2016 IEEE First Conference on Connected Health: Applications, Systems and Engineering Technologies

Cloud-based Secure Logger For Medical Devices

Hung Nguyen*, Bipeen Acharya*, Radoslav Ivanov*, Andreas Haeberlen*, Linh T.X. Phan*,
Oleg Sokolsky*, Jesse Walker†, James Weimer*, William Hanson‡, and Insup Lee*
*Dept. of Computer and Information Science, University of Pennsylvania, PA, U.S.A.
Email: {hungng,acharyab,rivanov,ahae,linhphan,sokolsky,weimerj,lee}@cis.upenn.edu
†Intel Labs, Intel Corporation
Email: jesse.walker@intel.com
‡Hospital of the University of Pennsylvania, PA, U.S.A.
Email: bill.hanson@uphs.upenn.edu

***Abstract*—A logger in the cloud capable of keeping a secure, time-synchronized and tamper-evident log of medical device and patient information allows efficient forensic analysis in cases of adverse events or attacks on interoperable medical devices. A secure logger as such must meet requirements of confidentiality and integrity of message logs and provide tamper-detection and tamper-evidence. In this paper, we propose a design for such a cloud-based secure logger using the Intel Software Guard Extensions (SGX) and the Trusted Platform Module (TPM). The proposed logger receives medical device information from a dongle attached to a medical device. The logger relies on SGX, TPM and standard encryption to maintain a secure communication channel even on an untrusted network and operating system. We also show that the logger is resilient against different kinds of attacks such as Replay attacks, Injection attacks and Eavesdropping attacks.**

I. Introduction

Medical devices today are becoming increasingly sophisticated and capable of handling more functionality, storing more information and interacting with patients better. In addition, medical device interoperability offers numerous advantages in terms of usability, patient safety and treatment efficacy. For instance, a number of interconnected medical devices can provide a more comprehensive picture of patient information that reduces medical errors and health care costs for patients.

While there are a number of benefits to medical device interoperability, it exposes medical devices to a new attack surface through the communication network. Recent research has shown that a vehicle can be hijacked through vulnerabilities in a vehicle's communication system [1] or through sensor spoofing [2]. Medical devices are now vulnerable to similar attacks since they are starting to join local to global networks and storing information in the network cloud. In addition to disrupting medical devices, such attacks have the potential of stealing confidential patient information and even affecting patient safety (e.g., an attacker could cause a medical device to diagnose a diabetic patient with excess glucose causing a life threatening situation called hyperglycemia). Therefore, interoperable medical systems need to be integrated with proper security and safety measures to prevent threats to patient safety and information. The need is so high that the U.S. Food and Drug Administration has recently unveiled a draft guidance for interoperable medical devices [3] with the intention to provide a reasonable assurance of safety and effectiveness for these devices.

A cloud-based data logger capable of keeping a complete record of medical device information from multiple network-integrated devices would be effective in adverse event analysis. Such a data logger operating in an adverse environment must be able to provide: confidentiality and integrity of stored data, tamper-detection and tamper-evidence. We focus on these security properties as they are sufficient for forensic analysis in case of attack happens.

There has been much work done on developing tamper-proof loggers. Some designs rely on published commitments to provide tamper-proofness and require a gosship protocol for distribution [4]–[7]. Others require some form of trusted hardware such as monotonic counter or secure memory. For instance, Sarmenta et al. [8] and Sinha et al. [9] use Trusted Platform Module (TPM) as a trusted computing base to guarantee tamper-proofness. However, they either assume an adversary who cannot perform sophisticated hardward attacks (e.g., proping memory or launching side-channel attack) or assume features that do not exist on current TPMs.

There has also been work on logging for medical devices. A prominent example is OpenICE, which provides a recording application on top of a framework for integrating healthcare devices and clinical applications [10]. It is, however, important to point out that OpenICE was not built with security features in consideration. To the best of our knowledge, our design is the first cloud-based secure logger applied for medical device.

