改ざんが特定可能なサプライチェーン全体でのデータ連携基盤の提案

堀 遥[1]　Le Hieu Hanh[1]　小口 正人[1]

概要：
脱炭素社会実現に向け，複数企業を横断する分散データベースシステムでトレーサビリティを表現し，カーボンフットプリントを管理するようなデータ管理基盤の実現が期待される．我々はこれまでに自動車製造業を題材に，部品各種の製造時 CO_2 排出量の可視化，およびデータの改ざん発生を検知するための検証機能をブロックチェーンベースで実装してきた．
一方で，このシステムにはいくつか懸念点が残されている．一点目に，検証機能では改ざんの発生自体は検知可能であるものの，その発生箇所を特定することができない．二点目に，カーボンフットプリントデータを全て平文で処理・保管している．この課題に対し，我々はハッシュ木の階層的なハッシュ構造と排他的論理和（XOR）の性質に着目し，改ざん箇所の特定とデータの安全性を両立する手法を検討した．提案手法をハッシュ部品木と呼び，これらの実現のため，3 つのメインプロセス，ハッシュ部品木の生成・更新・検証プロセスを実装した．
提案手法の評価実験では，第一にメインプロセスの実行時間を調査した．いずれも期待通りの挙動を得られたほか，より実行頻度の高いと想定される更新・検証プロセスでは最大で 1.5 秒程度に抑えられた．第二に，検証プロセスに対して改ざん率を変化させた実験では，改ざん率の上昇に伴って実行時間も増加する傾向が確認された．一方で，改ざん特定率は，いずれの改ざん率でも 100% を達成した．これらの結果より，提案手法は既存手法と比較しても高い実用性を有しており，現実的な CFP 管理基盤の構築に向けた有効なアプローチであると結論付けられる．

Proposal for a supply chain data integration platform with identifiable falsification detection

HARUKA HORI[1]　HIEU HANH LE[1]　MASATO OGUCHI[1]

1. はじめに

1.1 研究背景

2050 年までにカーボンニュートラル達成という目標を表明する 150 以上の国々において，多様な取り組みが推進されており，その一つにカーボンフットプリントがあげられる．カーボンフットプリント (以降，CFP：Carbon Footprint of Products) とは，図 1 のように製品のライフサイクルの各過程で直接的・間接的に排出された温室効果ガスを CO_2 排出量に換算し，製品単位で表示する仕組みである．

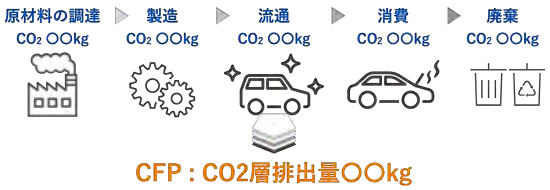


図 1: カーボンフットプリント概要

そして現在，サプラーチェーン全体で CFP を管理して，製品単体の CFP が可視化できるような分散データ基盤の開発が期待されている．このようなデータ基盤実現のメリットととして，一点目にホットスポットの特定より，効率的な削減につながること，二点目に可視化により低炭素・脱炭素製品が積極的に選ばれること，三点目にサプラ

[1] お茶の水女子大学
Ochanomizu University

イチェーン上の他事業者による排出削減が自社の削減とみなされるため，各社の削減の可能性が広がることが挙げられる．

CREST にて進行中のプロジェクト「検証可能なデータエコシステム」[1] では，データに付随する信頼度や来歴を第一級のデータとしてサポートし，任意のデータが検証可能であるようなデータエコシステムの研究開発を目指しており，実証実験として自動車製造業における CFP 管理に取り組んでいる．我々は中でも，データエコシステムの基盤となる分散データベースシステムに，安全性を提供する役割を担当する．

1.2 データ連携基盤に関連する各国の動向

サプライチェーンを横断して連携するようなデータ基盤の構築により，社会課題や経済課題の解決を目指す動きは，各国で推進されている．例えばドイツでは，Catena–X [2] が自動車業界を中心とした企業向けに安全で標準化された企業間データ連携サービスを提供している．欧州では 2023 年 8 月に施行された欧州バッテリー規則にて，バッテリーのライフサイクル全体にわたる CO_2 排出量や資源リサイクル率の欧州委員会への開示が求められており，Catena–X は規則への対応を支援している．さらにアメリカでは，MOBI [3] がブロックチェーン・DLT を基盤としたデータ連携基盤で自動車産業に挙げられる 6 つの課題の解決を目指す．6 つの各テーマで標準デジタルインフラの企画・開発が推進されている．

一方日本では，経済産業省が独立行政法人情報処理推進機構とともに Ouranos Ecosystem [4] [5] プロジェクトが進む．Ouranos Ecosystem のもとでの業種横断的なシステム連携の実現を目指し，人流・物流 DX 及び商流・金流 DX に先行的に着手している．本システムの基盤にはブロックチェーンが用いられており，スマートコントラクト技術によるデータ主権の確保を実現している．

1.3 研究目的

CFP 管理基盤では，複数企業を横断する分散データベースシステムでトレーサビリティを表現し，CFP データを集約・算定・保管することが求められる．自動車製造業という実証実験シナリオにおいては，以下の三つの課題が挙げられる．一点目は複数の異なる企業が分散システムに参加するため，異なる企業間の連携の信頼性を担保する必要がある．二点目に改ざんは外部攻撃だけでなく，自社内においても発生する可能性がある．近年，自動車の品質不正の事例も多く，CFP も例外ではない事から，改ざん検知機能の実現が求められる．三点目は部品は複数の部品を組み立てて作られるため，階層的な構成をしており，製品単位の CFP 値の計算には再帰的な探索が必要となる．

我々はこれまでに，一点目と二点目の課題に対して，Peer to Peer ネットワーク上のデータ検証機能の導入を検討してきた．さらに，三点目の課題に対しては，一度算出した CFP の値を保管し，検証時に再利用することで再帰処理の一部省略を図った．システムの実装にあたっては，実世界で進められている CFP 管理基盤を参考に，ブロックチェーンプラットフォーム Hyperledger Iroha を基盤とする．そして Hyperledger Iroha におけるスマートコントラクト技術である組み込みコマンドを用いて，分散システム上での高速なデータ検証を行い，改ざん検知するようなシステムを実装・評価してきた [6].

