

Delegating Secure Logging in Pervasive Computing Systems

Rafael Accorsi and Adolf Hohl

Department of Telematics,
Albert-Ludwigs-Universität, Freiburg
{accorsi, hohl}@iig.uni-freiburg.de

Abstract. Logging is a central service in computing systems. It lays the foundation for accountability and audit services in computing systems, as well as for other accessory services. While providing logging services in traditional computing systems is a relatively smooth process, it turns to an intricate task in pervasive computing systems. In this context, we present two contributions addressing this problem. First, we develop an approach to securely log information in marginally trusted collectors. Second, we investigate the question of how to securely delegate our logging protocol to a relay equipped with trusted-computing modules.

1 Introduction

Irrespective of the underlying computing paradigm and application, logging services are relevant. Logging services collect and store events communicated by devices within the system, possibly using an intermediary relay, and thereby lay the foundation for accountability and audit services in computing systems. For this to happen, log data must be authentic, for *ex falsum quod libet*, that is, from a falsity everything follows.

Protecting the security of log data, expressed in terms of its authenticity, is thus of foremost importance. In this paper, we focus on pervasive computing systems, i.e., systems allowing for the omnipresent availability of computing power, communication and data.¹ A distinguishing feature of pervasive computing environments is their inherent mixed-mode design, that is, the seamless combination of resource-poor and resource-rich devices: While resource-poor devices lack storage, computing power or energy supply, resource-rich devices do not.

The pervasive setting poses several challenges to security [26], in particular to secure logging. Problems arise when, e.g., the device notifying log events is not the same as the service collecting log data, or when a relay between the device and the collector intermediates the logging process. To counter these problems, two possibilities can be taken into account. One can devise lightweight logging protocols to accommodate the resource limitations imposed by resource-poor devices, or apply to methods to delegate log tasks to external, marginally

¹ We consider this as a result of advances in distinct aspects of computing systems, namely autonomy [1], pervasiveness [23], and reachability [12] of devices.

trusted resource-richer devices offering relay or collector services. The concern behind lightweight methods is the well-known trade-off between performance and security [13], where privileging performance entails in fragile security guarantees. Delegation of log tasks has a similar consequence, as it is non-trivial to devise methods to ensure and obtain evidence that the delegatee carries out the tasks as expected.

In this paper, we report on a logging mechanism to securely store log data in, and delegate log tasks to marginally trusted collectors. Our main contributions are two-fold. Assuming devices' scarceness of storage, we first propose an approach to securely store log data in marginally trusted collectors. The protocol we propose leverages the techniques proposed by [25] and, additionally, provides for non-repudiation of the collector's receipt of log data. Second, assuming that not only storage is scarce, but also the underlying computing power, we investigate the question of how much of our protocol can be (securely) delegated to relay. To this end, we employ trusted-computing platforms to allow for remote attestation and, thereby, to obtain strong security guarantees regarding the behavior of the relay.

Overall, our work sheds a light on the increasingly relevant, yet obscure area of logging and accountability in pervasive computing systems and its combination with secure hardware. At investigating this topic, the first difficulty is the characterization of an appropriate attacker model. Although we sketch such a model, in this paper we merely focus on illegal actions violating the authenticity of log data. To this end, we define what authenticity (and, thereby, security) of log data means and show how it can be harmed.

To distribute log tasks to marginally trusted collectors and relays is not a trivial task. The protocol we propose in §3 assumes that the device has enough computing power to use well-known cryptographic primitives to protect log data and delegate storage to a remote collector. Since this approach is considerably costly and only partially applicable to pervasive computing, we extend our approach with underlying remote attestation techniques based on trusted-computing platforms to delegate the whole process to a relay. As we see below, this considerably saves computation cycles and, as such, could be applied in pervasive computing environments. We remark that, in general, logging mechanisms such as the one we present can also be misused to violate individual's privacy. Although we see privacy as a central issue in pervasive computing environments, we deliberately do not address it in this paper.

This paper is organized as follows: in §2, we introduce the scenario underlying our research into logging and characterize the security properties of log data. We describe our logging protocol in §3, and discuss its adequacy and limitations regarding the expected security properties. In §4 we investigate how to delegate log tasks to marginally trusted relays using trusted-computing platforms, and discuss our results. In §5, we briefly evaluate our approach and suggest alternative approaches to cope with delegation of log tasks. In §6, we report on related work, and draw conclusions and state the upcoming research issues in §7.