In this paper, we address the problem of designing such a logger by making use of the Intel Software Guard Extensions (SGX) [11] and the TPM [12] whose use with the logger would help to prevent tampering of stored logs. SGX enables the creation of a secure container (called an enclave) that protects the integrity and privacy of the data inside it by isolating it from privileged software [13]. TPM is a cryptographic chip that provides shielded storage for authentication of stored information. We propose a design that leverages the guarantees provided by the SGX and the TPM to provide tamper-evidence to our system. Being an SGX enclave process that is protected from unauthorized access, our logger attempts to defend against most kinds of attacks discussed above. Additionally, our design is implementable today, using existing machines.

978-1-5090-0943-5/16 $25.00 © 2016 IEEE
DOI 10.1109/CHASE.2016.48

The remainder of this paper is organized as follows. In Section II, we formulate a problem statement and explore the attack space that a secure logger operates in. Section III then discusses the design architecture of the secure logger and provides a background on various components of the design. Section IV provides details of the logger's operation. In Section V, we discuss the guarantees and limitations of our design. In Section VI, we summarise our work and provide avenues for future research.

II. Problem Statement

As discussed in the previous section, our goal is to design a secure logger in the cloud for medical devices. In particular, we aim to develop a centralized logger that gets medical device and patient information from a dongle attached to each medical device. The data that it receives is stored as secure logs and can be later retrieved by an auditor for verification and analysis of tamper. Figure 1 illustrates a high-level overview of the logger's environment.

A secure logger in the cloud must satisfy certain security properties; specifically, it guarantees:

- *Confidentiality*: The system ensures that sensitive information is prevented from being illegitimately accessed by an unauthorized party.
- *Integrity*: The system ensures that data collected from medical devices is consistent, upon later retrieval.
- *Tamper-detection*: The system is able to detect any unauthorized access to the protected data.
- *Tamper-evidence*: Upon detection of tamper, the system also provides evidence to identify the attack.

We now explore the attack scenarios that a cloud based logger needs to defend against to meet these requirements. We illustrate the possible attack scenarios for our work in Figure 2 using the three dimensional *attack space* proposed by Teixeira et al. [14] that comprises of the adversary's system model knowledge, its disruption resources and disclosure resources. Disruption resources are resources that enable an attacker to affect system operation and availability while disclosure resources can be defined as resources that allow an attacker to obtain confidential information. System model knowledge often provides an attacker insider knowledge of the system to perform more complex attacks. In our case, an attacker would gain unauthorized knowledge of the secure logger. A combination of these resources would give the attacker power to cause more severe damage.

We follow these attack dimensions and discuss three attack scenarios under the system model introduced earlier and their potential impact. Even though the attack space is bigger, we limit this work to these three scenarios as we believe these are the most relevant for a logger operating in the cloud. Section III explains how our system defends against each attack.

1) **Eavesdropping Attack.** In an eavesdropping attack, the adversary utilizes the disclosure resources to gain unauthorized access to confidential data. In general, the attack relies on monitoring communication channels and gaining unauthorized information. An eavesdropping attack does not affect normal system operation. Obtained data from the attack can be patient data, medical device and logger technical information, which are potential system knowledge. An eavesdropping attack also allows the attacker to perform an unauthorized read on the disk.

2) **Injection Attack.** This particular scenario is where the attacker's goal is to inject false data or wipe evidence from the logger. By combining system knowledge and disruptive resources, he may try to tamper with the communication channels. The attack can be an injection of new messages, changes in the message content, re-ordering of message sequence, or a Denial of Service (DoS). Furthermore, it is also possible for the content in the storage to be altered.

3) **Replay Attack.** A replay attack scenario can be considered as a combination of eavesdropping attack and injection attack. The attacker first intercepts the transmission of data with the purpose of capturing a valid sequence of data, and then retransmits that valid data to the receiver. In this manner, an attacker with malicious intent could feed legitimate looking data while carrying out an attack on the device. The system would have no idea of the attack and the attacker would successfully carry out a man-in-the-middle attack.

