Secure Logging of Retained Data for an Anonymity Service

Stefan Köpsell, Petr Švenda

Secure Logging of Retained Data for An Anonymity Service

Stefan Köpsell[1] and Petr Švenda[2]

[1] TU Dresden, Germany, <*sk13@inf.tu-dresden.de*>
[2] Masaryk University, Czech Republic, <*svenda@fi.muni.cz*>

Abstract. The recently introduced legislation on data retention to aid prosecuting cyber-related crime in Europe also affects the achievable security of systems for anonymous communication on the Internet. We have analyzed the newly arising risks associated with the process of accessing and storage of the retained data and propose a secure logging system, which utilizes cryptographic smart cards, trusted timestamping servers and distributed storage. These key components will allow for controlled access to the stored log data, enforce a limited data retention period, ensure integrity of the logged data, and enable reasonably convenient response to any legitimated request of the retained data. A practical implementation of the proposed scheme was performed for the AN.ON anonymity service, but the scheme can be used for other services affected by data retention legislation.

1 Introduction

The recently introduced legislation on data retention affects—at least in some countries (e.g., Germany)— systems for anonymous communication on the Internet such as AN.ON [BeFK00] or TOR [DiMS04]. These systems alter the source IP-addresses of users and these alterations should be logged and accessible on request from legal authorities (cf., [BeBK08] for a description of the legal obligations and the usefulness of the retained data).

Standard secure logging mechanisms such as [MaTs09] protect the logged records sufficiently against unauthorised access (confidentiality), unauthorised modification (integrity) and in some cases attempt to ensure availability of records. But when applied to the needs of data retention logging on the logging entity side, newly arising risks remain unsolved as the attacker model has changed. Potentially sensitive data are present on logging entity side as a result of compliance with data retention legislation. The logging entity can be forced to reveal, delete or modify this data—threats that did not exist before as there was no need to store such data in the first place. Specifically, threats related to the data retention period must be addressed and mitigated.

Note that the risk for a user to be deanonymised, if the operators of the chosen anonymity servers behave *dishonestly*, exists before the introduction of data retention. But if the operators are *honest*, the attacker gains an additional advantage of mounting a successful attack on the anonymity of a given user

with the help of retained data. Moreover, it is now possible for the attacker to start his attack *after the fact* (i.e. after the activity, the attacker wants to deanonymise took place). This was not possible before, as the attacker had to log the anonymised and encrypted traffic at the time of this activity in order to analyze it later on.

Completely new risks arise from the fact that the logged data is used for law enforcement. One such attack results in the risk of the attacker modifying the logged data so that an innocuous user of the anonymity service becomes suspicious. For an operator of an anonymisation server, the new risk is that an attacker forces him to modify the logged data in such a way (or at least in a way which hides the criminal activities of the attacker). So, our solution should not only protect the users of the anonymity service but also its operators.

A demand for the practical implementation originates from the needs of the AN.ON anonymity service. This anonymity service has been open to public since 2000 and has to fulfil the legal obligations given by the data retention legislation. But the proposed logging scheme can be used for other services affected by the data retention legislation as well. More generally, the scheme can be used for any logging service where the logged records are accessible only for a limited time period or where knowledge of cryptographic secrets might lead to personal threats of the holder.

Our paper is organised as follows: the first section describes the requirements for data retention logging and summarizes related work. The second section describes the logging scheme and analyzes security of the scheme. This section also provides an overview of the steps involved in logging and answering requests. Selected properties of practical implementation and results of the performance analysis are given in section three, followed by conclusions in section four.

1.1 Legal and operational requirements on logging of retained data

In this section, we summarize the requirements for the retained data and the logging procedures. These are general requirements applicable to any service which needs to be compliant with the EC data retention directive. They can be derived from the legal obligations (R1–R4) and the operational needs (R5). Moreover, they can be classified as functional requirements (R1; what the system should *do*) and non-functional requirements (R2–R5; how the system should *be*).

R1: Logged data has to include all statutory categories of data. Article 5 of the data retention directive describes what types of services have to retain which data categories. National implementations of the directive could extend this. This functional requirement basically states that some meaningful data has to be logged and that logging of (e.g.) random data would not be sufficient.

R2: Logged data have to be deleted after a specific period of time. This means that logged records cannot be accessed outside a given data retention period. In the following text we use the term "outdated" to describe a property of a given item (cryptographic key, log entry etc.) to which the access should be prevented because the related retention period already expired.

R3: Logged data need to be accessible, so that requests from law enforcement agencies can be answered without undue delay.

R4: Logged data have to be secure, so that no access to the logged data by unauthorised person is possible. This requirement covers confidentiality as well as integrity of the logged records. Note that in our case the integrity means that the operator can detect if the logged data have been altered—it is not necessary that the operator proves something to the third party.

R5: The cost of logging has to be reasonable. It includes the monetary costs (e.g. initial necessary investments, operational costs) but also the degradation of the overall performance of the system as well as the organisational overhead.

