Shaping Blockchain Technology for Securing Supply Chains

Yong Zhi Lim[1,2](✉), Jianying Zhou[1], and Martin Saerbeck[2]

[1] Singapore University of Technology and Design, Singapore, Singapore
yongzhi_lim@mymail.sutd.edu.sg, jianying_zhou@sutd.edu.sg
[2] Digital Service Centre of Excellence, TÜV SÜV Asia Pacific, Singapore, Singapore
{yong-zhi.lim,martin.saerbeck}@tuvsud.com

Abstract. Purchases in supply chains involve a network of suppliers, manufacturers, logistics or even customers needed for the procurement of goods or services. These are needed to operate a supply chain efficiently and allow timely deliverables to consumers. In our work, we identify and map a typical business process to demonstrate how we can securely allow participants to interact with smart contracts and discover potential use cases for supply chains.

Keywords: Blockchain · Smart contracts · Supply chains

1 Introduction

The rise of Blockchain is arguably attributed to the use of *Bitcoin* for financial transactions. It currently has the world's highest market cap and is the costliest cryptocurrency to date [1]. Its hype has evolved over the past decade and seen the rise of different consensus algorithms, with claims of providing higher hash rates and transactions per second.

As businesses continue to embrace and migrate towards digitization of services, P2P DLT (peer-to-peer distributed ledger technology) or blockchain plays a crucial role and has seen growing interest in adapting it with discovery and deployment of potential use cases in the supply chain sector.

Supply chains, used interchangeably with Supply Chain Management (SCM), is a network of carriers and sellers to allow procurement of goods or services to buyers. This process is constantly optimized over time to save costs and allows for a quicker production cycle.

Cryptocurrencies are not the only reason for the adoption of blockchain technology. A blockchain-enabled supply chain will provide security, transparency, authenticity and trustworthiness [19]. However, the technology is not entirely foolproof, being susceptible to various attacks. This creates a barrier for any supply chain wishing to adapt blockchains. We study existing industry standards to identify and adopt best practices to protect the blockchain.

J. Zhou et al. (Eds.): ACNS 2021 Workshops, LNCS 12809, pp. 3–18, 2021.
https://doi.org/10.1007/978-3-030-81645-2_1

Previous studies which have claimed to successfully deploy a blockchain in supply chains are private in nature, due to the usage of a permissioned blockchain [6]. On the contrary, this defeats the purpose of transparency despite transactions being traceable and with little to none industry-specific knowledge for secure implementation by other parties. To date, most literature describe the benefits of deploying a blockchain but with a lack of practical implementations.

In this paper, we include the identification and mapping of a typical business process to demonstrate how an electronic bill of lading (eBL), which bridges several standards, coded in *Solidity* that allows participants in *Ethereum* to interact with smart contracts and discover potential use cases for supply chains. We also identify current attacks on smart contracts and challenges ahead.

2 Background

2.1 Current State of Purchases in Supply Chains

Purchases in supply chains involve a network of suppliers, manufacturers, logistics or even customers needed for the procurement of goods or services. From the procurement of raw materials, these are needed to operate a supply chain efficiently and allow timely deliverables to consumers. Figure 1 briefly shows how a supply chain perform procurement of raw materials that is supplied by its vendors, going through several processes to manufacture and package the goods, before transportation and reaching out to its consumers.

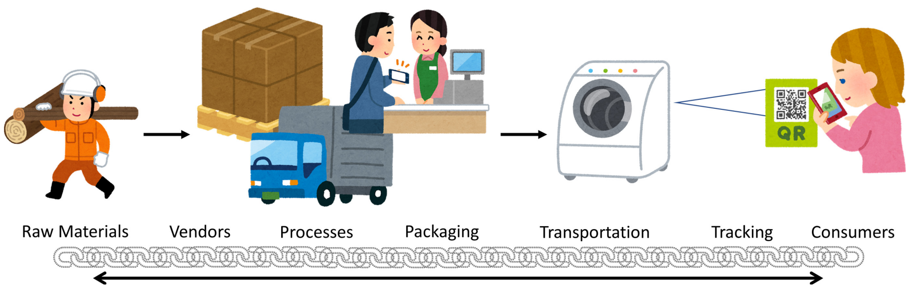

Fig. 1. A Typical Supply Chain Process. Images under free-use from https://www.irasutoya.com by Takashi Mifune

2.2 Procure-to-Pay Process

A typical procure-to-pay process in supply chains (reflected in Fig. 2) generally consist of the following procedures involving 3 parties, the buyer $\mathbb{B}$, seller $\mathbb{S}$ and carrier $\mathbb{C}$:

