Data Certification Strategies for Blockchain-based Traceability Systems

Giacomo Zonneveld, Giulia Rafaiani, Massimo Battaglioni and Marco Baldi
Dipartimento di Ingegneria dell'Informazione
Università Politecnica delle Marche
Ancona, Italy
Email: {g.zonneveld, g.rafaiani, m.battaglioni, m.baldi}@univpm.it

2025 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) | 979-8-3315-4135-4/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICBC64466.2025.11114484

***Abstract*—The use of blockchains for data certification and traceability is now well established in both the literature and practical applications. However, while blockchain-based certification of individual data is clear and straightforward, the use of blockchain to certify large amounts of data produced on a nearly continuous basis still poses some challenges. In such a case, in fact, it is first necessary to collect the data in an off-chain buffer, and then to organize it, e.g., via Merkle trees, in order to keep the size and quantity of certification data to be written to the blockchain small. In this paper, we consider a typical system for blockchain-based traceability of a production process, and propose and comparatively analyze some strategies for certifying the data of such a process on blockchain, while maintaining the possibility of verifying their certification in a decentralized way.**

***Index Terms*—blockchain, data certification, Merkle tree, traceability.**

I. Introduction

Blockchain technology, introduced in 2009 to support the first cryptocurrency [1], and then generalized to the concept of distributed ledger technology (DLT), now lends itself to many applications other than just monetary transactions. These include data notarization and certification applications, in which the features of the blockchain are exploited to make it play the role of a notary and prove the existence of a certain amount of data at a certain instant in time (within some temporal resolution). Typical applications of this use of blockchain are those related to agri-food supply chain traceability [2], [3] and production processes traceability in general, as well as traceability and data certification for Internet of Things [4] and Industry 4.0 [5] applications. Another example of traceability for which blockchain-based solutions are suitable is that of Proof of Attendance, in which the aim is to certify users' participation in various kinds of events, such as cultural events, tourism initiatives, or their attendance to class or work [6], [7]. In such a case, instead of production or supply chain data, the data to be certified will be attendance certificates or similar data. It should be noted that, in all these cases, for reasons of confidentiality as well as data size, it is not possible to write the data itself onto the blockchain, but only certification data derived from it, from which the data itself cannot be reconstructed.

In fact, blockchain technology provides wide support for data certification. Thanks to its properties such as immutability, once data are recorded into the blockchain, they become unalterable, which ensures data integrity over time. The transparent and distributed nature of blockchain ensures trust, since anyone can directly verify the information contained in the public ledger. Given its nature, however, blockchain technology cannot be used as a database. In fact, blockchain is not meant for storing big amounts of data, even because this would be inefficient and very expensive. Moreover, the public nature of the blockchain negates the possibility to use it for storing personal or sensitive information. For this reason, blockchain is usually adopted to notarize the existence of a document and preserve its integrity without revealing its content. The corresponding data are then securely stored in an off-chain repository or in distributed file systems such as IPFS (InterPlanetary File System).

Data certification is essential to ensure authenticity and integrity of information, increasing trust, and reducing the risks of fraud or manipulation. Moreover, it ensures data integrity, making any changes to the original content easily detectable. However, when data are stored outside the blockchain and need to be certified in the blockchain, the problem of keeping the certification synchronized with the data themselves arises, especially when an almost continuous flow of data needs to be certified, as in manufacturing process traceability applications. In addition, the cost of certification must be taken into account in applications of this kind, because every transaction made on blockchain has a cost, and making a very large number of transactions could result in costs that are too high for this kind of applications. It therefore becomes necessary to design a system that interposes itself between the data source and the blockchain, and allows for an efficient and cost-effective data certification process, without sacrificing the decentralization and certification granularity features that blockchain can offer.

A. Our contribution

In this paper, we present a blockchain-based data certification model and propose and compare some possible strategies for generating certification data at regular intervals.

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, by project FOLOU funded by the European Union's Horizon Europe research and innovation program (GA: 101084106), and by MIMIT, under FSC project "Pesaro CTE SQUARE", CUP D74J22000930008.

We analyze and evaluate different approaches in order to identify the most efficient strategy. Specifically, we examine two approaches: one where a blockchain transaction is created for each piece of data to include its hash, and another that leverages Merkle trees to organize document hashes, thereby reducing the number of transactions required. We also propose a cost analysis that is useful to design the most suitable system for data certification according to the final user requirements. For the second mentioned approach, we explore the implementation of Merkle tree generation and Merkle proof extraction processes, addressing practical challenges such as handling transaction refusals during the upload of new data for certification. Transaction failures, commonly arising from fluctuating gas prices and network congestion, can significantly increase costs and introduce inefficiencies, with recent analyses indicating relatively high failure rates during periods of high congestion[1].

Finally, we explore a use case focused on certifying data related to event attendance and attraction visits. Finally, we present numerical results, focusing on the average execution time of the processes described above.

