symmetry

MDPI

Article

Trusting Testcases Using Blockchain-Based Repository Approach

Abdulla Al Zaabi *,†, Chan Yeob Yeun † and Ernesto Damiani

Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology, Abu Dhabi 127788, United Arab Emirates; chan.yeun@ku.ac.ae (C.Y.Y.); ernesto.damiani@ku.ac.ae (E.D.)

* Correspondence: abdulla.alzaabi@ku.ac.ae

† These authors contributed equally to this work.

Abstract: Modern vehicles have evolved to support connected and self-driving capabilities. The concepts such as connected driving, cooperative driving, and intelligent transportation systems have resulted in an increase in the connectivity of vehicles and subsequently created new information security risks. The original vehicular ad-hoc network term is now emerged to a new term, Internet of Vehicles (IoV), which is a typical application of symmetry of Internet of Things (IoT). Vehicle manufacturers address some critical issues such as software bugs or security issues through remote updates, and this gives rise to concerns regarding the security of updated components. Moreover, aftermarket units such as those imposed by transportation authorities or insurance companies expose vehicles to high risk. Software testing aims to ensure that software products are reliable and behave as expected. Many commercial and open-source software products undergo formal certifications to increase users' confidence in their accuracy, reliability, and security. There are different techniques for software certification, including test-based certification. Testcase repositories are available to support software testing and certification, such as the Linux Test Project for Linux kernel testing. Previous studies performed various testing and experimental evaluation of different parts of modern vehicles to assess the security risks. Due to the lack of trusted testcase repositories and a common approach for testing, testing efforts are performed individually. In this paper, we propose a blockchain-based approach for a testcase repository to support test-based software and security testing and overcome the lack of trusted testcase repositories. The novel concept Proof-of-Validation to manage global state is proposed to manage updates to the repository. The initial work in this study considers the LTP test suite as a use case for the testcase repository. This research work is expected to contribute to the further development in including evidence generation for testing verification.

Keywords: autonomous; vehicles; software; testing; certifications; blockchain; oracles; hyperledger

Citation: Al Zaabi, A.; Yeun C.Y.; Damiani E. Trusting Testcases Using Blockchain-Based Repository Approach. *Symmetry* **2021**, *13*, 2024. https://doi.org/10.3390/sym13112024

Academic Editors: Kuo-Hui Yeh, Chunhua Su and Shi-Cho Cha

Received: 31 August 2021
Accepted: 12 October 2021
Published: 26 October 2021

1. Introduction

Software systems play an important role in many domains of our life, such as healthcare, transportation, telecommunication, government, and business. Thus, ensuring the correctness and reliability of applications and software systems that support these domains is very crucial, irrespective of whether they are open-source or commercial. Government agencies and standard bodies are continuously developing regulations and guidelines for the development and deployment of software systems. Certification of software products aims to increase users' confidence in their correctness and reliability. Different certification techniques exist for software systems. The common ones are model-based certification and test-based certification. Model-based certification relies on model-based testing, where an abstract model representing a software system to be certified has properties as claimed by the developer or vendor. Test-based certification relies on executing testcases and generating evidence to support the claims or properties of the software system [1]. Software certification is not limited to commercial software systems; open-source software systems undergo software certifications as well.

Blockchain is a digital, secure, decentralized collection of records/transactions. It can be likened to a distributed database in the form of blocks, which maintains records while preserving the integrity of the chain or database. Blockchain eliminates the need for a trusted and centralized authority to govern or verify the integrity of data. The introduction of oracles in blockchain widens the scope of blockchain to data not stored on the chain (off-chain). Blockchain is now being adopted in various applications in different domains, such as cryptocurrency, healthcare, and energy [2,3].

Existing software testing and certification techniques require the testcases to provide proofs that given properties hold. Moreover, the test process is usually carried out by a trusted evaluation body in a controlled environment. There are existing testcase repositories to support test-based software testing. Due to the absence of centralized or trusted testcases for software and security testing of autonomous vehicles, the testing procedures are implemented individually.

In this paper, we present an approach to support testing activities for software products and systems. The proposed approach aims to contribute to the area of testcase development and provide a trustworthy repository, using blockchain-based technology. While testcase repositories exist for some domains, such as the Linux Test Project (LTP), there is no open repository based on distributed technology. In our future works, we aim to support not only testcase generation but also the certification of software systems in a trusted decentralized manner. The contribution of this article is therefore threefold: We (i) define blockchain-based repository for test-based software testing, (ii) describe a validation mechanism named Proof-of-Validation (PoV) to manage the global state of the blockchain for updates to the testcase repository, and (iii) introduce the autonomous vehicle conceptual model to support the development of testcases and define test categories.

The remainder of this article is organized as follows: Section 2 provides a greater emphasis on blockchain technology, including the types of blockchain oracles and different implementations of blockchain. Section 3 describes a conceptual model for autonomous vehicles. Section 4 describes our approach to the distributed repository of testcases based on blockchain. Finally, Section 5 presents our concluding remarks and the scope for future work.

2. Related Work

This section presents an overview of blockchain technology, blockchain oracles, software testing, and a conceptual model for autonomous vehicles.

2.1. Overview of Blockchain Technology

The term "blockchain" was introduced in the paper that first explained the concept of Bitcoin by Satoshi in 2008 [4]. Blockchains relies on consensus mechanisms to ensure the reliability and consistency of the data and transactions. Different blockchain technologies employ different consensus mechanisms. Some of the widely adopted consensus mechanisms include: Proof-of-Work (PoW), Proof-of-Stake (PoS), and Practical Byzantine Fault Tolerance (PBFT) [5].

Hyperledger has been created to advance cross-industry blockchain technologies. It is a multi open-source project, founded by the Linux Foundation. There are different projects under the umbrella of Hyperledger: Hyperledger Fabric, Hyperledger Sawtooth, and Hyperledger Indy, in addition to other tools and libraries such as Hyperledger Caliper and Hyperledger Ursa.

Hyperledger Fabric is a modular blockchain framework for developing a wide range of blockchain-based applications. It was designed considering a high level of privacy to allow for business organizations and government entities to utilize blockchain technologies in different applications. Notably, it supports the development of non-incentivized consensus algorithms. One of the key advantages of the Hyperledger Fabric framework is the support offered for different formats of ledger data. Moreover, the fully pluggable consensus mechanism is supported.

The Hyperledger Fabric architecture consists of different node types, which are communication entities of the blockchain. There are three types of nodes: client, peer, and orderer. A peer in the Fabric is responsible for validating transactions and maintaining the state of the ledger, and it can also have a special role called the endorser. The orderers are responsible for creating new blocks and updating the ledger by adding orders. All nodes are registered and authenticated via a Fabric special entity called the membership service provider (MSP). The Fabric utilizes special applications called chaincode, which are smart contracts written in either Go, Java, or other supported languages. The process given below is followed.

1. All nodes are registered in the MSP.
2. A channel is created with a ledger. The ledger is initialized during channel creation.
3. A policy is created with defined endorsement criteria.
4. A chaincode is deployed in the ledger.
5. Entity submits a transaction proposal to all required endorsers as specified in the endorsement policy. A transaction is created when an entity invokes a function in chaincode, which could be a read function or update function.
6. The proposal is validated by endorsers, and the chaincode is executed and ledger data are returned as a proposal response.
7. The transaction proposal, response, and ledger data are sent to the orderer as a transaction.
8. A block is created by the orderer using the transaction and the block is returned to the endorsers.
9. Endorsers update the state of the ledger and add the block after validation.

The ordering service in the Fabric can support only one order or multiple orderers (SOLO and Kafka). The Fabric does not employ the concept of incentive-based consensus protocols.

2.2. Overview of Oracles

Oracles are decentralized data feeds that provide external data to the blockchain [6,7]. Traditional blockchain implementations and smart contracts cannot communicate with data outside the blockchain network or off-chain data. Oracle blockchain comes into play to bridge the gap between off-chain and on-chain data, widening the scope of blockchain and smart contracts to operate beyond the on-chain data. Blockchain oracles are not the network itself or the data store for off-chain data; they are rather a layer that allows for the blockchain to query and authenticate external data sources as shown in Figure 1. In the absence of blockchain oracles, smart contracts are limited to on-chain data. The data enters the blockchain by oracles in different formats, such as votes from voting terminals or transactions from payment systems.

