Proceedings of the INMM & ESARDA Joint Annual Meeting
August 23 – 26 & August 30 – September 1, 2021, Vienna, Austria

SMART GATEWAYS ENHANCED WITH BLOCKCHAIN-DERIVED TECHNIQUES TO ENABLE THE DETECTION OF ALTERATIONS OF DIGITAL SEALS' LOGS

Roberto Spigolon **Marco Sachy** **Claudio Bergonzi**

Cristina Versino **Stefan Nonneman**

European Commission Joint Research Centre, Ispra, Italy

ABSTRACT

This paper shows the potential of securing events logs produced by safeguards digital seals installed in a nuclear facility using a smart gateway enhanced with append-only data structures derived from Blockchain technology. In nuclear safeguards, digital seals are on the rise thanks to their ability to register and remotely communicate their status and all relevant events to the inspectorate, thus enhancing its monitoring capabilities. The digital signature of logs, performed by the seal itself, provides a way to verify their origin and integrity during data transmission. In specific cases however, it may not be feasible to transmit such logs outside the facility, due to either technical or juridical/political decisions. In these cases, inspectors will still need to physically retrieve the logs registered in the seals' internal memories. While very robust and secure by design, one cannot exclude the possibility for a digital seal to be tampered with or simply to fail, losing all the recorded events from the last inspection. We therefore propose to employ a smart gateway as a collector to record in real-time the events and the status of the seals in the facility. Using techniques derived from Blockchain technologies, such as hash chaining and Merkle trees, we can facilitate the detection of log tampering since a modification of a record invalids the whole log. Moreover, by publishing periodically an anchor to the log state on a public Blockchain, one can detect also tampering of the smart gateway itself. In this scenario, an inspector would retrieve all seals' logs directly from the smart gateway, being able to verify with certainty if an alteration took place or not. In this paper we present experimental results obtained creating a Proof of Concept that simulates the described approach. We discuss strengths and limitations, and compare this approach to a scenario not employing a smart gateway. We conclude that a smart gateway enhanced with Blockchain-related technology enables the detection of alterations to digital seals' logs when they cannot be remotely transmitted.

1. INTRODUCTION AND USE CASE DESCRIPTION

Nuclear Inspectorates, like the IAEA and EURATOM, implement Nuclear Safeguards as part of their core mandate. Nuclear Safeguards are defined as measures put in place to prevent the diversion of declared nuclear materials by states under the Non-Proliferation Treaty and the absence of undeclared nuclear materials and activities. Nuclear Material Accountancy and Control (NMAC), is the most known measure and it can be summarised as a set of reporting obligations and the relative physical inspections carried out to verify that what has been declared corresponds to the reality.

If NMAC is focused on gathering information and checking their correctness, another branch of Nuclear Safeguards, known as Containment and Surveillance [1], strives to ensure continuity of knowledge, which translates to the capability of the Inspectorate to have authentic and uninterrupted knowledge on an item [2]. This branch of Nuclear Safeguards make use of an extensive set of devices and equipment. Seals are one of them. They are tamper-indicating devices used to secure nuclear materials, documents, environments or any other sensitive item that shall not be manipulated or altered without the inspectorates knowing it. Their raison d'être is simple: provide a way to detect (not to impede) if any of such unauthorized manipulations has taken place during the time between one inspection and the next one.

Traditionally, seals are passive devices: someone needs to physically check the seal to assess if it is intact or it has been broken by someone to access the sealed item. In the latter case, a broken seal means that we cannot trust anymore the item that was enclosed and sealed. In the case of nuclear materials, this means that continuity of knowledge has to be considered broken and therefore all necessary measurements and assessments must be re-done to regain confidence on the nature or quantity of the material itself. In case of documents or restricted environments (e.g. the inspectors' room) we have to assume that the integrity of the underlying information, IT systems or devices, has been violated and therefore take all necessary actions to reconstruct the knowledge written on such documents, and to rebuild the systems.

