MalRec: A Blockchain-based Malware Recovery Framework for Internet of Things

Ahmed Lekssays
alekssays@uninsubria.it
Università degli Studi dell'Insubria
Varese, Italy

Giorgia Sirigu
gsirigu@uninsubria.it
Università degli Studi dell'Insubria
Varese, Italy

Barbara Carminati
barbara.carminati@uninsubria.it
Università degli Studi dell'Insubria
Varese, Italy

Elena Ferrari
elena.ferrari@uninsubria.it
Università degli Studi dell'Insubria
Varese, Italy

ABSTRACT

IoT devices have been considered an attractive target for malware (e.g., botnets) due to their low computational resources and lack of security measures. The literature focuses on detecting malware, but less attention is given to recovery solutions. In addition, with the development of data processing regulations in different countries, a need for transparent recovery systems that can help organizations present their due diligence arises. This work proposes a blockchain-based backup policy enforcement framework for IoT where an organization can formalize backup policies and enforce them. We have run our solution under extensive tests that show that it can be deployed in real-life IoT environments, despite the limited computational resources of IoT devices.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

KEYWORDS

malware recovery, internet of things, backups

ACM Reference Format:
Ahmed Lekssays, Giorgia Sirigu, Barbara Carminati, and Elena Ferrari. 2022. MalRec: A Blockchain-based Malware Recovery Framework for Internet of Things. In *The 17th International Conference on Availability, Reliability and Security (ARES 2022), August 23–26, 2022, Vienna, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3538969.3544446

1 INTRODUCTION

According to PurpleSec[1], cyberattacks had a 600% increase during the COVID-19 pandemic in 2020. Hackers exploit security vulnerabilities to perform malicious activities such as data theft, espionage, etc. In general, cybercriminals attack different targets using malware—i.e., *malicious software*—such as viruses, worms, spyware, ransomware, etc. Indeed, in 2020, 61% of organizations experienced a malware incident spread among their own employees. This number grew to 74% in 2021 because smart-working increments the remote connection. One of the preferred scenarios for cyberattacks is represented by IoT devices[2]. IoT devices are estimated to reach 41.6 billion by 2025 with an overall annual growth rate of 28.7% from 2018 to 2025 [4]. IoT represents a unique opportunity to significantly increase productivity, reduce energy consumption, and improve the process in which they are used. However, they suffer from various security issues such as open ports, system updates not being performed, and weak security mechanisms. Attackers can easily compromise these devices and perform malicious actions based on various security issues, as stated by OWASP IoT Top 10.[3]

One of the main challenges for the recovery systems is ransomware attacks. Ransomware are a type of malware that encrypts files and make devices unresponsive. In 2021, only 6 ransomware families were responsible for a loss of $45 million paid in cryptocurrencies to their developers. They attacked, in the US only, 600 hospitals through medical IoT devices[4]. According to TrendMicro, a leading security company, IoT ransomware have been evolving in the last few years and they target IoT devices in the medical, food, and agriculture sectors.

In order to reduce the impact of different malware attacks on IoT systems, in this paper, we introduce a malware recovery framework, called MalRec, that allows compromised devices to recover and return to a safe state. Our solution is based on blockchain to store backup metadata and enforce *backup policies*, that is policies stating constraints on the backup process (such as the required number of replicas, the frequency of backups, etc).

MalRec exploits blockchain to help companies comply with different standards and regulations (e.g., GDPR, ISO/IEC 27040, NIST SP 800-171, etc.) by enforcing backup policies through smart contracts, which are autonomously executed programs hosted on blockchain. For example, suppose an organization must comply with a backup policy that states that the organization must have daily backups with 3 different replicas in different storage media. In that case, a smart contract can ensure that any submitted backup complies with this policy by verifying the submission time and the locations of the backups. Smart contracts can also be used to retrieve backups'

[1] https://purplesec.us/resources/cyber-security-statistics/

ARES 2022, August 23–26, 2022, Vienna, Austria

ACM ISBN 978-1-4503-9670-7/22/08.
https://doi.org/10.1145/3538969.3544446

[2] https://www.iotworldtoday.com/2021/09/17/iot-cyberattacks-escalate-in-2021-according-to-kaspersky/
[3] https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf.
[4] https://illinois.touro.edu/news/the-10-biggest-ransomware-attacks-of-2021.php

information that satisfies a specific filter, such as the submitter and the timeframe when the backups were submitted. Blockchain ensures a transparent and immutable verification of the policies that companies may want to enforce. In addition, it ensures that all interactions with the blockchain are authenticated through cryptographic primitives. To ensure the confidentiality of the stored backups MalRec exploits public-key cryptography to encrypt backups before uploading. Furthermore, it ensures the integrity of the backups via digital signatures. Blockchain, by design, is transparent, so it provides accountability and audit features since all actions are authenticated. MalRec is also agnostic from the used storage media. Hence, companies may choose to store backups locally or in the cloud (e.g., Amazon AWS, Google Cloud, Microsoft Azure, etc.), being able to comply with different regulations (e.g., GDPR) regarding the storage location. The availability of the backups is ensured by the selected storage service.

There are some challenges to ensuring a sound backup generation. For example, a tampered device may generate an invalid backup containing malware traces or corrupted data. The recovery system should avoid that a device downloads those backups during the recovery phase. Otherwise, it would spread the malware again after it has been cleaned up from it. To avoid this situation, MalRec label invalid backups by exploiting the available detection systems.