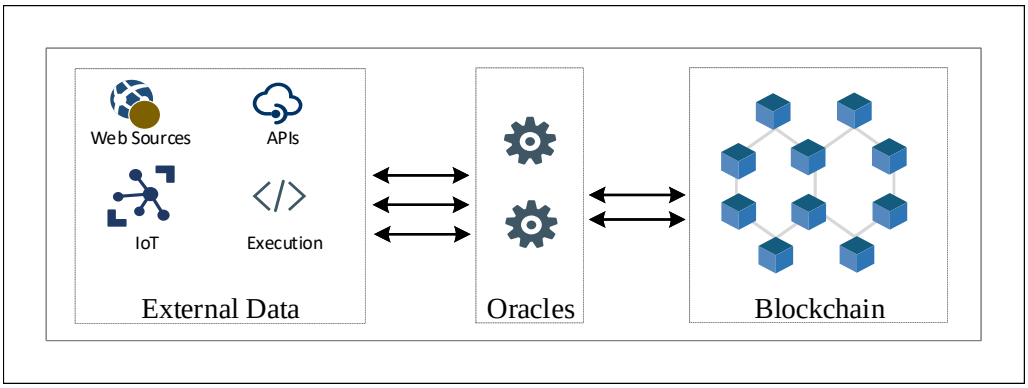

Figure 1. Interaction of oracles with blockchain and external sources.

Oraclize (or Provable Things) is the most widely used oracle in blockchain. It is a leading service for blockchain applications serving thousands of requests daily on platforms

such as Hyperledger Fabric and Ethereum. It providers a layer to allow for smart contracts to access and query data off-chain.

Moreover, it overcomes the limitations of decentralized oracles via the concept of authenticity proof. The authenticity proof verifies that the data fetched from external sources are genuine and not tampered with. The authenticity proofs can be built on trusted execution environments or auditable virtual machines. The authenticity proofs of Oraclize maintain the security model of blockchains. The data do not need to be standardized as smart contracts can access data from external sources.

Provable uses authenticity proofs to overcome the limitation of the trust model in oracles. It establishes a secure connection between smart contracts and external data sources. Therefore, data received or computations performed externally are transmitted securely between the smart contracts and external data sources.

ChainLink is a decentralized blockchain oracle service built with modularity in mind [8]. The fundamental functional objective thereof is to bridge off-chain and on-chain networks. ChainLink enables the interaction of smart contracts with external data sources in a trusted manner. The trust model in ChainLink is distributed between ChainLink nodes (off-chain) and blockchain (on-chain).

In ChainLink, the user smart contract (USER-SC) requests data or queries ChainLink nodes. The USER-SC interacts with ChainLink nodes through the ChainLink smart contract (CHAINLINK-SC). The CHAINLINK-SC is an on-chain smart contract consisting of three main contracts, namely, a reputation contract, an order-matching contract, and an aggregating contract. The reputation contract keeps track of the reputation of the oracle service provider. The order-matching contract is responsible for the service level agreement (SLA) aspects of the oracle service. It receives proposals of the SLA and keeps track of the SLA parameters. It also collects bids from oracle providers. The aggregating contract is responsible for aggregating responses from oracle provides, and it calculates the final result of the query.

2.3. Overview of Ipfs

The InterPlanetary File System (IPFS) is a peer-to-peer distributed file system used to store and version large data [9,10]. Files added to IPFS receive a unique hash that corresponds to the content of the file. The IPFS stack is divided into a stack of sub-protocols that are responsible for different functionalities such as identities, network, routing, exchange and naming. Similar to other peer-to-peer networks, the IPFS uses Distributed Hash Tables (DHTs) to coordinate and maintain metadata, which supports content discovery.

The block exchange in IPFS uses a novel protocol named BitSwap, which is a BitTorrent inspired protocol to exchange blocks between peers. In IPFS, "want_list" and "have_list" correspond to the blocks requested by clients and blocks that are being offered by other clients. The BitSwap protocol employs BitSwap Credit to incentivize nodes and to overcome the issue of free-loading. BitSwap ledger keeps track of transfer history and helps avoid tampering. During connection initialization, peers share their BitSwap ledger, and if the state of the ledger is incorrect, the peer loses its credit or debit and the ledger gets reinitialized.

2.4. Overview of Trust

Trust is an essential element in building confidence in software products. The general definition of trust refers to the firm belief in the truth and reliability made by individuals to the actions and consequences of other parties [11,12]. In software testing, trust is viewed in the system's ability to behave as expected and to be resilient. Due to the existence of multiple dimensions in software development such as software, hardware, environment, threat space, and others, establishing trust is difficult [13]. The International Standard Organization (ISO) document "Glossary of IT Security Terminology" provides the following definitions.

- Certification Authority: an entity trusted by one or more users to create and assign certificates.
- Certification: Procedure by which a third party gives written assurance that a product, process, or service conforms to a specified requirement.

In security certification, the trust in certificates issued by the certification authorities increases customer confidence in the claims made by the service provider on the security properties of the system. The cost of international certification of software systems remains very high [13]. Security evaluation activities based on test-based testing achieve a lower trust level than the one achieved by the full certification process. The assumption made in this research is that the evidence generated by the security testing is mainly based on the trust that a customer has in the testcases.

2.5. Overview of Software Testing and Certifications

There are different techniques for software verification and validation including model-based and test-based. The model-based verification is based on an abstract model that describes the target software or system undergoing testing whereas, in test-based testing, requirements are verified by the execution of testcases. In test-based testing, the outcome of testcase execution determines if the software meets the claimed or desired property. Given below are a few examples of There arethe different types of test-based software testing [13].

- Unit Testing: tests specific components or code blocks.
- Integration Testing: tests integration of different subsystems or software components.
- Acceptance Testing: tests the entire software product to ensure conformance with end-user requirements.

Each type of testing described above may involve different techniques based on the purpose of testing, which may require the design and execution of specific testcases. Below are some examples of some of the testing techniques.

- Functionality Testing: to ensure that the functions meet the intended software requirements.
- Performance Testing: to ensure the system stability and robustness under a specific workload.
- Regression Testing: to ensure that update in specific component did not introduce new issues in unchanged software components.
- Security Testing: to ensure that security mechanisms are implemented to protect sensitive data and aims to uncover vulnerabilities that may be exploited.

Because the article is focused on test-based testing, the remainder of this section briefly reviews some security certification schemes.

2.5.1. Common Criteria

Common Criteria (CC) is a general model for test-based certification. It is an internationally recognized set of guidelines for the security of information technology products [14]. It ensures that the processes of specification, implementation, and evaluation of any certified product are conducted in a thorough and standard manner. In other words, it ensures that a product has been independently verified to behave securely as measured against internationally agreed specifications.

In CC, the target system of software undergoing evaluation is called the target of evaluation (TOE). CC can be applied to a variety of computer systems including operating systems, databases, network devices, and smart cards. CC terms from [13] are explained in Table 1.

Table 1. Common criteria terms.

Term	Name	Description
CC	Common Criteria	Common Criteria Methodology for Information Technology Security Evaluation.
CEM	Common Evaluation Methodology	Common Evaluation Methodology for Information Technology Security Evaluation.
PP	Protection Profile	An implementation-independent set of security requirements for a category of products.
ST	Security Target	A set of implementation-dependent security requirements for a specific product.
TOE	Target of Evaluation	The product under evaluation.
TFS	TOE Security Functionality	The security functionality of the product under evaluation.
TSS	TOE Summary Specification	A description of how a TOE satisfies security functional requirements in the product under evaluation.
SFR	Security Functional Requirement	A requirement for security enforcement by the TOE.
SAR	Security Assurance Requirement	A requirement to assure the security of the TOE

Protection profile (PP) is an important component of CC. PP specifies the type of security requirements for a class of equipment. However, it does not provide details on how the specified security requirements are implemented. Instead, it provides generic security evaluation criteria to confirm the product's conformance with the security requirements to that family of information system products. Some examples of PPs are listed below.