III. Design of the Cloud-based Secure Logger

This section describes the design of the proposed secure logger. A dongle, with a corresponding driver, will be used as an attached module to a medical device. Being able to

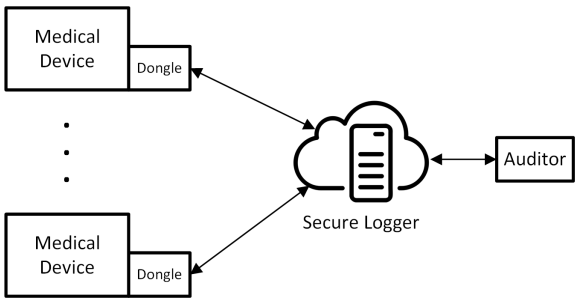


Fig. 1: A high-level overview of the logger.

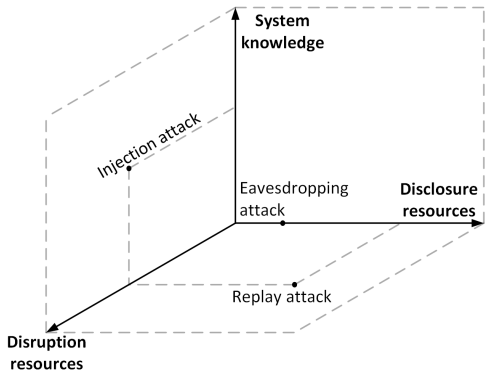


Fig. 2: Secure logger attack space.

communicate directly with the medical device, it will acquire the necessary information, use a monotonically increasing counter to stamp the data, and use SSL/TLS protocol to encrypt, provide integrity and transfer its contents to our logger, which is an SGX-enabled software application. This ensures that the logs received by our logger preserve integrity and confidentiality. The counter ensures that the information preserves freshness. When our logger receives medical device data, the logger first encrypts the content, and then generates and signs the hash of the entry. It then sends the log entry and corresponding signed hash to storage. It also keeps a copy of the latest signed hash in a TPM. With efficient hash chaining of stored logs, the latest hash stored in the TPM is sufficient to detect tamper because any kind of change or deletion of a hash value in a hash chain would make tamper evident. Upon detection of any attacks, the logger is able to sound an alarm on the system and raise alarm message on the User Interface. We discuss this design in detail in the following two sections.

In our design, we assume that the Intel SGX, the TPM and the dongle are implemented correctly and not compromised. We also make the usual assumptions that RSA and AES encryption are safe, an attacker cannot forge digital signatures and that the hash function is pre-image resistant, second pre-image resistant, and collision resistant with pseudo-random compression function.

In the remaining part of this section, we first describe various components of our design presented in Figure 3 and discuss the functionality of each component. We then propose proper usage of these components to establish secure communication channels to defend against the attack scenarios previously discussed.

A. Component Descriptions

1) Dongle: As a part of the system model, we have a dongle attached to the medical device to get information from the device and send to the logger. After establishing a connection with the logger, the dongle keeps sending messages to it without waiting for an acknowledgement for message receipt. The dongle also maintains a monotonically increasing counter that is used as a sequence number for the message before it is sent to the logger. Besides the counter, the dongle also maintains a queue of recently sent messages with the unique ID of the counter. If the logger has a missing sequence ID, it can ask the dongle again and raises an alarm if the dongle does not reply within allocated time. If it receives messages with duplicate ID, the logger immediately raises an alarm.

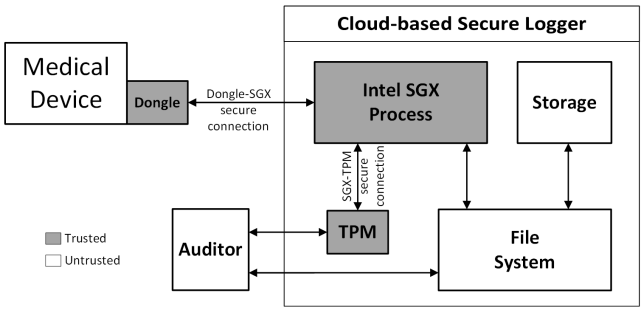


Fig. 3: Design architecture of the logger.

Before deployment, each dongle is assigned a private/public key pair with a corresponding digital certificate. The certificate includes information about the public key and the dongle identity and can be verified using the digital signature of the deployment team.

