Protecting Web Usage of Credit Cards Using One-Time Pad Cookie Encryption

Donghua Xu, Chenghuai Lu and Andre Dos Santos
College of Computing
Georgia Institute of Technology
U.S.A.
{xu,lulu,andre}@cc.gatech.edu

Abstract

The blooming e-commerce is demanding better methods to protect online users' privacy, especially the credit card information that is widely used in online shopping. Holding all these data in a central database of the web sites would attract hackers' attacks, impose unnecessary liability on the merchant web sites, and raise the customers' privacy concerns. In this paper we introduce and discuss in details the secure distributed storage of sensitive information using HTTP cookie encryption. We are able to employ One-Time Pads to encrypt the cookies, because encryption and decryption are both done by the server, which is an interesting characteristic overlooked by the existing systems. We implemented this protocol and showed that it is simple, fast and easy to program with.

1. Introduction

With the rapid expansion of the Internet, more and more people realize the convenience and efficiency brought by E-commerce. A recent study [2] showed that the online retail sales were expected to reach 65 billion dollars in North America in 2001. Most of these online transactions are carried out using credit cards. When a user purchases an item on the a merchant web site, she inputs her credit card number and expiration date in her browser, then the merchant site will debit her credit card account and ship the item to the user. Most web sites store the credit card information in a database after a transaction, in order to save the users from inputting the same credit card information repeatedly in the future. When the user makes a purchase next time, the web site can directly use the credit card information that she input last time.

With billions of dollars moving online by credit card transactions, the online safety of credit card numbers becomes a focus of public concern, especially as more and more large-scale online credit card thefts are reported. These thefts, as shown in Table 1, were all due to the fact that credit card numbers were stored in the merchant web sites' central databases.

Date	Web Site broken into	Credit Card numbers exposed
1/2000	CDUniverse.com	350,000
12/2000	CreditCards.com	55,000
12/2000	Egghead.com	3,700,000
3/2001	Bibliofind.com	98,000

Table 1:Exposed credit card numbers of hacked web sites

Table 1 only shows the most well publicized examples of the online credit card thefts. On one hand people like the convenience of online shopping, on the other hand they are worried about the safety of their credit card numbers. The fear of online credit card fraud has been holding back many people's desire to shop online, and hampering the growth of the E-commerce [6].

The reality of E-commerce is demanding a better way to protect users' privacy (especially credit cards) and reduce the merchant websites' liability. As a matter of fact, the merchant web sites do not need the customers credit card numbers after a transaction is carried out. The web sites store the credit card information only for the convenience of the same customer's next transaction. Unfortunately, as shown

above, storing this information in a central database always introduces big troubles after an attacker breaks into the server.

In this paper we introduce a scheme for distributed storage of sensitive information, in specific a One-Time Pad HTTP cookie encryption protocol, to avoid gathering credit card in a central database, while providing similar user convenience at the same time. In essence, our approach encrypts the credit card information using One-Time Pad, an unconditionally secure encryption method, and stores this information as cookies on the customers' computers. The central database of the web site only stores the one-time keys of the cookies, so even if an attacker breaks into the server, what she would obtain are the one-time keys which are just random strings to her without the corresponding cookies.

Cookie encryption *per se* is not new. And Park and Sandhu [5] also briefly mentioned a similar concept of distributed storage of private information using encrypted cookies. However they did not analyze the pros and cons of this scheme, neither did they propose any real life application of it. In this paper, we provide detailed discussions of the main related issues of this scheme, and introduce One-Time Pad to achieve perfect secrecy for the encryption. In addition, the credit card information that this scheme protects has its own characteristics that make this scheme even more secure in various respects, e.g. replay attack or malleability issue.

The rest of the paper is organized as follows. In section 2 we briefly introduce HTTP cookie and the cookie encryption problem. In section 3 we present our One-Time Pad encryption protocol. In section 4 we discuss and analyze the pros and cons of this protocol. Section 5 concludes this paper and briefly discusses our future work.

