Secure Cookies on the Web

JOON S. PARK AND RAVI SANDHU
George Mason University

Web servers and browsers typically use cookies to capture information for subsequent communications, which provides continuity and state across HTTP connections. At present, Internet e-commerce is somewhat limited in using cookies because sensitive information cannot be securely stored and communicated in typical cookies. Secure cookies offer a potential solution to this problem.

The World Wide Web facilitates e-commerce on the Internet via its underlying hypertext transport protocol, which carries all interactions between Web servers and browsers.[1] Since HTTP is stateless, however, it does not support continuity for browser-server interaction between successive user visits. Without a concept of a session in HTTP, users are strangers to a website every time they access the Web server.

Cookies were invented to maintain continuity and state on the Web.[2,3] They contain text-character strings encoding relevant information about the user. Cookies are sent to the user's hard drive or RAM via the browser while the user visits a cookie-using website. The Web server retrieves the user's information from those cookies when the user later returns to the same website. The cookie's purpose is to acquire information for use in subsequent server-browser communications without asking for the same information.

It is not technically difficult to encode cookies with relevant information. For instance, a merchant Web server could use a cookie that contains the user's name and credit card numbers. Although this would be convenient for users, it would also be risky. Because they are stored and transmitted in clear text, cookies are readable and easily forged. One way to solve this problem is to make cookies secure, and in this article we discuss several techniques that render cookies secure for carrying and storing sensitive data.

We show how secure cookies enable secure attribute services between (and without modifying) existing Web servers and browsers, and identify representative applications for this technology. Secure cookies are constructed using well-known cryptographic techniques such as message digests, digital signatures, message authentication codes, and encryption. The detailed mechanism depends on how these techniques are applied to implement secure cookies and to which Web services secure cookies are applied. Notably, secure cookies can be issued by one Web server for use by another, which facilitates secure attribute services.

Different techniques for secure cookies are needed to provide tradeoffs between security and convenience for end users, system administrators, and application developers. Since cookies are inherently user-pull,[4] we focus on the user-pull architecture in this article. Our focus is on integrity, authenticity, and confidentiality for nonanonymous Web transactions.

 1089-7801/00/$10.00 ©2000 IEEE

	Domain	Flag	Path	Cookie_Name	Cookie_Value	Secure	Date
Cookie 1	acme.com	True	/	Name_Cookie	Alice	False	12/31/2000
⋮			⋮		⋮		
Cookie *n*	acme.com	True	/	Role_Cookie	Manager	False	12/31/2000

Figure 1. An example of typical cookies on the Web.

COOKIES

Cookies serve many purposes on the Web, such as selecting display mode (for example, frames or text only), maintaining shopping cart selections, and storing user identification data.

All cookies are fundamentally similar. A typical cookie, shown in Figure 1, has several fields. Cookie_Name and Cookie_Value contain information a website would want to keep—for example, in the figure, the values of Name_Cookie and Role_Cookie are "Alice" and "Manager," respectively. Date is the cookie's valid lifetime. Domain is a host or domain name where the cookie is valid. Flag specifies whether or not all machines within a given domain can access the cookie's information. Path restricts cookie usage within a site (only pages in the path can read the cookie). If the secure flag is on, the cookie will be transmitted only over secure communications channels, such as the Secure Sockets Layer (SSL) protocol.[5] For detailed cookie specifications, see Kristol[2] and Moore.[3]

According to the current HTTP state management mechanism, whenever a browser requests a URL to a Web server, it sends only the relevant Cookie_Name and Cookie_Value fields (selected by means of the Domain and Flag fields) to the server. Cookies received by the server are used during this browser-server communication. If the server does not receive any cookies, however, it either works without using cookies or it creates new ones for subsequent browser-server communication.

A Web server can update cookies' contents whenever the user visits the server. The cookie-issuer is not important for validation; any Web server can issue cookies for other Web servers.

Security Concerns

Web servers often use cookies to identify visitors and their status. For instance, if a merchant website has a customer database containing information such as names, payment histories, and credit card numbers, the site uses cookies to store pointers to individual customer records. Because cookies are easily forged, it is reasonable to store a simple customer ID number (pointer) in a cookie rather than all the customer's information. This ID number is exposed in a cookie without exposing the actual customer data, although even this ID can be forged.

