SECURE AUDIT LOGGING WITH TAMPER-RESISTANT HARDWARE

Cheun Ngen Chong, Zhonghong Peng and Pieter H Hartel
Universiteit Twente, Enschede, The Netherlands.
{chong, zhong, pieter}@cs.utwente.nl

Abstract Secure perimeter schemes (e.g. DRM) and tracing traitor schemes (e.g. watermarking, audit logging) strive to mitigate the problems of content escaping the control of the rights holder. Secure audit logging records the user's actions on content and enables detection of some forms of tampering with the logs. We implement the Schneier and Kelsey's secure audit logging protocol [6], strengthening the protocol by using tamper-resistant hardware (an iButton) in three ways: Firstly, our implementation of the protocol works offline as well as online. Secondly, we use unforgeable timestamps to increase the possibilities of fraud detection. Lastly, we generate the authentication keys, core security of Schneier and Kelsey's protocol on the iButton to alleviate the possibilities of malicious client generating the bad keys. We provide a performance assessment of our implementation to show under which circumstances the protocol is practical to use.

1. Introduction

Digital content is so easily distributed, and dissociated from the metadata that describes owner, terms and conditions of use etc. that copyright infringement is rife. Secure perimeter schemes such as digital rights management (DRM) alleviate the problem in some cases [2] but most (if not all) DRM systems are vulnerable to attacks. The raw content can then be redistributed, severely damaging the interests of the rights holder. Tracing traitor schemes trace leaks of content to the users who can be identified, and ultimately whose behaviour can be recorded as evidence. Many techniques exist to rediscover the identity and thence the rights on the content, such as cryptography, digital fingerprinting, watermarking etc. In this paper, we assume that users can be identified, and we concern ourselves with the issue of gathering information on the user's behaviour.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35691-4_52
D. Gritzalis et al. (eds.), *Security and Privacy in the Age of Uncertainty*

Secure audit logging records the actions of a user on an item of content and does so in a manner that allows some forms of tampering with the log to be detected. We implement the Schneier and Kelsey secure audit logging protocol [6], using tamper-resistant hardware (TRH). For brevity in the sequel, we refer to Schneier and Kelsey [6] as "SK".

An audit log is an important tool to detect and to comprehend damages of a computer or network system caused by intrusions, defects or accidents. An audit log contains descriptions of noteworthy events. In our DRM experiment audit logs are generated in the user's personal computer (PC). The PC is a hostile environment (untrusted domain) because of its vulnerability against various malicious attacks. Therefore, the audit logs require protection to ensure integrity.

SK involves two parties: an untrusted machine and a trusted machine. The untrusted machine is not physically secure or sufficiently tamper-resistant. SK makes the audit logs survive the adversary's attacks. In other words, SK renders audit logs impossible for an adversary to *undetectably* view, forge and delete even after the untrusted machine is compromised by the adversary. Furthermore, the audit logs record all the actions performed by the adversary, including her attempts to compromise the untrusted machine.

The comment by SK that "the trusted machine may typically be thought of as a server in a secure location, or implemented in various ways, which includes a tamper-resistant token" has inspired us to use tamper-resistant hardware (TRH) as the trusted machine for secure logging. The TRH (or the trusted machine) we use is a Java iButton (*www.ibutton.com*) for the following reasons:

1 The iButton contains a programmable tamper-evident real time clock. The real time clock keeps time in $\frac{1}{256}$ second increments.

2 The iButton supports efficient implementations of common cryptographic algorithms.

3 The iButton version 1.1 provides up to 6kB non-volatile RAM, the more expensive version 2.2 contains approximately 134kB non-volatile RAM.

We use the iButton as a trusted device to aid in the audit log creation in the manner proposed by SK. An iButton is too small to store a log of any useful size in a cost effective manner: A typical PC contains 40GB storage, i.e. around 300,000 times more than the iButton.

Our DRM system has the usual Client/Server architecture. The Client is a user with her PC, which represents the untrusted domain. The Server is a trusted environment where content and license are stored.