1. 𝔹's purchasing department creates a Purchase Order (PO) in its ERP (Enterprise Resource Planning) system and sends it to 𝕊. The PO contains important information on:
   (a) Items for purchase (item description, item part number, order quantity, unit price, currency, total value, discounts, etc.)
   (b) Delivery instructions (delivery address, delivery date, incoterms, etc.)
   (c) Procurement references (purchase requisition number, quotation number, etc.)
   (d) Other information (buyer and seller information, payment terms, etc.)
2. 𝕊 acknowledges receipt and acceptance of the PO by returning a signed copy to 𝔹.
3. 𝕊 prepares the item together with a copy of the Delivery Order (DO) and Packing List (PL). Upon notice from 𝕊, ℂ arranges for shipment. ℂ also shares a copy of the Air Waybill (AWB) or the Bill of Lading (BL) with 𝕊.
4. Once the item is delivered to the designated address in 𝔹's warehouse, the warehouse personnel then inspects item and tallies it with the DO. The DO is signed physically to acknowledge receipt of the item that it is in good order and condition. The Goods Receipt (GR) is also done in 𝔹's ERP system.
5. 𝕊 sends a copy of the invoice to 𝔹's purchasing department. Some business practices may require additional approval on the invoice depending on the 𝔹's internal processes.
6. The invoice is then submitted to 𝔹's accounting department for processing. The accounting personnel checks and verifies invoice against the GR and PO information, which is part of the *three-way matching process*. The invoice is recorded in 𝔹's ERP system and contains information such as billing name and address, delivery address, invoice number and date, PO number, payment terms, item description and part number, quantity, unit price, currency, item amount, tax amount, 𝕊's bank information, etc.
7. Lastly, the invoice is scheduled and due for payment according to agreed payment terms. 𝔹's accounting personnel prepares and processes payment by cash, checks or bank transfers after approved internally. A copy of the payment detail/advice is sent to 𝕊 to match receipts.

Standards. According to the United States Code of Federal Regulations (CFR) Title 49, the bill of lading (BL) is a critical document that legally binds the buyer and seller with all relevant shipment information (e.g. addresses, reference numbers, shipping mark, etc.) [3,28]. As we push towards standardizations, alignment with the UNCITRAL (United Nations Commission on International Trade Law) Model Law on Electronic Transferable Records (MLETR) is crucial to ensure a common acceptance and quicker adoption by all [14].

Trade Terms. Better known as International Commercial Terms (Incoterms), trade terms are globally recognized terms by the International Chamber of Commerce (ICC) for international trade [15]. It provides rules for trading and the sale of goods. In its most current iteration, ICC has defined 11 terms: *Ex-Works*

