A Data Integration Platform with Identifiable Falsification Detection and its Evaluation in Automotive Parts Manufacturing

Haruka Hori
Dept. Computer Science
Ochanomizu University
Tokyo, Japan
haruka-h@ogl.is.ocha.ac.jp

Hieu Hanh Le
Dept. Computer Science
Ochanomizu University
Tokyo, Japan
le@is.ocha.ac.jp

Masato Oguchi
Dept. Computer Science
Ochanomizu University
Tokyo, Japan
oguchi@is.ocha.ac.jp

***Abstract*—To promote a decarbonized society, transparency in carbon footprint of products (CFP) data across supply chains has become critical. This paper proposes a blockchain-based platform for the automotive parts manufacturing industry that enables secure management and representation of each component's CFP while providing precise falsification localization capabilities. Unlike existing methods that handle CFP data in plaintext and cannot identify the exact location of falsification, our approach leverages Merkle tree structures and XOR operations to achieve both pinpoint identification of falsified components and secure CFP data management. In particular, by exploiting the commutative property of XOR operation, we eliminate the need for ordering information in hash aggregation, and by utilizing its self-inversion property, we can accurately distinguish between the components where falsification occurred and those merely affected by it. We implement and evaluate a prototype system using Hyperledger Iroha, blockchain-based open-source Distributed Ledger Technology framework. The execution time of falsification localization increased as the falsification rate grew. Specifically, when verifying a dataset corresponding to the components of an automobile, the process required up to 27.88 seconds, compared to 9.52 seconds in the previous work. The evaluation confirmed that execution time was sensitive to the number of parts and to duplication and falsification rates, while CPU and memory usage were largely unaffected by these two rates. Although the verification is slower than methods that cannot localize falsification, the proposed system provides the critical advantage of precise falsification detection while maintaining secure CFP data management.**

***Index Terms*—Distributed System, Blockchain, Supply Chain Management, Carbon Footprint of Products**

I. Introduction

The automotive industry faces increasing pressure to achieve carbon neutrality by 2050, driven by stricter global environmental regulations. For example, the European Union's Carbon Border Adjustment Mechanism and Japan's commitment to a 73% reduction from 2013 levels in greenhouse gas emissions by 2045 have made carbon footprint of products (CFP) management and representation across supply chains a critical requirement [1] [2]. The CFP is a framework that quantifies the direct and indirect greenhouse gas emissions generated throughout each stage of a product's life cycle, converts them into CO_2 equivalents, and presents the results on a per-product basis [3]. A CFP management platform must represent traceability in distributed database systems spanning multiple companies while aggregating, calculating, and securely storing CFP data. CFP management infrastructure represents traceability through distributed database systems spanning multiple enterprises, aggregating, calculating, and storing CFP data. However, several critical challenges must be addressed. First, because multiple different enterprises participate in the distributed system, it is essential to ensure the reliability of collaboration between different companies. Second, the system must implement a functionality to detect falsification that may occur both internally and externally. Third, supply chains typically have hierarchical configurations, requiring recursive exploration for CFP calculations. To address these challenges, we have previously proposed data verification functionality on peer-to-peer networks using blockchain smart contracts [4]. While effective in detecting falsification, the approach had several limitations: First, although the verification function could detect the occurrence of falsification, it could not identify the exact location where it occurred. Second, the design assumed the absence of component duplication, and thus did not adequately reflect real-world conditions. Third, all CFP data were processed and stored in plaintext, with insufficient consideration for information security.

This study addresses these limitations by combining cryptographic Merkle trees with the commutative and self-inversion properties of XOR operations. This approach enables both precise falsification localization and secure CFP data management.

The contributions of this study are summarized as follows.

- We propose a novel data structure, the *hashed component tree*, which, similar to Merkle trees that concatenate hash values, integrates multiple hash values using XOR operations. By leveraging the *commutative property* of XOR operation, the ordering information required in Merkle trees for hash aggregation is eliminated. The hashed component tree thus enables both secure CFP data

This research was partially supported by JST CREST JPMJCR22M2

management and precise falsification localization.
- We develop an efficient algorithm with *self-inverse property* of XOR operation that identifies the exact components affected by falsification, rather than simply detecting its presence.
- We implement and evaluate a prototype system using Hyperledger Iroha, blockchain-based open-source Distributed Ledger Technology (DLT) framework, demonstrating scalability from small (30 parts) to large (30,000 parts) supply chain scenarios.
- We provide a comprehensive performance analysis highlighting the trade-offs between security, accuracy, and computational efficiency.

The rest of the paper is organized as follows: Section II introduces related research, Section III provides background knowledge, Section IV presents the proposed system, Section V reports the experimental evaluation results, Section VI analyzes the proposed system, and Section VII offers the conclusions.

II. Related Work

Calculating the CFP in manufacturing involves tracking greenhouse gas emissions across complex supply chains. Each component's CFP includes both direct manufacturing emissions and accumulated emissions from all sub-components. This hierarchical structure creates dependencies where falsification at any level can propagate throughout the entire system.