2) Intel Software Guard Extensions (SGX): SGX is the attempt by Intel at solving the problem of executing software applications in a remote computer owned by an untrusted party, with integrity and privacy guarantees [13]. SGX, a set of new instructions and memory access changes in the Intel Architecture, attempts to solve this problem by trusting a hardware at the remote computer to instantiate an enclave which is used for computation and information exchange. Enclave code and data are contained in the Enclave Page Cache (EPC), which is a subset of the Processor Reserved Memory (RPM) protected by the CPU from non-enclave memory accesses. Each enclave designates an area called the Enclave linear Address Range (ELRANGE) which can be used to map the code and sensitive data stored in the enclave's EPC. Special CPU instructions, such as EENTER (to execute code) and EEXIT (to quit execution), must be used by the enclave's host process to interact with the enclave, and it happens in protected mode. Exceptions are raised when a non-enclave access to a memory is attempted by a software, and also when a code fetch is attempted from inside an enclave to an address range outside that enclave. Before initiating communication with an enclave, a remote application performs software attestation to make sure that it is communicating with the correct enclave.

3) Trusted Platform Module (TPM): TPM is a cryptographic chip that is available on many motherboards today. The TPM is primarily a cryptographic engine that can perform encryption and hashing, and store the state of the internal software [15]. However, we are mostly interested in the small amount of non-volatile memory (NVRAM) provided by TPM to shield data. Since protected data in NVRAM can only be accessed via specific commands within an authorized session, it will be sufficient for us to store the latest hash value obtained from our logger application to help with forensic analysis.

To prove that it is genuine and complies with TPM specification, each TPM is embed with an unique *Endorsement Certificate* (EC), which contains manufacturer name, model number, version and most importantly, the TPM public key. EC is, in essence, a digital certificate stating that TPM identity has been properly created and embedded.

B. Secure Communication Between Components

The logger is an SGX process operating in the cloud, which we assume to be untrusted. Our assumption that the network is untrusted makes it imperative that the communication channels between the dongle and the SGX process as well as the logger and the TPM are secure. Next, we discuss how we make these channels secure to defend against attacks.

1) Dongle-SGX: To create a secure communication channel between the dongle and the SGX, we use the SSL/TLS protocol. Note that the dongle and the SGX process are themselves trusted, and assumed behaving correctly. Since it is infeasible to do I/O operations (other than memory stores and fetches) from within an SGX enclave, not all the SSL/TLS protocol will be implemented inside the enclave. However, we can guarantee the security by ensuring the data sent out from the enclave is encrypted and protected following the protocol.

To establish a connection, firstly, a handshake protocol is initiated between the dongle and the SGX where, messages are exchanged between the two parties to agree on the cipher suite - a combination of key exchange protocol, authentication algorithm and encryption algorithm to be used for the SSL session. Each of them use software attestation to verify that they are communicating with a trusted process. In particular, the dongle's certificate is used by the SGX to verify the dongle's identity and vice-versa. After this exchange, a session key is agreed upon to be used for encryption in all communication henceforth. This ensures that message transmission between them is encrypted and authenticated.

2) SGX-TPM: In order to protect the confidentiality of a shielded object, TPM defines each object as a pair of $(authValue, data)$. Access to protected data requires uses of a *Protected Capability*, which is a set of commands with exclusive permission to access shielded storage, where $authValue$ is used by the caller to authorize the action [12]. Upon running the first time, the SGX process generates the $authValue$, encrypts and stores it in the logger storage. The process also sends command to the TPM to create the corresponding protected object. As a result, both SGX and TPM share the same $authValue$.

To establish a trusted channel with TPM, the SGX process sends a TPM2_StartAuthSession() command including an initial *nonceSGX*, a value to generate the session key and location of protected object in TPM. After executing successfully, TPM returns a session key and the initial *nonceTPM*. The primary use of a nonce is to prevent a message being sent multiple times: for each message, sender includes the last received nonce and a newly generated nonce so that the receiver can verify if the message follows correct sequence. All the commands and responses are sent with an HMAC value to provide assurance that the message was not modified and came from the trusted entity with access to the HMAC key, which is defined as the concatenation of session key and *authValue*.

IV. Operation of the Cloud-based Secure Logger

In this section, we discuss, in detail, the operations of the secure logger. In particular, we describe how the log is generated, how the SGX process operates to maintain secure logs and the auditor operates to verify log correctness and detect tamper.