1.2 Related work

Mechanisms for secure logging were previously described in the literature, e.g. [BeYe97, ScKe99, Acco05, Holt06, WSBL08, MaTs09]. One of the presented ideas was the use forward-secure MACs to protect the integrity of log entries. The use of hash chains ensures *forward integrity*, which means that any alterations of the log entries stored *before* the system was compromised could be detected.

The common idea of all of the mentioned schemes is to divide the timeline into several epochs. All log entries which belong to the same epoch are protected by the corresponding epoch key. Once the epoch is over, the key of that epoch is destroyed and a new one is generated for the next epoch. Usually the so-called key evolution scheme is used to derive the next key from the current one. Normally, one way function is used for key evolution. Thus, it is hard for an attacker who knows the key of the current epoch to calculate a valid key of any previous epoch.

But as analysed in [MaTs09] the systems described in [BeYe97, ScKe99, Holt06] suffer from a so-called *truncation attack*—"a special kind of deletion attack, whereby the attacker deletes a contiguous subset of tail-end log entries." The idea of using hash chains for log file protection (used in [ScKe99, Acco05, Holt06, WSBL08]) is patented (US patent 5978475).

The solution presented in [MaTs09] is based on the Forward-Secure Sequential Aggregate (FssAgg) authentication techniques. The key component of the FssAgg scheme is the sign-and-aggregate algorithm. This algorithm—which can be seen as a substitution of the forward-secure MACs used in other secure logging schemes—takes as an input the private key, certain data to be signed and the aggregate signature generated so far. It computes a new aggregated signature which covers the given input data and a new private key which is used for generation of the subsequent aggregate signature. The performance comparison of the various FssAgg schemes given in [MaTs09] demonstrates that even the fastest scheme still needs 5.55 ms for signing a single log entry. This means that the overall performance of our anonymity system would be significantly diminished.

2 The proposed scheme for secure logging

The following roles are represented in the scheme:

1. Mix **operator** – is responsible for general maintenance of Mix server(s) and logging required traffic data into protected log files. Mix operator does not need to be able to access content of the log files afterwards.
2. Law enforcement agency **officer** – will issue the data retention request backed up by court order. The usual procedure is to issue the order and receive the response in the plaintext. One cannot assume that the law enforcement agencies can easily change such procedures e.g. integrate new cryptographic mechanisms.
3. Data retention request **responder** – the entity responsible to collection and accession of protected log files, search for entries relevant to particular data retention request and responding to law enforcement officer. Serves as communication party for an officer.
4. External **storage(s)** – responsible for the keeping of the log files with redundancy required to provide the reliable backup and integrity protection.
5. Trusted **time source(s)** – responsible for providing the current date and time for the decision process about data retention period validity.

As the responsibility of Mix operator is only to keep logging software running and is usually the same person as the data retention requests responder, we will refer to both simply as an *operator* in the following text.

2.1 General assumptions and settings

We have developed a secure logging scheme primarily for our anonymity service called AN.ON. which is based on Mixes. A Mix [Chau81] is a server which forwards messages thereby ensuring that an outsider (e.g. an eavesdropper) cannot link incoming and outgoing messages. This is accomplished by a combination of several (cryptographic) mechanisms. In order to enhance the trustworthiness of the anonymity system, several Mixes are chained together. The sender of a given message can only be deanonymised if all Mixes along the path of his message reveal the linkage between the appropriate incoming and outgoing messages. Therefore, the use of multiple Mixes offers some protection against the dishonest Mix operators.

One can imagine our anonymisation service described below as a simple proxy which a user uses to hide its own IP-address, e.g. towards a Web-Server. Therefore, the proxy exchanges the IP-address of the user (IP_U) with its own IP-address (IP_P). This alteration of the source IP-address (together with a timestamp t) has to be retained (cf. requirement R1). For simplicity, we assume that the IP-address of the proxy will change rarely so that it is not necessary to store it with every log entry. Finally each log entry can be seen as a pair of IP-address and timestamp (in our example: (IP_U, t)). Multiple log entries are stored within one log file.

In addition to the Mixes, which are the logging servers generating and storing the logged data, two other parties are relevant to our setting: the Mix operators and law enforcement agencies. Mix operator is a legal or natural person responsible for the operation of a given Mix, and for the implementation of data retention, which include includes answering the requests for retained data by the law enforcement agencies.

As IP_P of the proxy will be visible in suspicious requests, the law enforcement agencies ask questions in the form of: "Who was using IP-address IP_P at time t_R". In order to answer such questions we need to search through our log files for all records with timestamps t_i for which: $t_R - \epsilon \leq t_i \leq t_R + \epsilon$. The need for the parameter ϵ reflects the fact that we cannot assume that all clocks of all servers are synchronised. The specific value of ϵ is usually given by law through technical regulations.

In order to facilitate the search process, log entries are stored and organized according to increasing timestamps t_i. This is also the natural order they were generated by the proxy.

