ForTrac: a Secure NFT-based Forward Traceability System for Providing Data Accuracy and Completeness

Fokke Heikamp
folmer.heikamp@research.deakin.edu.au
Deakin University, Centre for Cyber Resilience and Trust (CREST)
Geelong, Australia
Cyber Security Cooperative Research Centre
Perth, Australia

Lei Pan
l.pan@deakin.edu.au
Deakin University, Centre for Cyber Resilience and Trust (CREST)
Geelong, Australia
Cyber Security Cooperative Research Centre
Perth, Australia

Robin Doss
robin.doss@deakin.edu.au
Deakin University, Centre for Cyber Resilience and Trust (CREST)
Geelong, Australia
Cyber Security Cooperative Research Centre
Perth, Australia

Rolando Trujillo-Rasua
rolando.trujillo@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain
Cyber Security Cooperative Research Centre
Perth, Australia

Sushmita Ruj
sushmita.ruj@unsw.edu.au
University of New South Wales
Sydney, New South Wales, Australia
Cyber Security Cooperative Research Centre
Perth, Australia

ABSTRACT

Traceability is one of the most sought-after properties for supply chains because it facilitates a plethora of use cases like food recalls, counterfeit detection, and quality improvement by tracing the history of a product. Recently proposed traceability solutions rely heavily on blockchain technology because it offers data immutability and transparency. However, these solutions suffer from two significant limitations. Firstly, they do not validate traceability data for accuracy and completeness. So, how can we be sure that the data on the blockchain is correct and complete? Secondly, they provide a limited threat model or none at all. So, how can we be sure that the trace for a product has not been altered by an adversary?

To summarise, we propose ForTrac, an Ethereum-based forward traceability system for generic physical products. Each product is digitally represented as an ERC-721 token. In a forward traceability system, traces are built correctly by design. ForTrac uses a valid predicate that states the sequence of steps a product has to follow and how much time each step can maximally take.

We show that ForTrac is secure. Unlike previous work, our security analysis is not restricted to only the blockchain component. It also includes the communication protocol between the product reader, server, and smart contract. We show that our communication protocol guarantees data agreement between the reader and smart contract using Scyther. Finally, we demonstrate that our prototype is feasible by assessing costs and performance.

By using forward traceability and including a detailed security analysis we provide accuracy and completeness. Our valid predicate documents which events have to be collected for a trace to be complete. Data accuracy is provided by showing that the smart contract agrees with the data sent by the reader.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
BSCI '23, July 10, 2023, Melbourne, VIC, Australia
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0198-6/23/07…$15.00
https://doi.org/10.1145/3594556.3594608

CCS CONCEPTS

• **Security and privacy** → **Distributed systems security**; *Security protocols*; Formal security models; • **Applied computing** → **Supply chain management**; • **Computer systems organization** → *Peer-to-peer architectures.*

KEYWORDS

Blockchain, Smart Contracts, Supply Chain, Forward Traceability, Security Analysis, Scyther

ACM Reference Format:
Fokke Heikamp, Lei Pan, Robin Doss, Rolando Trujillo-Rasua, and Sushmita Ruj. 2023. ForTrac: a Secure NFT-based Forward Traceability System for Providing Data Accuracy and Completeness. In *The 5th ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI '23), July 10, 2023, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3594556.3594608

1 INTRODUCTION

The GS1 traceability standard defines traceability as "the ability to trace the history, application or location of an object" [11]. It is a much sought-after property in supply chains because it is known as an enabler technology [10]. Traceability helps improve supply chain efficiency [10, 11, 22], quality [10, 11], and security [11, 18, 29] among others. It is also used to detect counterfeits [10, 17, 30, 33] and functions as a component in supply chain provenance systems [15, 33]. The trend of the last few years is to implement traceability systems using blockchain technology because it offers data immutability and transparent computing through

smart contracts [13, 16, 28]. Many commercial traceability systems have adopted blockchain technology, like Everledger [9] and TradeLens [31]. It is also evidenced by a plethora of research articles on the subject of blockchain and supply chain traceability [8, 12, 16–18, 26, 28, 30, 32, 33].

However, blockchain does not solve all the challenges for supply chain traceability [25]. The GS1 traceability standard mentions four data quality aspects, including completeness, accuracy, consistency, and validity [11]. Consistency refers to the property that different systems agree on the same piece of data. Validity implies that all events are properly timestamped. Blockchain facilitates these two properties by design. Accuracy refers to if the data captured accurately reflects what happened. Completeness means that all data has been captured. Blockchain does not meet these properties by default. Current research focuses on how blockchain is used to collect traceability data, omitting how data accuracy and completeness are achieved.

Furthermore, most related work does not provide a security analysis; if it does, it only focuses on the blockchain aspect. We argue that all work on traceability systems should include a complete threat model and security analysis. It is well known that supply chains are common targets of attack [13, 16, 18]. It is also known that traceability helps detect these attacks [11, 29]. So, a traceability solution needs to be secure. The argument that data on the blockchain is immutable is not enough: the adversary could have influenced the data before it is submitted to the blockchain. Also, in case smart contracts are used, they could be malicious or contain vulnerabilities. So, a threat model and security analysis are necessary and should be applied to the whole traceability system, from the capture point to the blockchain.

So, our research questions are: "how can we design a traceability system that provides data accuracy and completeness" and "how do we provide evidence that our complete traceability solution is secure."

Our approach to improving data accuracy and completeness is by using forward traceability [14]. Forward traceability specifies that an event can only be added to the trace if a valid predicate evaluates it successfully. For our work, we use a path and time-based valid predicate that states the sequence of participants the product has to follow and how much time they have to complete each step. This approach is akin to the valid specification in the World Wide Web Consortium provenance recommendation [21], indicating its relevance. Our choice for this predicate is that it is the most simple and logical. Other valid predicates are not excluded.

We do a security analysis on our system to show evidence that it is secure. The security analysis focuses on the smart contract and the communication protocol between the reader, server, and smart contract. The analysis is done using informal logical reasoning. We also use Scyther to show that the data is not tampered with in the network.