MalRec is designed to allow organizations to manage backups in heterogeneous networks. It can be used with different computing paradigms, such as fog or edge computing, or with no layer between the IoT devices and the servers. The reference scenario for Malrec usage consists of single or multiple organizations with different IoT devices and possibly behind aggregators. When IoT devices generate real-time data, the data volume grows drastically, which brings a need for aggregators that receive these amounts of data and process them. These aggregators can serve as an edge or fog layer when IoT devices have limited computational and storage resources that do not allow them to store and process the data they generate. In our scenario, the aggregators can serve as a helper to aggregate and submit IoT devices' backups. Organizations define their own backup policies enforced through smart contracts. Moreover, the organizations' administrators control their respective IoT devices' public-key cryptography pairs used to encrypt and decrypt backups. Figure 1 shows an example of a possible scenario for MalRec.

Many efforts have been focusing on malware detection problem in IoT, but few are focusing on the malware recovery problem. Among them, the work in [6] investigates the problem of reliable IoT architectures (and backup systems as a main component for fault tolerance). It proposes a generic architecture for reliable and fault-tolerant IoT networks under different computing paradigms (e.g., for, edge, etc.). It relies on Mobile Agents that serve as resource and network monitoring agents to share the information about device links' states. In addition, they consider computing layer agents (e.g., fog, edge, etc.) as backup agents for the IoT devices.

The authors of [12] focused on personal health care data due to their extreme importance. They proposed an architecture for a reliable and fault-tolerant healthcare system composed of IoT devices and gateways (e.g., devices that aggregate data). Data of each IoT device is backed up in multiple gateways.

The work in [8] proposed a model to improve data availability that takes different circumstances into account (e.g., force majeure, technical interrupts, etc.). The model is based on the 3-2-1 backup strategy that requires the data to be stored in 3 replicas in two different locations where 1 of them is offsite.

MalRec differs from the above-discussed approaches on many levels. First, MalRec is not tied with only one use case or organization. It can support multiple use cases with different organizations at the same time, unlike the work in [6, 8, 12]. Second, unlike the work in [8, 12], MalRec proposes a modular architecture to enforce multiple backup policies at the same time. Third, MalRec is designed with recovering from attacks in mind. In this sense, it provides a gateway to report malware attacks able to automatically invalidates the backups containing malware traces. Finally, the discussed approaches are centralized solutions while MalRec is decentralized since it leverages blockchain to enforce the policies in a decentralized manner.

In summary, in this paper, we propose a blockchain-based backup policies enforcement framework that helps organizations comply with different regulations and standards regarding the processing of users' data. Thus, the main contributions of this proposal can be summarized as follows:

- it exploits blockchain to ensure the immutability, transparency, confidentiality, and security of users' backed up data;
- it implements a framework to recover data after a malware attack, leveraging on smart contracts for automatic backup retrieval and backup policies enforcement;
- it has been tested extensively the scalability of our framework by evaluating its performance in terms of throughput and latency.

Outline. The rest of the paper is organized as follows. In Section 2, we introduce blockchain and different backup techniques. Section 3 describes the requirements that drive the design of MalRec. Section 4 illustrates the overall workflow, and the details of backups upload, invalidation, and retrieval. Then, in Section 5, we discuss the implementation details. Section 6 analyzes the experimental results. Finally, we conclude the paper in Section 7.

2 BACKGROUND

In this section, we introduce some background information on backup techniques and blockchain.

2.1 Backup Techniques

The term "backup" indicates a set of activities that allow a system to perform recovery operations. It refers to the process of creating copies of data and storing it in a safe place to be accessed in case of an incident.

There exist three main backup types [5]. One is a full backup that is copying all data into external storage. Its main advantage is the speed of recovery; however, it requires large storage space and a long time to be created. For this reason, it is usually not suitable to be used alone, but in combination with the other two backup types: incremental and differential. The first one consists of copying only the data that differ from the previous backup. Indeed, it requires less storage and time to be created. In contrast, differential backup is a middle ground between the other two in terms of storage space and needed generation time, as it stores all the changes from the

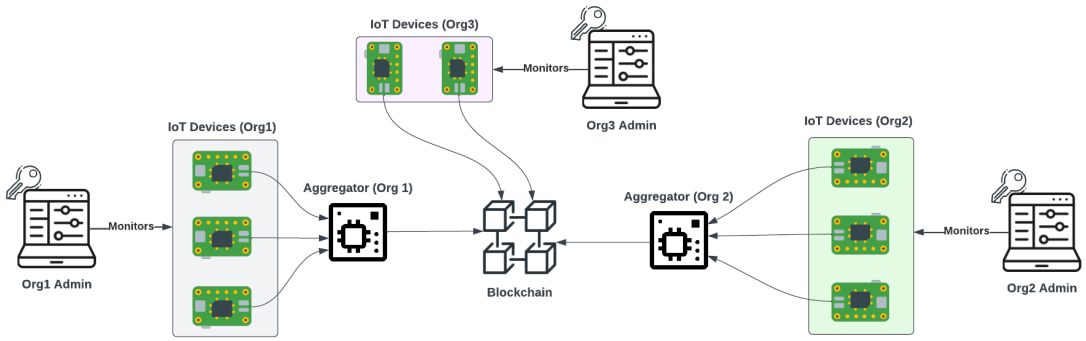

Figure 1: A Reference Scenario for MalRec

last full backup. Each organization has a different backup strategy. Some organizations might perform full backups, while others might prefer doing incremental or differential backups with a different rate.