We decided that not all log entries should be stored within a one single log file but rather multiple log files should be generated. In our case we store one log file per day. The reasons for this are twofold. On one hand, storing log entries in multiple files would simplify the process of deleting of outdated log entries. On the other hand, we propose that a dedicated machine, which has no connection to any communication network should be used for the processing of the law enforcement requests. Therefore the stored log file related to the timestamp in question needs to be transferred to that machine. This in turn would result in an overwhelming overhead if all log entries are stored within a single file.

The logged data has to be stored encrypted and integrity protected (cf. requirement R4). The encryption ensures that the content of the logged data can not be revealed without the knowledge of the secret key. Of course this is only true, if the server which logged the data was not compromised at the time of the data logging. The advantage of encrypting the logged data is that the data can be protected using available (probably insecure) backup mechanisms. Note that because of this backup, it is in generally not possible to (provably) delete the retained data. So the "deletion" has to be accomplished by cryptographic means (e.g., by destruction of a decryption key[3]).

2.2 Confidentiality

Confidentiality can be achieved by either symmetric or asymmetric encryption[4]. Asymmetric encryption has the advantage that no secret key needs to be stored

[3] Deletion of a single decryption key, which is not part of any backup, is much easier compared to ensuring that every backup copy of a given log file is deleted. This is especially true if the backup in place is not under full control of the operator of the anonymisation server itself. This in turn is the usually setting in dedicated hosting service scenarios.

[4] Basically we could also use tamper resistant trusted devices. But as such devices which are able to store large amount of logged data are not available for reasonable price to the operators of our anonymity servers we do not consider them here.

on the logging server but suffers from poor performance compared to the symmetric encryption. The use of symmetric cryptography leads to the problem where the secret key used for the encryption becomes vulnerable to attack. If the same key is used for the encryption of multiple log entries the attacker might be able to decrypt log entries generated *before* gaining control over the logging server.

As a compromise, we utilize a hybrid encryption scheme where the symmetric encryption is used for the log entries itself. The corresponding symmetric key k is stored within the log file using asymmetric encryption.

For efficiency reasons, we use an authenticated encryption scheme for symmetric encryption, AES-128 in Galois/Counter Mode (GCM). GCM [GrVi05] is a combination of counter mode for confidentiality and universal hashing (based on polynomial operation in the finite field GF(2^n)) for integrity protection. GCM is one of the NIST approved modes of operation [SP 800-38D] and is part of many (Internet) network protocols. GCM offers very good performance and is believed to be patent free.

We use the position of a log entry within a log file as initialisation vector (counter) for GCM. This allows random access to log entries based on their position within the log file. Thus, it is not necessary to decrypt the whole log file while answering a request of a law enforcement agency.

The same symmetric key is used for all log entries of a given log file and for every log file a new symmetric key is generated. Thus our "epoch" (cf. Section 1.2) is related to a single log file. As mentioned before, the key itself is encrypted using an asymmetric algorithm. Only the operator of the proxy is in possession of the private key. Of course, the keys for the asymmetric scheme change from time to time—but they cannot be changed to often (e.g. on a daily basis) due to the organisational overhead implied by the necessary key management.

Note that because we use the same symmetric key for a whole log file and generate log files on a daily basis, an attacker which compromises the logging server just before midnight might get the knowledge of the symmetric key for that day and thus compromise the confidentiality and integrity of the related log file. One way to mitigate such risk is to generate new log files more frequently. But if the attacker is smart enough to compromise the system and read out the symmetric key from the somewhat protected main memory, he is very likely to be smart enough to hide his traces. Thus the fact that the machine was compromised might be detected only after weeks or even months—making the protective advantage of more frequently generated log files negligible.

We decided to store the private key on a trusted device. Here, trusted device is a device able to control access to the private key. Note that we use this trusted device not only to prevent unauthorised access of third parties to the private key. Additionally, the access to outdated log files (cf. requirement R2) is prevented and the risk that the operator is forced by the attacker to decrypt outdated log files is mitigated by the use of trusted device usage. Therefore, an important property of the access control to the private key implemented by the trusted device is, that it not only depends on proper authorisation (e.g. password

of the operator) but *on the current time*. The idea is, that the trusted device denies decryption of a symmetric key if the related log file is outdated. Basically any device which has a TPM and fulfils the requirements on Sealed Storage of the Trusted Computing Group could be used. But as they are not yet widely deployed, we decided to use smart cards as a possible alternative.

In order to prevent access to outdated log files the date of the log file has to be bound to the symmetric key used for that log file. This binding can be accomplished by three different mechanisms. The first one uses a key derivation function to calculate the symmetric key k from the date d of the related log file and a random value k_r. We use KDF3 as proposed in [Shou01] with SHA-512 as hash function. Thus, the symmetric key k is calculate as: $k = \text{SHA-512}\left(0^{64}|k_r|d\right)$. k_r is stored in asymmetrically encrypted form within the log file. If later a decryption of the log file is required, the encrypted value of k_r is send together with d to the smart card which, after proper authorisation and verification that d is not outdated, outputs k.

