修士論文

ブロックチェーンを用いた
ソフトウェアビルドプロセス記録手法の提案

吉上 康平

2019 年 1 月 31 日

奈良先端科学技術大学院大学
情報科学研究科

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

吉上 康平

審査委員：

松本 健一 教授　（主指導教員）
安元 慶一 教授　（副指導教員）
石尾 隆 准教授　（副指導教員）
畑 秀明 助教　（副指導教員）

ブロックチェーンを用いた ソフトウェアビルドプロセス記録手法の提案*

吉上 康平

内容梗概

ソフトウェア・トレーサビリティはソフトウェアの品質を向上させる要素の１つである．特にビルドプロセスにおけるソフトウェア・トレーサビリティを確保し，複数組織間で共有することは，ライセンス遵守の証明や事後的な紛争解決を行う際の証拠として重要である．ソフトウェアは複製・改ざんが容易なため，技術の漏えいや悪意ある改ざんの可能性に常に晒されている．したがって，共有するソフトウェア情報を必要最低限にしながら，共有した情報に改ざんがないことを証明する必要がある．本研究では，ビルドプロセスを自動的に監視し，ビルドに使用されたファイル，およびビルドによって生成されたファイルの関係をブロックチェーン上に記録することで，複数組織間のソフトウェア情報の共有を，改ざんの恐れなく，検証可能な形で実現するための手法を提案する．また，提案手法の実現可能性を評価するために，Ethereum 上でスマートコントラクトとして実装し，ビルドプロセスの記録に要する時間，データ量，ブロックチェーンシステムの手数料を計測した結果から，現実的なコストで提案手法が実現可能であることを示した．

キーワード

ソフトウェア・トレーサビリティ，ブロックチェーン，情報共有，ビルドプロセス，複数組織間

*奈良先端科学技術大学院大学 情報科学研究科 修士論文, 2019 年 1 月 31 日.

Blockchain-based logger for Software Traceability on Build Process*

Kohei Yoshigami

Abstract

Software traceability is an element that improves the software quality. In particular, securing software traceability in the build process and sharing it among multiple organizations is important in proof of license compliance and post-dispute settlement. Since software can be easily duplicated and tampered with, there is a possibility of technology leakage or malicious tampering. Therefore, it is necessary to prove that there is no tampering with the shared information while minimizing the shared software information. In this research, the build process is automatically monitored, and the relationship between the file used for the build and the file generated by the build is recorded on the block chain. We propose a method to realize sharing of software information among multiple organizations in a verifiable manner without fear of tampering. In order to evaluate the feasibility of the proposed method, it was implemented as a Smart Contract on Ethereum. In the experiment, we measured the time required for recording the build process, the amount of data, the fee of the block chain system, and showed that the proposed method can be implemented at realistic cost.

Keywords:

software traceability, Blockchain, infomation sharing, build process, multiple organizations

*Master's Thesis, Graduate School of Information Science, Nara Institute of Science and Technology, January 31, 2019.

目 次

図 目 次

表 目 次

1. はじめに

ソフトウェアは日常生活に広く普及しており，高品質なソフトウェアを作成する必要性がますます高まっている．ソフトウェア製品の品質を向上させる要素として，ソフトウェア・トレーサビリティが存在する．Center of Excellence for Software & Systems Traceability（CoEST）によると，ソフトウェア・トレーサビリティとは「ソフトウェア製品を構成する一意に識別可能な人工物を相互に関連付け，必要なリンクを維持し，それらのネットワークを使用して，ソフトウェア製品とその開発プロセスの両方の質問に答える能力」と定義されている[1]．ソフトウェア開発において，ソフトウェアの構成部品，例えばソースコード，要求仕様書や設計書などは頻繁に変更・更新されるが，ソフトウェア・トレーサビリティが確保されている場合には，1 つの変更・更新が他の構成部品に与える影響を特定することが容易になる．また，人命や財産，環境などに損害を与えるような欠陥が発覚した際の原因の特定や責任の所在を明確にする証拠として利用できる [6].

従来，ソフトウェア・トレーサビリティは単一組織内でプロセス改善や公的な認証を取得するために活用されてきた．近年では, 人命に関わるような自動運転や医療情報といった複数組織間での情報共有が重要となる分野で，ソフトウェア・トレーサビリティを担保するための情報（以降，ソフトウェア情報）を共有する必要性が指摘されている [8]．しかし，複数組織間において，正確なソフトウェア情報を共有することは容易ではない．ソフトウェアの構成部品はデータの集合であるため，容易に複製可能である．そのため，立場の異なる複数の組織間では，利害関係の衝突から自組織の全ての情報を共有することが出来ない. 加えて，全ての情報が共有されないため，提供された情報が正確で改ざんされていないことを検証することが困難である．したがって，複数組織間で共有されたソフトウェア情報が正確で改ざんされていないことを，それぞれの組織に依存することなく，システムとして保証することが求められる．

CoEST の定義から，ソフトウェア情報は「ソフトウェア製品の構成部品」と「人工物間の関連性を定義するリンク」から構成される．しかし，組織内に大量

[1] http://www.coest.org/

に蓄積された人工物間のリンクを手作業で作成することは，コストが大きく，投資効果が小さいことが指摘されている [6]．そのため，従来研究の多くは，組織内部に蓄積された情報群から，自動的に関連リンクを生成する技術を開発してきた [2] [3]．しかし， これらの技術は主として単一の企業内部でのトレーサビリティの利用を前提としており，複数組織間における共有については想定していないと考えられる．

他方，食品や医療情報の分野ではブロックチェーンを用いてトレーサビリティを確保する試みが活発に行われている．ブロックチェーンは，信頼の拠り所となる第三者機関なしに，任意の 2 者間の電子的なやりとりの存在を保証するための仕組みである [9]．ブロックチェーンが持つ特性として，非中央集権性（特定の管理者に依存しないこと），透明性（取引の履歴が誰にも公平に公開されること），不変性（記録内容の変更・改ざんが困難であること）がある．これらの特性は，国際送金や決済など，複数の立場の異なる組織が情報を頻繁にやりとりする必要がある分野において有効性が認められている [12]．ブロックチェーンの特性を利用することで，どの組織にも依存せず，どの組織からも確認できる改ざん不可能なソフトウェア情報の共有が可能になると考えられる．

本研究では，特にビルドプロセスを対象とし，あるソフトウェアがどのような構成部品群から構成されているかというソフトウェア・トレーサビリティに着目する．ビルドとは，ソースコード群に対してコンパイルを行ったり，ライブラリのリンクを行うことによって最終的な実行可能ファイルを作成することであり，その工程がビルドプロセスである．ビルドプロセスのソフトウェア・トレーサビリティが確保できると，ライセンス遵守の証明 [10] や事後的な紛争解決のための証拠 [1] として用いることができる．

本研究の目的は，ビルドプロセスにおけるソフトウェア情報を，複数組織間で共有し，事後的な検証を可能とする環境を構築することである．目的達成のためのアプローチとして，ハッシュ化したビルドプロセスにおけるソフトウェア情報をブロックチェーン上に記録することで，複数組織間およびソフトウェア製品ユーザに向けたソフトウェア情報の共有を，改ざんの恐れなく，検証可能な形で実現するためのフレームワークを提案する．