2.2 Blockchain

Blockchain is a distributed ledger shared among a peer-to-peer network, in which members participate in its maintenance. The ledger stores as transactions all the activities and exchanges of resources that network participants make. Transactions are grouped into blocks containing a reference (i.e., the hash) of the previous one. In this way, a chain of blocks is generated, making them immutable. Indeed, the ledger is characterized by immutability: once a transaction is added, no one can tamper with it. Blocks are generated by block generator peers, that exploit public-key cryptography to broadcast them in the network. One of the main characteristics of blockchain is that it is decentralized: the activities of any individual node are coordinated and verified without the intervention of any central authority. This coordination is achieved through a procedure called *consensus*, according to which only the transactions that receive a sufficient number of approvals can update the ledger. There are different mechanisms for achieving consensus in a blockchain. For instance, some blockchains, like Bitcoin and Ethereum 1.0, adopt the *Proof of Work* (*PoW*) [9]. It consists of nodes (i.e., miners) in competition with each other to generate blocks. Each miner must solve a cryptographic puzzle, and the fastest receive a reward. Other blockchains, like Algorand[5] and Cardano[6], adopt the *Proof of Stake* (*PoS*)[9], in which nodes are randomly selected to generate blocks based on their stakes. In addition, other blockchains, like Hyperledger Fabric [2], rely on a *Byzantine Fault Tolerant* approach, which supports the ability of a network to reach consensus even if failures of some nodes occur. One of the most popular algorithms in blockchain for solving Byzantine problem is *Practical Byzantine Fault Tolerant* (*PBFT*) [13].

A blockchain can be classified as permissionless or permissioned. Permissionless blockchains are open for the public to join the network. On the other hand, permissioned ones are restricted to selected participants by the organizations that maintain the blockchain.

In this work, we use Hyperledger Fabric: a permissioned blockchain framework. It stands out from other blockchains due to its modular architecture, allowing the definition of pluggable components, such as the consensus protocol, the key management protocol, the ordering service, etc. This characteristic can be exploited to define a network suitable for different contexts. Further differences with other blockchains are the possibility to write smart contracts with general-purpose programming languages and the capability to group organizations in a consortium while having the possibility to keep the exchanged data private through channels.

We choose Hyperledger Fabric due to its elasticity, scalability, and flexibility which makes it suitable for several application domains, allowing the definition of customized smart contracts. It can process up to 3000 transactions per second and supports different general-purpose languages like Java, Golang, and JavaScript. Furthermore, it inherits the benefits of blockchain since it guarantees, in any case, the integrity of the data stored inside the blockchain thanks to the consensus procedure. Indeed, we rely on Fabric to store backup metadata as they simplify the retrieval process when devices need to download their backups. We introduce Hyperledger Fabric terminology in what follows (we refer the interested reader to [2] for more details).

Peers: A peer is a node that interacts with the blockchain by submitting a transaction, retrieving a transaction, and interacting with a smart contract. There are two types of peers: privileged and unprivileged. Privileged peers (e.g., organizations' admins) participate in consensus by running smart contracts and validating transactions. On the other hand, unprivileged peers (e.g., IoT devices) read and write the blockchain without running smart contracts or validating transactions.

Channel: A channel is the main communication mechanism that allows organizations within it to exchange messages with each other. There can be one or more channels inside the network in which different organizations may participate.

Blocks and **Transactions:** Blocks are a set of transactions. Transactions are issued by blockchain clients and they contain headers and payloads. Headers consist of cryptographic metadata (e.g., signatures) used to validate transactions. Payloads contain data that are used as input to the smart contracts that encode the application's business logic.

[5]https://www.algorand.com/
[6]https://cardano.org/

Ordering Service: It is an entity characterizing Hyperledger Fabric. It is composed of a set of orderers, which are nodes that collect all transactions and order them to be validated by all peers in the channel.

Smart Contracts: A smart contract (or chaincode in Hyperledger Fabric) is a program that defines the business logic of an application. In Hyperledger Fabric, it is programmed with general-purpose languages like Golang, JavaScript, and Java. They are triggered by submitting transactions and they are run by the privileged peers that agree on their output (i.e., consensus). The committee (i.e., the set of selected nodes by organizations) has copies of the same smart contracts that they run in an asynchronous way. Then, they agree on the execution's output (i.e., reaching consensus). Upon an agreement, the state of the blockchain changes (depending on what the smart contract was implemented to do). In this paper, we use the terms "smart contract" and "chaincode" interchangeably.

3 BASIC REQUIREMENTS FOR AN IOT BACKUP SYSTEM

In this paper, we propose a recovery solution that allows IoT devices to generate, store and retrieve backups efficiently and securely. The system design has been driven by several requirements, described in what follows.

Backup Policies Enforcement. Different organizations might have different backup strategies that they want to enforce, so MalRec needs to be flexible and modular in the sense that it should allow organizations to encode different strategies which satisfy their needs. For example, companies might want to adopt the 3-2-1 backup policy implying that there must be three backups in two different storage mediums and one of them must be offsite.