B. Related works

The characteristics of blockchain technology, such as immutability, integrity and time-stamping, well meet the requirements of a traceability system. For this reason, many different approaches for blockchain-based traceability are proposed in the literature. For example, several works are focused on the use of blockchain technology for food traceability [8]–[10] and for luxury items supply chain management [11]. Moreover, different approaches for blockchain-based academic certificate management have been proposed [12], [13], including some general platforms for data certification [14]. The authors in [15] explored how to integrate blockchain with a distributed file system, aiming to provide a system capable of combining the security of the former with the efficiency of the latter. The system proposed in [16] relies on an off-chain database for data storage and on a permissioned blockchain for data certification of manufacturing data. Although the blockchain properties are maintained, users requesting a certification verification still have to trust the nodes of the network. Regarding implementation aspects, in particular Merkle tree traversal, relevant alternatives are proposed in [17]–[19].

In this paper, we consider a general approach for data traceability using blockchain. In fact, the system we consider can be efficiently applied to different amount and different kinds of data to be certified, as well as in several applications. The approaches in the literature are either specifically designed for an application or, if more general, are often complex to use or proprietary. Moreover, the system we consider uses a public blockchain, while most blockchain-based traceability systems in the literature use private or permissioned blockchains.

[1]See transaction failure rate analysis at: https://dune.com/queries/2839305/4741578

C. Paper outline

The paper is organized as follows. In Section II, we outline the system model and evaluate the performance of two data certification approaches in terms of generation, storage, transactions, and verification cost. In Section III, we focus on the Merkle tree-based approach and discuss implementation aspects, also considering a use case concerning Proof of Attendance. Section IV contains numerical results, and Section V concludes the paper.

II. System model and certification approaches

We consider a system for blockchain-based data certification modeled as in Fig. 1. Its main components are:

- One data source, typically consisting of some production process generating traceability data in an essentially continuous manner.
- An off-chain data collection and processing facility, which stores the data produced in off-chain storage and processes them to obtain traceability data that are notarized on the blockchain at regular intervals.
- A public blockchain, like Ethereum, receiving transactions including certification data that are generated by the off-chain data collection and processing facility at regular intervals.
- A web app allowing users to verify the certification of some selected data, for example related to a product they purchased, directly querying the data source and the blockchain, without intermediaries.

We remark that this approach can be used for different types of data to be certified (e.g., supply chain, proof of attendance, etc.).

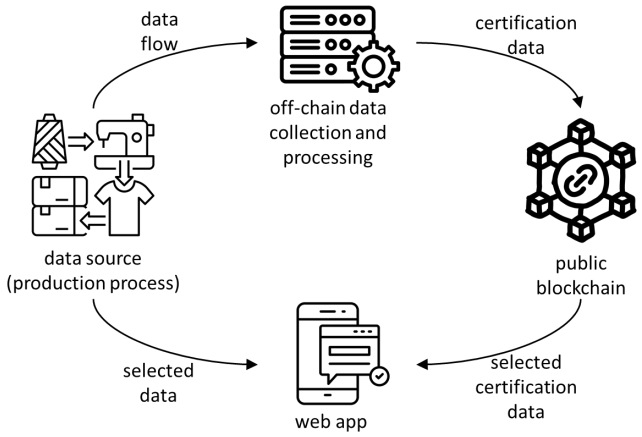

Fig. 1: Blockchain-based data certification system model.

In this paper we delve into one component of such a system, namely that related to the best strategy for generating certification data at regular intervals. For this purpose, let us consider two different approaches for data certification. In the first approach, every transaction contains the information to be certified, while in the second approach we aggregate more data in a single transaction, through the use of Merkle trees.

A. First approach: Single data certification

The first of the two schemes we consider is the simplest and most common one. Basically, we send a blockchain transaction that contains the hash of each piece of data that needs to be certified. The hash of the single piece of data is included in the "data" field of the transaction, allowing the information to be immutably stored into the blockchain. Then, the certified data are enriched with some parameters related to the blockchain transaction, such as the transaction ID, that are needed to verify data integrity.

In fact, the verification phase constitutes a crucial aspect of a certification protocol. In this case, when a user wishes to verify a specific piece of data, the system takes that information together with transaction data as input, computes the hash digest, locates the transaction on-chain and compares the digest stored inside the data field of the transaction with the one computed locally. If the two values match, the certification is confirmed as valid, attesting data integrity. Conversely, if they do not match, the integrity verification fails, indicating a possible alteration of the information.

B. Second approach: Multiple data certification

In the second approach, Merkle trees are used as a solution for organizing and compress the document digests. The primary goal of this approach is to limit the number of blockchain transactions to be generated, making the certification process more efficient and cost-effective. Compared to the previous approach, which requires a separate transaction for each piece of data to be certified, using a Merkle tree allows multiple data to be certified in a single transaction, which simply contains the root of the Merkle tree, thus saving considerable cost and processing time. This approach requires to collect data coming from the source and organize them into Merkle trees, whose numerosity (i.e., the number of leaves) is a design choice that takes into account various factors, with the final aim of minimizing costs. Having done so, each certified information is provided with a Merkle proof, which is a string that represents all the information participating in the Merkle tree, other than the piece of data itself. The Merkle proof allows the integrity of the information to be verified from the information itself and the Merkle tree root. To avoid the unnecessary occupation of storage space due to storing the Merkle proof for every data leaf, we can store the entire Merkle tree structure and compute Merkle proofs on the fly when some data item needs to be associated with its Merkle proof. Saving the tree also allows the nodes that are part of the proof to be quickly identified, speeding up the local root calculation during the verification phase.