The main inconvenience of passive seals is that they need a physical inspection to determine if they have been tampered or not. In contrast to this, modern seals are active devices: they are formed by an electronic device connected to the seal (e.g. an optical fibre cable, like in the EOSS seal [3]) that is capable of immediately recording the exact date and time of when a certain event happened (e.g. the fibre cable was disconnected). This greatly adds to the kind of information that a seal gives us, as it is now possible to know when a certain event took place rather than having to assume that the last time the seal was intact was the last physical inspection. Moreover, modern seals also have the capability to communicate remotely the registered events (or to respond to remote queries). Such capacity, if exploited properly, could give to the Inspectorate an early warning that something went wrong and that their intervention is needed as soon as possible.

Unluckily, the remote connectivity is not always possible. Whether it is national legislation that forbids the use of electronic communication means for sensitive installations, or technical issues (e.g. the lack of a reliable connection towards the Inspectorate), in many cases the inspectorates still have to rely on physical inspections to retrieve the seals' recorded events. This has some disadvantages:

1. The late detection of tampering events may lead to additional actions (and relative costs) to be performed to understand what happened and to rebuild the lost continuity of knowledge, whether an early detection could lower that.
2. The seal and its memory may broke before the physical inspection takes place, leading to the possible loss of the recorded events
3. Well-funded attackers may find a way to tamper with the seal and alter the recorded events before the next inspection takes place

To address with the second and the third issues, we studied the application of a smart gateway enhanced with Blockchain-derived techniques to build a tamper-evident log locally in the facility in order to both achieve data redundancy and to implement a more robust way to detect tampering of the seals' events. In Section 2 we explain the working principles of the tamper-evident log. To test if such system is actually capable of increasing the detection of alteration to the seals' events, we developed a Proof of Concept that resembles, at a high level, what a real scenario could be. Details on its architecture and results of the tests performed are discussed on Section 3.

2. A TAMPER-EVIDENT LOG BASED ON MERKLE TREES

Blockchain and related technologies, firstly appeared with Bitcoin [4] in 2009, make use of data structures specifically designed to prevent data manipulation. A Blockchain, as the word suggests, has a data structure in its core elements that is formed by a chain of blocks where each block is connected to the previous one by the insertion of the previous block's hash in its header. For the sake of clarity, in this paper we will refer to this data structure with the term *chain of blocks* to avoid confusion with the Blockchain intended as a network of nodes that independently agree on the content of their local version of the chain through a consensus mechanism.

In applied cryptography a hash results from a cryptographic operation that from an input of any size calculates an output of a fixed (usually smaller) size. For each input value there is a single output, but the opposite is not true: from an output, there are multiple (infinite in fact) input values that lead to it. The hashing function is therefore mathematically irreversible. Moreover, even a single bit changed on the input causes a huge difference on the output. Such difference, unless the used hashing algorithm has vulnerabilities, is not predictable, so that there is no way other than pure brute forcing to try to obtain a specific output [5]. This makes sure that each alteration of a single block inside the chain would make it invalid as the hashes would not correspond anymore. This chain of blocks is therefore a tamper-evident data structure in the sense that it is possible to spot if such data has been altered after their original insertion.

Another tamper-evident data structure commonly used on Blockchains is the Merkle tree [6]: a data structure, formed of nodes, that represents the shape of a tree seen from the upside down. The root element (also known as Tree Head) is the only element that has no parents, while the leaves elements are all those elements that do not have any child. All other elements have maximum two children. In a Merkle tree, each leaf is the hash of a piece of information, and all the other non-leaf nodes are the hashes of the concatenation of their children (**Figure 1**). As with the chain of blocks, also within a Merkle tree each alteration of any node would create inconsistences between the hashes.