Compliance with Storage Security Standards. International secure data storage standards, such as NIST SP 800-171 [10] and ISO/IEC 27040, or privacy laws, such as GDPR[7], focus on multiple properties to ensure secure backup storage, namely authentication and authorization, confidentiality, accountability and audit, integrity, incident response, and availability.

- **Authentication and Authorization.** Backup generation and uploading shall be authenticated operations where the entity that interacted with the system can be identified. In addition, the backup system shall make sure that the entity that contacted it is authorized to retrieve the backups.
- **Confidentiality.** The backups shall be encrypted with the public key of the entity that generated them. This requirement allows the encrypted backups to be shared anywhere (even in P2P file systems like IPFS) without compromising their confidentiality.
- **Accountability and Audit.** All entities' actions shall be preserved and no one shall be able to manipulate them. So, the backup system shall keep track of all actions by all entities where these actions are immutable.
- **Integrity.** The backup system shall preserve the integrity of the backups to avoid retrieving compromised backups that might allow remote attackers to control devices.
- **Incident Response.** The backup system shall allow the devices to return to a safe state even in the case of malware attacks where malware could be backed up as a part of a normal backup procedure.
- **Availability.** The backup system shall allow the retrieval of backups and their metadata at any given moment in time.

[7]https://eur-lex.europa.eu/eli/reg/2016/679/oj

4 SYSTEM ARCHITECTURE

In this section, we discuss MalRec's workflow consisting of three main steps: backup upload, backup invalidation, and backup retrieval. These steps are discussed in Sections 4.1, 4.2, and 4.3, respectively.

Our proposed solution exploits blockchain to i) enforce the backup policies through smart contracts, and ii) comply with storage security standards that require integrity, authentication and authorization, incident response, and accountability, as discussed in Section 3.

The system is composed of several organizations with different IoT devices that can be deployed with different computing paradigms (e.g., fog or edge) as shown in Figure 1. Each organization has an administrator peer that plays a fundamental role in key management, as we assume it owns the master key (see Section 5). The organization's admin generates a key pair for all peers belonging to its organization, which they will use to encrypt and decrypt their backups. However, if a device undergoes a malware attack, it may not access the private key for decryption operations because it might be encrypted by ransomware, for example. For this reason, admins monitor the state of the peers of their organizations, aiming to identify attacked devices. If needed, an admin can generate a new key pair, acquire all backups, encrypt them with the new public key, and then send them to the recovered device.

4.1 Backup Upload

In this section, we explain the backups' upload process, which is illustrated in Figure 2.

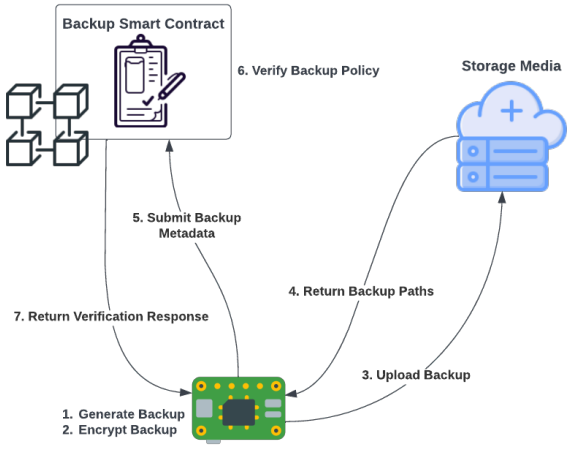

Figure 2: Backups' Upload

The process starts when a device generates the backup locally (step 1). MalRec adopts an initial full backup, followed by periodically incremental backups. Backups are logically stored in a linked list, in which each backup keeps a reference to the previous one

(See Definition 4.1). This data structure allows a device to check the correct sequence of the received backups. If a backup is marked as invalid because it contains signs of malware, the system must change the pointer of the involved backups to prevent the linked list from storing the tampered backup (See Section 4.2 for more details). Then, the device encrypts the backup with its public key (step 2). After that, it uploads the backup to the storage media (e.g., IPFS, Cloud, Local Server, etc.) (step 3), where the latter returns the backup's path (step 4). At this point, the device submits a backup metadata transaction, formally defined in Definition 4.1, to the blockchain. It is worth noting that: i) backups are signed to ensure integrity, so the backup metadata includes the checksum and signature; and ii) backup metadata includes the last submitted backupID that serves as a reference to keep all the backups of a device as a linked list for batch retrieval (See Section 4.3 for more details).

Definition 4.1. **Backup Metadata Transaction.** A backup metadata Transaction $backup_{md}$ is a tuple $\langle bID, deviceID, paths, checksum, sig, pbID, v, s, t\rangle$, where bID, $deviceID$, $paths$, $checksum$, sig, $pbID$, v, s and t are the unique backup identifier, the unique device identifier, the backup paths in storage media, backup's checksum, backup's signature, previous backupID, backup's validity (true by default), the backup's size, and the timestamp of the backup submission, respectively.