When the Client accesses the content piecemeal from the Server (e.g. by streaming), the latter is able to protect the content to some extent because the Client's actions can be monitored. However, when the Client downloads the content to the PC's non-volatile storage to accesses the content offline (i.e. disconnected from the Server), the Server is not able to monitor the Client's behaviour. We propose using secure audit logging with TRH to bring the security of offline DRM to the level of online DRM. The main contributions of this paper are:

1 To implement SK embedded in several auxiliary protocols and with the iButton, to support security of offline DRM.

2 To evaluate the performance of the implementation; thus investigating whether the iButton can be used effectively.

3 To strengthen SK by making sure that some of its security assumptions are valid by virtue of using the iButton. We generate core secrets and timestamps on the iButton instead of the untrusted PC.

To the best of our knowledge ours is the first attempt to implement SK and the first endeavour to analyse the performance of SK in general, and SK with iButton in particular.

A weakness of any system, which relies on TRH to coerce an untrusted Client into specific behaviour, is that the user may simply sever the connection between the Client and the TRH. We suggest a number of ways to discourage the Client from such behaviour: (1) The Server is designed so that it insists on the iButton being present to authenticate and authorize the Client; (2) Organizational policy (e.g. in a corporate intranet) is used to enforce the use of the iButton. In both cases users are provided an incentive to maintain communication with the iButton: *no iButton means no content.*

The main problem with audit logging is that at some stage a dishonest user may cheat by disabling the audit logging functionality of the DRM application at the Client. She is able to deny any actions she has performed during the offline period without evidences in the log.

At this point, the protection offered by DRM is weak on PCs but potentially stronger on consumer electronics appliances. The reason for this weakness is that PCs are open and programmable, whereas Consumer electronics appliances are more tamper-resistant than PCs and therefore somewhat more difficult to hack the DRM application. Therefore, the dishonest user will find it much harder to bypass a CE device (the Client) audit logging mechanism. The audit logging mechanism is

able to record all the user's actions, including her attempts to tamper with the audit logs.

The remainder of this paper is divided into sections as follows: Section 2 describes related work. Section 3 explains SK using the iButton. Section 4 discusses concisely the refinement we have made of SK. Section 5 gives our performance analysis on the implementation. Finally, section 6 concludes this paper also mentioning future work.

2. Related Work

Shapiro and Vingralek [8] survey several mechanisms to manage the persistent state in a DRM system, including protected digital content, audit trail, content usage counts and decryption keys. One of the mechanisms they mention is secure audit logging. The two secure audit logging methods they cite are Bellare and Yee as well as Schneier and Kelsey.

Bellare and Yee [1] (BY) propose a scheme to construct audit logs which possess the forward integrity property: The keys are altered on a regular basis by feeding different secret values to a family of pseudo-random functions that generate the MACs over the entire log entries. If an adversary is able to compromise the current MAC key, it is unfeasible for her to deceive the historical entries generated because she is not able to fabricate the MAC keys for previous log entries. "Forward integrity" can be viewed as an integrity analogue of "forward secrecy" [5]. A protocol possesses the forward integrity property if a compromise of long-term keys does not compromise the past keys. BY maintains the audit logs on untrusted machines.

Schneier and Kelsey [6] (SK) uses a linear hash chain [4] to link the entire audit log entries so that some forms of tampering can be detected. The hash-chain is constructed by hashing each previous hash value of each log entry, concatenating with some other values. SK provides a complete secure audit logging protocol, from the log creation to log verification and viewing. A trusted machine is needed during log file creation but not at every log entry creation. The untrusted machine needs to communicate with the trusted machine from time to time to create new log file and to validate log files. Similar to BY, SK shares the "forward integrity" property, but SK uses a collusion-resistant hash function to generate the keys for MACs generation of each log entry.

SK and BY share a security weakness. If an adversary is able to compromise the untrusted machine at time t, i.e. obtains the key at time t, she is able to forge the log entry at time t. SK and BY are able to make illicit deletion of audit logs detectable. However, they are not able to *prevent* unauthorized removal of complete audit logs. Both

SK and BY reckon that the deletion of log entries cannot be prevented by using cryptographic methods, but only by using write-only hardware such as CD-ROM, or paper printout.

