Title: Securing DevOps & Supply Chain Management with Blockchain

Abstract:

Blockchain technology has the potential to revolutionize the way DevOps and supply chain management are conducted. By providing a secure and transparent way to track and manage data, blockchain can help to improve visibility, traceability, and collaboration between different stakeholders. This can lead to a more secure and efficient software development and delivery process.

1. Introduction

DevOps and supply chain management are two important approaches to improving the security and efficiency of software development and delivery. DevSecOps is a security-focused approach to software development that emphasizes the integration of security throughout the entire software development lifecycle. The software supply chain is the process of managing the entire lifecycle of software, from conception to retirement.

Blockchain technology is a distributed ledger technology that can be used to create a secure and transparent way to track and manage data. Blockchain is a tamper-proof distributed ledger that records transactions in a chronological order. This makes it difficult to change or delete data without the knowledge of all participants in the blockchain network.

1.1 Background

In recent years, the software development industry has experienced significant growth and innovation, resulting in complex and interconnected supply chains. The software supply chain involves multiple stakeholders, mainly clients, business analysts, product managers, developers, DevOps CI/CD teams, Production support and monitoring, Audit and Compliance. Each stakeholder plays a crucial role in delivering high-quality software products to end-users.

1.2 Problem Statement

The ever growing need for delivering high-quality software products to end-users often leads to challenges within the software supply chain. One of the primary challenges is the lack of transparency. Due to the involvement of various stakeholders and the absence of a unified system, it becomes difficult to track and trace software development throughout their lifecycle. This lack of transparency can lead to delays, miscommunication, and difficulties in verifying the authenticity and integrity of software components.

Trust issues can arise when there is a lack of visibility and accountability among participants in the supply chain. Each stakeholder relies on others to fulfill their responsibilities and without a transparent and trustworthy system, doubts and conflicts can emerge. These trust issues can hinder effective collaboration and result in delays and inefficiencies in the software development process.

The software supply chain also faces challenges in managing and documenting requirements, especially in the vendor/client relationship. The process of capturing and communicating requirements accurately is crucial for successful software development. However, traditional methods often lack transparency and traceability, leading to misunderstandings and potential gaps between clients and vendors.

1.3 Research Objectives

The objective of this research paper is to deep dive into Blockchain technology and pitch forward an idea of internal blockchain mechanism that can help us with:

- **Software development**: Blockchain can be used to track the progress of software development and to ensure that all changes are made in a transparent and auditable way.
- **Software delivery**: Blockchain can be used to track the delivery of software to ensure that it is delivered on time and to the correct recipient.
- **Software licensing**: Blockchain can be used to track software licenses and to ensure that they are not being used illegally.

2. Software Supply Chain Overview

The software supply chain is the process of managing the entire lifecycle of software, from conception to retirement. It includes managing the development, testing, deployment, and maintenance of software.

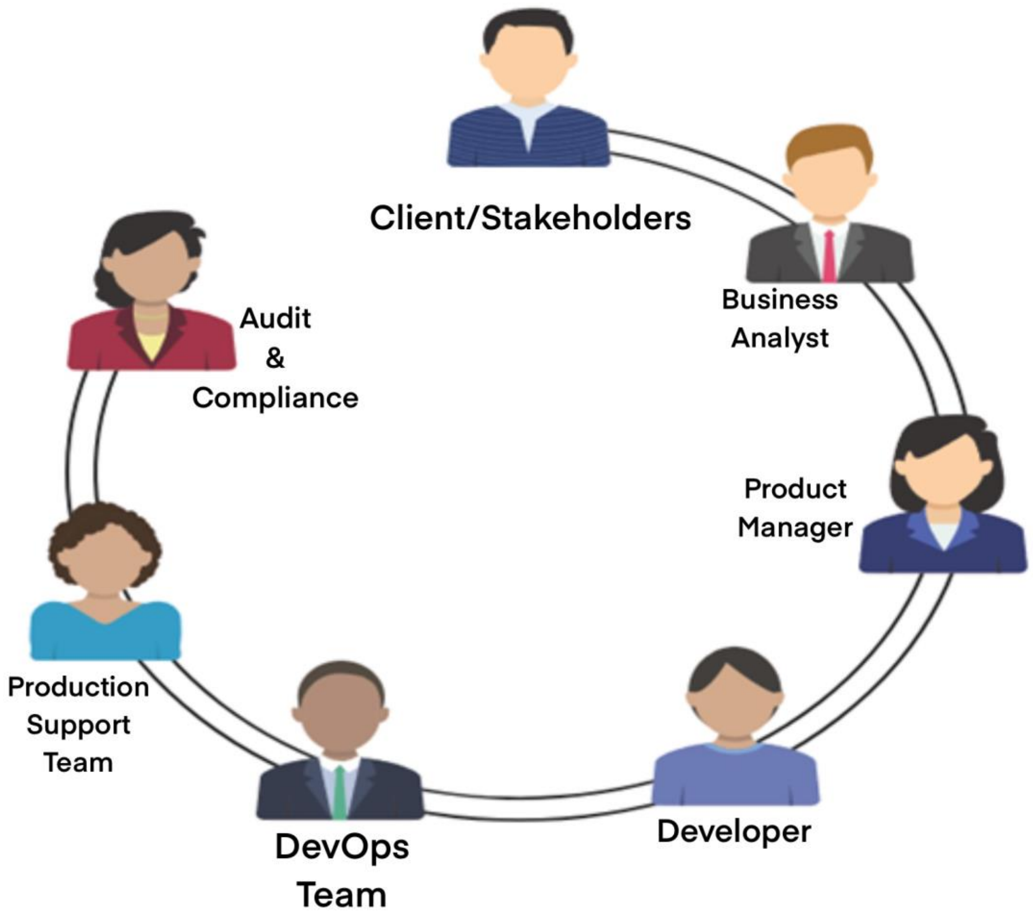


Fig 1: Software Supply Chain Overview showcasing the participants

2.1 Stakeholders and Roles

The software supply chain is becoming increasingly complex as software becomes more critical to businesses. There are many different stakeholders involved in the software supply chain, each with their own role to play.

2.1.1 Client (Internal/External)

The client is the person or organization that is requesting the software. They provide the requirements for the software and track its progress.

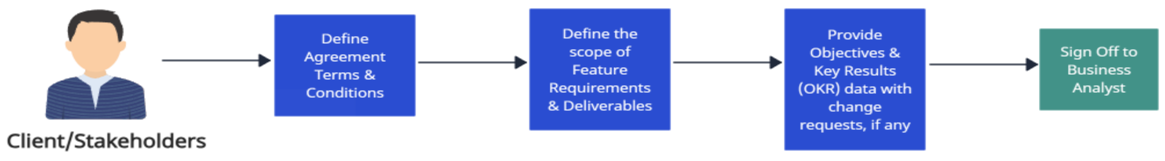

Client/Stakeholders
Define Agreement Terms & Conditions
Define the scope of Feature Requirements & Deliverables
Provide Objectives & Key Results (OKR) data with change requests, if any
Sign Off to Business Analyst

Fig 2: Role of Clients/Stakeholders in the supply chain

2.1.2 Business Analyst

The business analyst is responsible for understanding the client's requirements and translating them into technical specifications.

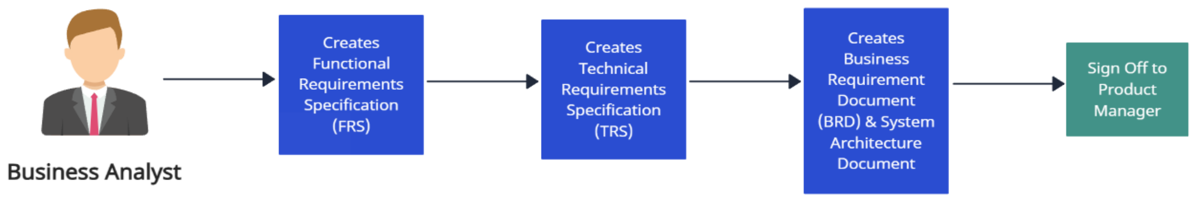

Business Analyst
Creates Functional Requirements Specification (FRS)
Creates Technical Requirements Specification (TRS)
Creates Business Requirement Document (BRD) & System Architecture Document
Sign Off to Product Manager

Fig 3: Role of Business Analyst in the supply chain

2.1.3 Product Manager

The product manager is responsible for the overall success of the software project. They ensure that the software meets the client's requirements and that it is delivered on time and within budget.

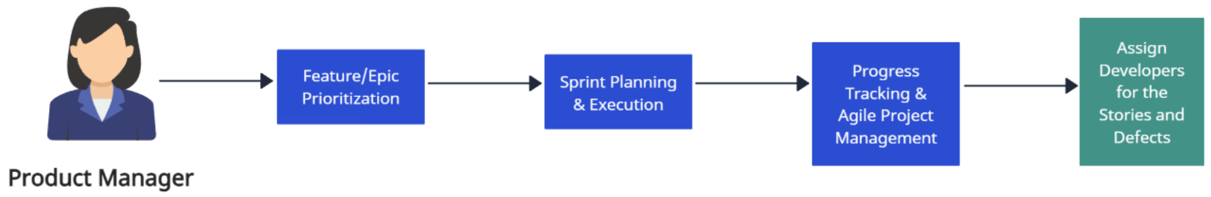

Product Manager
Feature/Epic Prioritization
Sprint Planning & Execution
Progress Tracking & Agile Project Management
Assign Developers for the Stories and Defects

Fig 4: Role of Product Manager in the supply chain

2.1.4 Developer

The developer is responsible for coding the software. They work with the business analyst and product manager to ensure that the software meets the requirements.

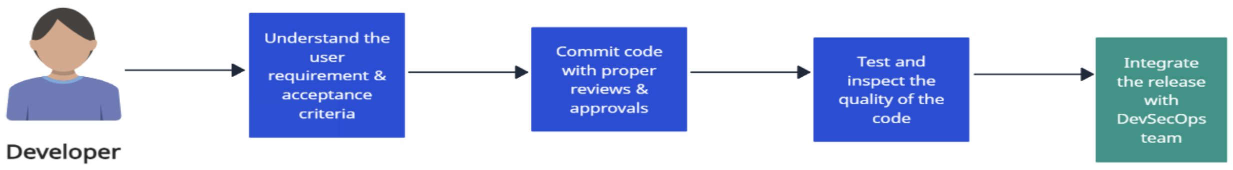
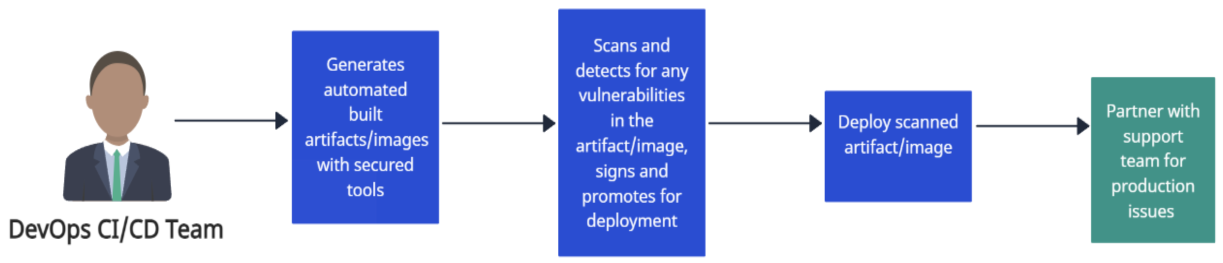

Developer
Understand the user requirement & acceptance criteria
Commit code with proper reviews & approvals
Test and inspect the quality of the code
Integrate the release with DevSecOps team

Fig 5: Role of Developer in the supply chain

2.1.5 DevOps CI/CD Pipeline

DevOps CI/CD is responsible for the continuous integration and continuous delivery (CI/CD) of the software. They ensure that the software is built, tested, and deployed automatically.

