Cookies On-the-Move: Managing Cookies on a Smart Card

Alvin T.S. Chan
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, SAR of China
cstschan@comp.polyu.edu.hk

ABSTRACT
Despite the widespread use and adoption of cookies as the basis for web applications to keep state information, cookies present some design issues that are yet to be fully addressed. The fact that cookies are stored on client-side’s memory means that they are tightly coupled to the machine that is interacting with the web server. Yet often, these cookies are initiated by web applications to identify user’s preferences and identifications. As the user moves across different machines to access the same site, the information previously recorded is lost and the web application has no way of restoring the state, unless the user revisits the same client machine, where the original cookies were set. This paper presents a novel solution to address the need for cookies to be “mobile” by leveraging on smart card to manage cookies, with the benefit of mobility in a pocket. We describe the design and implementation of the CookiesCard framework that uses smart card as a secure and mobile storage media to manage personalized cookies. The article presents the development of the CookiesCard proxy that directly interacts with the smart card to provide cookies management, while acting as an intermediary between the client browser and a web server.

Keywords
Smart card, Cookies, Web, mobile

1. INTRODUCTION
The explosive growth of the Internet has infiltrated and changed our way of living. As we accustom to the convenience of information access, increasingly we are using many services offered over the Internet, such as E-Banking, E-Shopping, E-Auction, etc. As web applications continue to advance in terms of service provisions, these applications often exercise complex interactions between the client and web server which require state retention across multiple request-reply cycles. Since its inception, HTTP [1,2] is the underlying transfer protocol for the web which offers simple client/server interaction that does not support state retention. In other words, the state information is not retained across the request-reply cycles of HTTP. As a result, work around this drawback has been proposed to resolve the condition of applying HTTP for dynamic web applications. Among the solutions, cookies[3] represent an attractive solution to provide state retention by allowing the web server to “plant” state information directly on a visiting client machine.

Netscape first introduced cookie in 1994, which aimed to solve the state retention problem by introducing new headers to be carried over HTTP. The current cookie specification can be found in RFC 2965. Cookies are small bits of textual information that a web site sends to a web browser. This information is stored within the client machine and return unchanged in subsequent visit to the same web site. The ability to store cookies on the user’s machine enables the web server to keep track of state information while interacting with the browser across a session. If necessary, the cookies can be kept beyond a session such that when users power off their machines the state information is retained and can be used again the next time the user visits the site that first created it. When properly designed, cookies can remember the user’s activities on the web site and provide additional functionality to the end user. One of the examples is to keep user information in a cookie and customize the pages in subsequent visits. For example, Microsoft’s Hotmail web site uses this approach to remember user’s preferences of display and hence can achieve customization without using database server, thus off loading the work load of servers on the web site.

Despite the widespread use and adoption of cookies as the basis for web applications to keep state information, cookies present some design issues that are yet to be fully addressed[3,4]. The fact that cookies are stored on client-side’s memory means that they are tightly coupled to the machine that is interacting with the web server. Yet often, these cookies are initiated by web applications to identify user’s preferences and identifications. As the user moves across different machines to access the same site, the information previously recorded is lost and the web application has no way of restoring the state, unless the user revisits the same client machine, where the original cookies were set. Storing cookies on a machine also presents security concerns, where cookies information stored as plain text files can easily be accessed, viewed, modified and deleted. The situation is made worst in a single user PC system where the browser is not equipped with necessary access control to distinguish between different users of the computer.

This paper presents a novel solution to address the need for cookies to be “mobile” by leveraging on smart card to manage cookies, with the benefit of mobility in a pocket. We describe the design and implementation of the CookiesCard framework that uses smart card as a secure and mobile storage media to manage personalized cookies. The article presents the development of the CookiesCard proxy that directly interacts with the smart card to provide cookies management, while acting as an intermediary between the client browser and a web server. In addition, the proxy is able to filter out any third party cookies which may be used to track the individual’s web-surfing behavior.
2. Cookies Issues and Management