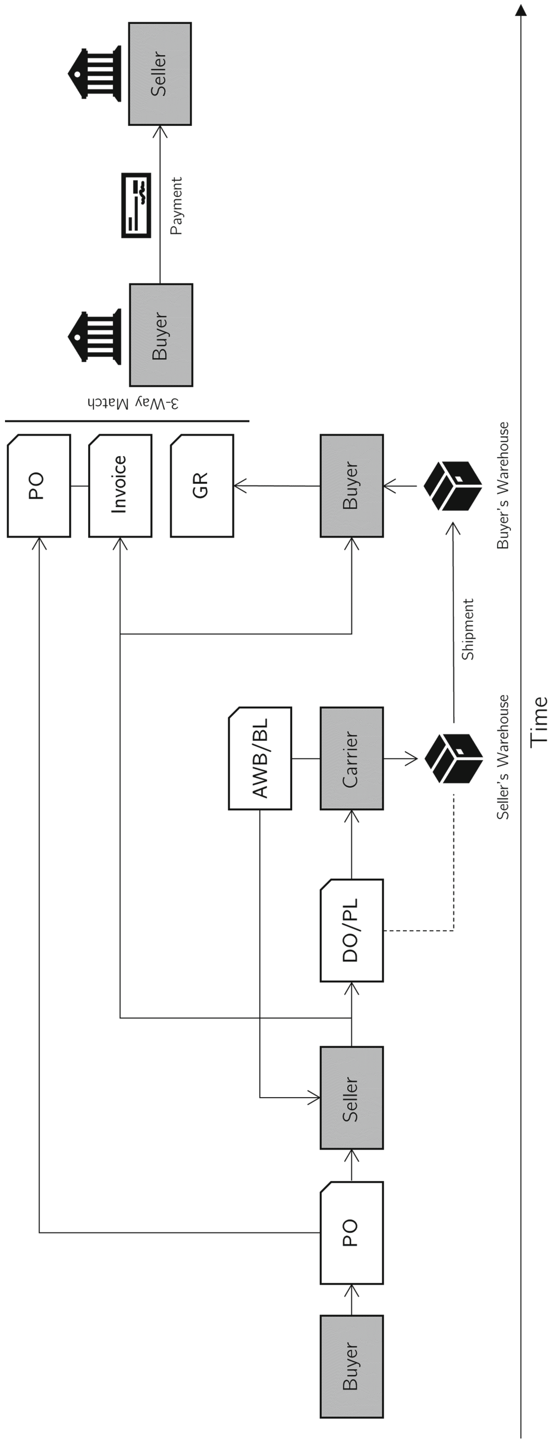

Fig. 2. A Typical Procure-to-Pay Process in the Supply Chain

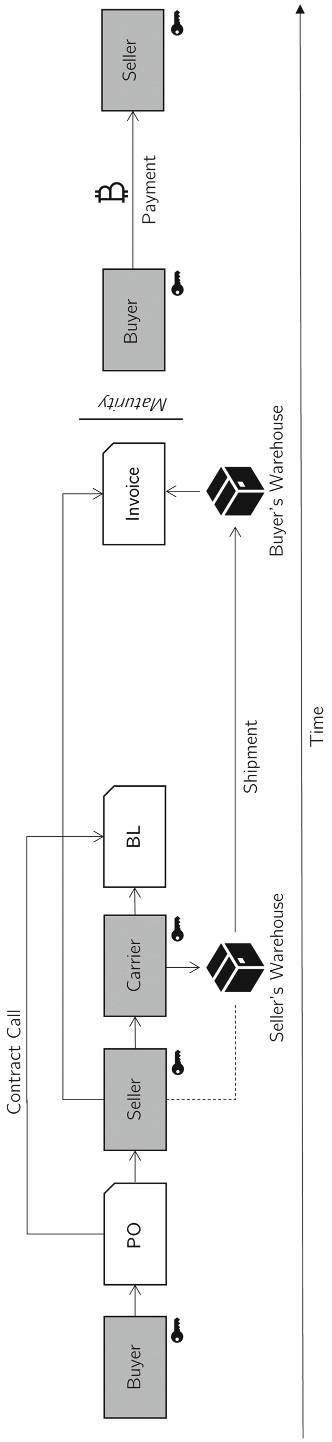

Fig. 3. A Blockchain-Enabled Process in the Supply Chain

(EXW), *Free Carrier* (FCA), *Carriage Paid To* (CPT), *Carriage And Insurance Paid To* (CIP), *Delivered At Place* (DAP), *Delivered At Place Unloaded* (DPU), *Delivered Duty Paid* (DDP) for any mode of transport and *Free Alongside Ship* (FAS), *Free On Board* (FOB), *Cost and Freight* (CFR) and *Cost, Insurance and Freight* (CIF) for sea & inland transportation. Figure 4 clearly indicates the rules which define the liabilities and transfer of risk that fall between the buyer and seller should an issue with shipping arise.

2.3 Blockchain and Smart Contracts

Although not made explicit in *Bitcoin*'s original work, a blockchain claims to facilitate a secure payment gateway with the use of digital signatures between parties, without the need of an intermediary. These transactions are then timestamped and hashed to create an on-going chain of blocks, hoping to outpace attackers [35].

The differentiating factor amongst different blockchains is perhaps their choice of smart contract language. Highly influenced by *Javascript*, it aims to have high readability and could be either be Turing (*Solidity* in *Ethereum*) or non-Turing (*Bitcoin* Scripts) complete. As such, *Bitcoin* does not allow loops, recursion or termination by its own.

Smart contracts are electronic forms of legal agreements which can automate decisions made between different parties based on a set of promises, including protocols within which the parties perform on these promises [42]. *Ethereum* deploys the *Ethereum Virtual Machine* (EVM) to execute these scripts. Once its source code is compiled and deployed, it becomes bytecode and is stored on the blockchan for retrieval.

2.4 Non-Fungible Tokens

Widely known as *ERC-721*, NFTs can represent ownership over digital (e.g. virtual collectables), physical assets (e.g. houses, unique artwork) or even negative value assets (e.g. loans, burdens and other responsibilities) [45]. An example first implemented by *CryptoKitties*, they are cryptocollectibles which represent a real-world analogy to assets like baseball cards or fine art [22]. In our use case, documents involved in the supply chain process, such as the bill of lading (BL) can be represented as a NFT. It allows the use of `safeTransfer`, `approve` functions and tracking of distinguishable documents [45].

2.5 Blockchain for Supply Chains

Can the use of smart contracts in a supply chain be trusted by its buyers, carriers and sellers? In today's digital age, we still lack information sharing between organizations due to centralized databases and manual exchanges of electronic documents. Supply chains can leverage on the benefits of a blockchain to enable greater speed and transparency between stakeholders. We introduce the use of smart contracts to disrupt the procure-to-pay process in supply chains. However, current threats on smart contracts exists and we must address them.