3. The Protocols

Figure 1 illustrates the protocols in our secure audit logging method. SK1 and SK2 are from SK. The others, P1 and P2 are our own protocols.

SK1 creates and closes an audit log. We focus on the technical details of the protocol and refer the reader to Reference [6] for the motivations behind the protocol and other details. We use the notations listed in Reference [6] to describe our protocols.

SK2 verifies and displays the audit logs to the Verifier. We have changed slightly the SK Verifier in that it does not store the Client's log file locally. The Verifier reads and verifies the log file remotely from the Server. In other words, the log file is stored securely in the Server, and the cryptographic processes are operated at the Server. Similar to SK1, Reference [6] elaborates this protocol.

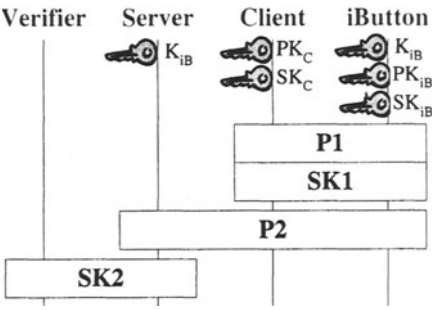

Figure 1. Overview of the secure audit logging method.

The Client and the iButton own different sets of key pairs. PK_C and SK_C are the public key and private key of the Client, respectively. PK_{iB} and SK_{iB} are the public key and private key of the iButton, respectively. The iButton and the Server share a secret key, K_{iB}. The public/private keys and the shared secret key are preloaded on the iButton before it is deployed.

We have made some assumptions that we believe to be reasonable while implementing the protocols. The main reason is to facilitate the implementation of SK in the resource constrained iButton.

1 Without restricting generality, there is only *one* Verifier, *one* Server, *one* Client and *one* iButton in the audit logging process. There is only *one* audit log file created from the process.

2 If the iButton is removed from the iButton reader halfway through an instruction then the log file is closed abnormally with a reason stated in the log.

3.1 P1

P1 generates the authentication keys, A_j and to generate the timestamp, d_j for recording the log entry. We have refined SK1 by deciding that the iButton should generate the keys and timestamps, as can be seen in Figure 2.

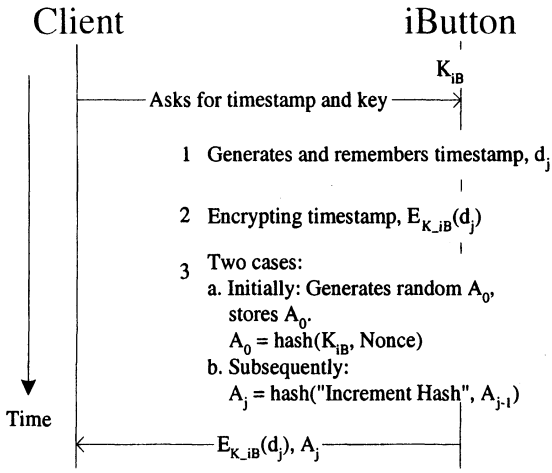

Figure 2. The P1 protocol for generating the initial authentication key A_0 and timestamp d from the iButton.

We improve SK1 to the extent that A_0, which represents the core security of SK1, and the timestamps are generated in a trusted subdomain. A fresh nonce, *Nonce* is generated and stored on the iButton. The purpose of *Nonce* is to ensure the freshness of the initial authentication key. The generated nonce is concatenated with the key, K_{iB}. Thereby, the initial authentication key cannot be generated by a malicious Client because she does not know the *Nonce* and K_{iB}.

We use the iButton real-time clock to generate the timestamps. We encrypt the timestamps generated using the key, K_{iB} shared between the iButton and the Server. By doing so, the timestamps cannot be manufactured by the Client (who does not have access to K_{iB}).

The Client first requests an encrypted timestamp and an authentication key from the iButton for the current log entry. The iButton generates the timestamp using its real-time clock and encrypts the time stamp with the iButton secret key, K_{iB}. The iButton then remembers the first timestamp, i.e. the timestamp for the initialisation log entry, with type *"LogFileInitializationType"* and also the last timestamp, for the close log entry, with type *"NormalCloseType"* or type *"AbnormalCloseType"*. We will come back to this in section 4.