Our work gives supply participants a mechanism to securely validate traceability data. Our solution provides data accuracy because the valid predicate dictates which events have to be collected. It also provides data accuracy because we show that the reader and smart contract agree on the collected data. This assumes that the reader is honest and collects product information correctly. The valid predicate also improves data accuracy because it limits the number of possible events at each state.

In this paper, our contributions are as follows:

- We extend the definition of forward traceability [14] and propose a more realistic valid predicate based on path and time constraints.
- We develop ForTrac: a forward traceability solution that uses non-fungible tokens (NFT) on the Ethereum network.
- We provide a detailed security analysis for our system.
- We experiment with performance and cost.

In Section 2 we will review related work. Section 3 defines forward traceability based on transition systems. It also defines the valid predicate we use in our solution. Our NFT-based solution is presented in Section 4. In Section 5, we show that our system achieves forward traceability. Section 6 contains our informal and formal security analysis. Section 7 contains the performance and cost analysis. Section 8 will summarize our contributions, provide future work, and conclude our article.

2 RELATED WORK

The academic literature shows an abundance of traceability solutions [18, 32, 33]. Some of these solutions are generic, while others are targeted at specific types of supply chains, like the food supply chain [24]. In recent years blockchain technology has been an important component in traceability solutions. The choice of blockchain is understandable. When reading a trace for a product, the user needs to know which entities helped create the trace and that it has not been tampered with. Blockchain offers both properties by providing data immutability and data integrity and authenticity through signatures. However, what blockchain cannot guarantee is data accuracy and completeness. Current work usually neglects these properties and only provides a method for collecting traceability data without verifying its completeness and accuracy. Furthermore, most works lack a threat model or focus only on the blockchain aspect.

Westerkamp et al. [33] developed an NFT-based traceability solution for modifiable goods. They introduce the concept of recipes to convert several input NFTs into an output NFT. The authors claim to achieve traceability because it is auditable but do not specify how an end-user collects such a trace. Their solution provides data accuracy and completeness about how a product is manufactured but not about the traceability of the product. Their work does not consider a threat model.

Kuhn et al. [17] propose a traceability solution for the assembly supply chain using ERC-1155 tokens. They implement a proof-of-concept using Ethereum smart contracts. They do not provide any security analysis or requirement, making it vulnerable to attacks. They do not impose constraints on how a trace evolves or explore alternative ways of validating data accuracy or completeness.

Toyoda et al. [30] propose a product ownership management system (POMS) based on smart contracts to combat counterfeiting. They use a hardcoded static supply chain, which can be used to determine data completeness. Hardcoded supply chains make the applicability to other supply chains limited. They provide a limited security analysis by focusing on impersonation attacks.

Sezer et al. [26] propose a conceptual and privacy-preserving blockchain-based traceability model. They provide a proof-of-concept in Ethereum and experiment with costs. Their security analysis is partial and mainly focuses on the privacy aspect of the smart contract. They do not consider the validation of traceability data.

Sunny et al. [28] conduct a literature study on existing traceability solutions. They also provide a proof-of-concept solution without security analysis. They achieve data completeness using pre-defined business logic at the cost of flexibility.

Heikamp et al. [14] introduce the concept of forward traceability and give an implementation in Ethereum. However, their proposed valid predicate only works on path constraints. They do include a minimal security argument but it is incomplete and is focused on the smart contract. We extend this work by using a valid predicate based on paths and time intervals and provide a structured security analysis.

Lou et al. [18] propose a secure and efficient blockchain-based supply chain framework (SESCF) that provides traceability. Their solution is based on a series of smart contracts that define the business logic for a specific supply chain. They include a security analysis but only focus on smart contracts.

Islam et al. [16] provide a solution that combines physically unclonable functions (PUF), radio frequency identifiers (RFID), and blockchain. However, their solution does not achieve traceability, it merely connects physical devices (tags) to the blockchain, which is a useful component for traceability. Their security analysis is informal and only focuses on the blockchain aspect.

To solve the data accuracy and completeness problem we built on the concept of forward traceability. Forward traceability requires a valid predicate that needs to be satisfied for the trace to be built. This improves accuracy and completeness by guaranteeing that the traces are built according to expectations. Furthermore, we provide an informal security argument for both the smart contract and the communication protocol. Verification tools help us ensure that there are no known attacks. A comparison of the important related work with this paper is provided in Table 1. The comparison shows what building block the author used, if it contains a threat model and if it provided data accuracy and completeness.

3 FORWARD TRACEABILITY

We define a traceability system as a transition system. Our system specifies three key elements: P denotes the set of all products, E is the set of all events, and S is the set of all states. A state s is a partial mapping from P to E^*, denoting a sequence of events that led to this state. The notation $s(p) = (e_1, \ldots, e_n)$ indicates that the events e_1 to e_n turned product p from the starting state into its current state.

Definition 3.1 (Supply chain traceability system). A supply chain traceability system is a transition system $(S, \rightarrow)$ where:

- S is a set of states. Each $s \in S$ is a partial mapping from P to E^*.
- A transition relation, denoted by $\rightarrow$, describes how the system progresses. $\rightarrow$ always increases in steps of one event at a time. So, if we have $s \rightarrow s'$, it implies that if $s(p) = (e_1, \ldots, e_i)$ then $s'(p) = (e_1, \ldots, e_i, e')$, for some product $p \in P$ and event $e' \in E$.

Our forward traceability extends a conventional traceability system with the assumption that the state transition relation $\rightarrow$ depends on a *valid* predicate. The predicate $valid(s)$ checks whether a state s is valid. For our definition, we keep the valid predicate abstract. The reason for this is that there are a lot of conceivable valid predicates. The suitability of a certain valid predicate might depend on the context. By keeping it abstract, our definition works on all conceivable valid predicates. Definition 3.2 defines a forward traceability system.