- Biometric Verification Mechanisms
- Base Protection Profile for Database Management Systems
- PP for the Gateway of a Smart Metering System
- Application VPN client/Client VPN Application, Version 1.0

2.5.2. The Trusted Computer System Evaluation Criteria

The Trusted Computer System Evaluation Criteria (TCSEC) is a standard developed by the Department of Defense of the United States (DoD) to address the issues of standardizing computer security controls. The TCSEC (also known as the Orange Book) was used to evaluate, classify, and select computer systems that deal with classified information for processing, storing, or retrieval. It was not only used for military systems but also by the government and industries. It was later replaced by CC [15].

The Orange Book defines the following divisions and classes for security: D, C, B, and A. Each division represents a set of security features that must be satisfied by a software system to be classified for the category as shown in Table 2.

Table 2. The Orange Book categories.

D Minimal Protection	C Discretionary Protection
Software systems that have been evaluated and failed to meet the requirements of other categories	C1 Discretionary Security Protection C2 Controlled Access Protection
B Mandatory Protection	**A Verified Protection**
B1 Labeled Security Protection B2 Structured Protection B3 Security Domains	A1 Verified Design-Beyond A1

2.5.3. Society of American Engineers Cyber Security Guidebook for Cyber-Physical Vehicle Systems

The Society of American Engineers (SAE) published an international guidebook for cyber-physical vehicle systems: SAE J3061 [16]. This guidebook defines a process framework for cyber-physical vehicle systems security lifecycle. It provides high-level guides

and information on the best practices for cyber-physical systems lifecycle. The framework aims to aid the identification and assessment of cyber security threats and the design of cyber security into cyber-physical vehicle systems. It defines a security lifecycle strongly influenced by the safety lifecycle defined in ISO 26262 [17]. SAE J3061 defines different phases of the system lifecycle: the concept phase, development, production operation, and service. In addition, it suggests supporting processes such as requirements, change, and quality management.

As a joint development effort between SAE and ISO, J3061 is not merged into ISO-SAE 21434 [18]. ISO-SAE 21434 considers all phases of the vehicle lifecycle; ranging from design and development, production, operation, and maintenance to decommissioning. The purpose of the standard currently being developed is to define a structured process to ensure cybersecurity engineering for in-vehicle systems and reducing the potential for a successful cyber attack. The SAE vehicle electrical system security committee started working on more in-depth guidance documents for the cybersecurity of automotive systems as illustrated in Table 3.

Table 3. SAE guides.

Guide	Description
SAE J3061-1	Automotive Cybersecurity Integrity Level
SAE J3061-2	Security Testing Methods
SAE J3061-3	Security Testing Tools
SAE J3101	Requirements for Hardware-Protected Security for Ground Vehicle Applications
SAE J3138	Guidance for Securing the Data Link Connector (DLC)

2.5.4. Linux Test Project

Before the introduction of LTP, Linux kernel testing was carried out informally. The latest kernel versions are tested by executing real applications on workstations or servers, and the identified bugs or performance issues would be reported by individuals or organizations [19]. Despite the efforts of many Linux developers in performing unit testing, there was no community-wide testing framework for Linux kernel. LTP was therefore introduced to provide the complete test suite for Linux kernels. LTP aims to improve the Linux kernel by facilitating automated testing of kernel functionalities.

LTP aims to allow users and developers to validate Linux kernels, specifically the stability and robustness of the kernel. Before the introduction of the LTP, there was no formal testing environment available, the LTP significantly advanced Linux testing and assurance. It facilitates both automated testing of kernel functionalities and testing of individual components via running of individual testcases. Furthermore, it allows developers to define new tests and integrate existing benchmarks and test results analysis.

LTP plays an important role as a part of daily testing activities for Linux kernel testing. At the beginning, LTP supported the writing of testcases in C. Currently, it supports the writing of testcases in Portable Shell. It was introduced by Silicon Graphics, but it is currently being maintained by many organizations and developers, including IBM, Cisco, Fujitsu, SUSE, and Red Hat. Moreover, it undergoes regular maintenance and cleanup. The latest version of the LTP test suite (September 2020) has over four thousand testcases.

LTP defines Pan as the test driver for the test suite. Users can define the list of testcases to run pass it to Pan for execution. Pan reports the output of the test program, that is, whether it passed or failed and where it failed.

In addition to automated testing, users can run individual test programs as mentioned earlier. The test programs in LTP are organized into multiple categories as follows:

- Kernel: contains test programs related to kernel, such as filesystems, io, ipc, and system calls.
- Network: contains test programs related to network, such as ipv6, multicast, rpc, and sctp.
- Commands: contains user-level commands, such as ar, ld, ldd, and nm used in application development.
- Misc: Miscellaneous tests that do not fall under the categories mentioned above, such as crash and floating-point math set of tests.

LTP was designed to allow developers to develop new testcases and contribute to the project. It provides basic templates for developers to write new testcases to guide developers in writing new testcases, and these templates follow LTP test program criteria.

LTP provides a set of guidelines and an LTP test interface to allow developers to write new testcases or modify existing ones. One of the main features of LTP is the ease of use, because of which the use of the test suite regularly in the development process is encouraged. Testcase templates are provided for developing new testcases, both in Portable Shell and C. Through the introduction of test writing guidelines and general rules, the aim of LTP is to keep uniform coding standard across the test suite.

Three main functions are defined in a testcase: main, setup, and cleanup. The entry point of the testcase is defined in the main function. The functions undergoing testing in the testcase are performed in the main function, and the outcome of the execution is reported. The initialization of resources for the testcase is defined in the setup function, including signals, signal handlers, and temporary files. The cleanup function frees the resources being initialized in the setup function in a reserved order. The cleanup is optional and may be called for at any point—for example, when a testcase fails in the main function. The test results are reported with variable int types. The Table 4 below illustrates some of the result types and descriptions thereof.

Table 4. LTP keywords for testcase results.

TTYPE	Description
TPASS	Test has passed
TFAIL	Test has failed
TINFO	General information message
TWARN	Warning message, which does not stop test execution
TBROK	Test broken message, which indicates failure in test preparation phase.
TCONF	Incompatible current configuration, such as syscall not implemented or unsupported architecture type

Each testcase is either written in C or Portable Shell and converted to binary. The testcases can have configuration stored in environment variables or passed through command line arguments. The output of the testcase execution is printed onto the stdout and reports. A basic template for C language testcase is shown below.

1. Self-contained: a testcase can be executed independently.
2. Testcase outcome: a pass or fail outcome of testcase execution must be detected within the testcase itself.
3. Return value: the return value should indicate the result of testcase execution. The value 0 is returned when testcase execution is successful.

2.6. Repository-Based Blockchain Approaches

Different blockchain approaches have been explored within the healthcare industry to manage the Electronic Health Records (EHR) of patients driven by the unified goals of improving privacy, reducing cost and improving healthcare services. One of the examples

is MedRec, which is a management system for EHRs developed by MIT [20]. Although MedRec is proposed in the context of medical records, it is a distributed system for personal information and identity control. The architecture of MedRec is based on Ethereum network, and it uses Proof-of-Authority as a consensus mechanism. Medical providers are enrolled as an authority when voted in by all nodes of the network. The architecture is extended to provide a network of trusted data repositories. Other studies proposed blockchain-based repositories for other domains, such as Virtual Network Functions (VNF) packages. VNF packages are usually shared and deployed in the form of packages. For easy deployment, VNF packages are available in marketplaces. The work in [21] is focused on addressing a common challenge of traditional VNF solutions, which is establishing a trusted computing environment. The author proposed a trusted repository for VNF packages to provide a decentralized approach for verifying the package integrity, without relying on third part remote attestation. The package repository is protected by a repository manager, which acts as an intermediate layer between the users and the repository. The Proof-of-Concept verifies the metadata of VNF packages, whereas the actual packages are stored in untrusted external data storage. BUNKER [22] extended the work in [21] and stored the hash of VNF package stored in traditional untrusted database system in the blockchain, along with the VNF package metadata.