Note that the key derivation function is a one-way function, i.e. calculating k while knowing d and k_r is straightforward. But calculating either d or k_r from the other two values ($\{k, k_r\}$ resp. $\{k, d\}$) is difficult. Thus, the key derivation construction ensures that someone who knows $k = \text{KDF3}\,(k_r, d)$ and d cannot learn $k' = \text{KDF3}\,(k_r, d')$ for any value $d' \neq d$. Otherwise an attacker who wants to learn k' of an outdated log file with date d' and who has access to the smart card can send a valid date d together with the encrypted version of k_r to the smart card.

The second line of defence is to include the date d within the asymmetric encryption of k_r. We use RSA-OAEP for that asymmetric encryption: $Enc = \text{RSA-OAEP}\,(k_r, d)$. Derived from the non-malleability security property of RSA-OAEP, one can conclude that it is hard for an attacker who knows only Enc and the public RSA key used, to construct a valid $Enc' = \text{RSA-OAEP}\,(k'_r, d')$; where d' is a valid date, such that he can learn anything about k_r from k'_r.

The third and final line of defence is to include MAC over d using k_r as a key within the asymmetric encryption. Thus $Enc = \text{RSA-OAEP}\,(k_r, d, \text{MAC}_{k_r}\,(d))$. For calculating the MAC we use AES-128-GCM. This construction should make it even harder to construct valid Enc' using valid date d'.

Note that the smart card needs to know the current date in order to check if d is valid or not. How this can be achieved is described in section 2.5.

2.3 Integrity protection

So far we have described the mechanisms used to protect the confidentiality of the log entries. Now, we want to explain how the integrity of a log file is protected. Note that the integrity of a single log entry can be verified through MAC generated during the authenticated encryption (GCM) of that log entry. As already stated in section 2.2, we use the position of a log entry as initialisation vector for GCM. Therefore, copying a log entry to another position could be detected. Finally, we append a footer to the log file which consists of the encrypted and integrity protected number of log entries stored within the log file. Thus deletion of log entries could be detected and as a result provides protection against the truncation attack.

The alternative attack on integrity is to delete a whole log file and create a completely new one. This is possible, because knowledge of the public key alone, used to encrypt k_r (see above), is needed.

In order to prevent this attack it is sufficient to protect the integrity of the pair (k_r, d). We propose multiple mechanisms to achieve this kind of protection, namely: digital signatures, distribution and trusted timestamping. As for every single mechanism the security depends on different assumptions, we propose to use all of them for enhanced protection.

The "digital signature" mechanism means that the logging server signs the encryption of the pair (k_r, d). Note that the private signature key used by the logging server has to be changed frequently (in our case on a daily basis). Otherwise an attacker might generate a valid signature even for a log file generated before the logging server was compromised. In order to facilitate the key installation and management process, the digital signature key pairs can be generated in advance (e.g. one for each day of the year). The date (d) for which a given key pair is valid is encoded as the validity period of the public certificate of the signature test key.

The "distribution" mechanism means that every artefact involved in the integrity verification process should be distributed in a way so that it is hard for the attacker to manipulate all of the copies simultaneously. One way to achieve this is to utilise censorship resistant P2P-networks such as FreeNet [CSWH00] or Free Haven [DiFM00]. Another possibility is to send an artefact to a number of people. In order to prevent denial of service attacks by compromising only one copy, some form of threshold voting can be introduced. The set of artefacts to be distributed should include at least a hash value of the encryption of the pair (k_r, d). If digital signatures are used, the public key certificates should be distributed immediately after their generation.

The "trusted timestamping" mechanism means that every artefact mentioned above should be timestamped. As mentioned in section 2.2 and further explained in section 2.5, trusted timestamping servers are already used to prevent access to outdated log entries. Thus, we can use the same set of servers with little reorganization.

2.4 Searching for log entries

In order to answer requests of the law enforcement agencies (i.e. search for log entries) it is not necessary to decrypt a whole log file nor to check the integrity of a whole log file. It is sufficient to:

V1 verify the integrity of the encryption of the pair (k_r, d) (depending on the protection mechanisms chosen).

V2 verify the integrity of the number of log entries stored within the footer of the log file. This includes checking if the stored number of log entries equals the actual number of log entries found in the log file. This can be easily done, because each log entry as well as the header and the footer of a log file is of constant size.

V3 verify the integrity of every log entry "touched" during the search process. We use a binary search to find the first entry i for which $t_i \geq t_R - \epsilon$ and $t_{i-1} < t_R - \epsilon$. Starting from this entry, we sequentially decrypt the individual entries until we find the last entry i' for which $t_{i'} \leq t_R + \epsilon$.

The need for of V1–V3 follows directly from the considerations in section 2.3. V1 ensures that the whole log file is not generated by the attacker, whereas V2 ensures that any deletion of records from the end of the log file can be detected.