A. Tamper-Evident Log

The logger keeps a secure record of outputs from medical devices that is used for adverse events analysis. Based on the proposal of PeerReview [6] with some modifications in the way the hash chain is stored, a tamper-evident log is defined as an append-only linear list with each log entry $e_k = (s_k, c_k)$ where s_k is a strictly increasing, contiguous sequence number and c_k is the entry content. Additionally, each log also includes a hash value $h_k = hash(h_{k-1}||s_k||hash(c_k))$, which is the hash of the concatenation of the previous hash, the sequence number and the hash of the content. The base hash h_{-1} is predefined. In addition, h_k is signed by the SGX process using the private key and results in the corresponding sh_k. The final hash chain contains all the signed hash values sh_k.

Algorithm 1 Hash chain implementation

Variables:

$e \leftarrow$ list of log entries
$sh \leftarrow$ list of signed hash values
$k \leftarrow$ current iteration number
$s_k \leftarrow$ current sequence number
$h_{k-1} \leftarrow$ previous hash value
$m \leftarrow$ chunk constant

1: **procedure** ADDENTRY(content c)
2: $h_k \leftarrow hash(h_{k-1}||s_k||hash(c))$
3: $e_k \leftarrow (s_k, c)$, $sh_k \leftarrow$ sign(h_k)
4: **if** $k - 1 \mod m \neq 0$ **then**
5: sh.delete(sh_{k-1})
6: $k \leftarrow k + 1, s_k \leftarrow s_k$.next()
7: **return** sh_{k-1}

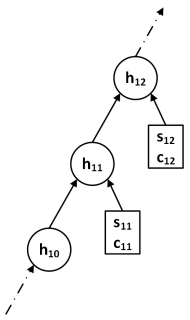
1: **procedure** TRUNCATE
2: $sh_i \leftarrow sh$.start()
3: sh.delete(sh_i)
4: **for** $j =\leftarrow i + 1$ to $i + m$ **do**
5: e.delete(e_j)

In practice, storing all the signed hash values is not required: given sh_i, sh_j where $i < j$, the hash chain from i to j can be re-computed to verify the integrity of the log. Therefore, we maintain a constant m indicating that the logger only stores signed hash value sh_i for every m log entries $(e_{i+1}, e_{i+2}, ..., e_{i+m})$. When the log storage is full and there is a need for log truncation to prevent log overflow, we can safely remove sh_i and all the log entries from $e_i + 1$ to e_{i+m}. Given sh_{i+m}, we are still able to re-compute the hash chain from e_{i+m+1} to the latest log entry as illustrated in Figure 4.

The log is the most important part of our design because the entire design relies on the logger being able to maintain a secure, untampered log of medical device information. The use of strictly increasing sequence numbers (line 6) and signed hash values (line 3) are important to ensure that logging is tamper-evident. We will discuss this process in detail in the next section when we present Auditor operation.

B. Logger Operation

Our logger has three stages: start-up, logging and shutdown. We explain each stage briefly and provide pseudocode of how to implement each stage. The algorithms below describe



(a) Hash chain

Log Entries	Hash Chain
e_k	sh_k
e_{k-1}	
⋮	⋮
e_{i+m+1}	
e_{i+m}	sh_{i+m}
e_{i+m-1}	
⋮	
e_{i+1}	
truncated	sh_i

(b) Storage content

Fig. 4: A sample hash chain and content in logger storage. Hash sh_i can be used to recompute hash chain from $e_i + 1$.

the operations between a single logger and a single dongle. However, this can easily be extended to multiple dongles as the logger can maintain connections with multiple dongles simultaneously.

Start-up: At logger start-up, the logger queries the latest log entry e_k, its signed hash sh_k from the File System service, the latest signed hash and the latest system status from the TPM. If the stored sighed hash is different from the one of TPM or any of the verifications during start-up fails, the logger will raise alarm. Otherwise, the logger stores the current sequence number s_k and the message sequence number n. Then, it attempts to establish a connection with the dongle as mentioned above. If the dongle does not respond, the logger stands by and waits for incoming connection from the dongle. If there are no errors, it is ready to go into the logging stage.