Although SSL can foil network threats, it can protect cookies only while they are on the network.

Privacy—or lack thereof—is one of the main concerns about cookies in the popular press. Cookies allow Web servers to track a user's browsing behavior, although cookies cannot release contents of a user's hard drive because a cookie is written in a text file. Privacy concerns, however, are outside the scope of this article.[6]

Security Threats

There are three types of threats to cookies: network threats, end-system threats, and cookie-harvesting threats. All three are easy to implement. First, cookies transmitted in clear text on the network are susceptible to snooping (for subsequent replay) and to modification by *network threats*. Although SSL can foil such threats, it can protect cookies only while they are on the network.

Second, once the cookie is in the browser's end system, it resides on the hard drive or memory in clear text. Such cookies can be trivially altered by users and easily copied from one computer to another, with or without the cooperation of the user on whose computer the cookie was originally stored. We call this the *end-system threat*. The ability to alter and copy cookies lets attackers easily forge cookies' information and impersonate other users.

	Domain	Flag	Path	Cookie_Name	Cookie_Value	Secure	Date
IP_Cookie	acme.com	True	/	IP_Cookie	129.174.100.88	False	12/31/2000
Pswd_Cookie	acme.com	True	/	Pswd_Cookie	hashed_password	False	12/31/2000
Sign_Cookie	acme.com	True	/	Sign_Cookie	Signature_of_Alice	False	12/31/2000

Figure 2. Authentication cookies. The top one contains an IP address for user authentication; the middle one, a hashed password; and the bottom one, a digital signature of the cookie owner.

Finally, if an attacker collects cookies by impersonating a site that accepts cookies from users (who believe that they are communicating with a legitimate Web server), the attacker can later use those harvested cookies for all other sites accepting them. We call this the *cookie-harvesting threat*.

SECURE COOKIES

Secure cookies provide three types of security services: authentication, integrity, and confidentiality. Authentication verifies the cookies' owner. Integrity protects against unauthorized modification of cookies. Finally, confidentiality protects against the cookies' values being revealed to an unauthorized entity.

User Authentication

Since typical cookies do not support authentication, a malicious user can simply snatch cookies from other users and impersonate the real owner to the server that accepts those cookies. To solve this problem, we introduce three possible authentication methods for cookies. Authentication cookies can be address-based (IP_Cookie), password-based (Pswd_Cookie), or digital-signature-based (Sign_Cookie). Figure 2 shows each of the three types. We can use one of those authentication cookies with typical cookies, depending on the situation.

Address-based authentication. An IP_Cookie grabs the user's IP address for address-based authentication. As the IP address is one of the environment variables for Web users, a Web server (cookie issuer) can readily obtain the user's IP address and put it into the IP_Cookie. Whenever the user (say, Alice) tries to access a Web server (which accepts the IP_Cookie), the server first checks if Alice's current IP address matches the one in the IP_Cookie she sent. If they are identical, the server believes that Alice is the real owner.

Address-based authentication is a convenient authentication mechanism because the authentication process is transparent to users, but such a method is not always desirable. For example, what if Alice's IP address is dynamically assigned to her computer whenever she connects to the Internet? In this case, although Alice consistently uses the same computer, the IP_Cookie she received on a previous Internet connection is invalid once the IP address changes.

Furthermore, if Alice's domain uses a proxy server, an attacker can collect Alice's cookies, including IP_Cookie, by cookie-harvesting and easily impersonate her through the same proxy server, because the proxy effectively provides the same IP number to the users in the domain. In addition, we cannot avoid *IP spoofing*, which is a technique for gaining unauthorized access by sending messages to a computer with a trusted IP address.

Password-based authentication. Password-based authentication supports dynamic IP addresses or proxy servers and avoids IP spoofing. Passwords are transmitted from browser to Web server and should be protected on the network by means of SSL. If the Web server obtains Alice's passwords and puts the hashed password into Pswd_Cookie, other Web servers can authenticate the cookie owner later using the Pswd_Cookie. Alice must type the same passwords whenever she tries to access other servers accepting the cookie. If the hash of the passwords matches the one in Pswd_Cookie, then the server believes Alice to be the real owner. Alternatively, servers can use encrypted passwords in the Pswd_Cookie to authenticate the cookies' owner (a detailed encryption process is described in the "Providing Confidentiality" section).