Blockchain technology-based repositories gained significant attention in the field of academic diploma issuance and verification to overcome the issues of the increasing number of forged academic certificates. Blockcerts is a blockchain-based application for academic diplomas management developed by MIT [23]. The application is blockchain agnostic, which allows Blockcerts to work in tandem with different blockchain technologies such as Bitcoin, Ethereum, and Hyperledger. The primary limitations of Blockcerts included bulk certificates upload and revocation system [24]. Docschain was proposed to overcome some of the known limitations of Blockcerts, and it introduced several features for document verification including bulk documents upload and hard copies of previously awarded certificates.

2.7. Analysis of Security Testing

The testing-related literature discussed earlier, ensures that the system undergoing testing behaves correctly to increase the confidence and trust of users. The rationale behind such testing frameworks and schemes is to provide a common language and unify the understanding of requirements testing and evaluation. The common criteria provide a catalog of reusable security functional and assurance requirements and require an evaluation body to inspect and analyze all evidence provided by developers. Other types of testcases were originally created by open source communities such as LTP. The LTP evolved to include thousands of testcases and is now managed by closed communities. As of the time of writing of the article, there are no trusted open testcase repositories for security testing, especially in the area of autonomous vehicles.

In this article, we address the preceding problem by proposing a blockchain-based approach for trusted testcases repository. The scheme relies on test-based security testing with a trusted testcase repository based on the blockchain technology. With the absence of defined threats, it is extremely difficult to provide assurance on the security measures taken. For a clear understanding of the scope of autonomous vehicles, and to support the development of testcases, a conceptual model was developed for autonomous vehicles. The proposed conceptual model aims to group related autonomous vehicle components to support the identification of security threats and the development of testcases. A future direction of this research is to evaluate security testing and certification in the area of autonomous vehicles.

3. Enhanced Autonomous Vehicle Conceptual Model

This section describes the enhanced conceptual model of autonomous vehicles that was introduced in our previous work [25] with an enhanced edge cloud section of the

model. The model focuses on grouping related autonomous vehicles components into a category. The model defines three categories consisting of three main areas: vehicle, communications, and edge cloud as shown in Figure 2. One of the main aims of the conceptual model is to define categories for autonomous vehicle testing.

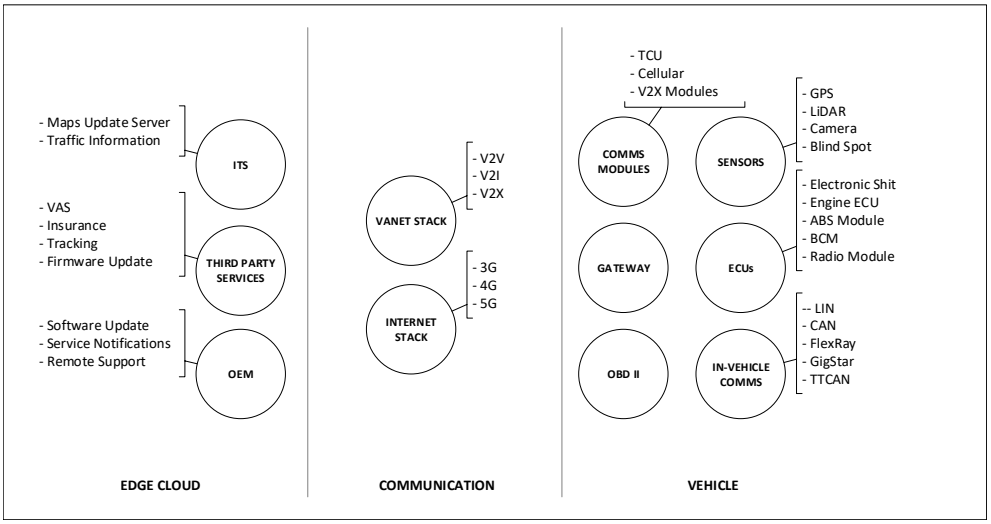

Figure 2. Autonomous vehicle conceptual model.

3.1. Vehicle

The vehicle section of the conceptual model covers the physical boundary of the vehicle including related components. The components that are relevant to this research and that support the security modeling and analysis of autonomous vehicles are considered. The components of the vehicle section are categorized into three main groups—electronic control units (ECUs), sensing, and inter-vehicle communication. The ECUs behave as the brain behind the vehicle operation, sensors are responsible to feed ECUs with the required information to operate accordingly. Inter-vehicle communication is focused on the internal communication between different components of the vehicle, including internal networks. Several types of networks exist in modern vehicles autonomous vehicles with different speeds, autonomous vehicles may incorporate additional networks. Based on the functionality and speed required for ECUs, each ECU is connected to at least one of the internal networks of the vehicle. To facilitate communication between different communication buses, a gateway ECU is used. Modern vehicles are now equipped with an On-board diagnostic port (OBD-II), which provides access to the vehicles' internal network and can be used to request information from various sensors or ECUs.

ECUs are responsible for performing essential operations such as power steering, fuel injection, controlling engine, and door locks. ECUs are categorized based on their functionality [26]. Examples of ECUs categories are Powertrain, Safety, and Body.

Sensors are the main source of data for ECUs to operate correctly and efficiently. Autonomous vehicles are equipped with various sensors to collect information about their surroundings. Examples of sensors equipped are those for GPS, cameras, and LiDAR. ECUs read data from sensors to aid navigation capability, display diving information, displace distance calculation, and many other types of data.

The inter-vehicle communication component of the conceptual model focuses on the communication networks in the physical boundary of the vehicle. The internal networks of vehicles are designed to allow different electronic components in the vehicle to communicate and share information. Vehicles are equipped with multiple internal networks. There are different types of networks in vehicles such as Controller area network (CAN), TTCAN, FlexRay, and local interconnect network (LIN) [27]. CAN is widely used in the automotive industry to support critical applications with a bandwidth of 125 kbit/s. The LIN bus runs at a lower bandwidth (10 kbit/s) to support less critical applications, such as the operation

of power windows and doors lock. FlexRay is a more expensive implementation of CAN, used commonly by BMW. It involves two channels at speeds of 5 and 10 Mbit/s [28,29].

3.2. Communication

In the proposed conceptual model, all external communication components are grouped in the communication later. Future intelligent transportation systems support the sharing of a large volume of information between vehicles and the transportation infrastructure. Moreover, vehicles may interact with each other to share different kinds of information messages such as collision avoidance, traffic information, or to support cooperative driving. In this model, the communication layer consists of two main components: VANET stack and Internet Stack.

The first component is VANET, which is the vehicular ad-hoc network (VANET). VANETs use the control channel of the dedicated short range communication (DSRC) to share information between vehicles. Different types of communication are employed in VANETs, namely, vehicle-to-vehicle (V2V), vehicle-to-station (V2I), vehicle-to-roadside unit (V2R), and vehicle-to-everything (V2X).

Autonomous vehicles share two types of messages among themselves or with intelligent transportation systems. These messages are either beacon messages or special purpose messages [30,31]. Vehicles broadcast heartbeat messages known as beacon messages. Special-purpose messages carry information regarding collisions and warning messages to nearby vehicles [32]. With recent advances in cellular networks, VANETs can either use DSRC or cellular networks as part of future intelligent transportation systems [33]. The Internet Stack component of the model considers the cellular network as the communication means. It includes 4G and 5G data links and physical links [34]. The original concept of VANETs has emerged to a new concept named Internet of Vehicles (IoV). The concept of IoV covers different kinds of technologies including vehicular communication, cellular network, and short-range communication [35–37].

3.3. Edge Cloud

The Edge Cloud layer of the model focuses on vehicle communication with external entities. Autonomous vehicles are expected to generate a large volume of data, which will be shared with various external entities such as intelligent transportation systems, government authorities, and original equipment manufacturers (OEMs). The expansion of wireless and cellular technologies help companies extend their network coverage and capacity to support edge cloud connectivity.