The blockchain hosts the *Backup Smart Contract*, shown in Algorithm 1, that checks if the previous backup is valid (lines 6-8) and if the backup's metadata satisfies the organization's backup policies (lines 9-20) (See Section 4.2 for more details) (step 6). To formalize backup policies, MalRec leverages a backup policy template bp_t, that can be used by organizations to specify their backup policies based on the number of replicas, frequency of the backup, storage location, size, etc. A backup policies template is formally defined in Definition 4.2. Finally, the *Backup Smart Contract* returns the verification response (step 7).

Definition 4.2. **Backup Policies Template.** A backup policies template bp_t is a tuple: $\langle r, f, o, s\rangle$, where r, f, o, and s are the total number of replicas (onsite and offsite), backup frequency, number of replicas stored offsite, and the backup size limit, respectively.

Example 4.3. Let us assume that an organization wants to adopt the 3-2-1 backup policy with a daily frequency and unlimited backup size. This policy will be modelled as follows: $bp_t = \langle 3, 1, 1, \infty\rangle$.

4.2 Backup Invalidation

The backup invalidation process starts when IoT devices suffer from malware attacks. In these critical situations, the backups could satisfy the backup policy in place, but they might contain traces of the attacking malware. This is managed by the *Malware Smart Contract* that is triggered by a malware transaction, defined in Definition 4.4. The latter is submitted by the organization's admin. The organization's admin monitors the devices that belong to his/her organization. So, when he/she gets an alarm of detected malware, it extracts the detection timestamp and estimates the start timestamp (e.g., by getting the earliest appearance of the malware in the network or specifying a threshold for the attack duration). Then, the *Malware Smart Contract* gets all the backups that fall within the estimated start and end of a malware attack. Then, it invalidates each one of them. Due to lack of space and the simplicity of the actions contained in the *Malware Smart Contract*, we do not present the pseudocode (we refer the interested readers to the open source implementation of MalRec discussed in Section 5).

Algorithm 1 Backup Smart Contract

Input: Backup Metadata Transaction $backup_{md}$
1: $policy \leftarrow getOrganizationPolicy(backup_{md}.deviceID)$
2: $offsite \leftarrow getOffsiteBackupsLength(backup_{md}.paths)$
3: $onsite \leftarrow getOnsiteBackupsLength(backup_{md}.paths)$
4: $latestBackup \leftarrow getLatestBackup(backup_{md}.deviceID)$
5: $backup_{md}.v \leftarrow false$
6: **if** $latestBackup.v == false$ **then**
7: $\quad rejectBackup(backup)$
8: **end if**
9: **if** $offsite + onsite < policy.r$ **then**
10: $\quad rejectBackup(backup)$
11: **end if**
12: **if** $offsite < policy.o$ **then**
13: $\quad rejectBackup(backup)$
14: **end if**
15: **if** $latestBackup.t - backup_{md}.t\ != policy.f$ **then**
16: $\quad rejectBackup(backup)$
17: **end if**
18: **if** $backup_{md}.s > policy.s$ **then**
19: $\quad rejectBackup(backup)$
20: **end if**
21: $backup_{md}.v \leftarrow true$
22: $acceptBackup(backup)$

Definition 4.4. **Malware Transaction.** A malware transaction m_{tx} is a tuple: $\langle deviceID, a_{start}, a_{end}\rangle$, where $deviceID$, a_{start}, and a_{end} are the infected $deviceID$, the estimated timestamps when the attack started and ended, respectively.

The backups are logically organized in a linked list since each backup metadata contains a reference to the previous backup. So, the *Malware Smart Contract* marks the infected device's backups in the infection timeframe as invalid. Thus, when a new backup metadata transaction is submitted, the *Backup Smart Contract* checks if the backup reference is valid. This check helps in retrieving only the valid backups to avoid further infection by the malware traces (See Section 4.3 for more details).

4.3 Backup Retrieval

The backups retrieval process (cfr. Figure 3) starts by getting the desired backup metadata by the *BackupID*. Then, the *Backup Smart Contract* checks if the querying device is authorized to retrieve that specific backup (step 2). The interactions with the blockchain in MalRec are authenticated through certificate authorities. So, the entities that submit transactions to trigger a smart contract are verified against the authorization rules that the organization's admin specified. This step is an additional security layer since the backups are already encrypted. After that, the *Backup Smart Contract* returns the requested backup metadata if the check is successful (step 3). After getting the backup metadata, the device extracts the backup path and queries the storage media (step 4). The latter returns the backup file (step 5). Finally, the device decrypts its backup (step 6).

It is worth noting that in step 1, the device may want to retrieve a single backup or a list of backups within some timeframe. So, in MalRec, there are 3 possibilities: i) query with *BackupID* that

returns a single backup that satisfies the backup identifier, ii) query with *DeviceID* that returns all the backups submitted by a specific device, and iii) query with *DeviceID* and a timeframe (start and end timestamps) that returns a list of backups submitted by the specified device within the specified timeframe.