This mechanism, however, is inherently vulnerable to *dictionary attacks* (attempts to gain access to

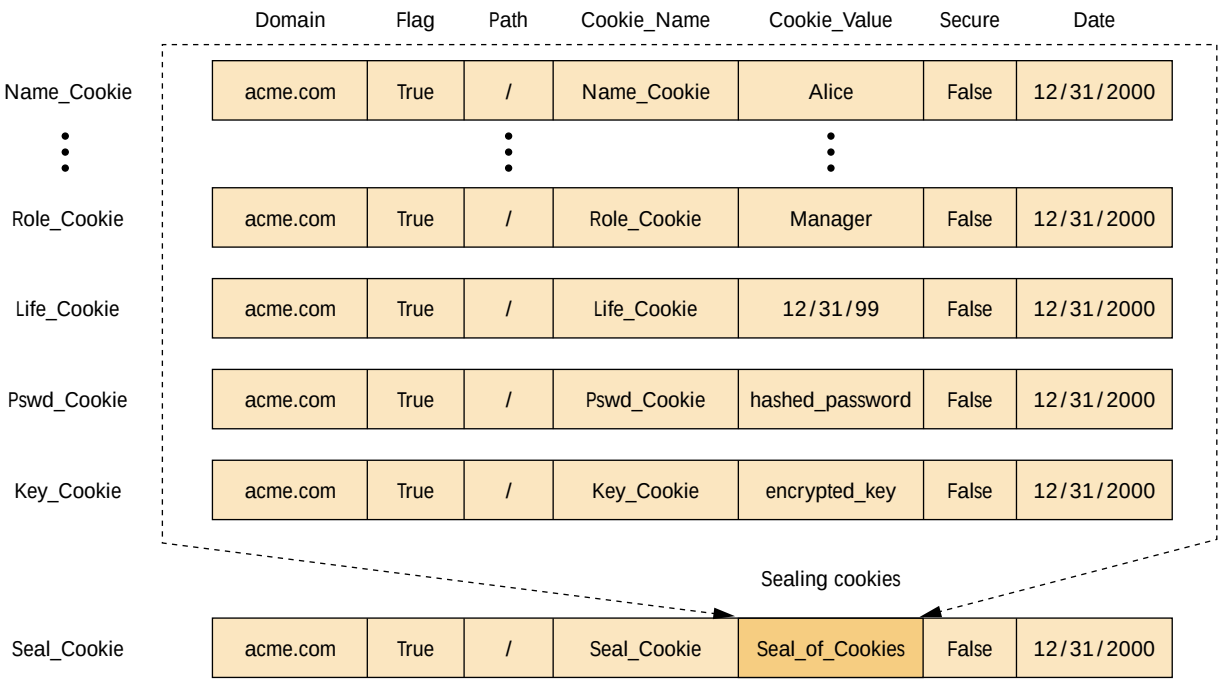

Figure 3. A set of secure cookies on the Web. The Pswd_Cookie can be replaced with either IP_Cookie or Sign_Cookie. Sensitive fields are encrypted in the cookies. Seal_of_Cookies can be either MAC or a signed message digest of cookies.

a resource by trying passwords or keys from a precompiled list of values), since the Pswd_Cookie reveals hashed or encrypted passwords. This technique requires users to enter passwords for authentication whenever they connect to the site, whereas using an IP_Cookie is transparent to users.

Digital-signature-based authentication. If Web servers know users' public keys, digital signature technologies like DSA[7] or RSA[8] can be used to authenticate users with cookies. In this method, users need additional browser software to generate a cookie that contains a signed time stamp. For instance, when Alice needs to access a remote Web server that knows Alice's public key, Alice's computer generates a time stamp and creates the Sign_Cookie shown in Figure 2, which has Alice's digital signature (signed with her private key) on the time stamp. When Alice connects to the remote Web server, it receives Alice's Sign_Cookie and verifies the signature with Alice's public key.

Secure cookies can also work with other authentication services such as Radius[9] and Kerberos.[10] Alice's authentication information (which depends on the authentication protocol) can be stored in a set of secure cookies and used in conjunction with that protocol.