Incoterm		Transfer of Risk	Liabilities											
			Export Packaging	Loading Charges	Delivery to Port/Place	Export Duty, Taxes & Custom Clearance	Origin Terminal Charges	Loading on Carriage	Carriage Charges	Insurance	Destination Terminal Charges	Delivery to Destination	Unloading at Destination	Import Duty, Taxes & Custom Clearance
Any Mode of Transport														
Ex-Works	EXW	At Buyer's Disposal	S	B	B	B	B	B	B	*	B	B	B	B
Free Carrier	FCA	On Buyer's Transport	S	S	S	S	B	B	B	*	B	B	B	B
Carriage Paid To	CPT	At Carrier	S	S	S	S	S	S	S	*	S	B	B	B
Carriage And Insurance Paid To	CIP	At Carrier	S	S	S	S	S	S	S	S	S	B	B	B
Delivered At Place	DAP	At Named Place	S	S	S	S	S	S	S	*	S	S	B	B
Delivered At Place Unloaded	DPU	At Named Placed Unloaded	S	S	S	S	S	S	S	*	S	S	S	B
Delivered Duty Paid	DDP	At Named Place	S	S	S	S	S	S	S	*	S	S	B	S
Sea & Inland Transportation														
Free Alongside Ship	FAS	Alongside Ship	S	S	S	S	S	B	B	*	B	B	B	B
Free On Board	FOB	Onboard Vessel	S	S	S	S	S	S	B	*	B	B	B	B
Cost and Freight	CFR	Onboard Vessel	S	S	S	S	S	S	S	*	B	B	B	B
Cost, Insurance and Freight	CIF	Onboard Vessel	S	S	S	S	S	S	S	S	S	B	B	B

B: Buyer
S: Seller
Asterisk: Negotiable

Fig. 4. Incoterms 2020

3 Design and Implementation

To fully automate the procure-to-pay process in supply chains on the blockchain, we introduce `supplyInvoice`, a smart contract. We show the proposed simplified process in Fig. 3 should a blockchain be deployed for the supply chain. As such, all required information should be obtained from the bill of lading (BL). The BL and invoice are forms of NFTs, which tags itself as a digitized legal document on the blockchain.

Current known open implementations with eBL or invoices exists in [10] and [4] using *Solidity* but not with any standardization, tokenization or use with trade terms.

With specified trade terms in the BL, `supplyInvoice` is able to automatically execute the liabilities should a problem in the shipping process occur. This punishes parties once the liability falls under them and the goods have been transferred (e.g. goods have left seller, goods out for delivery). Several assumptions are made to complete this process: 1) parties perform immediate monetary transfer (no delayed payment terms), 2) no involvement of escrows (third-party) and 3) only honest parties are involved.

3.1 Data Structures

Order. contains necessary information to facilitate communication, payment and successful shipment of goods between the buyer $\mathbb{B}$, seller $\mathbb{S}$ and carrier $\mathbb{C}$. The following structure contains:

- `buyer` address to digital wallet to facilitate payment
- `seller` address to digital wallet to facilitate payment
- `referenceNumber` an unique identifier to allow tracing and easy reference
- `tradeTerms` stipulated in Sect. 2.2 to clearly define liabilities
- `shippingMark` an identifier labelled on the shipped product

3.2 Business Logic

Mapping `supplyInvoice` described in Sect. 2.2, the following function prototypes provide a simplified process to complete a typical procure-to-pay process for interaction in the blockchain.

- `createOrder` is created by $\mathbb{B}$ for the initial order and prompts to populate all necessary fields in the `order` struct to facilitate procurement of goods or services.
- `createInvoice` acknowledges the newly created order and prompts $\mathbb{S}$ to prepare goods for shipment. Prepare payment for $\mathbb{C}$.
- `createLading` is created by $\mathbb{C}$ the BL for the shipping process.
- `assignTradeTerms` assigns the rules and define liabilities between $\mathbb{B}$ and $\mathbb{S}$.
- `negotiateTradeTerms` negotiates specific liabilities between $\mathbb{B}$ and $\mathbb{S}$ for negotiable incoterms stipulated in Fig. 4.