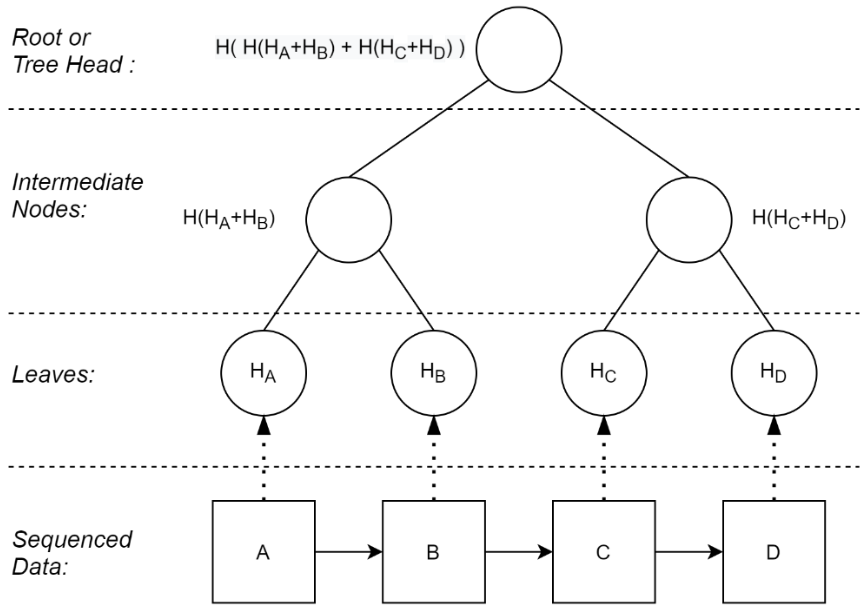

Figure 1: Structure of a Merkle tree based on the four sequential records A, B, C and D

Merkle trees are more suited than simple chain of blocks to build tamper-evident logs because they enable the verification of both the inclusion of specific record on the log and the consistency of two different versions of the same tree (i.e. the most recent version of the tree has indeed been derived from the previous version, respecting the append-only property) without checking the complete log, while on a chain on blocks each consistency proof requires the verification of the complete chain.

The most important example of their employment to build tamper-evident logs is the Certificate Transparency project [7], born to address the lack of transparency on the work of Certificate Authorities. Within such initiative, each Certificate Authority implements a public tamper-evident log based on a Merkle Tree, and each new certificate has to be appended to this log. Auditors keep those logs monitored to verify that the Certificate Authorities are acting with integrity. Moreover, Internet browsers can independently verify that the certificates they receive while browsing have actually been recorded on these logs.

A tamper-evident log based on a Merkle Tree periodically publish or advertise the current tree head, which is therefore a public information. To verify the inclusion of a specific record into the log, we need to reconstruct the operations that lead from that record to the tree head. If we can verify that is indeed possible to connect the record to the advertised tree head, we know for sure that such specific record has been included in the tree. An example is depicted in **Figure 2**. The same kind of process can be used to obtain a consistency proof, which is the proof that a tree containing *n* elements, represented by its tree head, was built starting from a previously advertised tree head of an *m* elements tree, where *m* < *n*. In other words, a consistency proof verifies that the append-only property of the tree was indeed respected.

