceil \log c \rceil$ queries.

When l is sufficiently large – in practice, say, when cache cookies are 80 bits long – the memory structure S is large enough to render brute-force search by a cache-sniffing server impractical. Suppose, for example, that a server plants a secret, k -bit string $x = x_0x_1 \dots x_k$ into a random location in memory in the browser of user i ; that is, it selects $s \in_U 2^l - k - 1$, and sets $Q_i[s + i] = x_i$ for $1 \leq i \leq k$. It is infeasible for a second server interacting with user i to learn x – or even to detect its presence.

We can employ hidden data of this kind as a way to protect the privacy of cache cookies.

Variant memory structures: There are other, more query-efficient encoding schemes for the structures M and M' . For example, we may realize an m -bit block of data in M as follows. Let $\{P_i[s], P_i[s+1], \dots, P_i[s+c]\}$ represent the memory block in question. We pre-pend a leading predicate $P_i[s-1]$. The value $P_i[s-1] = '0'$ indicates that the block is active: It has not yet been erased. To encode the block, a server may change any predicate may be changed to a '1.' $P_i[s-1] = '1'$ indicates that the block has been erased. A drawback to this approach is that erasure does not truly efface information: The information in an "erased" block remains readable. Full effacement is sometimes desirable, as in the rolling pseudonym scheme we shall present.

On the other hand, in TIF-based cache-cookies, as we now explain, use of leading predicates for memory blocks is essential.

2.2 TIF-based cache-cookies

A drawback to any cache-cookie scheme based on browser history is the short lifetime of stored information. Firefox and Internet Explorer retain history information for nine days by default, a parameter that most users are unlikely to modify.

A different type of browser-cache information has a longer lifetime. *Temporary Internet files* (TIFs) are files containing information embedded in Web pages. Browsers cache these files to support faster display when a user re-visits a Web page. Temporary Internet Files have no associated expiration; they persist indefinitely. (Browsers do, however, cap the amount of disk space devoted to these files, and delete them so as to maintain this cap.)

In order to place a TIF X in a browser cache, a server can serve content that causes the downloading of X . A server can verify whether or not a given browser contains a given TIF X in its cache by displaying a page containing X . If X is not present in the cache, then the browser will request it; otherwise, the browser will not request X , but instead retrieve its local copy. In order not to change the state of a cache-cookie for whose presence it is testing, a server must in the latter case withhold the request for X . While this may trigger a "401" error, manipulation of cache files can take place in a hidden window, unperceived by the user.

Cache-cookies based on TIFs do in fact carry access-control restrictions, a useful privacy feature. Browsers reference TIFs by means of URLs. When a browser requests TIF, therefore, it refers to the associated domain, not to the server that is displaying the page containing X . In this regard, TIF-based cache-cookies of this type are like first-party cookies: Only the site in control of the URL corresponding to a TIF can detect the presence of that TIF in a browser cache.

3 Schemes for User Identification

Felten and Schneider note that cache cookies can serve the same function as ordinary cookies, permitting Web sites to identify users [4]. Cookies, however,

are by browser policy only accessible by the domain that has set them. As we have noted, cache cookies are accessible by *any* domain. We propose some cache-cookie constructions for identification that support access-control policies very like those for ordinary cookies, that is, that restrict access to the domain that set them.

Like ordinary cookies, cache cookies can serve as a tool for recording data about users or simply for identifying. Of course, a server could simply write the user ID to a particular location in a memory structure M . This ID, however, would ultimately be readable by any third party. (Any user can record which locations in M that the server accesses, and thereby determine the location to which the server writes ID.)

We therefore consider two ways to structure user identifiers using cache cookies in a privacy enhancing way, a *tree-based* scheme and a *rolling pseudonym* scheme. The tree-based scheme restricts cache-cookie access to a site that possesses an underlying secret key. The rolling-pseudonym approach decouples appearances of user identifiers to disrupt third-party tracking attempts.

