UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

A Blockchain-based Solution for Connected Vehicle Networks

Paulo Alexandre dos Santos Álvares

Mestrado em Engenharia Informática
Especialização em Arquiteturas, Sistemas e Redes de Computadores

Dissertação orientada por:
Naércio David Pedro Magaia

2021

Agradecimentos

A realização da dissertação foi uma tarefa árdua e trabalhosa, especialmente por ocorrer durante uma pandemia global. Este trabalho marca, também, o final de um ciclo de 5 anos de estudos nesta segunda casa que é a FCUL, onde cresci imenso como aluno e indivíduo.

Ao meu orientador, o Professor Doutor Naércio Magaia, agradeço pela paciência, ajuda, orientação e motivação que me deu durante a produção deste trabalho e pelas oportunidades e portas que me ajudou a abrir durante este período. Por ser incansável e ajudar até mesmo quando era menos oportuno, agradeço-lhe sinceramente. Este trabalho enquadra-se no âmbito do projecto SEEDS (H2020-MSCA-RISE sob o número 101006411).

À minha família, agradeço pelo apoio e carinho durante este processo, por ajudarem da melhor forma que podiam.

Aos meus amigos Rita Vieira, Rita Silva, Cláudia Lopes, Pedro Cecília, Rita Rebelo, Mariana Caldeira, Joana Santos, André Santana, Afonso Falcão e Antonio Carmona, agradeço por todas as brincadeiras e distrações, pela paciência e pelas palavras de coragem quando mais precisava. Ao Rodrigo Albino, por estar lá sempre, nos piores e melhores momentos, a relembrar-me que depois da tempestade vem a bonança.

A realização desta dissertação não seria possível sem a ajuda, apoio e carinho destas pessoas.

“Faithless is he that says farewell when the road darkens.”

J. R. R. Tolkien in The Lord of the Rings: The Fellowship of the Ring

Resumo

Previa-se que, em 2020, existiriam cerca de 26 mil milhões de dispositivos ligados à internet, com veículos a ocupar uma grande percentagem deste número. A Internet de Veículos (IdV) é um conceito que se refere à conexão e cooperação de veículos e dispositivos inteligentes numa rede através da produção, partilha e processamento de dados e que tem como objetivo a melhoria das condições de trânsito, do tempo médio de viagem e do conforto dos utilizadores, ajudando à redução dos níveis de poluição e acidentes. No entanto, a partilha de dados privados, como a localização, tem de ser completamente protegida de modo a salvaguardar os veículos e os seus condutores, visto que esta informação pode ser utilizada por agentes maliciosos, o que pode comprometer a segurança dos veículos e dos seus possíveis passageiros, especialmente com o desenvolvimento de carros com condução autónoma.

Blockchain é uma tecnologia relativamente recente (i.e., 2008) que garante confiança entre dispositivos numa rede através do uso de mecanismos de criptografia e protocolos de consenso em ambientes distribuídos e não confiáveis, como as redes IdV. Assim sendo, tem-se feito muita pesquisa sobre como implementar e integrar aplicações e soluções baseadas em *Blockchain* em redes IdV, visto que a primeira oferece soluções para problemas críticos que existem na segunda, como a falta de segurança e privacidade. Porém, estas implementações têm de ter em conta os recursos limitados dos dispositivos na IdV, já que não são suficientes a nível de poder computacional e energia para suportar sistemas tradicionais de *Blockchain*, que utilizam protocolos de consenso baseados em *Proof-of-Work*. Protocolos de consenso deste tipo requerem a resolução de um problema matemático com cálculos extremamente complexos, que consomem bastantes recursos, a um nível energético, destes dispositivos e exigem a existência de capacidades computacionais elevadas, ou os tempos de execução dos algoritmos será tão grande que a eficiência dos mesmos não é suficiente para estes ambientes com trocas constantes de informação. Esta questão é uma das maiores limitações para a implementação de soluções baseadas em plataformas de *Blockchain* na IdV.

Esta Dissertação tem como foco, então, introduzir e explicar uma proposta de solução para o problema referido, através da implementação de duas alternativas leves baseadas em primitivas criptográficas para atingir consenso e garantir confiança entre os nós que participam na rede, com resultados semelhantes a mecanismos tradicionais de *Blockchain* em termos de segurança e com menos consumo de recursos computacionais e energia. Estas soluções também são denominadas como *Communication-Enforcing Schemes* (CES) porque requerem que o nó que “assina a mensagem” (i.e., faz *hash* do bloco) tenha de receber toda a mensagem de quem envia, ou o resultado será incorreto. Esta propriedade impede a existência de um problema proeminente em redes *peer-to-peer* (P2P), como pode ser o caso das IdV: o *free riding*, que consiste em nós que utilizam a rede para seu benefício sem participar ativamente na mesma, o que viria a beneficiar os outros *peers*.

O primeiro esquema apresentado, *CBC-RSA*, divide a mensagem recebida em blocos, faz o hash de cada divisão e une a informação toda novamente. No entanto, para evitar que o tamanho do resultado final seja maior que a mensagem original, aplica-se um protocolo de *Cipher-block*

chaining (CBC) para manter o tamanho do resultado. Para o processo de verificação, o algoritmo realiza o processo no sentido inverso, desencriptando sequencial a mensagem de trás para a frente, com as chaves públicas do nó que criou a assinatura, para poder comparar os valores. Se o bloco tiver sido alterado, depois de ser introduzido na *Blockchain*, os valores desencriptados serão diferentes do que os que serão obtidos da mensagem atual.

O segundo esquema, chamado *Two-root RSA*, divide a mensagem recebida em blocos, também, e cria um vetor inicial de bytes, preenchido de forma aleatória. Com essa informação, constrói uma equação polinomial de grau igual ao número de divisões da mensagem original. De seguida, o algoritmo tenta encontrar duas raízes desse polinómio e, se não encontrar, cria um vetor inicial novo e repete o processo até encontrar duas raízes válidas. Para fazer a verificação de dados, o algoritmo utiliza o vetor inicial e a mensagem original, cria a equação polonimal e utiliza as duas raízes encontradas para verificar se as mesmas ainda o são (raízes) para a equação criada com mensagem guardada no respetivo bloco. Se o bloco tiver sido alterado de forma maliciosa, o valor resultante será diferente de 0 para as raízes fornecidas, o que será detetado.

Estes dois algoritmos foram implementados recorrendo à linguagem de programação *Golang*, que é a linguagem base das plataformas de *Blockchain* mais conhecidas, como o *Hyperledger* e o *Ethereum*. Foram, posteriormente, introduzidos na plataforma *Ethereum*¸ pois era a que tinha um protocolo baseado em *PoW* incorporado de início, ao contrário do *Hyperledger*, que utiliza uma versão do protocolo *RAFT*, pelo que seria necessário criar um terceiro protocolo *PoW* para fazer comparações em termos de consumo e tempos de computação.

Para obter os dados, foram realizados testes em dispositivos diferentes, um com menos recursos computacionais (Raspberry Pi com 4GB RAM) e outro, um computador regular (8GB RAM). Estes testes serviram para comparar as diferenças entre os tempos de computação das criações de *hashes* de um computador regular e de um dispositivo semelhante aos que existem em redes IdV. Para simular um caso real, onde existe interação entre veículos, os dados de troca de mensagens entre veículos foram obtidos através de um simulador de redes veiculares (Veins). Esse registo de troca de mensagens foi, posteriormente, transformado em transações entre os nós da rede *Ethereum*, que foram aplicadas durante a execução de vários nós, a funcionar na mesma rede, para simular uma situação real de troca de mensagens entre os veículos e introdução de informação na *Blockchain*, e assim obter-se resultados o mais semelhantes à realidade.

A avaliação de desempenho mostra que os mecanismos de consenso apresentados são, aproximadamente, até 12000 vezes mais rápidos a fazer o *hashing* dos blocos e até 40000 vezes mais rápido a verificar essa *hash*, que o protocolo utilizado atualmente na plataforma *Ethereum*, o *Ethash*. Para encriptar a informação, o *CBC-RSA* é mais rápido que o *Two-root RSA*, devido à complexidade dos cálculos matemáticos do segundo, mas, na verificação de integridade, os lugares invertem-se, sendo o *Two-root RSA* geralmente mais rápido, visto que o processo deste é bastante mais simples que a desencriptação necessária para a verificação de uma assinatura com *CBC-RSA*.

Com os resultados obtidos, é possível concluir que os algoritmos apresentados se mostram, então, como soluções viáveis para o problema de recursos reduzidos presente nos dispositivos da IdV, utilizando menos recursos computacionais e requerendo menos tempo de execução que um método tradicional de consenso baseado em *PoW*, como é o caso do *Ethash*, garantido segurança, assegurada matematicamente pelas provas apresentadas no desenho dos algoritmos. O facto de

serem desenvolvidos em *Golang* adiciona uma camada de flexibilidade de implementação e integração noutros sistemas, visto que é uma linguagem recente e utilizada nas maiores plataformas de *Blockchain* utilizadas para pesquisa científica e desenvolvimento de aplicações, como é o caso do *Hyperledger*.

Futuramente, a generalização dos algoritmos para mais divisões da mensagem inicial é uma das propostas de desenvolvimento, de forma a avaliar, em termos de desempenho e tempo de encriptação e criação de assinaturas, de que modo o aumento do número de divisões poderá afetar a segurança e a eficiência dos protocolos, seja positiva, seja negativamente. Alterações ao modo de mineração de blocos do *Ethereum*, para algo periódico ou baseado em eventos poderia ser, também, uma mais-valia para a redução no consumo de recursos em dispositivos limitados a este nível.

Palavras-chave: Internet de Veículos; *Blockchain*; Partilha de informação; Criptografia; Consenso distribuído

Abstract

It had been predicted that by 2020, nearly 26 billion devices would be connected to the internet, with a big percentage being vehicles. The Internet of Vehicles is a concept that refers to the connection and cooperation of smart vehicles and devices in a network, through the generation, transmission, and processing of data that aims at improving traffic congestion, travel time, and comfort, all the while reducing pollution and accidents. However, this transmission of sensitive data (e.g., location) needs to occur with defined security properties to safeguard vehicles and their drivers, since this data could be used by attackers.