Definition 3.2 (Supply chain forward traceability system). A supply chain forward traceability system is a special type of supply chain traceability system where the transition relation $\rightarrow$ is based on a valid predicate. That means $s \rightarrow s'$ implies that if $s(p) = (e_1, \ldots, e_i) \wedge valid(s)$ then $s'(p) = (e_1, \ldots, e_i, e') \wedge valids(s')$, for some product $p \in P$ and event $e' \in E$. This can be read as: given a `valid` state s, it can transition into another state s' if the trace increases by 1 step and if state s' is also `valid`.

Intuitively, the forward traceability system moves forward in steps of one event and only if the *valid* predicate allows it. Figure 1 shows an example forward traceability system where events a, b, and c are applied to the product p.

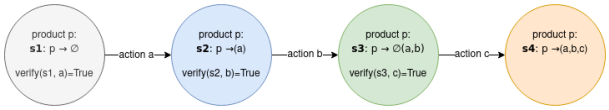


Figure 1: An Example Forward Traceability System for a Product p

To implement a forward traceability system, we need to define a concrete *valid* predicate. For our proof-of-concept, we use a predicate that defines the path a product has to take and how much time each step can maximally take. When a manufacturer creates a product it has a good idea of what path a product will take. This is especially true in certain types of supply chains like the built-to-order supply chain [3]. Because a manufacturer has experience with creating and delivering products, it also knows how much time an average product takes at each step. So, if a product is processed by an unknown actor, or not processed within the specified time interval, it represents an anomaly. By taking these two properties (time and identity) into account, we create a simple but effective valid predicate.

We represent an event using a tuple consisting of 'actor', 'data', and 'timestamp'. We denote the actor of an event as $actor(e)$ and the timestamp of an event as $timestamp(e)$. Our valid predicate is defined in Definition 3.3.

Definition 3.3 (Path and time based valid predicate). Given a path $a_1, \ldots, a_n$, a sequence of time intervals $t_1, \ldots, t_{n-1}$, and a state s for product p, we say that a state is valid iff $|s(p)| \leq n$ and for each event e_i in $s(p)$, $actor(e_i) = a_i$ and for each e_i in s, where $i > 0$, $timestamp(e_i) - timestamp(e_{i-1}) < t_i$.

4 FORTRAC

We design and implement a proof-of-concept forward traceability system using Solidity smart contracts to demonstrate the system's

Table 1: A Comparison of Related Work

✓: yes, **O**: partial, **X**: no				
Author	**Building Blocks**	**Data Accuracy**	**Data Completeness**	**Threat Model**
Westerkamp et al. [33]	ERC-721	O	O	X
Kuhn et al. [17]	ERC-1155	X	X	X
Toyoda et al. [30]	Smart Contract	X	✓	O
Sezer et al. [26]	Smart Contract	X	X	O
Sunny et al. [28]	Smart Contract	X	✓	X
Heikamp et al. [14]	Smart Contract	O	✓	O
Lou et al. [18]	Smart Contract	X	✓	O
Islam et al. [16]	Smart Contract	✓	X	O
This paper	ERC-721	✓	✓	✓

feasibility. We use the time and path constraint-based valid predicate from Definition 3.3.

In our scenario, we define a supply chain as a sequence of participants, from manufacturer to consumer. Each participant has a back-end server and at least one product reader. In our proof-of-concept we only consider one reader per participant to keep the design as simple as possible. It would be trivial to extend this to multiple readers per participant. The product reader reads the information from the product. We stay technology agnostic about what kind of readers are being used and assume that they can correctly identify the product. Real-life systems could use radio frequency identification (RFID), near-field communication (NFC), or bar-codes, all of which are well known for their use in supply chains. The back end communicates with the smart contract and reader. Figure 2 shows a basic scenario. Our system is built under four assumptions. Firstly, we assume that a reader always identifies a product accurately so that product or tag spoofing is always detected. This assumption implies product authentication. In real-life systems, product authentication is typically weakened to tag authentication, according to a few available solutions [1, 27]. Secondly, we assume proper access control is in place so that only the relevant parties can create smart contracts and interact with them. Thirdly, we assume that both the reader and server have a key pair for encrypting and signing messages. Fourthly, our system assumes perfect encryption, meaning that the encryption schemes cannot be broken. A list of symbols used in the rest of this paper is given in Table 2.

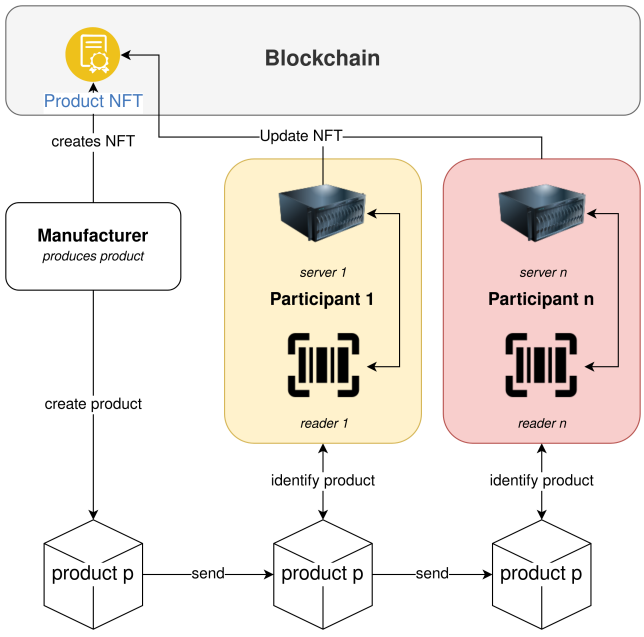


Figure 2: ForTrac Scenario

Table 2: Symbols and Explanations

Descriptor	Description
R	Reader
C	Smart Contract
S	Server
pid	Product identifier
$pk(S)$	Public key of server S. Encryption is denoted as $\{data\}pk(S)$
$sk(S)$	Secret key of server S. Signing is denoted as $\{data\}sk(S)$
$pk(R)$	Public key of server R. Encryption is denoted as $\{data\}pk(R)$
$sk(R)$	Secret key of server R. Signing is denoted as $\{data\}sk(R)$

4.1 Communication Protocol

Our smart contract is purely digital, however, it documents the flow of a physical product. It is well-known that there are difficulties representing physical goods as NFTs [2, 23] because it is challenging to keep the state of the physical and the digital object synchronized. So, we have to consider how to insert the data of physical goods accurately into the smart contract. Assuming that communication between the reader and product is secure and correct, it becomes a three-party protocol between the reader, server, and the smart contract.