Since its inception, HTTP is designed to be a stateless protocol that exercises simple request-response interaction between the browser client and the web server. Each request-reply cycle is independent such that no state information is retained across the cycles. To address this shortcoming, cookies are introduced by Netscape back in 1994 as a quick and convenient way to embed state information within HTTP through the use of Cookie and Set-Cookie primitives[3] in the request and response headers, respectively. In addition to cookies, there are alternative methods to introduce state retention such as hidden fields in HTML, appending state information in URL and recording client’s IP address. In using hidden fields and URL approach, web application needs to be carefully designed within the server-side application to maintain and ensure consistent state transfer between HTML forms. In the event when user breaks the flow of a session (for example by clicking the back button or visit another site), a state roll back to resume the session may caused inconsistency in state maintenance by the web application. Take the example of a shopping cart web application where the user has just selected a list of items to be purchased. The identifications of the dropped items are captured in the hidden fields embedded within the HTML form. At this point, the client may choose to move to another site for casual browsing and to return to the shopping site only to know that the state information on the items selected is lost1. In contrast, cookies non-reliance on content-based maintenance and capturing of state information make it easier to build web applications that are robust and scalable. Instead, state information is captured and stored in the client’s memory, which is associated with the domain and path of the web application that set the cookies. Regardless of the track of client’s browsing, the browser will automatically send the cookies information to the original web application that sets it. In this way, the web application is able to keep track of the user within a session and across different sessions.

2.1 Cookies Management

The increasing adoption of the WWW as the underlying infrastructure to deliver pervasive electronic commerce over the Internet has driven the popularity of cookies as the favorite choice for managing state retention over HTTP. Unless technically inclined, many users are unaware of the wide-spread use of cookies that are “planted” on their machines as they surf across different web sites. Cookies initiated by the web sites are kept in the browser’s memory, which are used across a session. Often these cookies are set with expiry dates that allow the original web site to control the exact expiry dates of the cookies, which often extend beyond a session. In these cases, cookies are written into a stable storage within the client’s machine and are stored as persistent files. Fortunately, most off-the-shelves browsers are equipped with some form of cookies management facilities that allow users varying degree of control over the use of it[4]. Generally, most browsers offer the following forms of control:

Cookies Acceptance. Realizing the potential cause of paranoia for users in having web sites secretly setting unsolicited cookies, browsers are designed with options to allow users to decide on the level of acceptance of cookies. In the extreme case, user can choose to reject all cookies originating from all web sites. With the pervasive use of cookies in many modern web sites, this option will seriously limit user’s surfing experience, which in some cases, may render it impossible for user to visit a web site. Alternatively, a user may choose to be alerted on the receipt of each cookie and to make online decision of whether to accept the cookie originating from a web site. The acceptance of cookies can also be set based on selected trusted web sites. In this case, the browser provides the option for user to list the web sites that are considered trust worthy, and to allow these sites to freely set cookies on the client’s machines.

Cookies Size. Cookies are essentially ASCII represented texts that contain information necessary for a web site to retain the session state information on the client machine. Often, each cookie does not take more than tens of bytes to store the state information required by the web application. Yet, in the specification of cookies usage, a web site is technically able to store up to 20 cookies, each with a maximum size of 4K bytes and for a total of 300 cookies on each machine. To reduce the potential of cookies spamming on a client machine and to reduce the risk of arbitrarily large cookies, browsers are often equipped with the option for users to control the size and number of cookies they accept.

Third Party Cookies. Cookies that are sent from web servers other than the original server that the user is currently visiting is known commonly as third party cookies. Importantly, these cookies are often set by the third party web sites without user’s awareness and knowledge. These cookies are often used by advertising companies as a means to track the surfing behaviors and profile of a user with the aim of providing targeted delivery of ads that are of interest to the user. While this may be argued as a desirable service, where users may as well receive ads that correspond to their interests than random untargeted ads. The fact that users’ profiles are being tracked in the form of third party cookies exposes the controversy of privacy, where advertisers are challenged on the rights of collating profiles of users without their knowledge. To address this concern, browsers are equipped with control to allow users to filter out and reject cookies that are sent from URLs that are not coming from the original site that is being visited.