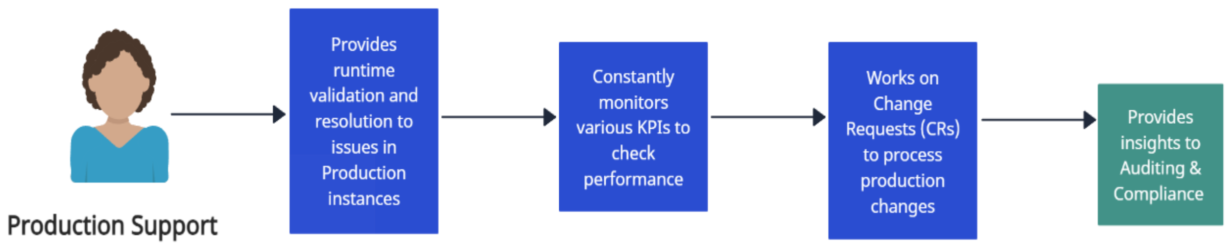


DevOps CI/CD Team
Generates automated built artifacts/images with secured tools
Scans and detects for any vulnerabilities in the artifact/image, signs and promotes for deployment
Deploy scanned artifact/image
Partner with support team for production issues

Fig 6: Role of DevOps CI/CD team in the supply chain

2.1.6 Production Support Team

Production support and monitoring is responsible for ensuring that the software is running smoothly in production. They track any issues that arise and work to resolve them quickly.

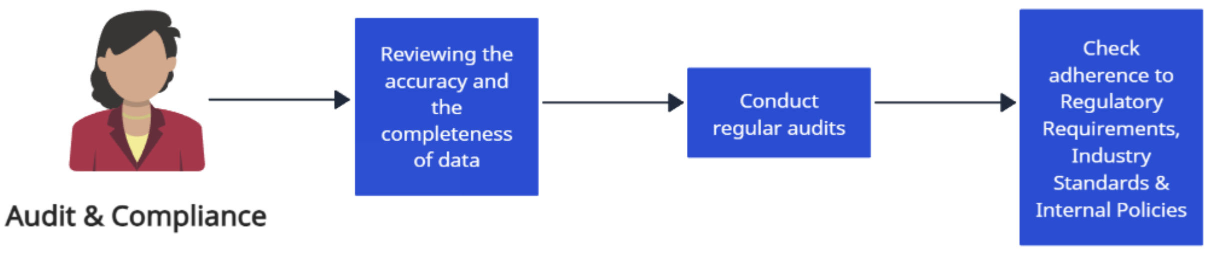


Production Support
Provides runtime validation and resolution to issues in Production instances
Constantly monitors various KPIs to check performance
Works on Change Requests (CRs) to process production changes
Provides insights to Auditing & Compliance

Fig 7: Role of Production Support Team in the supply chain

2.1.7 Auditing & Compliance

Auditing and compliance is responsible for ensuring that the software complies with all relevant regulations. They also audit the software to ensure that it is secure and reliable.



Audit & Compliance
Reviewing the accuracy and the completeness of data
Conduct regular audits
Check adherence to Regulatory Requirements, Industry Standards & Internal Policies

Fig 8: Role of Auditing & Compliance in the supply chain

3. <u>Blockchain and Distributed Ledger</u>

Blockchain uses a Distributed ledger technology which can provide a secure and transparent way to track and manage the software supply chain. This can help to improve visibility, traceability, and collaboration between different stakeholders.

3.1 Introduction to Blockchain

Blockchain technology is a system of recording information in a way that makes it difficult or impossible to change, hack, or cheat the system. A distributed ledger is a database that is shared across a network of computers. Blockchain technology and distributed ledgers are often used together, and they have the potential to revolutionize many industries. Some of the key features of blockchain technology are:

- **Immutability**: Once data is added to a blockchain, it cannot be changed or deleted. This makes blockchain technology ideal for recording transactions and other sensitive data.
- **Transparency**: All transactions on a blockchain are publicly visible. This allows for greater transparency and accountability.
- **Security**: Blockchain technology is very secure. It uses cryptography to protect data from unauthorized access.
- **Scalability**: Blockchain technology can be scaled to handle a large number of transactions.

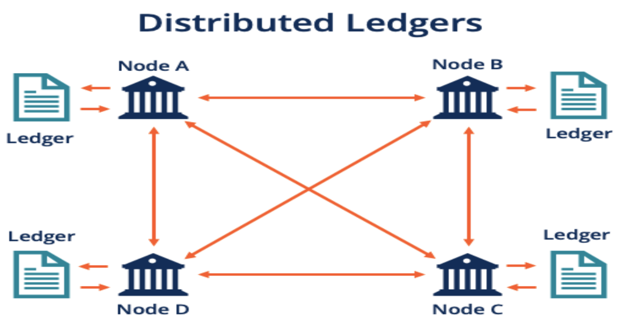


Fig 9: Distributed Ledger high level working diagram

3.2 Benefits and Challenges of Blockchain in Software Supply Chain

Some of the key benefits of using blockchain technology are:

- **Increased security**: Blockchain technology can help to increase the security of transactions and data.
- **Reduced fraud**: Blockchain technology can help to reduce fraud by making it more difficult to tamper with data.
- **Improved transparency**: Blockchain technology can help to improve transparency by making all transactions publicly visible.
- **Enhanced efficiency**: Blockchain technology can help to improve efficiency by streamlining transactions and reducing the need for intermediaries.
- **Enhanced traceability**: Blockchain technology can help to improve traceability by providing a secure and transparent record of transactions.

Most common challenges of using blockchain technology are:

- **Complexity**: Blockchain technology can be complex to understand and implement.
- **Cost**: Blockchain technology can be expensive to implement.
- **Regulation**: Blockchain technology is still in its early stages, and there is a lack of regulatory clarity.
- **Environmental impact**: The energy consumption of blockchain technology is a concern.

4. <u>Design and Architecture of Blockchain-Based Software Supply Chain</u>

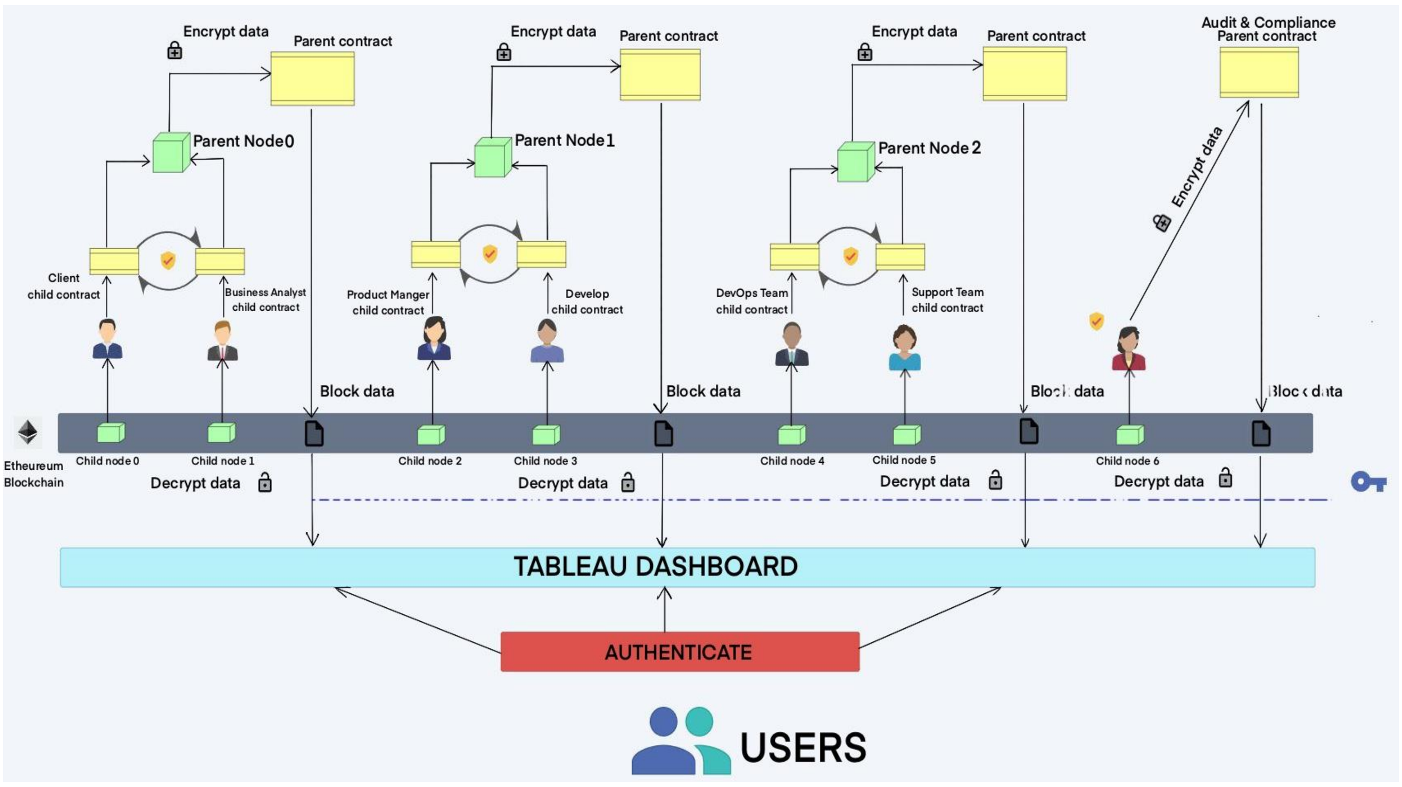


Fig 10: High Level Design Diagram of the Blockchain based Software Supply Chain

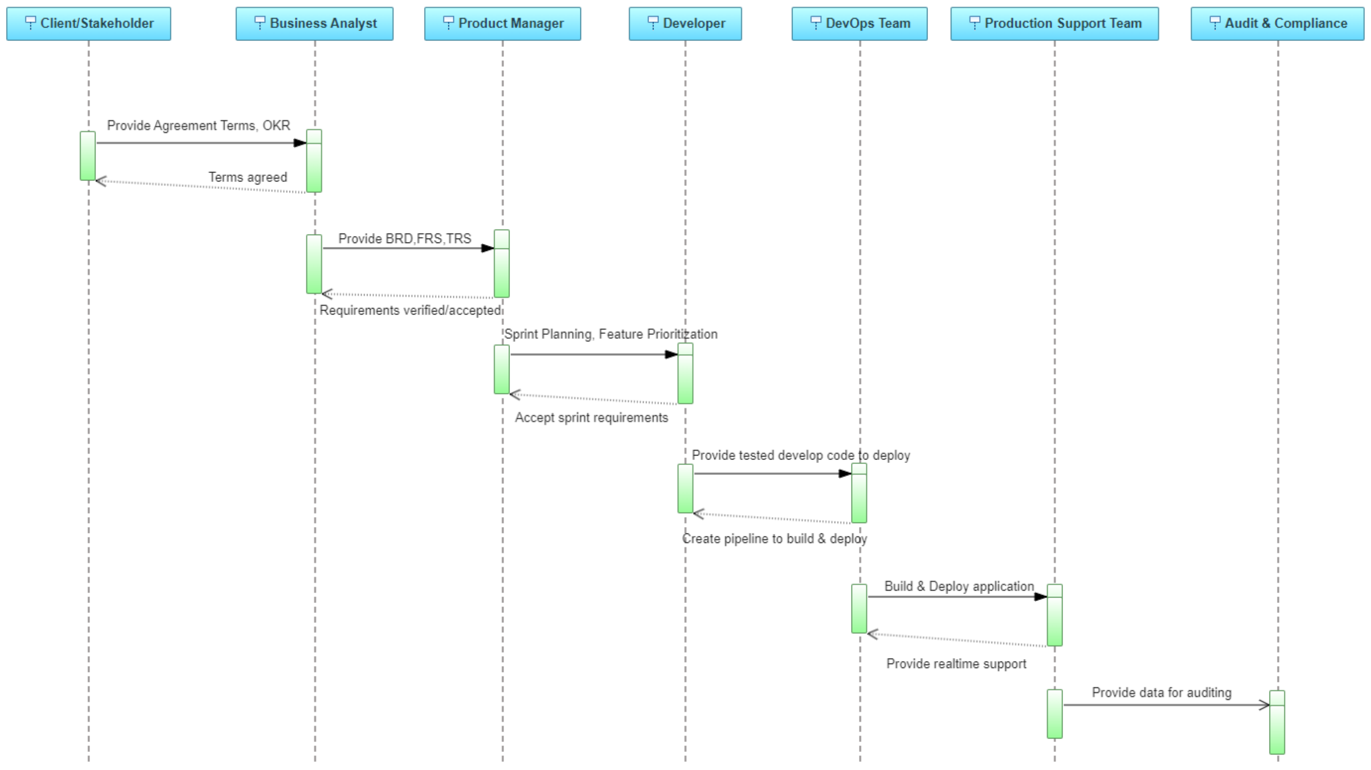

Client/Stakeholder
Business Analyst
Product Manager
Developer
DevOps Team
Production Support Team
Audit & Compliance
Provide Agreement Terms, OKR
Terms agreed
Provide BRD,FRS,TRS
Requirements verified/accepted
Sprint Planning, Feature Prioritization
Accept sprint requirements
Provide tested develop code to deploy
Create pipeline to build & deploy
Build & Deploy application
Provide realtime support
Provide data for auditing

Fig 11: Sequence diagram showcasing the execution of child node contracts in the blockchain

4.1 System Components

4.1.1 Blockchain Network

The blockchain network serves as the foundation of the software supply chain solution. It comprises a decentralized network of nodes that collectively maintain the blockchain ledger. In the context of this white paper, the blockchain network allows participants, such as vendors, business analysts, product managers, developers, and other relevant parties, to interact and contribute to the shared ledger. The blockchain network ensures the immutability and transparency of the data stored on the ledger.