本論文の構成は次のとおりである．続く 2 章で，過去における関連研究について述べる．3 章で，本研究で想定するユースケースについて述べる．4 章で，本研究で提案する手法について述べる．5 章で，本研究で行った実験について述べる．6 章で，本論文の結果を補強する考察を述べる．最後に 7 章で，本論文のまとめを述べる．

2. 関連研究

2.1 ソフトウェアトレーサビリティ

ソフトウェア・トレーサビリティはソフトウェアが持つ重要な品質の 1 つであるが，その担保には多大な時間的・金銭的なコストが必要となる [5]．そのため，これまでソフトウェア・トレーサビリティの確保にかかるコストを軽減するための手法が多数提案されてきた．Asuncion らはトピックモデリングを用いてソフトウェア製品のユースケース，設計書，およびソースコード間のリンクを自動生成することを提案している [3]．また Ali らは，SVN/Git で管理されたレポジトリのマイニングから，ソフトウェア製品の要件と，それぞれの実装ソースコード間のリンクを推定する手法を開発している [2]．しかし，これらの技術は主として単一組織内でのトレーサビリティの確保支援を目的としており，複数組織間における共有については想定していない．

実社会におけるソフトウェア・トレーサビリティ確保の取り組みとして，規格化した複数のソフトウェアメトリクスの組み合わせを，開発期間全体を通して継続的に計測し，その結果を利用者に開示することが行われてきた．例えば，StagE project では，41 のソフトウェア情報をまとめたものを「ソフトウェアタグ」と定義し，プロジェクトの成果物にソフトウェアタグを付与した上でユーザーに届けることを提案している [11]．しかし，この技術は共有する情報の内容を策定したものであり，生成したソフトウェア情報を共有する方法や，共有されたソフトウェア情報が改ざんされていることは想定していない．

2.2 ブロックチェーン

ブロックチェーンは分散型台帳技術であり Bitcoin の中核技術として開発された．Bitcoin は，信頼の拠り所となる第三者機関なしに，任意の 2 者間で決済処理を行うことを目的として作られたシステムである．[9]. ブロックチェーンの構成要素は,（i） Peer-to-Peer ネットワーク上 の任意の 2 点間のトランザクションの全履歴を，各ネットワーク参加者（ノード）が個別に保有するという構造，および,（ii）各ノードが持っているトランザクションの全履歴を，特定の管理者に依存せずに，全体で同期しながら更新するためのプロトコルの 2 つである．特に Bitcoin における（ii）のプロトコルには，Proof of Work（PoW）と呼ばれる方式が採用されている [9].

ブロックチェーンは，P2P ネットワークで発生した複数のトランザクションを 1 つにまとめて「ブロック」とし，それらを時系列で繋げていくことで，トランザクションを記録する．PoW の仕組みでは，各ブロックにその内容を証明するためのハッシュ 値と，計算に大きな手間のかかる乱数列（nonce）が付与される．ある時点で記録されたブロックのハッシュ値は，1 つ前のブロックのハッシュ値に依存するように構成されるため，あるブロックの内容が改ざんされると，直ちにそれ以降のブ ロックとの整合性が失われる．加えて，ハッシュ値の整合性が失われると，各ブロックに付与された nonce も整合性を失うが，nonce の再計算には実現困難なほど大きな計算能力が必要となるため，現実的には記録内容の改ざんは不可能となる．

ブロックチェーンは，その非中央集権性，透明性，および改ざん耐性の高さから，送金や決済以外の分野でも活用が進められている．特に，立場の異なる複数の組織が共同で情報を共有・蓄積するための仕組みとして，医療情報やサプライチェーン管理の分野での活用が積極的に検討されている [12]．ソフトウェア工学分野においても，例えばブロックチェーンベースシステムの脆弱性分析 [7] やブロックチェーンを用いた Web サービスの無停止化 [4] などで，ブロックチェーンの活用・分析が始まっている．

医療情報や食品情報と比較して，ソフトウェアは次の 2 つの特徴を持つと考えられる．

（1）ソフトウェア製品は多数（数百-数万以上）の人工物の集合から構築される．

（2）ソフトウェア製品を構成する人工物（特にソースコード）はコピーや流用のリスクが高い．

これらのソフトウェア固有の特徴は，医療情報や食品を対象にした トレーサビリティ確保の手法を，そのままソフトウェア製品に適用することを困難にすると考えられる．本研究では，情報の共有方法として利便性の高いブロックチェーンを，ソフトウェア固有の特徴を考慮しつつ適用することで，複数組織間でどの組織にも依存せず，改ざんのないソフトウェア情報の共有を可能にする．

3. 想定するユースケース

複数組織間でのソフトウェア開発には様々な形態が存在する．ここで想定するケースとして，注文組織（クライアント）が請負組織（ベンダー）にソフトウェア開発を委託する場合を考える．図1は，想定するユースケースの全体像である．開発後もベンダーがメンテナンスを行うために，ソフトウェアの著作権はベンダーに帰属するものとする．クライアントはソフトウェアの実行ファイルをベンダーから受け取り，ソフトウェアを利用する．ベンダーはクライアントとの間で，ソフトウェア・トレーサビリティ確保のために，実行ファイルの納品ごとにソフトウェア情報の共有を行うことが義務付けられている．開発終了後の保守期間においても，修正ごとにソフトウェア情報の共有を行う．

想定するユースケースでは，ベンダーにほぼ全ての情報が集約されており，クライアントの元には実行ファイルと共有されたソフトウェア情報のみが存在している．まず，このケースでクライアントには以下のようなリスクが生じる．

- ソフトウェアの不具合などから事後的な検証が必要となった場合，ベンダーは共有したソフトウェア情報に合致するように証拠の隠滅や改ざんを行う恐れがある．
- 事前のソフトウェア情報の共有時に，ベンダーが問題の起こりそうな部分を排除した虚偽のソフトウェア情報を共有する恐れがある．

一方で，このケースにおいてベンダーには以下のようなリスクが生じる．

- ソースコードやライブラリをソフトウェア情報として共有した場合，クライアントによって複製され再利用されてしまう恐れがある．

このケースでは，クライアントとベンダーは利害関係が対立している．クライアント側は共有された情報が信用できないという点が，ベンダー側は共有を行うソフトウェア情報をなるべく限定したいという点がソフトウェア情報を共有する際のボトルネックになっている．解決策として，第三者機関にソフトウェア情報を保管する方法が考えられるが，開発費用の増大や情報漏洩のリスクが増大することから現実的な解決策ではない．