Blockchain is a fairly recent technology that guarantees trust between nodes through the usage of cryptography mechanisms and consensus protocols in distributed, untrustful environments, like IoV networks. As such a lot of research has been made in implementing the former in the latter to impressive results, as Blockchain can cover and offer solutions to a lot of problems of IoV networks. However, these implementations have to deal with the challenge of resource constraints in IoV, since they do not suffice for the computational and energy requirements of traditional Blockchain systems, which is one of the biggest limitations of Blockchain implementations in IoV.

This dissertation aims at introducing and explaining a proposed solution to this problem by implementing lightweight alternatives based on cryptographic primitives to achieve consensus on the global state and ensure trust between the nodes that participate in the IoV network, with similar results to regular Blockchain mechanisms in the terms of security properties and less resource consumption in terms of computational power and energy usage.

Performance evaluation results show that the proposed consensus mechanisms are approximately 12000 times faster when hashing a block and up to 40000 times faster when verifying said hash, than the Proof-of-Work-based protocol that is presently used in Ethereum, Ethash.

Keywords: Internet of Vehicles; Blockchain; Cryptography; Distributed Consensus; Lightweight Protocol

Table of contents

List of Figures

List of Tables

Summary of Acronyms

Acronym	Definition
IoV	*Internet of Vehicles*
IoT	*Internet of Things*
ITS	*Intelligent Transportation System*
VANET	*Vehicular Ad-hoc Networks*
V2X	*Vehicle-to-everything*
V2V	*Vehicle-to-vehicle*
V2I	*Vehicle-to-infrastructure*
V2P	*Vehicle-to-pedestrian*
V2S	*Vehicle-to-sensor*
V2R	*Vehicle-to-roadside unit*
DSRC	*Dedicated short-range communications*
QoS	*Quality of Service*
PoW	*Proof of work*
PoS	*Proof-of-stake*
BFT	*Byzantine Fault Tolerance*
PBFT	*Practical Byzantine Fault Tolerance*
PoAc	*Proof-of-activity*
PoB	*Proof-of-burn*
PoET	*Proof-of-elapsed time*
PoC	*Proof-of-capacity*
PoA	*Proof-of-authority*
PoI	*Proof-of-importance*
PoL	*Proof-of-Luck*
PoX	*Proof-of-eXercise*
SDN	*Software-Defined Network*
IBFT	*Istanbul Byzantine Fault Tolerance*
YAC	*Yet Another Consensus*
dapps	*Decentralized applications*
VM	*Virtual Machine*
CES	*Communication Enforcement Schemes*
CBC	*Cipher Block Chaining*
RSA	*Rivest-Shamir-Adleman*

Capítulo 1 Introduction

1.1 Motivation

The Internet of Vehicles (IoV) is a decentralized technology that expands from the preexisting Vehicular Ad-hoc Networks (VANETs) [1] while aiming at large-scale city-level coverage. IoV objectively offers the means of communication between nodes (vehicles, infrastructure, sensors, etc.) in a network by unifying various technologies.

IoV aims to provide services and solutions that improve comfort, fuel consumption, and traffic congestion seamlessly, or that allow for media streaming, file sharing, etc., all the while ensuring the satisfaction, security, and privacy of vehicles and users in the network and real-life [2].

IoV faces challenges on many fronts [3], such as dealing with resource constraints of devices in the network (e.g., low battery, reduced computational power), which limits algorithm and application development, the lack of means to deal with constant node mobility in a seamless fashion, which can increase latency and decrease performance due to frequent handovers, or even in big data management, since is constantly generated by devices in IoV networks, it can cause network congestion and reduce performance.

However, the most important challenge to face is security and privacy, due to the sensitive nature of the shared data (i.e., location, state of the car, personal data, pictures). This is because vehicles work in unprotected, heterogeneous, and vulnerable environments that cannot ensure trust between nodes by themselves and open up the possibility for cyber-attacks and exploitations [4], which are imperative to deal with as one mistake, or malicious attack, could lead to accidents and, in worst-case scenarios, endanger human lives. The trust problem could be solved by introducing a third-party authority that validates every transaction between devices, but not only does this introduce a single point of failure, but it also decreases the throughput of operations, which is not beneficial in IoV scenarios.

The Blockchain, firstly presented as the underlying technology of Bitcoin [5], is a distributed ledger (or database) of transactions between nodes in a network that ensures various security and privacy properties, such as data authentication, data non-repudiation, privacy, traceability, and immutability, through the implementation of cryptography, digital signatures and consensus protocols. Through these characteristics, and later the implementation of Smart Contracts, Blockchain can solve the trust problem of decentralized networks without a third-party authority, removing the single point of failure problem. Blockchain also provides the benefit of interoperability since it is deployed on top of an overlay Peer-to-Peer (P2P) network. The sum of all these assets makes Blockchain an ideal technology to solve the security and privacy problems of IoV.

1.2 Problem

Even with the great benefits Blockchain presents, there is a problem with weight of the underlying technologies and their effects on IoV devices and their restrictions. Typical Blockchain implementations use consensus protocols that propose a hard-to-solve problem that requires a lot of computational power and energy consumption and that proves to be detrimental to the resource-constrained devices (i.e., sensors) that proliferate in IoV networks. With these high power and energy requirements for traditional Blockchain implementations, the referred devices can barely compute the algorithms or end up computing them in greater time periods, which is not desirable, for example, when reporting accidents and other emergency events, or even during regular communication times, since the blockchain will not receive and take note of these communications in a useful period of time. These events require fast communication times and a fast spread of information. If a Blockchain system was to be introduced in IoV, then the consensus algorithms used on the creation of blocks need to be as fast as possible, while retaining security properties, as to cause as little an impact as possible on the communication processes.

1.3 Objectives

Given the resource constraints of IoV elements, i.e., vehicles, Roadside Units, among others, the goal of this dissertation is to implement and evaluate lightweight cryptographic alternatives to the Blockchain's hard-to-solve consensus mechanism, making it require fewer resources while achieving similar results in efficacy. These implementations will then be integrated into a vehicular network scenario through the usage of a Blockchain platform that has pluggable consensus and allows for the integration of own protocols (e.g., Hyperledger, Ethereum) and a network simulator compatible with the latter (e.g., NS-3, Veins).

1.4 Contributions

As a result of this work, two research papers were elaborated/published.

The first article is entitled "Blockchain-Based Solutions for UAV-assisted Connected Vehicle Networks in Smart Cities: A Review, Open Issues, and Future Perspectives" [6]. It reviewed the Internet of Vehicles and Blockchain state-of-the-art, with some references on unmanned aerial vehicles and smart cities. It was published on the 12th of March of 2021, in Telecom. Telecom is an open-access, peer-reviewed journal of communications and networks that is published by the Multidisciplinary Publishing Institute (MDPI). As of the time of writing, the article has reached 1896 Full-text views and 1150 Abstract views.

Another article, entitled "Lightweight Blockchain Consensus Mechanisms for the Internet of Vehicles" has been submitted for publication in a top-tier international journal.

1.5 Document structure

The document is organized as follows:

- Chapter 2 describes the background and related work on the area, presenting the basic ideas, definitions, applications, and challenges of the Internet of Vehicles and Blockchain, and how they have been used together.
- Chapter 3 describes the design and implementation of the proposed approaches in the Ethereum Blockchain platform.
- Chapter 4 depicts the evaluation process, results, and conclusions, with remarks on limitations encountered along the way and impediments found with Ethereum and Golang.
- Chapter 5 presents the conclusion of this report and, by association, of this work.

Capítulo 2 Background and Related Work

In order to better understand the scale and importance of the resource restraint problem on Blockchain implementations and development, and how the Internet of Vehicles and Blockchain meet, this part of the report will present a review on the *state of the art* of the Internet of Vehicles and Blockchain as of current research, introducing the concepts, architecture, benefits, applications, and challenges of both technologies. Finally, some applications of Blockchain for the Internet of Vehicles are described, and multiple tools utilized to simulate and evaluate these applications are listed and compared.

2.1 Internet of Vehicles

It had been predicted by R. James *et al* [7] that, by 2020, there would be, at minimum, 26 billion devices connected in some form to the Internet. D. Gary [8] predicted that this number would reach double that, up to 50 billion. Out of this number, vehicles are expected to occupy a large percentage, making it extremely important to invest in research on open problems and vulnerabilities that this number of connected devices brings out to the open. The main purpose of IoV is to enhance the efficacy, efficiency, and comfort of transportation and the facility level of cities, reduce costs, and ensure customer satisfaction [2].

2.1.1 Definition

With the growth of entities in VANETs over the years, new requirements appear that have to be fulfilled in order to appeal to the user's needs. In a vehicular network, vehicles frequently produce enormous amounts of data (from internal mechanical data to data related to the road or traffic state). The processing, analyzing, and evaluation of these large amounts of data is an arduous task that VANETs cannot handle due to the limited processing power of their devices. The limited applications of entrusted Internet services [9] and the connection/disconnection of vehicles due to getting in and out of the coverage area [10] are also noticeable constraints in VANETs, which drive the evolution towards the IoV. As such, vehicles in VANETs need to become "smart" objects that work cooperatively to ease tasks' weight. This smart cooperation marks the line where VANETs start evolving into the IoV, as defined by the work of A. Islam *et al* [11].

Another way IoV demarks itself from VANETs is in the sense that it aims at large coverage areas (city-scale or even global) and by aiming the integration of two technological visions: vehicle's networking and vehicle's intelligence, as proposed by F. Yang *et al* [12], with a focus on integrating objects (humans, vehicles, units) and environments as to build an intelligent network. The coalition of smart vehicular systems and cyber-physical systems brings the possibility of developing a global network that offers services and gives quality-of-life improvements to drivers and service providers, helping to reduce traffic congestion, pollution, and accidents.

Moreover, IoV can be seen as an application of an IoT technology in an *ITS* technology, which is also the result of merging three different networks: the intervehicular network (from the vehicle to other vehicles), the intravehicular network (inside the vehicle – cyber-physical system) and the vehicular mobile Internet (from the vehicle to other objects on the network) [13]. It is an enormous distributed system for wireless communication and data exchange on a *vehicle-to-everything* (V2X) mode with defined protocols and data interaction standards, like IEEE 802.11p. These modes are represented in Figure 2.1.