4.1.2 Validator Nodes

A validator node refers to a participant responsible for validating transactions and maintaining the consensus of the Blockchain network. Validator nodes play a critical role in ensuring the integrity and security of the blockchain, particularly in permissioned or private blockchain networks.

4.1.3 Smart Contracts

Smart contracts are self-executing agreements that run on the blockchain network. They enable automation, enforce business logic, and facilitate interactions between participants in a transparent and tamper-resistant manner. In the proposed solution, smart contracts are utilized to define the contractual terms and conditions between different stakeholders in the software supply chain.

4.1.4 User Interfaces

User interfaces provide the means for participants to interact with the blockchain-based software supply chain system. UIs can be web-based or application-based interfaces that allow stakeholders to view and update information, submit transactions, access reports and analytics, and perform various actions within the system. The user interfaces ensure ease of use and accessibility for participants from different roles and responsibilities, such as vendors, business analysts, product managers, developers, and other relevant users.

4.1.5 APIs and Integrations

APIs and integrations enable seamless communication and data exchange between the blockchain-based software supply chain system and external systems or services. API integrations can include project management tools, development environments, testing frameworks, deployment systems, monitoring platforms, and more. By integrating with external systems, the blockchain-based solution can leverage existing infrastructure and enhance interoperability.

4.2 Data Structure and Transactions

In the context of the blockchain-based software supply chain solution described in the research paper, the data structure and transactions refer to how information is organized and stored on the blockchain ledger and how participants interact with the ledger through transactions. This section outlines the key aspects of data structure and transactions.

4.2.1 **Data Structure**: The data structure on the blockchain ledger is typically organized as a series of blocks, with each block containing a set of transactions. Each transaction represents a specific action or event within the software supply chain. The data structure is designed to ensure immutability, transparency, and integrity of the recorded information. It can include various data fields, such as timestamps, participant identifiers, artifact details, contractual terms, requirements, development records, deployment status, KPIs, CR data, and other relevant information.

4.2.2 Transactions: Transactions are the fundamental units of interaction with the blockchain ledger. Participants in the software supply chain initiate transactions to record and update information on the ledger. These transactions can include actions like registering software artifacts, updating requirements, submitting development records, marking deployment status, tracking KPIs, and recording CR data. Each transaction is securely and immutably recorded on the blockchain ledger, providing a transparent and auditable history of actions taken by the participants.

When a participant initiates a transaction, they sign it with their digital signature to verify their identity and authorization. The transaction is then broadcasted to the blockchain network, where it is validated and added to a block by consensus mechanisms, such as proof-of-work or proof-of-stake. Once added to the blockchain, the transaction becomes a permanent part of the ledger and cannot be altered or deleted.

The data structure and transactions within the blockchain-based software supply chain solution ensure that the entire lifecycle of software artifacts and related information is securely and transparently recorded. Each participant's actions, updates, and contributions are timestamped and permanently stored on the blockchain ledger, providing an auditable history and enhancing transparency and trust throughout the supply chain.

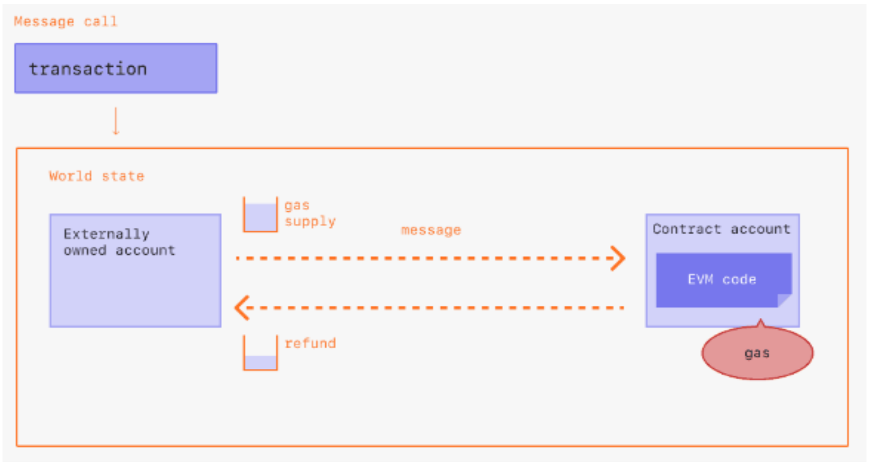


Fig 12: Sequence diagram of a typical transaction in an Ethereum network

It is also possible to implement a private Ethereum network that does not require gas fees for transactions and the transactions can be executed only by the node validators. The network will not be minable and no reward for completing a block since it's private.

To do this, we would need to use a consensus mechanism that does not require gas fees, such as Proof of Authority (PoA). PoA is a consensus mechanism where a small number of trusted nodes, called validators, are responsible for validating and adding blocks to the blockchain. In a PoA network, there is no need for gas fees because the validators are not competing to add blocks to the blockchain. Instead, the validators are selected by the network administrators and they are rewarded with tokens for their services. To prevent spam and DoS attacks, we can set a minimum gas price for transactions. This will ensure that only legitimate transactions are included in blocks. We can also configure the network so that only node validators can execute transactions. This will prevent unauthorized users from making changes to the blockchain.

5. Overview of the components in Blockchain-based Software Supply Chain Solution

This section provides an overview of the key components and their interactions within the solution:

- Blockchain Network and Distributed Ledger
- Smart Contracts for Stakeholders
- Client and Stakeholder Contracts for Feature Development
- Business Analyst Contracts for Document Storage
- Product Manager, Developer, and DevOps Child Contracts for Agile and Sprint Records
- Deployment Contract for Deployment Status and KPI Data
- Production Support Contract for CR Data
- Data Encryption with SHA-256 and Combined Hashing
- Integration with Existing Systems and Tools

5.1 Blockchain Network and Distributed Ledger

In the proposed software supply chain solution, Ethereum blockchain is chosen as the underlying network to establish a distributed ledger. Ethereum is a decentralized platform that supports smart contracts and provides a robust infrastructure for building blockchain-based applications. In this context, the Proof of Authority (PoA) consensus mechanism is employed to achieve consensus among participants. Let's delve into the key aspects of the blockchain network and distributed ledger:

5.1.1 Ethereum Blockchain Network: The Ethereum network consists of multiple nodes, each operated by participants such as vendors, business analysts, product managers, developers, DevOps teams, deployment personnel, and production support teams. These nodes form a distributed network, ensuring that the ledger is replicated and synchronized across all participating nodes. The Ethereum network provides a secure and reliable infrastructure for executing smart contracts and recording transactions.

5.1.2 Proof of Authority (PoA) Consensus Mechanism: The PoA consensus mechanism is implemented to achieve consensus among participants in the Ethereum network. In PoA, a select group of authorized nodes, known as validators or authorities, are responsible for validating and confirming transactions. These validators are trusted entities within the software supply chain ecosystem would be all the stakeholders and developers in the Software Supply Chain with Access Control set in place. The PoA mechanism ensures faster transaction confirmations compared to other consensus mechanisms like Proof of Work (PoW) or Proof of Stake (PoS).

5.1.3 Validator Nodes and Block Validation: Validator nodes play a crucial role in maintaining the integrity of the blockchain network. Validators use their private keys to sign the blocks, providing authenticity and verifying their authority. Once a block is created, it is added to the blockchain, and the participating nodes update their local copies of the distributed ledger. In this solution, we will be using the concept of child and parent nodes.

The parent node typically plays a role in coordinating and managing the communication between its child nodes. It may relay information, propagate transactions or blocks, and ensure the synchronization of data across the network. The parent node is responsible for maintaining the network's integrity and facilitating the consensus process.

Child nodes receive information, transactions, and blocks from their parent node and rely on it for synchronization and validation. They contribute to the network by relaying transactions, validating blocks, and participating in the consensus mechanism.

5.1.4 Block Finality and Consensus: In the PoA consensus mechanism, block finality is achieved through the consensus of authorized validators. Once a block is created and validated by the authorized nodes, it is considered final, and subsequent blocks build upon it. Consensus is achieved by the agreement of the authorized nodes, eliminating the need for computationally intensive tasks like mining in PoW. This consensus mechanism ensures efficient transaction processing and a more predictable block confirmation time.

To create a private Ethereum network, we will need to:

1. Install the Ethereum client software.
2. Create a genesis block.
3. Create a private keystore.
4. Invite other users to join the network.

To ensure that all nodes on the network validate all transactions, we can PoA consensus mechanism that requires a certain number of nodes to approve a transaction before it is added to the blockchain. This ensures that all transactions on the blockchain are valid and cannot be tampered with.

To enable PoA consensus mechanism on a private Ethereum chain, we will need to modify the genesis block of the network. The genesis block is the first block in the blockchain, and it contains the initial configuration of the network.

To modify the genesis block, we will need to add the following parameters:

- **poc-enabled**: This parameter must be set to true to enable PoA consensus.
- **poc-validators**: This parameter is a list of addresses of the nodes that will be validators in the PoA network.

Once we have modified the genesis block, we will need to restart the Ethereum client software. The client software will then create a new blockchain with PoA consensus enabled.

Here is an example json configuration of a genesis block that enables PoA consensus:

```
{
  "config": {
    "chainId": 1,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "daoForkBlock": 0,
    "daoForkSupport": false,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "istanbulBlock": 0,
    "muirGlacierBlock": 0,
    "genesisTime": "2023-06-22T12:00:00Z",
    "difficulty": "1",
    "gasLimit": "10000000",
    "alloc": {},
    "poc-enabled": true,
    "poc-validators": ["0x1234567890abcdef", "0x1234567890abcdef", "0x1234567890abcdef"]
  }
}
```

This genesis block enables PoA consensus and specifies that certain addresses will be validators in the PoA network. Once you have created the genesis block, you can start the Ethereum client software and the network will be created with PoA consensus enabled.

5.2 Smart Contracts for Stakeholders

In the blockchain-based software supply chain solution, smart contracts are utilized to establish agreements and define the terms and responsibilities between the client and various stakeholders involved in the software development lifecycle. These contracts serve as a foundation for collaboration, transparency, and accountability. Let's delve deeper into the details of client and stakeholder contracts:

5.2.1 Client and Stakeholder Contracts for Feature Development

5.2.1.1 Agreement Terms and Conditions: Client and stakeholder contracts outline the terms and conditions that govern the working relationship between the client and each stakeholder. These terms include project scope, timelines, deliverables, pricing, payment terms, intellectual property rights, confidentiality agreements, and dispute resolution mechanisms. By recording these terms on the blockchain, the contracts provide a transparent and tamper-proof record of the agreed-upon terms.