The problem of privacy protection in cache-cookie identifiers bears an interesting similarity to that of privacy protection in radio-frequency identification (RFID) systems. In both cases, the identifying system is highly resource-constrained: RFID tags have little memory and often little computational power, while cache-cookie systems cannot invoke computational resources on the client side, but must rely on memory accesses alone. Similarly, in both systems, the aim is to permit a server to identify a device (a tag or browser) without the device revealing a static identifier to potential privacy-compromising adversaries. Our tree-based scheme bears some similarity to the RFID-identification system of Molnar and Wagner [9, 8], while our pseudonym system is somewhat like that of Juels [6].

We also briefly consider how *secret* cache-cookies can aid in *authenticating* users that a server has already identified.

3.1 Identifier trees

The idea here is to create a tree, called an *identifier tree*, whose nodes correspond to secret cache-cookies. Each user is associated with a distinct leaf in the tree; planted in the browser of the user are the d secret cache-cookies along the path from the root to this leaf. To identify a user, a server interactively queries the user's browser so as to perform a depth-first search of the identifier tree. The aim of the server is to identify the user's unique leaf. This search is feasible only for a server that knows the cache cookies associated with nodes in the tree.

As an example, consider a binary tree T of depth d . Let ρ denote the root node. For a given node n within the tree, let $n \parallel '0'$ denote the left child, and $n \parallel '1'$, the right child. Thus, for every distinct bitstring $B = b_0b_1 \dots b_k$ of length k , there is a unique corresponding node n_B at depth k . The leaves of T are the set of nodes n_B for $B \in \{0, 1\}^d$.

Let $x_B \in S$ be a secret key associated with node n_B . The set of secret keys $X = \{x_B\}$ might be selected uniformly at random, or might be generated pseudorandomly based on a short, secret, master key.

Suppose that user i is associated with leaf $n_{B^{(i)}}$, where $B^{(i)} = b_1^{(i)} b_2^{(i)} \dots b_d^{(i)}$ is a unique d -bit identifier. When the user visits the server, the server queries the user's browser to determine whether it contains x_0 or x_1 in its cache. On learning that $x_{b_1^{(i)}}$ is present, the server then queries the user's browser to determine whether it contains $x_{b_1^{(i)} \parallel 0}$ or $x_{b_1^{(i)} \parallel 1}$ in its cache, etc. Ultimately, the server discovers the full path of secret keys to $n_{B^{(i)}}$.

Of course, this scheme can be extended to identifier trees with any desired degree. For example, consider an identifier tree with degree $\delta = 2^k$, where d is a multiple of k , and the number of leaves is $L = 2^d$. The depth of such a tree, and consequently the number of stored secret cache-cookies, is d/k ; so too is the number of rounds of queries required for a depth-first search, assuming that each communication round contains the δ concurrent queries associated with the currently explored depth. Therefore, higher-degree trees induce lower storage requirements and round-complexity. On the other hand, higher-degree trees induce larger numbers of queries. Assuming δ (concurrent) queries per level of the tree, the total number of queries is $\delta d/k = 2^k d/k$. It is useful to observe that a tree of degree $\delta = 4$ is strictly preferable to a binary tree, i.e., degree $\delta = 2$: The total number of queries for both types of tree is identical, while the round-complexity for a tree with $\delta = 4$ is half that of a tree with $\delta = 2$.

The problem of collusion: Without knowledge of the set X of secret keys that composes the identifier tree, a third party cannot extract the IDs of participating users.

A collusion among users, however, does pose some threat to privacy. Users can pool the secret keys along their respective paths in order to obtain partial information about the secret keys associated with T . Given a tree of sufficient depth, even a large, cheating coalition of this kind would gain complete or near-complete knowledge of only the top layers of the tree. This would be sufficient to identify users in some measure – in particular, to fingerprint them according to the positions of their respective paths in the top layers of T . We note, however, that servers can already fingerprint users to some extent based on their browser types, IP addresses, and so forth.