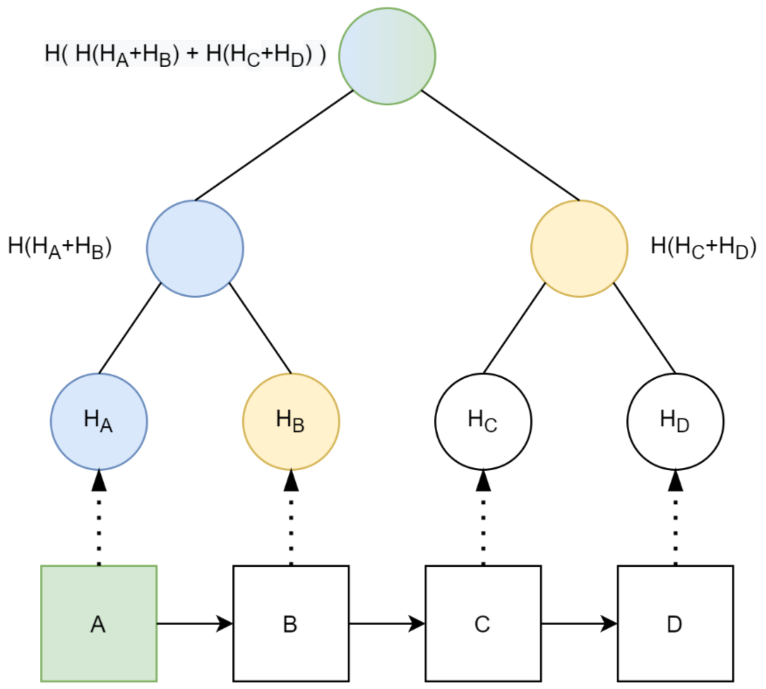

Figure 2: Example of the Proof of Inclusion needed to verify that the record A has indeed been used to produce a known tree head. In green are highlighted the known information (A and the tree head), in yellow the intermediate nodes included in the proof, and in blue the calculated intermediate nodes needed to verify if the re-calculated tree head corresponds with the known one

In our seal-related scenario, we are interested in having a tamper-evident log where we are able to detect if: (1) a specific seal event was altered after the insertion in the log; (2) a specific seal event was removed after the insertion in the log; and (3) the complete log was reconstructed from scratch to hide manipulations

The Merkle tree based log here described is able to satisfy all these conditions. The first two are easily detected as manipulation or deletion of the data that is used to build the Merkle tree will make it immediately invalid: on the first case, there will be a mismatch between one the tree's leaves and the hash calculated from the altered event; on the second case, the Merkle tree will have a leaf that does not have any more a corresponding event. The third case, instead, greatly depends on the periodical publication of the tree heads, as we would not be able to detect that a Merkle tree has been completely reconstructed without knowing the previously advertised tree heads. For this reasons, it is important that such tree heads are stored on a secure repository that ensures the integrity of its data. For this scope, in this paper we propose to use a Public Blockchain, since:

1. A Public Blockchain is not owned nor controlled by any particular entity. Therefore we do not have to trust a third entity to behave correctly.
2. A Public Blockchain, thanks to the high number of nodes and the resilience of its consensus mechanism, is capable of providing a high level of integrity of the data recorded on it, so much that it is nowadays common to say that a Public Blockchain guarantees the practical immutability of its data.
3. For the properties of the hash operation explained before, a tree head alone does not convey any type of information whatsoever on the underlying data and therefore cannot be considered confidential information.

To do so, the tamper-evident log will have to connect to a local Blockchain node that will broadcast the tree heads to the Blockchain network. An internet connection may not be fundamental for this broadcast, as nowadays there are already some solutions that broadcast blockchain transactions using satellites. Nevertheless, in case relying on a Public Blockchain is not practical due to a complete absence of any kind of external connection whatsoever, it is nevertheless possible to use a traditional repository or a write-once media, provided it is separated from the log and both physically and cyber-secured at the highest level. After the first inspection, the inspectors could retrieve the recorded tree heads and bring them to the inspectorate, creating a baseline of tree heads that will enable the detection of the complete log reconstruction.

3. PROOF OF CONCEPT ARCHITECTURE, IMPLEMENTATION AND TESTING

To verify the feasibility of the system foreseen in this paper, we built a Proof of Concept (PoC) that resembles and simulates the context described in the Introduction section. The PoC is formed by the following components:

- A set of simulated digital seals;
- The Gateway service, responsible for querying the seals and checking the validity of the received events;
- The Tamper-evident Log Service that receives the events and register them on a DB, updating the Merkle Tree;
- A Public Blockchain node, used to store periodically the Merkle tree heads.

The simulated seals were developed as web services (one per seal) that accept queries regarding the availability of events, and reply sending them back. We took into consideration the Active Optical Loop Seal (AOLS) currently under development at the Joint Research Centre, and recreated (at a high-level) what is the currently proposed workflow to get an event from this specific seal: the Gateway first query the seal asking for the number of available events, getting such number as an answer; then it manually queries each single event, embedding the sequence number of the event into the query. In the real world different communications protocols than http would most probably be used. It is not relevant for the purpose of this work to identify in this phase which are the proper communication means, protocols, and media (i.e. wired or wireless).

The simulated seal is programmed in a way to create a new event each minute. Each event is a 89 byte record that follows a well-defined structure. In our simulation, we are using a set of 65 pre-generated events.