A. Industry Data Spaces and Regulations

Across regions, cross-company data spaces are being advanced to tackle societal and economic challenges via trusted data sharing. In Europe, *Catena-X* provides a standardized, interoperable data ecosystem for the automotive value chain with data sovereignty and compliance support, including preparation for the EU battery passport and related regulations [5], [6]. The EU Battery Regulation mandates life-cycle transparency such as carbon footprint declarations and digital product passports, accelerating supply-chain-wide CFP management [7]. Concrete deployments are emerging; e.g., Volvo's battery passport, developed with Circulor, records provenance and carbon footprint using blockchain to meet EU requirements. Furthermore, interoperability with the Ouranos Ecosystem, described later in this paper, is also being pursued [8].

In Japan, METI's *Ouranos Ecosystem* advances an interoperable, cross-industry dataspace architecture toward Society 5.0, with reference models and implementation programs for logistics/commerce data sharing and data sovereignty [9], [10]. Public materials describe blockchain/DLT and smart contracts as enabling technologies for secure, tamper-resistant data exchange within such dataspaces.

B. Studies on Blockchain for Supply-Chain CFP and Traceability

Recent studies have explored the use of blockchain as a foundation for managing CFP and enhancing sustainability across supply chains. Pekel and Yayla [11] proposed a Blockchain-Based Carbon Footprint Management framework that treats CFP as a verifiable product attribute. By embedding CFP into blockchain records, they demonstrated how product-level emissions can be securely tracked and disclosed, ensuring data integrity across organizational boundaries.

Wang et al. [12] introduced a conceptual framework highlighting how blockchain improves supply chain integration capability and reduces carbon emissions. Their study emphasized the role of transparency, trust, and smart contracts in achieving low-carbon collaboration among enterprises.

Saberi et al. [13] conducted a comprehensive review on Blockchain technology in supply chain management, identifying innovations, applications, and challenges. They provided a taxonomy of use cases, including traceability, counterfeit prevention, and environmental performance, which underscores blockchain's versatility in addressing supply chain sustainability issues.

Kouhizadeh and Sarkis [14] proposed a blockchain-based approach for multi-echelon sustainable supply chains. They analyzed how distributed ledgers can support sustainability at different tiers of supply chains, with particular attention to coordination and information sharing mechanisms.

Finally, Gu [15] explored blockchain-enabled statistics and management of carbon emissions in the automotive industry supply chain. Their study highlighted the potential of blockchain to provide secure and transparent aggregation of emissions data, serving as a practical foundation for CFP monitoring.

While these works demonstrate the potential of blockchain for improving transparency, traceability, and sustainability in supply chains, they primarily focus on detection and management of emissions. In contrast, our study introduces the concept of a *hash component tree*, which not only detects CFP falsification but also enables its precise localization by exploiting XOR operation aggregation and path-based disambiguation in hierarchical product structures.

III. Background

A. Blockchain

Blockchain is a distributed ledger that links data in chronological order as blocks. It was invented based on a paper posted by Satoshi Nakamoto in 2008 [16]. Each block contains the hash value of the previous block and is connected like a chain, which makes the blockchain highly tamper-resistant. Furthermore, because blockchain is managed in a distributed manner by its participants, it is characterized by fault tolerance and high availability.

Some blockchains, such as Ethereum [17], also support smart contracts. A smart contract is a mechanism by which a contract is automatically executed on the blockchain once predefined rules are satisfied. Because it does not require intermediaries and is executed based on pre-programmed conditions, it is not only highly efficient but also provides transparency and tamper resistance. In Hyperledger Iroha [18], which is used in this study, smart contracts are implemented as built-in commands.

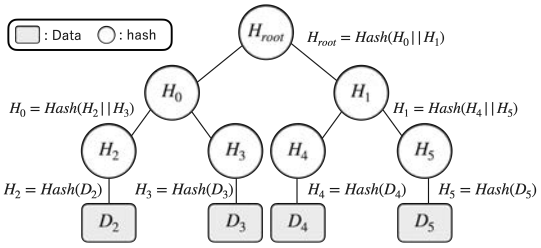


Fig. 1: Example of a Merkle tree

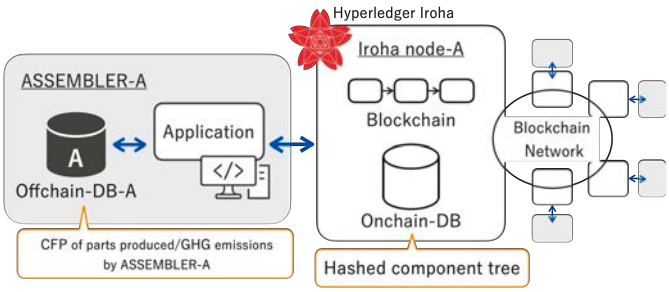


Fig. 2: Overall structure of the proposed system

B. Merkle Tree

A Merkle tree is a binary tree data structure that stores hash values [19]. Fig. 1 shows an example of a Merkle tree with four data items. The leaf nodes $H_2 - H_5$ store the hash values of the data items $D_2 - D_5$. Intermediate nodes store the hash of the concatenated child node values, calculated upward from the leaves. For example, in Fig. 1, intermediate node H_0 is the hash of the concatenation of H_2 and H_3. Ultimately, the Merkle root H_{root} is obtained.