Modern vehicles have already started delivering remote support to vehicles. OEMs execute remote updates to vehicles, for applying patches, fixing issues, or enhancing features. Many services, such as media services, fleet management, and vehicle tracking, and other symmetry applications are currently offered by third-party providers. Given below are some examples of sub-components of edge cloud:

- Traffic information
- Smart city integration
- Maps update service.
- OEM firmware update.
- Third-party dongles firmware update.
- Value Added Services.

The edge cloud services are utilized using the communication layer of the autonomous vehicle conceptual model, which consists of VANET communication stack and cellular communication technologies. The VANETs communication stack is based on the IEEE 802.11p standard for wireless access. Another standard that is focused on secure V2V and V2I communications is the IEEE 1609, which defines the standards, architecture, and interfaces.

Nowadays, insurance companies price insurance tariffs based on driving behavior and history. As a result, various companies install OBD-II compatible dongles to monitor

driving behavior and ensure compliance with the insurance policy. Vehicles are expected to be connected to edge cloud services to accelerate smart and green transportation [38]. As part of smart city integration and intelligent transportation system management [39–41], vehicles are expected to be connected to transportation authorities and share various types of information.

Moreover, the first and last mile is attracting the interest of public transportation. Intelligent transportation systems are envisioned to deliver various services using edge cloud such as high-definition maps, real-time traffic information, dynamic path planning, and other automated driving services.

4. Blockchain-Based Trusted Testcase Repository

This section describes the design of a trusted repository for testcases using blockchain technology. An initial proof of concept (PoC) was implemented herein to assess the challenges in implementing the proposed solution. We focused on the blockchain-based repository for the LTP using Hyperledger Fabric. The LTP suite represents a realistic repository of testcases that are currently being used by many developers in their daily Linux kernel development and testing activities. The PoC was implemented using Hyperledger Fabric and the InterPlanetary File System (IPFS) as a method of distributed data storage.

4.1. Overview of Proposed Approach

The application SDK takes the transaction from the UI, builds a proper proposal, and then signs it with the information of the user who placed the request. SDK sends the proposal to each peer of the organization for proposal validation, who validates the proposal and forwards it to the chaincode. The chaincode reads the required values out of the world state to perform the transaction and produces new results that should be written into the world state. It does not make any changes to the world state at this point. Instead, it stores the testcase file into a private data collection. If the transaction does not go through, the private data collection gets cleaned up after some transactions.

The results are sent to peers from the validating organization for endorsement. The peers endorse the transaction by validating the response and signing the results from the chaincode. The endorsements are sent back to the application SDK. The application takes the endorsements and creates a real transaction based on the transaction proposal, results from the chaincode, and digital signatures from each of the peers. The transaction is then sent to the orderer, who puts it into a block. The orderer does not perform any additional validation. It distributes the block to each of the peers to enter a validation phase to ensure conformance with the endorsement and that the endorsement policy is met if the world state changes since the proposal is received. In this scenario, the testcase is only added if at least two endorsements are received from the validating organization. At this point, the details of the testcase are only accessible to organization 1, which is the validating peer organization. Any new testcase added to the private collection store of organization 1 is sent to a pending validation data store.

Once the transaction is completed and a block is added, the testcase is added to the private data store of organization 1. Peers from organization 2, the client peers, have no access to this data store, which is the pending validation store for testcases. The application SDK notifies validators of a new testcase addition in the private datastore. Peers from validating nodes will have the ability to review the testcase and either approve or reject the testcase. If a sufficient number of members of organization 1 approve the testcase, it gets added to a public data store shared between organizations 1 and 2, as shown in Figure 3. At this point, the testcase is accessible to all peers of the network and can be queried. The accessible data storage is managed by a data service that acts as a wrapper for IPFS layer. The actual testcase is stored in IPFS and referenced by a hash value in the block. Upon retrieving testcase from blockchain, the testcase is read from the IPFS via the IPFS wrapper.

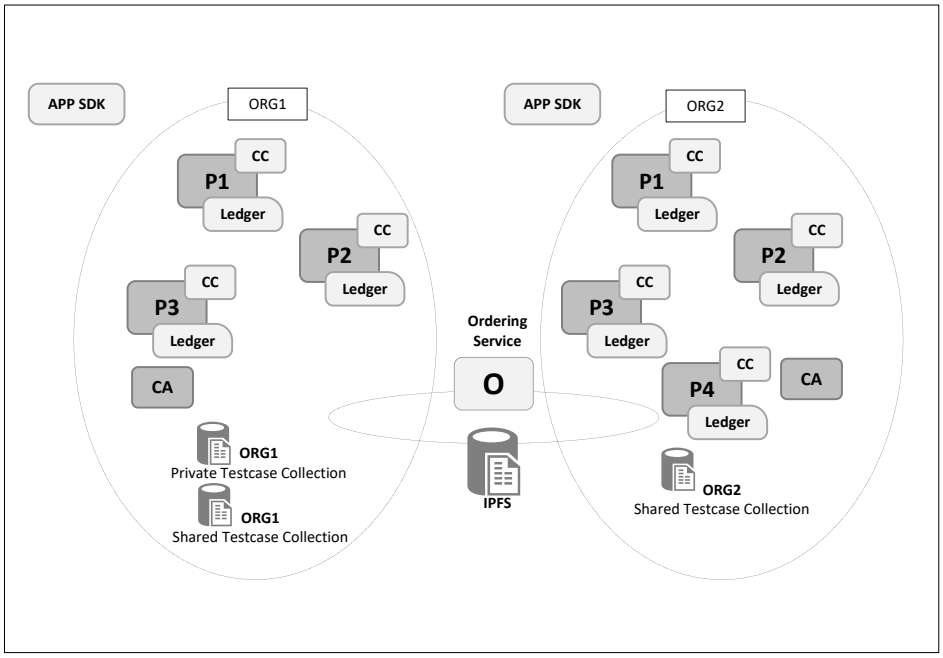

Figure 3. Blockchain using Hyperledger Fabric.

4.2. Technology and Architecture

4.2.1. Blockchain Technology

Hyperledger was created to advance cross-industry blockchain technologies. Hyperledger Fabric supports the development of non-incentivized consensus algorithms, and one of its key advantages is the support offered for different formats of ledger data. Moreover, it supports a fully pluggable consensus mechanism.

4.2.2. Validation Mechanism

The PoV is a newly proposed concept for managing the testcase repository. The PoV algorithm relies on several special nodes known as validating nodes. There are two different types of nodes in this algorithm: validators and client peers.

The validator nodes act as the administrators of the system, similar to those in the case of Proof-of-Authority. To become a validating node, the node must either be confirmed by at least two active members of the validator nodes or must have shown successful block addition to the network. This means that the client peer must have submitted a number of testcases that have been added to the blockchain network. The number of active validation node confirmations or a number of blocks added by a specific peer's transactions can be predefined in the endorsement policy.

In the proposal, a participant node can submit a proposal for a new testcase to be added to the blockchain repository. The testcases will not be added to the public ledger of the confirmed testcases blockchain network but to a ledger of pending verification testcases. At this time, nodes that have a validating role can confirm the testcase based on the predefined policy. Once the testcase is verified by validation nodes, the block is added to the public ledger with a verified status and can be queried by client peers. A high-level process is shown in Figure 4. Initially, the details of the testcase are only visible to nodes in the network that have a validation role. The testcase becomes accessible to other peers in the network once validated by authorized peers.