一方で，現実装には次の二点の懸念が残されている．一点目に，データ検証機能では改ざんの発生を検知できるものの，発生箇所を特定することができない．二点目に，CFP データを全て平文で処理・保管しており，情報セキュリティ上の配慮が不十分である．

これらの課題を解消し，先行研究より安全で高性能な実装を目指して，本稿では，データ構造の一種であるハッシュ木に着想を得た「ハッシュ部品木」を提案する．提案システムはハッシュ部品木の生成・更新・検証の三つのメインプロセスから構成される．ここで，ハッシュ部品木では，ハッシュ木のようにハッシュ値の連結を繰り返してルートハッシュを求めるのではなく，排他的論理和（XOR）演算を繰り返すことで各ノードのハッシュ値を決定する．特に検証プロセスでは，XOR の自己反転性を応用して改ざん特定を実現する．また，提案システムの実用性を評価するため，三つのメインプロセスそれぞれの性能を調査した．

1.4 本稿の構成

本稿は以下の通り構成される．第 2 節では本研究の前提知識の概要を説明する．第 3 節では，自動車製造業における製品単体の CFP データを算定・ハッシュ部品木を用いて検証する手法を提案する．第 4 節では，提案手法の有効性を確認するため実装システムの評価実験とその結果を述べる．第 5 節では第 4 節の結果を踏まえたシステムの考察を行い，第 6 節でまとめる．

2. 関連知識と関連研究

本節では，まず提案手法の基盤となるブロックチェーンの技術的概要を説明したのちに，本稿で採用したブロックチェーンプラットフォーム Hyperledger Iroha を紹介する．次に，提案手法で応用したハッシュ木の特徴を紹介する．最後に，ブロックチェーンベースのデータ管理基盤に関する関連研究を取り上げる．

2.1 ブロックチェーン

ブロックチェーンは，ブロックと呼ばれるトランザクションを記録する単位を生成し，これをチェーンのように連鎖するデータ構造である．2008 年に Satoshi Nakamoto

により投稿された論文 [7] に基づき，暗号通貨 Bitcoin [8] の公開取引台帳としての役割を果たすために発明された．仮想通貨の基礎技術としての厳格性を担保する仕組みはチェーンにある．一つのブロックにはハッシュ値が付与される．このハッシュ値は，前のブロックのハッシュ値と新規のトランザクションの内容やタイムスタンプなどから算出される．すなわち，ハッシュ値によりブロックの関連が生まれ，これがチェーンとなる．よってブロックチェーンは高い改ざん耐性を持つと言われる．また，Peer to Peer 型のブロックチェーンネットワーク上に参加する各ノードがブロックチェーンを管理するため，欠損ブロックを他のノードから補うことができ，耐障害性と可用性に優れるといった特徴を持つ．

Etherum [9] に代表される一部のブロックチェーンには，スマートコントラクトが実装されている．スマートコントラクトとは，ブロックチェーン上で事前に指定されたルールを満たすと，自動的に契約が締結される仕組みである．仲介者を必要とせず，あらかじめプログラミングされた契約条件や内容をもとに実行されるため，非常に効率的であるほか，高い透明性と耐改ざん性を持つ．また，スマートコントラクトの内容はブロックチェーン上に記録される．本研究で使用する Hyperledger Iroha では組み込みコマンドとしてスマートコントラクトが導入されており，API を通して実行される．

2.2 Hyperledger Iroha

Hyperledger Iroha [10](以下，Iroha という) は 2019 年 5 月に Hyperledger [11] により GA リリースされたパーミッション型ブロックチェーンプラットフォームである．パーミッション型ブロックチェーンは，透明性には欠ける一方で，中央集権型ネットワークでプライバシーが確保され，マイニング不要のため高いパフォーマンスを発揮する．こうした特徴から，パーミッション型は単一の企業や組織内での運用に有効であり，特に銀行間の取引や証券取引での活用が促進されている．

他のパーミッション型ブロックチェーンと比較した Iroha の特徴として以下の 3 点があげられる．

- YAC コンセンサスアルゴリズムを採用
- 組み込みコマンドによるスマートコントラクトの実現
- 汎用的で柔軟な権限機構の実装

YAC アルゴリズムは，BFT：ビザンチン耐性を備えており，高速かつ低遅延である．

2.3 ハッシュ木

ハッシュ木とは，ハッシュ値を格納する二分木のデータ構造である．Merkle-Tree とも呼ばれ，1979 年に Ralph C. Merkle によって発明された [12]．図 2 はデータ数 4 のハッシュ木の例である．葉ノード $H_2 - H_4$ にはデータ $D_2 - D_4$ のハッシュ値が格納される．中間ノードには子ノードのハッシュ値を連結しハッシュ化したものが葉から上方向に演算，格納される．図の例の場合，中間ノード H_0 の場合，H_2 と H_3 を連結してハッシュしたものとなる．最終的に Merkle ルート H_{root} が得られる．

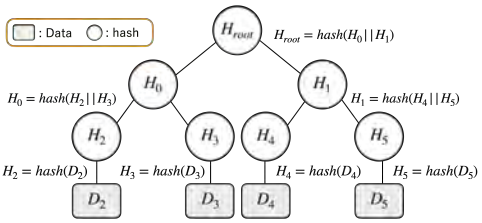


図 2: ハッシュ木の例

ハッシュ木には，大規模データの整合性を高速に検証できるという特徴を持つ．例えば，図 2 中のデータ D_2 を検証する場合，D_2，H_1，H_3 だけを用い，Merkle ルート H_{root} を再算出する．この値が既存の H_{root} と一致すれば検証成功であり，不一致であれば検証失敗となる．このように，n 個のデータから生成されたハッシュ木では，$log_2(n)$ 個のデータを用い，計算量 $O(log_2(n))$ で検証処理を行うことができる．

2.4 関連研究

高い機密性が求められるデータ管理システムにおけるブロックチェーンの技術的な可能性について，これまでに幾つかの論文が述べられている [13–16]．管理データのハッシュ値やポインタ，アクセス権などを含めたメタデータでブロックを構成し，ブロックチェーンに追加することで，安全性の担保されたメタデータを用いたデータ管理を実現している．