3.2 Rolling-pseudonym scheme

Another measure for enforcing the privacy of pseudonyms is to change them on a regular basis. For example, a server might refresh pseudonyms on a weekly basis. Let $f : \{0, 1\}^* \rightarrow \{0, 1\}^d$ represent a cryptographic one-way function, e.g., a hash function. The server computes $f(i \parallel j)$ as the pseudonym in time period j of user i . The server keeps track of the current pseudonym of each user. When the user logs in during time period j , the server erases any pseudonym still active in the browser, and replaces it with the current one. (The server could use a cipher

here in lieu of a one-way function, and decrypt i and j , but this would incur more communication overhead.)

To manage the memory of pseudonyms, the server can designate a particular memory location in M for the storage of pseudonyms associated with time period j . When a user logs in, the server must, working backward, locate the last active pseudonym. To ensure uniform memory management, i.e., to ensure that erasure patterns are not uniquely identifying, the server must erase all memory locations corresponding to previous weeks – back to the starting point of the browser history (or some estimated upper limit that is uniform across users).

Of course, with this scheme, an adversary can link appearances of a user's browser within a given time period. Time periods, however, can be quite short – as short as a few days, perhaps.

Remarks: Of course, the secret key for these two identifier schemes can be delegated to a site other than the one that sets the cache cookies. This possibility can be useful for maintaining seamless user identification across associated sites. It is also subject to abuse if not carefully monitored.

Cache cookies do not automatically support timestamps, of course, but a server can cause an identifier to expire simply by erasing it. Servers can even write their cache-cookie use policies as data appended to identifiers.

3.3 Denial-of-service attacks

In our rolling-pseudonym scheme, pseudonyms must reside in regions in M that an attacker can easily determine by observing the behavior of the server. Thus a malicious server can erase pseudonyms. This is a nuisance, of course, but it is no worse than the result of a flushed cache of conventional cookies. Indeed, a server can detect when an attack of this kind has occurred, since it will cause invalid erasures. To mitigate the effects of this type of attack, a server can keep secret the blocks of memory to be used future pseudonyms.

An attacker can mount a denial-of-service attack against the identifier-tree scheme by adding spurious paths to a user cache. If the attacker inserts a path that does not terminate at a leaf, the server can detect this corruption of the browser data. If the attacker inserts a path that does terminate at a leaf, the result will be a spurious identifier. To prevent this attack, the server can include a cryptographic checksum on each identifier associated with the browser (embedded in a secret, user-specific location in M). The problem, of course, is that an attacker that has harvested a valid identifier can also embed its associated checksum. Instead, therefore, a checksum must be constructed in a browser-specific manner. For example, the checksum can be computed over *all* valid identifiers in the browser. Provided that the server always refreshes all identifiers simultaneously, this checksum will not go stale as a result of a subset of identifiers dropping from the cache. The checksum can serve to weed out spurious identifiers.¹

¹ Note that an attacker that embeds spurious identifiers in caches is providing a forensic trail, since the attack must either have registered or compromised the associated accounts.

Another form of denial-of-service attack is for the attacker to write a large number of spurious paths to a cache in order to prolong the time required for the server to perform its depth-first search. It is unclear how to defend against this type of attack, although a server should be able to detect it, since the cache will either contain paths that do not terminate in leaves, or will contain implausibly many leaves.

To prevent long-lasting effects of such corruption in the identifier-tree scheme, a server can refresh trees on a periodic basis.

To avoid the need for storing identifiers in a lookup table, of course a server can compute an identifier as a ciphertext on either a timestamp or counter along with the user's identity (user name). In the presence of long user names, however, this approach is not always space-efficient.

3.4 Secret cache-cookies

We have described two schemes for user identification using cache cookies. In cases where users are identified by other means, like conventional cookies, cache cookies can still play a useful role in user login. Rather than supporting user identification, they can support user *authentication*, that is, confirmation of user identity. While cache-cookies have privacy vulnerabilities that ordinary cookies do not, they also have some privacy-enhancing strengths, like resistance to pharming, as we shall explain.