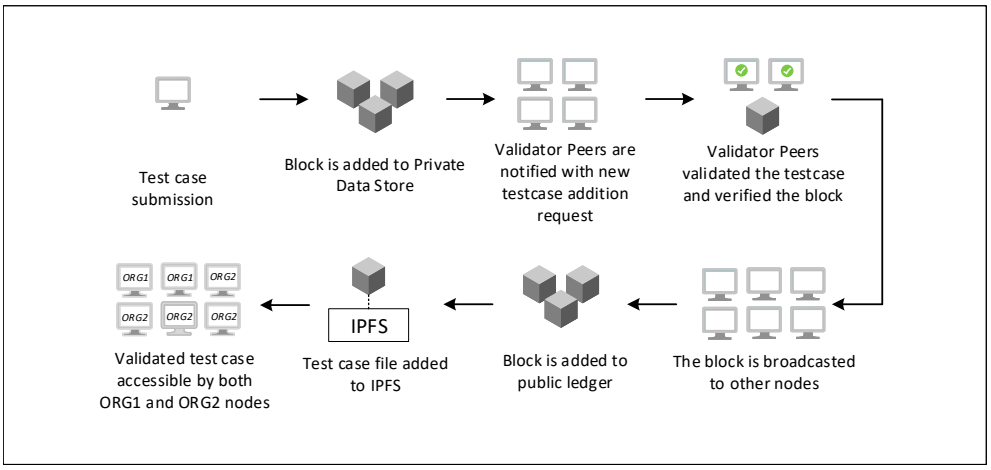

Figure 4. Proposed validation process for testcases blocks.

4.3. Solution Design

The proposed approach consists of different components—some components act as front-end for testers and verifiers interaction and others to manage testcases and interaction with IPFS. Below are some of the components of the front-end part of the proposed solution.

- Testcase Registration: Using the system, developers can submit new testcases or make bug fixes in existing testcases.
- Testcase Verification: Verifies acceptance or rejection of new testcase submission or updates existing testcase.
- Testcase Query: Developers and testers can query the system for testcases, using defined parameters or categories of testing.

The other part of the system is the blockchain-based repository back-end, and it comprises the following components:

- Repository Manager: responsible for registration and categorization of new testcases.
- Test Manager: responsible for searching and filtering based on defined parameters to allow users to select a set of testcases applicable for the target system undergoing testing.
- Data Storage: testcases are stored using distributed data storage, named IPFS [9].

4.3.1. Testcase Registration

The testcase registration is used by developers to submit new testcases to the repository or to update an existing testcase. As part of new testcase registration, developers must provide a short description of the testcase, testcase category, and testcase file. This is represented by a client peer in Hyperledger. The transaction for adding a new testcase will be added to a ledger for pending validation testcases. The role of validation is given to a specific set of nodes as per the proposed validation mechanism.

4.3.2. Testcase Validation

The testcases added to the repository have a validation status. The validation of a testcase checks if the testcase follows the rules and guidelines of testcases and whether or not it is a valid testcase. The verification process of testcases is delegated to a certain set of authorized nodes that act as the administrators or verifiers. The number of validating nodes required can be specified by the endorsing policy of Hyperledger. When a testcase is verified by the required number of validating nodes, a new block is created for the testcase, and it is added to the public ledger.

4.3.3. Query and Filter

The blockchain-based repository can be queried to identify suitable testcases for a specific testing activity. This should allow users to query a class of testcases for a specific component. For example, developers can query the repository for testcases related to security vulnerabilities of the Linux kernel by filtering the common vulnerabilities and exposures (CVE) category of testcases. The chaincode responsible for query and filtering performs the query operation on the public ledger of validated testcases, and it can be invoked by any node in the network, regardless of its role.

4.3.4. Testcase Manager

The testcase manager is responsible for handling all creation and query requests for testcases. For testcase creation, the testcase manager collects all required information and generates additional parameters for the creation of testcases. For querying testcases, the testcase manager collects the required information to perform testcases lookup from the repository manager. The Table 5 shows the details stored for a testcase.

Table 5. Testcase registration parameters.

Attribute	Description	Example
Testcase ID	Identified of testcase	35md..9sdj
Testcase Version	Testcase version	1.0.3
Summary	Testcase summary	[. . .]
Testcase Category	Testcase category	Kernel
Verification Status	Testcase verification status	Unverified, Verified
Verifiers	Verifiers public keys	key1, key2, key3
File Hash	Testcase file hash	Fgsg,,,,asdert3
File type	Testcase file type	Portable Shell or C

4.3.5. Repository Manager

The repository manager is based on a smart contract. The repository manager stores all relevant testcase information in the blockchain network. The repository manager does not store testcase files in the blockchain network to avoid incurring costs of storing large amounts of data in the blockchain network. Instead, it utilizes the data storage to store the testcase file and keeps a copy of the hash of the file only. The data storage is based on IPFS, which is explained in the next section. The integrity of the testcases is preserved by storing the hash of the file in the blockchain. If the testcase file is tampered with, the hash of the file will change, and the file will not be accessible to the system.

4.3.6. Distributed Data Storage

There are different approaches for data storage, such as traditional database and distributed file storage. In this article, the choice of storage is distributed file storage. In distributed file storage, the data are stored in a peer-to-peer network. An example of distributed file storage is IPFS [9]. In IPFS, the file registered in the network can be accessed using network protocol by using the hash of the data. The hash is named the file key, and it can be stored in multiple nodes, preserving the hash of the file. This guarantees the integrity of the file. If the data are altered, the hash value will be different, and the file could not be queried as shown in Figure 5.

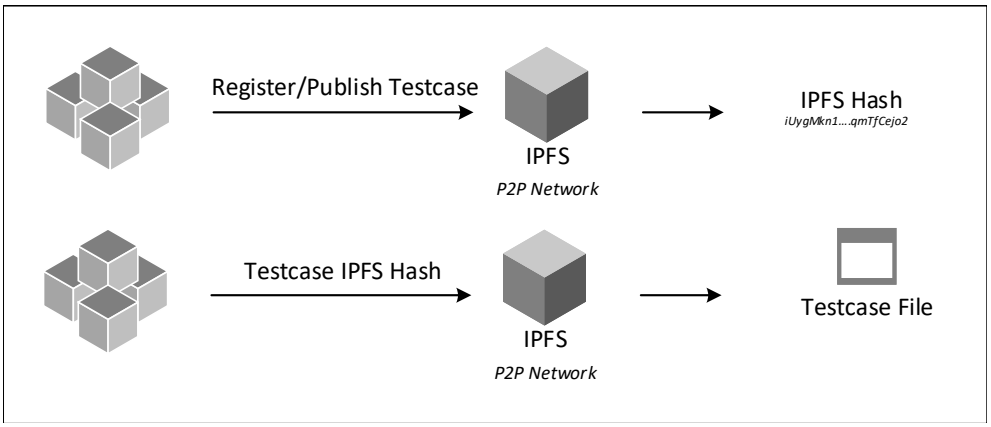

Figure 5. Blockchain usage of IPFS.

4.4. Implementation Analysis

The proposed approach delegates the role of central authority and trusted evaluation of testcases to the validating nodes. For simplicity, we assumed that a set of validating nodes are predefined in the initial phase of the network deployment. The proposed approach lacked the integration of authentication and authorization middleware for the front end of the proposed approach. The validating nodes access should be protected by using authentication and authorization.

The implementation is based on Hyperledger Fabric and IPFS docker images with correct network mapping to allow docker containers of blockchain to communicate with the IPFS service. The first step is to start a fabric network and deploy chaincode implementation in golang. The interaction with the network is carried out using command line, and the user interface will be developed at a later stage. The interaction with the network requires specifying the role of peer; in our case, the peer either has a validator role or a client role. As a validator or a client peer, registering a testcase can be performed by invoking the required function in the test registration chaincode as shown in Figure 6. In the case of missing testcase information, an error message is returned.

```
# Setting up verifier environment variables
echo "REGISTRING A NEW TESTCASE"
export TESTCASE_PROPERTIES=$(cat /opt/test-data/test5.txt)
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
    orderer.example.com --tls --cafile ${PWD}/organizations/
    ordererOrganizations/example.com/orderers/orderer.example.com/msp
    /tlscacerts/tlsca.example.com-cert.pem -C mychannel -n secured -c
     '{"Args":["RegisterTestcase"]}'  --transient "{\"Testcase\":\"
    $TESTCASE_PROPERTIES1\"}"

# In case of successful registration
2021-08-22 06:44:58.423 PDT [chaincodeCmd] chaincodeInvokeOrQuery ->
    INFO 001 Chaincode invoke successful. result: status:200
```

Figure 6. Successful addition of testcase.