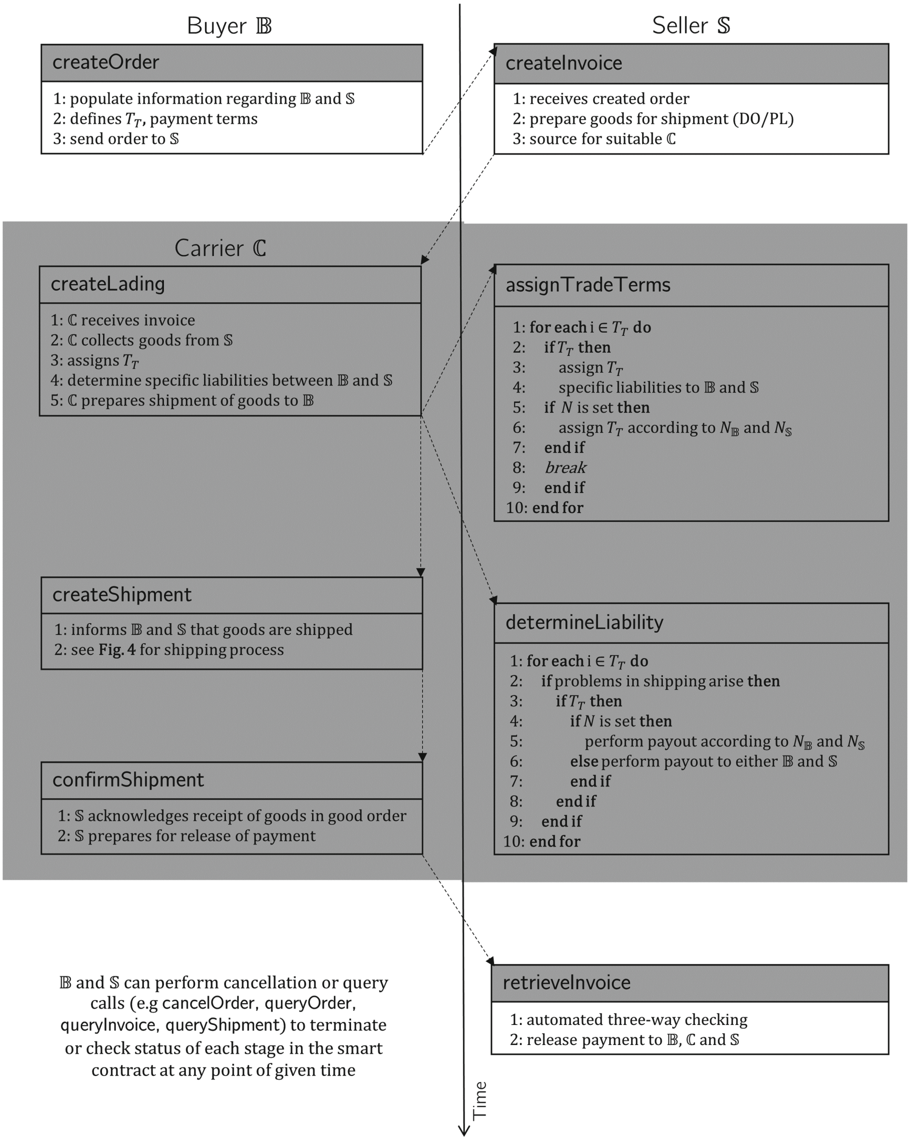

Fig. 5. Automated process in `supplyInvoice` involving 3 different parties, buyer 𝔹, seller 𝕊 and carrier ℂ. A solid line is represented with time, whereas dotted lines represent a typical flow occurring in the smart contract.

- `determineLiability` determines the final liability, should an issue with the shipping occur.
- `confirmShipment` confirms the completion of the shipping process, provided 𝕊 acknowledges receipt of the goods in good order.

– `retrieveInvoice` finally retrieves the invoice and releases payment for $\mathbb{B}$, $\mathbb{C}$ and $\mathbb{S}$, upon maturity.

Additionally, parties can perform the following queries to `supplyInvoice`:

– `queryOrder` queries the specified order for information. Performed only by $\mathbb{B}$ or $\mathbb{S}$.
– `queryInvoice` queries the specified invoice for payment information. Performed by $\mathbb{B}$, $\mathbb{C}$ or $\mathbb{S}$. Access is mutually exclusive between these parties.
– `queryShipment` obtains information regarding the movement of the goods and at which stage is it currently at (e.g. Preparation for export, loading, delivery to port/place, etc.). Performed only by $\mathbb{B}$ or $\mathbb{S}$.

Should the order fail to materialize due to non-agreement of terms, the $\mathbb{B}$ or $\mathbb{S}$ can issue a cancellation via `cancelOrder` or `cancelInvoice` respectively. Figure 5 shows how this business logic is carried out between $\mathbb{B}$, $\mathbb{C}$ and $\mathbb{S}$.

3.3 Implementation

To test this implementation, we wrote `supplyInvoice` in *Solidity*.[1] Using *Ganache* and *Truffle*, we were able to test its functionalities in a private *Ethereum* network. To further solidify security, we utilize the `SafeMath` and the `ERC721` libraries in *OpenZeppelin* to ensure best practices [7].

4 Threats

Although such an implementation may seem robust, a comprehensive study of current threats is needed to further understand how we can secure the use of smart contracts within supply chains.

4.1 Analysis on Smart Contract Attacks

Smart contracts are no different from a usual computer application. They can be affected by bugs or poorly implemented code which allow attackers to exploit or bypass rules. Despite being exposed to cyberattacks in the past, the *Ethereum* blockchain sets an example of a robust and secure network. It is covered in many peer-reviewed research with its security vulnerabilities documented in great detail. Past attacks include The Decentralized Autonomous Organization (*TheDAO*) attack [44] and the Parity Wallet hack [38] were key examples of reentrancy and access control issues that costed *11.5M ETH* (50M USD in 2016) and *150k ETH* (30M USD in 2017) respectively.