Some vendors of anti-phishing platforms, such as PassMark Security [10], already employ conventional cookies and other sharable objects as authenticators to supplement passwords. Because conventional cookies (and similar sharable objects) are fully accessible by the domain that set them, they are vulnerable to *pharming*. A pharming attack creates an environment in which a browser directed to the Web server legitimately associated with a particular domain instead connects to a spoofed site. A pharmer can then harvest the cookies (first-party or third-party) associated with the attacked domain. Even the use of SSL offers only modest protection against such harvesting of cookies. It is generally difficult for a pharmer to obtain the private key corresponding to a legitimate SSL certificate for a domain under attack. But a pharmer attacking an SSL-enabled site can disable SSL in a simulated site (and perhaps display a deceptive, if incorrect lock icon in the browser window). Alternatively, the phisher can use an incorrect certificate and simply rely on the tendency of users to disregard browser warnings about certificate mismatches. Pharming attacks thus undermine the use of cookies as supplementary authenticators.

Secret cache cookies are resistant to pharming. A secret cache-cookie, very simply, is a secret key y_i specific to user i that is stored in a secret, user-specific location u_i in M . Once the user identifies herself and perhaps authenticates with other means, e.g., a password or hardware token, a server can check for the presence of the user-specific secret cache-cookie y_i as a secondary authenticator.²

² Even though the location u_i is itself secret, it is important to use a secret key y_i , rather than to plant a single indicator bit at u_i : If a server merely checks for the existence of a bit at u_i , a saavy attacker can simulate the presence of such a bit.

of course, many variants are possible. For example, a server can periodically update both y_i and u_i to protect against transient data compromises.

Secret cache-cookies are resistant to basic pharming attacks because they do not rely on domain names for server authentication. In order to access the key y_i , a server must know the secret location u_i associated with a user. A pharmer cannot learn this information without compromising a victim's machine (at which point the victim is subject to a broad range of attacks), or compromising the link between a server and client, which secure sites generally harden with SSL.

This use of secret cache-cookies is complementary to the use of cookies. It has the advantage of functioning even for users that employ cookie-cutters (but not cache-flushing tools, of course). As with conventional cookies, and mechanisms like IP-tracing, there is a drawback to secret cache-cookies: A user who changes browsers loses the benefit of the secondary authenticator.

4 Audit Mechanisms

As we have explained, cache-cookies can serve as a mechanism for abuse of user privacy; browser histories in particular are accessible to any querying server, as is timing information on cached files. (In the absence of timing attacks, cache files, as we have explained, contain access restrictions much like those of first-party cookies.) In this section, we enumerate a few possible strategies for detecting and preventing misuse of cache-cookies.

As noted above, there are many possible abuses of cache-cookies stored on browsers in the natural course of their operation. For example, a server that mines data from a browser history cache can profile a user according to which political sites she has visited. Here we focus instead on the use and abuse of the general read/write memory structures for cache-cookies that we have described here, and focus particularly on user identifiers. A server can abuse these structures in two ways: (1) By tracking users with cache-cookies when users do not wish for such tracking to occur and (2) By making cache-cookie information available to third parties against the wishes or simply in the absence of knowledge of users.

Briefly, then, we have proposed ways in which cache-cookies can be deployed with support for user privacy. The question now is how can users be sure that servers are deploying cache-cookies appropriately?

It should be noted that in some measure, cache-cookie reads and writes are transparent. This is to say that with the appropriate software, e.g., a browser plug-in, a client can detect which cache-cookies a server has accessed. By detecting which portions of its caches are "touched," for example, a client can determine the pattern of browser probing. (Of course, these remarks do not apply to timing-based cache cookies. Thankfully such cookies are relatively difficult to exploit.) Bortz, Jackson, Boneh, and Mitchell [], for instance, propose ways in which modified browsers can provide users with the ability to unify their privacy policies to extend beyond conventional cookies to cache-cookies. Of course, similar tools can serve instead to detect and track server-side behavior. Auditing,

rather than universal browser changes, can also help enforce user privacy. For example, if identifier-tree leaves are tagged with the domain of the server that sets them, client-side software can detect third-party sniffing of identifiers.