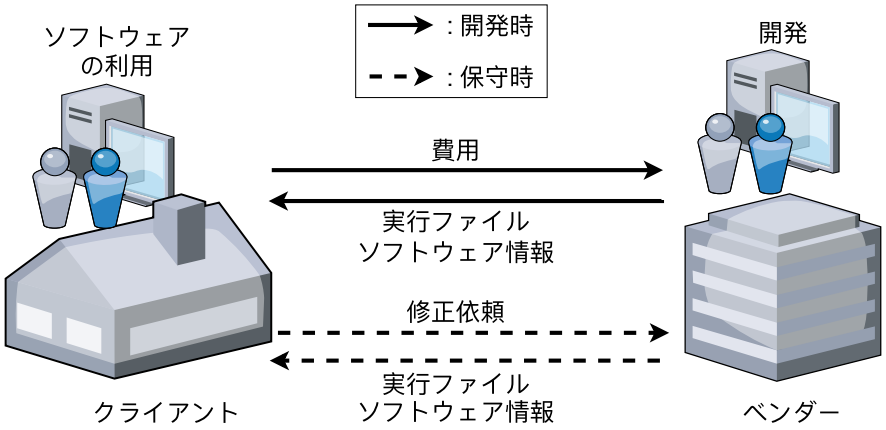

図 1 想定するユースケースの全体像

4. 提案手法

本研究の目的は，ビルドプロセスにおけるソフトウェア情報を共有することである．また，3 章の内容から，以下の条件を満たす必要がある．

- 共有するソフトウェア情報は最低限に限定されている．
- 共有された情報が改ざんされていないことが証明されている．

本研究の提案手法は，上記の条件を満たして，ブロックチェーンを用いてビルドプロセスにおけるソフトウェア情報の記録と共有を行う手法を提案する．提案手法はソフトウェア情報の記録と共有を行う開発者が，ビルド可能な状態であることが前提である．提案手法は，あるビルドコマンドが与えられているとき，ビルドプロセスに実際に用いられた入出力ファイルを特定し，それらの情報を記録することができる．提案手法の具体的な手順は以下のとおりである．

（Step 1） ビルドプロセスの監視

ビルドプロセスにおいて用いた入力ファイル，ビルドプロセスにおいて生成された出力ファイル，ビルドプロセスに用いたコマンドを抽出する．

（Step 2） 入出力ファイルの保管

入力ファイル，出力ファイル，コマンドのバックアップを作成する．

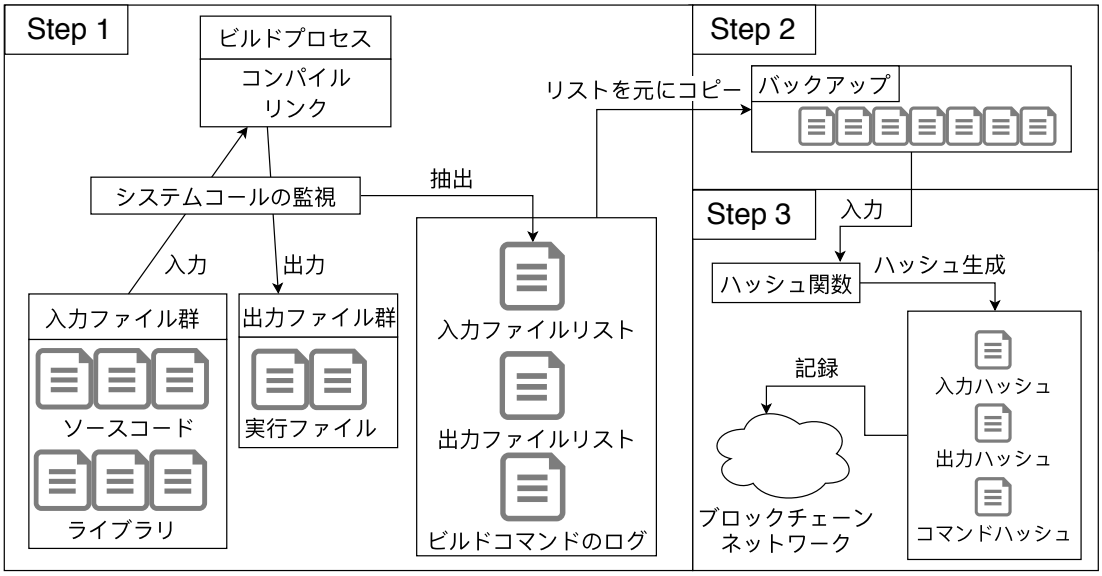

図 2 提案手法の全体像

（Step 3） ブロックチェーンへのビルドプロセスにおけるソフトウェア情報の記録

入力ファイル，出力ファイル，コマンドのハッシュ値を求める．導出したハッシュ値を入力，出力，コマンドに 1 つづつ代表するハッシュ値ができるように再計算し，合計 3 つのハッシュ値を生成する．得られた入力ファイル，出力ファイル，コマンドのハッシュ値をブロックチェーン上に記録する．

図 2 は提案手法の全体像である．

提案手法は，ビルドプロセスにおけるソフトウェア情報を不可逆な値であるハッシュ値に変換することによって，共有するソフトウェア情報を最低限に限定している．また，不変性の特徴を持つブロックチェーン上にハッシュ値を記録することで事後的な改ざんを不可能にしている．

この提案手法によって，作成された実行ファイルはある時点においてあるソース群から作成されたことが証明できる．そのため，ライセンス遵守の証明や事後的な紛争解決のための証拠として用いることができる．

次節以降で手順ごとの詳細を説明する．

4.1 (Step 1) ビルドプロセスの監視

本節では，ビルドプロセスを監視し，ビルドプロセスで用いた入力ファイル，ビルドプロセスにおいて生成された出力ファイル，ビルドプロセスに用いたコマンドを抽出する．ビルドプロセスとはコンパイルやライブラリのリンクなどを行うことで最終的な実行ファイルを作成する工程のことである．入力ファイルとしては，ソースファイル，ライブラリのヘッダファイル，ライブラリのバイナリファイル，設定ファイルなどがある．出力ファイルとしては，実行ファイルやデバッグ用のメモリマップファイルなどがある．どのファイルが実際にビルドプロセスで使用されたかを，ビルドプロセスを監視して抽出する．

コンパイルやリンクを行う際，コンピュータはシステムコールと呼ばれる機構を用いて，OS の中核であるカーネルの機能を使用する．中でも，open や openat というシステムコールは，ファイルの入出力を行うため，ビルドプロセスにおいてこのシステムコールの実行を監視すると，入出力ファイルの抽出が可能となる．Linux ではプログラムが実行するシステムコールの監視を行うことのできる strace という機能が提供されている．strace は一般的に，作成したプログラムが意図した挙動とは異なる挙動をするなど，デバッグの用途で用いられる．strace だけではなく，Linux，Free BSD，Mac OS X，Windows などのほとんどの OS でシステムコールをトレースするために必要なツールが存在する．提案手法では，strace を用いてビルドプロセスにおける入出力ファイルの抽出を行う．