Additional vulnerabilities include arithmetic issues (integer underflow or overflow), unchecked return calls, denial of service (DoS), pseudo-randomness, front-running, timestamp dependence and off-chain issues [36]. In light of these

[1] https://github.com/limyz/supplyInvoice.

issues, researchers are tackling these vulnerabilities by analysing the bytecode statically or dynamically or through the study of current transactions performed in *Ethereum*. Many overlapping vulnerabilities which these tools can solve proves the difficulty working between *Solidity* and EVM bytecode.

Static Analysis is performed prior to deployment as bytecode to the EVM. *Oyente*, uses symbolic execution to check for transaction-ordering, timestamp dependence, mishandled exceptions and reentrancy [32]. *ZEUS* uses formal verification for analyzing safety properties of smart contracts [30]. *Maian* checks for unrestricted smart contract actions [37]. *Securify* on the other hand, checks security properties of the EVM bytecode of smart contracts [43]. *Vandal* introduces a framework for detecting security vulnerabilities in smart contract bytecode rapidly, outperforming Oyente, EthIR [18], Mythril [34], and Rattle [41]. *Vandal* extracts logic relations from smart contract bytecode for logic-based analysis [20]. *ETHBMC* is able to capture inter-contract relations, cryptographic hash functions, and memcopy-style operations in smart contracts and claims to be faster than Maian and teEther [23].

Dynamic Analysis is performed at runtime after deployment to the EVM. *ContractFuzzer* uses fuzzing, restricted to the Application Binary Interface (ABI) specifications to find vulnerabilities in smart contracts [29]. *Sereum* prevent reentrancy attacks without requiring any semantic knowledge of the contract [40]. *ECfChecker* dynamically checks if the Effectively Callback Free (ECF) object is feasible and executable [26]. *teEther* actively locates an exploit for a contract given only its binary bytecode [31]. More recently, *TXSPECTOR* [47] and [17] leverages on *Datalog*, a language implementing first-order logic with recursion [27], which allows scalability to detect smart contract vulnerabilities.

Furthermore, an evolving online approach to detect smart contracts attacks include *SODA*, which developed 8 applications containing attack detection methods exploiting major vulnerabilities [21] and *EVMPatch*, which features a bytecode rewriting engine which hardens smart contracts [16].

Despite having research directions dictating the discovery and protection of vulnerabilities in smart contracts, it is still difficult to prevent zero-day vulnerabilities from occurring. Potential research directions in smart contracts analysis is growing and desired to further make the use of them secure.

4.2 Privacy Concerns

NFTs are not enumerable as a private registry of property ownership, or a partially-private registry. As such, privacy cannot be attained because an attacker can simply call `ownerOf` for every possible `tokenId` [45].

However, there is a trade-off in between determining privacy, transparency and the choice of processing these off-chain or the use of permissioned blockchains. By leveraging on existing standards for digitalization of documents in Sect. 2.2, use of such an implementation promotes openness and quicker adoption as supposed to a permissioned blockchain; where only invited users are allowed access. Future implementations to *Eth2* may provide a more robust implementation to increase privacy using zero-knowledge proofs such as *zk-SNARKs* (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge, used in *ZCash*) or *zk-STACKs* (Zero-Knowledge Succinct Transparent Argument of Knowledge) [13].

5 Challenges

Despite growing threats, several challenges also exist in overcoming barriers for the adoption of blockchains in businesses, specifically discovering use cases for supply chains.

5.1 Rising Gas Costs

Gas fees exist in *Ethereum* to help keep the network secure by charging a fee for every computation that is executed. This prevents accidental or intentional infinite loops or other computational wastage and serves as a limit to the number of computational steps of code it can execute. Denoted in *Gwei*, each *Gwei* is equal to 10^{-9} *ETH* [5].

As of writing, average gas costs having risen over 700% to almost 200 *Gwei* from over a year ago [46]. This makes written smart contacts with large number of lines of code computationally expensive and impractical for execution in the network. Despite having *MadMax* to detect gas-focused vulnerabilities [25], *EIP-1559* will include a transaction pricing mechanism that dynamically expands/contracts block sizes along with the introduction of *Eth2* [2]. As *Ethereum* is currently in transition towards *Eth2*, we have yet to see how this will greatly affect implementations [11].

Additionally, permissioned blockchains may choose not to employ costing to deploy smart contracts onto the network. This may be counterintuitive for an already invited small pool of users in a permissioned blockchain to constantly innovate due to the lack of rewards. Implementations within a permissioned blockchain may not be easily audited or standardized since it is not made known to the public domain. A comparison with cost may prove difficult but permissioned blockchains do not have digital currencies and means to transact directly.