中でも，Neha Mishra らの研究 [13] では，PDV：Personal Data Vault と呼ばれる個人の生涯にわたるデジタルドキュメントを，検証可能かつ安全な方法で保管，保存，保護，共有するためのフレームワークの開発し，これにブロックチェーンプラットフォーム Hyperledgr Iroha を採用している．各文書は暗号化，圧縮され，クラウドに安全に保存され，文書の保存先 URL がメタデータとして Hyperledger Iroha に入力されるほか，システムに書類を追加する際のユーザ認証などにスマートコントラクトを利用している．また，開発システムは Markov Tree を用いた予測プリフェッチ機能を備えており，次に発生する要求を予測，事前実行する．

以上の関連研究と 1.2 節で紹介した事例より，業種を横断しながら安心安全なデータ流通を叶えるためにはブロックチェーンは有効であると考えられる．

3. 提案システム

提案システムは，自動車製造業における CFP を，サプ

ライチェーン全体で算定・管理することを目的とする．本システムで算定された製品の CFP は，第三者機関による評価や削減シミュレーションへの活用を想定しており，サプライチェーン全体における二酸化炭素排出量削減の支援に資することが期待される．なお，本システムにおける各プロセスでの障害発生時の対処やエラーハンドリングに関する詳細な議論は，本稿の対象外とする．

本節では，まず本稿で用いる用語および提案するハッシュ部品木の定義を示し，続いて提案システムの詳細な設計について述べる．

3.1 定義

3.1.1 部品木

本稿では，製品を構成する部品の親子関係を木構造のように表現したものを部品木と呼ぶ．図 3 に示した例は部品 P_0 の部品木である．P_0 は $\{P_1, P_2, P_3\}$ で構成され，P_1 は $\{P_4, P_5\}$ で構成される．

ここで，各部品は自身をルートとする部品木を形成しているが，全部品木の部品木を統合させるとルートには自動車が位置することになる．また，一つの部品木であっても異なる ASSEMBLER で製造される部品が存在することに注意する．

3.1.2 CFP

CFP とは，製品のライフサイクルの各過程で直接的・間接的に排出された温室効果ガスを CO_2 排出量に換算し，製品単位で表示する仕組みである．一方で，本稿では自動車のライフラサイクルにおいて，製造部分にフォーカスしており，CFP は自動車製造に携わる企業の CO_2 排出量だけが含まれる．

部品 P_0 が図 3 のように構成される場合, P_0, P_1 の CFP_0, CFP_1 は次のように求める．この時，P_n を製造した ASSEMBLER が製造過程を通して排出した温室効果ガス排出量を $CO_{2(n)}$ とする．

$$CFP_0 = CO_{2(0)} + \{(CO_{2(1)} + CFP_1) + CO_{2(2)} + CO_{2(3)}\}$$
$$CFP_1 = CO_{2(1)} + (CO_{2(4)} + CO_{2(5)})$$

3.1.3 ハッシュ部品木

ハッシュ部品木は，部品木に対応して生成される構造である．各部品にはハッシュ値が与えられるが，この値はハッシュ木の構成手順に着想を得て，複数のハッシュ値を組み合わせることで決定される．

図 4 はハッシュ部品木の一例である．図 3 に示す部品 P_0 の部品木に対し, 図 4 のような対応するハッシュ部品木が構成される．ここで，ハッシュ部品木における各部品のハッシュ値の定義は，子部品を持たない部品 $\{P_2, P_3, P_4, P_5\}$ と，2 個以上の子部品を持つ部品 $\{P_0, P_1\}$ とで異なる．

具体的なハッシュ値の定義は以下の通りである．

- 子部品を持たない任意の部品 P_l のハッシュ値：
  $H_l = hash(CFP_l)$
- n 個の子部品 $\{P_{b-1}, P_{b-2}, \cdots, P_{b-n}\}$ 持つ任意の部品 P_b のハッシュ値 (n≥2)：
  $H_b = hash(CFP_b) \oplus H_{b-1} \oplus H_{b-2} \oplus \cdots \oplus H_{b-n}$

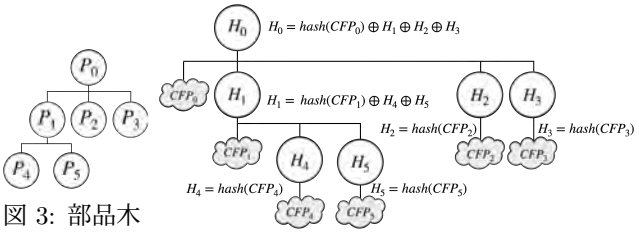


図 3: 部品木の例

図 4: 図 3 に対するハッシュ部品木の値

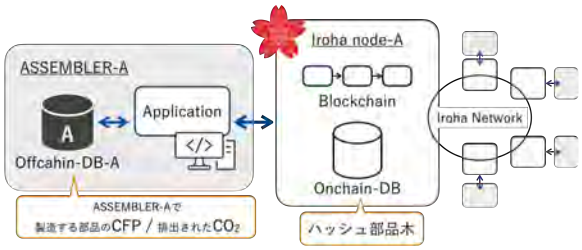


図 5: 提案システムの全体像

ハッシュ部品木を生成する際には，子部品を持たない任意の部品から順に，親部品に向かってハッシュ値が定まっていく．ここで，通常のハッシュ木では，複数の子ノードのハッシュ値を順序を保った上で連結し，それを再度ハッシュ化することで親ノードのハッシュ値を得る．一方，本シナリオでは部品間に順序性が存在しないため，子部品のハッシュ値を順序に依存せずに統合する方法として，排他的論理和（XOR）を用いて親部品のハッシュ値を算出する．このように，提案手法ではハッシュ木の構成原理を一部応用しつつ，部品関係の特徴に即した処理を導入している．

ハッシュ部品木の値は，部品に対応する CFP 値やハッシュ値が変更された場合，他の部品におけるハッシュ値も大きく変化するという性質を持つ．

3.2 提案システム概要

図 5 は提案システムの全体像である．サプライチェーンに属する製造企業 (ASSEMBLER) はローカル環境に Offchain-DB と Application を持つ．Offchain-DB には自身が製造する部品の CFP データと ASSEMBLER で発生した CO_2 排出量が平文で保管されている．Application では提案システムのメインプロセスを実行する．