The iButton then generates a random key, as the initial authentication key, A_0 if it is the first log entry the Client constructs; otherwise, the iButton hashes the previously existing authentication key, A_{j-1} concatenated with a message to generate the next authentication key, A_j. The iButton stores the initial authentication key, A_0 and current authentication key, A_j for generating subsequent keys, A_{j+1}. After finishing the generations, the iButton sends the encrypted timestamp and the authentication key back to the Client.

3.2 P2

P2 synchronizes the iButton clock to the Server clock, which is a trusted clock. P2 also sends the log file maintained at the Client and the corresponding initial authentication key on the iButton to the Server, as shown in Figure 3. The Server and the iButton share a secret symmetric key, K_{iB}, which is used to encrypt the data exchanges between the iButton and the Server (and also the timestamps) from the Client.

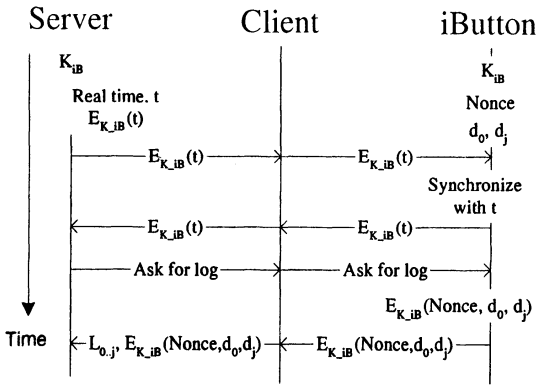

Figure 3. The P2 protocol of synchronizing the iButton real-time clock and sending audit logs to the Server.

The Server encrypts its current time with K_{iB}, and sends the encrypted time to the iButton for time synchronization. The iButton then

decrypts the message and adjusts its real time according to the received time. Once the time is synchronized, the iButton sends back the accepted encrypted time to the Server as an indication that the time is synchronized.

After the clock synchronization, the Server sends a request message to the Client asking for the available log file. The Client forwards the request to the iButton. The iButton sends the encrypted *Nonce* (the secret used to generate the initial authentication key), as well as initialisation timestamp, d_0 and close timestamp, d_j to the Server via the Client. At the same time, the Client sends the available log file (corresponds to the ID_{log} of *Nonce*) to the Server. In the DRM system, P2 is transparent to the Client.

4. SK Refinement

We have refined SK by introducing two auxiliary protocols, P1 and P2. P1 lets the trusted iButton instead of the untrusted client generate the authentication keys and the timestamps. The iButton remembers the timestamps for initialization and closing of the log. Additionally, the Client only possesses the encrypted timestamp from the iButton for audit logging, i.e. the integrity and confidentiality of the timestamps are ensured. P2 enables the iButton to transfer the encrypted initial authentication key and stored timestamps using the shared secret key with the Server. In other words, P2 is able to guarantee the integrity of the initial authentication key and the timestamps during the transmitting process.

As pointed out by Schneier and Kelsey [6], [7], there is a security weakness in SK. If an adversary is able to compromise the Client by getting hold of the key A_t, directly after it has been generated at time t, the adversary is able to falsify the log entry at time t. The adversary is also able to create more counterfeit log entries and remove some log entries. The Verifier is not able to detect the frauds because the adversary is able to construct another "truthful" hash-chain and MAC over the entire log entries using the compromised authentication key.

In our refinement of SK, by using encrypted timestamps, we are able to detect some of the aforementioned frauds. If the adversary wishes to create more log entries, she has to obtain the cooperation of the iButton to generate valid timestamps. The adversary does not have the right key, so she cannot fabricate arbitrary timestamps herself. The adversary can reuse genuine encrypted timestamps, but the verifier will notice missing or duplicate time stamps, or time stamps that are presented out of order. The adversary will also be caught truncating the log file when the time of

truncation does not match the time of the last transaction remembered by the iButton. In case of tampering, the user can be held responsible for the entire period between the log initialization time and close time.