2. HTTP Cookie and cookie encryption

The web traffic is mostly composed of HTTP traffic. HTTP is a state-less protocol, so designed to make it convenient for multiple clients to access multiple servers arbitrarily. Each page-request from the client is processed independently on the server. By nature these HTTP page-requests are not associated with each other. However sometimes - in fact, nowadays most of the time - it is necessary for a server to maintain the state of client accesses. Cookie [3] was designed to help the server manage states. Cookies work as follows: the server processes a page-request from the client, then returns a cookie along with the page that the client requested; the next time the client sends a page-request to this server, the client sends that cookie to the server as well. Thus, the server knows what the state of this client on the last page-request was. Cookies are now used extensively by Web servers, to keep track of the clients' state and provide more convenient service to users.

The clear-text nature of cookie implies that a malicious intermediary between the client and the server would be able to intercept/modify the cookies. Therefore the standard specification of cookie [3] emphasizes that "...information of a personal and/or financial nature should only be sent over a secure channel."

However, even if the communication channel is secure, cookies can still be easy targets on the users' computers. There are various ways for a malicious party to steal this kind of information from the users' personal computers, ranging from Trojan horses to java-script bug exploits. For example, the E*Trade web site once encrypted users' passwords in cookies. An E*Trade cookie could be hijacked by a malicious third party using the ``cross-site scripting'' technique, while the encryption of the password was so weak that a cryptography expert could break it in 20 minutes [12]. Therefore, if a cookie contains sensitive information, it must be strongly encrypted.

One might argue that sensitive information should never appear in cookies in the first place. Instead, the server could generate a nonce string (or a Session ID), and send this nonce string in the cookie to the client, while storing this nonce string along with the sensitive information itself in the local database. When the cookie returns from the client, the server could extract the nonce string from the cookie, and retrieve the information corresponding to the nonce string from the database. By doing this, the client side only stores a nonce string in the cookie, which means nothing for an intruder even if she took hold of the cookie.

However, sending only nonce strings in the cookies implies storing all users' personal information in a central database, and there are serious risks in putting all eggs into one basket. A shopping web site with thousands of customers might have thousands of credit card numbers in its local database. Such databases are always luring hacker attacks, and would cause big troubles once the database servers were broken into, as shown by the credit card theft examples in section 1. Furthermore, because of privacy concerns, customers might not like to have their credit card numbers

gathered in a remote database. Although web sites can make promises that they will never mine or abuse customers' private information, some customers would feel more comfortable if the web sites simply erase their private information as soon as the information is used.

A better way for the merchant web site to reduce its own liability and protect the users' privacy is to strongly encrypt the sensitive information and send it in a cookie back to the users' computers, instead of storing the sensitive information in the web server's database.

One notable example of the usage of cookie encryption is Microsoft's Passport technology [4] that allows a user to sign into multiple web sites by inputting user id and password only once. The central Passport server issues and authenticates encrypted cookies with the user's browser, and the participating web site only needs to redirect the browser to the Passport server when authentication is needed.

However, this technology still maintains all users' information in the Passport server's central database, raising users' concern of privacy, luring hackers' attacks, and forming an actual single point of failure on the Internet. In comparison, what we are proposing is to let the user keep her own sensitive information in a manner that is guaranteed to be secure, and not to store this information on the server at all.

Although Park and Sandhu [5] already mentioned that cookie encryption could be used for distributed storage of sensitive information, they did not propose or analyze any specific case that this would bring benefits over current use of nonces and session IDs. In this paper we introduce a better encryption scheme – One-Time Pad – and show where this would bring benefit to real world applications/systems.