ASSEMBLER はそれぞれの Iroha node で Iroha のブロックチェーンネットワークである Iroha Network に参加する．Iroha node 上では Blockchain と Blockchain をもとに構成される Onchain-DB が動作しており，これらは全 Iroha node で同期される．Onchain-DB にはルートを自動車とするハッシュ部品木が保管される．

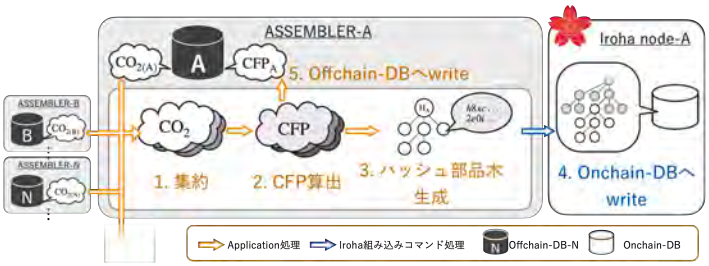


図 6: ハッシュ部品木生成プロセスの流れ

提案システムは 3 つのメインプロセスを持つ．一つ目は，ハッシュ部品木の生成プロセスである．各部品の CFP からハッシュ部品木を生成し，CFP データを検証可能な形で保管するためのプロセスである．二つ目は，ハッシュ部品木の更新プロセスである．任意の ASSEMBLER の CO_2 を更新する場合に，ハッシュ部品木も更新するプロセスである．三つ目は，ハッシュ部品木の検証プロセスである．本研究の挑戦の一つである，Peer to Peer 上でのデータ検証を行うプロセスである．次の節でそれぞれのプロセスの詳細を述べる．

3.3 ハッシュ部品木生成プロセス

ハッシュ部品木生成プロセスは，CFP データを検証可能な形で保管するためのプロセスである．新規部品をシステムに追加する際など初期設定として実施する．処理の流れを図 6 と以下に示す．図中では例として，ASSEMBLER-A で製造される部品 P_A のハッシュ部品木を生成する．

■ 生成プロセス

1. **集約**：サプライチェーンに参加する全 ASSEMBLER の CO_2 を集約.
2. ***CFP* 算出**：P_A の部品木に含まれる部品の CFP を算出.
3. **ハッシュ部品木生成**：定義に従い，P_A のハッシュ部品木のハッシュ値を算出.
4. **Onchain-DB へ write**：iroha の組み込みコマンドで実行．Blockchain にブロックが追加される.
5. **Offchain-DB へ write**：4. が成功すれば，2. の CFP が各 Offchain-DB に格納される．失敗の場合，入力値などにエラーがある可能性がある.

3.4 ハッシュ部品木更新プロセス

任意の ASSEMBLER の CO_2 値が更新された場合，そのASSEMBLER で製造された部品を含む部品木は，対応するハッシュ部品木の更新が必要となる．本プロセスは，すでに Onchain-DB に格納されたハッシュ部品木の中で，更新が必要な部品のみに対して選択的に更新を行うプロセスである．ここで，更新後も CFP データが検証可能である状態は維持される．処理の流れを以下に示す．例として，ASSEMBLER-A で製造される部品 P_A の部品木を更新する．

■ 更新プロセス

1. **パス取得**：P_A からルート部品までのパスを取得.
2. **集約**：サプライチェーンに参加する全 ASSEMBLER の CO_2 を集約.
3. ***CFP* 算出**：1. のパスに含まれる部品の CFP を算出.
4. **ハッシュ算出**：定義に従い，1. のパスに含まれる部品のハッシュ値を算出.
5. **Onchain-DB へ write**：iroha の組み込みコマンドで実行．Blockchain にブロックが追加される.
6. **Offchain-DB へ write**：5. が成功すれば，3. の CFP が各 Offchain-DB に格納される．失敗の場合，入力値などにエラーがある可能性がある.

3.5 ハッシュ部品木検証プロセス

ハッシュ部品木検証プロセスでは，任意の部品木の CFP データの改ざんの有無を検証する．Onchain-DB 上に格納されたハッシュ部品木を用いるため，前提としてハッシュ部品木生成プロセスが 1 度以上実行されている必要がある．また，改ざんは Offchain-DB でのみ発生するとする．

ハッシュ部品木の更新プロセスの流れを以下に示す.

■ 検証プロセス

1. **既存のハッシュ部品木取得**：Onchain-DB から比較対象のハッシュ部品木取得.
2. **新規ハッシュ部品木生成**：生成プロセスと同様にハッシュ部品を生成.
3. **比較**：1. と 2. のハッシュ部品木を比較する．2. のハッシュ値の中で 1. のどのハッシュ値にも一致しない部品が出力される．すなわち 出力がなければ検証成功で終了．出力があれば検証失敗であり，4. に進む.
4. **特定**：改ざん特定処理を実行.

検証プロセス手順 4 の特定では，XOR の自己反転性を活用して改ざん箇所を特定する．自己反転性とは $a \oplus a = 0$ という性質である．図 4 のハッシュ部品木の例を用いて挙動の詳細を解説する.

まず，特定処理の必要性について説明する．P_4 の CFP_4 が Offchain-DB に改ざんが発生し，検証プロセス手順 2 で得られるハッシュ値が，正しくは $H_4 = hash(CFP_4)$ であったところが，$H_4' = hash(CFP_4')$ となったとする．P_4 の親部品である P_1 や，さらにその親部品である P_0 のハッシュ値の算出に改ざん後のハッシュ値 H_4' が使用される．即ち，H_4' は伝搬され，正しいハッシュ値 H_1, H_0 に対し，

改ざん発生後のハッシュ値は H'_1, H'_0 のようになる．

$$H_1 = hash(CFP_1) \oplus H_4 \oplus H_5$$
$$H_0 = hash(CFP_0) \oplus H_1 \oplus H_2 \oplus H_3$$
$$H'_1 = hash(CFP_1) \oplus H'_4 \oplus H_5$$
$$H'_0 = hash(CFP_0) \oplus H'_1 \oplus H_2 \oplus H_3$$