The verification phase, indeed, is more complex than for the previous approach. In fact, verification requires local computation of the Merkle root, starting with the document whose certification is to be verified and the corresponding proof. The proof is obtained after locating the correct tree, the position of the considered data, and all the nodes of the tree needed to re-calculate the root. When the user has the proof and the data needed to locate the blockchain transaction (which contains the original on-chain Merkle root), the actual verification function begins, following these steps:

1) Compute the hash digest of the data.
2) Extract the original Merkle root from the blockchain transaction associated to that piece of data.
3) Locally compute the root through the Merkle proof and the digest found in step 1).
4) Compare the values obtained in steps 2) and 3); if they match, the verification is successful, otherwise fails.

C. Comparison of the proposed approaches

In the above sections we have discussed two approaches for data certification: one based on individual transactions for each data entry and the other based on data organization in Merkle trees. Although both methods have the same purpose, there are significant differences in terms of cost and performance. The parameters considered for the performance evaluation are:

- *Generation cost*. Given N data items to be certified, the first approach requires to compute N hash digests and to include them in N transactions. In the second approach, instead, the Merkle tree needs to be built, meaning that the number S of digests to be computed is given by $S = (2N - 1)$. Therefore, the generation cost $C_{\text{generation}}$ in the first case is given by $C_{\text{generation}} = N \cdot C_{\text{hash}}$, where C_{hash} is the cost of computing one hash digest, while in the second case $C_{\text{generation}} = (2N - 1) \cdot C_{\text{hash}}$.
- *Storage cost*. In the first approach, no data needs to be stored. When a user wants to verify some information, they provide as input the information itself, containing the related transaction data. This information is then hashed, and the result is compared with the data contained in the blockchain transaction. As for the second approach, instead, we need to provide a Merkle proof (and possibly store the Merkle tree to improve performance). Therefore, the storage cost C_{storage} is given by $C_{\text{storage}} = S \cdot d$, where d is the average dimension of the leaf node.
- *Transactions cost*. It is straightforward to note that the first approach has higher monetary cost that the second approach. Indeed, in the first case, we send N transactions for N data items, while in the second case we only need one transaction for the overall N data items. Therefore, the transactions cost will be $C_{\text{transaction}} = N \cdot p$, where p is the average cost for sending a transaction in a public blockchain. The transaction cost of the second approach will instead be just $C_{\text{transaction}} = p$. We would like to underline that we consider a public blockchain because it provides higher resilience and decentralization with respect to a private or permissioned blockchain.
- *Verification cost.* The cost of verifying a piece of data previously certified is different for the two approaches. In the first case, the verification cost is simply the cost of computing one hash digest, that is, $C_{\text{verification}} = C_{\text{hash}}$. As for the second approach, the Merkle proof needs to be computed. The complexity of this operation is in the order of $O(\log_2(N))$; therefore, in this case, we assume $C_{\text{verification}} = \log_2(N) \cdot C_{\text{hash}}$.

Summarizing, the total cost is given by:

$$C_{\text{tot}} = C_{\text{generation}} + C_{\text{storage}} + C_{\text{transaction}} + C_{\text{verification}}.$$

For the first approach, i.e., the single data certification, the total cost is hence given by:

$$C_{\text{tot}} = N \cdot C_{\text{hash}} + N \cdot p + C_{\text{hash}}.$$

Instead, the total cost for the multiple data certification, requiring the Merkle tree calculation, would be:

$$C_{\text{tot}} = (2N-1) \cdot C_{\text{hash}} + S \cdot d + p + \log_2(N) \cdot C_{\text{hash}}.$$

Looking at these overall costs, we can conclude that there is not an a priori ideal configuration, but the most suitable approach should be identified on the basis of the specific application and its requirements. However, one aspect that must also be considered is that, in some applications, the monetary cost of transactions might be prioritized over the computational cost, and in that case the second approach is certainly preferable, as it reduces the number of transactions made on the blockchain by a factor of N. In particular, in traceability applications, we usually have an almost continuous flow of data needing to be certified. Therefore, the approach that uses a Merkle tree appears to be preferable, since it minimizes the transaction costs. For this reason, in the following sections, we provide an efficient implementation of the data certification approach based on Merkle trees. We also analyze the performance of the proposed approach in constructing a Merkle tree with N data entries to be certified, as well as its performance in verifying previously certified information.

III. Merkle Tree Generation and Merkle proof extraction

Let us consider a Merkle tree implementation based on recursion and nodes indexing to keep the structure simple, easily understandable, and cost efficient. In particular, we focus on the analysis of the two most common operations applied on such a structure, which are the Merkle tree generation and the Merkle proof extraction. By leveraging the use of indexing, we guarantee low traversal costs from the root of the tree to the interested leaf. Each node in a Merkle tree with N leaves and M nodes, denoted as n_i, $i \in \{1, \dots, M\}$, is defined by:

- idx_i such that $1 \leq \text{idx}_i \leq M$, the node index.
- v_i, the hash computed on the concatenation of the hashes of n_i children. If n_i is a leaf node, v_i is the hash of the data block.
- n_i^l, the left child (null if n_i is a leaf).
- n_i^r, the right child (null if n_i is a leaf).