提案手法で用いる strace コマンド例を示す．

% strace -ff -e trace=open,openat -o @filePath @buildcmd

strace の-ff オプションはビルドプロセスにおいて，子プロセスも監視対象として，出力を行うためのオプションである．strace の-e オプションは，ビルドプロセスにおいて，どのようなシステムコールをトレースするかを指定することができるオプションである．本実験ではファイルの入出力に関わる open または openat のシステムコールを出力している．strace の-o オプションは，システムコールのトレースを行った結果を出力することができるオプションである．@filePath は結果を出力するファイルパスを表す．-ff オプションと組み合わせることによって，プロセスごとにシステムコールをトレースした結果を出力することが可能となる．

```
open("/dev/urandom", O_RDONLY)          = 3
open("/usr/bin/dpkg-buildpackage", O_RDONLY) = 3
open("/etc/perl/sitecustomize.pl", O_RDONLY) = 4
open("/usr/share/perl/5.24/strict.pm", O_RDONLY) = 4
open("/usr/share/perl/5.24/warnings.pm", O_RDONLY) = 4
open("/usr/lib/x86_64-linux-gnu/perl/5.24/Cwd.pm", O_RDONLY) = 4
open("/usr/share/perl/5.24/Exporter.pm", O_RDONLY) = 5
open("/usr/share/perl/5.24/vars.pm", O_RDONLY) = 5
open("/usr/share/perl/5.24/warnings/register.pm", O_RDONLY) = 6
open("/usr/share/perl/5.24/XSLoader.pm", O_RDONLY) = 4
open("/usr/lib/x86_64-linux-gnu/perl/5.24/auto/Cwd/Cwd.so", O_RDONLY|O_CLOEXEC) = 4
open("/usr/share/perl/5.24/File/Temp.pm", O_RDONLY) = 4
open("/usr/share/perl/5.24/Carp.pm", O_RDONLY) = 5
open("/usr/lib/x86_64-linux-gnu/perl/5.24/File/Spec.pm", O_RDONLY) = 5
open("/usr/lib/x86_64-linux-gnu/perl/5.24/File/Spec/Unix.pm", O_RDONLY) = 5
open("/usr/share/perl/5.24/constant.pm", O_RDONLY) = 6
open("/usr/share/perl/5.24/File/Path.pm", O_RDONLY) = 5
open("/usr/share/perl/5.24/File/Basename.pm", O_RDONLY) = 6
open("/usr/lib/x86_64-linux-gnu/perl/5.24/Fcntl.pm", O_RDONLY) = 5
open("/usr/lib/x86_64-linux-gnu/perl/5.24/auto/Fcntl/Fcntl.so", O_RDONLY|O_CLOEXEC) = 5
open("/usr/lib/x86_64-linux-gnu/perl/5.24/IO/Seekable.pm", O_RDONLY) = 5
open("/usr/lib/x86_64-linux-gnu/perl/5.24/IO/Handle.pm", O_RDONLY) = 6
open("/usr/share/perl/5.24/Symbol.pm", O_RDONLY) = 7
open("/usr/share/perl/5.24/SelectSaver.pm", O_RDONLY) = 7
open("/usr/lib/x86_64-linux-gnu/perl/5.24/IO.pm", O_RDONLY) = 7
open("/usr/lib/x86_64-linux-gnu/perl/5.24/auto/IO/IO.so", O_RDONLY|O_CLOEXEC) = 7
open("/usr/lib/x86_64-linux-gnu/perl/5.24/Errno.pm", O_RDONLY) = 5
open("/usr/lib/x86_64-linux-gnu/perl5/5.24/Scalar/Util.pm", O_RDONLY) = 5
open("/usr/lib/x86_64-linux-gnu/perl5/5.24/List/Util.pm", O_RDONLY) = 5
open("/usr/lib/x86_64-linux-gnu/perl5/5.24/auto/List/Util/Util.so", O_RDONLY|O_CLOEXEC) = 5
```

図 3　strace コマンド実行後の出力ファイルの例

@outputFilePath は出力を行いたいファイルパスを表す．@buildcmd はシステムコールのトレースを行いたいビルドプロセスに必要なコマンドを表す．本実験で用いたビルドプロセスに必要なコマンドの例を示す．

% dpkg-buildpackage -us -uc -b

dpkg-buildpackage は Debian パッケージのビルドを自動化するプログラムである．-us -uc オプションでビルドプロセスにおいて，署名を行わないように指定している．-b オプションでビルドプロセスにおいて，バイナリのみをビルドするように指定している．図 3 は strace コマンドを実行した際の出力ファイルの例である．

上記の strace コマンドを実行すると，ビルドプロセスで使用した入力ファイルやビルドプロセスによって生成された出力ファイルの一覧が，ログファイルとして生成される．生成されたログファイルから入力ファイルパスと出力ファイルパスを取得し，入力と出力に対して，それぞれ 1 つづつのテキストファイルに結合し，出力する．入力ファイルと出力ファイルの判断基準は，システムコールのフ

ラグ部分に “O_CREAT” という記述があれば出力，なければ入力として分類する．また，strace コマンド実行時に記述するビルドコマンドを自動的に収集し，テキストファイルに出力する．

4.2 (Step 2) 入出力ファイルの保管

本節では，4.1 節で出力した，入出力ファイルのリストを元に，ビルドを行った時点の入出力ファイル群のバックアップを作成する．まず，4.1 節で出力した入出力ファイルリストには，ビルドプロセスにおける中間生成ファイルなど最終的には存在しないファイルが混入しているため，ファイルの存在確認を行う．入出力ファイルリストを元にファイルの存在確認を行い，存在していたファイルのパスのみをテキストファイルに出力する．本提案手法では，事後的な検証を行うため，入出力ファイルのディレクトリ構成を保ったままバックアップを作成する必要がある．つまり，別途ファイルを保管するための容量を確保する必要がある．よく用いられる cp コマンドの parents オプションを利用することによって，本節で出力したファイルの存在確認済み入出力ファイルパスのリストを元にディレクトリ構成を保ったままコピーを行い，新しく作成したディレクトリ内に保管する．

4.3 (Step 3) ブロックチェーンへのビルドプロセスにおけるソフトウェア情報の記録

本節では，4.2 節で保管した入出力ファイルを元にソフトウェア情報を生成し，ブロックチェーン上へ記録する．単純な方法としては，出力された全てのファイルを記録することが考えられる．しかし，全てのファイルをブロックチェーン上に記録することは現実的ではない．ブロックチェーンには透明性という特徴があり，記録したデータは全て公開される．つまり，共有するソフトウェア情報は最低限に限定されているという制約条件を満たさなくなる．加えて，ブロックチェーン上でトランザクションを発生させるためには手数料が必要となる．ビルドプロセスにおいて入出力ファイルは多数存在するため，それら全ての情報をブロックチェーン上に記録すると多大な金銭的コストが発生する．

そこで，本提案手法では，入出力ファイルに対してハッシュ化を行う．ハッシュ化とは，データをアルゴリズムに従ってハッシュ値と呼ばれるランダムな値に不可逆変換することである．本提案手法では，Bitcoin にも採用 [9] され，現在も有効な攻撃が発見されていない SHA-256 というアルゴリズムを採用している．ハッシュ化を行うことで，ハッシュ値がどのようなデータから生成されたかを知っている者以外には無意味なランダム値となる．また，入力するデータが少しでも違えば，出力されるハッシュ値は異なる値となるため，ハッシュ値から元のデータを推測することは現状では不可能である．事後的な検証を行う際，ハッシュ値を比較することでファイルの改ざんの有無を容易に検知することができる．4.2 節で保管した入出力ファイルを元に全ての入力ファイル，出力ファイルにハッシュ化を行い，ハッシュ値と元となったファイルパスをテキストファイルに出力する．図 4 は，ハッシュ化を行なった後の出力ファイルの一部である．

次に，共有するソフトウェア情報は最低限に限定されているという制約条件を満たしつつ，ブロックチェーン上に記録を行う際の手数料を低下させるために，入力，出力，コマンドを代表するハッシュ値を計算する．具体的には，ハッシュ化において出力された図 4 のようなテキストファイルを元にハッシュ値のみのテキストファイルを作成する．ビルドプロセスの実行順序による差異をなくすためにハッシュ値のソートを行う．また，複数回同じファイルを参照している場合があるため，重複しているハッシュ値の重複を取り除いた．入力，出力，コマンドそれぞれのハッシュが記されたテキストファイルに対して，ソートを行い重複を取り除いたハッシュ値のみのテキストファイルをハッシュ化し，得られたハッシュ値をそれぞれの代表ハッシュ値とした．図 5 は，入力，出力，コマンドの代表ハッシュ値の出力例である．