Client-to-server authentication is handled directly in our secure-cookie mechanism. When server-to-client authentication is required, we can use the server's public-key certificate by means of SSL. Mutual authentication can be achieved by the SSL handshake protocol, if the client certificate—required by an optional SSL configuration—is sent to the server from the client. However, some clients do not have their client certificates, and some SSL packages do not yet support client-to-server authentication.

Providing Integrity

Cookies also have integrity problems. For instance, an attacker can copy Alice's IP_Cookie and edit it with an IP number, then later impersonate Alice to a Web server. Even Alice can change the contents of her own cookies. Figure 3 shows a set of secure cookies, and Figure 4 (next page) shows how the secure cookies are used on the Web.

The Cookie_Value field of the Life_Cookie shows the lifetime (expiration date) of the secure-cookie set and enables the Web server to check the integrity of the secure-cookie set's lifetime if the cookies are valid. Even though the browser sends only the relevant Cookie_Name and Cookie_Value fields to the Web server (selected by means of the

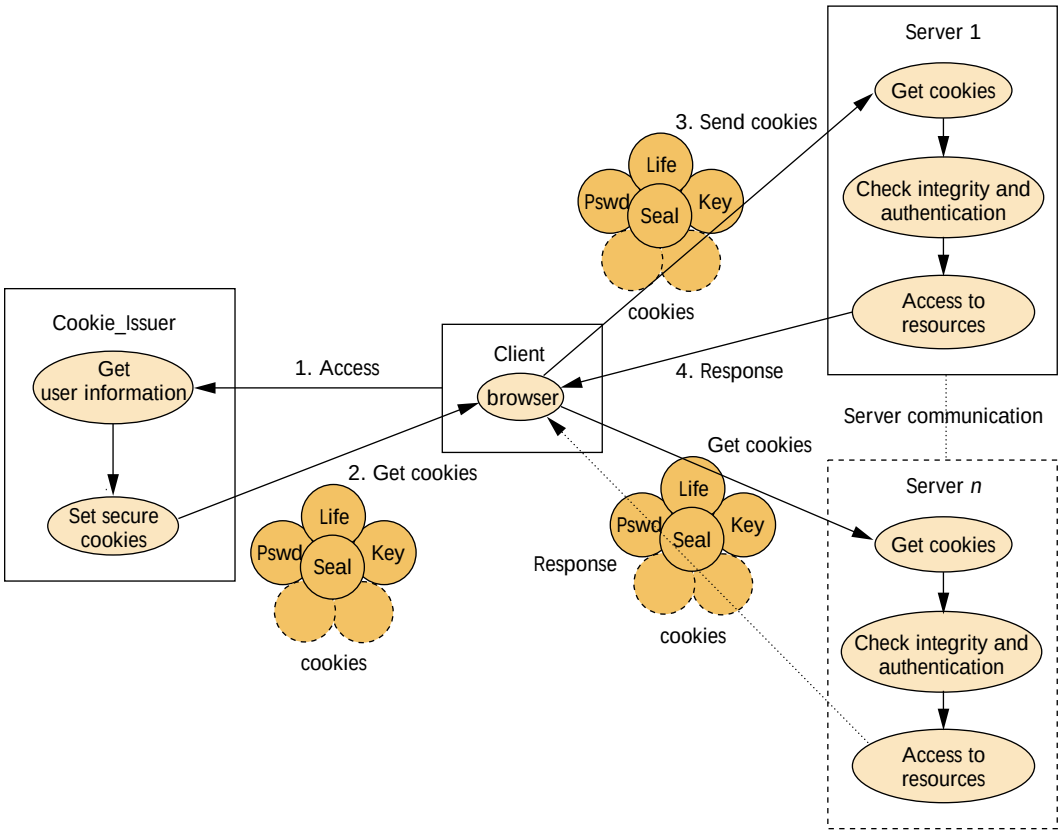

Figure 4. Using secure cookies on the Web. The Pswd_Cookie can be replaced with either IP_Cookie or Sign_Cookie.

Domain and Flag fields), the cookie-issuing server can establish a policy with the Web servers in the domain to check the integrity of other fields. For example, if the policy presets the values of the Domain, Flag, Path, and Secure fields with acme.com, True, /, and False, respectively, the Web server uses those values to check the cookies' integrity.