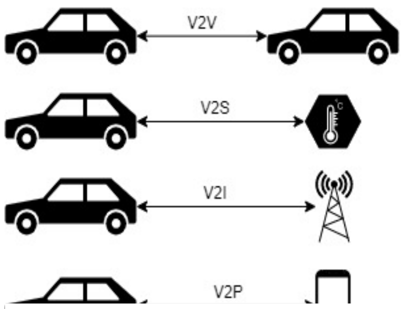


Figure 2.1 - V2X communication modes. In order, they are:
- *vehicle-to-vehicle (V2V),*
- *vehicle-to-sensor (V2S),*
- *vehicle-to-infrastructure (V2I),*
- *vehicle-to-pedestrian (V2P),*
- *vehicle-to-road-side-unit (V2R).*

2.1.2 Architecture

In terms of architecture, IoV can be interpreted in various ways. For example, researchers have divided the architecture into three [14], four [15], or even five layers [9]. Figure 2.2 presents a side-by-side comparison of the four aforementioned architectures.

Three-layer architecture

The three-layer architecture, presented by L. Nanjie [14], is divided based on the interactions with the technologies in the IoV environment, as follows:

- The *sensor layer* is responsible for the sensors in the vehicles and surrounding infrastructure.
- The *communication layer* is responsible for the wireless connections between entities in various vehicle-to-everything (V2X) modes (Figure 2.1).
- The *data processing layer* is responsible for holding statistics tools and storage (acts as the intelligence of the IoV network and provides big data processing to vehicles, providing decision-making in risk situations).

Four-layer architecture

The four-layer IoV system architecture, presented by CISCO [15], is divided as follows:

- The *end-point layer*, which is responsible for the vehicles (and sensors), V2V communications, and software.
- The *infrastructure layer*, which handles all communication technologies used by entities.
- The *operation layer*, responsible for policy enforcement and flow-based management of network providers.
- The *service layer* handles the services offered by the cloud infrastructure that are connected to the network.

Five-layer architecture

The five-layer architecture, proposed by O. Kaiwartya *et al* [9], is divided as follows:

- The *perception layer* handles data gathering, data digitization, and transmission, and energy optimization at lower layers.

- The *coordination layer* is not only responsible for transferring data to the artificial intelligence layer in a secure manner, but also being responsible for dealing with the heterogeneity of the network structure, unifying received information.
- The *artificial intelligence layer* holds the cloud infrastructure, which stores, processes, and analyzes the data from layers below, utilizing this analysis for decision making, while also managing services provided by cloud systems.
- The *application layer* is responsible for providing intelligent services to end-users, based on the processed and analyzed information of the AI layer, that serves for service discovery from smart applications.
- The *business layer*, a novelty from the last two architectures, which is responsible for giving foresight on the development of business models, based on data analysis and statistics, which come from the *application layer* and is later transformed by analysis tools, while utilizing decision making for budgeting and optimization usage of resources.

Seven-layer architecture

The work of J. Contreras-Castillo *et al* [10] refers that the three and four-layered proposed models have weaknesses and do not contemplate important concerns of IoV systems, like the need for the existence of layers for security and data dissemination/transmission, the communication between the driver and the vehicle (interface) and network congestion. The authors then present a more complete layered architecture interpretation which divides the IoV into seven layers:

- The *processing layer*, as the name suggests, is responsible for big data processing using various cloud computing technologies, which helps the development of strategies for business models.
- The *control and management layer* manages the network service providers in IoV, utilizing policies and functions for that objective.
- The *communication layer* is responsible for selecting the best network to serve the needs of the user.
- The *data filter and pre-processing layer* analyzes data to avoid network congestion by the transmission of irrelevant information.
- The *data acquisition layer* collects data from their respective sources (sensors, infrastructure connection points, other vehicles, etc.).
- The *user interface layer* is responsible to deal with how the information is passed from the vehicle and sensors to the driver and users inside the car.
- Finally, the *security layer*, transversal to the six ones noted before, and is one of the major differences between this architecture and the first two presented before, being responsible for all security properties guaranteed, using proposed solutions to mitigate the damage from cyberattacks and malfunctions.

Comparison

It is possible to conclude that the four architectures share some similarities themselves. All the architectures have layers that are responsible for sensors, vehicles, and data collection (*sensor*, *end-point*, *perception*, and *data acquisition layers*). Layers that handle and manage communications between entities are also present in all the architectures (*communication*, *infrastructure, perception* and *coordination*, and *communication layers*). The *data processing, service, artificial intelligence* and *application*, and the *processing layers* are responsible for data processing and decision making, among other services, usually provided by the cloud infrastructure. The management of network providers is only present in the four, five, and seven-layered architectures in the *operation*, *coordination*, and *control and management layers*, respectively. Only the five and seven-layered architectures have layers that are responsible for or help in the development of business models: the *business* and *processing layers*, respectively. Finally, only the seven-layered architecture has layers that handle data pre-processing and security. While the five-layer architecture does not present a layer for security, they mention the secure transmission of information between layers and the authors mention the usage of protocols to ensure the secure exchange of data.

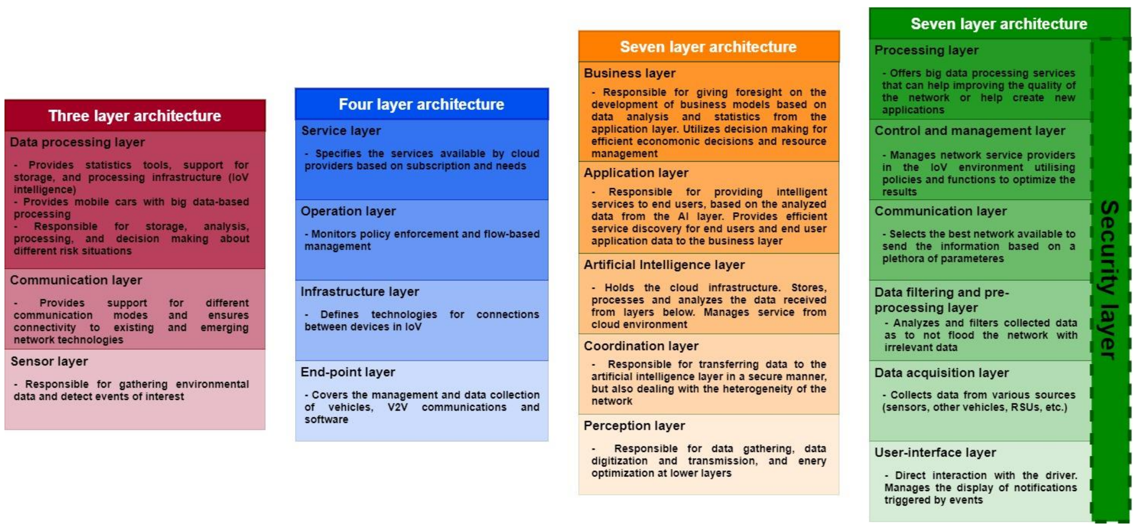

Three layer architecture
Data processing layer
- Provides statistics tools, support for storage, and processing infrastructure (IoV intelligence)
- Provides mobile cars with big data-based processing
- Responsible for storage, analysis, processing, and decision making about different risk situations
Communication layer
- Provides support for different communication modes and ensures connectivity to existing and emerging network technologies
Sensor layer
- Responsible for gathering environmental data and detect events of interest
Four layer architecture
Service layer
- Specifies the services available by cloud providers based on subscription and needs
Operation layer
- Monitors policy enforcement and flow-based management
Infrastructure layer
- Defines technologies for connections between devices in IoV
End-point layer
- Covers the management and data collection of vehicles, V2V communications and software
Seven layer architecture
Business layer
- Responsible for giving foresight on the development of business models based on data analysis and statistics from the application layer. Utilizes decision making for efficient economonic decisions and resource management
Application layer
- Responsible for providing intelligent services to end users, based on the analyzed data from the AI layer. Provides efficient service discovery for end users and end user application data to the business layer
Artificial Intelligence layer
- Holds the cloud infrastructure. Stores, processes and analyzes the data received from layers below. Manages service from cloud environment
Coordination layer
- Responsible for transferring data to the artificial intelligence layer in a secure manner, but also dealing with the heterogeneity of the network
Perception layer
- Responsible for data gathering, data digitization and transmission, and enery optimization at lower layers
Seven layer architecture
Processing layer
- Offers big data processing services that can help improving the quality of the network or help create new applications
Control and management layer
- Manages network service providers in the IoV environment utilising policies and functions to optimize the results
Communication layer
- Selects the best network available to send the information based on a plethora of parameteres
Data filtering and pre-processing layer
- Analyzes and filters collected data as to not flood the network with irrelevant data
Data acquisition layer
- Collects data from various sources (sensors, other vehicles, RSUs, etc.)
User-interface layer
- Direct interaction with the driver. Manages the display of notifications triggered by events
Security layer

Figure 2.2 - Comparison between the various layered architectures for IoV.
With each new layer added, each layer's job gets more specific or even more, objectives are purposes are added, in order to fulfill the needs of the network, applications, and vehicles.

E. Benalia et al [3] state that the three and five-layer architectures do not take into account the security challenges in IoV. It is possible to also add the four-layer architecture to this affirmation, giving our past analysis above. They further notice that even the most complete seven-layered architecture does not take advantage of new paradigms and technologies, like the usage of 4G/5G communications to deal with high latencies and low bandwidth, or the usage of paradigms like edge and fog computing for pre-processing and management.

The authors then present a three-layered generic architecture which is divided as follows:

- The *terminal layer* (or *IoV layer)* that is responsible for gathering information on the road through *mm-wave V2X* communication modes (*mm-wave*, or millimeter-wave, is a new 5G network technique that can promise multigigabit communication services [16]).
- The *edge computing layer*, that has:
    - The *fog infrastructure sublayer,* that is responsible for data processing, analysis, computing storing, networking, and security.
    - The *fog virtualization sublayer*, which is divided in:
        - *upper level*, which has two planes that manage the network (*control plane* that manages the local and global data planes, and provides programmability and flexible management, and *global data plane* that contains forwarding and data processing devices).