On the server side, a number of standard tools can support privacy auditing, among them:

1. **Software inspection:** An organization with a published privacy policy can rely on audits of its servers to ensure the use of compliant software.
2. **Trusted computing modules (TPMs):** A server containing a TPM can provide assurance to an external entity that it has a particular software configuration. This configuration can include software that protects the privacy of user databases as in, e.g., [11].
3. **Platform for Privacy Preferences (P3P):** P3P [1] is a standard by which a browser and server can negotiate the exchange of information based on their respective privacy policies. Extensions to P3P can naturally underpin server-side determination of how and whether or not to deploy cache-cookies. P3P provides no external means of enforcement, however, i.e., it presumes compliance by a participating server.
4. **Mixnets and proxies:** Privacy-protections on the server side can be distributed among multiple servers or entities. For example, an authentication service could identify users by means of third-party cookies simulated via cache cookies. This service could translate user identifiers into pseudonyms on behalf of relying sites.

On the client side, various forms of direct auditing of server behavior are possible, as we have explained. Here, however, we propose two new ideas. First, we discuss the idea of lightweight JavaScript-based computing on the client to restrict server access to information. We also propose a special mechanism specific to identifier trees, and called *proprietary identifier-trees*.

4.1 Client-side computing

As discussed in [12], JavaScript code in Mozilla and Internet Explorer can test for the presence of cache cookies in browser history caches and set boolean variables accordingly. We observe, therefore, that a server can transmit code to a browser that applies any desired function f to a set of cache-cookies. Additionally, the output of the function can be written to the cache memory structure M , permitting its retrieval by the server. The read and write operations implemented by f can, of course, be applied to secret locations in M . These novel observations lead to two intriguing possibilities, among many others:

Privacy-preserving data mining: A server can transmit a function f that harvests cache-cookie information (or, for that matter, general cache information) and filters out information that it is inappropriate for the server to access. For example, suppose that a user has agreed to release information about whether she has visited a news site in the past month, but not which news site

she has visited. The function f can compute a single bit indicating whether the user has visited any of a fixed list of such sites. If so, the function can write a '1' bit to a location in M ; otherwise, it writes a '0'. To protect this one-bit piece of information against sniffing by other parties other than the original, querying server, f can write to a secret session-specific or user-specific location in M . Of course, a server can transmit any function it likes, including functions that infringe privacy. This mechanism of client-side computation, however, provides more transparency and accountability than server-side filters can.

Zero-footprint crypto tokens: The function f can compute a cryptographic function useful for user authentication. For example, a server can write a secret key y_i to a secret location u_i in cache memory M that is specific to user i . To authenticate user i , the function f can compute the cryptographic hash of y_i with some nonce or timestamp, and write the result to a session-specific, secret location in M . Of course, a drawback to this approach is that the code implementing f necessarily reveals the secret location u_i , making it vulnerable to extraction by an adversary. (As mentioned above, cache-cookies based on cache files are domain-specific, and therefore resist such attacks; we have not yet determined, however, whether Javascript can operate effectively on this variety of cache-cookie.)

One way to address the problem of leakage of u_i is to change u_i with every successful login attempt. This narrows the window of time available to an adversary to extract u_i . Alternatively, the zero-footprint crypto token can serve as a second factor; that is, f can be uploaded to a client only upon successful authentication with a separate mechanism.