Merkle trees have the advantage of efficiently verifying the integrity of large-scale data efficiently. In a Merkle tree generated from n data items, the verification process can be performed using $log_2(n)$ data items with a computational complexity of $O(log_2(n))$.

C. XOR Operations

XOR operations possess unique mathematical properties that make them suitable for cryptographic applications:

1) Identity : $a \oplus 0 = a$
2) Self-inverse : $a \oplus a = 0$
3) Commutativity : $a \oplus b = b \oplus a$
4) Associativity : $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

These properties enable efficient computation and reversal operations critical for our falsification localization mechanism.

IV. Design And Approach

This sections describes the overall system design, then provides detail definitions, and finally explains the two main processes used in the designed system.

A. System Design

Fig. 2 shows the overall structure of the proposed system. Each assembler belonging to the supply chain has an Offchain-DB and an application within its local environment. The Offchain-DB stores the CFP and GHG values of parts manufactured by the company in plaintext. The application executes the main processes of the proposed system.

Each assembler participates in the blockchain network of Hyperledger Iroha through its Iroha node. On each Iroha node, the blockchain and the Onchain-DB constructed from it are maintained, and these are synchronized across all nodes. The Onchain-DB stores the hashed component tree rooted at the automobile.

The proposed system consists of two main processes to enable CFP management with falsification localization. The first is the tree generation process of the hashed component tree, in which each part's hashed component tree is generated, and the CFP data are stored in a verifiable form. The second is the verification process using the hashed component tree, which identifies falsification in the managed CFPs through peer-to-peer data verification, one of the challenges addressed in this study. Definitions and details of each process are described in the following subsections.

B. Definitions

This section introduces the definitions of component trees, CFPs, and hashed component trees as used herein.

1) Component Tree

In this study, we define a component tree as a representation of the parent-child relationships among parts that constitute a product in a tree-like structure. An example is shown in Fig. 3, which illustrates the component tree of part P_0. Part P_0 consists of $\{P_1, P_2\}$, and part P_1 consists of $\{P_2, P_3$ (two units)$\}$. A part such as P_2, which is used in multiple assemblies, is referred to as a duplicated part. Here, each part forms its own component tree with itself as the root; when integrating all component trees, the root becomes the automobile. It should also be noted that a single component tree may include parts manufactured by different companies.

2) CFP

In this study, we focus only on the manufacturing phase in the life cycle of an automobile. Therefore, the CFP is calculated solely based on the GHG emissions generated by companies involved in the manufacturing of automobile parts. Fig. 4. illustrates the schema defined in this paper for the automotive parts manufacturing industry, where supplied parts (PARTSUPP) are provided by suppliers (SUPPLIER), assembled by factories (ASSEMBLER) through assembly processes (ASSEMBLE PROCESS), and then integrated into parent parts (PARENTS PART). By hierarchically assembling parent parts (PARENTS PART) from supplied parts (PARTSUPP), the final automobile is produced.

Let the CFP of part P_n be CFP_n, and let the GHG emissions produced during the manufacturing process by the assembler of P_n be GHG_n. Then, for parts P_0 and P_1 in Fig. 3, their CFP values CFP_0 and CFP_1 are obtained as follows:

$$CFP_0 = GHG_0 + (CFP_1 + GHG_2) \quad (1)$$

$$CFP_1 = GHG_1 + (GHG_2 + GHG_3 \times 2) \quad (2)$$

3) Hashed Component Tree

A hashed component tree is a structure generated from the corresponding component tree. Each part is assigned a hash value, which is determined by combining multiple hash values through XOR operations, inspired by the construction of Merkle trees. If the CFP value or the hash value of a part is modified, the hash values of other related parts will inevitably change.

As an example, for the component tree in Fig. 3, the corresponding hashed component tree is constructed as shown in Fig. 5. Here, the definition of the hash value for each part depends on whether the part is a single part without children $\{P_2, P_3\}$ or a composite part with two or more children $\{P_0, P_1\}$. The specific definitions are as follows. ***Hash****(x)* denotes the output of the SHA-256 hash function applied to x, and ***Path****(x,y)* denotes the path from part x to part y.

- Any single part P_S: $\boldsymbol{H_S = Hash(CFP_S)}$
- Any composite part P_C with n child parts $\{P_{c\_1}, P_{c\_2}, \cdots, P_{c\_n}\}$ $(2 \leq i \leq n)$: $\boldsymbol{H_C = Hash(CFP_C) \oplus H_{c\_1} \oplus H_{c\_2} \oplus \cdots \oplus H_{c\_n}}$ If P_{c_i} is a duplicated part, H_{c_i} is replaced as follows: $\boldsymbol{Hash(H_{c\_i} || Path(P_C, root))}$

In contrast to the Merkle tree introduced in Section III-C, where the hash values of two child nodes are concatenated in order, the component tree has no inherent ordering among parts. Therefore, XOR operation, which satisfies the commutative property, is adopted to combine hash values independently of order. However, when duplicated parts are included in the hashed component tree, their self-inverse property may cancel each other out at upper nodes. To address this, the path from the parent part of the duplicated part to the root part (automobile) is concatenated to the hash value. Because this path is unique, it prevents cancellation due to the self-inverse property while still enabling falsification localization.