We now present a concrete example of the difference between the original SK and our version as shown in Figure 4. An adversary, who owns a protected document and an associated license, starts the content renderer on the Client. The first log entry, initialization log, is then generated. Subsequent log entries are generated when she turns to other pages of the document. At time t_4, the adversary successfully steals the key A_4. She does not stop browsing the document, but keeps reading until time t_{10}. She closes the document at time t_{11} and so the log file is closed. She wants to deny the fact that she has viewed the document from t_4 till t_{10}. Instead the adversary wants the Verifier to believe that she has viewed the document until t_3. The adversary would wish to do so for example when she is charged on a pay-per-view basis. The adversary thus removes the log entries from t_4 onwards, and creates a false close log entry using timestamp t_4. As she possesses the key A_4, she is able to construct a valid hash-chain and forge the MACs for this fraudulent log. In our version of SK, the Verifier is able to detect the forgery because the iButton remembers the log closing time (t_{11}), which in the scenario above does not match t_4. The original SK protocol does not detect this situation.

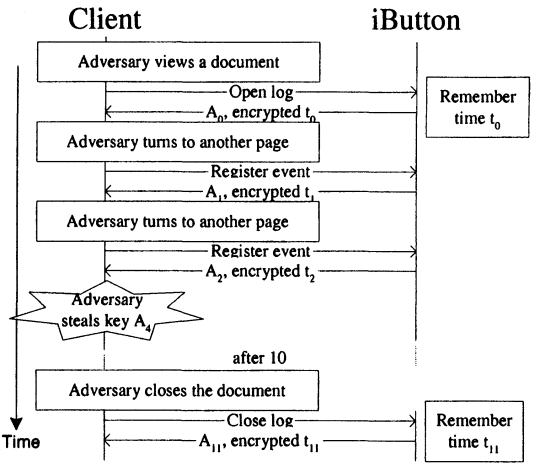

Figure 4. An adversary views a protected document and steals the key at time $t = 4$, during the logging process.

Suppose that the Client cheats by using key A_4 instead of A_{11} to close the log file. The next time the client connects to the server to get

new content she will need the cooperation of the iButton to authenticate herself. This gives the iButton the chance to present the latest key A_{11} to the server, and so the cheating Client will be found out.

Using encrypted time stamps improves the security of the protocol but weaknesses remain. For example if after a perfectly legitimate run of the protocols the user starts viewing the content using an application that does not log any actions, then this will not be noticed.

5. Performance Analysis

We are interested to know how the iButton affects the performance of the Client during audit logging. This allows us to determine if the iButton is practical for secure audit logging in a DRM system. We have run several performance tests on the implementation. We have used a PC machine Pentium III 800MHz with 256MB RAM as the Client machine for performing the tests.

We have measured the time for creating different numbers of log entries, from 1 to 10 on the Client. The graph is nearly a straight line. Generating 1 log entry takes approximately 1 minute.To explore why it takes roughly 1 minute to generate only 1 log entry, we have measured the time spent for performing cryptographic operations on the iButton. We believe that the cryptographic operations are the main causes to the long time taken for generating log entries.

We read the start time on the iButton right before the iButton starts the calculation under investigation. We read the stop time on the iButton once it stops the process. The time taken is the value of deducting the start time from the stop time. The result is then transmitted back to the Client. As only time with seconds-precision is available on the iButton (the software does not provide access to the $\frac{1}{256}$ second accuracy of the hardware clock), we have run the process repeatedly on the iButton, dividing the time measures by the number of repetition. We take the average value of 20 measurements as our final value using the standard deviation as error margin.

We evaluate the time spent for encrypting/decrypting a message of various size in bytes (from 8 to 128) using the 64-bit key DES algorithm on the iButton. We realize that the times are around 200 ms for large message (size bigger than 128 bytes). We measure the time spent for hashing a message of sizes range from 8 to 128 bytes using the SHA1 message digest algorithm. The time spent for hashing 56 bytes is almost double the time spent for hashing 48 bytes. This is due to the message padding to 64 bytes [3]. We also measure the time consumed for encrypting a message of sizes from 8 bytes to 128 bytes, using 128-bit public key

of RSA algorithm; and decrypt using the corresponding private key on the iButton. The RSA encryption takes averagely 25 seconds, while 22 seconds are needed for RSA decryption. We measure the time needed to sign a message using SHA1 and RSA and to verify the signed message. It takes 4 to 5 seconds to sign a message on the iButton, but it takes 5 to 6 seconds to verify the signature.