5.2.1.2 Definition of Feature Requirements: Client and stakeholder contracts contain a detailed description of the desired features and functionalities of the software. The client communicates their requirements, expectations, and objectives to the stakeholders, who then translate them into technical specifications. These contracts ensure that all parties involved have a clear understanding of the feature requirements, reducing the risk of miscommunication or misunderstandings.

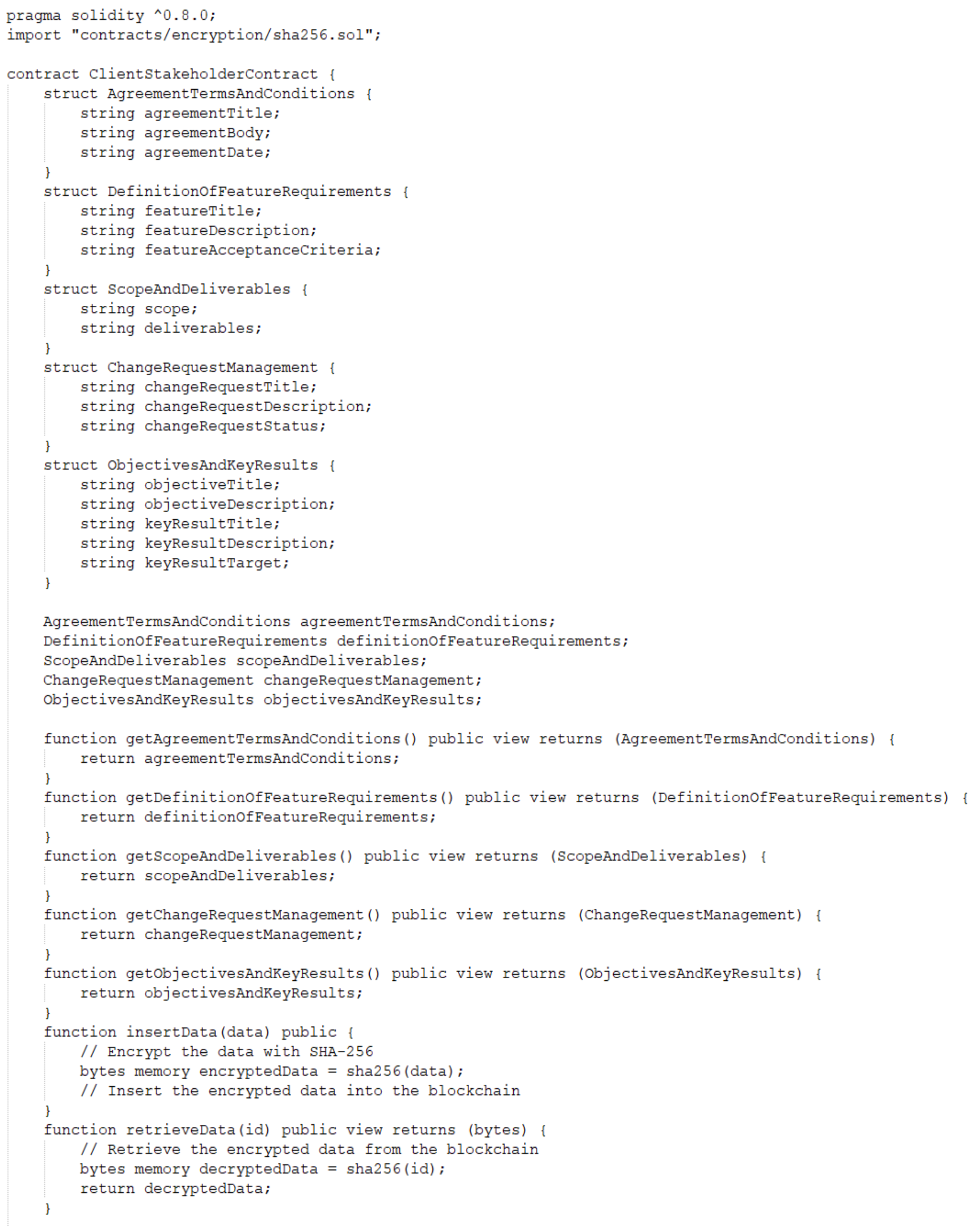
5.2.1.3 Scope and Deliverables: The contracts outline the scope of work and deliverables for feature development. This includes specifying the expected outcomes, milestones, and timelines for each feature. Stakeholders, such as developers and product managers, refer to these contracts to understand their responsibilities and the deliverables they are accountable for. This ensures that the development process stays on track and aligns with the client's expectations.

5.2.1.4 Acceptance Criteria: To ensure client satisfaction and successful feature delivery, acceptance criteria are defined within the contracts. These criteria outline the specific conditions that must be met for the client to accept the completed features. They may include functional requirements, performance benchmarks, usability guidelines, and any other relevant parameters. By including acceptance criteria in the contracts, stakeholders have clear guidelines for developing features that meet the client's expectations.

5.2.1.5 Change Request Management: During the feature development process, the client may request changes or modifications to the initial requirements. The contracts establish procedures for managing change requests, including documentation, impact analysis, approval processes, and any associated costs or timeline adjustments. By formalizing the change request process within the contracts, both the client and stakeholders have a clear understanding of how changes are managed and integrated into the development process.

5.2.1.6 Objectives and Key Results (OKRs) Data: OKRs are a strategic planning framework that helps organizations set and track goals and key results. In the context of the software supply chain, OKRs can be established by the business analysts to align development efforts with the broader organizational objectives. The business analyst contracts enable the storage and management of OKRs data within the smart contracts. This ensures that OKRs are transparent, accessible, and trackable by stakeholders throughout the software development lifecycle.

A sample Solidity contract code is shown below to achieve the insertion of encrypted data mentioned in the scope of Client and Stakeholders in the blockchain:



```
pragma solidity ^0.8.0;
import "contracts/encryption/sha256.sol";

contract ClientStakeholderContract {
    struct AgreementTermsAndConditions {
        string agreementTitle;
        string agreementBody;
        string agreementDate;
    }
    struct DefinitionOfFeatureRequirements {
        string featureTitle;
        string featureDescription;
        string featureAcceptanceCriteria;
    }
    struct ScopeAndDeliverables {
        string scope;
        string deliverables;
    }
    struct ChangeRequestManagement {
        string changeRequestTitle;
        string changeRequestDescription;
        string changeRequestStatus;
    }
    struct ObjectivesAndKeyResults {
        string objectiveTitle;
        string objectiveDescription;
        string keyResultTitle;
        string keyResultDescription;
        string keyResultTarget;
    }

    AgreementTermsAndConditions agreementTermsAndConditions;
    DefinitionOfFeatureRequirements definitionOfFeatureRequirements;
    ScopeAndDeliverables scopeAndDeliverables;
    ChangeRequestManagement changeRequestManagement;
    ObjectivesAndKeyResults objectivesAndKeyResults;

    function getAgreementTermsAndConditions() public view returns (AgreementTermsAndConditions) {
        return agreementTermsAndConditions;
    }
    function getDefinitionOfFeatureRequirements() public view returns (DefinitionOfFeatureRequirements) {
        return definitionOfFeatureRequirements;
    }
    function getScopeAndDeliverables() public view returns (ScopeAndDeliverables) {
        return scopeAndDeliverables;
    }
    function getChangeRequestManagement() public view returns (ChangeRequestManagement) {
        return changeRequestManagement;
    }
    function getObjectivesAndKeyResults() public view returns (ObjectivesAndKeyResults) {
        return objectivesAndKeyResults;
    }
    function insertData(data) public {
        // Encrypt the data with SHA-256
        bytes memory encryptedData = sha256(data);
        // Insert the encrypted data into the blockchain
    }
    function retrieveData(id) public view returns (bytes) {
        // Retrieve the encrypted data from the blockchain
        bytes memory decryptedData = sha256(id);
        return decryptedData;
    }
```

A sample code snippet is added in order to access the above data in Blockchain UI Dashboard using node:

```
const fs = require("fs");
const tableau = require("tableau");

const clientStakeholderContract = new tableau.ClientStakeholderContract("https://localhost:8543/");

const agreementTermsAndConditions = clientStakeholderContract.getAgreementTermsAndConditions();
const definitionOfFeatureRequirements = clientStakeholderContract.getDefinitionOfFeatureRequirements();
const scopeAndDeliverables = clientStakeholderContract.getScopeAndDeliverables();
const changeRequestManagement = clientStakeholderContract.getChangeRequestManagement();
const objectivesAndKeyResults = clientStakeholderContract.getObjectivesAndKeyResults();

const dashboard = tableau.createDashboard();
dashboard.addWorksheet("Agreement Terms and Conditions", agreementTermsAndConditions);
dashboard.addWorksheet("Definition of Feature Requirements", definitionOfFeatureRequirements);
dashboard.addWorksheet("Scope and Deliverables", scopeAndDeliverables);
dashboard.addWorksheet("Change Request Management", changeRequestManagement);
dashboard.addWorksheet("Objectives and Key Results", objectivesAndKeyResults);

const dashboardId = dashboard.save();

const accessControl = {
  allowedUsers: ["admin", "user1", "user2"],
};

tableau.publishDashboard(dashboardId, accessControl);
```

This script will first retrieve the data from the blockchain contracts. Then, it will create a Tableau dashboard and add the data to the dashboard. Finally, it will publish the dashboard to Tableau Server. The dashboard will be accessible to users who are listed in the access control object.

5.2.2 Business Analyst Contracts for Document Storage

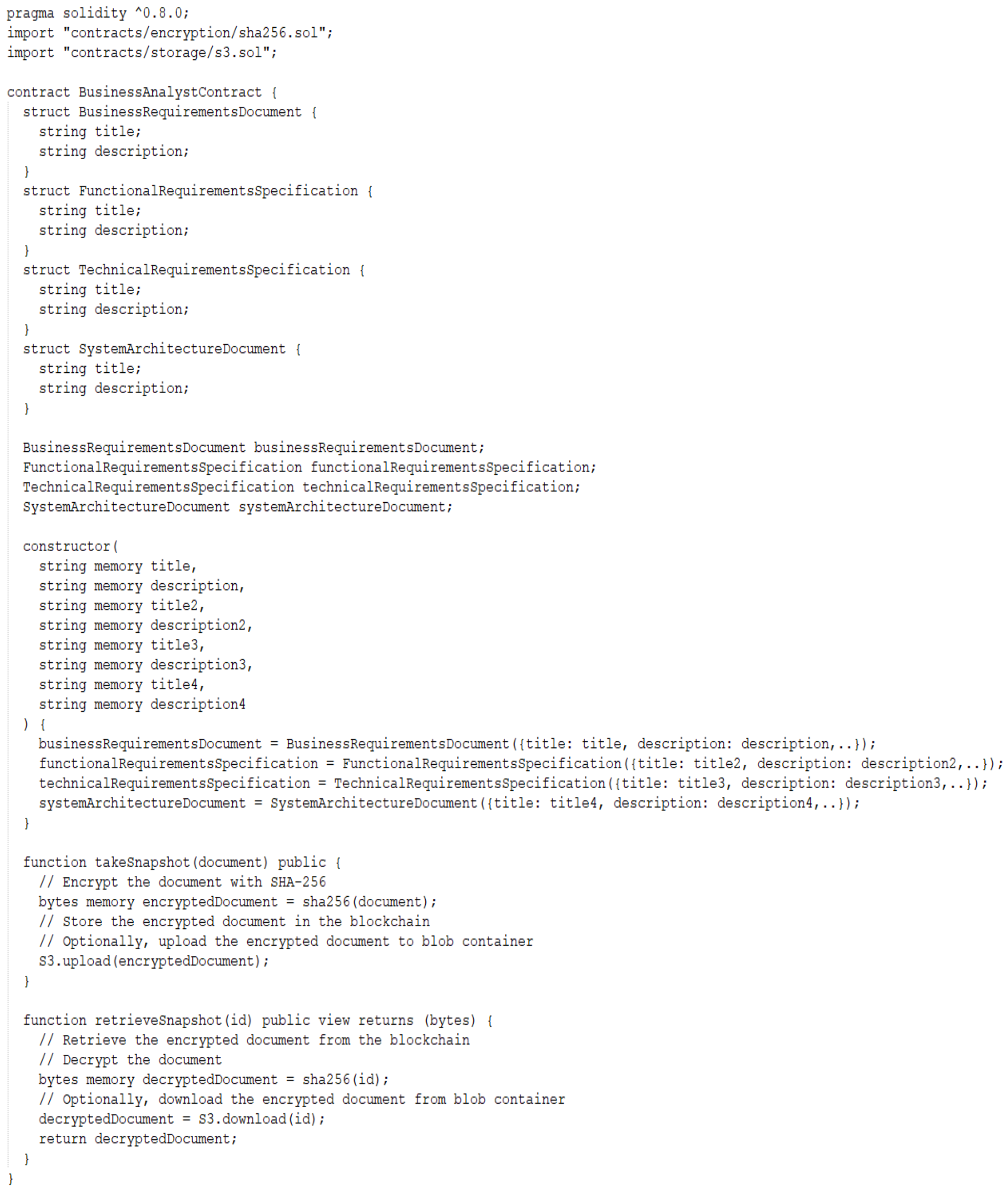
5.2.2.1 Business Requirements Document (BRD): The BRD outlines the high-level business objectives, goals, and constraints of the software project. BAs collaborate with clients and stakeholders to understand their business processes, needs, and vision. They conduct interviews, workshops, and analysis to gather requirements and define the scope of the project. BAs then translate these requirements into a structured document that captures the business context, user personas, use cases, and non-functional requirements. By keeping a snapshot of these documents on the blockchain, the contracts provide a transparent and tamper-proof record of the agreed-upon terms.