A policy is defined for the data collection to segregate data access between client peers and validator peers. In our implementation, we use two different data collections named TestcaseCollection and TestcaseCollectionReceipts. The first collection is non-persistent data collection, which is accessible by validator peers and contains all data related to a testcase, including the test script. The second data collection is persistent data collection, which contains high level information about the testcase, along with verification status. Figures 7 and 8 show examples of data stored in each collection. Once the testcase is verified, the testcase content gets written in IPFS and the hash of the file is stored in the block of the TestcaseReceipts

data collection. The source code for the implementation described in the study will be made available upon request.

_id	category	filetype	summary	version
CVE-00001	CVE	portable c	This test attempts to cause a buffer overflow u…	1.0
CVE-00002	CVE	portable c	High impact NMI bug on x86_64 systems 3.13…	1.0
CVE-00003	CVE	portable c	The test checks that we can not implicitly mar…	1.0
CVE-00004	CVE	portable c	Test for CVE-2016-7042, this regression test c…	1.0
CVE-00005	CVE	portable c	This tests for a use after free caused by a rac…	1.0

Figure 7. Table of some of the fields of the data stored in Testcase Collection.

_id	category	summary	filetype	filecontent
CVE-00001	CVE	This test attempts to cause a buffer overflow using t…	portable c	Ly8gU1BEWC1MaWNlbnNlLUlkZW50aWZpZXI6IE…
CVE-00002	CVE	High impact NMI bug on x86_64 systems 3.13 and …	portable c	Ly8gU1BEWC1MaWNlbnNlLUlkZW50aWZpZXI6IE…
CVE-00003	CVE	The test checks that we can not implicitly mark AIO …	portable c	Ly8gU1BEWC1MaWNlbnNlLUlkZW50aWZpZXI6IE…
CVE-00004	CVE	Test for CVE-2016-7042, this regression test can cr…	portable c	Ly8gU1BEWC1MaWNlbnNlLUlkZW50aWZpZXI6IE…
CVE-00005	CVE	This tests for a use after free caused by a race bet…	portable c	Ly8gU1BEWC1MaWNlbnNlLUlkZW50aWZpZXI6IE…

Figure 8. Table of some of the fields of the data stored in Testcase Receipts Collection.

Once a testcase is verified by the validating peers, the testcase file is added to IPFS and hash to access file is added to the block. The queries performed by peers from either validating or client organizations return testcase information including the hash to access the testcase file. The obtained hash can be used to query the testcase file directly from IPFS.

The Hyperlegder Fabric provides the ability to separate data across multiple channels. This was utilized in the proposed approach to provide data segregation between testcases that are being validated by the validating nodes and the trusted testcases that are available in the shared data collection. The IPFS storage for testcases removes the overhead of storing testcases in the blockchain network. Instead, metadata of the testcases are stored in the ledger. The testcases' integrity is preserved by storing the hash of the testcase in the public ledger. Any modification of the testcase file without updating the corresponding block will result in a mismatch between the hash of the file stored in the ledger and the IPFS.

5. Conclusions and Future Work

Test-based security testing ensures that a software product satisfies its properties of data confidentiality, integrity, and availability. Modern and autonomous vehicles are expected to be highly connected and involve a large volume of messages exchanged and various applications of symmetry. In such critical systems, different parts of the vehicles are expected to perform their functions correctly and safely. As vehicles become more autonomous and connected driven by the envisioned intelligent transportation systems, the attack surface for malicious attempts on the communications among them multiplies.

The Common Criteria defines a numerical rating named Evaluation Assurance Levels from 1 (low) through 7, which is the highest. The EAL was used to describe the depth and rigor of an evaluation. A higher EAL means that the target system has been extensively verified. Protection profile in Common Criteria specifies the type of security requirements for a class of equipment. The TCSEC standard addresses the issues for standardizing computer security controls and is used to evaluate and classify computer systems in regards to the processing and storage of classified information. The TCSEC defines several classes of security ranging from D (Minimal Protection) through A (Verified Protection). Due to the lack of flexibility in mapping security features to security requirements, the TCSEC was not adopted by many organizations. The Linux Test Project provides a complete test suite for Linux Kernel testing, specifically the stability and robustness of the kernel. Furthermore, the LTP allows developers to define new tests and integrate existing benchmarks and test results analysis. The LTP defines templates and guidelines to allow developers and researchers to build their own testcases and contribute to the LTP project. The LTP is maintained by a closed community and submitted testcases are not guaranteed to be added to the LTP repository of testcases. The proposed approach aims to provide a trusted testcase repository for multiple domains based on blockchain technology.

Several blockchain-based repositories were discussed in the article. BlockCerts application is a blockchain agnostic system that can run in tandem with different blockchain technologies. BlockCerts is a digital repository that is used to store and share academic diplomas. Docschain was proposed to overcome some of the limitations of BlockCerts, such as bulk document upload and the support of the verification of hard copies. These repository-based approaches use blockchain technology for specific domains and purposes such as academic credentials, electronic health records, or deployment package verification.

This paper therefore presents a blockchain-based repository for software testing using Hyperledger Fabric and IPFS. The consensuses mechanism proposed is a non-incentivized mechanism named Proof-of-Validation. Network nodes can play a role in validating testcases if they are either selected by other peers or have shown successful block addition to the public ledger of the blockchain network. Writing testcases for autonomous vehicle testing will require a set of guidelines and basic templates, and we plan to develop the same during this research. The proof-of-concept considered the LTP suite as a use case for the repository. In order to use portable shell testcases provided as part of the LTP, testcase files are added to the repository based on their category. Two channels are created in Hyperledger Fabric to separate ORG1 and ORG2 peers, which are validating peers and client peers. Each channel contains its own world state and ledger.

A future direction of this research is to support evidence generation for testcase executions. The adoption of blockchain oracles will support both off-chain and on-chain generation of evidence and execution of test scripts. The off-chain execution will extend the coverage of security testing and support various testing environments. This research aims to extend the approach of testcase repository to trusted testcase execution using the off-chain techniques described in this article.

Author Contributions: Conceptualization, A.A.Z., C.Y.Y. and E.D.; methodology, A.A.Z.; software, A.A.Z.; validation, A.A.Z. and C.Y.Y.; formal analysis, A.A.Z. and C.Y.Y.; investigation, A.A.Z.; writing—original draft preparation, A.A.Z.; writing—review and editing, A.A.Z. and C.Y.Y.; visualization, A.A.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CAN	Controller area network
CC	Common criteria
CVE	Common vulnerabilities and exposures
DSRC	Dedicated short range communication
ECU	Electronic control unit
IPFS	InterPlanetary File System
LIN	Local interconnect network
LTP	Linux Test Project
MSP	Membership service provider
OBD	On-board diagnostic port
OEMs	Original equipment manufacturers
POV	Proof-of-Validation
PP	Protection profile
SAE	Society of American Engineers
SLA	Service level agreement
TCSEC	Trusted Computer System Evaluation Criteria
TOE	Target of evaluation
VANET	Vehicular ad-hoc network