5.2 Integration

As smart contracts require a definite solution, it is difficult to code in accordance to current regulatory obligations, governance or standards that needs interpretability by humans.

Overall Security in Blockchain. As mentioned earlier in Sect. 4, tackling the security of smart contacts is simply a part of the blockchain ecosystem. We will need to study and consider the greater impact of verifying the source of the information that is being recorded into the blockchain [39] and the growing concern of APTs (Advanced Persistent Threats) [9].

Interoperability with ERP Systems. *SAP SE*, a german multinational company popularly known for its ERP software, has various application programming interfaces (APIs) with which one can access data within its systems [8]. However, in its community forums, *SAP* has limited smart contract functionality with *Hyperledger Fabric* or *Multichain* and has seen obsolescence [33]. This further challenges how interactivity and exchanges can occur between deeply ingrained proprietary accounting systems and evolving blockchain technologies.

Unique Use Cases. Although the presented workflow may apply to most common supply chains, customization might be required to better fit use cases. Such implementations may include custom clearances, dangerous goods, insurance claims, taxation or any additional special rules or regulations. The existing smart contract can be modified and extended so it can be upgraded while preserving their address, state, and balance [12].

5.3 Cross-Contracts on Different Blockchains

Even though the electronic bill of lading (eBL) can be adopted into a non-fungible token (NFT), some blockchains may not be capable of accepting such tokens due to non-compliance of the *ERC-721* standard. This also includes the deployment of smart contracts, which different blockchains require to be rewritten into another language for proper compilation and use. A possible direction for this is to utilize the *Inter-Blockchain Communication* (IBC) protocol in *Cosmos* [24].

5.4 Framework

There is a current lack of an agnostic framework which determines the characteristics of deploying a secure blockchain in the supply chain. Potential metrics could cover feasibility, performance and pruning requirements.

6 Conclusion

Our work has demonstrated how we can map a typical business process to the *Ethereum* blockchain by writing smart contracts in *Solidity*. This agnostic approach not only provides a secure supply chain but also simplifies and automates several processes to facilitate greater transparency and ease of access to various parties via the use of smart contracts. We identified existing problems and challenges for adoption and also provide potential future research directions to enable blockchain for supply chains.

References

1. BTCBUSD—Binance Spot. https://www.binance.com/en/trade/BTC_BUSD
2. EIP-1559 - Fee market change for ETH 1.0 chain. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md
3. Electronic Code of Federal Regulations (eCFR). https://www.ecfr.gov/cgi-bin/text-idx?SID=6885de90742b035794f3c377745ff932&mc=true&node=pt49.5.375&rgn=div5
4. fabiojose/ethereum-ex. https://github.com/fabiojose/ethereum-ex
5. Gas and fees—ethereum.org. https://ethereum.org/en/developers/docs/gas/
6. IBM Food Trust - Blockchain for the world's food supply. https://www.ibm.com/blockchain/solutions/food-trust
7. OpenZeppelin. https://openzeppelin.com
8. SAP API Business Hub. https://api.sap.com
9. Security Advisory for SolarWinds. https://www.solarwinds.com/securityadvisory
10. Smart0tter/tradefinance. https://github.com/Smart0tter/TradeFinance
11. The Eth2 upgrades—ethereum.org. https://ethereum.org/en/eth2/
12. Upgrading smart contracts - OpenZeppelin Docs. https://docs.openzeppelin.com/learn/upgrading-smart-contracts
13. ZK-STARKs - EthHub. https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-starks/
14. UNCITRAL Model Law on Electronic Transferable Records—United Nations Commission On International Trade Law (2017). https://uncitral.un.org/en/texts/ecommerce/modellaw/electronic_transferable_records
15. Incoterms 2020 - ICC - International Chamber of Commerce (2020). https://iccwbo.org/resources-for-business/incoterms-rules/incoterms-2020/
16. EVMPatch: timely and automated patching of ethereum smart contracts. In: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, Vancouver, B.C., August 2021. https://www.usenix.org/conference/usenixsecurity21/presentation/rodler
17. Smart contract vulnerabilities: vulnerable does not imply exploited. In: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, Vancouver, B.C., August 2021. https://www.usenix.org/conference/usenixsecurity21/presentation/perez
18. Albert, E., Gordillo, P., Livshits, B., Rubio, A., Sergey, I.: EthIR: a framework for high-level analysis of ethereum bytecode. CoRR abs/1805.07208 (2018). http://arxiv.org/abs/1805.07208
19. Azzi, R., Chamoun, R.K., Sokhn, M.: The power of a blockchain-based supply chain. Comput. Ind. Eng. **135**, 582–592 (2019)
20. Brent, L., et al.: Vandal: a scalable security analysis framework for smart contracts. CoRR abs/1809.03981 (2018). http://arxiv.org/abs/1809.03981
21. Chen, T., et al.: SODA: a generic online detection framework for smart contracts. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, 23–26 February 2020. The Internet Society (2020). https://www.ndss-symposium.org/ndss-paper/soda-a-generic-online-detection-framework-for-smart-contracts/
22. CryptoKitties: Cryptokitties—technical details. https://www.cryptokitties.co/technical-details