A. Nodes indexing when N is a power of 2

The tree indexing when N is a power of 2 is performed using a bottom-up approach according to the following rules:

- The indexes of the leaves are defined as incrementing odd numbers, starting from 1 on the leftmost leaf.
- The index of each parent node p is computed as:

$$\text{idx}_p = \text{idx}_l + 2^h, \tag{1}$$

where idx_l represents the index of the direct left child and h corresponds to the height of the node children. For example, the index idx_p of the direct parent of each pair of leaves is computed as:

$$\text{idx}_p = \text{idx}_l + 2^h = \text{idx}_l + 2^0 = \text{idx}_l + 1.$$

An example of indexed Merkle tree is shown in Fig. 2, where, for the sake of conciseness, only the entry indexes are reported.

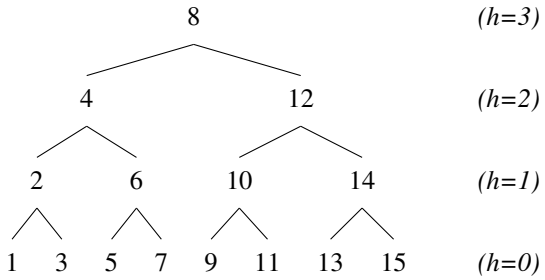

Fig. 2: Example of indexing of a Merkle Tree with 8 leaves.

If N is not a power of 2, a modification of this indexing procedure must be performed, as discussed next.

B. Merkle Tree generation procedure

Given a list of data items and a hash function H, hashing is applied to each element of the list to obtain the corresponding hashed data list. Each element of the hashed data list will be a leaf node of the Merkle tree. In general, for every pair of consecutive nodes (n_i, n_{i+1}) with hash values v_i and v_{i+1}, a parent node is obtained by computing the node hash value as $H(v_i||v_{i+1})$, where $||$ denotes concatenation, and the index through (1). Throughout the process, we keep two lists which are refreshed at every tree climb: a *values list* L_v and a *nodes list* L_n. L_n contains the information regarding the nodes created at the current layer. If we are not at the root layer, these nodes will be included as left and right children of their parent nodes. L_v, instead, contains the hash values v_i of the nodes created at the current layer. If we are not at root level, these hashes are used to compute the hash value of their parent nodes.

It can happen that the number of leaves is not a power of 2. This implies that at least a (non-root) layer contains an odd number of nodes. In our implementation, we need to bypass such a condition. So, before climbing the tree, if the length of L_n is odd and greater than 1, the last node of L_n is replicated, and its left and right children set as zero, as shown in Fig. 3.

When all the nodes of the current step have been processed, we move on to the next layer. Having L_v and L_n filled, the procedure iterates through such lists as follows. Let us denote as n_p^l and n_p^r the left and right child of a given parent node n_p, respectively. Then,

1) for every two consecutive elements of L_n, say n_i and n_{i+1}, it holds that $n_i = n_p^l$ and $n_{i+1} = n_p^r$ for a new parent node n_p;

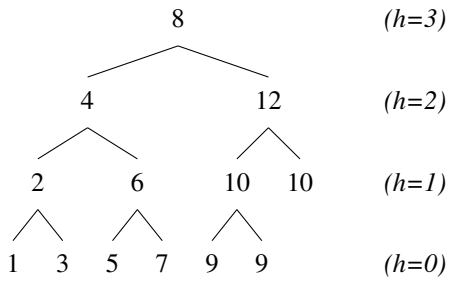

Fig. 3: Example of indexing of a Merkle Tree with 5 leaves.

2) n_p hash value is computed as $H(L_v[i] \,||\, L_v[i+1])$;
3) n_p index is computed with (1).

When the lists L_v and L_n have been completely explored, they are cleared. Such procedure goes on until L_v and L_n are populated with only one element, which is the Merkle root. The pseudo-code is shown in Algorithm 1, which includes the indexing procedure described in Section III-A as a special case. The Merkle tree generation has a time complexity in the order of $O(N)$.

C. Proof extraction

Having built a Merkle tree and knowing the index of the desired leaf, the proof can be extracted by traversing the tree from the root to the leaf following the nodes indexes as follows. We define as idx_i the index of the leaf and with idx_n the index of the node that is being traversed. Then,

- if $\mathrm{idx}_i > \mathrm{idx}_n$, store the hash value of the left child node and move on the right sub-tree;
- if $\mathrm{idx}_i < \mathrm{idx}_n$, store the hash value of the right child node and move on the left sub-tree;
- else, the leaf has been found.

Such an operation has a time complexity in the order of $O(h) = O(\log_2(N))$, where h represents the tree height and N represents the number of leaves. An example of proof extraction through nodes traversing with indexes is shown in Fig 4.