Logging: Upon receiving messages from the dongle, the logger first encrypts the content using its key, and generates and signs the hash of the new entry. It then sends the log entry and corresponding signed hash to storage. After that, the logger writes the signed hash in the TPM[1], which is where it keeps a copy of the latest signed hash. It also checks if the previous signed hash should be kept based on m and deletes if needed.

Shutdown: Upon receiving shutdown signal, the logger simply finishes storing all the received events and writing latest hash to TPM before exit.

C. Auditor Operation

The auditor is a software application that is responsible for auditing the logs and detecting tamper. It is a trusted SGX process that shares the logger's encryption key. By recomputing the entire hash chain from the first signed hash and the log entries, the auditor can compare with the last signed hash from TPM and verify the digital signature. If any of the verification steps failed, the auditor raises an alarm. If an attacker attempts to change a log entry in storage, it would result into a completely different hash value, and a different hash chain, and thus the attacker would be detected.

[1]Each TPM flash location can accept only a finite number of writes before it wears out. We can overcome the limitation by having the SGX process keep track of the number of write and move to another location after a pre-defined number of times.

V. Discussion Of Security Guarantees

We described the requirements for a secure logger in the cloud as: confidentiality and integrity of transmitted and stored data, tamper-detection and tamper-evidence. In this section, we discuss how we ensure that the design meets the requirements under each attack scenarios.

A. Eavesdropping Attack

Since the goal of an eavesdropping attack is gainning unauthorized access to confidential data, it only affects the confidentiality property of the system. To prove the preservation of confidentiality under this kind of attack, first consider the information exchanged between the dongle and the logger. To elaborate, in the event that an adversary intercepts information exchange between the dongle and the logger, he is not able to obtain information from that interception. The SSL/TLS exchange protocol has been proven to be secure [16] and is widely used in information exchange over the internet. We encrypt messages between the dongle and the logger using the SSL/TLS exchange protocol and hence, an eavesdropping attacker cannot cause damage.

Similarly, the secure logger also ensures that an eavesdropping attacker cannot learn log contents from the logger SGX process and logger storage. As a part of the trusted SGX enclave process, the logger is protected from unauthorized access. As a result, an eavesdropper cannot get information from the logger process. Further, the SGX process encrypts log entries before sending them to storage. Hence, even if an attacker attempts to attack the data storage, the encryption ensures that the stored data does not lose confidentiality.

Note that the communication between SGX and TPM is not encrypted. However, since the transfer data is not confidential, the confidentiality property still holds.

B. Injection Attack

An injection attack affects all the security properties of the logger. Beside preserving the confidentiality similarly under eavesdropping attack, the secure logger guarantees the integrity of the data via ensuring tamper-detection and tamper-evident properties. Any unauthorized access and message altering will cause data inconsistency and will be detected with detailed evidence.

Particularly, messages from the dongle to the logger are stamped with a monotonically increasing counter and sent over SSL/TLS secure communication channel. The integrity of the message is provided by the SSL/TLS protocol and the counter such that any changes in the message content will cause a failure in hash verification and any changes in message sequence will also cause a failure in counter verification of the receiver. Such failure is the evidence provided by the system to the auditor that an attempted attack has happened.

Additionally, the communication between SGX and TPM relies on the HMAC function, whose secret key is only known

by the SGX and the TPM. Together with random nonce value for each message, the logger ensures the integrity of this channel.

Finally, as mentioned above, we generate and sign hash to store log contents to the file system and storage. Since each hash value depends on its previous hash, any kind of alteration of the stored content creates a completely different hash chain branching off at the affected sequence number and an auditor is able to detect tampering [6]. The sequence ID of the affected log entry can be used as a tamper-evidence.

For the specific attack such as DoS attack, when the attacker's goal is making the logger unavailable, it won't affect the operation of the dongle or logger. Because the dongle keeps sending messages to the logger without waiting for acknowledgement, all the lost messages during attack can be retrieved after the logger detects missing message sequence IDs and queries the dongle again. However, such defense depends on the size of message queue that the dongle keeps.

C. Replay Attack

Consider a replay attack as the combination of eavesdropping attack and injection attack, it also affects all the security properties. Since the injected messages are authentic, to preserve security properties, the logger must be able to detect any duplicate messages. This can be achieved by the monotonically increasing counters as described in the injection attack.