5.2.2.2 Functional Requirements Specification (FRS): Building upon the BRD, BAs develop the FRS, which provides detailed specifications for the desired system functionalities. BAs work closely with stakeholders to elicit functional requirements, including specific features, user interfaces, data flows, and system behaviour. The FRS document serves as a blueprint for developers, guiding them in implementing the required functionalities. BAs ensure that the FRS aligns with the business objectives and accurately reflects the client's needs.

5.2.2.3 Technical Requirements Specification (TRS): In collaboration with technical stakeholders such as developers and architects, BAs contribute to the creation of the TRS. This document specifies the technical requirements, infrastructure, performance metrics, and integration guidelines for the software system. BAs work closely with technical teams to understand the feasibility of implementing certain features, identify technology dependencies, and define technical constraints. The TRS ensures that the system can meet the desired functional and performance requirements.

5.2.2.4 System Architecture Document: The System Architecture Document provides an overview of the software system's structure, components, and interactions. BAs collaborate with technical stakeholders, such as solution architects and system designers, to define the system's architecture. This includes identifying the system's modules, interfaces, data flows, and dependencies. BAs ensure that the system architecture aligns with the functional and technical requirements, enabling efficient development, integration, and scalability of the software solution.

A sample Solidity contract code is shown below to achieve the insertion of encrypted documents and optionally uploading to blob storage containers such as AWS S3 mentioned in the scope of Business Analyst in the blockchain:



```
pragma solidity ^0.8.0;
import "contracts/encryption/sha256.sol";
import "contracts/storage/s3.sol";

contract BusinessAnalystContract {
  struct BusinessRequirementsDocument {
    string title;
    string description;
  }
  struct FunctionalRequirementsSpecification {
    string title;
    string description;
  }
  struct TechnicalRequirementsSpecification {
    string title;
    string description;
  }
  struct SystemArchitectureDocument {
    string title;
    string description;
  }

  BusinessRequirementsDocument businessRequirementsDocument;
  FunctionalRequirementsSpecification functionalRequirementsSpecification;
  TechnicalRequirementsSpecification technicalRequirementsSpecification;
  SystemArchitectureDocument systemArchitectureDocument;

  constructor(
    string memory title,
    string memory description,
    string memory title2,
    string memory description2,
    string memory title3,
    string memory description3,
    string memory title4,
    string memory description4
  ) {
    businessRequirementsDocument = BusinessRequirementsDocument({title: title, description: description,..});
    functionalRequirementsSpecification = FunctionalRequirementsSpecification({title: title2, description: description2,..});
    technicalRequirementsSpecification = TechnicalRequirementsSpecification({title: title3, description: description3,..});
    systemArchitectureDocument = SystemArchitectureDocument({title: title4, description: description4,..});
  }

  function takeSnapshot(document) public {
    // Encrypt the document with SHA-256
    bytes memory encryptedDocument = sha256(document);
    // Store the encrypted document in the blockchain
    // Optionally, upload the encrypted document to blob container
    S3.upload(encryptedDocument);
  }

  function retrieveSnapshot(id) public view returns (bytes) {
    // Retrieve the encrypted document from the blockchain
    // Decrypt the document
    bytes memory decryptedDocument = sha256(id);
    // Optionally, download the encrypted document from blob container
    decryptedDocument = S3.download(id);
    return decryptedDocument;
  }
}
```

5.2.3 Product Manager, Developer, and DevOps Sub-Contracts for Agile and Sprint Records

In the proposed blockchain-based software supply chain solution, the Product Manager, Developer, and DevOps teams are crucial participants responsible for software development, delivery, and deployment. To ensure transparency and accountability during the Agile and sprint-based development lifecycle, sub-contracts are established to record and track their activities. These sub-contracts are integrated within the parent contract and contribute to the overall integrity of the software supply chain.

A sample of the parent contract maintaining these child contract in the blockchain in added below:

```
pragma solidity ^0.8.0;

import "contracts/encryption/sha256.sol";

contract DevSecOps {

  struct ProductManager {
    string title;
    string description;
  }

  struct Developer {
    string title;
    string description;
  }

  struct DevOpsCICD {
    string title;
    string description;
  }

  ProductManager productManager;
  Developer developer;
  DevOpsCICD devopsCICD;

  constructor(
    string memory title,
    string memory description,
    string memory title2,
    string memory description2,
    string memory title3,
    string memory description3
  ) {
    productManager = ProductManager({title: title, description: description,..});
    developer = Developer({title: title2, description: description2,..});
    devopsCICD = DevOpsCICD({title: title3, description: description3,..});
  }

  function takeSnapshot(contract) public {
    // Encrypt the contract with SHA-256
    bytes memory encryptedContract = sha256(contract);
    // Store the encrypted contract in the blockchain
  }

  function retrieveSnapshot(id) public view returns (bytes) {
    // Retrieve the encrypted contract from the blockchain
    // Decrypt the contract
    bytes memory decryptedContract = sha256(id);
    return decryptedContract;
  }
}
```

5.2.3.1 Product Manager Child Contract:

The Product Manager sub-contract is designed to capture the Agile and sprint records managed by the Product Manager throughout the software development process. It serves as a centralized repository for documenting project milestones, feature prioritization, and progress tracking. The sub-contract includes the following features:

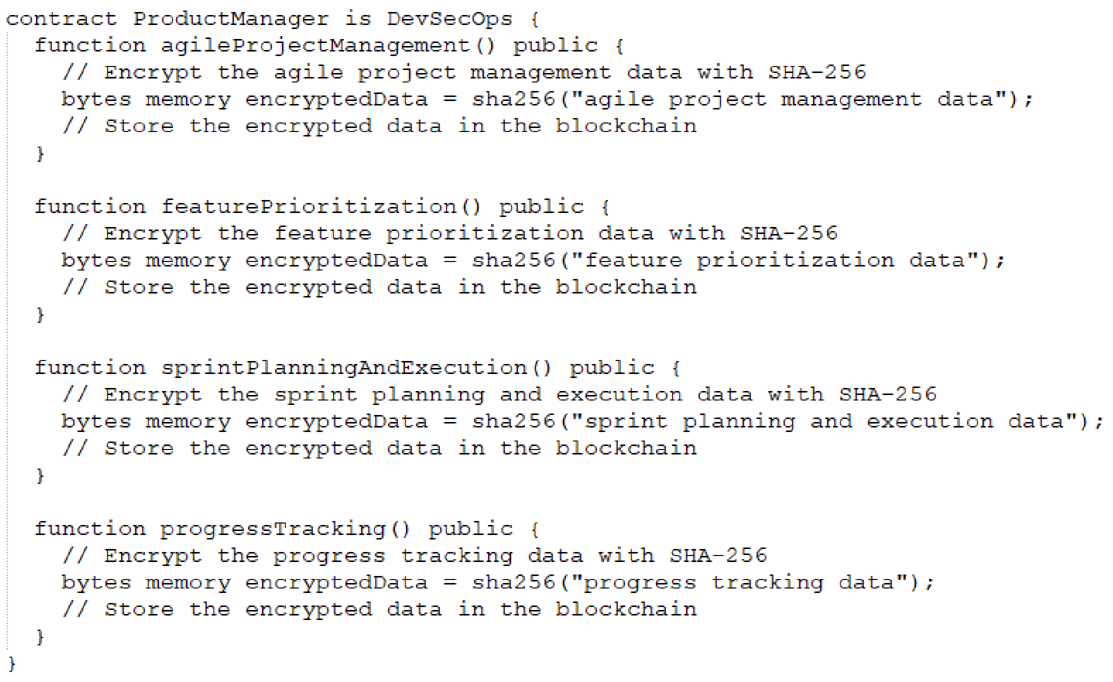
5.2.3.1.1. **Agile Project Management**: The sub-contract enables the Product Manager to define project goals, plan sprints, allocate resources, and track progress using Agile methodologies such as Scrum or Kanban.

5.2.3.1.2 **Feature Prioritization**: The Product Manager can outline and prioritize software features, ensuring that the development team focuses on the most critical requirements.

5.2.3.1.3 **Sprint Planning and Execution**: The sub-contract facilitates sprint planning, defining the tasks, timelines, and deliverables for each sprint. It also tracks the completion status of individual tasks.

5.2.3.1.4 **Progress Tracking**: The sub-contract captures the progress of each sprint, providing visibility into completed tasks, pending items, and any blockers or impediments faced during the development process.

The code snippet of the child contract is added below with reference to Product Manager:



```
contract ProductManager is DevSecOps {
  function agileProjectManagement() public {
    // Encrypt the agile project management data with SHA-256
    bytes memory encryptedData = sha256("agile project management data");
    // Store the encrypted data in the blockchain
  }

  function featurePrioritization() public {
    // Encrypt the feature prioritization data with SHA-256
    bytes memory encryptedData = sha256("feature prioritization data");
    // Store the encrypted data in the blockchain
  }

  function sprintPlanningAndExecution() public {
    // Encrypt the sprint planning and execution data with SHA-256
    bytes memory encryptedData = sha256("sprint planning and execution data");
    // Store the encrypted data in the blockchain
  }

  function progressTracking() public {
    // Encrypt the progress tracking data with SHA-256
    bytes memory encryptedData = sha256("progress tracking data");
    // Store the encrypted data in the blockchain
  }
}
```

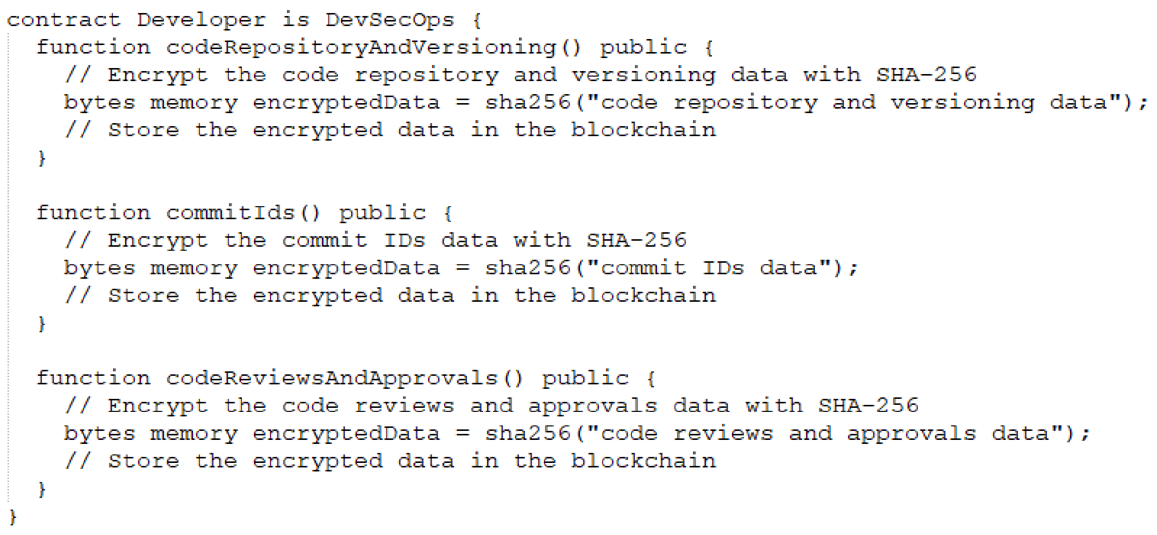
5.2.3.2 Developer Child Contract:

The Developer sub-contract focuses on the activities performed by individual developers involved in the software development process. It captures the code changes, testing activities, and other relevant information. Key features of the Developer sub-contract include:

5.2.3.2.1 **Code Repository and Versioning**: The sub-contract integrates with code repositories, such as Git, to store and manage code changes made by developers. It maintains a version history, ensuring traceability and accountability.