The smart contract needs to achieve data agreement with the reader. We know something is wrong if there is a mismatch between what the reader collects and what the smart contract stores. We can

provide data agreement by ensuring that the reader is authenticated to the smart contract.

We designed a communication protocol that provides mutual authentication between the reader and server using the public key Needham-Schroeder-Lowe (NSL) protocol [19]. Furthermore, we provide authentication between the reader and smart contract, and the server and smart contract by including a message that contains a nested signature by the reader and server respectively. We enforce checking if every message generated is fresh to prevent replay attacks. The complete overview of the communication protocol is provided in Table 3. The random nonces, c_1, c_2, and c_3, should only be used once. Furthermore, C, S, and R, refer to the identities of the smart contract, server, and reader respectively.

Our protocol does not achieve secrecy/confidentiality between the server and smart contract because we modelled the message sequence based on how Ethereum works. Ethereum, by default, does not provide confidentiality since one of the main objectives is transparency.

4.2 Smart Contract

In our system, the smart contract stores the traces and product information. The products in our system are represented as non-fungible tokens complying with the ERC-721 NFT standard. NFTs are unique digital objects that cannot be duplicated. NFTs also ensure a unique digital representation of each physical object. We implemented the NFT mechanism using the OpenZeppelin ERC721 library. By using a well-known and audited 3rd party library, we limit coding mistakes. Our smart contract has three main functionalities: creating NFTs, updating traces, and trace verification.

When creating an NFT, the manufacturer specifies the product identifier, product name, the pre-defined path it has to take, and the time intervals it should be processed. The product identifier is unique by using codes like the electronic product code (EPC) [4]. The smart contract simply checks if a product with this identifier already exists. The product name is a non-empty string. The path is an address sequence representing the participants the product has to follow in order. The time intervals specify how much time each step can maximally take.

A trace can be updated by calling the 'update' method. The 'update' method expects a product identifier and some event data. The smart contract checks if the product identifier exists. If the product identifier exists, the smart contract checks whether the caller is authorized to process the product by checking if the caller is defined in the next step in the path. The smart contract also checks if the difference between the current and previous timestamp is within the predefined time interval for this step. If all checks hold, the event is appended to the trace.

The complete smart contract in Solidity is provided in Listing 1.

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract ProductNFT is ERC721("Product", "PRODUCT") {

    struct Event {
        address participant;
        string data;
        uint timestamp;
    }

    struct Product {
        string name;
        Event[] trace;
        address[] path;
        uint[] intervals;
    }

    mapping (uint => Product) private id2product;

    function createProduct(uint productId, string memory _name, address[] memory _path, uint[] memory _intervals) public returns (uint256) {
        require(bytes(id2product[productId].name).length == 0, "cannot register a product with the same ID");
        require(_intervals.length == _path.length - 1, "intervals should be of length path - 1");
        require(bytes(_name).length != 0, "name should not be empty");

        Product storage product = id2product[productId];
        product.name = _name;
        product.path = _path;
        product.intervals = _intervals;

        _safeMint(msg.sender, productId);
        return productId;
    }

    function verify(uint productId, uint depth) view private returns(bool) {
        Product storage product = id2product[productId];
        for(uint i = product.trace.length - depth; i < product.trace.length; i++) {
            require(product.trace[i].participant == product.path[i], "trace and path don't match");
            if(i != 0) {
                require((product.trace[i].timestamp - product.trace[i - 1].timestamp) < product.intervals[i - 1], "should have been processed within the time interval");
            }
        }
        return true;
    }

    function verify(uint productId) view public returns(bool) {
        Product storage product = id2product[productId];
        return verify(productId, product.trace.length);
    }

    function update(uint productId, string memory data) public returns(bool) {
        Product storage product = id2product[productId];
        require(product.trace.length != product.path.length, "trace ended");
        product.trace.push(Event(msg.sender, data, block.timestamp));
        require(verify(productId, 1), "event cannot be added");
        return true;
    }
```

Table 3: ForTrac's Communication Protocol

Reader $R[sk(R), pk(R), pk(S)]$	**Server** $S[sk(S), pk(S), pk(R)]$	**Smart Contract** $C[pk(R), pk(S)]$
$pid, data \leftarrow$ read from product $c_1 \leftarrow random()$ $m_1 \leftarrow \{R, c_1\}pk(S)$ $\xrightarrow{m_1}$		
	$R, c_1 \leftarrow \{m_1\}sk(S)$ $c_2 \leftarrow random()$ $m_2 \leftarrow \{S, C, c_1, c_2\}pk(R)$ $\xleftarrow{m_2}$	

$S, C, c_1', c_2 \leftarrow \{m_2\}sk(R)$
if $c_1 \neq c_1'$ **then**
$abort$("S Authentication Failed")
end if
$m_{sig} \leftarrow \{C, c_2, pid, data\}sk(R)$
$m_3 \leftarrow \{m_{sig}\}pk(S)$
$\xrightarrow{m_3}$		

	$m_{sig} \leftarrow \{m_3\}sk(S)$
	$C, c2', pid, data \leftarrow \{m_{sig}\}pk(R)$
	if $c_2 \neq c_2'$ **then**
	$abort$("R Authentication Failed")
	end if
	$c_3 \leftarrow random()$
	$m_4 \leftarrow \{C, c_3, m_{sig}\}sk(S)$
	$\xrightarrow{m_4}$	

		$C, c_3, m_{sig} \leftarrow \{m_4\}pk(S)$
		$C, c_2, pid, data \leftarrow \{m_{sig}\}pk(R)$
		if c_2 **or** c_3 have been used before **then**
		$abort$("Replayed message")
		end if