3. One-Time Pad Cookie Encryption

OTP(One-Time Pad) was first proposed in 1926 by G. Vernam [10] to encrypt wire and radio communications. In 1949 C. Shannon [8] proved the perfect secrecy of OTP. The essence of OTP is to generate a random string of at least the length of the message to be the key of this message, then XOR the key with the message to produce the ciphertext. In order to decrypt the ciphertext, XOR the ciphertext with the key again. It is impossible to crypt-analyze OTP, since every message has a different random key, and there is no way to distinguish the ciphertext from a random string.

Although OTP is simple, fast and unconditionally secure, it is not widely used. There are two obstacles that prevent OTP from being used in generic applications. The first is the generation of random numbers. Although we recognize that this is a big obstacle for using OTP encryption we do not address any new solution for this problem in this paper. We assume that if the second obstacle, the key distribution problem, can be solved then pseudo random number generator with periodic reseeding can be used. Although the resulting algorithm would not be a one-time pad on its most strict sense, it would provide a reasonable approximation to the one-time pad.

In the usual scenario of secure communication, encryption of a message is done by one party, and decryption of the same message is done by another party, with both parties sharing the same key for the same message. However the OTP encryption requires a new key for every new message, which in turn requires a mechanism to distribute keys so that both parties can have the same key for a message while no third party could intercept or tamper with these keys. If such a mechanism exists for generic applications, then the encrypting party could have used this mechanism to send the message directly without encryption. In reality, no such practical mechanism exists *a priori*, so the fear of leaking the keys out is keeping people from using OTP extensively.

However, it is easy to notice that in applications where encryption and decryption are performed by the same party, the key distribution problem is not an issue anymore. This means that we can employ OTP on such encryption tasks, and achieve the perfect secrecy.

One application that shows such a characteristic is cookie encryption: encryption and decryption of the cookies are done by the same party – the server. The client simply stores and sends back the cookie, but does not participate in cookie encryption/decryption. In other words, the client does not use the information in the cookie. Therefore the keys are solely used by the server, instead of being shared by two parties.

This characteristic shows up only when the party that performs both encryption and decryption does not want to or is not able to store the sensitive information after its use. Otherwise, if the encrypting/decrypting party can and wants to store the information, then this party could have simply generated a nonce string to send to the other party, while storing the real information with the nonce string in the local database, since the other party does not need the real information

anyway. What makes cookie encryption so special is the web sites' motive that they should respect users' privacy and reduce the web sites' own liability. This is the main reason the web sites do not want to store customers' credit card information after its use.

Based on this observation, we developed an OTP cookie encryption protocol to protect online users' privacy. The protocol is straightforward. In short, the web server stores the OTP keys in a local database and encrypt/decrypt the cookies using these keys. Figure 1 shows the flow charts for cookie encryption and Figure 2 shows the flow chart for decryption. When a web server needs to encrypt a cookie before sending it out, the web server follows the upper chart. When the web server receives a cookie from a client, the web server follows the lower chart to decrypt the cookie. Notice that after the decryption, whether or not to remove the key from the key database would depend on the web designer's choice. More details will be discussed in section 4.6.

The web server also runs a clean-up program periodically to check the expiration times in the key database against the current time, and deletes all the expired keys. There are several reasons why a key expires. The main security related reason is to provide a mechanism to limit the time the server stores the key, hence limiting the web server's liability in case an attacker breaks into both the database and a user's computer. Another reason is that sometimes users might leave a web site and not come back anymore, or chooses to clean her cookies even if she comes back next time, or the users' system might simply crash, in which cases the cookies would not return to the web server anymore. Therefore the corresponding keys in the web server's key database would become zombies, occupying the disk space and the space of index string unnecessarily. We should have a separate program to clean up the zombie keys, in order to save disk space and index string space over time.