It is best not to preset an expiration date for all cookies under a domain policy, however, because each secure-cookie set may require a different lifetime. The Life_Cookie is used to solve this problem. Since each cookie is deleted from a user's machine automatically after its expiration date, the integrity of the cookie set would be changed (if cookies have different lifetimes). Therefore, it is reasonable to set the same expiration date for all the cookies that are stored in the Cookie_Value of the Life_Cookie.

If necessary, we can include other typical cookies in the secure-cookie set, containing information to be protected, such as credit card numbers, according to applications. The optional Key_Cookie facilitates encryption of sensitive data, as we describe later. In some cases, if we do not need cookies to be confidential—for example, if they lack sensitive information—we clearly do not need encryption.

Finally, the Seal_Cookie determines if cookies have been altered. The Seal_Cookie's contents depend on the cryptographic technologies used—essentially, either a public- or secret-key-based solution. The key distribution between servers can be achieved by key agreement algorithms such as RSA[8] or Diffie-Hellman.[11]

Public-key-based solution. With public-key cryptography, the cookie-issuing server uses a message digest algorithm, such as MD5[12] or SHA,[13] to create a message digest from the rest of the cookies. It then signs the message digest using its private key and puts the signature in the Seal_Cookie. Consequently, all secure cookies are stored in the user's computer.

When the user connects to a Web server—which accepts and can verify the cookies—next time, the browser sends the relevant secure cookies to the server. The server verifies the signature in the Seal_Cookie using the cookie-issuing server's public key and the values set by the policy (between the

cookie-issuing server and the Web server). A successful integrity verification means the cookies have not been altered.

In the public-key-based solution, only the server with the private key for the Seal_Cookie can update the contents of the secure cookies; thus for updates, the user must return to the cookie-issuing server with the private key. (If the private key is shared among other servers, it is possible to update the secure cookies at multiple sites.)

Secret-key-based solution. With secret-key cryptography, the cookie-issuing server creates a message authentication code (MAC) from the rest of the cookies using a key-dependent, one-way hash function such as HMAC[14] and puts it in the Seal_Cookie. When the user connects to a Web server, the server obtains all relevant cookies from the browser. If the Web server shares a secret key with the cookie-issuing server, the server creates a MAC from the cookies and the values set by the policy, and compares it with the one in the user's Seal_Cookie. If both MACs are identical, the Web server believes the cookies have not been altered.

If we use the secret-key-based solution for secure cookies, any server with the shared secret key can update the cookies' contents. For instance, a Web server can extend the cookies' expiration date, which means the user need not return to the original cookie-issuing server to update the cookies.

Providing Confidentiality

To prohibit individuals, perhaps even the cookie owner, from reading sensitive information in cookies, the Web server can encrypt names, roles,[15] credit card numbers, and so on. We use the Key_Cookie, shown in Figure 3, to store an encrypted session key, which is used to encrypt sensitive information in other cookies. The session key can be encrypted either by the server's public or secret key. We can encrypt the cookie's contents directly with the server's secret key or public key, eliminating the need for the Key_Cookie. However, for better secure services, we recommend a session key to encrypt the cookie's contents. For public-key encryption, we recommend separate public-key pairs for encryption and digital signature because a server may share the private key for information decryption with other servers while keeping the other private key for digital signature secret.

When a Web server receives secure cookies, including the Key_Cookie, it decrypts the Key_Cookie's Cookie_Value using its master (private key or secret key)—which can be shared with other servers—to get the session key. With this key, the server decrypts and reads the information in the other cookies. If the cookies' contents were encrypted directly by the server's secret or public key, the server decrypts the information using the corresponding key.

Enhancement

An organization might have hundreds of Web servers, all requiring integrity, authentication, and confidentiality services in cookies. In such an environment, it is unwise to make individual Web servers share a secret key with others in the domain, as this increases the likelihood that the key will be exposed. To support more secure service, verification servers can share the secret key with other verification servers to verify, decrypt, or update cookies issued by cookie-issuing servers in the domain.

When a Web server receives secure cookies from a user, the server forwards them to a verification server, which verifies the cookies and sends the result to the Web server over a secure channel. The verification server can, if necessary, decrypt the cookies' encrypted values or update the cookies' information. Finally, the Web server trusts, and uses, the decrypted and verified information received from the verification server.