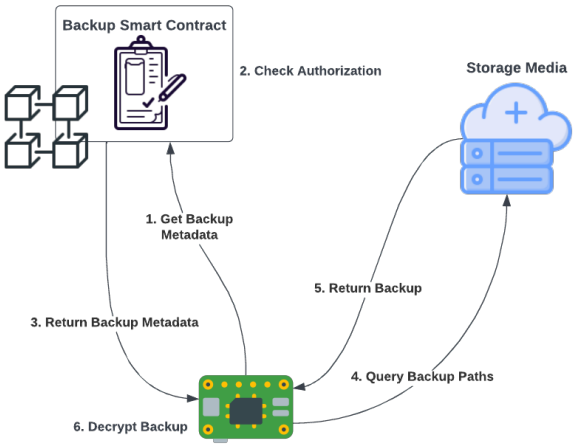

Figure 3: Backups' Retrieval

Once the device receives the requested backups' metadata from blockchain, it extracts each backup's paths to download a replica from the storage media. Once downloaded, the device can start a validation procedure to verify the file integrity by calculating the checksum (i.e., hash) and comparing it with the checksum used to download the data. If there is a match, the data is reliable and can be used. Finally, the device can begin the recovery procedure with the downloaded backups.

The described backup retrieval process is for a normal operation of the system where the backups are safe. However, when the backups of an infected device contain malware traces, additional measures should be taken. The procedure for recovering an infected device is shown in Figure 4. In order to recover an infected IoT device, the admin gets and decrypts the backup (step 2). Then, it generates a new key pair (step 3) that is used to encrypt the decrypted backup (step 4). After that, it sends the new key pair via a secure channel to the device (step 5). Finally, it uploads the backup to the storage media (step 6), gets its path (step 7), and submits the backup metadata to the blockchain (step 8). The *Backup Smart Contract* checks the validity of the latest backup (Algorithm 1 - lines 6-8) and the compliance of the backup to the adopted backup policy (lines 9-20) (step 9). If any of the checks fail, the backup is rejected. Then, it returns the verification response (step 10). After that, the device can query the *Backup Smart Contract* with its *deviceID* to get its backups similarly to the normal procedure described in Section 4.3.

4.4 Discussion

In this section, we discuss how MalRec satisfies the requirements presented in Section 3.

Backup Policies Enforcement. MalRec is designed to adopt different backup policies that companies might want to enforce.

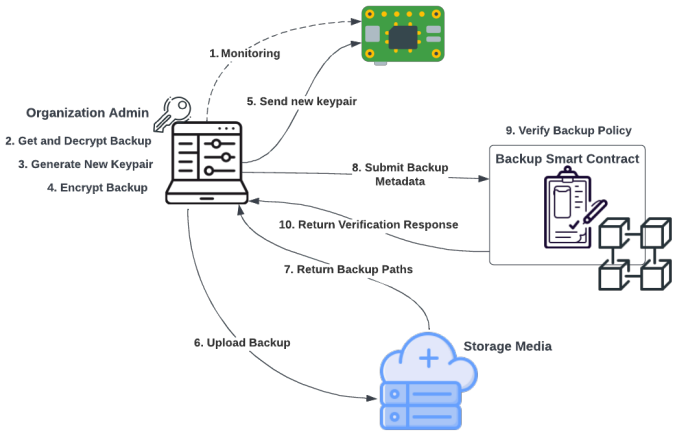

Figure 4: Recovering an infected device in MalRec

MalRec provides blockchain-based backup policies enforcement by formalizing backup metadata and enforcing their validity through the *Backup Smart Contract*.

Compliance with Storage Security Standards.

- **Authentication and Authorization.** MalRec submits backups metadata on blockchain (precisely Hyperledger Fabric). The latter checks the transactions' authentication through certificates. In addition, it enforces authorization through secure channels, so only authorized entities can join. For example, in a collaborative setting, each organization can have its own communication channel where the organization's admin specifies the authorization rules.
- **Confidentiality.** MalRec backup uploading procedure (cfr. Section 4.1) requires the backups to be encrypted with the owner's public key, so the encrypted backups can be shared anywhere (even in P2P file systems like IPFS) without compromising their confidentiality.
- **Accountability and Audit.** Since MalRec leverages blockchain for backup uploading and retrieval, all actions of users that interact with the blockchain are preserved and no one can manipulate them since they are immutable. So, MalRec provides all the necessary information for auditing and ensuring accountability.
- **Integrity.** MalRec ensures that the metadata (and hence the backups) are not modified through 1) checksums and 2) immutability of the blockchain.
- **Incident Response.** MalRec provides the utilities to react in case of an incident (e.g., malware attack). If malware was backed up, MalRec provides utilities through *Malware Smart Contract* to invalidate backups as discussed in Section 4.2.
- **Availability.** MalRec ensures that backups' metadata are available and immutable at all times. Companies might want to store backups in different locations to comply with certain regulations, so the availability of the backups is dependent on the storage medium. However, MalRec is agnostic of the storage medium, so it can be used with the storage medium that companies find suitable for their use case.

5 IMPLEMENTATION

We simulate IoT devices as Docker[8] containers. Since IoT devices usually have limited resources, we create constrained Docker containers in terms of computational resources to simulate real situations. We rely on IPFS[9], a distributed file system, as our storage medium. We choose IPFS to show that MalRec can be shipped as a decentralized solution. Thus, the reference implementation of MalRec is based on IPFS. It is worth noting that IPFS preserves the integrity by design since the identifier of an uploaded backup is its checksum, so any tempering with the stored backup would be identified. We use Golang to implement MalRec smart contracts: the backup and the malware smart contract. The first provides endpoints to add backups and retrieve them, while the second provides an endpoint to report malware that will automatically invalidate backups that have malware traces.