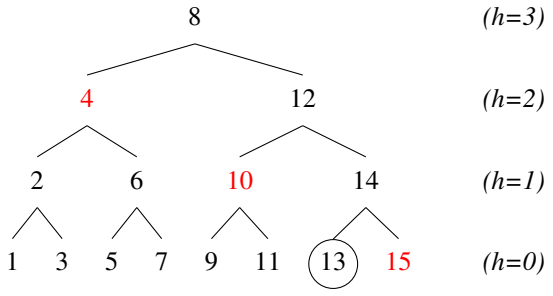

Fig. 4: Example of proof extraction of leaf with index 13. The indexes in red represent the nodes whose hash digests are collected to compose the Merkle proof.

Algorithm 1 Merkle Tree Construction

Input: List of data items $D = \{d_1, \ldots, d_n\}$, hash function H
Output: A properly constructed Merkle tree with root node

Initialize:
$L_v \leftarrow \emptyset$ ▷ Values list
$L_n \leftarrow \emptyset$ ▷ Nodes list
Hash input data:
for $i \leftarrow 1$ to n **do**
  $v_i \leftarrow H(d_i)$ ▷ Hash each data item
  Create leaf node n_i with index $2i-1$ and value v_i
  L_v.append(v_i)
  L_n.append(n_i)
end for
Build tree bottom-up:
$h \leftarrow 0$ ▷ Current height
while $|L_n| > 1$ **do** ▷ Continue until reaching the root
  $L_v^{new} \leftarrow \emptyset$ ▷ New values list for next level
  $L_n^{new} \leftarrow \emptyset$ ▷ New nodes list for next level
  if $|L_n|$ is odd **and** $|L_n| > 1$ **then**
    Duplicate last node in L_n
    Duplicate last value in L_v
  end if
  for $i \leftarrow 0$ to $|L_n| - 1$ **by** 2 **do**
    $n_i \leftarrow L_n[i]$ ▷ Left child
    $n_{i+1} \leftarrow L_n[i+1]$ ▷ Right child
    $v_i \leftarrow L_v[i]$ ▷ Left child hash value
    $v_{i+1} \leftarrow L_v[i+1]$ ▷ Right child hash value
    $v_p \leftarrow H(v_i \,\|\, v_{i+1})$ ▷ Compute parent hash value
    $idx_p \leftarrow n_i.index + 2^h$ ▷ Calculate parent index using (1)
    Create parent node n_p with index idx_p, valued v_p
    Set n_i as left child of n_p
    Set n_{i+1} as right child of n_p
    L_v^{new}.append(v_p)
    L_n^{new}.append(n_p)
  end for
  $L_v \leftarrow L_v^{new}$ ▷ Update lists for next level
  $L_n \leftarrow L_n^{new}$
  $h \leftarrow h + 1$
end while
return Indexed Merkle tree, $L_n[0]$

D. Multi-level Indexed Merkle Tree

Basically, the system considers that a general transaction workflow would be built over a waiting queue mechanism in which, for a defined time, clients can submit data to be certified. After a trigger is activated, which can be automatic (for example, time-based) or manual (certification request), the Merkle tree is built and saved off-chain, then the Merkle root is inserted in a data transaction which is submitted to the network. Using Ethereum, the client requesting the certification must define a gas price that they are willing to pay. If the gas price is not enough, the transaction gets refused by the network and the client must apply again for a

new certification request after having tuned the gas price. In the meantime, it could happen that new data to be certified have been collected, and such data are not included in the Merkle tree related to the refused transaction. The simplest solution would be to generate a new Merkle tree and a new transaction, but since the previous one has not yet been written onto the blockchain, a more efficient solution can be adopted. In particular, it is possible to incorporate certification of the new data into the previous transaction, which can be updated before being resubmitted.

We propose a solution that addresses this issue by building a Merkle tree that can be extended without requiring a rearrangement in leaves positioning. This way, even if the network rejects the data transaction several times, it is possible to add new data without having to build another Merkle tree or assign updated indexes to previously uploaded data. Although leaf rearrangement is a simple solution, it cannot be used when the leaf position of a data item needs to be known at the time of data storage, for example to extract the Merkle proof later without having to explore all the leaves. By adopting a multi-level approach, such a requirement is fulfilled. In fact, the predefined leaf index, assigned when generating the Merkle tree for a failing transaction, is still valid when generating a new multi-level indexed Merkle tree, as explained next.

We propose a multi-level indexed Merkle tree that consists of a combination of single indexed Merkle trees, built on different levels (see Fig. 5 for an example). Let us assume that the Merkle tree containing the highest layer is labeled as level-0 Merkle tree. We say that a Merkle tree lives in the l-th level if index representing its root is a vector with $l+1$ entries. The number of levels is predetermined and depends on the considered application. The last level is the one dealing with transaction failures and, in practice, the number of subtrees it will contain by the end of the certification process cannot be known.

For example, let us consider a 2-level indexed Merkle tree, containing m subtrees at level 1. The level-1 subtrees are indexed first, as follows. Let n_w be a node of the z-th subtree, $0 \leq z \leq m-1$, then the index idx_w of such node is $\mathrm{idx}_w = [2z+1, j]$, where j is found according to (1). Once a subtree is fully constructed, it is appended to the level-0 Merkle tree, with its root as a leaf node. Then, the indexing of the level-0 Merkle tree is again carried out following (1), with the right entry of the indexes of the leaf nodes being ignored during this step.