2.2 Personal and Mobility of Cookies

Cookies provide a general mechanism for web site to maintain state information across multiple HTTP request-response cycles. The ease of maintaining and robustness of cookies to retain state even across multiple sessions, have motivated the use of cookies to capture personal information to provide personalization based on user’s profile[4]. For example, web sites such as Amazon retain user’s profile in the form of a cookie that represents an index to a back-end database. The database pointed by the cookie contains important information on user’s preferences, shipping addresses, past transactions, reading interests, etc. Often this is a desirable and convenient service to end user, which enhances the surfing experience to the site with personalized web pages that target user’s specific interests and profile. In some sites, cookies are even used as a way to store personal information such as user login identification and password. With the identification information stored on client’s machine, the web application is relinquished of the need to maintain a server-side user access control database. In short, cookies are increasing used as a means to capture personalized

---

1 Unless client continuously click the back button to roll back to the session before the break.
information that is maintained on the client’s machine. This appears to be a reasonable and convenient solution if user operates on a fixed machine, where the cookies pertaining to the user is tightly coupled to the workstation. Such assumption is invalidated once user operates on another machine, perhaps from a public machine, where cookies capturing personalized information are lost. In summary, using cookies to maintain personalized information presents the following issues:

**Machine dependent.** The concept of cookies being stored on a client machine is mismatched with the increasing use of personalized cookies where they are tightly coupled to individuals, who are often mobile. Cookies are written to persistent storage as disk files in the computer. In a single user operating system, these cookies are shared among all users of the computer, where no differentiation among users’ cookies is maintained. This is not an issue on a multi-user operating system, where cookies are separately maintained based on the associated login user account. In both situations, maintaining personalized cookies on a machine presents the problem of cookies mobility when user moves to operate on another computer.

**Security and Privacy issues.** Another problem with cookies is that many users may find them obtrusive. They are stored on disk and leave a persistent record of a visited Web site. This is particularly sensitive when the user operates on a public machine where cookies of individuals are potentially exposed and can easily be read, copied and modified.

The ubiquitous use of the WWW as an information and commerce delivery platform, coupled with the benefits of cookies to support state retention, have motivated the need to address the management of “mobile” cookies that is tightly coupled to individuals. In particular, cookies need to be mobile-enabled so that personalized information may be retained and deployed across different client machines. In this paper, we present the novel design and implementation of the CookiesCard framework that uses smartcard as the solution for cookies mobility in a pocket. The lightweight and portable characteristic of smartcard presents an attractive solution to securely store and manage cookies associated with the user, while offering the ease of mobility in the pocket.

### 3. Smart Card Application Development

Building on the same spirit as the original Java, Sun Microsystems has recently developed the so-called Java Card [5] programming architecture for smart cards to facilitate the concept of “write once, run on all cards”. The idea of building a lightweight Java Virtual Machine (JVM) around the processing architecture provides the programmer with the opportunity to develop applications on smart cards using familiar Java language and programming paradigms. Importantly, the Java Card’s JVM hides away the card’s manufacturer proprietary technology with a common system interface.

#### 3.1 Application Protocol

To standardize the communication protocol between the off-card program and the on-card application, ISO has taken the initiative to define the ISO standard known as ISO/IEC 7816 [6]. The standard defines command messages and response formats returned by the card. The unit of transfer between these messages and response formats is known as Application Protocol Data Units, in short APDU. Figure 1 shows the communication flow between the off-card program and on-card application via the terminal card reader. Development of smart card applications under this approach requires a detailed understanding of the card’s command set and the need to construct primitive APDUs for exchanges of messages and responses. The Java Card software package provides classes for the construction and parsing of these primitive APDU messages. The fact still remains that the off-card program and on-card application communicate in a non-standardized byte format, which is often unstructured and ad-hoc in nature.