The tamper-evident log has been developed using the code of the open source project Trillian [8] [9]. Such code is designed with resilience and scalability in mind, and is therefore divided in components (**Figure 3**). The events to be stored are sent to the log by what in the Trillian project is called a Personality [10], here renamed *Gateway Service* for our purposes. Its duties are to query the seals to get new events, verify their authenticity (i.e. check their digital signature) and verify the correctness of their format. All compliant events are sent to the *Log Server*, which will record them on the database, as *unsequenced* events ready to be processed. The *Signer* is the most important component, as it is responsible to get all new unsequenced events, give them a unique ordering (so that they may become sequenced) and evolve the Merkle tree associated to the log using the new events, digitally signing the new tree heads.

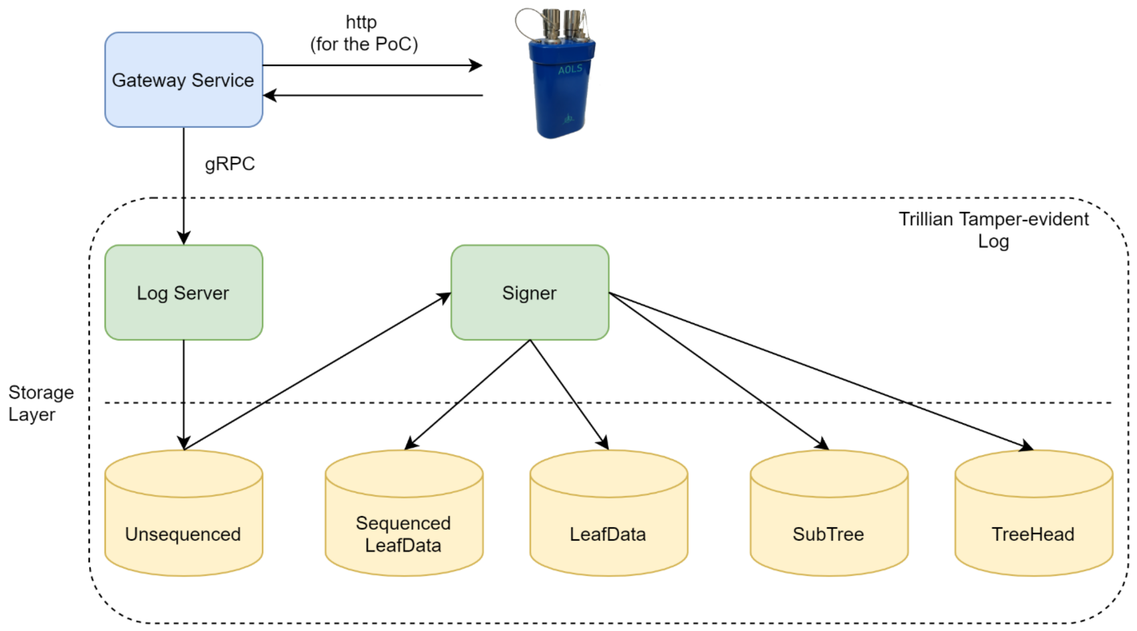

Figure 3: Internal structure of the Trillian based tamper-evident log

Each component may be implemented on a separate server. Multiple instances of the same component may be spawned, if needed, to add redundancy and scalability to the system. Note however that, for the specific nature of the log where it is important to have a deterministic order of insertion of the events, there may be only one active Signer at a time. Therefore, for that specific role, an election is periodically held, and multiple Signer instances will guarantee resilience but not scalability. In our PoC we are running each component of the tamper-evident log on the same machine.

The Public Blockchain node is used to periodically store the tree heads on the Blockchain. Its implementation depends on the specific Blockchain used. We relied for this PoC on a previous research [11] that focused on registering the emission of new digital certificate on a Public Blockchain. In that research we experimented with Bitcoin [4], Ethereum [12] and Algorand [13]. The specific choice of which one to use is not in scope of this paper, and not relevant to its results.