        - *lower level* that contains the local data plane which has nodes that forward and receive data to and from fog and cloud computing.
    - The *fog service sublayer* that offers traditional fog services adapted to IoV, like:
        - *Fog Vehicular Infrastructure as a Service* (offers data processing, storage, and analysis while having the capability of adopting another infrastructure to serve other needs).
        - *Fog Vehicular Platform as a Service* (offers different operational systems and computational environments to ensure the fulfillment of the heterogeneous needs of drivers and vehicles).
        - *Fog Vehicular Software as a Service* (offers fog-based software which is divided into user applications and safety applications, which will be explained below).
- The *cloud computing layer*, that provides services just like the latter four mentioned architectures: big data processing, analysis, storage, and analytics tools, which can then help develop business models.

This final layered architecture was made with the intent to allow more flexible dissemination of data in IoV while taking into account the advantages of new rising technologies, such as cloud and fog computing paradigms that increase network performance. While this architecture is the most complete, not every project needs the same notion of architecture to fulfill its needs, and it all depends on what are the required aspects that need to be considered to develop the idea and what each architecture layout has to offer. If the project does not or cannot envelop fog computing techniques, then the best architecture to follow is one of the other four. If the architecture is not a big factor, then the first three-layer architecture is the best one to use as the basis for the project in question, aiming for simplicity.

2.1.3 Applications

Applications for the IoV vary in functionality and objectives. Researchers have divided these applications according to the services they aim to provide and in what shape or form they provide them.

Taxonomy

Following the work of F. Yang *et al* [12] and W. Wu *et al* [17], IoV applications can be divided into categories. Figure 2.3 presents the following taxonomy of IoV applications.

Firstly, the *User* or *Infotainment applications*, which are applications that provide value-added services and have multiple requirements in terms of real-timeliness or communications. These applications range from video/music streaming, file transfer to weather information and local point of interest information. The services these applications offer can also be further divided in:

- *cooperative local services* – relates to infotainment from local-based services (i.e. point of interest notification and media downloading).
- *global internet services* – relates to data obtained from services like insurance management and parking zone information, which are constantly updated.

Some *User applications* that have been developed are, for example, the Cooperative Video Streaming over Vehicular Networks [18] that provides basic QoS over 3/3.5G networks. In this application, helpers (other vehicles) can voluntarily share bandwidth with requesters and improve QoS by transmitting video through the established dedicated short-range communications (DSRC) channels.

Next, F. Yang [12] and W. Wu [17] diverge on how they approach the taxonomy of the rest of the applications. While the latter has two more categories, which are *Safety applications* and *Transportation efficiency applications*, the former does not do this differentiation. *Safety* applications, that aim at providing services to ensure safe driving through notifications and, possibly, car control. Given that it is the most researched technology, *Safety applications* mainly refer to collision avoidance systems (CAS), vehicle-based systems that serve two purposes:

- *collision warning* – warns the driver that a collision is about to occur, and it can also warn when a collision happened down the road, to prevent congestion.
- *driver assistance* – controls the car for steady-state or for an emergency intervention (like braking before a collision).

CAS can be extended to cooperative collision avoidance systems (CCAS), where vehicles share their information with neighboring vehicles in order to diminish even more the chance of collisions.

There is also some research on speed limiting applications that prevent speeding. M. Abdelsalam *et al* [19] developed a system based on RFID technology, with RF transmitters on roads, RF receiver modules on vehicles, and engine control units to establish speed values. One of the earliest works with CCAS was CarTALK 2000 [20], which developed *Cooperative assistance systems*, information, and warning functions.

Finally, *Transportation efficiency applications* are applications that aim at improving efficiency for vehicles and drivers similarly, by providing solutions that improve efficiency in things like fuel consumption and travel time. These applications can be divided into four different categories, based on their main work environment and objectives:

- *intersection control* – the biggest research area in efficiency solutions, as stated in both papers, these applications aim at controlling traffic at intersections, which requires complex solutions to reduce waiting time and retain fairness. They can be split into two types of approaches:
    - *traffic-light-based* - applications schedule traffic lights based on traffic volume with V2V (the cluster of vehicles at the intersection makes decisions) or V2I (a controller makes the decisions) communication approaches.
    - *non-traffic-light-based* – applications apply maneuver manipulation, controlled by the intersection controller, to drive on the intersection or vehicle scheduling algorithms.
- *route navigation* – also known as vehicular network-based navigation, it is researched to avoid the negatives of using GPS-based approaches. It utilizes parameters like real-time traffic information, fuel consumption, speed, and road condition data, etc. to choose the best route for the vehicle.

- *cooperative driving* – applications that aim at coordinating a group of vehicles, so they drive like one. This improves energy efficiency, traffic flow, and helps prevent accidents.
- *parking navigation* – applications that use algorithms in order to track optimal routes that lead to the closest available parking zones.

R. H. Huang *et al* [21] developed a cooperative adaptive driving application, a variation of cooperative adaptive cruise control, utilizing mobile edge computing and platooning techniques, to avoid accidents and improve traffic flow. H. Kowshik *et al* [22] present an algorithm for multiple vehicles that enforces safety in intelligent intersections through time slot allocation. W. Wu *et al* [23] developed algorithms that schedule vehicles with V2V communications, based on distributed mutual exclusion. P. Y. Chen *et al* [24] and K. Collins *et al* [25] present route navigation systems based on the vehicle's fuel consumption and traffic congestion on roads, respectively.

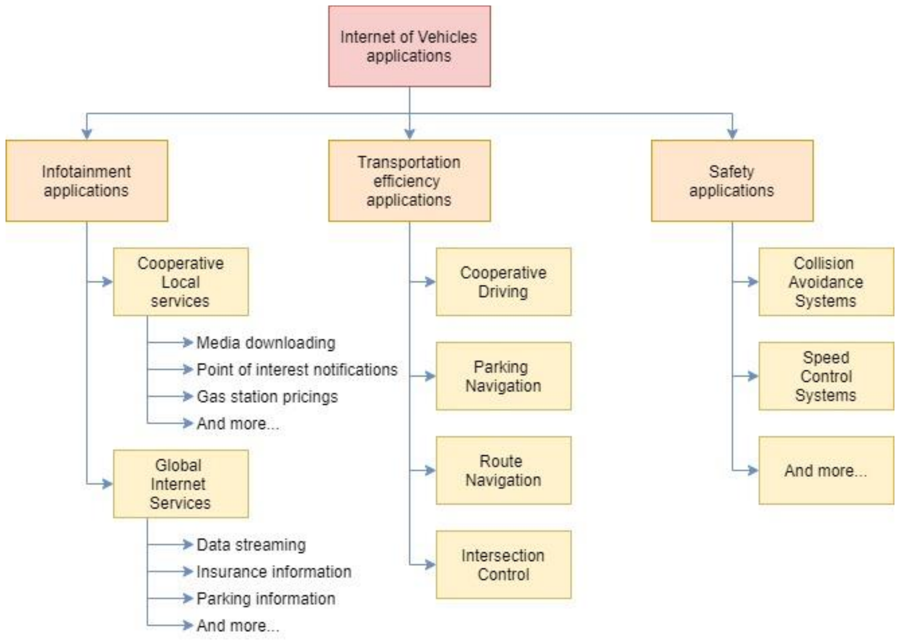


Figure 2.3 - Taxonomy of applications for the Internet of Vehicles.

Social Internet of Vehicles

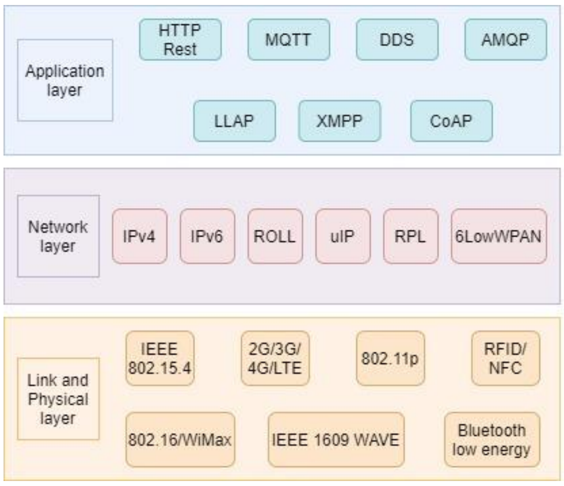
An evolution of the paradigm of IoV, Social Internet of Vehicles (SIoV) allows for the creation and management of relationships between vehicles in IoV, based on the context they are inserted, network architecture, or application requirements, behaving like a social network of vehicles [26]. These webs of relationships can introduce vehicles to other participants and enable vehicles to autonomously create new relationships that prove to be beneficial, be it because they improve traffic efficiency (like through sharing road information) or because these vehicles have data useful for installed applications [27]. This paradigm opens doors to new applications and can help the development of the applications mentioned above, given the focus on data exchange and trust relationships between vehicles and between infrastructural nodes. Adding to this, the usage of Deep Learning in the application scenario is also a possibility to get better results, as it has been researched and evaluated for IoT [28].

2.1.4 Main challenges

IoV faces challenges on many fronts, especially because it is such a recent paradigm with a lack of research.

Standardization

One of the main challenges in IoV is the need for the development of standards and protocols to achieve interoperability and ease the development of applications. Many consortia and organizations are trying to develop these standards and protocols that answer to the requirements posed by IoV and there are already some prominent protocols for these systems, as showcased in Figure 2.4. These protocols, proposed in the work of A. Al-Fuqaha et al [29], are utilized in IoT as well, as the two paradigms share similarities. Even so, with the imminent turn to the fifth generation, 5G, these protocols will probably need to be revised.



HTTP REST - HyperText Transfer Protocol REpresentational State Transfer;
MQTT - Message Queueing Telemetric Transport;
DDS - Data Distribution Service;
AMQP - Advanced Message Queueing Protocol;
LLAP - Lightweight Local Automation Protocol;
XMPP - Xtensible Message and Presents Protocol;
CoAP - Constrained Aplication Protocol
IPv4 - Internet Protocol version 4;
IPv6 - Internet Protocol version 6;
ROLL - Routing Over Low power and Lossy networks;
uIP - micro Internet Protocol;
RPL - Routing Protocol for Low power and Lossy Networks;
6LowWPAN - IPv6 over Low power wireless personal area networks
RFID - Radio Frequency IDentification;
NFC - Near Field Communication

Figure 2.4 - Protocols on the Internet of Vehicles.
Most of these protocols are already existing ones that are used in almost every type of network. However, it is expected that different protocols are constructed just for IoV, due to the specificity of requirements.