2 Preliminaries and Security Properties

We classify the components involved in logging services according to their role. In this setting, a *device* is a service that generates a log message; a *relay* is a service that receives a log message and forwards it to another service; and a *collector* is a service that receives a message and, in one way or another, archives it. We employ the following notation in this paper:

- d_i denotes the i device, r_i the i relay and c_i the i collector; D denotes log data.
- $\{X\}_K$ denotes the encryption of message X under K . $Sign(X)_K$ stands for the signature of X with an asymmetric key K .
- X, X' stands for the concatenation of the messages X and X' .
- K_s stands for the public-key of the service s and K_s^{-1} s 's private-key.
- $MAC_K(X)$ denotes the message authentication code of X under K .
- $Hash(X)$ is the one way hash of X .

While we do not prescribe a particular set of algorithms to implement the cryptographic primitives above, we assume that these functions fulfill the expected properties in that, e.g., it is infeasible for an attacker to provoke collisions of hash values

2.1 Characterizing the Security Guarantees

Log data can only be used if it is authentic. We define authenticity as the simultaneous fulfillment of the following properties:

- *integrity*: log data faithfully reflects the reality. That is, the information logged is consistent, accurate, and correct.
- *uniqueness*: log data shall not allow for parallel realities, i.e., it is impossible to intercept log data sent from d_1 to c_1 and to resend it (possibly in modified form and claiming a different device identity) to c_2 . This entails confidentiality of log data during the transmission, for log data transmitted in clear can be easily duplicated.

These requirements characterize authenticity properties of log data and are implemented using different cryptographic techniques. Log services based on these techniques need to ensure *tamper evidence*, i.e., attempts to illicitly modify log data must be detectable to a verifier [15], and *forward integrity*, that is, log data contains sufficient information to confirm or rebuke allegations of log data modification before the moment of the compromise [6]. Tamper evidence and forward integrity are necessary since absolute tamper resistance is, in practice, infeasible [5]. This specially holds in pervasive computing systems [28].

2.2 Threat Model and Attacks

The goal of the attacker is to access log data and, thus, violate its integrity and uniqueness. For this, the attacker uses different strategies, which we model using

a slightly modified version of the Dolev-Yao attacker model [10].² In order to model attacks upon stored log data, we extend the capabilities of the attacker. Namely, once he gets to the logfiles, he can read, delete, and modify stored data. We assume that the attacker cannot misuse cryptography, i.e., he can only perform cryptographic operations using the appropriate cryptographic keys.

Attacks upon log services have different facets and, as such, target different participants. Faithfulness and uniqueness can be attacked using (possibly a mixture of) the following strategies:

- *replay of log messages*: the attacker records a set of messages M sent by a device d to collector c . Later, he attacks d and, in order to hide collected evidence about the attack, sends a (refreshed) set M to c .
- *integrity of sent and stored log data*: the attacker has access to the communication medium and can modify data during the transmission. Moreover, if the attacker gains access to log files at the collector c , he can read and modify them at will.
- *authenticity of device*: the attacker takes over a service, modifies its identity, and starts to send log messages to a collector.
- *confidentiality of log messages*: during the transmission, the attacker intercepts and reads messages sent in clear-text over the network. Similarly, confidential log data stored in a collector may be accessible to an attacker.
- *availability of log service*: the attacker floods the collector c with irrelevant log messages, so that legitimate messages sent by the devices are not logged. The attacker can also use this setting to attack the devices sending messages to c , since his attempts are not protocolized.

3 An Approach to Security Logging

Our method to providing secure logging services in pervasive computing systems builds on and extends the techniques proposed in [25]. To simplify matters, we consider that a device and a collector communicate either without an intermediary relay, or that the relay does not misbehave. Under these assumptions, our protocol consists of the following steps:

1. *mutual authentication of services*: apart from authentication, the device and the collector also agree on a secret value pv_0 . This is a proof value used at the acknowledgement phase; its goal is to ascertain authenticity of log messages.
2. *initialization and construction of the logfile*: the device is in charge of applying cryptographic techniques to safeguarding the integrity and uniqueness of its logfile. We assume that chunks of log data are sent from the device to the collector.