```
59 }
```

Listing 1: ForTrac Solidity Code

5 PROOF OF FORWARD TRACEABILITY

We show that ForTrac achieves forward traceability by creating a one-on-one mapping between the smart contract and our definition given in Section 4. The valid predicate under consideration is our path and time-based valid predicate given in Definition 3.3. Our definition contains two parts: states and a transition relation. To show that ForTrac is indeed a forward traceability solution, we will show that it possesses both parts.

To show that ForTrac implements forward traceability, we first need to show that it introduces the concepts of Events. In ForTrac, an event is represented as an Event struct (line 7). Each function call to the 'update' function triggers the creation of a new event. It uses `msg.sender`, the timestamp by using `block.timestamp`, and the data provided as a function parameter as the values for this new event. This Event struct is used as a building block for traces.

A state is defined as a sequence of events for a given product. In ForTrac, a state is represented in the 'trace' property of the Product struct (line 15). The mapping id2product (line 20) maps a product identifier to a Product struct. So, each product has a sequence of events associated with it. During the execution of 'createProduct' (line 22) the trace property is implicitly set to []. So, each manufactured product starts with an empty trace. It shows that ForTrac implements the states as defined in Definition 3.2.

ForTrac updates a trace by first tentatively adding a new event consisting of `msg.sender`, `data`, and `block.timestamp` (line 55). It then calls the 'verify' function on the last event (line 56). Given that i is the index of the new event, the verify function checks two things. Firstly, does the `msg.sender` of event i match the expected actor in the path at index i (line 39)? Secondly, is the timestamp of event i minus the timestamp of event $i-1$ within the time

interval specified in the interval array (line 17) at index i (line 41)? If both conditions hold, the event remains added. In case one of the conditions does not hold, the verify function will raise an error to revert the update function, resulting in the last event being removed. An event is added when it meets the same two conditions as given in the forward traceability Definition 3.3, indicating that they implement the same function.

By mapping the components of ForTrac onto the forward traceability definitions, we have provided good arguments that ForTrac achieves forward traceability.

6 SECURITY ANALYSIS

Our security analysis investigates three attacks: data tampering attacks, replay attacks, and impersonation attacks. Resistance against these attacks indicates data that our system achieves data integrity and authenticity. We show resistance against these attacks using a formal verification tool called Scyther and give some informal arguments.

6.1 Threat Model

We assume a Dolev-Yao adversary [7] who has complete control over the network. We further assume that the attacker can compromise at most one in three participants, consistent with the conventional setting [6]. We also assume perfect cryptography, indicating that an encrypted message can only be decrypted with the corresponding key. Lastly, we assume that a reader correctly identifies a product.

6.2 Scyther Results

We prove the security properties of our communication protocol by using Scyther. Scyther is an automated security protocol verification tool that runs under the Dolev-Yao intruder model with the assumption of perfect cryptography. Scyther trims the state space by specifying a maximum number of runs [6]. Empirical studies have shown that five runs are usually sufficient, so we stick to this parameter. Scyther can verify whether a communication protocol satisfies specific claims [7]. Using a sequence of *Running* and *Commit* claims, Scyther verifies if two parties agree on a piece of data. For instance, if we need to prove that two parties (A and B) share data (d), then B must have made a claim $Running(A, d)$ before A makes a claim $Commit(B, d)$ [20]. The $Weakagree$ claim specifies that agent A knows that when it starts the protocol with B, B at some point will execute the protocol with A [5]. So, it provides authentication between A and B. The *Alive* claim is a weaker form of authentication: if A initiates the protocol with B, B at some point executes the protocol with some agent. The $Nisynch$ claim specifies that all parties agree that they were communicating and that the messages are sent and received in order [5]. The $Niagree$ claim is a weaker form of $Nisynch$ in that it does not require the correct message order.

The communication protocol between the reader, server, and the smart contract should provide the following properties.

- **Mutual authentication:** The server and the reader know that they are communicating with each other.
- **Reader to smart contract authentication:** The reader should be authenticated to the smart contract.
- **Server to smart contract authentication:** The server should be authenticated to the smart contract.
- **Data agreement:** Reader, server, and the smart contract should agree on the data being transmitted. This is modelled with the *Commit* and *Running* claims.

Figure 3 shows the results of the Scyther tool. The SPDL script to get these results is given in appendix A. The 'commit' claims show that the smart contract agrees with the reader and server on the data. It means that the data transmitted by R will be accurately stored in the smart contract. The $Nisynch$ claims show that every party agrees on the messages being sent. The weakagree and alive properties of the smart contract show that both the reader and server are authenticated to the smart contract.

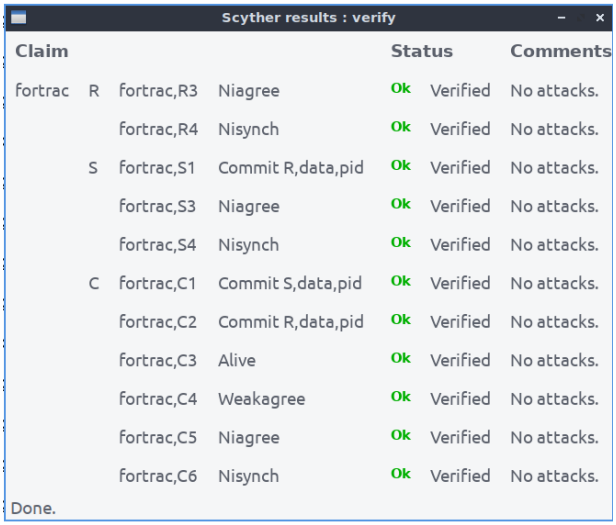


Figure 3: Scyther Results

6.3 Data Tampering Attack

Data on the blockchain is immutable under the assumption that at least 51 per cent of the nodes are honest. Blockchain protects data from being tampered with. In this paper, we store our data in a smart contract. Data in a smart contract can be changed, but it will leave an audit trail. Since our smart contract does not have any functionality for removing or changing data, it remains immutable.

Assuming that the trace for a product p can only be updated by a participant x, an adversary can try to influence the data being sent to the smart contract. In our communication protocol, there are two points where an attacker can try to tamper with the data field: between reader and server, or between server and smart contract. We assume that the reader has identified the product correctly and contains the correct data. The message between the reader and server containing the product information is encrypted using the public key of the server. The message is also signed with the private key of the reader. Furthermore, the data is only sent once the reader and server are mutually authenticated. Therefore, an adversary would need the secret key $sk(R)$ of the reader to sign a message. The message between the server and the smart contract is signed