最後に，代表ハッシュ値をブロックチェーン上に記録する．本提案手法では，Ethereum というブロックチェーンネットワークを使用する．Ethereum は，スマートコントラクトと呼ばれる，ブロックチェーン上で動作するプログラムを構築することができる公開型のブロックチェーンネットワークである．スマートコントラクトは，関数を定義することができ，内部にストレージを持つため，ハッシュ値を配列に格納するような関数を定義することで，ブロックチェーン上へのソフ

トウェア情報の記録と共有が可能となる．付録 A は，本提案手法で用いた，ハッシュ値を記録するためのスマートコントラクトのソースコードである．入力，出力，コマンドの代表ハッシュを格納可能な配列を持ち，それぞれの変数に値を代入するための関数を定義している．また現在の配列の長さを確認するための関数と配列の中身を確認するための関数も定義している．スマートコントラクトのデプロイや実行は Ethereum Foundation の Web ページ[2]を参照されたい．

Ethereum は Web ブラウザ上でブロックチェーン上の情報を確認できる Etherscan.io[3]を提供している．スマートコントラクトをブロックチェーン上へデプロイすると，アドレスが割り振られる．割り振られたスマートコントラクトのアドレスを知っていれば，Web ブラウザ上で必要な情報を得ることが可能となる．つまり，ソフトウェア情報を確認する組織は，ブロックチェーンネットワークに参加していなくとも必要な情報を確認することが可能である．ブロックチェーン上にデプロイされているスマートコントラクトの，ストレージを変更する関数を実行すると，手数料が発生する．手数料は Ethereum 内の通貨である Ether（2019 年 1 月 31 日 3:50 UTC 時点 1Ether 11,832.10 円）で支払う．手数料を支払うとトランザクションが発生し，ブロックチェーン上で承認されるとスマートコントラクトの状態が変化する．図 6 は，付録 A のスマートコントラクトをデプロイした後に recordHash 関数を実行した際に発生したトランザクションを Etherscan で表示した例である．トランザクションが承認された時刻を示すタイムスタンプや入力として与えられたデータなどを確認できる．一度承認されたトランザクションは改ざんすることが不可能なため，共有された情報が改ざんされていないことが証明されているという制約条件を満たしている．

[2]https://www.ethereum.org/greeter
[3]https://etherscan.io/

```
0b326c6b13496e1ba10058597e3679a318d90484ef506400d52f324cdc460a67 : /usr/lib/locale/ja_JP.utf8/LC_COLLATE
b8e2e9289cfef8e9b301dd63ba0e08960cfd49a967387727f42e40023f9da3f2 : /usr/lib/locale/ja_JP.utf8/LC_TIME
5172617c05a37b20bf980ca047b35da4ccc281be9672df40b267dbc0a7d69c09 : /usr/lib/locale/ja_JP.utf8/LC_NUMERIC
ededfbc80c0abcd252105d9a087aa4e89deacd9b2d1102cb2dc7d8cc0cdc353c : /usr/lib/locale/ja_JP.utf8/LC_CTYPE
08111990c3c4513778b48b18ac4509899c876b26eac4071ed223b67d152db421 : /usr/bin/dh_shlibdeps
a36dbd07ce948e3ba32c6a5431a58836c9eb3255d73aa3d6c7846dd16a53f827 : /etc/perl/sitecustomize.pl
3f100533c2abc1cd0b20060bfe71372c28443b13356fd5604133a69e844fc88b : /usr/share/perl/5.24/strict.pm
6677ba2321f7c3135854ca6f6b0d8bcaaac3fcf72dd182058a03d83705f457d8 : /usr/share/perl/5.24/warnings.pm
0e707f6c9273736f137eed9710aa2eec7edf582f86813abc8820fb982c4bf101 : /usr/lib/x86_64-linux-gnu/perl/5.24/Cwd.pm
b9db7ff262f5d5bcc8ac7c827b0453c7e514c3cb1dd6a30fad96a25de2a76991 : /usr/share/perl/5.24/Exporter.pm
665260cd8f2ce2ae717e29e87c8f5499b7154ac6f20a4bb8c222ee4456639aa5 : /usr/share/perl/5.24/vars.pm
962110d3faf3e55749a260d97edaf8a9d31293b0f33d2e7771663a9ef364aa94 : /usr/share/perl/5.24/warnings/register.pm
```

図 4　ハッシュ化を行なった後の出力ファイルの例

```
inputHash   : 391e357e7590e357d31e0defab973f146dd8a79a9db2360d50630fd3e2719b80
outputHash  : abe1470c4287a1ee5c66099af31c10eba6c06f445df93eaa6ecaf4bd5c9fa4bc
commandHash : 8a7c860d104d828dfb8edcc7dd5f544a2c3e6f8fe9d419e6b8879291c3e3277e
```