4.2 Proprietary identifier-trees

Controlling access to universally accessible cache-cookies, those based on browser history in particular, is fundamentally a digital-rights management problem. To ensure against inappropriate sharing of cache-cookie data, we can appeal to the "digital signet" concept of Dwork, Lotspiech, and Naor [3]. Briefly stated, their idea is to protect against one entity, Alice, sharing proprietary information with a second entity, Bob, by making a key to the information transmissible in one of two ways: (1) As a short secret that includes private information that Alice would be reluctant to share with Bob, e.g., her credit-card number, or (2) As a secret so long that it is cumbersome to transmit.

Our idea is to link the underlying secret for the tree for our identifier-tree scheme in section 3.1 to a secret belonging to the server, namely the private key for its SSL certificate. We call the resulting structure a *proprietary identifier-tree*. To share a proprietary identifier-tree with another party, a server must either transmit a large portion of the tree (at a minimum, the path for every user), or compromise its SSL certificate.

For meaningful use of proprietary identifier trees, we require an additional property not present in the Dwork et al. scheme (as their scheme is purely secret-key based). We would like a third party to be able to *verify* the linkage between

the identifier tree and the SSL certificate. In this sense, we draw on the ideas of Camenisch and Lysyanskaya [2] and Jakobsson, Juels, and Nguyen [5]. They show how sets of digital credentials may be created such that the secret key for one credential leaks the private key associated with another, and demonstrate how one party can *prove* such linkage of credentials.

For the particular case of identifier trees, we can employ a *verifiable unpredictable function* (VUF) [7] as proposed by Micali, Rabin, and Vadhan. This is a function f with a corresponding secret s such that $f_s(m)$ is efficiently computable for any message m ; additionally, knowledge of s permits construction of an efficiently checkable proof of the correctness of $f_s(m)$. Knowledge of the value of f_s at any number of points does not permit prediction of f_s on a new point. Micali et al. demonstrate an RSA-based construction. Briefly stated, for RSA modulus N (with some short, additional public information), the value $f_s(m) = r^{1/p_m} \bmod N$, where p_m is a unique prime corresponding to message m according to a public mapping; no additional information is needed for verification.

Given this use of a VUF, a user can verify that the path corresponding to its identifier is consistent with an identifier-tree linked to the SSL certificate of a given server. If many users – or auditors – perform such verification, then it is possible to achieve a high level of assurance of server compliance with the scheme.

Of course, it is feasible for a server to circumvent disclosure of its private SSL key by transmitting portions of its identifier tree. For example, with a tree of depth 80, 1024-bit digital signatures, and a base of 1,000,000 users, the size of the associated tree data would be slightly more than 10GB. Thus, even without sharing its private key, a server can plausibly share its set of user identifiers. The more important aspect of our scheme is the fact that, without sharing its private key, a server must share *updates* to the identifier tree when new users join. The resulting requirements for data coordination represent a substantial technical encumbrance and disincentive in our view. Additionally, the ongoing relationship and coordination required for such updates would expose more evidence of collusion to potential auditors.

Remark: VUFs have a property that is not essential for proprietary identifier-trees: The function f is deterministic, and thus the value $f_s(m)$ is unique. Without this property, a simpler scheme is possible. A random value r is assigned to the root node. Now, for any a child in position i , the associated value is computed as a digital signature on the value of the parent concatenated with i (suitably formatted). Digital signatures take the form of RSA signatures, with the SSL certificate defining the public key. Provided that the signature scheme carries the right security properties, e.g., signatures are existentially unforgeable under chosen-message attacks, an adversary cannot guess the value of unrevealed secrets in the identifier tree. That is, the ability to construct any unrevealed portion of the identifier tree implies knowledge of the private key for the SSL certificate. This signature-based scheme, however, lacks the crisp security properties of one based on VUFs. For example, the signer, that is, the creator of the

identifier tree, can embed side-information in signatures. Thus, careful construction and analysis – beyond the scope of this paper – are warranted for such a scheme.