by the server and contains data signed by the reader. The adversary needs the private key of the reader to tamper with the product information.

The data cannot be tampered with because the data is wrapped and nested inside two signatures and because the server and reader are mutually authenticated.

6.4 Replay Attack

Replay attacks happen when an old message is re-transmitted by an adversary. If successful, an attacker can use replay attacks to send old messages masquerading as new. Replay attacks are prevented by making sure messages are fresh, which means that every message is unique.

The communication between the reader and server is mutually authenticated by using the public key Needham-Schroeder-Lowe (NSL) algorithm [19]. Data is only transmitted after authentication. We use two nonces to ensure that all messages between the reader and server are fresh. This is confirmed by the Scyther results with the *Niagree* claims. Therefore, the freshness prevents an adversary from launching a replay attack between the reader and server.

In Ethereum, resistance against replay attacks is implemented by using a nonce. We modelled this in our message sequence diagram given in Figure 3. The message sent to the smart contract contains two nested signatures. Both signatures contain fresh nonces so that the messages cannot be replayed. This setup prevents a replay attack between the server and the smart contract.

The resistance against replay attacks is further confirmed by the fact that the Scyther results show that the *Niagree* and *Nisynch* claims hold.

6.5 Impersonation Attack

An impersonation attack occurs when an adversary tries to masquerade as another entity, by any means necessary. A replay attack is a specific form of an impersonation attack. Between the reader and server, impersonation is not possible because they are mutually authenticated. All interactions with the smart contracts contain signed messages. Hence, unless the adversary knows the private key, it cannot forge a message. Replay attacks are prevented by the unique nonce contained in each Ethereum transaction.

7 PERFORMANCE AND COST ANALYSIS

To assess our solution's feasibility, we conduct an analysis of performance and cost. We executed our experiments on an Ubuntu 22.04 VM with two virtual CPUs and 8GB of memory. Furthermore, we used Ganache 2.5.4 as a local Ethereum test net with 25 accounts each containing 1000 ether.

We set up our experiments using an external truffle script. The script creates 64 products with path lengths ranging from 1 to 64. For each product, we follow the same structure: create a product and update it until the last step. For each create and update, the time and costs are measured. Time is measured by wrapping each create or update between two '`time.Now`' calls. The difference indicates the execution time in milliseconds. The costs are measured by using the '`getBalance`' function in the web3 library.

Figure 4 shows the path length plotted against time in milliseconds. The execution time scales linearly with the path length. A path of length 64 takes about 32 seconds to process. This shows that our solution is scalable in terms of time.

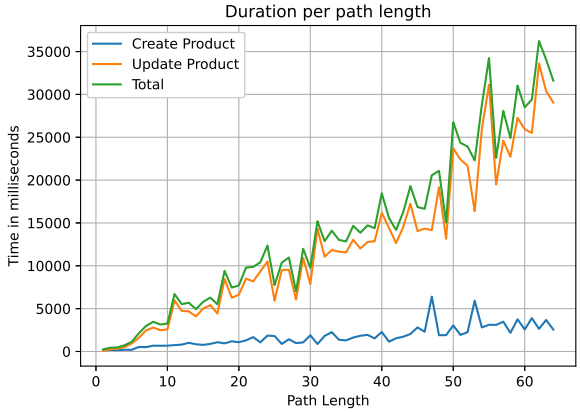


Figure 4: Time per Path Length

Figure 5 shows the cost of our solution in gwei with the assumption that a gas costs 20gwei. The assumption is consistent with existing research [26, 33]. Similar to the execution time, the cost scales linearly. Since the number of updates reflects the number of participants, the cost per participant remains constant. This result shows that our solution is scalable in terms of cost as well.

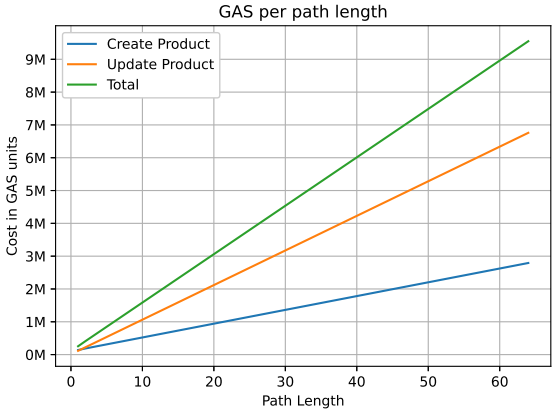


Figure 5: Cost in gwei per Path Length

The detailed results are listed in Table 4.

Time and costs are affected by many factors, like the amount of data being sent to the smart contract and the current capacity. The most important thing is that cost and time should not increase faster than linear. We argue that performance is a more important factor than cost because there are cost mitigating solutions available. Firstly, products can be batched together. So, instead of tracing a single product, we can trace a batch with n products This effectively means the costs can be divided by n. Secondly, there are smart contract optimizations. For example, we can consider storing data

Table 4: Results for path length 1, 16, and 64

	Length	Cost (gwei)	Time (ms)
Create	1	139272	143
	16	774113	885
	64	2791922	2554
Update	1	114787	100
	16	1696777	5403
	64	6759145	29049
Total	1	254059	243
	16	2470890	6288
	64	9551067	31603

off-chain and provide a pointer to this data on-chain. Thirdly, many consortium or private blockchain solutions do not require gas. So, it can be considered to move to a private blockchain solution, for example Hyperledger.

8 CONCLUSION

In this article, we have presented a forward traceability solution using ERC-721 tokens, commonly known as NFTs. The NFT tokens are used to store the trace of the physical product. We extend an earlier defined valid predicate based on paths considering time intervals and show that ForTrac achieves forward traceability. We have shown evidence that our solution is secure by doing an informal security analysis and using a formal verification tool called Scyther. Most significantly, we provide evidence that our solution is secure against replay, impersonation, and data tampering attacks. By providing experiments on performance and costs we have shown that our solution is feasible as well.

ForTrac achieves data completeness by using a path and time-based valid predicate. The trace is complete when there are no more steps to take. ForTrac contributes to data accuracy by ensuring that only one actor at a time can update the trace of a product. Furthermore, ForTrac provides trust that the data is correct by showing that reader and smart contract agree on the transmitted data, limiting the possibility of tampering. We have shown evidence that our solution is secure by proving an informal security analysis and using Scyther.

Our approach has a few limitations. Firstly, our solution assumes that the manufacturer of a product knows the path it is going to take and how long each step takes. There are cases where this assumption is not realistic. Therefore, we will extend our implementation to include other valid predicates, better facilitating dynamic supply chains. Secondly, our solution will be extended by introducing privacy requirements to mitigate privacy risks. Lastly, in this paper, we assumed the reader could authenticate the product accurately. This is however a challenging problem. In future work, we will explore how we can improve the product to smart contract binding.

ACKNOWLEDGMENTS

The work has been supported by the Cyber Security Research Centre Limited whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme.

REFERENCES