H'_1, H'_0 は H_1, H_0 と一致しないため，検証プロセスの手順 3 では $[P_4, P_1, P_0]$ が出力される．これは P_4 からルート P_0 のパスであり，改ざん部品候補となる．この場合，改ざん部品の候補に対して実際に改ざんされたのは P_4 の CFP_4 のみである．一方で，同じパスを出力しても，パス上の他の部品の CFP に改ざんが発生している可能性もあることから，改ざん発生箇所が P_4 の CFP_4 のみと確定することはできない．例えば，P_4 の CFP'_4 に加えて，P_1 の CFP_1 が改ざんされ CFP''_1 になった場合，詳細は以下のようになる．

$$H''_1 = hash(CFP''_1) \oplus H'_4 \oplus H_5$$
$$H''_0 = hash(CFP_0) \oplus H''_1 \oplus H_2 \oplus H_3$$

H'_1 と H''_1 は区別して改ざんの発生有無を判断しなければならないため，特定処理が必要となる．

続いて，改ざん特定処理の手順を以下に示す．入力には，検証プロセス手順 3 で得られた，改ざん部品候補を表すルート部品へのパスが与えられる．任意の部品 P_N について，本来の正しいハッシュ値を H_N，改ざんの影響があるハッシュ値を H'_N と表す．Onchain-DB に保管されたハッシュ値は全て正しいという前提であり，H_N と表される．手順の説明にあたり，パス $[P_{entry}, \cdots, P_{root}]$ を入力として与え，検証プロセス手順 2 で算出された各部品のハッシュ値は $[H'_{entry}, \cdots, H'_{root}]$ であったとする．

■ 改ざん特定処理

(i) **改ざん候補の確定**：パスの始点である部品 P_{entry} の改ざん発生が確定．

(ii) **XOR 演算による差分計算**：Onchain-DB より正しいハッシュ値 H_{entry} を取得．パス上の各部品のハッシュ値に $H'_{entry} \oplus H_{entry}$ を XOR 演算により適用し，改ざんに由来する差分を算出．

(iii) **再比較による改ざん判定**：ii. のハッシュ値を Onchain-DB に保管された値と照合．一致の場合，i. の改ざん発生部品を返して処理を終了．不一致の場合，新たに改ざん部品候補のパスが得られる．これを入力として i. に戻る．この時，パス上の部品のハッシュ値は ii. で算出した値を利用する．

特定処理の具体的な挙動を，図 4 のハッシュ部品木で P_1, P_4 に改ざんが発生したケースを用いて説明する．まず，検証プロセス手順 3 までを終え，改ざんが発生している可能性のある部品のパス $[P_4, P_1, P_0]$，ハッシュ値 $[H'_4, H'_1, H'_0]$ が得られる．手順 (i) より，P_4 の改ざんが確定する．手順 (ii) では XOR の自己反転性を応用した差分算出が行われ，手順 (iii) では Onchain-DB の高信頼な値との比較が再度行われる．具体的な計算式を以下に示す．

$$xor\_hash_4 = H'_4 \oplus H_4$$
$$H'_4 \oplus xor\_hash_4 = H'_4 \oplus (H'_4 \oplus H_4) = H_4 \quad (1)$$
$$H'_1 \oplus xor\_hash_4$$
$$= hash(CFP'_1) \oplus H'_4 \oplus H_5 \oplus (H'_4 \oplus H_4)$$
$$= hash(CFP'_1) \oplus H_4 \oplus H_5 \quad (2)$$
$$H'_0 \oplus xor\_hash_4$$
$$= hash(CFP_0) \oplus H'_1 \oplus H_2 \oplus H_3 \oplus (H'_4 \oplus H_4)$$
$$= hash(CFP_0) \oplus \{hash(CFP'_1) \oplus H_{on4} \oplus H_5\} \oplus H_2 \oplus H_3 \quad (3)$$

式 (1) は正しいハッシュ値 H_4 と一致するが，式 (2), (3) ではそれぞれ H_1, H_0 と一致せず，新たにパス $[P_1, P_0]$，ハッシュ値 $[H'_1 \oplus xor\_hash_4, H'_0 \oplus xor\_hash_4]$ が得られる．手順 (i) に戻り，P_1 の改ざんが確定する．手順 (ii), (iii) が以下のように行われる．

$$xor\_hash_1 = (H'_1 \oplus xor\_hash_4) \oplus H_1$$
$$= hash(CFP'_1) \oplus hash(CFP_1)$$
$$(H'_1 \oplus xor\_hash_4) \oplus xor\_hash_1$$
$$= hash(CFP'_1) \oplus H_4 \oplus H_5 \oplus (hash(CFP'_1) \oplus hash(CFP_1))$$
$$= hash(CFP_1) \oplus H_4 \oplus H_5 = H_1 \quad (4)$$
$$(H'_0 \oplus xor\_hash_4) \oplus xor\_hash_1$$
$$= hash(CFP_0) \oplus H'_1 \oplus H_2 \oplus H_3 \oplus (H'_1 \oplus H_1)$$
$$= hash(CFP_0) \oplus H_1 \oplus H_2 \oplus H_3 = H_0 \quad (5)$$

式 (4), (5) より手順 (iii) で新たに得られるパスはないことから，P_1, P_4 での改ざんが確定する．

4. 実験

本節では，提案システムの改ざん特定の機能性と性能を調査する．

4.1 実験概要

提案する 3 つのメインプロセスのうち，生成および更新プロセスの手順は，Application 処理と Iroha 組み込みコマンド処理に大別される．具体的には，Onchain-DB への書き込み処理を担うのが Iroha 組み込みコマンド処理であり，それ以外の処理は Application 処理に含まれる．なお，更新プロセスにおいては，Iroha 組み込みコマンドは用いず，Application 処理のみで完結する．

表 1: 実験対象の部品木の高さと総部品数

高さ	2	3	4	5	6	7
総部品数 [個]	6	331	156	781	3,906	19,531

本稿では，Application 処理に対して 2 種類の実装を用意した．1 つ目の実装では，Python のデータ分析ライブラリである Polars を主に用いた．具体的には，各メインプロセスに必要となるデータを，On/Offchain-DB に接続して Polars の DataFrame に読み込んだのちに Python でデータを加工をすることで，Application 処理を実現した．2 つ目の実装では，Python から Onchain-DB に対して SQL クエリを発行することで処理を実現している．具体的には，各メインプロセスに必要となるデータは Onchain-DB の一時テーブルに保管し，Python から一時テーブルへの SQL クエリを発行する．両者の違いは，目的データの算出処理を Polars の DataFrame に対して行うか，PostgreSQL の Table に対して行うか，という点である．