² We remark, however, that one challenge in pervasive computing systems is the development of an adequate attacker model [28]. For ongoing research on attacker models for pervasive computing systems see [8].

3. *acknowledgement of receipt from collector*: the collector computes a hash value based on the device’s messages and sends it signed together with a timestamp and protocol sequence number back to the device. The device then stores this unambiguous piece of information, as it demonstrates that the collector received the chunk of log data correctly and can be held accountable for attacks upon this data.

For our initial aim, we focus on the last two steps; authentication will play a role in §4, where we leave out the aforementioned assumptions and consider an unreliable relay.

3.1 Initializing the Logfile

Assuming that the device and the collector successfully prove their identities to each other and agree on a secret value A_0 , d creates the log file by inserting the first entry into it.

$$L_0 = \left[\begin{array}{|c|} \hline W_0 \\ \hline \end{array} \right] \left[\begin{array}{|c|} \hline \{D_0\}_{K_0} \\ \hline \end{array} \right] \left[\begin{array}{|c|} \hline Y_0 \\ \hline \end{array} \right] \quad \left[\begin{array}{|c|} \hline Z_0 \\ \hline \end{array} \right]$$

Fig. 1. Initial log entry L_0 and authenticator Z_0

All the entries in the logfile have the same format. We illustrate the initialization entry L_0 in Fig. 1, where the fields stand for the following information:

- W_0 is a permission mask to regulate the access to the log entry L_0 . According to [25], at the initialization phase this entry type may be set to **LogfileInitializationType**.
- $\{D_0\}_{K_0}$ is the symmetrically encrypted log data for the entry L_0 and K_0 is a random session key. To provide the necessary security guarantees, D contains not only the event to be logged, but also a timestamp and a protocol identifier. (The former states the actuality of the entry, the latter avoids harmful protocol interactions between different protocol steps.)
- Y_0 stands for the first link of a hash-chain.³ The actual initialization value of Y_0 is randomly generated.
- $L_0 = W_0, \{D_0\}_{K_0}, Y_0$ is the initial log entry.
- Z_0 is the message authentication code of L_0 defined as $MAC_{pv_0}(L_0)$. This piece of information is used to compute the proof value associated to the whole chunk of log entries and, thence, is *not* send along with L_0 to the collector.

The process of initializing the logfile includes an authenticated response of the collector in order to detect possible tampering.

³ In the simplest form, a hash-chain Y can be inductively defined as $Y_1 = Hash(Y_0)$ and $Y_n = Hash(Y_{n-1})$.

3.2 Adding Log Entries

After creating the logfile, the device starts adding entries to the logfile. This process encompasses the following operations:

1. $A_j = \text{Hash}(A_{j-1})$ denotes the authentication key of the j th log entry. The confidentiality of this information is essential for the security of the log data, as it is used, either directly or indirectly, in every step of the protocol. Thus, we assume that the computation of the new value overwrites irretrievably the previous value.
2. $K_j = \text{Hash}(W_j, A_j)$ is the cryptographic key with which the j th log entry is encrypted. This key is based on the permission mask W_j . Thus, only allowed services may access the entry.
3. $Y_j = \text{Hash}(Y_{j-1}, \{D_j\}_{K_j}, W_j)$ is the j th value of the hash-chain. Each link of the hash-chain is based on the corresponding encrypted value of the log data. This ensures that the chain can be verified without the knowledge of the actual log entry.
4. $L_j = W_j, \{D_j\}_{K_j}, Y_j$, i.e., the log entry consists of the permission mask, the encrypted log data and the hash-chain value.
5. $Z_j = \text{MAC}_{pv_j}(\text{Hash}(L_j))$ is the authenticator of the j th log entry, where pv_j is defined as $pv_j = \text{Hash}(Z_{j-1}, pv_{j-1})$. Note that we compute the message authentication code for the whole entry instead of a field of it (in the case of Schneier and Kelsey, the hash-chain value Y). While this increases the computational cost involved in calculating of this field, it is necessary to ensure the acknowledgment phase.

This process describes how the j th log entry is computed. We address the overall security guarantees provided by this protocol phase in §3.4.