In **Figure 4** we depicted the high-level schema of the PoC Architecture, together with the workflow used to store events on the log. The PoC was initially developed and executed on a Linux machine with an Intel I7 and 16 GB of RAM. Considering that there may be specific context where the tamper-evident log must run on standalone devices small in size and that may have to run on batteries, we tested the feasibility of having the Gateway and the Log services running on a Raspberry PI 4, Model B (CPU: 1.5GHz 64-bit quad-core Arm Cortex-A72 CPU, RAM: 4GB). Using the codebase as it is, without optimizations for embedded devices, we did not face any performance issue. We are therefore quite confident that the system may also run on even less powerful devices, if needed, with the proper optimizations. Obviously, the storage needs will vary based on the number of events stored. More tests on this aspect will be carried out in the following phases.

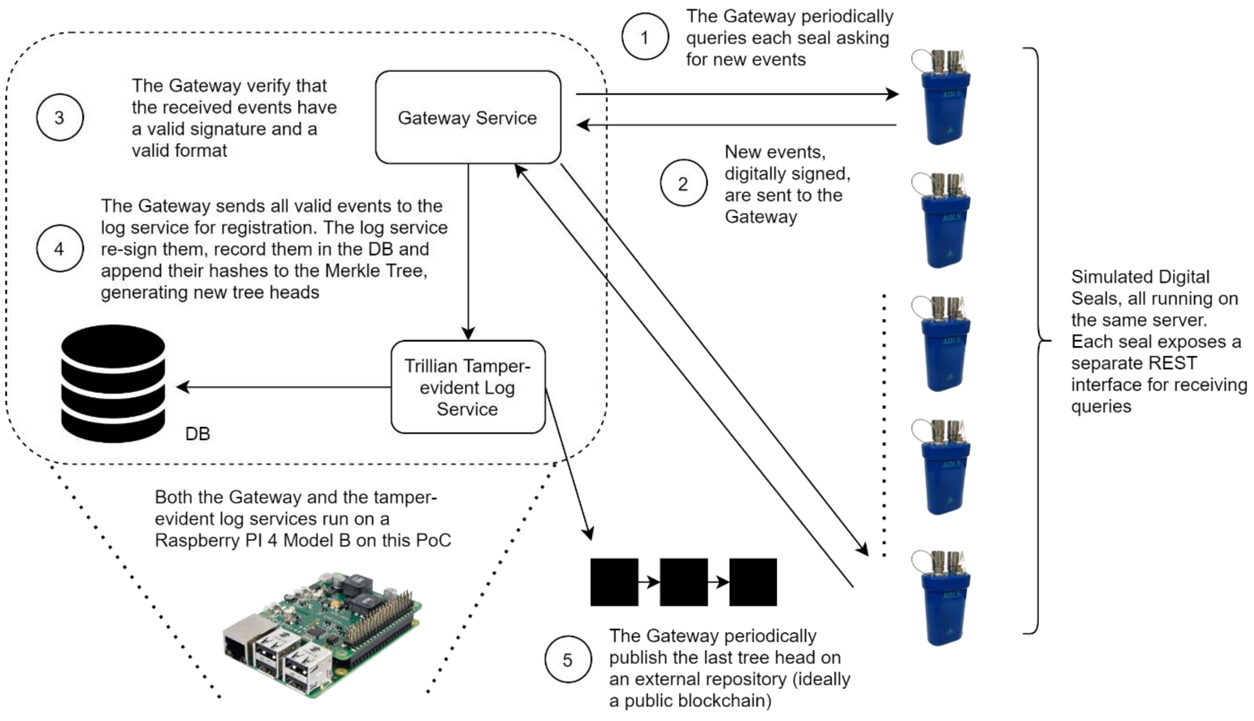
1
The Gateway periodically queries each seal asking for new events
3
The Gateway verify that the received events have a valid signature and a valid format
Gateway Service
2
New events, digitally signed, are sent to the Gateway
4
The Gateway sends all valid events to the log service for registration. The log service re-sign them, record them in the DB and append their hashes to the Merkle Tree, generating new tree heads
Simulated Digital Seals, all running on the same server. Each seal exposes a separate REST interface for receiving queries
Trillian Tamper-evident Log Service
DB
Both the Gateway and the tamper-evident log services run on a Raspberry PI 4 Model B on this PoC
5
The Gateway periodically publish the last tree head on an external repository (ideally a public blockchain)