Resource constraints

While preprocessing information can lead to less network congestion and more accurate results in higher layers, the computing resources (computing power, energy power, storage space, etc.) limitations of devices in lower layers can limit the usefulness of doing it and even end up hindering the security aspect of the system, since the usage of cryptographic algorithms can become limited. The challenge here can lead both ways: make more powerful hardware or make algorithms for preprocessing/security that require fewer computing resources while maintaining the minimum levels of satisfaction for their implementation to be worth.

Service management

Given the large number of services that can be provided in IoV environments, it becomes a challenge for vehicles to manage services to obtain optimal solutions for them at a given point in time, minimizing costs and delivery time.

Node mobility

In IoV environments, the nodes (vehicles) are constantly moving, as they move in the real world. This regular mobility and the rapid topology changes related present a challenge when trying to keep vehicles connected to the network and the Internet, since they lead to packet loss, link failures, and even frequent network disconnections. But there is a difference from IoV to regular networks affected by mobility: the mobility in IoV is somewhat predictable since vehicles and their movement are limited to road layout and topology, road signs, and other vehicles.

Scalability

It is also extremely important that the IoV can have numerous amounts of devices connected at a given time without a significant decrease in performance, ensuring high scalability. Otherwise, it will not serve its purpose of achieving large-scale coverage and guaranteeing that services reach the users.

Data dissemination, data routing, and data management

The IoV also opens doors to the integration of different services and technologies [30]. With the heterogeneity of entities, services, and networks in IoV environments, disseminating and routing data from the source to the target destination, while guaranteeing a high quality of service becomes difficult. These heterogeneous environments also limit the coordination and collaboration of different networks and subnetworks. There is also the problem of data management, given that the entities in IoV generate massive amounts of data, making it difficult to aggregate, storage, process, analyze and provide decision making over this data. The challenge here is to design new protocols and schemes that can ensure that not only the data reaches the target under certain QoS limits, but also enable cooperation between networks and ease big data management.

Security

The tolerance to heterogeneity and interoperability in IoV systems, which is strictly needed to allow these thousands of different vehicles, sensors, and other components to communicate in the network, brings out a bigger need for data security. Zhang [4] states that vehicles operate in unprotected and vulnerable environments, with vulnerabilities in the cloud, V2V, and local communications. The security vulnerabilities in communications in V2X modes could open the possibility for cyber-attacks by manipulation of data streams or connection points [10]. Another danger is the possibility of hacking vehicles, which could lead to accidents and fatalities. In 2015, hackers at the Black Hat hacking conference demonstrated how they hacked vehicles through their PCs, far away from the vehicle, and it should be seen as a threat to security [10]. As such, researchers have proposed various ideas on how to reduce these vulnerabilities, while fulfilling the security requirements of IoV and safeguarding the network and its participants. B. Mokhtar et al [31] and S. Sharma [32] have listed the security requirements for IoV, which are summarized in Table 2.1.

In SIoV, with an even bigger need for data availability, connectivity, and autonomy of vehicles, there is a problem in maintaining security, especially, the privacy may be compromised due to secondhand data sharing. The new applications can also emphasize security vulnerabilities, due to inherent issues in communications or no control in data usage.

Table 2.1 - Security requirements for IoV [31].
These requirements have to be met in order to protect the vehicles and other devices that proliferate IoV environments.

Security requirement	Description
Data authentication	Vehicles identities should be verified when transferring data
Data integrity	Sent and received data should be verified to ensure correct data transferal
Data confidentiality	Data transmission between vehicles should be secret
Access control	Vehicles should be allowed to access services they are entitled to
Data non-repudiation	Ensure vehicle should not be able to deny the authenticity of another vehicle
Availability	Communication between vehicles should be ensured even under bad conditions in the event of an attack
Anti-jamming	Malicious vehicles cannot interrupt communications between other vehicles
Impersonation	A vehicle cannot impersonate another entity in the network
ID traceability	A vehicle’s identity can be retrieved from sent messages
Vehicle privacy/anonymity	Sent messages can only be accessed by authorized vehicles and remote nodes. The vehicle's identity should be hidden.

2.2 Blockchain

Blockchain is a technology that is deeply studied by researchers given its distributed aspect, which allows it to be implemented over P2P overlay networks, which are common nowadays, and the security benefits it brings, which are mentioned below, in Section 2.2.3.

2.2.1 Definition

Blockchain was firstly introduced in 2008, by Satoshi Nakamoto [5], as the technology behind Bitcoin, a virtual currency exchange system that uses cryptography (digital signatures) to avoid the implementation of trust-based models and, consequently, the requirement of a trusted third-party participant to validate transactions. Blockchain is described as a distributed ledger/database of transactions that can assure various security properties by the usage of consensus protocols and cryptography algorithms [33].

Given the description presented in [34], the ledger takes the form of a distributed chain of blocks, as represented in Figure 2.5, where, generally, every block has the following parameters:

- its hash,
- the hash of the previously added block,
- a timestamp
- a nonce (for calculations),
- the number of transactions,
- a Merkel tree of past transactions.

The ledger is distributed because every node in the network has knowledge of the chain of transactions, making them public for all, which also removes the single point of failure problem, which affects trust-based models with third-party authenticators. Before being added to the chain, a new block must be validated by the participants of the network through the usage of distributed consensus protocols, some of which are listed below. This, with the aid of cryptography algorithms and safe communication techniques, helps assure data integrity and traceability once a block enters the Blockchain while solving the problem of double-spending, as explained by Satoshi. The usage of consensus protocols removes the need for a third-party authority in transactions, which would reduce the system's throughput, and makes these transactions automatic.

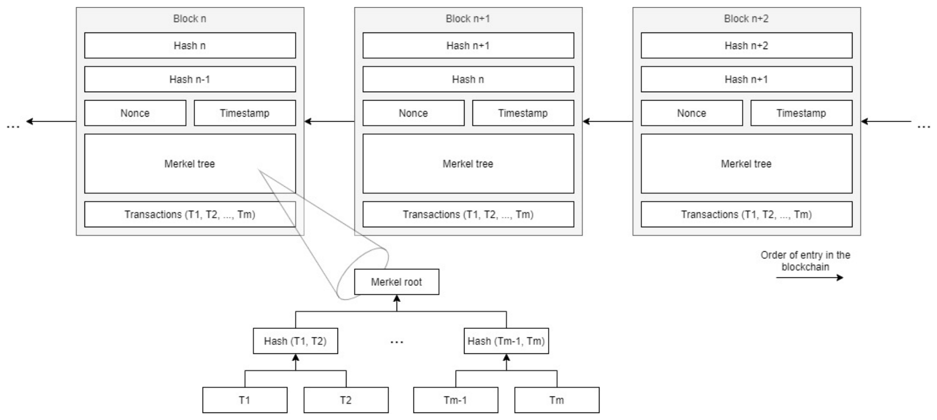


Figure 2.5 - Blockchain and Merkel tree structures.
In this case, this slice of the chain showcases three blocks, which are linked by each block saving their parent's hash inside itself.

Merkel tree

The Merkel tree each block has the root hash which is the hash of all transactions and each consequent non-leaf node is a hash of the concatenation of the two values below, forming a binary tree of hashed transactions. Given that transactions are hashed, any attempt at altering with transactions that have already been executed will result in different resulting hashes for the Merkel tree, allowing for easy detection of tampered transactions.

Chain forks/discrepancies

On somewhat rare occasions, two blocks might be appended to the Blockchain at the same time, pointing to the same parent block. Blockchain systems rely on consensus mechanisms to solve these forks [35]. For example, in systems that use the *proof-of-work* consensus protocol, the longest chain is the one picked since it signifies a bigger investment in effort [5]. In *proof-of-stake*, the chain with the most total consumed coin age is chosen [36].

2.2.2 Technologies

Blockchain utilizes various technologies to achieve its famous security and privacy properties, being consensus protocols and cryptography techniques the main two.

Cryptography

Each user has two keys: a public key and a private key. As stated in the work of M. A. Ferrag et al [37] and A. Dorri et al [38], asymmetric encryption is used for communications, where the public key is utilized for encrypting transactions, and the private key is utilized to decrypt the transactions. The private key is also used to sign transactions before sending and the receivers can verify this signature by using the sender's public key. This verification is made by every node that receives the transaction, which then disseminates it further. When the nodes that take part in the transaction verify it and accept it, it is validated, placed on a timestamped block with the required hash of the previous block. This block is then broadcasted back to the network, verified by the nodes, and then added to the Blockchain.

Consensus protocols

Blockchain utilizes distributed consensus protocols to validate transaction blocks before they are inserted into the Blockchain, which ensures data immutability. More than one consensus protocol can be implemented together, to fulfill application requirements.

The pioneer consensus protocol for Blockchain was proof-of-work [5]. In PoW, the nodes are tasked to solve arduous mathematical puzzles, a task also called "mining". Once a nonce is found that gives the block's hash a given format (usually hash has to start with n zero value bits), the node that found it broadcasts the solution to the network and when, at least, 51% (the majority) of the nodes verify it, it can be added to the Blockchain. The PoW computation costs, especially at the energy level, can difficult the implementation in new systems that rely on lightweight nodes, like IoV.

Following the work of A. Kiayias et al [36], PoS introduces a new concept of coin age, which is defined as currency amount times holding time. This new value can then be used by the owner to "pay himself", through a special transaction called coinstake which enables the generation of a block for the owner, while consuming the specified coin age. The hash of the block generated needs to reach a certain hash target protocol, but in a limited search space (in contrast with PoW's unlimited search space) greatly reducing the energy consumption and, consequently, Blockchain's energy dependency. PoS also grants better block generation and transaction confirmation speeds, since only one block is created each cycle, with fewer nodes proposing blocks by round [39]. In Delegated PoS, a group of witnesses is voted by the nodes to generate blocks and are shuffled after a certain condition is met (be it time restrictions or blocks produced [40]), and are paid for each block generated. Leased PoS is an enhance of PoS to solve the This is solved by having these nodes leasing their stakes to richer nodes and, when these richer nodes get the chance to generate a block, both get the reward [41].