図 5　入力，出力，コマンドの代表ハッシュ値の出力例

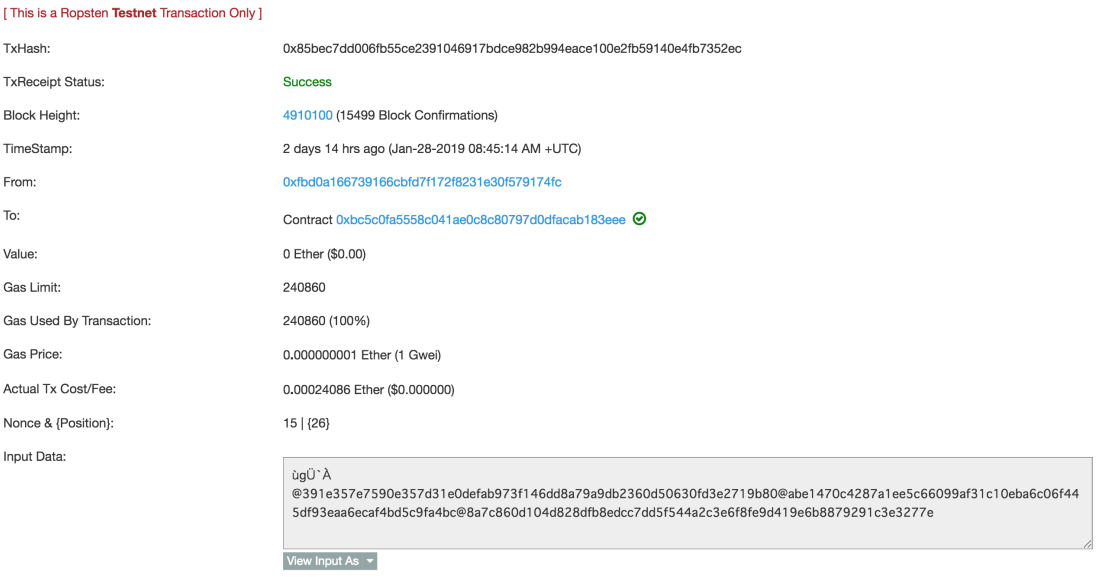

図 6　スマートコントラクト実行時のトランザクションを Etherscan で確認した例

5. 実験

5.1 実験概要

本実験の目的は，以下の 2 点である．

1. 提案手法で共有したソフトウェア情報で事後的な検証が可能かどうか確かめる．

2. 提案手法で増加するコストの量を確かめる．

まず，1 つ目の目的のためのアプローチ（以降，実験 1）として，提案手法を複数のパッケージに適用し，導出したソフトウェア情報が一致するか確かめる．その後，一部のファイルに改ざんを行い，改ざんを検出できるか確かめる．具体的には，仮想環境上でデータセットに対して提案手法を適用して，ソフトウェア情報を導出する．導出したソフトウェア情報を比較し一致しているかどうか確かめる．その後，一部のファイルに改ざんを行い，改ざんを行なったデータに提案手法を適用し，ソフトウェア情報を導出する．改ざん前のソフトウェア情報と改ざん後のソフトウェア情報を比較し，改ざん箇所を検出できるか確かめる．本実験において，導出したソフトウェア情報が一致するということは，ビルドプロセスにおいて用いた入力ファイル群，出力ファイル群とビルドコマンドの全てが一意に識別可能であることを意味する．また，改ざん箇所を検出できるということは，ソフトウェアの構成要素のリンクを維持できていることを意味する．1 章のソフトウェア・トレーサビリティの定義から，ソフトウェア情報の構成要素が満たすため，ソフトウェア・トレーサビリティを確保することができたと言える．したがって，提案手法で共有したソフトウェア情報で事後的な検証が可能であると言える．

2 つ目の目的のためのアプローチ（以降，実験 2）として，提案手法と通常のビルドプロセスにおいて，データ量，金銭的コスト，実行時間の 3 点を計測し比較する．提案手法は通常のビルドプロセスに加えて様々な処理を行なっているため，コストが増加することは自明である．どの程度の増加量であるかを確かめることで，本提案手法の評価を行う．

5.2 実験環境

実験において使用した環境を以下の表 1 に示す.

表 1 使用した環境

OS / ツール	バージョン
macOS Sierra	10.12.6
Debian GNU/Linux	9.6
Virtual Box	5.2.22
Python	3.5.3
strace	4.15

また，ブロックチェーンネットワークは，Ethereum の実験環境であるテストネット Ropsten を利用した.

5.3 データセット

本論文では，Linux ディストリビューションの 1 つである Debian が提供するパッケージを対象とする．本実験では，Debian が提供する iso イメージファイル内に含まれているパッケージを対象に実験を行う．iso ファイル内に含まれているパッケージを抽出した結果，248 個のパッケージが含まれていた.

一般的に，コンピュータにパッケージをインストールする場合は，利用する環境に合わせてすでにビルドやコンパイルされた実行ファイルをダウンロードしインストールを行う．本実験では実際に手元のコンピュータ上でビルドを行った際に得られる，ビルドプロセスのログが必要となるため，パッケージのソースコード群（以降，ソースパッケージ）のみをダウンロードしている.

本実験で用いたデータセットは以下の手順で取得できる.

（Step 1）Debian が提供する iso イメージファイルをダウンロード[4]し，ファイル中に含まれるパッケージの名前を抽出する．本実験で用いた iso イメージファイルのファイル名は “debian-live-9.6.0-i386-xfce.iso” である.

[4]https://cdimage.debian.org/debian-cd/current/amd64/iso-dvd

（Step 2）ゲスト OS 内で，“apt-get source @パッケージ名” コマンドを用いて，ソースパッケージをダウンロードする．@パッケージ名は（Step 1）で抽出したパッケージの名前を表す．

（Step 3）ゲスト OS 内で，“apt-get build-dep @パッケージ名” コマンドを用いて，ソースパッケージのビルドに必要な周辺パッケージをインストールする．@パッケージ名は（Step 1）で抽出したパッケージの名前を表す．

5.4 実験手順

5.4.1 実験 1

1. データセットからランダムに 50 件のパッケージを抽出し，Virtual Box を用いて，Debian GNU/Linux を仮想環境に構築する．
2. 仮想環境上でランダムに抽出したソースパッケージに対して提案手法を適用し，ハッシュ値を導出する．
3. 導出されたハッシュ値を比較し，一致しているか確かめる．
4. 1 つのファイルを改ざんし，提案手法を適用する．得られたソフトウェア情報と手順 1 で導出したソフトウェア情報を比較し改ざんファイルを検出する．

5.4.2 実験 2

1. Debian GNU/Linux の仮想環境を 2 つ作成する．
2. ランダムに抽出した 50 個のソースパッケージに対して，作成した仮想環境上で通常のビルドプロセスと提案手法を適用する．
3. 通常のビルドプロセスと提案手法にかかるコストを計測し比較する．