The fact that V1–V3 are sufficient derives basically from the observation that data, which is not input to the search algorithm cannot influence the result of that search algorithm. Therefore, it does not matter if log entries not "touched" during the execution of the search algorithm are manipulated by the attacker. Also, the integrity verification of a given log entry ensures that this log entry is in the correct position within the log file, as this position is used as initialization vector.

2.5 Trusted timestamping servers as reliable time source

For the enforcement of data retention period, the smart card needs to know the current date. Smart cards usually do not have an internal clock. Therefore, the current date has to be set from the outside. An operator can set the current date, but this introduces the risk of operator being forced to set an expired date, enabling the attacker to get access to outdated log entries.

In order to mitigate this risk, we decided that the only the source of time for the smart card should be (external) trusted timestamping servers (TTS). Therefore, an additional logical step is introduced in the process of answering data retention requests, which is activated during every key recovery process. When a key recovery from the smart card is requested, the smart card creates its own unique nonce, sends it to the PC application which then creates a time stamp request according to the "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)" [RFC 3161] for every TTS with this nonce included. The TSP requests are then sent to the trusted time servers[5]. The smart card verifies the signed TTS responses, including its own challenge nonces and eventually updates the internal time according to the time stamps provided (i.e. by means of some majority decision algorithm). The irrelevant parts of TTS response (e.g., chain of TTS certificates) outside digitally signed part with time and nonce can be stripped off on the PC console to speed up the processing on the smart card. Note, that the public certificates of the trusted time servers can be installed immutably on the smart card during initialisation.

2.6 Overall overview

The overall process of initialisation, generation of log entries and answering law enforcement requests is depicted in figure 1 and figure 2.

[5] As the smart card itself has no ability to directly communicate with time servers we use the PC console as a transparent proxy, with no possibility to undetectably modify TTS response.

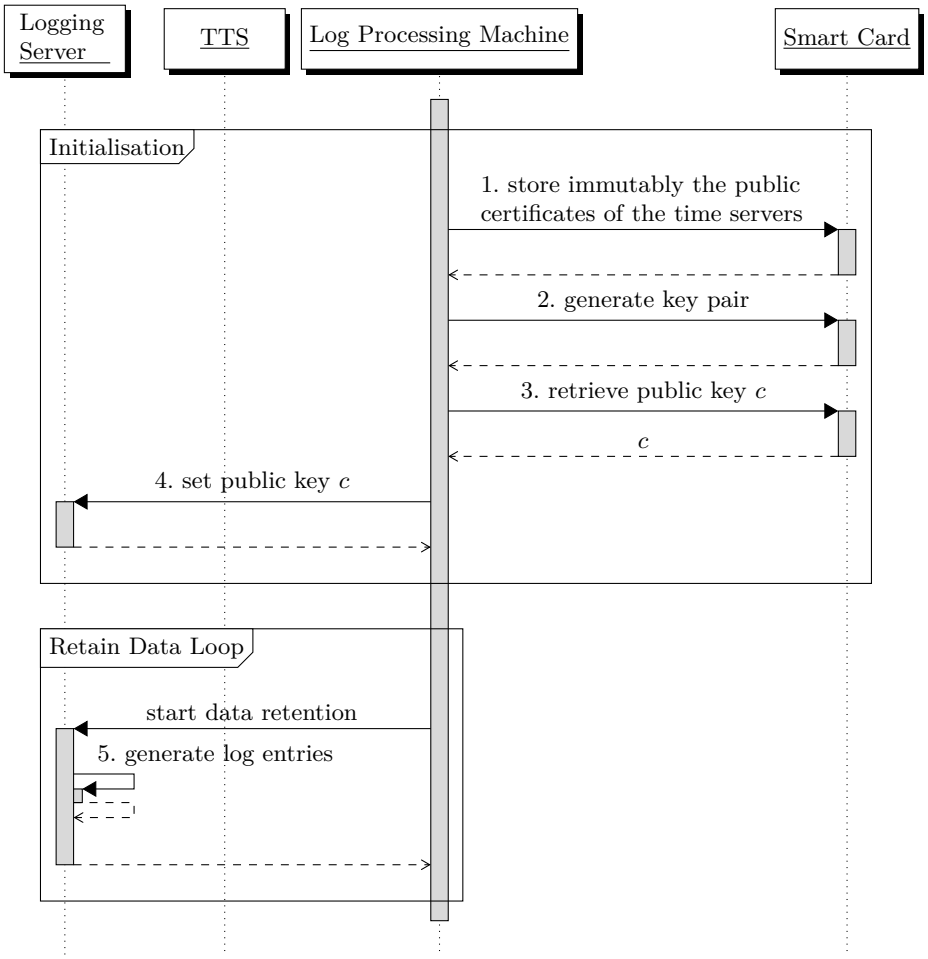

Fig. 1. Logical steps of the data retention compliant logging—initialisation and logging.

The following steps are executed only once during smart card initialisation:

1. The public certificates of the signature keys of the trusted timestamping servers are immutably stored on the smart card,
2. A unique RSA-2048 key pair is generated on-card (the private key never leaves the card),

3., 4. The public key c is exported to the logging server.