We use CouchDB as the state database since it provides indexes that increase the performance of queries. So, a device can query efficiently all of its backups, or the ones within a time frame. The source code of the application is open source and is available publicly on GitHub.[10]

The design of our solution is modular, so the process of monitoring and defining the files and folders to be backed up can be chosen by the user. Our solution monitors a list of folders and a list of file extensions (e.g., .TXT), so when there is a change, it compresses the changed and new files into one compressed folder (i.e., in .TAR.GZ format).

As discussed in Section 4.1, once a device generates a backup, it encrypts it with the public key before uploading it to a storage medium (i.e., IPFS for our reference implementation). However, the need for a mechanism to keep the asymmetric keys secure from being tampered with by malware arises because if a device cannot access its private key, it cannot decrypt its backups as shown in Section 4.2. For this reason, we support a hierarchical key schema for generating asymmetric keys. According to this procedure, each element in a hierarchy computes the keys for all of those located at an underlying level [11]. In our solution, we have one admin peer for each organization that owns a master key and monitors its devices. This peer exploits the master key to generate a new key pair for each underlying peer.

In our solution, we use BIP32[11], a standard used in Bitcoin and Ethereum to generate **Hierarchical Deterministic Wallets** (HD Wallets) (i.e., hierarchical keys). Whenever a device needs to retrieve backups, at first, it has to acquire its metadata by querying blockchain, as discussed in Section 4.3, and then it can download them from IPFS using the backup path.

The three types of queries that a peer can execute are implemented in three different endpoints. First, to get a single backup, a peer must invoke the chaincode with the *QueryBackup* function. The latter requires a *backupID* as a parameter, which is used to check if the related metadata exists in the ledger. If so, they are returned to the peer. Second, to get all its backups' metadata, it invokes *QueryBackupsByDeviceID* function by sending its *deviceID* as a parameter. Finally, to download backups submitted during a time frame, it invokes the *QueryBackupsByTimestamps* function, which receives as parameters the start timestamp, end timestamp, and the *deviceID*. It is worth noting that both *QueryBackupsByDeviceID* and *QueryBackupsByTimestamps* exploit the indexes feature of CouchDB.

[8]https://www.docker.com
[9]https://ipfs.io/
[10]https://github.com/lekssays/malrec
[11]https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

6 EXPERIMENTAL RESULTS

We adopt Hyperledger Caliper [12] as blockchain performance benchmark framework to evaluate the performance of our system. More precisely, we are interested in throughput and latency. The tests were executed on a computer running Ubuntu 20.04 LTS, with Intel® Core™ i7-9700K CPU, 16GB RAM, and 500 GB SSD. We simulate each peer in a constrained Docker container with 512MB of RAM. Finally, we support a block size of a maximum of 10MB, which can contain at most 500 transactions. In our tests, we do not consider the time required to download a backup from IPFS since it depends on external factors, such as the internet connection speed, the backup size, etc. Therefore, our tests start to monitor when a transaction is submitted and finish when the device receives backup metadata from the blockchain. We test over a network of three organizations with three peers for each one, including one admin peer that participates in consensus.

We perform three different tests, one for each type of query, described in what follows.

- **queryBackup**: to get a specific backup by sending its backupID;
- **queryDeviceID**: to get all the backups owned by a specific device;
- **queryTimestamp**: to get all backups within the time frame, defined by a start and an end timestamp, and belonging to a specific device.

Furthermore, to evaluate how our solution behaves in heavy loads, we execute each test by setting Caliper workers to send an increasing number of transactions: starting from 10,000, we reach 50,000, until 100,000 transactions.

The *throughput* results are shown in Figure 5. Throughput is evaluated in Transaction Per Seconds (TPS) versus the number of transactions per second. We observe that the throughput is decreasing according to the increasing number of transactions since the throughput of *queryBackup* is 897.7 TPS with 10,000 transactions, 870.1 TPS with 50,000, and 859.2 TPS with 100,000 transactions. For *queryDeviceID*, the maximum throughput is 301.7 TPS; then, it decreases to 279.9 TPS and finally to 276.0 TPS. We get the same trend with *queryTimestamp* where the throughput starts from 528.1 TPS, then 519.9 TPS, and 500.0 TPS, at last.

We can observe a difference between the throughput of *queryBackup* and the other two tests. This difference is caused by the type of access to CouchDB because a backup can be retrieved directly with the *backupID* in *queryBackup*. For *queryDeviceID* and *queryTimestamps*, further processing is required because they are range queries. We also observe a difference between the throughput of *queryTimestamps* and *queryDeviceID*. Instead, both queries are optimized as the indexes are stored in a B+ tree. This is the data structure used by CouchDB to store data, in which each leaf node can store the reference to one or more documents. The time