3.3 Acknowledgement Phase

The last phase of the protocol focuses on the collector and its goal is to provide an irrefutable evidence regarding collector's possession of the chunk of log data sent by the device, as well as chunk's integrity. To this end, the following steps are carried out:

1. by receiving the chunk L_j-L_k (with $j < k$), the collector computes for each entry L_i the corresponding A_i and Z_i values.
2. after $k - j$ iterations, the collector gets obtains $\text{MAC}_{pv_k}(\text{Hash}(L_k))$.
3. the signed proof value $\text{Sign}(\text{MAC}_{pv_k}(\text{Hash}(L_k)))_{K_c^{-1}}$ is sent to the device. This message includes a timestamp and protocol step identifier.
4. the device then checks whether the authenticator matches with the authenticator computed during the second phase. If yes, the devices frees the storage by deleting the chunk.

This concludes the protocol. The device can now ascertain the integrity of the logfile, as well as (selectively) delegate access to its entries.

Table 1. Devices’ computational cost associated to running the logging protocol

Protocol phase	Kind of cryptographic operation				
	PSG	MAC	Hash	SC	AC
Initialization	3	1	–	1	–
Appending an entry	–	1	4	1	–
Acknowledgement	–	1	–	–	1

On the Computational Cost. In pervasive computing systems, the applicability of an approach depends on the computational effort involved in running its implementation. To this end, we use the number of cryptographic operations carried out by the device while running the protocol, as shown in Table 1, where PSG stands for pseudo-random generation, SC for symmetric cryptography, and AC for asymmetric cryptography. For certain resource-poor devices, this effort can be an overkill. In §4, we present an approach to delegate these task to a relay, and compare the associated effort.

3.4 Discussion on the Required Security Guarantees

Assuming that the device accurately archives the events its sensors communicate, guarantees regarding integrity are safeguarded by the proof value sent by the collector in the last protocol phase. The message authentication code employed by the collector proves that it possesses the corresponding secret A and demonstrates that every log entry is a faithful representation of the original log data. It should also be noted that although the collector possesses the secret A , it cannot read the log data D sent by the device, as it is sent in encrypted form.

This fact is closely related with the concept of forward integrity. If an attacker takes over a collector c at time t , he is able to act as if c were not compromised. For this, he merely keeps sending the proof values as prescribed by the protocol. But, since he cannot obtain the previous values of A , it is impossible for him to read log entries archived before t . Thus, these entries fulfill confidentiality and integrity properties.

Regarding uniqueness, assuming that the cryptographic functions employed in the algorithms are sound, uniqueness is guaranteed by the use of timestamps and the hash-chains. Timestamps bring along a synchronization problem between the device and the collector. Although the probability of an attack due to timing confusion might be negligible in practice, the hash-chain works in a complementary manner and prevents old messages from being seamlessly and undetectably added to the chain.

This leads to the need for tamper evidence. The hash-chain employed in the protocol lets us verify the integrity of the whole chain by inspecting the last link in the chain. The proof value send by the collector demonstrates that the log data has not been tampered with during the transmission or by a malicious process acting between the device and the collector. Hash chains also make it possible to detect whether (series of) log entries were eliminated.

3.5 Limits of the Logging Protocol

The logging protocol we present above is based on a number of assumptions, which outline and complement the underlying technical restrictions. We now briefly discuss the most significant limitations.

- Timepoint of a successful attack: the integrity of the logfile can only be ensured up to the point in time at which the attacker takes over the collector (or device). Once an attacker succeeds in compromising a machine, he can misuse log information at will.
- Deletion of log entries: it is always possible for an attacker to completely (and irretrievably) delete the log file or entries thereof, in particular when we assume that no write-only hardware or media (e.g. WORM disk, paper print-out, optical jukebox) is at hand. Tamper evidence allows us to detect the deletion.
- Collector’s behavior: although the device receives an evidence that the collector appropriately receives the log data, it cannot be sure that it stores the data. To obtain this kind of security guarantees, below we employ trusted computing-based techniques (see §4).