This reasoning easily generalizes to k-level Merkle trees. Moreover, with this approach, indexes are resistant to transaction failures. In fact, the leaf index assigned to the data already has a complete path traversal considering all the tree levels. Let us assume we adopt a 2-level tree where level 1 manages transaction failures and level 0 is used to obtain the final Merkle root. If there were $m-1$ transaction failures, the whole multi-level indexed tree will contain m subtrees at level-1. Since the subtrees are incrementally appended on an "as needed" basis, it is possible to obtain the definitive leaf index regardless of the final value of m.

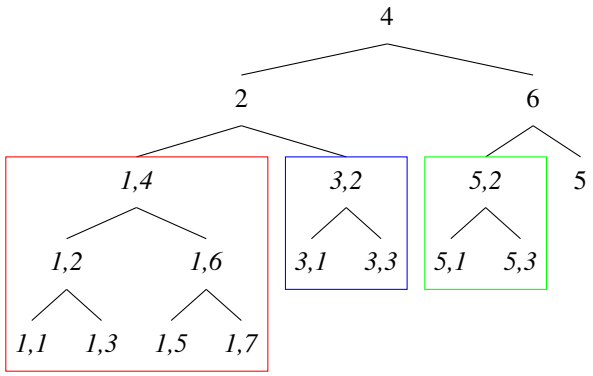

Fig. 5: Example of a multi-level Merkle tree obtained before attempting to forward the transaction for the third time. Each colored box represents a different indexed Merkle tree.

E. Use case: Proof of Attendance

As stated in the Introduction, an interesting application for traceability using blockchain is related to the Proof of Attendance, i.e., certifying users' participation in various kinds of events. For this reason, the infrastructure described in the previous sections has been applied to a use case regarding the blockchain-based certification of participation in cultural events and touristic initiatives. In both cases, when attending an event or visiting an attraction, participants are asked to prove their attendance by scanning some QR code. Since events have a limited time for attending, the certification on blockchain is done only after the event itself is ended by generating a single Merkle tree, where each leaf corresponds to a participant. For attractions, instead, since there is no time limit, the attractions manager can decide when to apply for certification. The problem related to the transaction failure, thoroughly described in the previous section, can occur in the latter case. Let us suppose that a set of attendants have registered their presence and are waiting for their attendance to be certified, the attractions manager applies for a transaction on blockchain, but the transaction fails. In the meantime, other attendants might have scanned the QR code. As a result, if no action is undertaken to modify the Merkle tree approach, it would be necessary to apply for two transactions and spend a double price: one for the failed transaction, which needs to be forwarded again, and one for the new one. Generally, an attractions manager might have multiple attractions with different QR codes. Attendants of each event would be managed through different Merkle trees. When paying for attendance certification, the attractions owner would need to pay a price

$$P = p \cdot \sum_i (1 + F_i)$$

with p being the price of a data transaction, i being an index for the attractions, and F_i being the number of transactions failed for the i-th attraction. By leveraging the multi-level indexed Merkle tree it is possible to minimize P by generating a single Merkle tree as follows:

- level 0 is the top-level Merkle tree, which contains the final Merkle root to be forwarded to the blockchain;
- level 1 contains the sub-trees a_i representing the attractions;
- level 2 contains the sub-trees $t_{i,j}$, representing failed transactions (with index j) for the i-th attraction.

As a result it holds that $P = p$, independent of the number of failures. In Fig. 6, an example of tree for the management of the attendance for two attractions is proposed. In particular, in the figure, the tree is obtained before attempting to forward the transaction for the second time for certificating attendance at two events. The sub-trees $t_{1,1}$ and $t_{2,1}$ contain attendances related to a previously failed transaction. If a_2 did not have any attendants when the failed transaction was requested, but some attendants registered before the second attempt, $t_{2,1}$ will contain such attendants and $t_{2,2}$ would only contain the Merkle Root of $t_{2,1}$ for balancing purposes.

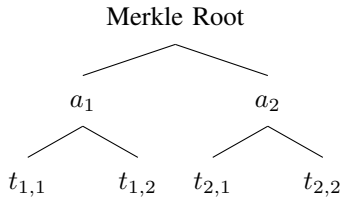

Fig. 6: Example of a multi-level Merkle tree used in the Proof of Attendance use case.