![Figure 1 Communications using APDU](image)

### 4. CookiesCard Architecture

In addition to the motivation of decoupling cookies from client’s physical machine and to support cookies mobility, an important requirement of developing CookiesCard is the need to avoid re-engineering existing technology infrastructure. Realizing the issues of privacy and security posed by cookies, most modern browsers such as Netscape and Microsoft Internet Explorer, are equipped with some form of cookies management options to allow end users to set the policies for managing cookies on their machines[4]. The available policies are often dictated by the browser’s developer, which may not provide adequate options to effectively manage and control the flow of cookies. One possible approach in the development of CookiesCard is to extend and modify existing browser design to support mobility of cookies across different machines. Importantly, the CookiesCard-enabled browser must be able to perform cookies management in tandem with the smart card’s applications to provide effective control over the influx of cookies. The recent release of Netscape Communicator source code in 1998 to promote open source development of the Mozilla2 browser project makes it an ideal candidate for the development of CookiesCard over the browser’s platform. While feasible, this approach may significantly restrict the portability of CookiesCard to operate across heterogeneous system platforms and browsers. An alternative approach is to develop the CookiesCard architecture based on a proxy server approach. The proxy server represents a local server that sits between an existing client browser application and a remote web server. It is responsible to intercept all client’s HTTP requests to the designated web server and if necessary, parse and process the request. In a similar manner, the proxy server is responsible to capture the web server’s HTTP responses to the client browser and if necessary, parse and extract the cookies from the response headers. Importantly, the proxy server architecture provides a unique method and opportunity to introduce added services to an

---

2 “Mozilla” was the original code name for the product that came to be known as Netscape Navigator, and later, Netscape Communicator. The name Mozilla is now used as the generic term referring to internet client software developed through the open source project.
end-to-end client/server web computing. In short, the proxy server architecture has the following benefits:

**Enhance performance through service extension:** Proxy architecture enables intermediary server to be inserted between an end-to-end client/server computing model. In which case, the server may be configured to add service to the computation without the need to change the underlying HTTP protocol. An excellent example of such service is a HTTP cache proxy, where requests and responses are intercepted and cached on local memory. Subsequent requests for the same web objects may be directly served by the cache proxy instead of the real server.

**No modification to client and server:** A significant benefit of the proxy architecture is that the protocol served between the client and server requires no modification. This represents ease of deployment of enhanced services to existing computation.

In this project, we make use of the proxy architecture to provide cookie management facility based on smart card. The CookiesCard’s proxy is designed to intercept and parse all HTTP requests and responses in order to facilitate cookie management on the proxy and smart card. Cookie set by the server is transparently captured by the proxy and stored locally. Conversely, request destined for a URL site in which a cookie is associated with will be appended with the cookie to be returned to the server. Since all cookies set are intercepted and managed by the proxy, the client browser is effectively freed from managing the influx of cookies, thereby decoupling them from the client’s machine.

### 4.1 System Architecture and Implementation

Shown in **Figure 2** is the system architecture of the CookiesCard framework. The proxy server, running on the same host as the web browser, acts as an intermediary to capture and process HTTP request and reply messages transversing between the browser and the web server. In addition, it communicates to the smart card’s applications via standardized APDU messages.

![Figure 2 Architecture of CookiesCard](image)

The proxy server is comprised of two modules namely, the Interceptor module and the Off-card Cookies Manager module. The Interceptor module is responsible to continuously listen for HTTP requests arriving from the web browser and the corresponding responses coming from the web server. On receiving a HTTP request, it is required to parse the request and to check with the Off-card Cookies Manager cache on the need to append cookies destined for the URL site. Similarly, on receiving a HTTP response message from a web site, the Interceptor is required to parse the response headers for `Set-Cookie` primitives, extract cookies from the header if there exists and pass them to the Off-card Cookies Manager, and to return the remaining response contents to the web browser. The Off-card Cookies Manager is responsible to manage the cookies cache and to process cookies from the HTTP response. To accelerate the searching and updating process, the module implements the `CookieHash` class that manages the caching of cookies on the main memory. As its name implies, it is implemented using a hierarchy of hash tables, which first map the domain key to the entry in the path hash table. The value mapped to the path key is a *vector*, which holds the cookies which are valid for the corresponding domain and path. The logical structure of the `CookieHash` class is illustrated in **Figure 3**. In addition, the Off-card Cookies Manager acts as a host interface that directly interacts with the smart card’s applets via APDU messages to provide user access control, cookies uploading and cookies updating on the smart card storage media. To minimize the storage space required, all cookies stored within the smart card are text compressed using direct Huffman coding scheme.