After this initialisation, the logging server can generate encrypted and integrity-protected log files (step 5) as described in section 2.

Finally, a request for the retained data is answered by executing the following steps:

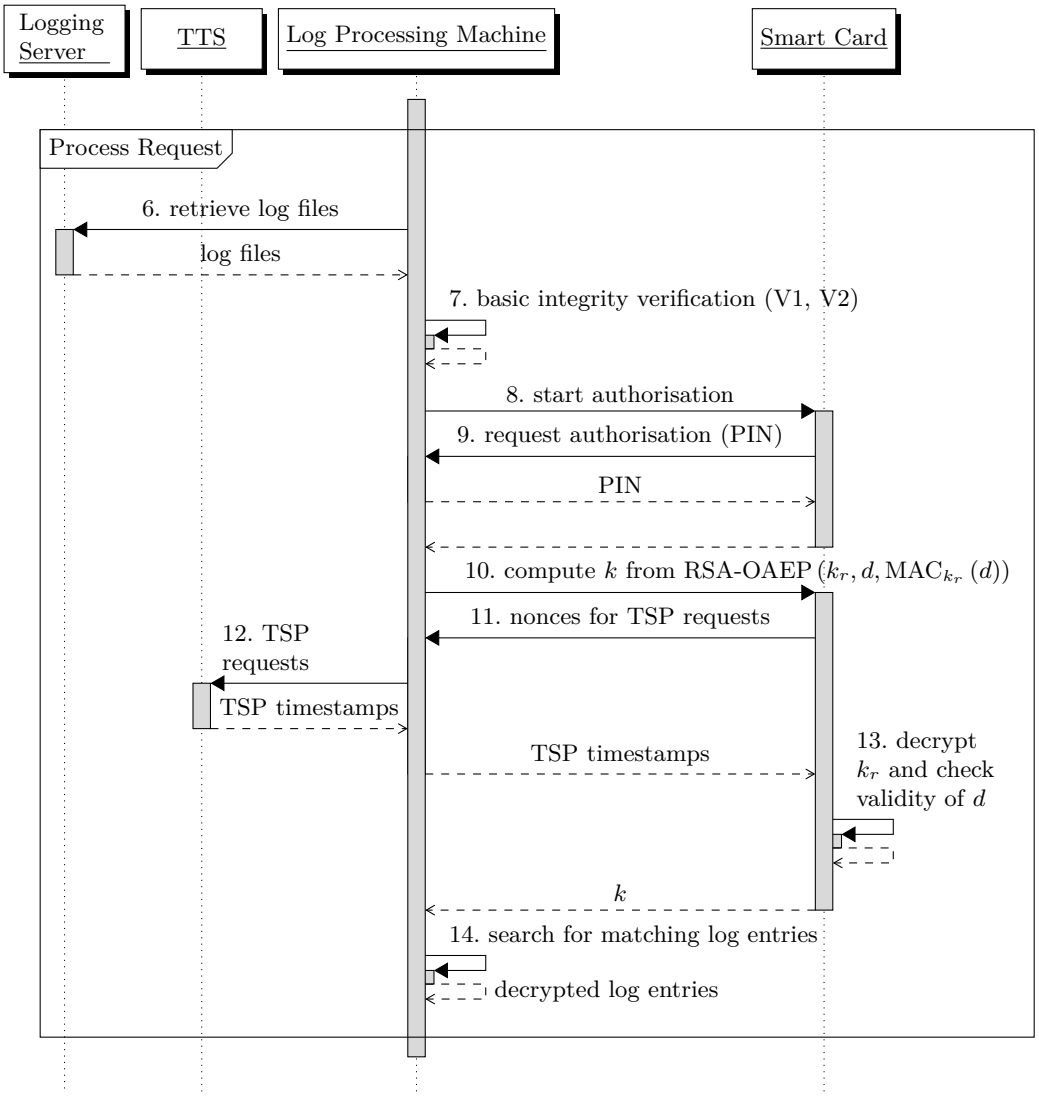

Fig. 2. Logical steps of the data retention compliant logging (continued)—processing requests.

6. The log files in question (according to the date) are transferred to a dedicated machine used for a processing of data retention requests.
7. After initialization of the log file processing tool, the basic log file integrity is verified according to V1 and V2 of section 2.3.

8., 9. The smart card is inserted into the reader connected to the dedicated machine. The user authenticates himself with his user PIN. Note, that in the case that the smart card reader has its own display and keypad, the PIN is

entered directly on the smart card reader and not on the dedicated machine as shown in figure 2 step 9.

10. The encryption of k_r stored in the log file is sent to the smart card.

11.,12. The smart card generates the nonces used for the requests to the remote trusted timestamping servers. These requests are generated by the log processing tool and sent to the trusted timestamping servers (step 12). The responses from the trusted timestamping servers are received and relayed by the log processing tool to the smart card. The smart card verifies the validity of the received timestamps.