23. Frank, J., Aschermann, C., Holz, T.: ETHBMC: a bounded model checker for smart contracts. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 2757–2774. USENIX Association, August 2020. https://www.usenix.org/conference/usenixsecurity20/presentation/frank
24. Goes, C.: The Interblockchain Communication Protocol: An Overview (2020)
25. Grech, N., Kong, M., Jurisevic, A., Brent, L., Scholz, B., Smaragdakis, Y.: Madmax: surviving out-of-gas conditions in ethereum smart contracts. Proc. ACM Program. Lang. **2**(OOPSLA) (2018). https://doi.org/10.1145/3276486
26. Grossman, S., et al.: Online detection of effectively callback free objects with applications to smart contracts. CoRR abs/1801.04032 (2018). http://arxiv.org/abs/1801.04032
27. Immerman, N.: Descriptive Complexity. Springer, Heidelberg (1999). https://doi.org/10.1007/978-1-4612-0539-5
28. International Cargo Express: Bill Of Lading Explained: The Complete Beginner's Guide (2019). https://www.icecargo.com.au/bill-of-lading
29. Jiang, B., Liu, Y., Chan, W.K.: ContractFuzzer: fuzzing smart contracts for vulnerability detection. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ASE 2018, pp. 259–269. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3238147.3238177
30. Kalra, S., Goel, S., Dhawan, M., Sharma, S.: ZEUS: analyzing safety of smart contracts. In: 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, 18–21 February 2018. The Internet Society (2018). http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_09-1_Kalra_paper.pdf
31. Krupp, J., Rossow, C.: Teether: gnawing at ethereum to automatically exploit smart contracts. In: 27th USENIX Security Symposium (USENIX Security 18), pp. 1317–1333. USENIX Association, Baltimore, MD, August 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/krupp
32. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. CCS 2016, pp. 254–269. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2976749.2978309
33. Misiorek, G.: SAP Hyperledger Retirement - SAP Q&A (2021). https://answers.sap.com/questions/13220261/sap-hyperledger-retirement.html
34. Mueller, B.: b-mueller/smashing-smart-contracts: Write-ups on security analysis of Ethereum smart contracts using symbolic execution and constraint solving (2018). https://github.com/b-mueller/smashing-smart-contracts
35. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. Technical report, Manubot (2019). https://git.dhimmel.com/bitcoin-whitepaper
36. NCC Group: Decentralized Application Security Project (DASP) - Top 10 (2018). https://dasp.co/
37. Nikolic, I., Kolluri, A., Sergey, I., Saxena, P., Hobor, A.: Finding the greedy, prodigal, and suicidal contracts at scale. CoRR abs/1802.06038 (2018). http://arxiv.org/abs/1802.06038
38. Palladino, S.: The Parity Wallet Hack Explained - OpenZeppelin blog (2017). https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/
39. Reyna, A., Martín, C., Chen, J., Soler, E., Díaz, M.: On blockchain and its integration with IoT. Challenges and opportunities. Future Gener. Comput. Syst. **88**, 173–190 (2018). https://doi.org/10.1016/j.future.2018.05.046, https://www.sciencedirect.com/science/article/pii/S0167739X17329205

40. Rodler, M., Li, W., Karame, G.O., Davi, L.: Sereum: protecting existing smart contracts against re-entrancy attacks (2018). http://arxiv.org/abs/1812.05934
41. Stortz, R.: crytic/rattle: evm binary static analysis. https://github.com/crytic/rattle
42. Szabo, N.: Smart contracts: building blocks for digital markets. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html
43. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Bünzli, F., Vechev, M.: Securify: practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS 2018, pp. 67–82. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3243734.3243780
44. Vessenes, P.: Deconstructing the DAO attack: A brief code tour (2016). https://vessenes.com/deconstructing-thedao-attack-a-brief-code-tour/
45. Entriken, W., Shirley, D., Evans, J., Sachs, N.: EIP-721: ERC-721 Non-Fungible Token Standard. https://eips.ethereum.org/EIPS/eip-721
46. YCharts: Ethereum Average Gas Price. https://ycharts.com/indicators/ethereum_average_gas_price
47. Zhang, M., Zhang, X., Zhang, Y., Lin, Z.: TXSPECTOR: uncovering attacks in ethereum from transactions. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 2775–2792. USENIX Association, August 2020. https://www.usenix.org/conference/usenixsecurity20/presentation/zhang-mengya