IV. Numerical results

To evaluate the performance of the proposed indexed Merkle tree, we compare our approach with two public libraries: *merkletreejs* [20] and *merkle-tools* [21]. In particular, we compare the average times required to execute the Merkle tree generation and the Merkle proof extraction. All tests were carried out on an Apple Macbook Pro 2021 with a Apple M1 Pro chip and 16 GB of RAM. First, to limit the variability caused by the random leaves data generation, we generate leaves data only once and then we start measuring performance on the aforementioned operations. We assess the time complexity in relation to the number of leaves and we do it by applying the same function for 250 000 iterations. In our tests, we consider the following leaf sizes: $\{10, 57, 100, 157, 200, \ldots, 900, 957, 1000, 1100, 1157, 1300, 1357, 1500, 1557, 1700, 1757, 2000\}$. On the results obtained for each size, we apply a 5% trimmed mean on measured times. For both operations tested, we report a figure representing the overall results as the number of leaves increases, and Table I, analyzing execution times, together with a percentage increase comparing the libraries with our approach..The results reported in Fig. 7 show that all the approaches follow a trend strictly related to the number of leaves. Compared to existing libraries, our approach requires less time to build a tree, and such a difference becomes more evident as the number of leaves increases. Our approach shows higher efficiency, with *merkletreejs* being slower by a quantity ranging between 19.32% and 28.86%, and *merkle-tools* having greater delays ranging between 24.68% and 47.55% (see Table I).

TABLE I: Merkle tree generation performance

	Average execution time (ms)		
N	merkletreejs [20]	merkle-tools [21]	our approach
100	0.1182 (+21,23%)	0.1222 (+25,35%)	0.0975
200	0.234 (+19,32%)	0.2445 (+24,68%)	0.1961
500	0.6071 (+22,25%)	0.6399 (+28,85%)	0.4966
1000	1.2471 (+24,44%)	1.4788 (+47,55%)	1.0022
1500	2.2192 (+20,3%)	2.489 (+34,93%)	1.8447
2000	3.1123 (+28,86%)	3.43 (+42,02%)	2.4152

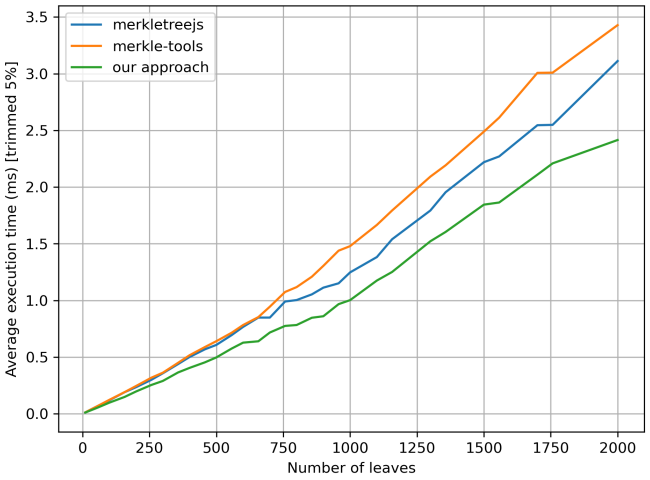

Fig. 7: Merkle tree generation performance

Fig. 8 shows the results of the Merkle proof extraction. Owing to the index-based exploration of the tree, it is possible to maintain a lower execution time, that scales rather slowly with the number of leaves. Differences with *merkletreejs* and *merkle-tools* are more noticeable with delays in time ranging from 177.78% to 200% for the first one and from 181.25% to 233.33% for the second one (see Table II).

TABLE II: Merkle proof extraction performance

	Average execution time (ms)		
N	merkletreejs [20]	merkle-tools [21]	our approach
100	0.00048 (+200%)	0.00045 (+181,25%)	0.00016
200	0.00079 (+192,6%)	0.0009 (+233,33%)	0.00027
500	0.00087 (+190%)	0.00096 (+220%)	0.0003
1000	0.00094 (+184,85%)	0.0011 (+233,33%)	0.00033
1500	0.00102 (+183,33%)	0.00117 (+225%)	0.00036
2000	0.001 (+177,78%)	0.0012 (+233,33%)	0.00036

These results show that, from a data certification perspective, where an agile Merkle proof extraction needs to be performed, the use of the indexed Merkle tree can significantly reduce the execution time.

V. Conclusion

We have studied some issues related to blockchain-based data certification, with particular reference to traceability applications, where an almost continuous flow of data needs to be