References

1. Platform for privacy preferences (P3P) project. World Wide Web Consortium (W3C). Referenced 2005 at <http://www.w3.org/p3p>.
2. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *Eurocrypt 01*, pages 93–118. Springer-Verlag, 2001. LNCS no. 2045.
3. C. Dwork, J.B. Latspiech, and M. Naor. Digital signets: Self-enforcing protection of digital information (preliminary version). In *ACM Symposium on the Theory of Computing (STOC)*, pages 489–498, 1996.
4. E. W. Felten and M. A. Schneider. Timing attacks on Web privacy. In *ACM Conference on Computer and Communications Security*, pages 25–32. ACM Press, 2000. Referenced 2005 at <http://www.cs.princeton.edu/sip/pub/webtiming.pdf>.
5. M. Jakobsson, A. Juels, and P. Nguyen. Proprietary certificates. In B. Preneel, editor, *RSA Conference Cryptographers Track (CT-RSA)*, pages 164–181. Springer-Verlag, 2002. LNCS no. 2271.
6. A. Juels. Minimalist cryptography for low-cost RFID tags. In C. Blundo and S. Cimato, editors, *The Fourth International Conference on Security in Communication Networks – SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 149–164. Springer-Verlag, 2004.
7. S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *Proceedings of the 40th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 120–130, 1999.
8. D. Molnar, A. Soppera, and D. Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography – SAC 2005*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.
9. D. Molnar and D. Wagner. Privacy and security in library RFID : Issues, practices, and architectures. In B. Pfitzmann and P. McDaniel, editors, *ACM Conference on Communications and Computer Security*, pages 210 – 219. ACM Press, 2004.
10. PassMark Security. Company Web site, 2005. Referenced 2005 at <http://www.passmarksecurity.com/>.
11. S.W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Sys. J.*, 40(3):685–695, 2001.
12. SecuriTeam Whitepaper. Timing attacks on Web privacy (paper and specific issue), 20 February 2002. Information provided by A. Clover. Referenced 2005 at <http://www.securiteam.com/securityreviews/5GP020A6LG.html>.

A Sandwich cookies

Our cache-cookie-based identifier schemes have a useful characteristic. It permits the co-existence of multiple identifiers; we use the term *sandwich cookies* to refer to cache-cookie identifiers for a single Web site that are resident in the same browser. In our tree scheme, provided that the server explores edges for

both children at every node, it can detect the presence of multiple paths in a cache, and thus the presence of multiple identifiers. Likewise, in our rolling pseudonym scheme, a server can reserve different portions of memory for different pseudonyms, and search these locations to check for the existence of multiple pseudonyms.

The ability of a server to detect the presence of multiple identifiers can be very convenient. Multiple users often access their Web accounts on a single machine through the same browser. This sharing often causes one user to "bump" the cookie of another browser. Sandwich cookies permit a server easily to manage multiple accounts in the same browser. For example, a server that detects identifiers for both Alice and Bob, can cause a browser pop up a script offering easy selection between the two accounts.

Of course, conventional cookies can be transformed into sandwich cookies; a cookie need merely store multiple identifiers instead of a single one, or expand a single identifier on the server end into multiple identifiers. With cache-cookies, however, it is possible to manipulate identifiers independently. In other words, sandwich cookies based on cache-cookies do not require any linking of separate user accounts.

Sandwich cookies can enable some simple enhancements to the user experience in Web browsing. Many Web sites at present maintain persistent user sessions by means of cookies: A user remains logged into the site while her browser contains an appropriate cookie. When a session is in force for one user on a particular Web site, another user, by logging into the same site, "bumps" the first. This second user's cookie takes the place of that of the first user, whom the Web site no longer recognizes.

Sandwich cookies permit an enhancement to the experience of users who share a browser to access the same Web site. Rather than allowing one user to bump another, a Web site can maintain a set of active user logins, and allow a user to choose her account identifier from a menu. (Various policies are possible for determining when an account identifier appears in the menu and when users must re-authenticate.)

PATENT APPLICATION SERIAL NO _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

11/03/2005 HDEMESS1 00000074 60732025
01 FC:2005 100.00 DP

PTO-1556
(5/87)