4 On Delegation of Tasks

In pervasive computing environments, devices with different resource constraints participate. Resource constraints such as a slow computing power caused by slow CPU or a battery-based power supply are reasons for delegating the processing of resource intensive tasks to resource-rich devices. In this case, a service on the relay can provide suitable services with the required protection goals of the log data. Such a shift of computation tasks to a virtual coprocessor is visualized in Fig. 2. A resource-poor device buys computing power and services from a powerful platform for outsourcing the computing intensive tasks.

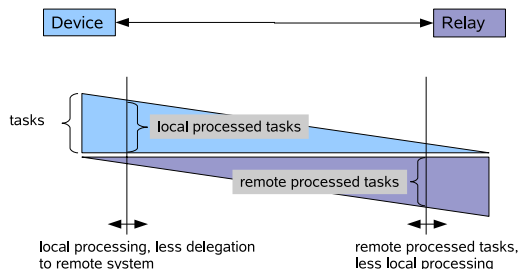


Fig. 2. Shift of computation tasks from a delegator to a delegatee

To central requirement for a successful delegation is the absence of negative side-effects regarding the underlying security guarantees. On the other hand, the necessary requirements for a virtual coprocessor are:

- *secure communications*: it must be possible to establish a secure connection to the virtual and remote coprocessor for confidential data communication. From the perspective of a resource-poor device, a resource effective security protocol is necessary.
- *exclusiveness*: the exclusiveness of the virtual coprocessor is necessary to prevent any side effects of other programs and actions on a system with the virtual coprocessor.
- *authentication*: to establish a trust relation between the logging device and the virtual coprocessor, the device authenticates selected attributes at the virtual coprocessor. Although device based authentication works fine when the device and the service are in the same administrative domain, problems arise when they are in distinct ones, as it could be in pervasive environments. Therefore, similar to Creese et al. [9] attributes of the service should be authenticated instead. Because of the *Undecidability Theorem* it is not possible to derive the behavior of a piece of code in general, the authentication of behavior could be done via code identification and a suitable mapping. In this case, it would be no longer necessary to authenticate its identity and rely on trust relationships of their maintainers.

4.1 Building an Adequate Coprocessor

A Trusted Computer (TC) platform provides an important building block for a virtual coprocessor. Trusted Computing technology [27] is widely available and is used below to outsource the computations for secure logging from a resource-poor device to a in general marginally trusted relay. TC platforms provide functionality to faithfully transmit attributes of a platform to a remote challenger via the so-called *remote attestation*. This functionality of TCG-compliant platforms allow the authentication of executed code via a Trusted Platform Module (TPM). The TPM acts as a secure logbook of executed code stores their hashes in the so-called platform configuration registers (PCR) for an audit of the platform later.⁴ Therefore, it is the duty to report every code hash from the startup of the platform. The values in the PCRs can be communicated faithfully and by mapping the authenticated code fragments to a set of properties of the system, one can decide, whether the system is secure in his sense.

With the use of a TC-Platform, a virtual coprocessor could be easily authenticated. For the delegation of a logging task, the relay system has to prove the existence of an operating system [20] with a resistant process isolation to provide exclusiveness for a virtual coprocessor. Therefore, the relay has to run a system with this attribute which is known by the device. Furthermore, for a secure communication, avoiding any time delay attacks when authenticating a relay using remote attestation is used [14].

⁴ The TCG specifications up to 1.1b prescribe 16 registers with 160 bit; TCG 1.2 specifies at least 24 registers.

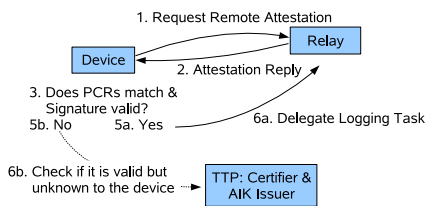


Fig. 3. Remote attestation of the relay

4.2 Delegation of Secure Logging Task

If the behavior of a service could be authenticated, a service doing the job as expected can be selected and authenticated. Although Trusted Computing does not allow to authenticate a certain behavior, it supports the authentication of a binary which behaves as expected. Taking for granted that the service is not interfered by other activities on the relay, this one could be seen as a virtual secure coprocessor of the device. In this case, the secure logging task can be delegated to the remote relay lying not necessarily in the same administrative domain.