5.2.3.2.2 **Code Reviews and Approvals**: The sub-contract facilitates code review processes, allowing developers to submit their code changes for peer review. It captures feedback, suggestions, and approvals, ensuring quality control and adherence to coding standards.

The code snippet of the child contract is added below with reference to Developer:



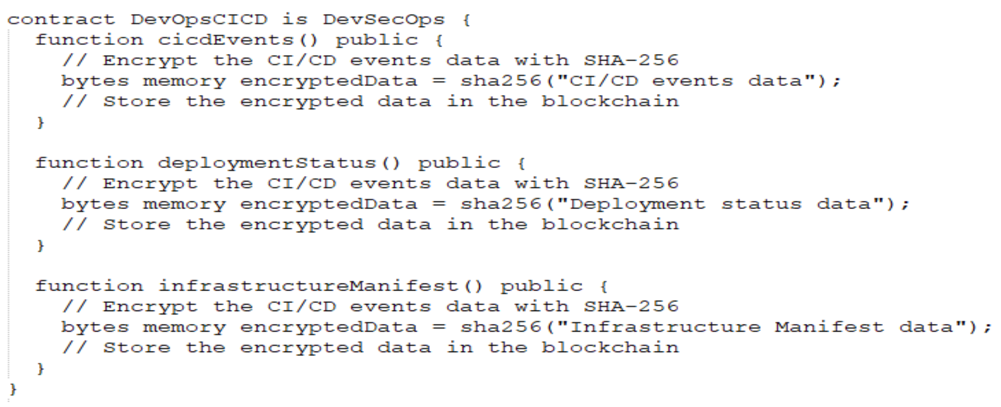
```
contract Developer is DevSecOps {
  function codeRepositoryAndVersioning() public {
    // Encrypt the code repository and versioning data with SHA-256
    bytes memory encryptedData = sha256("code repository and versioning data");
    // Store the encrypted data in the blockchain
  }

  function commitIds() public {
    // Encrypt the commit IDs data with SHA-256
    bytes memory encryptedData = sha256("commit IDs data");
    // Store the encrypted data in the blockchain
  }

  function codeReviewsAndApprovals() public {
    // Encrypt the code reviews and approvals data with SHA-256
    bytes memory encryptedData = sha256("code reviews and approvals data");
    // Store the encrypted data in the blockchain
  }
}
```

5.2.3.3 DevOps CI/CD Child Contract:

The DevOps CI/CD sub-contract focuses on the continuous integration, deployment, and delivery aspects of the software supply chain. It ensures seamless collaboration between the development and operations teams and facilitates efficient deployment processes. Key features of the DevOps CI/CD sub-contract include:

- **Continuous Integration and Continuous Deployment (CI/CD)**: The sub-contract tracks the integration and deployment activities carried out by the DevOps team. It captures information about build processes, test automation, and deployment pipelines, ensuring a smooth and automated release cycle.
- **Deployment Status and Rollback**: The sub-contract records the deployment status of each software release, including successful deployments, rollbacks, and any issues encountered during the process. It provides a transparent view of the deployment history and allows for quick identification and resolution of deployment-related issues.
- **Infrastructure and Configuration Management**: The sub-contract tracks the management of infrastructure resources and configuration settings necessary for software deployment. It ensures consistency and scalability by maintaining an auditable record of infrastructure changes.

The code snippet of the child contract is added below with reference to DevOps CICD:



```
contract DevOpsCICD is DevSecOps {
  function cicdEvents() public {
    // Encrypt the CI/CD events data with SHA-256
    bytes memory encryptedData = sha256("CI/CD events data");
    // Store the encrypted data in the blockchain
  }

  function deploymentStatus() public {
    // Encrypt the CI/CD events data with SHA-256
    bytes memory encryptedData = sha256("Deployment status data");
    // Store the encrypted data in the blockchain
  }

  function infrastructureManifest() public {
    // Encrypt the CI/CD events data with SHA-256
    bytes memory encryptedData = sha256("Infrastructure Manifest data");
    // Store the encrypted data in the blockchain
  }
}
```

5.2.4 Deployment Contract for Deployment Status and KPI Data

In the blockchain-based software supply chain solution, a Deployment Contract is utilized to track the deployment status and Key Performance Indicator (KPI) data of the software solution. This contract provides transparency and accountability in the deployment process, ensuring that all relevant stakeholders have access to real-time information regarding the deployment status and performance metrics.

5.2.4.1 Deployment Status Tracking: The Deployment Contract includes mechanisms to record and track the status of software deployments. It captures essential information such as the deployment date, target environment (e.g., development, staging, production), deployment method (e.g., manual, automated), and the individuals or teams responsible for the deployment process. This information is stored on the blockchain, providing an immutable record of each deployment event.

5.2.4.2 Key Performance Indicator (KPI) Data: The Deployment Contract also facilitates the collection and storage of KPI data related to the deployed software solution. KPIs are performance metrics that measure the success and effectiveness of the deployed software. They can include metrics such as response time, uptime, error rates, and user satisfaction. The Deployment Contract allows for the recording of KPI data in a standardized format, ensuring consistent and reliable measurement of software performance.

5.2.4.3 Real-time Updates: One of the advantages of utilizing blockchain technology is the ability to provide real-time updates on the deployment status and KPI data. The Deployment Contract can be programmed to receive and update information automatically from monitoring systems or external sources. This ensures that stakeholders have access to the most up-to-date information regarding the deployment progress and performance of the software solution.

5.2.5 Production Support Contract for CR Data

In the blockchain-based software supply chain solution, a Production Support Contract is utilized to track Change Request (CR) data during the production phase of the software solution. The contract enables efficient management and monitoring of CRs, ensuring timely resolution of issues and seamless maintenance of the software in the production environment.

5.2.5.1 Change Request (CR) Tracking: The Production Support Contract includes mechanisms to track CRs raised during the production phase. A CR represents a request for modifications or enhancements to the software solution to address issues or improve functionality. The contract captures essential information related to each CR, such as the CR number, description, priority, impact, and the individuals or teams responsible for handling the request.

5.2.5.3 Real-time Updates: Similar to other contracts in the blockchain-based solution, the Production Support Contract provides real-time updates on CR data. It allows stakeholders, including production support teams and clients, to access and monitor the status of CRs in real-time. Updates can include changes in the CR status, assignment to specific teams or individuals, progress updates, and resolution details. This real-time visibility enhances transparency and enables stakeholders to collaborate effectively in addressing CRs.

5.2.6 Data Encryption with SHA-256 and Combined Hashing

In the blockchain-based software supply chain solution, data encryption with SHA-256 hashing techniques with implementation of Merkle Tree are employed to ensure the security and integrity of the information stored within the blockchain. Let's explore these techniques in detail:

5.2.6.1 SHA-256 Hashing:

In the software supply chain solution, each component, such as client requirements, specifications, deployment status, KPI data, and CR information, can be hashed individually using SHA-256. This ensures that the integrity of each data element is maintained and any unauthorized modification can be identified. Furthermore, the generated hash values can be stored in the blockchain alongside the corresponding data, allowing for easy verification and detection of tampering attempts.

5.2.6.2 Combined Hashing with Merkle Tree:

A Merkle tree is a type of hash tree that is used in blockchain technology to verify the integrity of data. It is a data structure that can be used to efficiently store and verify the authenticity of large amounts of data.

In terms of data structure, it is a binary tree, where each node in the tree is a hash of the two nodes below it. The root node of the tree is a hash of all of the leaves in the tree. To verify the integrity of a data block, the hash of the data block is calculated and compared to the hash of the corresponding leaf node in the Merkle tree. If the hashes match, then the data block has not been tampered with.

Using the concepts behind Merkle trees, we can use Combined Hashing technique to verify the authenticity of blocks of data that are added to the blockchain. When a new block is added to the blockchain, its hash is calculated and stored in the Merkle tree. The hash of the new block is then combined with the hashes of the previous blocks in the blockchain to create a new root hash. The root hash of the new block is then stored in the blockchain.

A flowchart diagram on how Merkel Tree algorithm works and generates a Merkel root out of the leaf hashes:

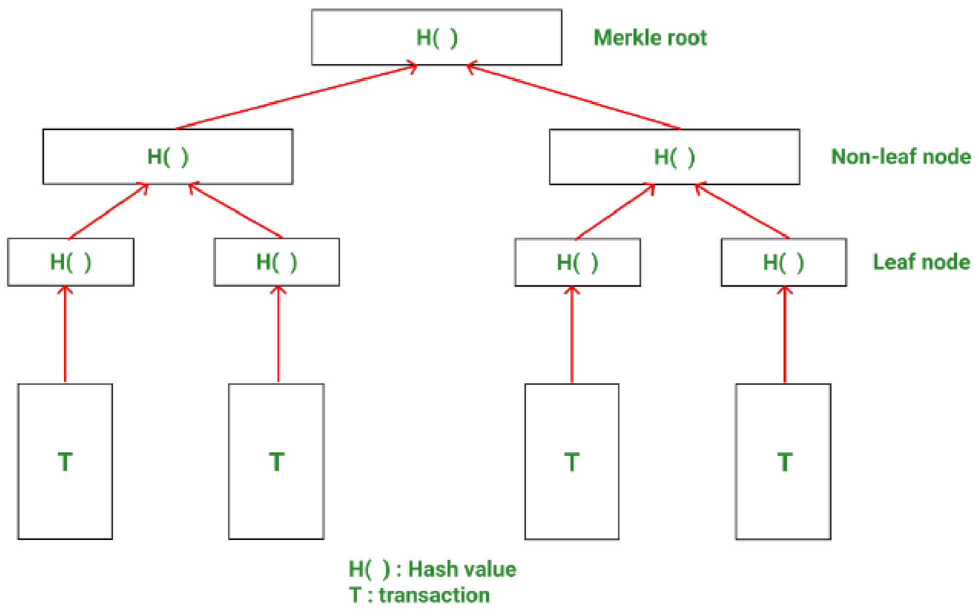


Fig 13: High level diagram of a Merkel Tree implementation of leaf node hashes

A sample Solidity contract is added below to visualize how we can encrypt data from each of the different child contracts into the parent contract and generate a Merkel root out of the hashes generated from the transactions executed by each validator nodes acting as participants:

```
pragma solidity ^0.8.0;

contract SupplyChain {

  struct Data {
    string data;
    uint256 encryptedData;
  }

  mapping(bytes32 => Data) private dataStore;

  function encryptData(string memory data) public {
    bytes32 hash = sha256(data);
    uint256 encryptedData = uint256(hash);
    dataStore[hash] = Data(data, encryptedData);
  }

  function decryptData(bytes32 hash) public view returns (string memory) {
    Data memory data = dataStore[hash];
    return data.data;
  }

  // Function to add clientStakeHolderContract data
  function clientStakeHolderContract(string memory requirements) public {
    bytes32 hash = sha256(requirements);
    encryptData(requirements);
  }

  // Function to create BDD documents
  function BusinessAnalystContract(string memory bddDocuments) public {
    bytes32 hash = sha256(bddDocuments);
    encryptData(bddDocuments);
  }

  // Function to store Agile and sprint records
  function devSecOpsContract(string memory agileAndSprintRecords) public {
    bytes32 hash = sha256(agileAndSprintRecords);
    encryptData(agileAndSprintRecords);
  }

  // Function to hold deployment status data
  function deploymentContract(string memory deploymentStatusData) public {
    bytes32 hash = sha256(deploymentStatusData);
    encryptData(deploymentStatusData);
  }

  // Function to hold production support data post deployment
  function productionSupportContract(string memory kpiDataPostDeployment) public {
    bytes32 hash = sha256(kpiDataPostDeployment);
    encryptData(kpiDataPostDeployment);
  }

  // Function to generate a sha value of the data it contains
  function generateShaValue(string memory data) public view returns (bytes32) {
    return sha256(data);
  }

  // Function to generate a sha value out of all the sha values calculated from all the above functions
  function generateFinalShaValue(dataStore) public {
    // Get the address of the child contract
    address merkelTreeGeneratorContract = 0x1234567890abcdef;
    merkelTreeGeneratorContract.call(dataStore);
  }
}
```