Another common consensus protocol is Byzantine Fault Tolerance, where nodes vote for a majority decision. Various implementations of BFT have been implemented, like Practical BFT [42]. A proposer node, which changes in round-robin order, broadcasts a pre-prepare message which can only be accepted by the rest of the nodes if they have not accepted one already. If the majority does accept, the proposer sends a prepare message. Once other nodes are prepared, the proposer sends a commit message to all of them and, once the message is accepted, the state of the Blockchain is altered in each node. PBFT adds a timeout condition to tolerate faulty primary nodes (proposer)[43]. There's also a delegated BFT, where nodes are divided into two roles:

ordinary and bookkeepers. Ordinary nodes do not take part in deciding consensus, only choosing which bookkeeper nodes they back. A random bookkeeper is selected, and it broadcasts its transaction data to the network and, when 66% of bookkeepers accept the data as valid, it is added to the Blockchain [44]. The majority of BFT protocols can handle 1/3 of faulty nodes.

Proof-of-Activity, as described by I. Bentov et al [45], is a consensus protocol that is a hybrid between PoW and PoS. The protocol starts by using complex mathematical tasks (the pseudorandom number and the finding of the satoshi, the creation of an empty block header, and the derivation of pseudorandom stakeholders from a hash). When nodes have enough stake, PoS algorithms start to take place (nodes with more stake have higher chances of being chosen). Forks in the chain are dealt with just like in PoW (longest chain represents the biggest amount of effort) and the fees for transactions are divided by all found stakeholders.

Presented as an alternative to PoW and PoS [46], in the Proof-of-Burn protocol nodes are encouraged to spend their currency, and only then can they generate blocks and get the respective rewards. In Slimcoin [47], PoB is used as PoS, where the higher the amount of burned currency, the higher the chances of generating a block in the next round (instead of coin age). The burning of currency also keeps its value overall, since there is less currency after the operation, making it rarer. The nodes burn the currency and receive a proof-of-burn, a string that proves that the burning took place, that is then used to receive a given reward [48].

The Proof-of-Elapsed-Time was first proposed by Intel for its Sawtooth Lake platform. In this protocol, each node generates a random time value that has to follow a distribution set by the scheme. This time represents the waiting time before the block can generate blocks and can be updated every time the node generates a new block. This protocol solves the energy dependency problem of PoW and the “one CPU one vote” problem presented by Satoshi Nakamoto [49].

The Proof-of-Capacity protocol is a protocol that works similarly to PoS, where the “stake” is the capacity, or storage space, of the hard drive of nodes. The bigger this value, the higher the chances of nodes get to generate blocks [50].

The Proof-of-Authority protocol was originally planned for Ethereum based private networks. This protocol has a set number of nodes (called authorities) that achieve consensus between them to generate blocks requested by clients. Every time a “step” (the unit that demarks time in PoA) passes, a new leader is elected from the authorities [51]. This protocol works better in private and consortium type Blockchains where there is a known set number of nodes.

In the Proof-of-Importance protocol, the nodes have a ranking that grows with each successful validation of blocks and transactions made. Nodes with higher rankings have higher chances of generating blocks and, as such, the network itself has a higher trust value between nodes [52]. It differentiates itself from PoW and PoS because it allows smaller nodes, with fewer stakes or CPU power, to also participate in the network, making it so only participating nodes (the only ones beneficial to the network) get rewards.

In Proof-of-Luck, a random number is assigned to each block (its luck). Every time a node wants to start to generate a block, it waits a set interval of time, receiving other blocks from other nodes and, if during this interval the node receives a luckier block, it substitutes its own (only if the parent block is the same). If after that interval, his block is still the luckiest, the node broadcasts his block to the network and proceeds to generate the hash of the block header for the next block [53]. In this protocol, forks are common and, as such, nodes verify the total luck of each chain and choose the one with the best total, as it represents the chain with the closest desirable behavior.

In the work of A. Shoker [54], the Proof-of-eXercise protocol, based on PoW, is proposed and tries to solve some of PoW's problems, like Puzzle hardness, Block sensitivity, and Easy verification. In this protocol, nodes are challenged to solve computation-intensive problems (massive matrix operations), instead of solving hash-related tasks. This proves that the computing power of the Blockchain can also be used to solve real scientific problems, which opens doors for investigation.

Smart contracts

Szabo N. proposed the idea of smart contracts in the 90s, defining them as a "computerized transaction protocol that executes the terms of a contract". In the context of Blockchain systems, smart contracts can be defined as small pieces of executable software programs that execute, in an automatic and independent fashion, instructions, when certain previously accorded conditions are met [55]. These executions are accounted for as transactions for storing purposes, being inserted in a block, and added to Blockchain every time they are executed. Given the aforementioned properties of smart contracts, they offer appropriate access control and contract enforcement. It is also possible to conclude that these contracts are deterministic, since every input returns a defined output, even if the contract is called repeated times.

The smart contract lifecycle is composed of four phases, depicted in Figure 2.6:

1. **Creation**:
    a. Negotiation of obligations, prohibitions, and rights.
    b. It is an iterative process with multiple rounds until an agreement is reached.
2. **Deployment**:
    a. Deployed to platforms on top of Blockchains.
    b. Digital assets of involving parties are locked.
    c. A new contract has to be created for new changes, due to the immutability property of Blockchains.
3. **Execution**:
    a. Automatically executed when the previously negotiated conditions are met.
    b. Resulting transactions and updates are stored in the Blockchain.
4. **Completion**:
    a. Every transaction has been completed and the currency has been transferred/removed to/from the involving wallets.
    b. The digital assets are unlocked and available for other transactions.

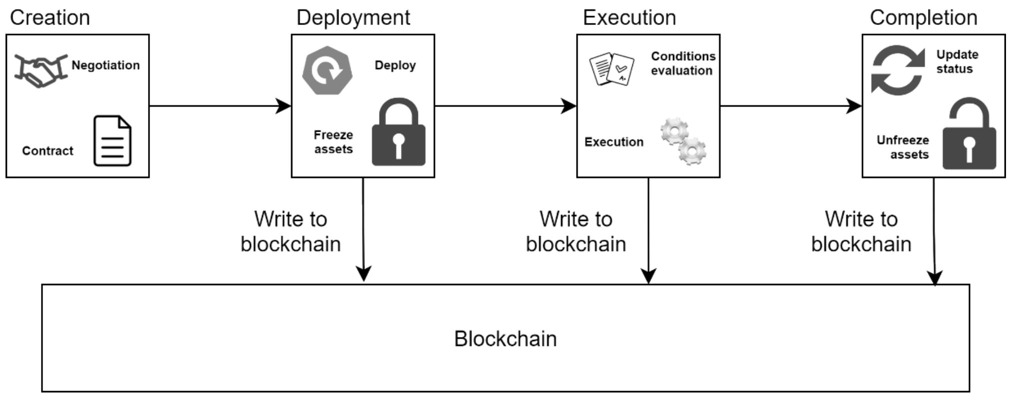


Figure 2.6 - Smart contract lifecycle.

2.2.3 Benefits and challenges

Types of Blockchain

There are three types of Blockchains: public, private, and consortium Blockchains, which are a combination of the first two.

Public Blockchains have to be decentralized, immutable, preserve data non-repudiation, transparency, and traceability of transactions but have low scalability and flexibility. Public Blockchains do not require permission for access (permissionless), which means any node can enter and leave the network at any time.

Private Blockchains can be the opposite: centralized, mutable, do not preserve data non-repudiation, transparency, or traceability of transactions, but have high scalability and flexibility. They require permissions before accessing the Blockchain (permissioned), which means nodes require permissions to access the network, call functions, or even see data.

In the middle of both, there are Consortium Blockchains. These Blockchains can be partially decentralized, partially immutable, partially refusable, and partially traceable while achieving somewhat good scalability and flexibility. They are permissioned as well.

The usage of these three types of Blockchain depends on where and how they are to be deployed and implemented in the environment. Usually, private Blockchains are used in private networks. Public Blockchains are the only type that has to guarantee a certain number of properties, or the safety is compromised, while Consortium and Private Blockchains can have any degree of any property, depending on what they are used for.

Table 2.2 - Summary of comparison between types of Blockchain on various security and network parameters. Private Blockchains offer the possibility for the properties mentioned under it, but they can also hold the properties listed under public Blockchains. However, the same does not apply backwards, due to the nature of the networks each type of Blockchain is implemented on top of.

Property	Type of Blockchain		
	Public	**Private**	**Consortium**

Centralization	Decentralized	Centralized	Partially decentralized
Transparency	Transparent	Nontransparent	Partially transparent
Traceability	Traceable	Traceable	Partially traceable
Mutability	Immutable	Mutable	Partially immutable
Data Repudiation	Non-refusable	Refusable	Partially refusable
Scalability	Low	High	Good
Flexibility	Low	High	Good
Permission	Permissionless	Permissioned	Permissioned

Given the information above, the following is deductible: Blockchain assures *Data authentication* since every user has a pair of keys and their public key is identifiable. As such, it also ensures the property of *Data non-repudiation*, because since data is hashed with the user's keys, the data is connected to that user and its actions cannot be denied. Each private and public key is unique and since they are used to sign block hashes and transaction records, *Impersonation* becomes almost impossible. *Privacy* is maintained, but not completely, by making Blockchain addresses anonymous (but the key management authority will always know what vehicle a key belongs to). *Data integrity* is maintained since data blocks have hashes that are calculated once and, if the data is tampered with, it would generate a new, different hash, which would be detected rapidly, as mentioned above. The properties of *Data confidentiality* and *Access control* are ensured by the usage of cryptography in the exchange of messages and smart contracts. Finally, Blockchain can also assure *Availability*, by allowing communication in heterogeneous environments, since it can be deployed in overlay P2P networks. These properties of the Blockchain cover most of the security requirements of IoV systems mentioned in Table 2.2.