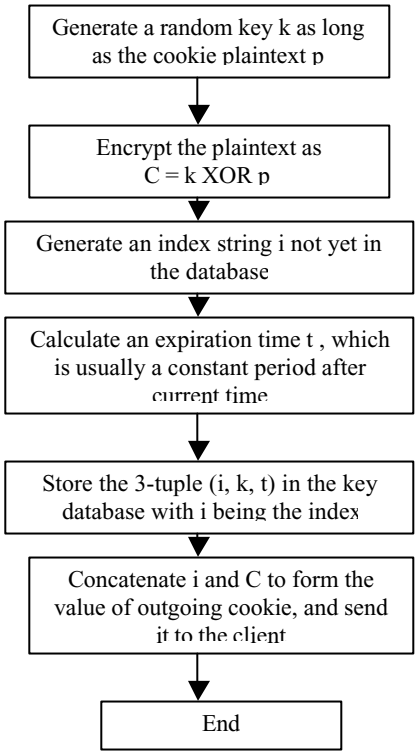

Figure 1: Cookie Encryption

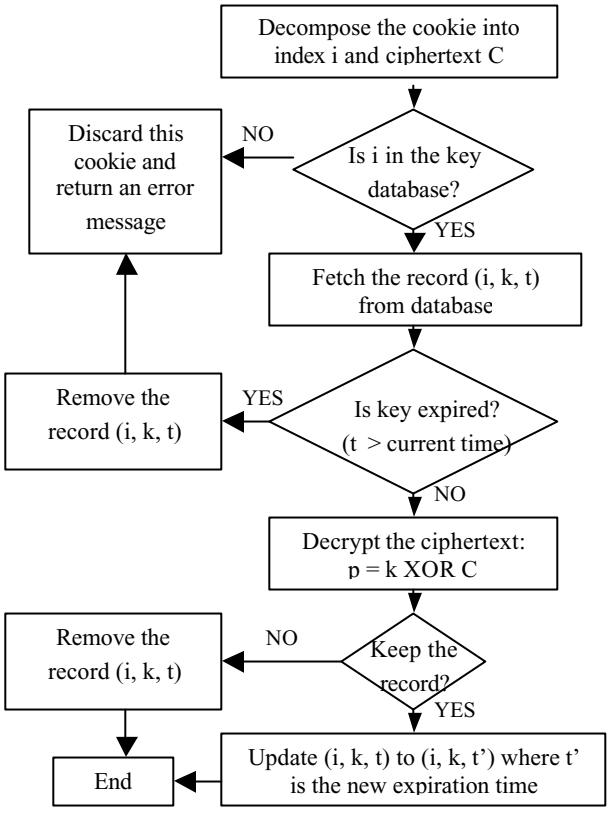

Figure 2: Cookie Decryption

4. Analysis

The protocol described on this paper was designed to protect online usage of credit cards, and can be generalized to be a scheme of distributed storage of information that protects other kinds of sensitive information. We will analyze and discuss the advantages and related problems of this protocol in the following subsections.

4.1. Advantages of Distributed Storage of Sensitive Information

In general, the main advantage of distributed storage of the customers' sensitive information is that, the risk is also distributed, and the damage caused by compromises is limited. If an attacker breaks into the database server of the web site, she only gets a table of the one-time keys, which is useless without the corresponding encrypted cookies. If the attacker breaks into the user's personal computer or by other means gets hold of the user's encrypted cookie, she also has no practical way to crack it. If the attacker breaks into both the server and the user's computer, then this user's cookie is compromised, but all the other users' cookies are still safe, as long as the attacker is not able to get their cookies. In the worst scenario the attacker breaks into the server and has a way of listening to all the traffic of this server. The clients that send the credit card cookies to this server in this period would have their credit cards exposed. However, the other users that do not send the credit card cookies to this server in this period are still safe. Hence the loss on the worst case scenario is limited by the length of period that the attacker can stay on the server. This is much better than gathering all users' credit card numbers in a single database, and exposing all of them once the server is compromised.