![Figure 3 Logical Structure of CookiesHash Class](image)

The smart card is comprised of four modules, each implemented as a JavaCard applet class. The Access Control applet is responsible to process APDU message containing the PIN input by the user when the card is first inserted into the reader. The PIN is extracted by the applet and compared against the stored PIN. Once authenticated, the Off-card Cookies Manager can communicate with the rest of the applets residing within the smart card. As part of the security feature, a configurable maximum retry counts can be set. For each time the PIN is entered wrongly, the retry count is decremented. Once the count reaches zero, the applet will permanently block the access to the card’s contents and it can no longer communicate with any off-card applications. The Storage Manager applet implements the file system storage to contain the cookies and cookies management policies configuration file. Although the smart card file system is defined in ISO7816-4, it is specified as low-level hierarchy-based file system organized into a collection of three elementary file types, namely the master file (MF), dedicated file (DF) and elementary file (EF). The main function of the Storage Manager is to encapsulate the organization of these low-level elementary file types to an object-based file system abstraction. The On-card Cookies Manager applet is responsible to manage the storage and indexing of the cookies based on the domain and path names that set the cookies. The Cookies Storage applet provides an object-level abstraction of the cookies to enable robust manipulation and processing of cookies information.
5. CookiesCard Implementation

An important requirement in the implementation of the CookiesCard framework is to provide support of portability and execution over heterogeneous execution platform. This has naturally led us to the selection of Java as the core development platform. In addition, we make use of the OpenCard Framework (OCF)[5] which supports open standard that provides vendor independent framework for developing host side applications. The OCF primarily supports 100% pure Java smart card application that matches the requirements and objectives of CookiesCard. The OCF provides encapsulation on the details of communicating with the smart card reader by exposing standardized and open application programming interfaces (APIs). As it turns out, we can extend the reusability of the program modules, and therefore to make our system to be compatible with all Java Cards, given that the vendors have provided a driver class for its smart card reader based on the OCF standard.

On the smart card side, we make use of the Java Card software development kit for the development of portable smart card applets. The applets are developed and tested to operate on the GemXpresso 211PK smart card developed by Gemplus. While the spirit of developing Java Card is similar to mainstream Java to support object-oriented development and true code portability across smart card systems, the features and capabilities of Java Card Runtime Environment (JCRE being akin to JVM) are significantly streamlined and abstracted to match the very basic executing resources of smart card. The GemXpresso 211PK is based on an 8-bit microcontroller, with 32Kbytes ROM, 32 Kbytes EEPROM and 2 Kbytes RAM.

To truly exercise the potential and functionality of the framework, we have successfully tested CookiesCard to operate over several popular web sites and also set up experimental test bed to evaluate the complex interactions between the various components of CookiesCard. Shown in Figure 4 is the screen shot of the CookiesCard proxy program that is implemented on a Java platform, which supports true portability across heterogeneous systems.

6. Conclusions

This paper presents the design and implementation of CookiesCard. The successful design and implementation of the prototype has demonstrated the benefits of managing cookies on a smart card to facilitate cookies mobility in a pocket. Users are able to “carry” their personal cookies on a smart card which promotes strong security and avoid privacy intrusions. This is in sharp contrast to current deployment of cookies which is strongly coupled to machine and presents security challenges and potential privacy intrusions. Instead of associating cookies to individuals, cookies are inherently machine-dependent in current implementation such that they are tightly coupled with computers which browse the web sites rather than individual user who is browsing the site. With the development of CookiesCard, cookies are stored to the individual user’s smart card, so that cookies are coupled to users but not with machines. The system employs a proxy architecture that acts as an intermediary between the web and smart card. Importantly, the proxy captures transiting cookies and manages the cookies in tandem with user’s smart card. In this project, smart card forms a secure storage media by requiring user to submit his/her PIN to verify the card holder identity. A prototype system to test and validate the concept has been successfully implemented on a Java platform for the proxy, while Java Card is used on the smart card.

7. ACKNOWLEDGMENTS

This project is supported by the Central Earmarked Research Grant B-Q453.

8. REFERENCES