[12]https://hyperledger.github.io/caliper/

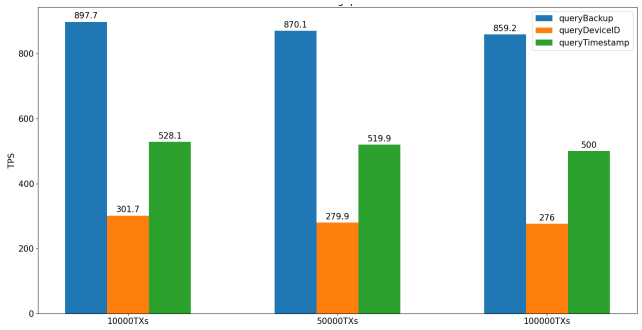

Figure 5: Throughput by varying the number of transactions

complexity of the search operation in a B+ tree is $\mathbf{O}(\log_2 \mathbf{t} \log_{\mathbf{t}} \mathbf{n})$ for any indexed query, where $\mathbf{t}$ is the maximum number of keys per node, and $\mathbf{n}$ is the number of keys stored in the tree. The difference between the two queries is due to the dimension of the traversed space inside the B+ tree during the search operation. For *queryTimestamps*, at first, the search space is reduced at the range of the start and end timestamps, and then the check of *deviceID* is performed in the smaller range of nodes. Since the entries are sorted by timestamp, the time complexity of retrieving them with a binary search is $\mathbf{O}(\log \mathbf{n})$ where n is the total number of entries. On the other hand, for *queryDeviceID*, the search space is larger. The time complexity of retrieving all the entries is $\mathbf{O}(\mathbf{n})$ where n is the total number of entries. Hence, the I/O operations to the disk are higher. The obtained throughput shows that MalRec can be deployed in a real-life IoT environment since our solution can tolerate a high load.

The *latency* is given by the time elapsed between the moment in which the transaction is submitted, and the moment in which the transaction is included in the blockchain. It can be considered a delay and is expressed in seconds. The average latency of our solution, considering all the types of queries for each number of transactions, is **0.02** seconds. Since backups are not generated in short time windows (e.g., 1 second), a latency of **0.02** fits our use case even with extreme cases where transactions are submitted concurrently in a time window of less than 1 millisecond. Thus, MalRec provides low latency which allows it to be used in large IoT networks.

7 CONCLUSION

IoT devices are heavily prone to malware attacks and are usually victims of botnets due to their lack of security measures. The literature focuses on detecting malware, but less attention is given to recovery solutions. In addition, with the development of data processing regulations in different countries, a need for transparent recovery systems that can help organizations present their due diligence arises. This paper proposes a blockchain-based backup policy enforcement framework for IoT where an organization can formalize its backup policies and ensure their enforcement. We have run extensive tests to show that our solution can be deployed in a real-life IoT environment, despite the limited resources of IoT devices. MalRec ensures the availability of the metadata and can be extended to ensure the availability of the backups data in a scalable way. Thus, there is a need for faster verification of data availability without the need to download the entire data or compromise the data's privacy. Different solutions have been proposed to address this problem such as relying on fraud proofs [1], zero-knowledge proofs, or polynomial commitments [7] on data's erasure coding [3]. The feasibility and adoption of such schemes in the context of MalRec remain a future work.

ACKNOWLEDGMENTS

This work has received funding from the Marie Sklodowska-Curie Innovative Training Network Real-time Analytics for Internet of Sports (RAIS) supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 813162. Additionally, it has been partially supported by CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927. The content of this paper reflects only the authors' view and the Agency and the Commission are not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. 2018. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044* (2018).
[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*. 1–15.
[3] SB Balaji, M Nikhil Krishnan, Myna Vajha, Vinayak Ramkumar, Birenjith Sasidharan, and P Vijay Kumar. 2018. Erasure coding for distributed storage: An overview. *Science China Information Sciences* 61, 10 (2018), 1–45.
[4] Larry Dignan. [n. d.]. *IoT devices to generate 79.4ZB of data in 2025, says IDC*. https://www.zdnet.com/article/iot-devices-to-generate-79-4zb-of-data-in-2025-says-idc/
[5] Mario Dudjak, Ivica Lukić, and Mirko Köhler. 2017. Survey of database backup management. In *27th International Scientific and Professional Conference Organization and Maintenance Technology*.
[6] Jitendcr Grover and Rama Murthy Garimella. 2018. Reliable and fault-tolerant IoT-edge architecture. In *2018 IEEE sensors*. IEEE, 1–4.
[7] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Polynomial commitments. *Tech. Rep* (2010).
[8] Mykhaylo Lobur, Serhiy Shcherbovskykh, and Tetyana Stefanovych. 2020. Availability Audit of IoT System Data Reserved by 3-2-1 Backup Strategy based on Fault Tree and State Transition Diagram Analysis. In *2020 IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT)*, Vol. 1. IEEE, 343–346.
[9] Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. 2019. Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities. *IEEE Access* 7 (2019), 85727–85745.
[10] National Institute of Standards and Technology. 2020. *Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations*. Technical Report. U.S. Department of Commerce, Washington, D.C. https://doi.org/10.6028/NIST.SP.800-171r2
[11] Alfredo Santis, Anna Ferrara, and Barbara Masucci. 2007. Efficient Provably-Secure Hierarchical Key Assignment Schemes. *Theoretical Computer Science - TCS* 412, 371–382.
[12] Min Woo Woo, JongWhi Lee, and KeeHyun Park. 2018. A reliable IoT system for personal healthcare devices. *Future Generation Computer Systems* 78 (2018), 626–640.
[13] Xiandong Zheng and Wenlong Feng. 2021. Research on Practical Byzantine Fault Tolerant Consensus Algorithm Based on Blockchain. *Journal of Physics: Conference Series* 1802, 3 (03 2021), 032022. https://doi.org/10.1088/1742-6596/1802/3/032022