13. The smart card decrypts the encrypted value of k_r. If the enclosed value d is still valid, the smart card calculates k and returns k to the log processing tool.

14. The log processing tool searches for the requested records and generates a report which can be sent to the law enforcement agency.

3 Remarks on the practical implementation

3.1 Smart card

We used the smart card with JavaCard platform for our implementation.

We use RSA-2048 as a basic asymmetric encryption primitive, which is implemented in hardware on the smart cards we used (JCOP-4.1 with JavaCard v2.2.1, SmartMX cryptographic processor). The key generation and the basic decryption functions are fast. The OAEP mode and SHA-512 hash function are not available on our platform[6] so it was necessary to implemented it on the software level with significant performance impact on decryption of k_r. The time required for retrieval of one key was approximately 90 seconds with the current setup. Nevertheless this time period is still practically useful, provided that the law enforcement agencies do not request hundreds of files per day. Significant performance improvement can be obtained with 32-bit smart cards, which might increase the speed more than twice due to the faster execution of arithmetic operations with larger operand. Smart cards with hardware support for SHA-512 algorithm will provide key recovery process with less than ten seconds. Although such smart card chips already exist, they were not available to us for our implementation. But driven by the new JavaCard 3.0 specification, it is anticipated that more powerful smart cards will be available for end users in the near future.

Note that the GCM mode is not supported by the current JavaCard specification as well as TSP timestamping requests. Both need to be implemented in the software. Fortunately, TSP uses only standard cryptographic primitives (RSA, SHA-1) which are part of the hardware in current smart cards.

[6] In fact, these functions are not available on most of the currently available smart cards.

3.2 Logging performance

So far we assumed that for every log entry a separate authenticated encryption using AES-128-GCM is performed. Given that the size of a log entry is relatively small compared to the AES-128-GCM block size this would lead to poor encryption performance and significant storage overhead. Therefore, we decided to group multiple log entries into a single block. As a consequence, the authenticated encryption carried out in blocks. According to [GrVi05], the block sizes between 256 and 1024 bytes lead to good performance results. Moreover, the block size should be a multiple of the AES-128 block size. On the other side not too many log entries should be grouped together within a single block as this could negatively impact the possibility of random access to an arbitrary log entry. Given these constraints and the actual size of a log entry, the number of log entries per block are calculated automatically by the logging server.

It is important to mention that we use this vector size as it requires no pre-processing of the initialization vector.

For our implementation of the cryptographic operations on the logging server we used the "Zork GCM 0.9.5" code (http://www.cryptobarn.com/gcm/). Without any extensive optimisations, we measured a speed of more than 85 MByte/s for block sizes ranging from 256 to 4096 bytes. The measurements were performed using an Intel Core 2 DUO T7700 2.4 GHz CPU. In case of our AN.ON system a single log entry requires less than 20 bytes with the cryptographic overhead for a single log entry being less than 0.25 μs. This is notably faster than the 5.55 ms previously reported by [MaTs09] for an Intel dual-core 1.73 GHz.

Given that every log entry is related to an asymmetric decryption operation of the anonymisation algorithm, which takes roughly 1 ms, the computational overhead introduced by the data retention is negligible (cf. requirement R5).

3.3 Search performance

Our dedicated search tool is written in Java utilizing the "Bouncy Castle" cryptographic library (http://www.bouncycastle.org/). The mostly used servers of our AN.ON system generated log files with speed of roughly 85000 blocks per day. Because each block contained 128 log entries, the whole log file contains more than 10 million log entries.

Processing of the whole log file (i.e. decrypting and checking the integrity of every single block) required about 630 seconds (measured using SUN Java 1.6 and an Intel Core 2 DUO T7700 2.4 GHz CPU). Thus, we needed approx. 7.5 ms per block. Altogether the processing time needed by our tool (e.g. less than 30 seconds for the search leading to roughly 2800 log entries ($\epsilon = 10$ s)) is negligible compared to the overall time need for answering a request by the law enforcement agency (i.e. checking the validity of the request itself, transferring the right log files to the dedicated machine, obtaining the decryption key from the smart card etc.). In summary, we conclude that our logging scheme fulfils the requirement R3.

4 Conclusions

The compliance with the new data retention directive introduces not only benefits for the law enforcement agencies, but also additional risks for the users and operators of the communication service need to be mitigated. We have proposed, implemented and start into the practical usage a secure logging service based on a combination of log file encryption, key recovery with smart cards and data retention period enforcement via trusted timestamping servers. Several categories of attackers with different capabilities and levels of access to the system were analyzed.

The main contribution lies in the design and implementation of a practical system that allows logging required data with only modest impact on performance of our anonymity service, which complies with the legal requirements and does provide additional protection for the holder of cryptographic secrets necessary to access the logged records. The records can be accessed only if a cryptographic smart card and its owner are present and the retained data is not outdated. An operator cannot be forced to reveal logged records outside the data retention period, because the period is enforced directly on the smart card with the help of trusted timestamping servers.