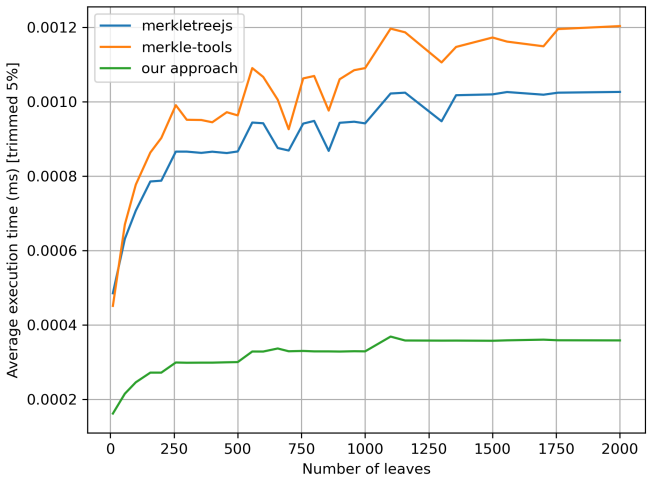

Fig. 8: Proof extraction comparison

certified by writing certification data onto a public blockchain. We considered and compared two possible strategies for organizing the data to be certified and computing the related certification information. The second one, in particular, makes use of Merkle trees and is particularly suitable for keeping the monetary cost of blockchain transactions low when the data to be certified are many and generated in an almost continuous manner. We have also considered a practical use case, concerning the certification of attendance at cultural events or tourism initiatives, showing an efficient implementation of Merkle trees generation and Merkle proof extraction. The techniques considered have also been experimentally evaluated, and our numerical results show the practical feasibility and advantages of our approach, even in the presence of a continuous flow of data to be certified. Last but not least, we showed how the use of multi-level Merkle trees can allow managing transaction failures in an efficient way. As a hint for possible future work, we expect to perform an in-depth security analysis of the system, possibly using automated assessment methods. Moreover, we are evaluating the use of efficient Merkle tree implementations for decentralized authentication based on fuzzy sources [22], [23].

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] Y. Wang, J. Han, and P. Beynon-Davies, "Understanding blockchain technology for future supply chains: a systematic literature review and research agenda," *Supply Chain Management*, vol. 24, no. 1, pp. 62–84, 2019.

[3] L. Compagnucci, D. Lepore, F. Spigarelli, E. Frontoni, M. Baldi, and L. Di Berardino, "Uncovering the potential of blockchain in the agri-food supply chain: An interdisciplinary case study," *Journal of Engineering and Technology Management*, vol. 65, p. 101700, 2022.

[4] T. M. Fernández-Caramés and P. Fraga-Lamas, "A Review on the Use of Blockchain for the Internet of Things," *IEEE Access*, vol. 6, pp. 32 979–33 001, 2018.

[5] U. Bodkhe, S. Tanwar, K. Parekh, P. Khanpara, S. Tyagi, N. Kumar, and M. Alazab, "Blockchain for industry 4.0: A comprehensive review," *IEEE Access*, vol. 8, pp. 79 764–79 800, 2020.

[6] H. Ardina and I. G. Bagus Baskara Nugraha, "Design of a blockchain-based employee attendance system," in *2019 International Conference on ICT for Smart Society (ICISS)*, vol. 7, 2019, pp. 1–4.

[7] D. Dreyfus, "Blockchain technology in the supply chain management classroom: Proof of attendance protocols," *Decision Sciences Journal of Innovative Education*, vol. 22, no. 4, pp. 260–269, 2024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/dsji.12312

[8] A. Mendi, "Blockchain for Food Tracking," *Electronics*, vol. 11, p. 2491, 2022.

[9] IBM (International Business Machines Corporation), "Food Trust," https://www.ibm.com/blockchain/solutions/food-trust.

[10] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, "Blockchain-based traceability in Agri-Food supply chain management: A practical implementation," in *2018 IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany)*, 2018, pp. 1–4.

[11] T.-M. Choi, "Blockchain-technology-supported platforms for diamond authentication and certification in luxury supply chains," *Transportation Research Part E: Logistics and Transportation Review*, vol. 128, pp. 17–29, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1366554519303540

[12] E. F. G. Dias, "Ethereum smart contracts for educational certificates," 2018.

[13] A. Rustemi, F. Dalipi, V. Atanasovski, and A. Risteski, "A systematic literature review on blockchain-based systems for academic certificate verification," *IEEE Access*, vol. 11, pp. 64 679–64 696, 2023.

[14] "Blockcerts," https://www.blockcerts.org/.

[15] E. Nyaletey, R. M. Parizi, Q. Zhang, and K.-K. R. Choo, "BlockIPFS-blockchain-enabled interplanetary file system for forensic and trusted data traceability," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 18–25.

[16] D. Costa, M. Teixeira, A. Pinto *et al.*, "High-performance blockchain system for fast certification of manufacturing data," *SN Applied Sciences*, vol. 4, p. 25, 2022.

[17] M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo, "Fractal Merkle Tree Representation and Traversal," in *Topics in Cryptology — CT-RSA 2003*, M. Joye, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 314–326.

[18] M. Szydlo, "Merkle Tree Traversal in Log Space and Time," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 541–554.

[19] P. Berman, M. Karpinski, and Y. Nekrich, "Optimal trade-off for Merkle tree traversal," *Theoretical Computer Science*, vol. 372, no. 1, pp. 26–36, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397506008693

[20] "merkletreejs: Construct Merkle Trees and verify proofs in JavaScript." accessed: 01-2025. [Online]. Available: https://github.com/merkletreejs/merkletreejs

[21] "merkle-tools: Tools for creating Merkle trees, generating merkle proofs, and verification of merkle proofs." accessed: 01-2025. [Online]. Available: https://github.com/Tierion/merkle-tools

[22] N. Abo Alzahab, G. Rafaiani, M. Battaglioni, F. Chiaraluce, and M. Baldi, "Decentralized Biometric Authentication based on Fuzzy Commitments and Blockchain," in *Proceedings of the Sixth International Conference on Blockchain Computing and Applications (BCCA) 2024*, Nov. 2024.

[23] P. Santini, G. Rafaiani, M. Battaglioni, F. Chiaraluce, and M. Baldi, "A blockchain consensus protocol based on fuzzy signatures," in *Proc. of 2023 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2023.