C. Tree Generation Process

The hashed component tree generation process is performed to store CFP data in a verifiable form. It is carried out during initialization, such as when a new part is added to the system.

The tree generation process proceeds as follows.

1. **Aggregation**: Aggregate the GHG values from all assemblers participating in the system.
2. ***CFP* Calculation**: Compute the CFP of each part from the aggregated GHG values.
3. **Hashed Component Tree Generation**: Compute the hash value of each part according to the definition of the hashed component tree.
4. **Write to Onchain-DB**: Store the hash values from Step 3 in the Onchain-DB using built-in commands of Iroha. A new block is added to the blockchain after consensus is achieved.
5. **Write to Offchain-DB**: If Step 4 succeeds, store the CFP values from Step 2 in each Offchain-DB. If Step 4 fails, the process is undone.

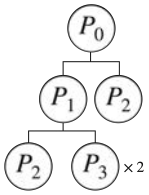


Fig. 3: Example of a component tree.

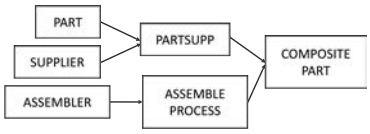


Fig. 4: Schema for the CFP.

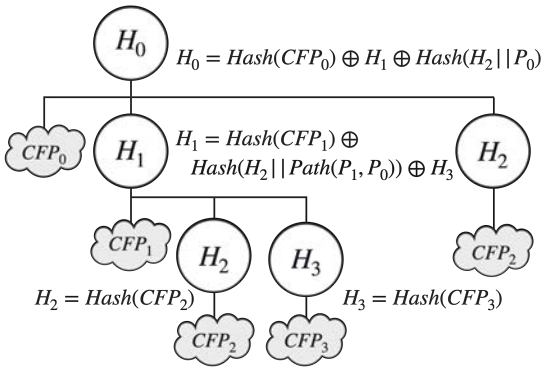


Fig. 5: Hashed component tree values corresponding to Fig. 3.

D. Verification Process

The verification process checks whether the CFP data of a component tree have been falsified. Because it relies on the hashed component tree stored in the Onchain-DB, the tree generation process must have been executed at least once beforehand. Falsification is assumed to occur only in the Offchain-DB.

The verification process proceeds as follows.

1. **Obtain Existing Hashed Component Tree**: Retrieve the hashed component tree for comparison from the Onchain-DB.
2. **Generate New Hashed Component Tree**: Generate a new hashed component tree in the same manner as in the tree generation process.
3. **Comparison**: Compare only the root hash of the two hashed component trees obtained in Steps 1 and 2. If they match, the verification succeeds and the process ends. If they do not match, compare the entire hashed component trees and extract the parts with inconsistent hash values (called suspicious parts).
4. **Falsification Localization**: Input the suspicious parts obtained in Step 3 into Algorithm 1.

When proceeding to Step 4, the suspicious parts extracted in Step 3 are given as input. Suspicious parts may include both the parts where falsification actually occurred and those affected during the construction of the hashed component tree. To distinguish between these two types, the self-inverse property of XOR operation is applied to eliminate the effects of falsification. Note that the impact of falsification propagates to all parts along the path from the falsified part to the root part. Thus, the suspicious parts also represent the collection of paths from falsified parts to the root part.

1) Falsification Localization Process

Algorithm 1 shows the falsification localization process. Here, the function $GetCorrectHash(x)$ connects to the Onchain-DB to retrieve the hash value of part x, the function $Xor()$ computes the XOR operation of hexadecimal values given as arguments, the function $Hash()$ computes the hash of its input, and the function $Path(x, y)$ returns the path from part x to part y.

2) Example of Algorithm 1 Behavior

The behavior of Algorithm 1 is illustrated using the parts tree shown in Fig. 5. Assume that the CFP of part P_2, denoted CFP_2, has been falsified and replaced with CFP_2'. After completing Step 3 of the Verification Process, because P_2 is a duplicate part, two paths are obtained: Path1 : $[P_2, P_1, P_0]$ and Path2 : $[P_2, P_0]$. The corresponding hash values are defined as follows:

$$H_2' = Hash(CFP_2') \tag{3}$$

$$H_1' = Hash(CFP_1) \oplus Hash(H_2'||Path(P_1, P_0)) \oplus H_3 \tag{4}$$

$$H_0' = Hash(CFP_0) \oplus H_1' \oplus Hash(H_2'||P_0) \tag{5}$$

We now explain the behavior of Algorithm 1 when $P = [Path1, Path2]$ is given as input. First, Path1 is retrieved from P, and the falsification of its starting point P_2 is confirmed. While traversing upward along the path, the influence of H_2' is eliminated through XOR operations. Since P_2 is a duplicated part, the duplicate-handling procedure is applied:

$$\begin{aligned} &H_1' \oplus \{Hash(H_2'||Path(P_1, P_0)) \oplus Hash(H_2||Path(P_1, P_0)\} \\ &= Hash(CFP_1) \oplus Hash(H_2||Path(P_1, P_0)) \oplus H_3 \\ &= H_1 \end{aligned} \tag{6}$$