A sample contract is added that can help us implement the Merkel Tree algorithm to generate a Merkel root from the leaf nodes containing hashes of transactions executed by the participants in the blockchain:

```
contract MerkleTree {

  struct Leaf {
    uint256 hash;
  }

  struct Node {
    uint256 hash;
    Leaf left;
    Leaf right;
  }

  Leaf[] leaves;
  Node root;

  constructor() {
    leaves.push(Leaf({ <clientStakeHolderContract hash> }));
    leaves.push(Leaf({ <businessAnalystContract hash> }));
    leaves.push(Leaf({ <devSecOpsContract hash> }));
    leaves.push(Leaf({ <deploymentContract hash> }));
    leaves.push(Leaf({ <productionSupportContract hash> }));

    root = buildTree(leaves);
  }

  function buildTree(Leaf[] leaves) internal returns (Node) {
    if (leaves.length == 1) {
      return Node({ hash: leaves[0].hash });
    } else {
      var i = leaves.length >> 1;
      Node left = buildTree(leaves.slice(0, i));
      Node right = buildTree(leaves.slice(i));
      return Node({
        hash: keccak256(left.hash, right.hash),
        left: left,
        right: right,
      });
    }
  }

  function getRootHash() public view returns (uint256) {
    return root.hash;
  }
}
```

5.2.7 Integration with Tableau for Blockchain UI Dashboard:

Tableau is a popular data visualization and business intelligence tool that allows users to create interactive dashboards and reports. Integrating Tableau with the blockchain-based solution enables stakeholders to gain valuable insights and monitor the performance of the software supply chain in a user-friendly and visually appealing manner.

Here's an explanation of the integration with Tableau for the Blockchain UI Dashboard:

5.2.7.1 Data Extraction and Preparation: The first step is to extract the relevant data from the blockchain network. The blockchain stores data in a structured format, typically in the form of transactions or blocks. This data needs to be processed and transformed into a format suitable for visualization in Tableau. Depending on the blockchain platform used, APIs or custom scripts can be utilized to extract and prepare the data for integration with Tableau.

5.2.7.2 Connecting Tableau to the Blockchain Data: Tableau provides various connectivity options to different data sources, including databases, APIs, and files. To integrate with the blockchain data, Tableau can be connected to the prepared dataset. This connection enables Tableau to access and retrieve the required information from the blockchain network.

5.2.7.3 Data Visualization and Dashboard Creation: Once the connection is established, Tableau offers a range of data visualization capabilities to create interactive dashboards and reports. The stakeholders can design and customize the dashboard layout, incorporating charts, graphs, tables, and other visual elements to represent the software supply chain data effectively. The data can be presented in a format that provides insights into key metrics, trends, and performance indicators.

5.2.7.4 Real-time Data Updates: As the blockchain data is continually updated with new transactions and blocks, the integration with Tableau can be designed to provide real-time data updates on the dashboard. This ensures that stakeholders have access to the most current information and can monitor the software supply chain in real-time. The integration can leverage Tableau's data refresh capabilities to periodically fetch the latest data from the blockchain network.

5.2.7.5 Interactive Data Exploration: Tableau allows users to interact with the data on the dashboard, enabling drill-down capabilities and filtering options. This interactivity empowers stakeholders to explore the software supply chain data from different perspectives, gain deeper insights, and identify patterns or anomalies. Users can select specific time frames, filter data based on criteria, and analyze the performance metrics in a dynamic and intuitive manner.

5.2.7.6 Collaboration and Sharing: Tableau provides collaboration and sharing features that allow stakeholders to collaborate on the dashboard, annotate insights, and share the visualizations with other team members. This facilitates better communication and decision-making across the software supply chain stakeholders. The shared dashboards can be accessed through web browsers or Tableau's mobile applications, ensuring accessibility and convenience.

To integrate Tableau with a private Ethereum network to fetch the data in the blockchain, we will need to:

1. **Install the Tableau Connector for Ethereum**: The Tableau Connector for Ethereum is a free add-on that allows you to connect Tableau to a private Ethereum network. We can download the connector from the Tableau website.

2. **Create a connection to your private Ethereum network**: Once we have installed the connector, we need to create a connection to your private Ethereum network which can be achieved by adding the following information:

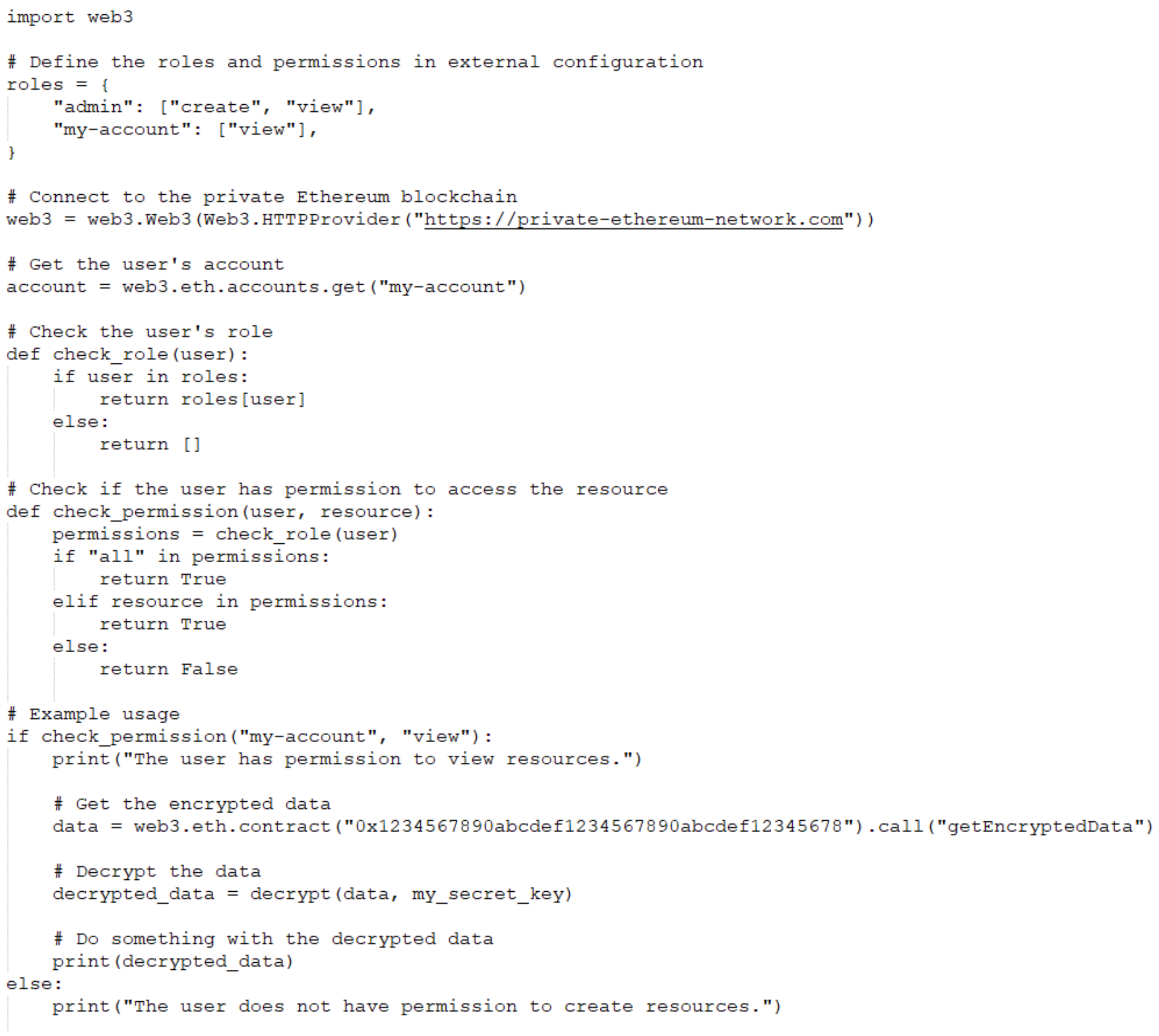
- The name of your network.
- The IP address of the node that is running the Ethereum client software.
- The port number that the node is listening on.
- The private key of a node that is authorized to access the network.

3. **Create a dashboard that visualizes the data from the blockchain**: Once we have created a connection to our private Ethereum network, we can start creating a dashboard that visualizes the data from the blockchain. We can use the Tableau Connector for Ethereum to query the blockchain and retrieve data and then use this data to create charts, graphs, and other visualizations.

5.2.8 API integration with the Blockchain

API integration can be useful in blockchain technology in a number of ways, including:

- **Enabling communication between different blockchain**: API integration can be used to enable communication between different blockchain. This can be useful for a variety of purposes, such as cross-chain transactions, asset transfers, and data sharing.
- **Making blockchain technology more accessible to developers**: API integration can make blockchain technology more accessible to developers. This is because APIs provide a standard way for developers to interact with blockchain, which can simplify the development process.
- **Enhancing the functionality of blockchain-based applications**: API integration can be used to enhance the functionality of blockchain-based applications. This is because APIs can be used to add new features and functionality to applications, such as the ability to interact with other applications or to access external data sources.
- **Improving the security of blockchain-based applications**: API integration can be used to improve the security of blockchain-based applications. This is because APIs can be used to implement security features, such as authentication and authorization, which can help to protect applications from unauthorized access.



```python
import web3

# Define the roles and permissions in external configuration
roles = {
    "admin": ["create", "view"],
    "my-account": ["view"],
}

# Connect to the private Ethereum blockchain
web3 = web3.Web3(Web3.HTTPProvider("https://private-ethereum-network.com"))

# Get the user's account
account = web3.eth.accounts.get("my-account")

# Check the user's role
def check_role(user):
    if user in roles:
        return roles[user]
    else:
        return []

# Check if the user has permission to access the resource
def check_permission(user, resource):
    permissions = check_role(user)
    if "all" in permissions:
        return True
    elif resource in permissions:
        return True
    else:
        return False

# Example usage
if check_permission("my-account", "view"):
    print("The user has permission to view resources.")

    # Get the encrypted data
    data = web3.eth.contract("0x1234567890abcdef1234567890abcdef12345678").call("getEncryptedData")

    # Decrypt the data
    decrypted_data = decrypt(data, my_secret_key)

    # Do something with the decrypted data
    print(decrypted_data)
else:
    print("The user does not have permission to create resources.")
```

5.2.9 Integration with Existing Systems and Tools

In the context of the blockchain-based software supply chain solution, integrating with existing systems and tools is essential to ensure smooth collaboration, interoperability, and leverage the functionalities provided by those systems

5.2.9.1 Integration of Blob Containers for Document Storage: The integration involves establishing a connection to the blob storage service and implementing the necessary functionality to upload, download, and manage the BRD, FRS, TRS documents. This integration ensures that the BDD documents are securely stored, easily accessible, and can be associated with the corresponding contracts or transactions within the blockchain.