References

1. Anisetti, M.; Ardagna, C.; Damiani, E.; Polegri, G. Test-based security certification of composite services. *ACM Trans. Web (TWEB)* **2018**, *13*, 1–43.
2. Chen, W.; Xu, Z.; Shi, S.; Zhao, Y.; Zhao, J. A survey of blockchain applications in different domains. In Proceedings of the 2018 International Conference on Blockchain Technology and Application, Xi'an, China, 10–12 December 2018; pp. 17–21.
3. Choi, M.K.; Yeun, C.Y.; Seong, P.H. A Novel Monitoring System for the Data Integrity of Reactor Protection System Using Blockchain Technology. *IEEE Access* **2020**, *8*, 118732–118740. [CrossRef]
4. Nakamoto, S.; Bitcoin, A. A Peer-to-Peer Electronic Cash System. 2008; Volume 4. Available online: https://www.debr.io/article/21260.pdf (accessed on 27 September 2021).
5. Bellini, E.; Iraqi, Y.; Damiani, E. Blockchain-based distributed trust and reputation management systems: A survey. *IEEE Access* **2020**, *8*, 21127–21151. [CrossRef]
6. Beniiche, A. A study of blockchain oracles. *arXiv* **2020**, arXiv:2004.07140.
7. Al-Breiki, H.; Rehman, M.H.U.; Salah, K.; Svetinovic, D. Trustworthy blockchain oracles: Review, comparison, and open research challenges. *IEEE Access* **2020**, *8*, 85675–85685. [CrossRef]
8. Ellis, S.; Juels, A.; Nazarov, S. Chainlink a decentralized oracle network. *Retrieved March* **2017**, *11*, 2018.
9. Benet, J. Ipfs-content addressed, versioned, p2p file system. *arXiv* **2014**, arXiv:1407.3561.
10. Chen, Y.; Li, H.; Li, K.; Zhang, J. An improved P2P file system scheme based on IPFS and Blockchain. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 2652–2657.
11. Bulinska-Stangrecka, H.; Bagienska, A. Investigating the links of interpersonal trust in telecommunications companies. *Sustainability* **2018**, *10*, 2555. [CrossRef]
12. Shehada, D.; Yeun, C.Y.; Zemerly, M.J.; Al-Qutayri, M.; Al-Hammadi, Y.; Hu, J. A new adaptive trust and reputation model for mobile agent systems. *J. Netw. Comput. Appl.* **2018**, *124*, 33–43. [CrossRef]
13. Damiani, E.; Ardagna, C.A.; El Ioini, N. *Open Source Systems Security Certification*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2008.
14. Kruger, R.; Eloff, J.H. A common criteria framework for the evaluation of information technology systems security. In *Information Security in Research and Business*; Springer: Berlin/Heidelberg, Germany, 1997; pp. 197–209.
15. Qiu, L.; Zhang, Y.; Wang, F.; Kyung, M.; Mahajan, H.R. *Trusted Computer System Evaluation Criteria*; National Computer Security Center, Citeseer: Dublin, Ireland, 1985.
16. Schmittner, C.; Ma, Z.; Reyes, C.; Dillinger, O.; Puschner, P. Using SAE J3061 for automotive security requirement engineering. In Proceedings of the International Conference on Computer Safety, Reliability, and Security, Trondheim, Norway, 20–23 September 2016; Springer: Cham, Switzerland, 2016; pp. 157–170.
17. *Road Vehicles—Functional Safety*; Standard, International Organization for Standardization: Geneva, Switzerland, 2011.
18. *Road Vehicles—Cybersecurity Engineering*; Standard, International Organization for Standardization: Geneva, Switzerland, 2021.
19. Larson, P. Testing Linux with the Linux Test Project. Ottawa Linux Symposium. 2002; p. 265. Available online: https://courses.cs.vt.edu/cs5204/fall05-gback/papers/ols2002_proceedings.pdf#page=265 (accessed on 8 March 2021).
20. Nchinda, N.; Cameron, A.; Retzepi, K.; Lippman, A. MedRec: A network for personal information distribution. In Proceedings of the 2019 International Conference on Computing, Networking and Communications (ICNC), Honolulu, HI, USA, 18–21 February 2019; pp. 637–641.
21. Keller, M. Design and Implementation of a Blockchain-Based Trusted VNF Package Repository. Ph.D. Thesis, University of Zürich, Zürich, Switzerland, 2019.
22. Scheid, E.J.; Keller, M.; Franco, M.F.; Stiller, B. BUNKER: A Blockchain-based trUsted VNF pacKagE Repository. In Proceedings of the International Conference on the Economics of Grids, Clouds, Systems, and Services, Leeds, UK, 17–19 September 2019; Springer: Cham, Switzerland, 2019; pp. 188–196.
23. Caldarelli, G.; Ellul, J. Trusted academic transcripts on the blockchain: A systematic literature review. *Appl. Sci.* **2021**, *11*, 1842. [CrossRef]
24. Rasool, S.; Saleem, A.; Iqbal, M.; Dagiuklas, T.; Mumtaz, S.; Qayyum, Z.U. Docschain: Blockchain-Based IoT Solution for Verification of Degree Documents. *IEEE Trans. Comput. Soc. Syst.* **2020**, *7*, 827–837. [CrossRef]
25. Zaabi, A.O.A.; Yeun, C.Y.; Ernesto Damiani, G. An Enhanced Conceptual Security Model for Autonomous Vehicles. *Adv. Sci. Technol. Eng. Syst. J.* **2020**, *5*, 853–864. [CrossRef]
26. Wyglinski, A.M.; Huang, X.; Padir, T.; Lai, L.; Eisenbarth, T.R.; Venkatasubramanian, K. Security of autonomous systems employing embedded computing and sensors. *IEEE Micro* **2013**, *33*, 80–86. [CrossRef]
27. Szydlowski, C.P. *Can Specification 2.0: Protocol and Implementations*; Technical Report, SAE Technical Paper; SAE: Warrendale, PA, USA, 1992.
28. Talbot, S.C.; Ren, S. Comparision of fieldbus systems can, ttcan, flexray and lin in passenger vehicles. In Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems Workshops, Montreal, QC, Canada, 22–26 June 2009; pp. 26–31.
29. Wolf, M.; Weimerskirch, A.; Paar, C. Secure in-vehicle communication. In *Embedded Security in Cars*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 95–109.

30. Buttyán, L.; Holczer, T.; Vajda, I. On the effectiveness of changing pseudonyms to provide location privacy in VANETs. In *European Workshop on Security in Ad-Hoc and Sensor Networks*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 129–141.
31. Raya, M.; Hubaux, J.P. The security of vehicular ad hoc networks. In Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks, Alexandria, VA, USA, 7 November 2005; pp. 11–21.
32. Papadimitratos, P.; Buttyan, L.; Holczer, T.; Schoch, E.; Freudiger, J.; Raya, M.; Ma, Z.; Kargl, F.; Kung, A.; Hubaux, J.P. Secure vehicular communication systems: Design and architecture. *IEEE Commun. Mag.* **2008**, *46*, 100–109. [CrossRef]
33. Bariah, L.; Shehada, D.; Salahat, E.; Yeun, C.Y. Recent advances in VANET security: A survey. In Proceedings of the 2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall), Boston, MA, USA, 6–9 September 2015; pp. 1–7.
34. Seo, H.; Lee, K.D.; Yasukawa, S.; Peng, Y.; Sartori, P. LTE evolution for vehicle-to-everything services. *IEEE Commun. Mag.* **2016**, *54*, 22–28. [CrossRef]
35. Sharma, S.; Kaushik, B. A survey on internet of vehicles: Applications, security issues & solutions. *Veh. Commun.* **2019**, *20*, 100182.
36. Vijayarangam, S.; Chandra Babu, G.; Ananda Murugan, S.; Kalpana, N.; Malarvizhi Kumar, P. Enhancing the security and performance of nodes in Internet of Vehicles. *Concurr. Comput. Pract. Exp.* **2021**, *33*, 1. [CrossRef]
37. Almehrezi, F.R.; Yeun, C.Y.; Yoo, P.D.; Damiani, E.; Al Hammadi, Y.; Yeun, H. An Emerging Security Framework for Connected Autonomous Vehicles. In Proceedings of the 2020 7th International Conference on Behavioural and Social Computing (BESC), Bournemouth, UK, 5–7 November 2020; pp. 1–4.
38. Forecast, C.C. Global connected car market to grow threefold within five years. In *GSMA Connected Living Programme: MAutomotive*; GSMA: London, UK, 2013.
39. Lu, N.; Cheng, N.; Zhang, N.; Shen, X.; Mark, J.W. Connected vehicles: Solutions and challenges. *IEEE Internet Things J.* **2014**, *1*, 289–299. [CrossRef]
40. Liu, N. Internet of Vehicles: Your next connection. *Huawei WinWin* **2011**, *11*, 23–28.
41. Kim, K.; Kim, J.S.; Jeong, S.; Park, J.H.; Kim, H.K. Cybersecurity for autonomous vehicles: Review of attacks and defense. *Comput. Secur.* **2021**, *103*, 102150. [CrossRef]