Encrypted Versus Secure Cookies

Some commercial products, such as getAccess (http://www.encommerce.com/productbrief.asp), use encrypted cookies. However, such products differ substantially from our approach. Simply encrypted cookies support confidentiality, but they cannot support authentication and integrity. For instance, a user (say, Alice) can use another's (say, Bob's) cookies even though she cannot read the encrypted cookies' contents. Furthermore, nonencrypted parts in the cookies can be altered without notice to the cookie issuer. Since there is no way to check the cookies' real owner and integrity of copied or forged cookies, those cookies should not be stored in the user's computer after each session. This means the user must receive new cookies from the cookie-issuer for every session, which causes the stateless problem of HTTP between Web browsers and server.

Our secure cookies, on the other hand, can be stored in the user's computer, even when it is off, after each session. This is possible because the secure cookies can be provided integrity and authentication services as well as encryption. Therefore, once the user obtains secure cookies, the cookies' information can be used until the cookies expire. This approach completely solves the stateless problem of HTTP and security problems in typical cookies.

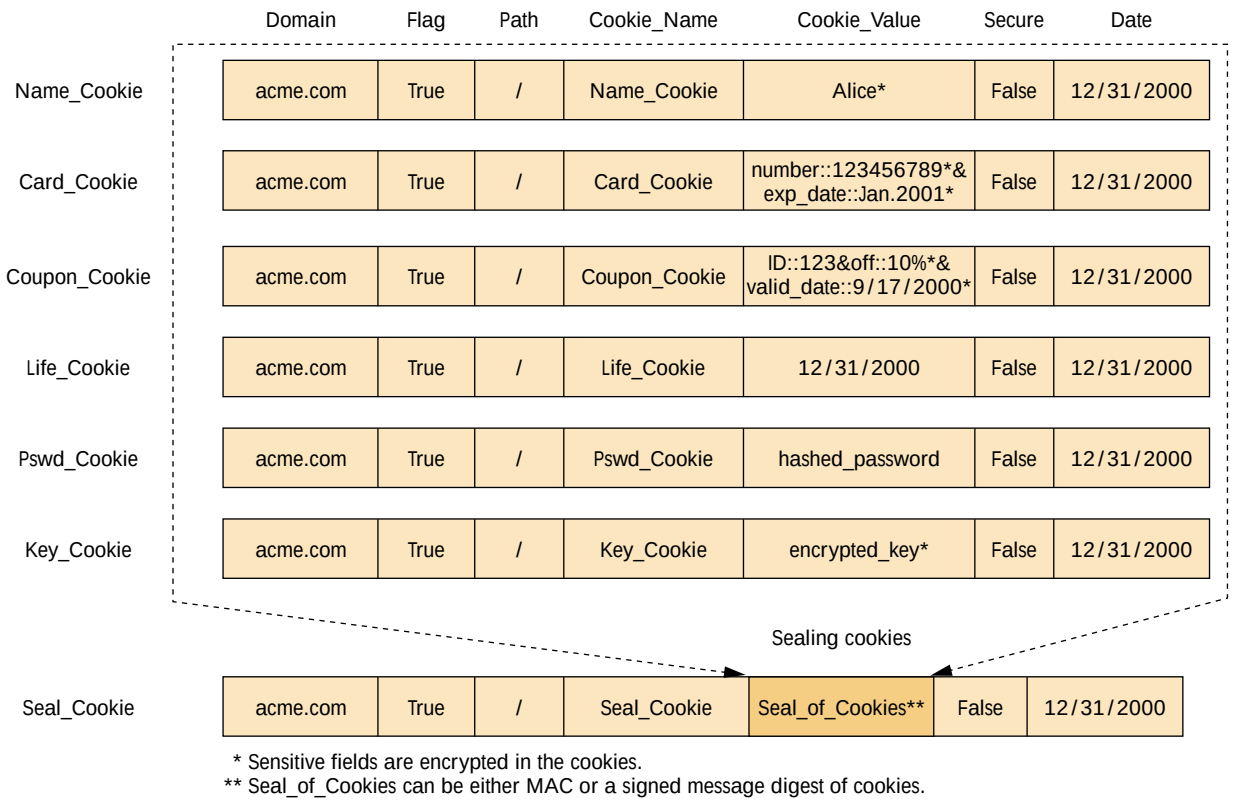

Figure 5. An example of secure cookies for electronic transactions.