In the scenario, we assume a resource poor device. It would be advantageous to use a simple representation of valid relay configurations to enable a fast authentication of the virtual coprocessor (including the platform which provides it). Therefore, we propose to represent the platform configuration by a few sets of valid relay configurations. We do not pay attention to the history which led to this configuration. This has the consequence, that other valid configurations are excluded.⁵ Assuming a static platform configuration, the same functionality can be achieved using the so-called *sealing* functionality.

To establish a the secure communication, meaning a device can authenticate a qualified relay when transmitting log data an authentication of the corresponding service access point of the log daemon has to be performed. The relay transmits the hash of a generated authentication secret. Later on, this secret is used to authenticate a prior platform a remote attestation was requested from. One selected PCR is used to carry the fingerprint of this authentication secret to get it signed during the remote attestation procedure. Thus, we misuse a PCR as a signing channel using a so-called *attestation identity key* (AIK).⁶ This saves a further key pair and certificate for signing purposes.

The device then authenticates a virtual secure coprocessor via remote attestation. A signed set of the PCR is replied. If they match with a known set of

⁵ We remark that different calling orders of code usually result in other values in the PCR of the TPM.

⁶ One of many certificates to sign PCRs, also known as platform identity. They are issued based on whether the platform is a valid TCG system.

values (representing an untampered virtual coprocessor), the device proceeds. Otherwise the device needs to contact a trusted third-party to acquire information about acceptable relay configurations [21].

The delegation of the logging task requires the following steps and computational costs for the device:

1. *authentication of the virtual coprocessor*: the relay authenticates itself with its platform configuration. Hence, every relay with the same platform configuration is also qualified for a the delegation of the logging task. The platform configuration consists of the PCRs. These registers are signed during remote attestation. The computational cost for the device is to compute the hash of the transmitted registers and perform the verification of a RSA signature.
2. *authentication of its service access point*: the service access point of the virtual coprocessor has to be authenticated. For this step, a selected PCR is used to transmit the hash of a public key (previously generated) of the secure coprocessor during the remote attestation procedure before. When the device connects to the relay, the relay presents the public key. With that, the device can verify the signed hash and the former platform configuration belongs to this relay. Then the device performs a challenge which only could be answered correctly by the corresponding relay. While [14] uses a RSA key pair, an algorithm suitable for resource-poor devices should be selected. The computational cost for the device is to perform a hash of the public key and encrypt a challenge in either RSA or a more computational efficient asymmetric cipher.
3. *setup of a secure channel*: the challenge sent in the step before can be used as a key for a symmetric cipher. Both, the device and the relay know that key solely and can communicate in a confidential way using any protocol which provides a symmetric encryption.

For each log entry the device transmits data to the relay. Additional computation arise for the encryption. Assuming that a log entry has an overall length of about 100 bytes, about 13 cycles with a symmetric block cipher (with 8 byte per block) are necessary to encrypt the log data. Table 2 summarizes the necessary cryptographic operations of the device when the logging task is delegated.

Table 2. Devices'computational cost associated to delegate the log task

Protocol phase	Kind of cryptographic operation				
	PSG	MAC	Hash	SC	AC
Initialization	1	–	1	–	2
Transmitting an entry	–	–	–	1	–

5 Discussion and Evaluation

We now briefly compare the effort involved in using the protocol we propose, both with and without delegating the cryptographic operations involved in the protocol.

The initialization phase of the protocol consumes considerably less resources in the case without a TC platform, as it does not include asymmetric RSA encryption. Asymmetric RSA is necessary, since this algorithm is used for signing the attestation vector. Transmitting an entry is, according to our analysis, cheaper than appending it, as it only requires a symmetric encryption of log entries. Also, using the TC platform it is not necessary to add an acknowledgement phase; this could be handled with the delegatee, i.e., the relay.

Although computing power rises fast, it would be adequate for pervasive computing devices with low power and computing resources to save computing cycles and use other algorithms than RSA. Such algorithms are already available, e.g., multi exponent RSA, XTR or ECC [7]. While this specific issue could not be addressed with TCG platforms, the promising *XOM* (execute only memory) could be used [17]. In *XOM*, the attestation of an application is handled inside the application. The software designer has the freedom to select an algorithm which suits best. This would cut down the necessary computing cycles a device needs for an authentication procedure of the delegated logging service. Unfortunately, the *XOM* approach is in an academic stage and not available, therefore the TCG approach is used instead.