In addition, there is a simple way for a merchant web site to ensure that the credit card cookies are sent to the web site only when necessary. The cookie has a *path* option that specifies which path on the server requires this cookie. The web site can single out the program that processes the last step of a transaction, which really requires credit card number, and place the program in a path different from all other programs and web pages. When the web server sends the credit card cookie out, it sets the *path* option of the cookie to the directory of the program that actually processes the credit card. Thus, this cookie will only return to the web site when a credit card number is actually needed. An attacker could spoof the web site and trick the user's browser into sending the encrypted cookie. This attack maps to the case where the user's computer is compromised, and will have limited effect as already described.

Furthermore, the user has complete control over their private information after it is used. She can choose when to send a cookie, or even whether she wants to keep this cookie. If the user decides to erase it, then there are plenty of cookie management software that can help her delete the cookies from the local storage. As a result, the keys associated with these deleted cookies will become zombies in the server, and eventually be deleted too.

In summary, the distributed storage of credit card numbers using OTP can provide nearly- perfect secrecy of the user's privacy, and significantly reduce the web sites' liability of the online theft.

4.2. Advantages of OTP's Resource Requirement

For any encryption algorithm, using the same master key to encrypt multiple messages is generally not a good idea, since the more times a key is used repeatedly, the more likely cryptanalysis can be carried out. Using each key just once is a good way to make cryptanalysis improbable. In fact, as long as each encryption key is used only once, any other encryption algorithm (e.g. DES, AES) could also be used to achieve similar secrecy as OTP. However, compared to the other encryption algorithms where multi-rounds of complex logic/mathematical operations have to be performed, OTP has the obvious tremendous performance advantage. OTP is the fastest possible encryption algorithm: the encryption/decryption are both only a simple round of XOR operation on every bit of the plain text.

The main factor that could impact OTP performance is not the encryption/decryption themselves, but the generation of the random keys. The process of generating random numbers is usually far slower than the OTP encryption/decryption itself. Nevertheless, in case this becomes a major issue of the overall real-time performance, web servers can always choose to use pre-computed random number table to save the time of random number generation.

One seeming drawback of OTP is that the keys are as long as the plain texts, and have to occupy the server's storage space that is as large as the plain texts. In addition, since the keys are supposed to be random strings with very high entropy, compression does not work on the keys. However, the cookies that this scheme is encrypting are not large anyway(each cookie can be at most 4K bytes according the specification, but the credit card information is only tens of bytes). The storage requirement of the keys on the modern web servers is very trivial, and should not be a serious issue.

4.3. Randomness of OTP Keys

The perfect secrecy of OTP relies on the randomness of key generation. If the keys are not random, then the algorithm cannot be proven to provide perfect secrecy. True random numbers can be obtained by measuring a random physical process, such as cosmic radiation, or thermal noise. A web server connecting to such a measurement device might seem weird at the first thought. But it is well worth it if the web site is really concerned about the secrecy of its traffic.

There are numerous ways to generate random numbers [7] that a web site can choose from. The hardware and software implementations such as described in [9] are very easy, and a digital Geiger counter package such as described in [1] costs only a few hundred dollars. Therefore going for this kind of randomness should not cause too much trouble to the web site designers.

The other option is to use a good PRNG (Pseudo-Random Number Generator, such as surveyed in [11]), which from an initial state produces series of numbers that appear to be random to an observer. In some cases, using frequent re-seeding, the use of PRNGs can be appropriate. If a PRNG is used, the perfect secrecy of the algorithm cannot be achieved, but all other properties described in this paper can.

Therefore we can be reasonably confident that OTP encryption on cookies is unbreakable. The user's privacy is hence strongly protected, and the web site's liability is substantially reduced by the OTP encryption.