Figure 4: PoC high-level architecture and workflow steps to record events on the tamper-evident log

3.1. Execution Flow and Tamper detectionTests

The workflow to register events on the log is the following:

1. The Gateway service has a list of authorized seals installed in the facility. It periodically queries them to ask for the presence of new events, using the relevant commands that each seal implements.
2. If any new events (with regard to the last interrogation) are present, the gateway requires and receives them
3. The Gateway validates the signature and the format of the received events. If everything is valid, the received events are forwarded to the Log service
4. The Log service, in accordance to its architecture depicted in **Figure 3**, will sequence the received events, store them on the underlying database, and evolve the Merkle tree associated to the log using the hashes of the received events. The new tree head is signed and timestamped by the Log Service.
5. Periodically, the tree head of the Merkle tree (that is dynamically changing each time a new event is stored) is published on a Public Blockchain, via a separate Blockchain node. The publication is checked to guarantee it has been performed correctly.

To verify that the log developed within this PoC was indeed tamper-evident, we tested three types of attack scenarios, checking if we were able to detect that some tampering actually took place.

The first attack scenario consists of the alteration of a single event recorded on the log: we manually modified the entry of an event on the database. Such action immediately creates a mismatch between the event registered on the DB and its hash used to build the Merkle tree. Two cases here are possible:

1. The original hash is not modified. This keeps the Merkle Tree valid, but the inconsistence between the recorded event and its hash is detected during the verification process of the log, where all hashes are recalculated starting from what is present on the DB, and such mismatch is uncovered
2. The original hash is substituted with the one generated from the altered event. This makes the Merkle Tree invalid as the tree head is not anymore built starting from the leaves' hashes. Such problem is detected each time we require a consistency proof and verify it

The second attack scenario consists of the deletion of a single (or multiple) event recorded on the log: we manually deleted the entry of an event on the database. The results were that the Merkle Tree was deemed invalid because its size was not compatible anymore with the number of recorded events in the log. Moreover, checking the hashes on the tree with the hashes of the recorded event, it is possible to spot which leaf does not have a corresponding hash anymore. Considering as well that each time an event is added to the log, a new signed and timestamped tree head is generated and recorded, we know exactly at which time the missing event was originally inserted into the log. This information is available simply checking which the new tree head was when the orphan leaf was generated.

The third attack scenario consist on the complete erasure of the log, and its reconstruction either with completely new events or with the same ones with the exclusion of one or more undesired events. Performing this attack requires a total manipulation of the database where the log is stored. By doing so, the new reconstructed tree will indeed be formally valid from the log point of view. But we would nevertheless be able to spot that something happened comparing the published tree heads with the new ones, surely spotting some mismatches. The resilience to this last attack scenario is therefore dependent on the repository where the tree heads are periodically published. Such repository should therefore be physically located on a separate place, or at least be physically secured (i.e. put under seal), and hardened. An ideal solution prescribes that such repository be public, so that external observers may notice if some alterations take place, and not controlled by the same entity controlling the log. As said already before, a Public Blockchain seems to be the perfect solution as, thanks to its properties, it gives us strong data integrity features making the recorded data practically immutable. It should be important to note that tree heads should not be considered in any case confidential data, as they are "hash of hashes" and carries no information whatsoever on the underlying information.

Considering instead a scenario without the Gateway equipped with a tamper-evident log, as a comparison, we have that: (1) each seal must be individually interrogated to retrieve the recorded events; (2) there is no data redundancy, and each seal is a single point of failure; and (3) in case a specific kind of seal contained some vulnerabilities that enable a sophisticated attacker to alter or delete the recorded events, there could be no way to straightforward way to detect it.