APPLICATIONS

Four representative applications will illustrate the use of secure cookies. The applications determine which kinds of secure cookies are used. Regardless of application, however, at least one authentication cookie and the Seal_Cookie are required to frame basic security services.

User Authentication

Web servers can simply use secure cookies to authenticate their visitors. One or more authentication cookies (IP_Cookie, Pswd_Cookie, or Sign_Cookie) can support the Web server's user authentication service in conjunction with the Seal_Cookie, which supports the integrity service for the cookies.

Electronic Transactions (e-Commerce)

Because typical cookies lack security, a merchant site usually sets a cookie to hold a pointer such as an ID number, which is a key field in the site's customer-information database. However, the ID itself can be easily changed; moreover, this approach implies that a customer-information database must be maintained in a server. A disadvantage to this method is that if the server holding customer information is penetrated by an attacker, all of that information is vulnerable. Furthermore, a domain including multiple servers with multiple customer-information databases faces burdensome maintenance and synchronization tasks. Such data also arouse significant privacy concerns, as it can easily be misused. Users may feel more comfortable with servers that pledge not to maintain such data.

Secure cookies can solve these problems, especially in e-commerce. If a merchant site creates secure cookies like those shown in Figure 5, it does not need a customer-information database unless it tracks customer access histories, because each customer's information is distributed and stored securely in the secure cookies on the customer's hard disk. Secure cookies offer more security by eliminating customer-information databases that can cause a single-point failure; furthermore, the merchant reduces database maintenance costs.

In Figure 5, the Card_Cookie and Coupon_Cookie have pairs of cookie name and value in their Cookie_Value fields. Intuitively, a merchant site

should set the Card_Cookie to expire before the credit card's expiration date. For more convenient services, the merchant can issue special tokens for customers, such as electronic coupons, which contain the coupon's ID number, discount information, and expiration date. In this case, the merchant site must keep a record for the coupon by the ID to prevent *replay usage* (the same coupon's being reused).

Pay-Per-Access

Many pay-per-access websites provide various information services, such as performances, games, and movies. Secure mechanisms—such as secure cookies—to sell, buy, and use tickets to such sites are obviously needed.

Suppose Alice wants to buy access to such a site. After she pays, the server gives her a token that affords her access until the ticket expires. If Alice receives, say, a Ticket_Cookie that contains her access limit and valid date, along with other secure cookies, she alone can access the site as long as the Ticket_Cookie is valid.

To prevent replay attacks, a merchant site must keep information about tickets—such as accumulated usage of each ticket—at least until the Ticket_Cookie becomes invalid. This does not mean that the merchant site must keep all customer information; the merchant need track only the ticket ID and its accumulated usage. In this case, merchant sites need not update the cookies' contents. For instance, if Alice's accumulated access usage exceeds the access limit denoted in her Ticket_Cookie, then the merchant site rejects Alice's request.

Attribute-Based Access Control

If a user's attribute information is stored in cookies securely, as shown in Figure 3, Web servers can use those cookies for attribute-based access control. Since the attribute information is protected from possible security threats on the Web as well as in end systems, a Web server can verify and trust attributes, such as roles, in the secure cookies.

CONCLUSIONS

We have implemented role-based access control on the Web as a possible application for secure cookies.[16] We used CGI scripts and Pretty Good Privacy[17] to create and verify secure cookies cryptographically. The role server issues a set of secure cookies, including the user's authentication information (encrypted passwords or IP number), role, and the server's signature. These cookies are transferred to and stored in the user's computer. When the user connects to a Web server, the relevant cookies are transmitted to the Web server. After cookie-verification procedures, the server—which accepts those cookies and can verify them—trusts the information and permits the user access on the basis of roles.

Secure cookies offer security by eliminating customer-information databases that can cause a single-point failure.