While having many benefits, typical Blockchains does suffer from low throughput, specially PoW-based Blockchains like Ethereum [56], and the cryptographic functions can be way too computationally and energy-intensive for the resource-constrained devices that are now proliferating in new network paradigms like IoV, especially when using consensus protocols like PoW. There is also the debate between assuring complete privacy and anonymity or having data traceability, since achieving complete privacy and anonymity does not allow for the Certificate Authority (CA, which is the entity that assigns keys to nodes) to trace malicious transactions to the dishonest nodes in the network. A Public Key Infrastructure (PKI) is used by Certificate Authorities to achieve privacy while guaranteeing traceability, making it so there is no way that attackers can link public keys to real identities [35].

2.2.4 Applications

There are published surveys [57][58][59] that explore how Blockchain has been implemented in growing systems to assure the security and privacy needs of applications for various purposes.

Some areas where Blockchain applications have been developed or show promising future paths are as follows:

Finance

Blockchain started by being used for cryptocurrency exchanges over networks ever since its genesis in 2008. New cryptocurrencies are still being developed, but there is also an investment in research for their application in banking and financial development. Q. K. Nguyen [60] stated that Blockchain has the potential for sustainable development of the economy and financial growth.

Healthcare

Healthcare systems are full of very different technologies that have to work together to deliver the best service to users. As such, Blockchain is a great technology for guaranteeing interoperability in these highly heterogeneous systems [61]. There is also a research path involving the usage of electronic health records (EHR) and their relevance on improving health services responses and, consequently, their quality [62].

Governance

Blockchain can improve the management of citizen records and certification. One of the major research paths is e-voting. With e-voting, everyone with access to technology would be able to vote and engage more easily with the political life of their surroundings, without worrying about transportation or wasting time in lines. *FollowMyVote* [63] is a proposed project for e-voting with end-2-end communication, based on BitShares. *BitCongress* [64], a discontinued project, used Counterparty and Ethereum smart contracts to manage voters and votes.

Business and Industry

In terms of Business and Industry, Blockchain brings promising implementations. Mostly focusing on the problem of supply chain management and energy management. For the supply chain, Blockchain can serve to improve characteristics like visibility, optimization, food safety, or even the automatization of transactions between intermediaries. *Coindesk* [65] is a project developed by IBM and Walmart to help manage China’s pork meat market, to improve safety in the supply chain. In the energy management path, Blockchain can allow for cost reduction and better planning, and improve the energy market system for costumers and providers [66].

Internet of Things

Blockchain has been adapted for various needs and various functionalities in the Internet of Things. Recent researches utilize the Blockchain as a middleware to ensure security and cover the security problems of these paradigms, creating what is referred to as the Blockchain of Things (BCoT) [67].

Other relevant areas of R&D

Education, Security, Integrity Verification, and Data Management are also favorable areas of research for Blockchain implementations. The Internet of Vehicles is also a favorable research area for Blockchain implementations and is in the section discussed below.

2.3 Blockchain-based solutions on the Internet of Vehicles

Given the characteristics of Blockchain and the security challenges of IoV, research and development for implementing the former in the latter have been growing over the years. Most of these implementations try to solve some security problems regarding data sharing in IoV networks.

In the work of G. Rathee *et al* [68], a Blockchain-based framework to provide security, safety, and transparency for users, and vehicles, in cab-sharing scenarios (e.g., Uber) is proposed. In this framework, smart contracts are used so information is not accessed and altered wrongly by unauthorized parties (i.e., secondhand data sharing). Vehicles and IoT devices need to be registered in the network to access its services, via logging of information in a database and the Blockchain for traceability. IoT devices store their activities in the Blockchain, instead of vehicles, to ease detection of malicious behavior while maintaining a reasonable level of computational and storage costs.

For the Blockchain, managers are elected (a primary and a secondary, which takes over in the case of a faulty primary) to manage the Blockchain for some time and receive registration requests from other nodes, be they vehicles or singular devices. There are also miner and peer nodes, and the former helps validate the authentication of registration requests mentioned above. The managers then verify the requesters' authenticity and generate their public keys if the authenticity is verified. Implementing a rating-based system for providers allows users to evaluate which one will be chosen to give them a ride, providing a way to incentivize good behavior from providers. Providers can also choose which user to attend to based on multiple parameters. A Blockchain is maintained between the user and the provider to detect alterations of values (e.g., geographical position, time, route, among others).

The evaluation was made through a simulation in the NS2 network simulator with Blockchain techniques, aiming to get results under network congestion and compromised nodes, and comparing them with results from an already existing approach. An attacking scenario with an adversary model (i.e., malicious/hacked nodes present in the network) was utilized to test the proposed framework. Tests were made with malicious nodes passing as vehicles, IoT devices, and Blockchain nodes (miners and peers). The authors conclude that the proposed approach surpasses existing ones, with 86% success and accuracy rates, which are theorized to get higher with time due to the removal of ill-intended nodes. They also verify lower levels of network congestion with fake requests and that the possibility of attack with *n* compromised nodes was lower for the proposed approach. The altered amount of data, in the case of intruders messing with user's ratings, was also lower in the proposed approach. While still having some edges that need polishing, the proposed approach results achieved a 79% success rate compared with the already existing approach.

X. Wang *et al* [69] present a new scheme for vehicle registration and authentication in IoV based on Blockchain technologies, like Ripple and BFT consensus protocols, and smart contracts, with the purpose of eradicating the impact of malicious vehicles.

When a new node wants to join a contract group, it applies to it and is then evaluated by each node of the group through gathered data on that node in the infrastructure. If more than 51% of the nodes accept the node's integration, it is added to the group. Otherwise, it will be added to a watchlist of suspicious nodes and following applications to the contract group will be met with more restrictive conditions.

A distributed PKI system is utilized for trusted key distribution. For authentication, the vehicle sends a request to RSUs with their ID and public key. The RSU encrypts this data and sends it to the cloud service provider that, through the usage of consensus protocols, verifies the authenticity of the vehicle's identification.

The evaluation of this scheme was made using the Veins network simulator. The purpose was to simulate the average time and communication costs of the system. Veins provide a Mean operation that allows for checking time consumption in the various phases of authentication. The costliest phase was concluded to be the encryption in the key distribution center, with an average of 9 ms. The average time overhead for each layer was lower than 9 ms, which proves that the scheme can respond to requests promptly. The communication costs are, on average, 17 Kb for vehicle registration and 8 Kb for RSU verification, which is within the bounds of reasonability of technology for large-scale traffic networks.

J. Gao et al. [70] presented a system architecture for IoV where Blockchain, Software-Defined Networks (SDNs), and Fog computing. Blockchain solves security issues and settles trust between entities without a centralized trusted authority, as explained before. SDN technologies separate the control and data planes in a network to reduce structural complexity making nodes simply transmit data. At the same time, an SDN controller manages resource allocation, mobility, and rule generation. The fog computing approach helps reduce handovers in the network, which increases performance, and the RSUs and OBUs (onboard units) that are present in these fog zones, are SDN-enabled which means the SDN controller also controls them. The authors also define RSUH (RSU hubs), which manages the control overhead between the controller and RSUs. A reputation-based trust system was implemented between vehicles to detect falsified information and help vehicles transmit only useful information while reducing the impact of attackers.

The evaluation was made with MATLAB combined with the NS-3 network simulator, using Hyperledger Fabric as the Blockchain platform, and the nodes were virtual machines deployed in Ubuntu containers. The two metrics that were taken into consideration for measurements were Packet Delivery Ratio and Transmission Delay. The former metric showed to be influenced by distance (optimal results were between 200 m and 500 m, due to route discovery and an increase in hops), the propagation model is chosen, the number of packets sent (i.e., more packets meant more collisions and a lower ratio), the number of vehicles and their speed (i.e., dense networks cause packet loss and a lower ratio). The Transmission Delay was influenced by distance (optimal results between 200 m and 500 m, due to MAC and the number of hops to the destination), the number of packets (i.e., the higher the number, the higher the contention and, consequently, the interference between nodes) and the density of vehicles in the network (i.e., more vehicles leads to more congestion). Due to these results, it was deemed to be a viable solution to Blockchain-enabled IoV communication and trust problems.

M. Kamal *et al* [71] tackle the computation and energy power consumption in IoV during communications. They present changes to already existing protocols and software with no additional hardware requirements. The authors then propose low complexity solutions for operations like connectivity check, Blockchain development, and data provenance and forensics, including lightweight algorithms.

The experiments were made with three MICAz motes [72], a wireless measurement system, each placed in a different car. The results obtained were then transferred to MATLAB to produce a simulation. The Received Signal Strength Indicator (RSSI) was measured to check the performance of the solution. Tests were made for Adversary Detection: first without an adversary (with and without a filter applied to the results) and then with an adversary (man-in-the-middle), and the Pearson Correlation Coefficient was calculated to check for value correlations (-1 for anticorrelation and 1 for high correlation). The filter test showed smoother results and value changes than the unfiltered one, with a high correlation in both cases (0,9754 and 0,9349, respectively). With the man-in-the-middle attack, the results were disparate and presented close to no correlation (0,1282). Tests were also made for multimedia sharing, which proved to be able to detect forged images by hash comparison. The Blockchain itself, due to its immutable property and how the block structure works, also serves to detect tampering and malicious activity to data (as explained before). In terms of time complexity, it was calculated by increasing the size of the link fingerprint (the binary version of the RSSI). From 30 to 3840 bytes, the time complexity was $O(1)$, the lowest in cryptographic-based solutions. After 3840 bytes of size, the complexity turned to $O(2^n)$, which would never happen in ideal situations. As such, it is concluded that data sharing and authentication can be achieved with lightweight encoding mechanisms and procedures.

Table 2.3 summarizes and compares Blockchain solutions based on their motivation, what is proposed, and the evaluation tools used.

Table 2.3 - Comparison of Blockchain solutions based on the motivation behind the work, what is proposed and the evaluation tools utilized for the evaluation.

Blockchain Solution	**Motivation**	**Proposed Solution**	**Evaluation Tools**
Rathee et al. [68]	Ensure security, safety, and transparency	Blockchain framework with smart contracts	NS-2, Blockchain platform not specified
Wang et al. [69]	Reduce the impact of malicious nodes efficiently	Authentication and registration scheme	Veins
Gao et al. [70]	Increase performance with new paradigms	Blockchain-SDN-enabled solution with fog computing	MATLAB, NS-3, and Hyperledger Fabric
Kamal et al. [71]	Decrease power consumption	Various (lightweight algorithms, lesser complexity)	MICAz motes and MATLAB

2.4 Tools

Various tools are used to simulate and evaluate Blockchain solutions in IoV networks. On the one hand, there are network simulators that allow for network construction and protocol integration based on development requirements. On the other hand, there are Blockchain platforms that provide support for Blockchain functions in the desired network environment. Most of the time, these technologies are used together to run a Blockchain system on nodes of a simulated network to get results that resemble more closely the results that would be obtained in real-life scenarios.