ここで，ハッシュ部品木の構築には，SHA-256 によるハッシュ関数を使用し，16 進数で表現された値に対して XOR を用いることで構成を行った．また，Iroha 組み込みコマンド処理に関しては，C++で記述された既存の組み込みコマンドを用いず，本提案手法に特化した新たなコマンドを開発した．

提案手法の性能を評価するため，3 つの実験を実施した．4.3 節では，2 種類の Application 処理実装に対して実行時間の比較を行い，性能の差異を明らかにした．さらに，4.4 節では検証プロセスの詳細な性能評価を行った．

4.2 実験設定

実装の構築環境とマシンの性能は図 7 に示した．より実運用に近い環境を求める場合は実ネットワークでの評価が妥当であるが，Iroha ネットワークの性能は実ネットワークと仮想ネットワークで大きな差が見られないため [17]，本稿では Docker 環境を採用している．また，参加する ASSEMBLER ごとに Iroha をビルドする Ubuntu 22.04 LTS コンテナと Onchain/Offchain-DB の機能を提供する PostgreSQL コンテナをセットで構築する．本実験では，システムに参加する ASSEMBLER 数を 3 社とした．

検証対象となる部品木の生成には以下の制約を設けた．

- 完全 5 分木．表 1 に示すように，総部品数すなわち木の総ノード数は $\sum_{i=0}^{N} 5^i, \{N = 1, \cdots, 6\}$ となる．
- 部品木中の部品は重複しない．

ここで，本稿では評価実験の対象となる部品木を完全 5 分木で表現したものに絞っているが，実際の部品木はさらに多様であり，アンバランスなものも存在するということに注意されたい．

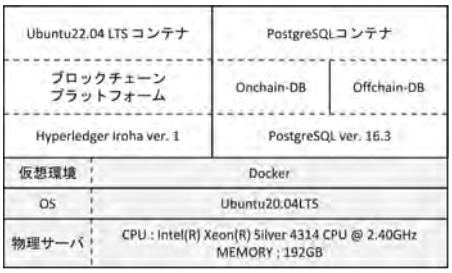


図 7: 構築環境

4.3 各実装の性能調査

4.1 節で述べたように，Application 処理の実装を 2 つのアプローチで行った．どちらの実装がより実用性に優れているか調査する．ここで，Python ライブラリの Polars を主に用いた実装を「Polars メイン実装」，Python から Onchain-DB である PostgreSQL に対して SQL クエリを発行する実装を「PostgreSQL メイン実装」とする．

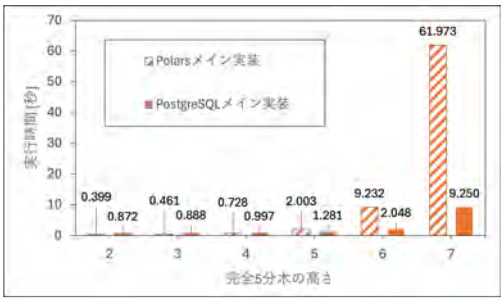
高さ 2-7 の完全 5 分木で表された 6 つの部品木を対象に，2 つの実装で実行時間の計測を行った．各プロセスの条件は次の通りである．

- 生成プロセス：高さ 2-7 の完全 5 分木で表された部品木に対応するハッシュ部品木を生成した．
- 更新プロセス：生成したハッシュ部品木からランダムに選んだ 1 つの葉部品に対して，更新プロセスを実行した．パス長，すなわち 1 回の実行に対する更新部品数はそれぞれ 2-7 となる．
- 検証プロセス：まず生成したハッシュ部品木からランダムに選んだ 1 つの部品の CFP を改ざんした．改ざんされた部品木に対して検証プロセスを実行した．

各プロセスは条件ごとに 5 回測定し，1 回の平均を求めた．結果を図 8 に示した．

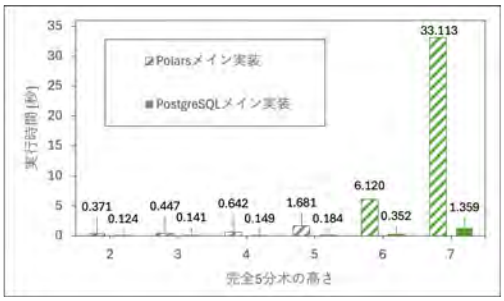
いずれのプロセス，実装においても完全 5 分木の高さが高くなるにつれて実行時間が増加している．これは右に行くにつれてプロセス中で扱うデータ量や処理量も増大することを考慮すれば妥当な結果であり，木の大きさが実行時間に影響を与えることが確認された．また，どちらの実装においても生成プロセスに最も時間を要することがわかった．

実験結果を踏まえ，2 つの実装の比較からより実用性に優れた実装を決定する．図 8(a) より，生成プロセスでは木の大きさが大きくなるにつれて PostgreSQL メイン実装がより高速という結果が得られた．高さ 7 の時に最大で，約 6.70 倍に達している．次に図 8(b) より，生成プロセスでは木の大きさが大きくなるにつれて Polars メイン実装がより高速という結果が得られた．高さ 7 の時に最大で，約 2.0 倍もの差がついている．木の高さに対する速度の関係が生成プロセスとは逆となった．最後に図 8(c) より，どの高さにおいても PostgreSQL メイン実装より高速であり，最大となる高さ 7 では約 24.4 倍の差がついた．ここで，各プロ



(a) 生成プロセスの実行時間の比較

(b) 更新プロセスの実行時間の比較



(c) 検証プロセスの実行時間の比較

図 8: Polars メイン実装と PostgreSQL メイン実装におけるメインプロセスの実行時間

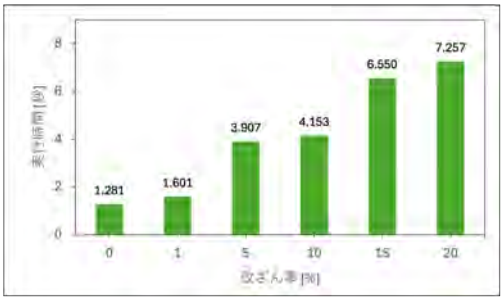


図 9: 改ざん率に対する総部品数 19531 個で構成されたハッシュ部品木の検証プロセス実行時間

セスの想定される実行頻度は，生成プロセスのみ初期設定として一回行えば良いため，更新，検証プロセスに比べて極端に少ない．また，更新プロセスの実行時間は，生成，更新プロセスの実行時間と相対的に比較すると安定して短時間である．以上を考慮すれば，2 つの実装のうち，実用性に優れているのは PostgreSQL メイン実装であると考えられる．よって 4.4 節の実験では PostgreSQL メイン実装を採用した．