The logger is also able to detect storage replay attack where the attacker clones a version of logger storage and later on restores it. Although the data is authentic, it is an older version. By comparing with the latest hash value protected in the TPM, the logger or the auditor can identify the attack.

Limitations: Our logging system is designed to make tampering of data detectable and thus, it detects and raises alarms when messages are altered. However, any behaviour that does not lead to altering of messages is not detected. For instance, when the logger does not get a reply from the dongle, it does not know if the dongle is under attack or just under normal shut-down. In addition, we protect the integrity of the information received by the dongle, but cannot guarantee the correctness of what it receives. Addressing these limitations are outside the scope of this paper. However, we do plan to explore and address these limitations in the future.

VI. Conclusion

In this paper, we have described a secure logging system that provides tamper-evidence to message logs that it receives from medical devices on a network. We leverage the use of trusted hardware in an untrusted network to design such a secure logger. The design guarantees that attacks attempting to modify or delete logs are detected.

This project is a first attempt at mitigating cyber threats to medical devices with their imminent increase in interoperability with other devices on a hospital network. This is an ongoing project and the next step is to implement our design on a simulated ICE environment. Then, we will experiment on a small part of the local hospital network. Furthermore, we also aim to extend the attack space by looking at attacks like Denial of Service in more detail. Finally, we plan to explore the Data Distribution Service (DDS) middleware for efficient data delivery and security guarantees for our logger.

Acknowledgment

This work was supported in part by NSF CNS-1505799 and the Intel-NSF Partnership for Cyber-Physical Systems Security and Privacy, NSF CNS-1035715, and the DGIST Research and Development Program of the Ministry of Science, ICT and Future Planning of Korea (CPS Global Center).

References

[1] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*. San Francisco, 2011.

[2] A. Rutkin, "spoofers use fake GPS signals to knock a yacht off course," MIT Technology Review, 2013.

[3] The Food and Drug Administration, "Design Considerations and Premarket Submission Recommendations for Interoperable Medical Devices - Draft Guidance for Industry and Food and Drug Administration Staff," 2016.

[4] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines." in *USENIX Security*, 1998.

[5] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, no. 1, pp. 21–41, 2004.

[6] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical accountability for distributed systems," in *ACM SIGOPS operating systems review*, vol. 41, no. 6. ACM, 2007, pp. 175–188.

[7] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging." in *USENIX Security Symposium*, 2009, pp. 317–334.

[8] L. F. Sarmenta, M. Van Dijk, C. W. O'Donnell, J. Rhodes, and S. Devadas, "Virtual monotonic counters and count-limited objects using a tpm without a trusted os," in *Proceedings of the first ACM workshop on Scalable trusted computing*. ACM, 2006, pp. 27–42.

[9] A. Sinha, L. Jia, P. England, and J. R. Lorch, "Continuous tamper-proof logging using tpm 2.0," in *Trust and Trustworthy Computing*. Springer, 2014, pp. 19–36.

[10] J. Plourde, D. Arney, and J. M. Goldman, "Openice: An open, interoperable platform for medical cyber-physical systems," in *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*. IEEE, 2014, pp. 221–221.

[11] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative Technology for CPU based attestation and sealing," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, vol. 13, 2013.

[12] Trusted Computing Group, "TPM 2.0 Library Specification," accessed: 2016-03-04. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_library_specification

[13] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Report 2016/086. http://eprint.iacr.org, Tech. Rep., 2016.

[14] A. Teixeira, D. Prez, H. Sandberg, and K. H. Johansson, "Attack models and scenarios for networked control systems," *Proceedings of the 1st international conference on High Confidence Networked Systems - HiCoNS '12*, pp. 55–64, 2012.

[15] J. D. Osborn and D. C. Challener, "Trusted platform Module evolution," *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, vol. 32, no. 2, pp. 536–543, 2013.

[16] H. Krawczyk, K. G. Paterson, and H. Wee, "On the security of the TLS protocol: A systematic analysis," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8042 LNCS, no. PART 1, pp. 429–448, 2013.