The log data of selected German AN.ON servers are protected with the proposed mechanism since 1st January 2009. So far, we did not receive any valid request for retained data from the law enforcement agencies. Therefore, at present, we can not evaluate how efficiently will the large number of log entries be handled, and we hope to provide further practical details in the near future.

Future work will focus on the problem of receipt creation. These receipts will contain provable information on all of the retained data that were released to the law enforcement agencies and serve as a official record (e.g., based on digital signatures and fair exchange protocols). While a seemingly straightforward task, the solution to this problem will have to avoid introduction of new risks for an operator (caused by possession of additional sensitive data on his side). Additional requirement that complicates the problem further is a need for a protection of the AN.ON users' privacy. The official record itself must not reveal any sensitive information (e.g. content of the retained data) to an outsider.

The authors would like to thank all anonymous reviewers, Jan Camenisch and Jakub Švenda for their valuable comments and Microsoft Research for the generous support which allowed the presentation of this work.

References

[Acco05] Rafael Accorsi: *Towards a secure logging mechanism for dynamic systems*; in Proc. of the 7th IT Security Symposium, São José dos Campos, Brasilien, November 2005.

[BeBK08] Stefan Berthold, Rainer Böhme, Stefan Köpsell: *Data Retention and Anonymity Services*; Proc. The Future of Identity in the Information Society - Challenges for Privacy and Security, FIDIS/IFIP Internet Security & Privacy Fourth

International Summer School, Springer, Boston, IFIP Advances in Information and Communication Technology, volume 298, 2009, 92–106.

[BeFK00] Oliver Berthold, Hannes Federrath, Stefan Köpsell: *Web MIXes: A System for Anonymous and Unobservable Internet Access*; Proc. of Privacy Enhancing Technologies Workshop (PET 2000), Springer, Berlin / Heidelberg, LNCS 2009, July 2000, 115–129.

[BeYe97] Mihir Bellare, Bennet S. Yee: *Forward integrity for secure audit logs*; Technical Report, University of California at San Diego, Dept. of Computer Science & Engineering, 1997.

[Chau81] David Chaum: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM 24/2, 1981, 84–88.

[CSWH00] Ian Clarke, Oskar Sandberg, Brandon Wiley, Theodore W. Hong: *Freenet: A Distributed Anonymous Information Storage and Retrieval System*; Proc. of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, Springer, Berlin / Heidelberg, LNCS 2009, July 2000.

[DiFM00] Roger Dingledine, Michael J. Freedman, David Molnar: *The Free Haven Project: Distributed Anonymous Storage Service*; Proc. of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, Springer, Berlin / Heidelberg, LNCS 2009, July 2000.

[DiMS04] Roger Dingledine, Nick Mathewson, Paul F. Syverson: *Tor: The Second-Generation Onion Router*; Proc. of the 13th USENIX Security Symposium, August 2004, 303–320.

[GrVi05] David A. McGrew, John Viega: *The Security and Performance of the Galois/Counter Mode (GCM) of Operation*; Proc. of Progress in Cryptology – INDOCRYPT 2004, Springer, Berlin / Heidelberg, LNCS 3348, 2005, 343–355.

[Guer09] Shay Gueron: *Intel's New AES Instructions for Enhanced Performance and Security*; Proc. Fast Software Encryption, Springer Berlin / Heidelberg, LNCS 5665, 2009, 51–66.

[Holt06] Jason E. Holt: *Logcrypt: forward security and public verification for secure audit logs*; Proc. of the 2006 Australasian Workshops on Grid Computing and E-Research, January 2006, 203–211.

[MaTs09] Di Ma, Gene Tsudik: *A new approach to secure logging*; ACM Transactions on Storage (TOS), vol. 5, issue 1, ACM, New York, March 2009.

[RFC 3161] C. Adams, P. Cain, D. Pinkas, R. Zuccherato: *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*; August 2001, Proposed Standard, http://www.rfc-editor.org/rfc/rfc3161.txt.

[ScKe99] Bruce Schneier, John Kelsey: *Secure Audit Logs to Support Computer Forensics*; ACM Transactions on Information and System Security (TISSEC), vol. 2, Nr. 2, 1999, 159–176.

[Shou01] Victor Shoup: *A proposal for an ISO standard for public key encryption*; Version 2.1, 20th December 2001, http://www.shoup.net/papers/iso-2_1.pdf, last accessed Juli, 28th, 2009.

[SP 800-38D] Morris Dworkin: *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*; U.S. Department of Commerce, National Institute of Standards and Technology (NIST), Information Technology Laboratory (ITL), November 2007, http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf, last access on December, 1st 2008.

[WSBL08] Karel Wouters, Koen Simoens, Danny Lathouwers, Bart Preneel: *Secure and Privacy-Friendly Logging for eGovernment Services*; in Proc. of the 2008 Third International Conference on Availability, Reliability and Security, IEEE Computer Society, 2008, 1091–1096.