CONCLUSIONS AND WAY FORWARD

In this work we tested the implementation of a tamper-evident log for storing the events of digital seals locally in a nuclear facility, especially in the case where direct connectivity of the Inspectorate to the seals is forbidden by the national law or not possible for technical reasons. For this scope, we reused the codebase of Trillian: an open source project initially develop for the Certificate Transparency initiative. We built a PoC that implements, inside a Raspberry Pi 4 Model B, the tamper-evident log and a Gateway service that is used to query the seals in order to get new events and store them on the log. The tests performed verified that the implemented log is indeed tamper-evident (i.e. it shows that an alteration of the recorded events took place) in the three attack scenarios hypothesized: (1) the alteration of a specific event on the log's database after its original insertion, (2) the deletion of a specific event already stored in the log, and (3) the complete erasure and reconstruction of the log to hide manipulation or deletion of the stored events. On all three cases, it was possible to spot that an alteration took place on the log. In particular for the third attack scenario, and in addition to what the Trillian project natively offer, the alteration of the complete log can be spotted only if the Merkle tree heads are periodically published or stored on a separate secure repository. We experimented using a Public Blockchain for this role, as its intrinsic features gives to the recorded data a "practical" immutability; nevertheless, different kind of repositories could be used for this role, as a write-only memory, public media (i.e. newspaper or social media), or a secured external repository. Obviously, the confidence of the absence of tampering in the third attack scenario is as strong as the repository used for this role.

The system here designed, other than increasing the assurance that the retrieved events have not been manipulated after their original recording on the log, also facilitate the inspectors work as they could retrieve all the events directly from a single place, and it also provides data redundancy, as the seals' events will not only be stored in their internal memory, but also on the tamper-evident log, enabling their recovery also in the case a seal fails losing its recorded events before the inspection.

In our future works, we will continue exploring this approach, as a complementary technology to digital seals, focusing also on its applicability in the operator installed seals use case, where additional security is fundamental to balance the responsibility sharing between the inspectorate and the operators.

REFERENCES

[1] International Atomic Energy Agency, Safeguards Techniques and Equipment, Vienna, 2011.

[2] D. S. Blair and N. C. Rowe, "A Global Perspective on Continuity of Knowledge: Concepts and Challenges," in *Proceedings of the 55th INMM Annual Events*, Atlanta, Georgia, 2014.

[3] G. Neumann, B. Richter, C. Korn, M. Goldfarb and M. Villa, "The Qualification of the EOSS Seal - Test Programme and Results," in *25th Annual Meeting - Symposium on safeguards and nuclear materials management - Proceedings*, Stockholm, 2003.

[4] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2009. [Online].

[5] R. Sobti and G. Geetha, "Cryptographic hash functions: a review," *International Journal of Computer Science Issues (IJCSI),* 2012.

[6] R. Merkle, *Secrecy, authentication and public key systems/ A certified digital signature,* Dept. of Electrical Engineering, Stanford University, 1979.

[7] B. Laurie, A. Langley and E. Kasper, "IETF RFC 6962 - Certificate Transparency," June 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6962. [Accessed 19 July 2021].

[8] Google, "Trust your data with a tamper-evident log," [Online]. Available: https://transparency.dev/. [Accessed 19 July 2021].

[9] Google, "Trillian Github Repository," [Online]. Available: https://github.com/google/trillian. [Accessed 19 July 2021].

[10] Google, "Trillian Personalities," [Online]. Available: https://github.com/google/trillian/blob/master/docs/Personalities.md. [Accessed 19 July 2021].

[11] R. Spigolon, M. Sachy, S. Nonneman, R. Neisse, I. Maschio and I. Nai Fovino, "Using Public Blockchain for the management of Public Key Infrastructure to strengthen Nuclear Safeguards," in *Proceedings of the INMM 61st Annual Meeting*, Baltimore, MD, USA, 2020.

[12] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," 2014. [Online]. Available: gavwood.com/paper.pdf.

[13] S. Micali and J. Chen, "Algorand," 26 May 2017. [Online]. Available: https://algorandcom.cdn.prismic.io/algorandcom%2Fece77f38-75b3-44de-bc7f-805f0e53a8d9_theoretical.pdf.