[1] Bander A Alzahrani, Khalid Mahmood, and Saru Kumari. 2020. Lightweight Authentication Protocol for NFC Based Anti-Counterfeiting System in IoT Infrastructure. *IEEE access : practical innovations, open solutions* 8 (2020), 76357–76367.
[2] Denis Avrilionis and Thomas Hardjono. 2022. From Trade-only to Zero-Value NFTs: The Asset Proxy NFT Paradigm in Web3. arXiv:2205.04899 [cs]
[3] Kai Bu and Yingjiu Li. 2018. Every Step You Take, I'll Be Watching You: Practical StepAuth-Entication of RFID Paths. *IEEE Transactions on Information Forensics and Security* 13, 4 (2018), 834–849. https://doi.org/10.1109/TIFS.2017.2768022
[4] Jaewook Byun and Jin Mitsugi. 2022. EPC Tag Data Standard (TDS).
[5] Cas Cremers and Sjouke Mauw. 2012. *Operational Semantics and Verification of Security Protocols.* Springer, Heidelberg [u.a.].
[6] Cas J. F. Cremers, Pascal Lafourcade, and Philippe Nadeau. 2009. Comparing State Spaces in Automatic Security Protocol Analysis. In *Formal to Practical Security*, Véronique Cortier, Claude Kirchner, Mitsuhiro Okada, and Hideki Sakurada (Eds.). Vol. 5458. Springer Berlin Heidelberg, Berlin, Heidelberg, 70–94. https://doi.org/10.1007/978-3-642-02002-5_5
[7] Danny Dolev and Andrew Yao. 1983. On the Security of Public Key Protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.
[8] Sanjeev Kumar Dwivedi, Ruhul Amin, and Satyanarayana Vollala. 2020. Blockchain Based Secured Information Sharing Protocol in Supply Chain Management System with Key Distribution Mechanism. *Journal of Information Security and Applications* 54 (Oct. 2020), 102554. https://doi.org/10.1016/j.jisa.2020.102554
[9] Everledger. 2023. Main Home - Everledger. https://everledger.io/.
[10] Patrizia Gartner, Martin Benfer, Andreas Kuhnle, and Gisela Lanza. 2021. Potentials of Traceability Systems - a Cross-Industry Perspective. *Procedia CIRP* 104 (2021), 987–992. https://doi.org/10.1016/j.procir.2021.11.166
[11] GS1 AISBL. 2017. GS1 Global Traceability Standard.
[12] Haya Hasan, Esra AlHadhrami, Alia AlDhaheri, Khaled Salah, and Raja Jayaraman. 2019. Smart Contract-Based Approach for Efficient Shipment Management. *Computers & Industrial Engineering* 136 (Oct. 2019), 149–159. https://doi.org/10.1016/j.cie.2019.07.022
[13] Vikas Hassija, Vinay Chamola, Vatsal Gupta, Sarthak Jain, and Nadra Guizani. 2021. A Survey on Supply Chain Security: Application Areas, Security Threats, and Solution Architectures. *IEEE Internet of Things Journal* 8, 8 (2021), 6222–6246. https://doi.org/10.1109/JIOT.2020.3025775
[14] Fokke Heikamp, Lei Pan, Rolando Trujillo-Rasua, Sushmita Ruj, and Robin Doss. 2022. Forward Traceability for Product Authenticity Using Ethereum Smart Contracts. In *Proceedings of the 2022 International Conference on Network and System Security*, Xingliang Yuan, Guangdong Bai, Cristina Alcaraz, and Suryadipta Majumdar (Eds.). Springer Nature Switzerland, Cham, 514–523.
[15] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. 2017. A Survey on Provenance: What for? What Form? What From? *The VLDB Journal — The International Journal on Very Large Data Bases* 26, 6 (Dec. 2017), 881–906. https://doi.org/10.1007/s00778-017-0486-1
[16] Md Didarul Islam, Haoting Shen, and Shahriar Badsha. 2022. Integrating Blockchain into Supply Chain Safeguarded by PUF-enabled RFID. *Internet of Things* 18 (May 2022), 100505. https://doi.org/10.1016/j.iot.2022.100505
[17] Marlene Kuhn, Felix Funk, Guanlai Zhang, and Jörg Franke. 2021. Blockchain-Based Application for the Traceability of Complex Assembly Structures. *Journal of Manufacturing Systems* 59 (April 2021), 617–630. https://doi.org/10.1016/j.jmsy.2021.04.013
[18] Menghui Lou, Xiaolei Dong, Zhenfu Cao, and Jiachen Shen. 2021. SESCF: A Secure and Efficient Supply Chain Framework via Blockchain-Based Smart Contracts. *Security and Communication Networks* 2021 (2021), 1–18.
[19] Gavin Lowe. 1995. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Inform. Process. Lett.* 56, 3 (Nov. 1995), 131–133. https://doi.org/10.1016/0020-0190(95)00144-2
[20] G. Lowe. 1997. A Hierarchy of Authentication Specifications. In *Proceedings of the 10th Computer Security Foundations Workshop*. IEEE Comput. Soc. Press, Rockport, MA, USA, 31–43. https://doi.org/10.1109/CSFW.1997.596782
[21] Tom De Nies. 2013. Constraints of the PROV Data Model. https://www.w3.org/TR/2013/REC-prov-constraints-20130430/.
[22] Petter Olsen and Melania Borit. 2013. How to Define Traceability. *Trends in Food Science & Technology* 29, 2 (2013), 142–150. https://doi.org/10.1016/j.tifs.2012.10.003
[23] Andrew Park, Jan Kietzmann, Leyland Pitt, and Amir Dabirian. 2022. The Evolution of Nonfungible Tokens: Complexity and Novelty of NFT Use-Cases. *IT Professional* 24, 1 (Jan. 2022), 9–14. https://doi.org/10.1109/MITP.2021.3136055
[24] Hiren Patel and Bela Shrimali. 2023. AgriOnBlock: Secured Data Harvesting for Agriculture Sector Using Blockchain Technology. *ICT Express* (July 2023), S2405959521000862. https://doi.org/10.1016/j.icte.2021.07.003
[25] Simon Pearson, David May, Georgios Leontidis, Mark Swainson, Steve Brewer, Luc Bidaut, Jeremy G. Frey, Gerard Parr, Roger Maull, and Andrea Zisman. 2019. Are Distributed Ledger Technologies the Panacea for Food Traceability? *Global Food Security* 20 (March 2019), 145–149. https://doi.org/10.1016/j.gfs.2019.02.002

[26] Bora Buğra Sezer, Selçuk Topal, and Urfat Nuriyev. 2022. TPPSUPPLY : A Traceable and Privacy-Preserving Blockchain System Architecture for the Supply Chain. *Journal of Information Security and Applications* 66 (May 2022), 103116. https://doi.org/10.1016/j.jisa.2022.103116

[27] Syed W Shah, Naeem F Syed, Arash Shaghaghi, Adnan Anwar, Zubair Baig, and Robin Doss. 2021. LCDA: Lightweight Continuous Device-to-Device Authentication for a Zero Trust Architecture (ZTA). *Computers & Security* (2021), 102351.

[28] Justin Sunny, Naveen Undralla, and V. Madhusudanan Pillai. 2020. Supply Chain Transparency through Blockchain-Based Traceability: An Overview with Demonstration. *Computers & Industrial Engineering* 150 (2020), 106895. https://doi.org/10.1016/j.cie.2020.106895

[29] Naeem Firdous Syed, Syed W. Shah, Rolando Trujillo-Rasua, and Robin Doss. 2022. Traceability in Supply Chains: A Cyber Security Analysis. *Computers & Security* 112 (2022), 102536. https://doi.org/10.1016/j.cose.2021.102536

[30] Kentaroh Toyoda, P. Takis Mathiopoulos, Iwao Sasase, and Tomoaki Ohtsuki. 2017. A Novel Blockchain-Based Product Ownership Management System (POMS) for Anti-Counterfeits in the Post Supply Chain. *IEEE Access* 5 (2017), 17465–17477. https://doi.org/10.1109/ACCESS.2017.2720760

[31] TradeLens. 2023. TradeLens | Supply Chain Data and Docs. https://www.tradelens.com/.

[32] Mueen Uddin. 2021. Blockchain Medledger: Hyperledger Fabric Enabled Drug Traceability System for Counterfeit Drugs in Pharmaceutical Industry. *International Journal of Pharmaceutics* 597 (2021), 120235. https://doi.org/10.1016/j.ijpharm.2021.120235

[33] Martin Westerkamp, Friedhelm Victor, and Axel Küpper. 2020. Tracing Manufacturing Processes Using Blockchain-Based Token Compositions. *Digital Communications and Networks* 6, 2 (May 2020), 167–176. https://doi.org/10.1016/j.dcan.2019.01.007

A SCYTHER SPDL FILE