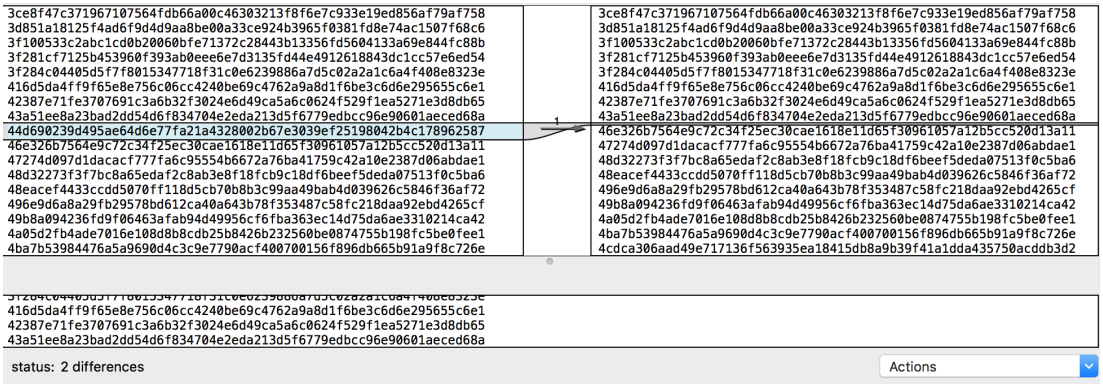

図 7　異なるハッシュ値の検出

5.5　実験結果

5.5.1　実験 1 の結果

仮想環境上で抽出したデータセットに対して提案手法を用いてハッシュ値を導出した．その後，再度抽出したデータセットに対して提案手法を用いてハッシュ値を導出した．パッケージごとに 2 つの導出したハッシュ値を比較したところ，全てのパッケージにおいて値が一致した．続いて，1 つのファイルに改ざんを行ってから提案手法を適用し，ハッシュ値を導出した．改ざんを行わずに提案手法を適用し，導出したハッシュ値と比較すると，入力，出力，ビルドコマンドを代表するハッシュ値のうち，入力を代表するハッシュ値の値が異なっていた．図 7 はファイルの差分を表示する opendiff コマンドを用いて入力のハッシュ値のリストの差分を表示したものである．左側のファイルが改ざんを行ってから，提案手法を適用し出力されたファイルで，右側のファイルが改ざんを行わずに，提案手法を適用し出力されたファイルである．“44d690239d495ae64d6e77fa21a4328002b67e3039ef25198042b4c178962587” というハッシュ値が改ざんを行なったファイルに存在し，改ざんを行なっていないファイルに存在していないことがわかる．図 8 は改ざんを行なったファイルのハッシュ値とファイルパスのリンクを記録したファイルを開いた様子である．上記のハッシュ値でファイル内検索を行なった結果，“/usr/lib/locale/ja_JP.utf8/LC_MEASUREMENT” というファイルが改ざんされたことを特定することができた．

```
9dac6f924d75b996f6badcf6bd4a8777361042854534c414ce4b94259740bdf1 : debian/control
1b23a3b63dcb21dbf0585175e36a0741e814b69f2a062be740b213a5ad97e18d : debian/files
236fcfa371be792edf13096f4cefa666f21a5676feb1bf2bd5f95f2301a08d49 : /usr/lib/x86_64-linux-gnu/perl/5.24/Digest/MD5.pm
f8d2c29e72bd9ead222b3fd91c612456d490657df0df64e5b9074f4a3ba9618c : /usr/share/perl/5.24/Digest/base.pm
e5e76f593b21b054243af8bbb91bcee9bcd50c291bfef1d88b5dd49e6e806848 : /usr/lib/x86_64-linux-gnu/perl/5.24/auto/Digest/MD5/MD5.so
44d690239d495ae64d6e77fa21a4328002b67e3039ef25198042b4c178962587 : /usr/lib/locale/ja_JP.utf8/LC_MEASUREMENT
a391ebdc62a04318f2e42aa25612a33cfbeae7d02416fbc7cfc89799a0c0659e : /usr/lib/x86_64-linux-gnu/perl/5.24/Digest/SHA.pm
be35d33e95cbe24bcb3f11c3ea5ca86c9e6f0df0f7c04256acdefcd59831fbed : /usr/lib/x86_64-linux-gnu/perl/5.24/auto/Digest/SHA/SHA.so
88d770b7e95d793c6250285c4d901365487c44e460a8ebdd4895685f7613d104 : /usr/share/dpkg/cputable
d8311834400bd4c74bcdc764968d13c44b5ac8745a9d06068c2209c51401b826 : /usr/share/dpkg/tupletable
c4b7054191b86e7f92df0fb2dc30ce229cd2691891413ea3486326d8a7691f2e : ../bterm-unifont_1.4_amd64.udeb
c4b7054191b86e7f92df0fb2dc30ce229cd2691891413ea3486326d8a7691f2e : ../bterm-unifont_1.4_amd64.udeb
c4b7054191b86e7f92df0fb2dc30ce229cd2691891413ea3486326d8a7691f2e : ../bterm-unifont_1.4_amd64.udeb
ceedb580d3903b66f52ed33e833d7d9b06765b1b9ab1fe8bbdd627d5895b2b73 : /etc/ld.so.cache
91a225e90a28326285fac19ea582e32bf02bbc081d6be78df6326cef8e97eee9 : /lib/x86_64-linux-gnu/libdl.so.2
```

図 8 改ざんされたファイルの特定

5.5.2 実験 2 の結果

248 個存在しているデータセットからランダムに 50 個のデータセットを抽出し，提案手法と通常のビルドを行なった際のコストを計測した．計測するコストはデータ量，金銭的コスト，実行時間の 3 つである．

データ量は，ダウンロードした際のソースファイルのサイズと提案手法を適用後に生成されるバックアップのサイズを比較する．データ量の計測結果を，表 2 に示す．

表 2 データ量の計測結果

（KB）	ソースファイルのサイズ	バックアップのサイズ
合計	864,312	7,840,988
平均	17,286	156,819
中央値	2,692	101,936

金銭的コストは，通常のビルドでは発生しないため提案手法のみ計測を行なった．recordHash 関数を実行した際の手数料は，（2019 年 1 月 31 日 3:50 UTC 時点）0.00024086 Ether で日本円にして 2.85 円である．本実験では提案手法を 50 回適用しているため，は通常のビルドと比べて 142.5 円の金銭的コストが増加した．また，スマートコントラクトをデプロイする時にも手数料は発生し，その金額は 0.000827501 Ether で日本円にして 9.75 円である．

提案手法は通常のビルドに加えて処理を行なっていることから，提案手法の実行時間が通常のビルドの実行時間よりも増加することが明らかである．データの計測結果を表 3 に示す．通常のビルドと比較して約 5 倍の実行時間がかかってい

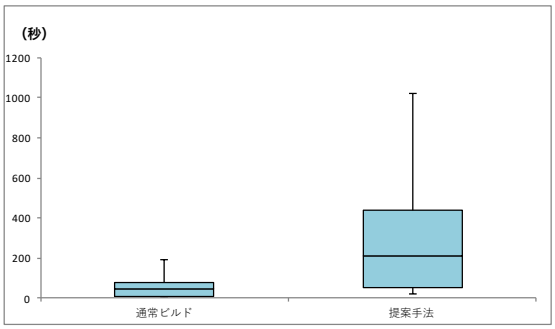

図 9 通常ビルドと提案手法の実行時間の箱ひげ図

ることがわかる．図 9 は通常ビルドと提案手法の実行時間の箱ひげ図である．y 軸の単位は秒である．全体的に通常ビルドよりも提案手法のほうが実行時間が長いことがわかる．図 10 は，バックアップを行なったファイルサイズと提案手法の実行時間の散布図であり，x 軸がファイルサイズ（KB）であり，y 軸が実行時間（秒）である．バックアップを行なったファイルサイズと提案手法の実行時間には比例関係があることが予測される．相関係数を計算すると 0.845 と強い正の相関があることがわかった．

表 3 実行時間の計測結果

（秒）	通常のビルド	提案手法
合計	5,330.96	25,633.43
平均	106.62	512.67
中央値	43.32	212.47

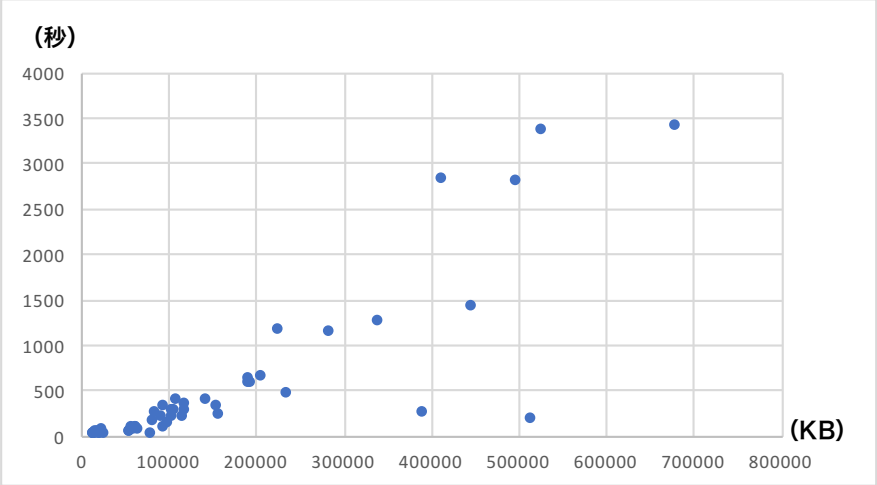

図 10 バックアップファイルサイズと提案手法実行時間の散布図

6. 考察

本研究の提案手法は，入出力ファイルをハッシュ化しているため，実行ファイルに日時情報や乱数を生成させるようなビルドプロセスには適用することができない．このようなファイルは提案手法の仕様上，ハッシュ化してしまうと整合性が取れなくなるため，例外として，バックアップはするがハッシュ化はしないなどの対策で回避する必要がある．