4.4. Resistance against Replay Attacks

Assume that an attacker somehow grabs a customer's cookie. Although the attacker would not be able to decrypt it, she could still send the cookie to a web site a number of times. If the cookie contains a credit card number, this attack could cause unwanted charges on the credit card accounts. Therefore web sites should erase a key from the database immediately after its use. In other words, the web server should generate a different key to encrypt the credit card number every time the web site receives an encrypted cookie from the client. Although this scheme does not solve all problems, an attacker would not be able to replay an old cookie used by a client because the key associated with this cookie will be deleted after the actual user sends the cookie to the server. Replay would only return an error message in this case.

However, the attacker can still succeed on replay attacks by sending a captured encrypted cookie to the web site before the actual user does so. Web sites should be encouraged to keep shipping address information linked to user ids to prevent this kind of attacks. Thus the harm that the attack might bring is to purchase an item that the customer actually does not want, and charge the credit card account. Nevertheless as the shipping address of the customer is not changed, whatever item the attacker purchases would still be delivered to the customer, not the attacker. Moreover, since the usage of each encrypted cookie is valid to only one web site, the attacker would not be able to use this credit card to make purchases on other web sites. In comparison, credit card information without site-specific encryption would allow the attacker to use it everywhere with her own shipping address, making the purchased items delivered to the attacker instead of the owner of the credit card.

4.5. Non-malleability

Other than trying to steal other people's credit card numbers, a mischievous attacker might try to create her own fake credit card cookie and send it to the web server, using an existing index string in an existing cookie, simply to create confusion on the web server. However this attack does not work as long as the attacker does not know the key of the cookie, because the server only gets a random string after decrypting the fake cookie. Since credit card numbers themselves have built-in verification mechanism, there is a chance

that the random string would not turn out to be a valid credit card number. In addition, it is very unlikely that the expiration date will be valid or that the attacker can guess the name of the owner of the card represented by the mentioned random number. Some integrity measures, such as hashing the credit card number, could also be used as an extension of this scheme to further reduce the chance that an attacker can fake valid credit card information.

Notice that repeated attempts of this attack might constitute a DoS (Denial of Service) attack. Preventing DoS attack is out of the scope of this paper, since there are many other ways that DoS attack can be carried out too.

On the other hand, a more successful attack might come from a customer that actually receives her own credit card cookie from the web server. She could compute the key of this cookie from the cookie and her credit card number. And after she gets this key, she could use the key to encrypt somebody else's credit card number to form a new cookie, and send this cookie back to the server. Then the server would charge the victim's credit card account and ship the purchased item to the attacker's address. However the prerequisite of this attack is that the attacker knows the victim's credit card number. And there is no effective way to protect the victim in this case anyway, since the attacker can virtually use the victim's credit card anywhere. Even if our scheme adopts some sort of authentication mechanism to allow the server to reject a received cookie that is different from what is sent out, it still cannot prevent the attacker from registering with victim's credit card information all over again.

4.6. Recovery of Network Connection Problem

In this protocol, if the web server erases every key after its associated cookie comes back, then the customer has to have a good network connection to the web server. Over a bad network connection, the customer's computer might not receive a new cookie from the server when loading a page; as a result the customer would use the old cookie to access the web server next time, and get an error message, since the old key was already removed from the server's key database. Then the customer would have to input the credit card number again.

To ease (but not completely resolve) this problem, the web server can choose not to delete the old key from the database after generating the new key. Instead, the server keeps the old key in the database some time longer. Most users react to a network connection failure by reloading the page in a short time, in which case the old cookie would still be able to find the old key in the database.

Keeping an old key after its use may imply that the protection against replay attack is weakened. However, we have already shown in section 4.4 that the impact of replay attacks can be small as long as the web server takes precaution to link user's shipping address. As in many decisions that take security into consideration, the web server needs to weight convenience against security.

4.7. Same User from Different Computers

A user might want to access the same web site from different computers, typically one home computer and one office computer. As long as the cookies are not synchronized between these computers, the user has to input the credit card information once on each computer the first time she uses that computer to purchase something from a web site. Thus each computer would store the same credit card number encrypted with a different key, and the server would keep the keys for all these cookies at the same time.