Using (6), the effect of H_2' is also eliminated from (5):

$$\begin{aligned} &H_0' \oplus \{H_1' \oplus H_1\} \\ &\quad = Hash(CFP_0) \oplus H_1 \oplus Hash(H_2'||P_0) \neq H_0 \end{aligned} \tag{7}$$

Next, we consider the behavior along Path2. The starting point P_2 has already been identified as a falsified part. By applying (7), the influence of H_2' is eliminated:

$$\begin{aligned} &H_0' \oplus \{H_1' \oplus H_1\} \oplus \{Hash(H_2'||P_0) \oplus Hash(H_2||P_0)\} \\ &\quad = Hash(CFP_0) \oplus H_1 \oplus Hash(H_2||P_0) = H_0 \end{aligned} \tag{8}$$

Since the influence of H_2' has been completely eliminated from all parts along the paths, the correct hash values are recovered. At this point, Algorithm 1 terminates. The only part confirmed as falsified is P_2, thereby completing the falsification localization.

V. Evaluation

In this section, we investigate the performance of falsification localization in the proposed system.

The main processes, tree generation process and verification process were implemented by issuing SQL queries from the application (Python) to temporary tables in the Onchain-DB (PostgreSQL). These temporary tables aggregate intermediate data required for the processes, such as each node's CFP data.

Algorithm 1 Falsification Localization

Require: Path List for suspicious parts P (2D Array)
Ensure: Set of confirmed falsified parts C

$PartInfo(string : part\_id, binary : hash, PartInfo : parents\_part)$
$C \leftarrow \emptyset$
$P_{next} \leftarrow [\ ]$ ▷ Next suspicious paths
while Number of elements in P is not 0 **do**
  $P_{\text{now}} \leftarrow \text{pop}(P)$ ▷ Obtain the first path
  $target \leftarrow \text{pop}(P_{\text{now}})$ ▷ Path start node
  $C \leftarrow C \cup \{target\}$ ▷ Confirmed falsified part
  $CheckedHash \leftarrow \text{GetCorrectHash}(target)$
  $parent \leftarrow target.parents\_part$
  repeat
    ▷ Eliminate the effect of falsification by XOR
    if $target$ is a duplicated part **then**
      $CheckedHash \leftarrow \text{Xor}\Big(parent.hash,$
      $\text{Hash}(target.hash \,\|\, \text{Path}(target, Root)),$
      $\text{Hash}(CheckedHash \,\|\, \text{Path}(target, Root))\Big)$
    else
      $CheckedHash \leftarrow$
      $\text{Xor}(parent.hash,$
      $target.hash, CheckedHash)$
    end if
    ▷ Re-compare with the correct value
    if $CheckedHash$ and
      $\text{GetCorrectHash}(parent)$ don't match **then**
      ▷ Still includes unidentified falsification
      $P_{next}.\text{append}(parent)$
    end if
    $parent.hash \leftarrow CheckedHash$
    $target \leftarrow parent$
  until $parent$ is $Root\ part$
  $P.\text{append}(P_{next})$
end while
return C

In addition, instead of using the existing built-in commands of Iroha written in C++, we developed new commands specifically optimized for the proposed method to handle the tree generation process.

A. Overview

We measured the execution times, CPU utilization and memory usage of the two main processes using randomly generated component trees under the following constraints.

- Total number of parts: 30 / 300 / 3,000 / 30,000.
- Duplication rate of parts: 0 / 10 / 20 / 30 / 40%.

For each combination of total number of parts and duplication rate, three component trees were generated. Here, 30,000 parts correspond to the approximate number of parts in a single automobile.

The implementation environment and machine specifications are shown in Fig. 6. Although evaluation on a real

network would be desirable for practical deployment, it has been reported that the performance of Iroha networks does not significantly differ between real and virtual networks when the latency among each node is small. Therefore, we adopted a Docker-based environment. For each participating assembler, we constructed a set of an Ubuntu 22.04 LTS container, in which Iroha was built, and a PostgreSQL container that provides Onchain/Offchain-DB functions. In this experiment, the number of assemblers participating in the system was simply set to three.

Here, the CPU utilization and memory usage were measured using the "docker stats" command. Note that the CPU utilization shown by docker stats represents usage relative to a single CPU core (100% = one core fully utilized). For instance, a value of 200% on a 16-core system corresponds to approximately 12.5% of the total CPU capacity.

B. Performance of the Tree Generation Process

First, we measured the execution time of the hashed component tree generation process for 60 component trees (three instances for each combination of total number of parts and duplication rate). Each measurement was repeated three times, and the average execution time, CPU utilization and memory usage per combination was calculated, as shown in Fig. 7(a), 7(b) and 7(c).