```
usertype Product;
usertype EventData;

protocol fortrac(R, S, C)
{
  const pid: Product;
  const data: EventData;

  role R
  {
    fresh c1: Nonce;
    var c2: Nonce;

    send_RS1(R, S, {R, c1}pk(S));
    recv_SR(S, R, {S, C, c1, c2}pk(R));
    claim(R, Running, S, data, pid);
    claim(R, Running, C, data, pid);
    send_RS2(R, S, {{C, c2, pid, data}sk(R)}pk(S));

    claim(R, Niagree);
    claim(R, Nisynch);
  }
  role S
  {
    var data: EventData;
    var c1: Nonce;
    fresh c2, c3: Nonce;

    recv_RS1(R, S, {R, c1}pk(S));
    send_SR(S, R, {S, C, c1, c2}pk(R));
    recv_RS2(R, S, {{C, c2, pid, data}sk(R)}pk(S));

    claim(S, Commit, R, data, pid);
    claim(S, Running, C, data, pid);

    send_SC(S, C, {C,  c3, {C, c2, pid, data}sk(R)}sk(S))
        ;

    claim(S, Niagree);
    claim(S, Nisynch);
  }
  role C
  {
    var data: EventData;
    var c2, c3: Nonce;

    recv_SC(S, C, {C, c3, {C, c2, pid, data}sk(R)}sk(S));

    claim(C, Commit, S, data, pid);
    claim(C, Commit, R, data, pid);
    claim(C, Alive);
    claim(C, Weakagree);
    claim(C, Niagree);
    claim(C, Nisynch);
  }
}
```

Listing 2: Scyther SPDL File