Rather than extend the current HTTP or cookie specification to satisfy our security requirements, we decided to stay with the standard HTTP and cookie specification for reasons of compatibility and current high usability. An area for future research would be the extension of our techniques for ensuring integrity, authenticity, and confidentiality of Web transactions for e-commerce. ■

REFERENCES

1. J. Gettys, J. Mogul, and H. Frystik, "HyperText Transfer Protocol (HTTP/1.1)," RFC 2616; available online at http://www.ietf.org/rfc/rfc2616.txt, June 1999.
2. D.M. Kristol and L. Montulli, "HTTP State Management Mechanism," work in progress; available online at http://ftp.ietf.org/internet-drafts/draft-ietf-http-state-man-mec-12.txt, Aug. 1999.
3. K. Moore and N. Freed, "Use of HTTP State Management," work in progress; available online at http://ftp.ietf.org/internet-drafts/draft-iesg-http-cookies-03.txt, Apr. 2000.
4. J.S. Park and R. Sandhu, "Smart Certificates: Extending X.509 for Secure Attribute Services on the Web," *Proc. 22nd National Information Systems Security Conf.*, U.S. Govt. Printing Office, Washington, D.C., 1999.
5. D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol," *Proc. Second Usenix Workshop on Electronic Commerce*, Usenix Press, Berkeley, Calif., Nov. 1996, pp. 29-40.
6. V. Mayer-Schonberger, "The Internet and Privacy Legislation: Cookies for a Threat?", *West Virginia J. Law and Technology*, West Virginia Univ. College of Law, Morgantown, W. Va., 1997.
7. Federal Information Processing Standards Publication, "Digital Signature Standard (DSS)," FIPS PUB 186, Nat'l Inst. of Standards and Technology, Gaithersburg, Md., 1994.

8. R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM*, Vol. 21, No. 2, Feb. 1978, pp. 120-126.
9. C. Rigney et al., "Remote Authentication Dial-In User Service (RADIUS)," RFC 2138, Apr. 1997; available online at http://www.ietf.org/rfc/rfc2138.txt.
10. B.C. Neuman, "Using Kerberos for Authentication on Computer Networks," *IEEE Comm.*, Vol. 32, No. 9, Sept. 1994.
11. W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory*, Vol. IT-22, No. 6, Nov. 1976, pp. 644-654.
12. R.L. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, Apr. 1992; available online at http://www.ietf.org/rfc/rfc1321.txt.
13. Federal Information Processing Standards Publication, "Secure Hash Standard," FIPS 180-1, Nat'l Inst. of Standards and Technology, Gaithersburg, Md., 1995.
14. M. Bellare, R. Canetti, and H. Krawczyk, "Keying Hash Functions for Message Authentication," in *Proc. Advances in Cryptography—CRYPTO 96*, Vol. 1109, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1996.
15. R.S. Sandhu et al., "Role-Base Access Control Models," *Computer*, Vol. 29, No. 2, Feb. 1996, pp. 38-47.
16. J.S. Park, R. Sandhu, and S. Ghanta, "RBAC on the Web by Secure Cookies," *Proc. 13th IFIP WG11.3 Conf. Database Security*, Kluwer Academic Publishers, Boston, 1999.
17. P.R. Zimmermann, *The Official PGP User's Guide*, MIT Press, Cambridge, Mass., 1995.

Joon S. Park works for the U.S. Naval Research Laboratory's Center for High Assurance Computer Systems via ITT Industries as a research scientist. His research interests include designing and implementing security services on the Web, workflow security, security assurance, CORBA/JAVA security, secure electronic commerce, security models, PKI, information systems policy and administration, and cryptography. Park received a PhD in information technology, specializing in information security, from George Mason University in 1999.

Ravi Sandhu is a professor of information and software engineering and the director of the Laboratory for Information Security Technology at George Mason University. His principal research interests are in information and systems security. Sandhu holds PhD and MS degrees from Rutgers University, and BTech and MTech degrees from IIT Bombay and Delhi. He is the founding editor-in-chief of the *ACM Transactions on Information and Systems Security* and is an editor for *IEEE Internet Computing*. He chairs the ACM's Special Interest Group on Security Audit and Control (SIGSAC).

Readers can contact Park at the Naval Research Laboratory, Code 5540, 4555 Overlook Ave. SW, Washington, DC 20375; jpark@itd.nrl.navy.mil; or Sandhu at George Mason University, MS4A4, 4400 University Dr., Fairfax, VA 22030-4444; sandhu@isse.gmu.edu.