As shown in Fig. 7(a), for 30 and 300 parts, the execution time increased as the duplication rate increased (e.g., from 0.80 second to 0.93 second for 30 parts, and from 0.71 second to 0.96 second for 300 parts), indicating that duplication handling affects execution time at small scales. Conversely, for 3,000 and 30,000 parts, the execution time decreased with higher duplication rates (e.g., from 1.63 second to 1.41 second for 3,000 parts, and from 11.55 seconds to 7.63 seconds for 30,000 parts), suggesting that the reduction in the number of parts to be processed outweighed the cost of duplication handling. In Fig. 7(b) and 7(c), the values increased as the number of parts increased.

Next, we examined the overhead of Iroha built-in command processing. Defining the remaining part as application processing, Fig. 7(d) shows the breakdown of execution time. The results indicate that Iroha built-in command processing accounts for approximately 70-81% of the total tree generation process, with a tendency for the proportion to increase as the duplication rate decreases. This overhead likely explains why there is no significant difference in execution time between 30 and 300 parts. Although the built-in command processing time strongly influences the total execution time of the tree generation process, the tree generation process itself is executed infrequently, so its impact on practicality is expected to be limited.

C. Performance of the Verification Process

We first evaluated the execution time, CPU utilization and memory usage of the verification process when the result was consistent, that is, when no falsification existed in the target component tree. Using the same component trees as in Section V-B, we measured the execution time three times for each case. The averages for each combination of total number of parts and duplication rate are shown in Fig. 8.

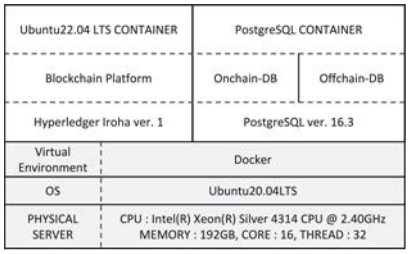
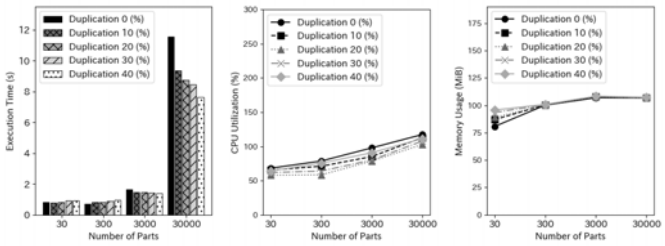
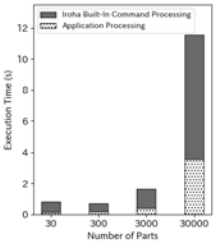


Fig. 6: Build environment



(a) Execution time. (b) CPU Utilization. (c) Memory Usage.



(d) Breakdown of execution time. (30,000 parts)

Fig. 7: Performance of the Tree Generation Process.

As illustrated in Fig. 8(a), for 30 and 300 parts, the coefficients of variation of execution time were 1.94% and 2.07%, respectively, indicating very small variance (e.g., from 0.14 second to 0.15 second for 30 parts, and from 0.15 second to 0.16 second for 300 parts). In contrast, for 3,000 and 30,000 parts, execution times tended to decrease as the duplication rate increased(e.g., from 0.29 second to 0.25 second for 3,000 parts, and from 2.08 second to 1.40 second for 30,000 parts. Compared with the tree generation process of the same component trees, the verification process required only 0.15-0.23 times the execution time. In this case, the verification process is completed by comparing the root hashes, and the difference from the tree generation process stems from the execution of Iroha built-in commands. Furthermore, when comparing the CPU utilization (Fig. 7(b)) and memory usage (Fig. 7(c)) of the tree generation process with those of the verification process (Fig. 8(b) and Fig. 8(c)), the tree generation process showed higher values for up to 300 parts, whereas for 3,000 parts and above, the verification process exhibited higher values.

Next, we evaluated the execution time when the verification result was inconsistent, that is, falsification occurred in the

target component tree. For the component trees used in Section V-B (excluding those with 30 parts), we randomly selected 1%, 5%, 10%, or 15% of the parts and falsified their CFP data with different values. For each falsified tree, the verification process was executed three times, and the average execution time, CPU utilization and memory usage per duplication rate was calculated and grouped by total number of parts, as shown in Fig. 9.

As illustrated in Fig. 9(a), the execution time increased as the falsification rate increased. This is attributed to the increased number of iterations required in the falsification localization process. We also evaluated the identification rate of falsified parts per execution. In all cases, the identification rate was 100%, meaning that all falsified parts were detected and the expected behavior was achieved. As shown in Fig. 9(b), for the component trees with 30,000 parts, the CPU utilization increased with higher falsification rates. In contrast, for the non-duplicated dataset, CPU utilization remained almost constant. No such relationship was observed in the case of 300 and 3,000 parts. In Fig. 9(c), memory usage remained almost constant regardless of the falsification rate. At the highest duplication rate of 40%, memory usage was approximately 47 MiB for 300 parts, 173 MiB for 3,000 parts, and 175 MiB for 30,000 parts. These values are roughly consistent with those shown in Fig. 8(c) for each total number of parts.