Inputting the same thing twice seems to be inconvenient, compared to the currently most used approach where all the credit card numbers are stored on the web site, where a user can log in to the web site from anywhere to make purchases without inputting the credit card repeatedly. However, since users tend to return to the same web site many times to purchase more items, inputting credit card numbers twice in the beginning should not really be a big issue.

5. Conclusions and Future Work

This paper gives a useful, real-world application of secure distributed storage of sensitive information using HTTP cookie encryption. A One-Time Pad method to achieve perfect secrecy of cookie encryption is proposed, based on the observation of the interesting characteristic of cookie encryption: the encryption and decryption are done by the same party. The pros and cons of this protocol are analyzed, and the comparison with the existing approaches shows that our protocol is able to protects users' privacy and reduce web sites' liability in a much stronger manner.

We have implemented this protocol as a C library, and designed a simple web site to demo the OTP cookie encryption. The library implements the key database based on GDBM, and the encryption and decryption functions can be called from the main CGI programs of a web site directly.

As future work, on one hand we intend to improve and expand this cookie encryption library and try to apply it to real world web sites; on the other hand we would like to apply similar OTP protocols to the other online privacy protection problems, where the party that repeatedly uses the sensitive information cannot or does not want to store the information locally.

One example of this kind of problem is network access proxy/anonymizer, where the proxy needs the IP of a file-request's originator, in order to return the file to the originator when the proxy gets the file; but at the same time the proxy also wants to protect the originator's privacy, and is not willing to store the originator's IP locally. Applying OTP to this problem might be especially helpful to peer-to-peer network's privacy protection.

Another example is secure remote file storage, where the user needs to download a file from a server to a PC to access it, and for privacy concerns the user does not want the server to store plain text of the file. We will explore the possibilities to use the OTP technique to address this issue in Microsoft's .NET service and other similar services.

References

[1] Internet Website, "Gamma-Scout," http://www.gammascout.com/, 2002.

[2] Margaret Kane, "Online spending to hit $65 billion," CNet Online Article: http://news.cnet.com/news/0-1007-200-5794394.html?tag=lh, 2001.

[3] D. Kristol and L. Montulli, "Internet RFC 2965: Http state management mechanism," Network Working Group, October 2000.

[4] Microsoft, "Passport Homepage," Online at http://www.passport.com/, 2001.

[5] Joon S. Park and Ravi Sandhu, "Secure Cookies on the Web," *IEEE Internet Computing*, July-August 2000, http://computer.org/internet, 2000.

[6] Reuters, "Fraud threat still haunts Net shoppers," CNet Online Article: http://news.cnet.com/news/0-1007-200-6270593.html, 2001.

[7] Terry Ritter, "Random Number Machines: A Literature Survey," Online at http://www.io.com/~ritter/RES/RNGMACH.HTM, 1997.

[8] C.E. Shannon, "Communication Theory of Secrecy System," *Bell Systems Technical Journal*, 28:656-715, 1949. Online version: http://www.cs.ucla.edu/~jkong/research/security/shannon.html

[9] John Walker, "HotBits: Genuine random numbers, generated by radioactive decay," Online at http://www.fourmilab.com/hotbits/, 1999.

[10] G.S. Vernam, "Cipher printing telegraph systems for secret wire and radio telegraphic communications," *Journal of American Institution of Electronic Engineering*, 45:252-259, 1926. Online introduction: http://www.pro-technix.com/information/crypto/pages/vernam_base.html

[11] Terry Ritter, "The Efficient Generation of Cryptographic Confusion Sequences", *Cryptologia*. 15(2): 81-139. 1991.

[12] Stuart McClure and Joel Scambray, "Etrade makes the 'hit' parade: Client-side hacking captures a big-time victim," *InfoWorld* Online Article: http://iwsun4.infoworld.com/articles/op/xml/00/10/09/001009opswatch.xml, 2000.