We do a back of the envelope calculation to confirm the measurement on the log entry generation time we obtain. The iButton takes less than 1 second for generating the timestamp and authentication key, i.e. to complete the protocol P1. The cryptographic operations on the iButton, as mentioned earlier consume most time. RSA private key decryption and public key encryption take approximately 24 seconds and 20 seconds, respectively. DES encryption and decryption need about 0.1 second, respectively. Signature generation and verification require roughly 5 seconds and 4 seconds respectively. These values are taken according to the size of the message we have used.

To conclude our performance analysis, as it takes about 1 minute just to generate 1 log entry, the iButton is not practical to be used in a system that requires frequent logging. However, if the system only logs the main events, such as playing a 4-minute song, reading an eBook, and other content access taking a longer time, we believe that the iButton is practical. Note that in our system logging overlaps with the actual content rendering.

For logging frequent events, we believe that we could use iButton as a bootstrap device for ensuring the trustworthiness of the first audit log entry, and we could do the subsequent log entries creation for frequent events without the presence of the iButton. This issue remains open for future investigation.

6. Conclusions and Future Work

We propose using secure audit logging in a DRM system where the Client is not permanently online. This allows the Server to obtain knowledge of the Client's behaviour during offline periods. We implement the Schneier and Kelsey (SK) secure audit logging protocol, using tamper-resistant hardware (an iButton) as the SK trusted machine.

The performance evaluation reveals that it takes about 1 minute for generating 1 log entry. We reckon that this is not practical for a system that requires frequent logging but feasible for a system that only needs to log the main events such as playing a 4-minute song. To make the iButton implementation also practical for recording more frequent events in future work we intend to use the current implementation recursively:

one entire log on the untrusted PC would then correspond to one log entry that involves the iButton. The performance could also be improved dramatically using a more powerful iButton.

The main problem with all secure audit logging protocols is that if a log entry at time t is compromised, then none of the log entries after time t can be trusted. Our use of securely encrypted time stamps can solve this problem in some (but not all) cases. We believe that we can improve the security further by asking the iButton to do a little more work. This remains open for future work.

Acknowledgement

We like to thank Dr. Jaap-Henk Hoepman for his valuable comments on this paper.

References

[1] Mihir Bellare and Bennet S. Yee. Forward integrity for secure audit logs. Technical report, UC at San Diego, Dept. of Computer Science and Engineering, November 1997. http:// citeseer.nj.nec.com/ bellare97forward.pdf.

[2] Cheun N. Chong, René van Buuren, Pieter H. Hartel, and Geert Kleinhuis. Security attribute based digital rights management. In *Joint Int. Workshop on Interactive Distributed Multimedia Systems/Protocols for Multimedia Systems (IDMS/PROMS)*, pages 339–352. Springer-Verlag, Berlin, November 2002.

[3] FIPS-PUB-180-1. Secure hash standard. Technical report, US Departmenet of Commerce/NIST, Washington D. C., United Stats, April 1995.

[4] Leslie Lamport. Password authentication with insecure communication. In *Communications of the ACM*, volume 24, pages 770–772. ACM Press, November 1981.

[5] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, chapter 12. CRC Press, 2001. ISBN: 0-8493-8523-7.

[6] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *The 7th USENIX Security Symposium Proceedings*, pages 53–62. USENIX Press, January 1998.

[7] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. In *ACM Transactions on Information and System Security*, volume 2, pages 159–176. ACM Press, May 1999.

[8] William Shapiro and Radek Vingralek. How to manage persistent state in DRM systems. In *Proceedings of the ACM Workshop in Security and Privacy in Digital Rights Management*, November 2001. http://www.star-lab.com/sander/spdrm/papers.html.