VI. Discussion

This section provides a discussion of the proposed method's performance.

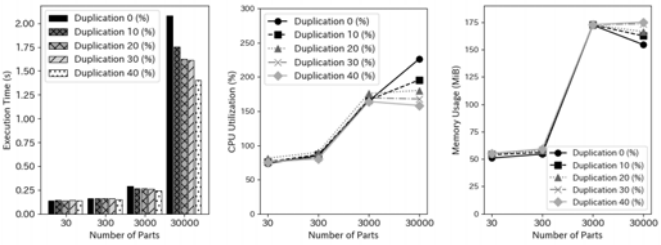
A. Tamper Resistance of the Hashed Component Tree

In this study, falsification detection and localization were achieved by constructing a hashed component tree using XOR operation. Here, we discuss the tamper resistance of the hashed component tree. As a premise, the Onchain-DB that stores the hashed component tree is highly reliable due to the properties of the blockchain. The following discussion assumes that the correctness of the hash values stored in the Onchain-DB is always guaranteed.

In the structure of the hashed component tree, the hash value of each node is derived from the XOR operation of the hash of its own CFP value and the hash values of its child nodes. Owing to this structure, if a CFP value is falsified within the component tree, the hash value of that node and its upper nodes will inevitable differ, thereby enabling falsification detection. In particular, in this method, falsification at any node propagates to all nodes along the path to the root. By tracing the propagation path, not only can the falsified location be identified, but falsification of an intermediate node's CFP can also be detected from inconsistencies in the XOR values.

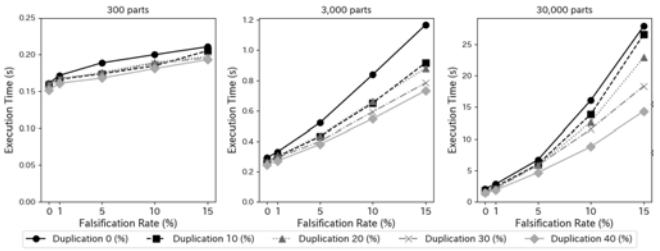
One potential concern is that simultaneous falsification at multiple points may coincidentally result in consistent XOR operation outputs, causing cancellation due to the self-inverse property of XOR operation at upper nodes. However, because CFP values are expressed as real numbers, deliberately manipulating multiple falsifications so that they cancel each other through XOR operation would require highly precise control. Combined with the difficulty of falsifying hash values stored in the Onchain-DB, such scenarios can be considered practically negligible. Furthermore, as SHA-256 with strong collision resistance is employed, the possibility of generating the same hash from different CFP values can also be disregarded in practice. Therefore, the reliability of the hash values themselves is high.

Overall, the proposed hashed component tree structure, by leveraging XOR operation, not only enables falsification detection but also supports localization. As long as the correctness of the hash values recorded in the Onchain-DB is guaranteed, the proposed method can be regarded as highly

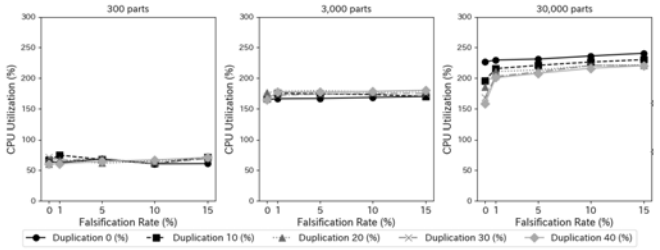


(a) Execution time. (b) CPU Utilization. (c) Memory Usage.

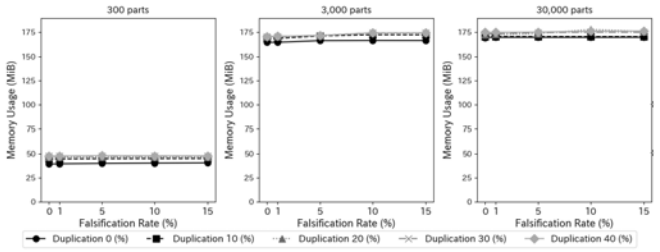
Fig. 8: Performance of the Verification process (consistent).



(a) Execution time. Note that the Y-axis scale is not uniform.



(b) CPU Utilization.



(c) Memory Usage.

Fig. 9: Performance of the Verification process (inconsistent).

tamper-resistant.

B. Comparison with Previous Work

We compare our study with previous work [4]. The previous study proposed a method for calculating CFP values within a component tree and detecting the presence of falsification. In contrast, this study aims for higher security and more efficient verification by proposing the hashed component tree, which enables falsification localization of CFP values, a capability not achieved in previous work.

From the perspective of execution speed, as shown in Section V.C, the verification process in this study required up to 27.88 seconds (30,000 parts, duplication rate 0%, falsification rate 15%), whereas the CFP calculation process in the previous work took 9.52 seconds under the same conditions. We achieved "localization of falsification," which was not possible in the previous approach, at the cost of approximately 18.36 additional seconds.