表 2: 実験対象の部品木の改ざん率と改ざん部品数

改ざん率 [%]	1	5	10	15	20
総部品数 [個]	195	977	1,953	2,930	3,906

4.4 ハッシュ部品木検証プロセスの性能調査

本節では部品木の改ざん率に対するハッシュ部品木検証プロセスの実行時間を調査した．高さ 7 の完全 5 分木（総部品数 19531 個）を対象とし，全部品のうち 1, 5, 10, 15, 20%の部品をランダムに選んで Offchain-DB 上の CFP を改ざんした．改ざん部品数は表 2 に示した通りである．改ざんされた部品木に対して，ルート部品の検証プロセスを 5 回実行し，1 回の平均を求めた．図 9 に改ざん率 0% で検証成功となる場合と併せて実行結果を示す．

改ざん率が高くなるにつれ，実行時間が長くなることが確認された．これは，検証プロセス手順 4 でのループ回数が増えることが要因にあげられる．また，同時に一度の実行での改ざん部品の特定率を調査したところ，いずれも特定率 100% となり，全ての改ざん部品を検出して期待通りの挙動を得られた．

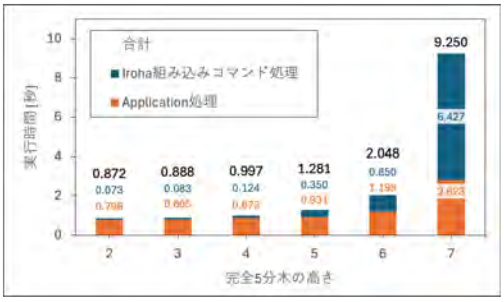
5. 考察

本節では，提案システムの実装についての考察を行う．

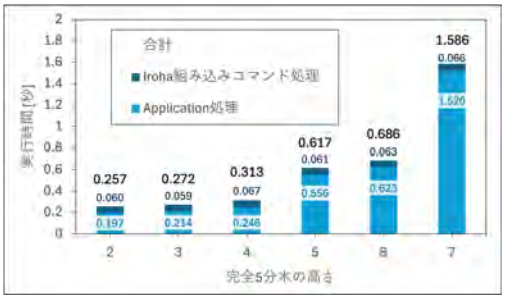
5.1 実験結果に基づく実用性評価

第 4 節の評価実験の結果から議論を行なっていく．ここで，以降の議論では PostgreSQL メイン実装を採用した実験結果のみについて言及している．4.3 節の実験結果より，第一に全プロセスはハッシュ部品木の大きさに影響を受けること，第二にメインプロセスのうち生成プロセスが最も低速であるという結論が得られた．生成プロセスの実行時間が遅くなる要因の調査のため，生成，更新プロセスの Application 処理と組み込みコマンド処理のそれぞれの内訳を図 10 に示した．生成プロセスの図 10(a) を見ると，両処理とも高さが高くなるにつれ処理時間が増加している一方で，総部品数 19531 個では Iroha 組み込みコマンド処理の実行時間が Application 処理の約 2.28 倍もに達した．更新プロセスの図 10(b) では，Application 処理の実行時間が右肩上がりである一方で，Iroha 組み込みコマンド処理は安定して 0.06 秒程度となった．よって，生成プロセスの実行時間が遅くなる要因の一つには Iroha 組み込みコマンド処理が挙げられるとわかる．生成プロセスにおいて，部品木の高さ h に対する Iroha 組み込みコマンドの実行時間 T は以下の近似式で表すことができる．

$$T = 7.96 \times 10^{-6} \times e^{1.943h}$$



(a) 生成プロセス　処理時間の内訳



(b) 更新プロセス　処理時間の内訳

図 10: 4.3 節実験の Application 処理と Iroha 組み込みコマンド処理の内訳

近似式より，Iroha 組み込みコマンド処理の実行時間は木の大きさに対して指数関数的増加が見られるとわかった．一方で，生成プロセスの実行タイミングは，システムに部品を登録する際の初期設定が基本である．一度生成プロセスによって Onchain-DB 上にハッシュ部品木を構築してしまえば，任意の ASSEMBLER-N の $CO_{2(N)}$ に更新があっても更新プロセスを実行して対応できる．よって，生成プロセスが提案システムの実用性に与える影響は低いと考える．

5.2　提案手法の改ざん耐性に関する議論

本稿では XOR を応用しハッシュ部品木を構築することで，改ざん検知・特定を実現した．本節ではこのような設計のハッシュ部品木の耐改ざん性について考察する．

まず，前提としてハッシュ部品木が保管される Onchain-DB は，ブロックチェーンの性能により高い信頼性を持つ．以降の議論は Onchain-DB に保管されたハッシュ値の正当性が常に担保されているという前提で進める．

ハッシュ部品木の構造においては，各ノードのハッシュ値は，子ノードのハッシュ値と自身の CFP 値のハッシュとの XOR により構成される．この構造により，部品木内で CFP 値の改ざんが発生した場合，必ずそのノードおよび上位ノードのハッシュ値に差異が生じるため，改ざん検知が可能となる．特に，本手法では，任意ノードの改ざんはルートまでのパス上の全ノードに影響を与える．改ざんが伝播するパスを追跡することで，改ざん箇所を特定できるほか，パスの中間ノードの CFP が改ざんされた場合も，XOR の不整合から当該ノードにおける異常を検出することができる．

一方で，複数箇所の同時改ざんにより，XOR 結果が偶然一致し，上位ノードでハッシュ値が一致してしまう「打ち消し」の懸念が考えられる．しかし，本構成では CFP 値が実数として表現されており，複数の改ざんを意図的に XOR で打ち消すには高精度の制御が必要であること，Onchain-DB 上のハッシュ値の改ざんが困難であることから，実用上その可能性は極めて低いと考えられる．また，ハッシュ関数には衝突耐性を有する SHA-256 を用いており，異なる CFP から同一ハッシュが生成される可能性も現実的には無視できる．従って，ハッシュ値自体の信頼性も高い．