本研究の提案手法は，バックアップしたファイルをソフトウェア情報を記録した組織が保持していることを前提としている．バックアップを保持していない場合は，本提案手法はソフトウェア・トレーサビリティを確保することができない．記録した組織がバックアップしたファイルを保持しているかどうかを定期的に確認する仕組みを取り入れることができれば，本提案手法の有用性はさらに向上することが見込まれる．

5.5.2 項の結果から，提案手法の実行時間は通常のビルドと比較して，約 5 倍の実行時間がかかることがわかった．これはファイルのコピーに時間がかかっていることが予想されるため，バックアップを行う処理を高速化することで実行時間を短縮することが可能であると考えられる．

5.5.2 項の結果から，バックアップを行なったファイルサイズと提案手法の実行

時間には強い正の相関があることがわかった．言い換えれば，バックアップファイルのサイズが大きければ大きいほど，提案手法の実行時間が長くなっている．これはファイルのコピーに時間がかかっていることを示している．

5.5.2 項の結果から，提案手法にかかるコストは，計測時点で 1 回につき 2.85 円と十分に安価な金額である．ブロックチェーンの手数料はデータ量が小さくなればなるほど低下する．そのため，現在記録している 3 つのハッシュをさらにひとまとめにしスマートコントラクトを編集することで，更に安価に提案手法を適用できるようになると考えられる．

本研究の提案手法は Linux 上の strace を利用しており，他の OS 上では機能しない可能性が考えられる．しかし，他の OS においても，システムコールの監視を行うなんらかの関数は存在しているため，現状 strace を利用している部分を対象の OS で利用可能な機能に書き換えることで適応することができると考えられる．

7. おわりに

本研究では，共有するソフトウェア情報を必要最低限に限定し，共有した情報が改ざんされていないことが証明されているという条件を満たして，ブロックチェーンを用いてビルドプロセスにおけるソフトウェア情報の記録と共有を行う手法を Linux 上で提案した．

提案手法の実現可能性を評価するために，ビルドプロセスの記録に要する時間，データ量，ブロックチェーンシステムの手数料を計測した．結果として，十分に計測可能なデータ量と時間，そして安価な手数料で実行可能であることを示した．今後の課題としては，提案手法が要するコストの低減と，他の OS への適応がある．

謝辞

本論文の執筆を進めるに当たり，多くの方々に御指導，御協力を賜りました．ここに御名前を記させて頂くと共に，深く感謝の意を示します．

主指導教員であり，本論文の審査委員を務めて頂いた，奈良先端科学技術大学院大学 情報科学研究科 松本 健一教授には，研究の進め方を始めとして，研究者としての在り方等，あらゆる面で非常に多くの重要な御指導・御助言を賜りました．また課外活動や奨学金申請の書類作成等，学生生活においても御協力を賜り，有意義な学生生活を送ることができました．心より感謝を申し上げます．

副指導教員であり，本論文の審査委員を務めて頂いた，奈良先端科学技術大学院大学 情報科学研究科 安元 慶一教授には，本論文が捉えるべき問題や意義について非常に貴重なご意見を賜り，本論文の質を向上させることができました．心より感謝を申し上げます．

副指導教員であり，本論文の審査委員を務めて頂いた，奈良先端科学技術大学院大学 情報科学研究科 石尾 隆准教授には，本論文の執筆や提案手法のアイデア等，研究に関する様々な面で御指導・御協力を賜りました．先生の支えがあったからこそ，ここまで研究を進めることができました．心より感謝を申し上げます．

副指導教員であり，本論文の審査委員を務めて頂いた，奈良先端科学技術大学院大学 情報科学研究科 畑 秀明助教には，研究の方針について御指導・御協力を賜りました．また学内プロジェクトでも様々な御指導・御助言を賜りました．心より感謝を申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学研究室の幾谷 吉晴氏には本論文の執筆において様々な御指導・御協力を頂きました．また研究計画を立てるにあたって重要な知識，考え方を御指導して頂きました．心より感謝を申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学研究室の高岸 詔子氏には，学生生活や課外活動における書類作成や諸連絡等，事務処理において多大なる御協力を頂きました．心より感謝を申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学研究室の OB，先輩，同級生，後輩の皆様には，学生生活において非常に仲良くして頂き有意義

な2年間を過ごすことができました．心より感謝を申し上げます．
最後に，私の意思を尊重し，常に支えてくださった父に深く感謝いたします．

参考文献

[1] 東京地判平成27年6月25日（h25（ワ）第18110号）. http://www.courts.go.jp/app/files/hanrei_jp/270/085270_hanrei.pdf.

[2] Nasir Ali, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. *IEEE Transactions on Software Engineering*, 39(5):725–741, 2013.

[3] Hazeline U Asuncion, Arthur U Asuncion, and Richard N Taylor. Software traceability with topic modeling. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, volume 1, pages 95–104. IEEE, 2010.

[4] Moritz Beller and Joseph Hejderup. Blockchain-based software engineering. In *Proceedings of the 41th International Conference on Software Engineering, ICSE 2019, Montréal, Canada, May 25 - May 31, 2019*, page to appear, 2019.

[5] Jane Cleland-Huang. Just enough requirements traceability. In *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*, volume 1, pages 41–42. IEEE, 2006.

[6] Jane Cleland-Huang, Orlena CZ Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: trends and future directions. In *Proceedings of the on Future of Software Engineering*, pages 55–69. ACM, 2014.

[7] G. Destefanis, M. Marchesi, M. Ortu, R. Tonelli, A. Bracciali, and R. Hierons. Smart contracts vulnerabilities: a call for blockchain software engineering? In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 19–25, March 2018.

[8] Salome Maro, Jan-Philipp Steghöfer, and Miroslaw Staron. Software traceability in the automotive domain: Challenges and solutions. *Journal of*

Systems and Software, 141:85 – 110, 2018.

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

[10] Sander van der Burg, Eelco Dolstra, Shane McIntosh, Julius Davies, Daniel M. German, and Armijn Hemel. Tracing software build processes to uncover license compliance inconsistencies. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 731–742, New York, NY, USA, 2014. ACM.

[11] S. Yamada, M. Ugumori, and S. Kusumoto. A software tag generation system to realize software traceability. In *2010 Asia Pacific Software Engineering Conference*, pages 423–432, Nov 2010.

[12] Peng Zhang, Jules White, Douglas C Schmidt, and Gunther Lenz. Applying software patterns to address interoperability in blockchain-based healthcare apps. *arXiv preprint arXiv:1706.03700*, 2017.

付録

A. ハッシュ値を記録するためのスマートコントラクトソースコード

```
pragma solidity ^0.4.24;

contract Trace_hash {
    struct Hash {
        string inputHash;
        string outputHash;
        string commandHash;
    }
    Hash[] public hashes;

    function recordHash (string _inputHash, string _outputHash,
        string _commandHash) returns (uint) {
            hashes.length++;
            hashes[hashes.length - 1].inputHash = _inputHash;
            hashes[hashes.length - 1].outputHash = _outputHash;
            hashes[hashes.length - 1].commandHash = _commandHash;
            return hashes.length;
    }

    function getHashesCount() public view returns(uint) {
        return hashes.length;
    }

    function getHash(uint index) public view
    returns (string, string, string) {
        return (hashes[index].inputHash,
        hashes[index].outputHash,
        hashes[index].commandHash);
    }
}
```