From the perspective of implementation, most of the processes in the previous work were executed within a single Iroha embedded command. In contrast, the implementation in this study relies mainly on the application layer implemented in Python. While embedded commands impose strong development constraints, application-layer implementation allows greater flexibility and easier maintenance.

Therefore, compared with the previous work, the proposed system offers better practicality.

VII. Conclusion and Future Work

In this study, to enhance the practicality and security of a distributed database system for managing CFPs of automobile parts developed in previous work, we proposed a "hashed component tree." In this structure, CFP values are hashed and then integrated by XOR operation according to the part hierarchy, thereby assigning a hash value to each part. By leveraging the commutative property of XOR operation, ordering information among parts can be omitted, and in the event of falsification, the specific falsified location can be identified using the self-inverse property of XOR operation. For duplicated parts, concatenating the path prevents cancellation caused by the self-inverse property. Evaluation experiments confirmed that the proposed system behaved as expected, with the falsification localization rate in the verification process reaching 100% in all cases. In addition, the results showed that execution time was sensitive to the number of parts and to duplication and falsification rates, while CPU and memory usage were largely unaffected by these two rates. Compared to previous work, the proposed system required longer processing times, highlighting a trade-off between speed and accuracy in data verification platforms.

For future work, we will address the distribution of the hashed component tree stored in the Onchain-DB. In the current implementation, the Onchain-DB is redundant across all Iroha nodes, and the hashed component tree rooted at the automobile is managed. Consequently, the Onchain-DB contains a large amount of data unrelated to the parts manufactured by a particular assembler. This raises concerns regarding unnecessary data storage, security, and increased load on each distributed node.

Acknowledgment

This research was partially supported by JST CREST JPMJCR22M2. The authors also express their gratitude to the CREST Group for their valuable discussions and support in conducting the experiments.

References

[1] (2023) Carbon border adjustment mechanism. European Commission. [Online]. Available: https://taxation-customs.ec.europa.eu/carbon-border-adjustment-mechanism_en

[2] J. Ministry of the Environment. (2025, Feb.) Japan's nationally determined contribution (ndc). [Online]. Available: https://www.env.go.jp/content/000291805.pdf

[3] *ISO 14067:2018 - Greenhouse gases - Carbon footprint of products - Requirements and guidelines for quantification*, Jan. 2018.

[4] H. Hori and M. Oguchi, "A study of blockchain-based metadata management and its use for data verification," in *Proc. of the 12th International Symposium on Computing and Networking Workshops(CANDAR2024)*, Okinawa, Japan, 2024, pp. 63–68.

[5] (2021) Catena-x. [Online]. Available: https://catena-x.net/

[6] (2023, Feb.) Digital product passports as the enabler for the circular economy. Catena-X. [Online]. Available: https://catena-x.net/wp-content/uploads/2025/05/DPP_Catena-X_FINAL.pdf

[7] "Regulation (eu) 2023/1542 of the european parliament and of the council of 12 july 2023," *Official Journal of the European Union*, vol. 1542, Jul. 2023. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX%3A32023R1542

[8] (2023, Mar.) Catena-x and ouranos ecosystem successfully demonstrate data space interoperability. Catena-X. [Online]. Available: https://catena-x.net/news/catena-x-and-ouranos-ecosystem-successfully-demonstrate-data-space-interoperability/

[9] (2023) Ouranos ecosystedm. Ministry of Economy, Trade and Industry, Japan. [Online]. Available: https://www.meti.go.jp/english/policy/mono_info_service/connected_industries/ouranos.html

[10] "Ouranos ecosystem dataspaces reference architecture model," White Paper, Information-technology Promorion Agency, Japan, Mar. 2025. [Online]. Available: https://www.ipa.go.jp/en/digital/architecture-guidelines/ouranos-ecosystem-dataspaces-ram-white-paper.html

[11] U. Pekel and O. Yayla, "Blockchain-based carbon footprint management," in *Cryptology ePrint Archive*, vol. 2024/1863, 2024.

[12] M. Wang, B. Wang, and A. Abarehii, "Blockchain technology and its role in enhancing supply chain integration capability and reducing carbon emission: A conceptual framework," in *Sustainability*, vol. 12, no. 24, 2020, p. 10550.

[13] N. Kumar, K. Kumar, A. Aeron, and F. Verre, "Blockchain technology in supply chain management: Innovations, applications, and challenges," Tech. Rep. 10020, 2019.

[14] V. K. Manupati, T. Schoenhrr, M. Ramkumar, S. M. Wanger, S. K. Pabba, and R. I. R. Singh, "A blockchain-based approach for a multi-echelon sustainable supply chain," *International Journal of Production Research*, vol. 58, no. 7, pp. 2222–2241, 2019.

[15] X. Gu, "Exploration of carbon emission statistics and management in the supply chain of automobile industry based on blockchain technology," *Academic Journal of Management and Social Sciences*, vol. 6, no. 1, pp. 97–100, 2019.

[16] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[17] (2015) Ethereum. [Online]. Available: https://ethereum.org/

[18] (2016) Hyperledger iroha. [Online]. Available: https://soramitsu.co.jp/iroha

[19] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the Theory and Application of Cryptographic Techniques*, 1987.