6 Related Work

A number of approaches have been proposed for securely logging information in computing systems. The majority of these approaches are based on *syslog*, a de facto standard log service whose functionalities are described in the RFC 3164 [18]. Security was not considered at developing *syslog*: it provides no device authentication; it ignores the original source of log data; it transmits and stores log data in clear-text; and uses UDP as underlying transport protocol. Security was thus an afterthought. Indeed, secure extensions have been developed.

- *syslog-ng*: *syslog*'s new generation aims at providing a secure, reliable log service [4]. It is backward compatible with RFC 3164 and its only advantage against *syslog* is the use of TCP as transport protocol. Encrypted and signed transmission and storage are still not provided.
- *syslog-sign*: This service adds origin authentication, message integrity, replay resistance, and detection of missing messages to *syslog* [16]. This is accomplished with a cryptographically signed message that contains the signatures of previously sent *syslog* messages. The contents of this special message is called “signature block.” To our knowledge, no implementation of *syslog-sign* is available to-date. *syslog-sign* does not provide confidentiality during the transmission of log data, nor does it account for encrypted storage of log

entries. Moreover, since the signature blocks may be deleted after the authentication, tamper evidence and forward integrity are only partially fulfilled.

- **syslog-pseudo**: Logfiles often store personal data and, thus, are a popular attack point. **syslog-pseudo** proposes an architecture to sanitize logfiles in unix-like systems [11]. This approach focuses exclusively on the privacy of users and does not take into account the security properties suggested in §2.1.
- **Reliable syslog**: While previous approaches are based on the RFC 3164, the reliable **syslog** is based on the RFC 3195 [19] and aims to implement reliable delivery for **syslog** messages. For this, it is built on top of BEEP, the Block Extensible Exchange Protocol [2]. BEEP is a framework for building application protocols that uses TCP and allows device authentication, mechanisms to protect the integrity of log messages, and protection against replay attacks. An implementation of reliable **syslog** is available [3].
- **Schneier and Kelsey**: This method is the starting point of our investigation. In essence, it aims at realizing tamper evidence and forward integrity. In addition to that, mechanisms to protect the integrity of log entries, as well as their confidentiality and access control are presented [25]. However, the assumptions underlying our approach, as well as the procedures involved in achieving the security guarantees, differ in various ways. The authors assume that the device and the collector are the same machine. They also assume that the owner of a device is not the same person as the owner of the secrets within the device. In this context, log services must be in place to determine whether there has been some fraud. To our knowledge, this method has not been implemented.

The use of trusted coprocessors to support secure logging has also been explored by Schneier and Kelsey [24]. In contrast to our work, they use trusted coprocessor to authenticate the source of log data but do not support delegation of the logging task.

Concerning trusted computing, current research focuses on minimal trusted computing bases and the integration of TCG compliant platforms. Promising work is done by Pfitzmann et al. [20]. They separate program execution by compartments, implemented on a micro kernel architecture. The interpretation of remote attestations is the concern of Sadeghi et al. [22]. They map values of PCRs and execution history of a TC platform to properties of a platform. In our scenario, applying this approach could lead to more platform configurations of a relay, accepted by a device.

7 Conclusion and Outlook

We have proposed an approach to securely distribute log services among marginally trusted collectors and relays. Our approach combines standard cryptographic techniques and secure hardware-based technologies to lay a fundamental basis for accountability and audit in pervasive systems. Our work sheds a light on, and allows us to characterize the difficulties of, deploying log services in pervasive computing environments.

This is merely the first step and many interesting research questions are still open. First, the permission mask can be used by authorized services—e.g. audit or configuration services—to obtain log data directly from a collector. For this, protocols to regulate delegation should be in place and analyzed against the protection goals presented in §2.1 in order to avoid harmful protocol interactions. We plan to examine these issues and develop protocols for these purposes.

Second, an adequate attacker model for pervasive systems is still lacking. Such a model should also take into account the mixed-mode setting of pervasive computing systems and its underlying features. At our institute, research is being carried out in this direction and we plan to investigate different attacker models and attacks against our protocol.