5.2.9.2 Agile Jira Integration: For seamless collaboration and tracking of Agile development processes, the solution can integrate with Agile project management tools such as Jira. Jira provides features for creating and managing user stories, tasks, sprints, and other Agile artifacts. By integrating with Jira, the solution can synchronize the Agile development-related data with the blockchain.

The integration with Jira involves connecting to the Jira API and implementing functionality to extract relevant data, such as user stories, sprint details, and task progress. This data can be stored within the blockchain, either directly or through associated sub-contracts, ensuring transparency and traceability of the Agile development process.

5.2.9.3 Integration in DevSecOps:

5.2.9.3.1 Integration with Code Repository (GitHub):

GitHub is a widely used version control system that allows developers to collaborate on code repositories. Integrating with GitHub enables the solution to gather commit IDs associated with code changes made by developers. These commit IDs provide a reference to the specific code changes made during the software development lifecycle.

The integration with GitHub involves establishing a connection to the GitHub API and implementing functionality to extract relevant commit IDs associated with specific code repositories. The commit IDs can then be stored within the blockchain, ensuring transparency and traceability of the code changes made by developers.

5.2.9.3.2 Integration with Code Quality Scan Tools (Sonarqube) Results:

Sonar is a code quality management platform that analyses codebases and provides detailed reports on code quality, maintainability, and security. Integrating with Sonar allows the solution to capture code quality results and metrics generated during the static code analysis process.

The integration involves extracting code quality results and metrics from Sonar, such as code smells, technical debt, or security vulnerabilities. This information can be stored within the blockchain, providing stakeholders with insights into the overall code quality of the software supply chain. By integrating with Sonar, the solution can ensure that code quality remains a priority throughout the development lifecycle.

5.2.9.3.3 Integration with Binary Manager (JFrog Artifactory) for Dependencies Tracking:

Artifactory is a binary manager that manages artifact storage and distribution. Integrating with Artifactory allows the solution to track build information and dependencies associated with the software supply chain.

The integration involves capturing build information, such as build numbers, timestamps, and associated dependencies, from Artifactory. This data can be stored within the blockchain, providing a comprehensive record of the build process and its dependencies. By integrating with Artifactory, the solution ensures transparency and traceability of the artifacts used within the software supply chain.

5.2.9.3.4 Integration with SAST/DAST (Blackduck/Checkmarx) tools for Scan Results:

Blackduck and Checkmarx are security scanning tools that analyze codebases for open-source vulnerabilities and security flaws. Integrating with these tools enables the solution to capture scan results and security-related findings.

The integration involves extracting scan results from Blackduck or Checkmarx and storing them within the blockchain. This allows stakeholders to have visibility into the security posture of the software supply chain and take appropriate actions to address any identified vulnerabilities or flaws. By integrating with these security scanning tools, the solution enhances the security and trustworthiness of the software supply chain.

5.2.9.3.5 Integration with Deployment Tools (UCD/Harness) for Deployment Reports and Signature Signing Information:

For effective deployment management, the solution can integrate with deployment tools such as IBM UrbanCode Deploy (UCD), Harness or similar tools. These tools facilitate the automation and orchestration of deployment processes and provide detailed reports on deployment status and progress.

The integration involves capturing deployment reports from deployment tools, which include information such as deployment timestamps, success/failure statuses, and associated artifacts.

In summary, integrating with existing systems and tools for DevOps ensures that relevant data from these tools, such as commit IDs, code quality results, build information, scan results, deployment reports, and signature signing information, is captured and stored within the blockchain. This integration enhances transparency, traceability, and security within the software supply.

5.2.9.4 Integration with Key Performance Indicator (KPI) Tools:

To monitor and track the performance of the software supply chain, the solution can integrate with KPI tools. Examples of KPI tools include Grafana, Splunk, or custom-built dashboards. These tools enable the visualization and analysis of various metrics and KPIs relevant to the software supply chain, such as deployment success rates, code quality metrics, or customer satisfaction scores.

The integration with KPI tools involves extracting the necessary data from the blockchain, such as deployment status, code quality records, or customer feedback, and feeding it into the KPI tool for visualization and analysis. This integration allows stakeholders to monitor the performance of the software supply chain and make data-driven decisions based on the insights provided by the KPI tools.

By analysing the KPI data and identifying areas for improvement, organizations can iterate on their software supply chain processes and make data-driven decisions to optimize performance, quality, and customer satisfaction. The blockchain-based solution ensures that the historical KPI data is securely stored and can be used for trend analysis and continuous improvement initiatives.

5.2.9.5 Integration with Change Request (CR) Management Tools:

To effectively manage change requests in the production environment, the solution can integrate with CR management tools such as ServiceNow or Jira Service Management. These tools provide capabilities for creating, tracking, and managing change requests, ensuring proper governance and control over the production environment.

The integration with CR management tools involves synchronizing the CR data between the blockchain and the CR management tool. This ensures that the CR information, including details such as request status, impact analysis, and approval history, is recorded within the blockchain and accessible to relevant stakeholders. It allows for transparency, traceability, and streamlined management of CRs within the software supply chain.

6. Security Considerations

Security considerations play a crucial role in ensuring the integrity, confidentiality, and availability of the system. This section highlights key security considerations that need to be addressed during the implementing of an internal Blockchain for Software development lifecycle:

6.1 Immutable Ledger: The immutability of the blockchain ledger ensures that once data is recorded, it cannot be altered or deleted. This feature enhances the integrity and tamper-resistance of the recorded information. However, it is essential to carefully validate and verify data before it is added to the blockchain to prevent malicious or erroneous data from being permanently stored.

6.2 Access Control: Access control mechanisms must be implemented to ensure that only authorized participants can interact with the blockchain-based software supply chain system. User authentication and authorization processes, such as digital signatures or cryptographic keys, can be used to control access to the system and validate participants' identities.

6.3 Secure Smart Contracts: Smart contracts should be thoroughly audited and tested to identify and mitigate vulnerabilities. Best practices, such as secure coding techniques, code review processes, and the adoption of standardized libraries, can enhance the security of smart contracts. Regular updates and patching of smart contracts should be carried out to address newly identified vulnerabilities.

6.4 Encryption and Data Privacy: Sensitive information, such as personally identifiable information (PII) or proprietary data, should be encrypted to ensure confidentiality. Encryption mechanisms, such as asymmetric encryption or homomorphic encryption, can be utilized to protect data stored on the blockchain or transmitted between participants.

6.5 Network Security: The underlying network infrastructure supporting the blockchain network should be secured to prevent unauthorized access and protect against attacks, such as Distributed Denial of Service (DDoS) attacks or network eavesdropping. Implementing secure communication protocols, firewalls, and intrusion detection systems can help safeguard the network.

6.6 Smart Contract Auditing: Regular auditing and code reviews of smart contracts should be conducted to identify and address any potential vulnerabilities or weaknesses. Third-party security audits by reputable firms can provide an additional layer of assurance and help uncover any security flaws.

6.7 Consensus Mechanism: The consensus mechanism used in the blockchain network should be carefully chosen to ensure the security and fault-tolerance of the system. Consensus algorithms, such as proof-of-work (PoW) or proof-of-stake (PoS), should be evaluated based on factors like network scalability, energy efficiency, and resistance to attacks.

6.8 Disaster Recovery and Backup: Adequate measures should be in place to ensure the availability and recoverability of the blockchain-based software supply chain system. Regular backups of the blockchain data and redundancy of network infrastructure can help mitigate the impact of system failures or catastrophic events.

7. Future Directions

7.1 Smart Contracts and Automation: Smart contracts, powered by blockchain technology, have the potential to automate various business processes and eliminate the need for intermediaries. In the future, organizations can explore more advanced and complex smart contracts to automate tasks such as payment processing, compliance verification, and asset management. Integration with artificial intelligence and machine learning algorithms can further enhance the capabilities of smart contracts, enabling autonomous decision-making and execution.

7.2 Decentralized Identity Management: Identity management is a critical aspect of organizations' operations, and blockchain offers a decentralized and secure solution for managing digital identities. In the future, blockchain-based identity management systems can enable individuals to have full control over their personal data and selectively share it with trusted entities. This can lead to more secure and privacy-preserving identity verification processes, reducing the risk of identity theft and fraud.

7.3 Sustainability and Green Initiatives: Blockchain can contribute to sustainability efforts by enabling transparent and efficient tracking of environmental impact data, supply chain carbon footprints, and renewable energy transactions. Future directions may involve integrating blockchain with IoT devices and sensors to collect real-time data on energy consumption, waste management, and carbon emissions. This data can then be securely stored on the blockchain, providing stakeholders with accurate information for making sustainable decisions.

8. Conclusion

The implementation of blockchain technology in the software supply chain brings numerous benefits and transformative possibilities for organizations. Throughout this white paper, we have explored the use of blockchain as a distributed ledger to provide transparency, security, and efficiency in the software supply chain process.

We began by discussing the importance of a transparent and reliable supply chain, where all participants, including vendors, business analysts, product managers, developers, DevOps teams, deployment and production support teams, can collaborate seamlessly. By leveraging blockchain technology, we can ensure that each participant has access to an immutable and shared ledger, which enhances transparency and accountability.

Furthermore, we explored the integration of key performance indicator (KPI) tools, such as Jira and other KPI tracking systems, to gather relevant metrics and data related to the software development lifecycle. This integration allows for real-time monitoring of project progress, quality metrics, and performance indicators, enabling data-driven decision-making and continuous improvement.

The utilization of blockchain technology also addresses security concerns by employing data encryption techniques and SHA-256 hashing. By encrypting sensitive data and generating unique hash values for each transaction, we can safeguard the integrity and privacy of information stored on the blockchain.

In conclusion, the implementation of blockchain technology in the software supply chain offers a range of benefits, including enhanced transparency, security, efficiency, and collaboration among stakeholders. By leveraging smart contracts, integrating with existing systems and tools, and ensuring data encryption and hashing, organizations can achieve a robust and reliable software supply chain process.

9. **Bibliography**

- Buterin, V. (2013). Ethereum Whitepaper. Retrieved from https://ethereum.org/whitepaper
- C. K. Chang, M. H. Chen, R. Chen, "A blockchain-based approach to enhancing software supply chain security", 2017 IEEE International Conference on Communications (ICC), May 2017
- B. Li, T. Li, K. L. Man, L. Shu, Z. Zhang, "Blockchain-based software engineering: Challenges and opportunities", Future Generation Computer Systems, Volume 95, December 2019
- R. B. Almeida, C. R. G. Sousa, T. Batista, "Blockchain for software supply chain management: A systematic literature review", 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), May 2019
- L.M. Yu, J.W. Lin, L.C. Hsieh, "A blockchain-based framework for supply chain visibility and traceability", International Journal of Information Management, Volume 102, October 2020
- M. Ivanov, D. L. Dolgui, A. Sokolov, "The impact of digital technology and Industry 4.0 on the ripple effect and supply chain risk analytics", International Journal of Production Research, Volume 56, Issue 1-2, January 2018
- M. Truby, M. Carboni, J. F. Renz, A. M. Spahr, S. S. Bandyopadhyay, "Blockchain for traceability in the food supply chain", Manufacturing Letters, Volume 20, December 2019
- L. M. S. Oliveira, M. D. F. Souza, M. V. L. Monteiro, "Blockchain technology in logistics: A literature review and future research directions", Computers in Industry, Volume 115, June 2020
- W. Wang, X. Jin, J. Sun, J. Yao, "Blockchain-based decentralized management of demand response programs in smart grids", IEEE Transactions on Smart Grid, Volume 10, Issue 1, January 2019
- F. Qian, H. Yu, F. Zeng, M. Zhang, L. Wang, "Towards trust and transparency in software supply chain with blockchain", 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), July 2018