以上より，提案するハッシュ部品木構造は，XOR を活用することで改ざんの検出だけでなく箇所の特定まで実現可能であり，Onchain-DB に記録されたハッシュ値の正当性が担保されている限り，高い耐改ざん性を有する設計であると評価できる．

5.3　先行研究との比較

先行研究 [6] との比較を行う．先行研究では，部品木内の部品の CFP の算出手法，CFP 改ざんの有無を検知する手法を提案した．これに対し本稿では，より高い安全性と検証機能の高性能化を目指して，ハッシュ部品木を提案しており，CFP の改ざん範囲を特定することができる．実行速度の観点では，本稿第 4 節のハッシュ部品木生成プロセスが最大で 9.3 秒程度かかったが，同条件での先行研究の CFP 算出処理は 0.72 秒である．処理量の増加を考えれば妥当と考えられる．実装の観点では，先行研究の実装の大半は，1 つの Iroha 組み込みコマンド内で完結している．これに対し，本稿の実装では Python を利用した Application 処理を主としている．これにより，組み込みコマンド処理は開発の制約が多いが，Application での実装は自由度が高いことからメンテナンスが容易となった．以上より，本稿の提案システムは，先行研究と比較し，より実用性に優れていると考える．

6.　まとめと今後の課題

本研究では，先行研究で開発した自動車部品の CFP を管理する分散データベースシステムの実用性と安全性を高めるべく，ハッシュ木の構築手法に着想を得たハッシュ部品木を提案した．3 つのメインプロセスとして，ハッシュ部品木の生成・更新・検証プロセスを Python とブロックチェーンプラットフォーム Hyperedger Iroha を用いて実装を行った．提案手法の実装は期待通りの挙動を示している．また，実行時間を調査した実験では，検証・更新・生成プロセスの順で高速となった．生成プロセスは他と比較し低速であったが，想定される実行頻度も低く，システム

の実用性は損なわれないと考える．さらに，検証プロセスの詳細実験も行い，改ざん率の増加に伴い実行時間の増大が観測されたが，いずれも改ざん特定率は 100% となった．以上より，本稿の提案システムは，先行研究と比較し，より実用性に優れていると考える．

今後の課題として，第一に，Onchain-DB に保管したハッシュ部品木の分散を行う．現在の実装では，Onchain-DB は全 Iroha node で冗長であり，ルートを自動車とするハッシュ部品木が管理されている．すなわち，Onchain-DB のハッシュ部品木には，任意の ASSEMBLER の製造部品に関連する部品以外のデータも多く含まれており，不要なデータを保管することによる安全性の懸念，各分散ノードの負荷増加という側面において改善の余地がある．Blockchain より構築される Onchain-DB の耐改ざん性に優れるという利点をより活用するためにも，新たなアプローチの検討・実装を行っていく．第二に，部品の重複の考慮である．本稿では部品は重複なしとして実装，評価実験を行っていたが，実際の自動車には同じ部品が複数個使われている．重複部品はメモ化のように値を保管しておくことで，更なる高速化が望めるため，これについても挑戦していく．

謝辞

本研究は一部，JST CREST JPMJCR22M2 の支援を受けたものである．

参考文献

[1] CREST : 検証可能なデータエコシステム. 入手先 ⟨https://www.kde.cs.tsukuba.ac.jp/crest/index.html⟩ (参照 2024-12-30).

[2] Catena–x. 入手先 ⟨https://catena-x.net/en/1⟩ (参照 2025-1-6).

[3] Mobi. 入手先 ⟨https://dlt.mobi/⟩ (参照 2025-1-6).

[4] 経済産業省. Ouranos ecosystem(ウラノス・エコシステム). 入手先 ⟨https://www.meti.go.jp/policy/mono_info_service/digital_architecture/ouranos.html⟩ (参照 2024-12-29).

[5] 経済産業省デジタルアーキテクチャ・デザインセンター (DADC). Ouranos ecosystem サプライチェーン上のデータ連携の仕組みに関するガイドライン (蓄電池 cfp·dd 関係). ⟨https://www.ipa.go.jp/digital/architecture/guidelines/scdata-guidline.html⟩(参照 2025-1-7).

[6] 堀遥, Le Hieu Hanh, 小口正人. ブロックチェーンベースのカーボンフットプリントデータ管理基盤の構築と自動車部品製造業への応用における評価. In *DEIM2025 8C-01*, 2025.

[7] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[8] Bitcoin. 入手先 ⟨https://bitcoin.org/ja/⟩ (参照 2024-12-30).

[9] Ethereum. 入手先 ⟨https://ethereum.org/ja/⟩ (参照 2024-12-30).

[10] Hyperledger foundation project iroha. 入手先 ⟨https://www.hyperledger.org/projects/iroha⟩ (参照 2024-1230).

[11] Hyperledger foundation. 入手先 ⟨https://www.hyperledger.org/⟩ (参照 2024-12-30).

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

[13] N. Mishra and H. Levkowitz. Pdv: Permissioned blockchain based personal data vault using predictive prefetching. In *BIOTC '21: Proceedings of the 2021 3rd Blockchain and Internet of Things Conference*, pp. 59–69, 2021.

[14] G. Zyskind, O. Nathan, and A. S. Pentland. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pp. 180–184, 2015.

[15] 萱原正彬, 本田祐一, 山田達大, Le Hieu Hanh, 串間宗夫, 小川泰右, 松尾亮輔, 山崎友義, 荒木賢二, 横田治夫. ブロックチェーンとプロキシ再暗号化を用いた共有範囲設定可能な医療情報管理. In *DEIM Forum 2019*, 2019.

[16] N. B. Truong, K. Sun, and Y. Guo. Blockchain-based personal data management: From fiction to solution. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8, 2019.

[17] 坂本明穂, 小口正人. 実ネットワークで接続されたデータ検証可能な分散データベースの性能に関する検討. マルチメディア，分散，協調とモバイル (DICOMO2024) シンポジウム, 2024.