Third, although the initial experiments using our logging protocol are promising, we do not have a practical handle on the complexity involved in running the protocol. An in-depth analysis should reveal the extent to which it can be applied in pervasive systems and, by this means, show when a delegation-based approach is necessary. This result is fundamental to the applicability of our approach.

Finally, if misused, such a powerful log mechanism is a privacy *endangering* technology, specially in an environment where virtually *every* piece of information can be collected and processed in order to derive further characteristics of an individual. We plan to address the non-technical implications caused by our approach to logging.

References

1. Autonomic computing. <http://www.research.ibm.com/autonomic/>, 2005.
2. BEEP. <http://www.beepcore.org>, 2005.
3. Reliable syslog. <http://security.sdsc.edu/software/sdsc-syslog/>, 2005.
4. Syslog-ng web site. http://www.balabit.com/products/syslog_ng, 2005.
5. R. Anderson and M. Kuhn. Tamper resistance: A cautionary note. In *2nd USENIX Workshop on Electronic Commerce*, pages 1–11. USENIX Assoc., 1996.
6. M. Bellare and B. Yee. Forward integrity for secure audit logs. Technical report, Univ. of California at San Diego, Dept. of Computer Science & Engineering, 1997.
7. E.-O. Blaß and M. Zitterbart. Towards acceptable public-key encryption in sensor networks. In *IWUC*, pages 88–93. INSTICC Press, 2005.
8. S. Creese, M. Goldsmith, R. Harrison, B. Roscoe, P. Whittaker, and I. Zakiuddin. Exploiting empirical engagement in authentication protocol design. In volume 3450 of *LNCS*, pages 119–133. Springer, 2005.
9. S. Creese, M. Goldsmith, B. Roscoe, and I. Zakiuddin. Authentication for pervasive computing. In volume 2802 of *LNCS*, pages 116–129. Springer, 2003.
10. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.
11. U. Flegel. Pseudonymizing unix log files. In volume 2437 of *LNCS*, pages 162–179. Springer, 2002.
12. G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, 1994.
13. M. Graff and K. van Wyk. *Secure Coding: Principles & Practices*. O’Reilly, 2003.
14. A. Hohl, L. Lewis, and A. Zugenmaier. Look who’s talking: Authenticating service access points. In volume 3450 of *LNCS*, pages 151–162. Springer, 2005.

15. G. Itkis. Cryptographic tamper evidence. In *Proceedings of the Conference on Computer and Communication Security*, pages 355–364. ACM Press, 2003.
16. J. Kelsey and J. Callas. Signed syslog messages. IETF Internet Draft, 2005. <http://www.ietf.org/internet-drafts/draft-ietf-syslog-sign-16.txt>.
17. D. Lie, C. A. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. C. Mitchell, and M. Horowitz. Architectural support for copy and tamper resistant software. In *ASPLOS*, pages 168–177, 2000.
18. C. Lonvick. RFC 3164: The BSD syslog protocol. 2001. <http://www.ietf.org/rfc/rfc3164.txt>.
19. D. New and M. Rose. RFC 3195: Reliable delivery for syslog. 2001. <http://www.ietf.org/rfc/rfc3195.txt>.
20. B. Pfizmann, J. Riordan, C. Stübke, M. Waidner, and A. Weber. Die PERSEUS Systemarchitektur, 2001.
21. J. Poritz, M. Schunter, E. V. Herreweghen, and M. Waidner. Property attestation: Scalable and privacy-friendly security assessment of peer computers. Technical Report RZ3548, IBM Corporation, 2004.
22. A.-R. Sadeghi and C. Stübke. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *Proc. of the 2004 Workshop on New Security Paradigms*, pages 67–77, 2005. ACM Press.
23. M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, pages 10–17, August 2001.
24. B. Schneier and J. Kelsey. Remote auditing of software outputs using a trusted coprocessor. *Future Generation Computer Systems*, 13(1):9–18, July 1997.
25. B. Schneier and J. Kelsey. Security audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2):159–176, May 1999.
26. F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, 2002.
27. Trusted Computing Group. TCG Backgrounder, May 2003.
28. J. Wang, Y. Yang, and W. Yurcik. Secure smart environments: Security requirements, challenges and experiences in pervasive computing. In *NSF Pervasive Computing Infrastructure Experience Workshop*, 2005.