2.4.1 Network simulators

Network simulators provide various properties and support for various types of networks. Given that the survey focuses on IoV, only simulators that support vehicular networks are considered for this work.

MATLAB

MATLAB was not made to be a network simulator specifically, but it is possible to simulate generic networks with it and Simulink. It has a VANET Toolbox [73] for vehicular network simulations, including support for V2V communication, but lacks development for V2I communications. MATLAB also offers 5G, LTE, and WLAN Toolboxes [74] [75] [76] for support of 5G, LTE, and WLAN technologies. Many other Toolboxes serve other purposes or support other technologies. The MATLAB community also has developed simulation software that is available for download. Even so, MATLAB is still used with other simulators to achieve better data visualization of simulation results.

NS-2 [77]

NS-2 is a generic discrete-event network simulator that supports simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks, mobile network simulations with node movement, protocols, and traffic connection. NS-2 has an extensive manual with information on topics like Routing, Packet forwarding, Applications, and Transport in the network. It also provides help for installation and application. Nam (Network Animator) is a tool that can be used for network visualization (including network topology and packet-level animation) with data from NS-2. A negative side of NS-2 is that, as mentioned by the authors, it can have bugs that affect results (some unreliability).

NS-3 [78]

NS-3 is a generic discrete-event network simulator that is well documented, easy to use, and debug, caters to the needs of the entire simulation workflow, from simulation configuration to trace collection and analysis. It supports real-time scheduling and network simulations, and as such real-world protocol implementation, and simulation for IP and non-IP-based networks.

NCTUns [79]

NCTUns is a high-fidelity and extensible generic network simulator that **s**upports distributed simulation of large networks over multiple machines. Has support for various protocols (WiMAX, TCP, UDP, 802.11p, etc.) and types of networks, including support for ITS (V2V and V2I communications). NCTUns is an open-source and open system architecture, which enables the implementation of new protocols.

Veins [80]

Veins is a discrete-event vehicular network simulator based on OMNet++ [81], a discrete-event simulation platform, and SUMO [82], a mobility simulator. Veins is open source, which allows for customization of protocols by developers. It features support for various technologies according to IEEE standards, including 5G, model implementation from MATLAB, and human-driven or autonomous vehicle mobility simulations.

Summary

All the network simulators mentioned above support various network technologies and protocols that are valuable for research and development. While MATLAB is not a network simulator, serving best for data visualization and data management after the simulation is concluded, it can still be used as a generic network simulator with Simulink. MATLAB providing multiple pre-existing algorithms that can be deployed in the graphical environment of Simulink. It also allows for the input of datasets to fuel the simulations while providing the data visualization functions it is known for. Ultimately, MATLAB and Simulink can be used as a vehicular network simulator with the usage of the available toolboxes mentioned above that offer various technologies that are common on vehicular network simulations. These features distinguish the MATLAB+Simulink simulator from other generic and vehicular ones.

NS-2 and NS-3 share similarities, with NS-3 being a bit newer, while NS-2 has the downside of unreliability from bugs even though it does have more thorough documentation, but both are popular network simulators. NCTUns' author states that it was built with a new methodology (kernel re-entering), which gives it an advantage against other regular simulators (like NS-2), which eases the simulator consumption of CPU. Finally, Veins is specifically a vehicular network simulator, focusing solely on vehicular nodes and road infrastructure and technology, and this makes it the most recommended for simulating IoV networks since simulating node mobility in generic simulators, like the other mentioned above, can be troublesome.

2.4.2 Blockchain platforms

Many Blockchain platforms exist on the market for various purposes and needs. Most of them are community-developed platforms and, hence, open-source.

Hyperledger

Hyperledger provides several Distributed Ledger Technologies for various enterprise business applications. These technologies were built to be integrable with each other, for mutual benefits.

Hyperledger Besu [83] is an Ethereum client made to interact with public Ethereum networks and private networks, that runs on an EVM. Even so, it has the proper permission schemes for consortium networks. Besu offers the possibility for the utilization of PoW, PoA, and IBFT consensus protocols, and it is programmed in Java. Through the integration of tools like Truffle, Remix, and web3j, Besu offers the means for smart contract and dapps development. However, Besu does not provide support for key management (creation, attribution, authentication), relying on a third-party key management tool, like EthSigner.

Hyperledger Burrow [84] is a Blockchain node and smart contract execution engine, that runs on an EVM, and is optimized for public networks, but can also work in permissioned (private or consortium) networks. Burrow is developed in Go and offers consensus through BFT with the Tendermint algorithm. It was made to make Blockchain implementation simple, light, and fast. Burrow provides developers with bare-metal implementation, instead of virtualization (containers). Smart contracts in Burrow are On-Chain and are written in the native language. It trades configurability and modularity with tight coupling of components, which removes the need to care about the underlying infrastructure.

Hyperledger Fabric [85] is a framework, developed in Go, for private permissioned networks. It is highly modular, configurable, and pluggable, giving the developers the freedom to choose what technologies to implement and how to implement them (e.g., consensus protocols). Even so, generic Fabric works with PoS, but it can be changed to any intended one. Smart contracts are installed in the nodes and are developed in Golang prior to version 1.0, inclusive, or in Javascript from version 1.1 forward, and run in container environments. That said, a new approach to how the execution of smart contracts is verified by peer nodes removes the worry of non-determinism since deviant results are filtered before consensus, which allows Smart contracts to be written in standard programming languages and executed in a parallel fashion. Finally, Fabric doesn't require the implementation of a cryptocurrency for consensus.

Hyperledger Iroha [86] is a simple Blockchain framework, written in C++, for private permissioned networks, that is made to help and ease application development with a variety of libraries, offering a lower complexity and easier management solution, with verifiable data consistency at low costs (with no mining, since it does not have a cryptocurrency) of Blockchain. It uses the high-performance YAC consensus protocol and provides built-in smart contracts (commands), which are On-Chain. Iroha presents robust permission and role-based access control over the nodes and the network.

Hyperledger Sawtooth [87] is a highly modular and configurable Blockchain framework for private networks that offers an easy environment for application developers by allowing them to program in any language. Contract abstraction allows developers to write contract logic in the language of their choice through SDK transaction processors. Sawtooth offers Raft, PBFT, and PoET as options for consensus protocols and a novelty is the fact that these consensus protocols can change during runtime, to cater to the needs of the users. In Sawtooth's architecture, there is only one node type, which eases development, simplifying interactions.

Ethereum [88][89]

Ethereum is a simple, highly modular, and programmable Blockchain platform made for public permissionless networks. It introduced the Ethereum VM, a Turing-complete machine, which is used by other Blockchain platforms (like Hyperledger). It was intended for digital currency payments and transactions with the need for third-party authority, with its digital currency unit being called Ether, but it can also be used for any other case of data exchange over a network. Ethereum works with PoW as a consensus protocol, but it is now in the midst of changing to PoS, due to PoW's high computational costs and energy consumption. It introduced fees or "gas" as a heuristic for miners, to choose transactions from the transaction pool, and users, to prevent infinite-loop attacks. Smart contracts for Ethereum are mostly written in Solidity (Serpent and LLL can also be used for writing contract logic).

Corda [90]

Corda is an open-source Blockchain platform for private networks, developed in Java. Corda differs from other Blockchain platforms in the fact that each of the nodes does not hold a copy of a universal Blockchain, but only a ledger of the transactions in which they took part. Communications between nodes are point-to-point and there are no message broadcasts about transactions. Each node in Corda is a Java VM environment with Corda services or CorDapps. Smart contracts are written in Java. Contracts can only verify internal validity and Corda implements Oracles that verify external data, when required, for transactions and contracts. Since there is no broadcast, Notary clusters are used for uniqueness consensus, to prevent double-spending of currency and validate transactions. Finally, consensus protocols for Corda are completely pluggable, giving the developers the freedom to choose the one that fits the best.

Tezos [91]

Tezos is an open-source platform for assets and dapps development in public networks. Tezos differentiates itself from other platforms by introducing self-amendment and on-chain governance, which allows for self-upgrades of the platform during runtime without heavy forks, which can lead to security leaks. Smart contracts in Tezos can be written in various high-level languages, but are always compiled to Michelson, which helps reduce the risk of smart contract exploits. Finally, in Tezos, the consensus is achieved through a delegated PoS protocol.

Table 2.4 presents a comparison of Blockchain platforms based on network types, consensus protocols, programming language, and smart contract languages.

Table 2.4 - Blockchain platform comparison based on the network types they are integrated onto, the consensus protocols utilized, the base programming language and the programming language used to create and run smart contracts.

Blockchain Platform	Type of Networks	Consensus Protocol	Programming Language	Smart Contract Language
Hyperledger Besu	Public (Ethereum) and private networks	PoW, PoA, IBFT	Java	Truffle, Remix, web3j (Tools)
Hyperledger Burrow	All (public optimal)	BFT (Tendermint algorithm)	Go	Various
Hyperledger Fabric	Private	Various (pluggable)	Go	Golang (<1.0)/ Javascript (>1.1)
Hyperledger Iroha	Private	YAC	C++	Various
Hyperledger Sawtooth	Private	Raft, PBFT, PoET	Various	Various
Ethereum	Public	PoW, PoS (2.0)	Various	Solidity, Serpent, LLL
Corda	Private	Various (pluggable)	Java	Java
Tezos	Public	PoS (delegated)	Various	Various (Michelson)

2.4.3 Discussion

Given the proposed work, a Blockchain platform that allows the integration of self-made consensus protocols is required since the implemented cryptographic alternatives have to be integrated as alternatives to already existing protocols. As such, platforms like Hyperledger and Ethereum are the most suitable for this work since they offer the pluggable consensus property, which allows developers to integrate their protocols into the platform and also work with PoW based consensus. Since Hyperledger's is made for private networks and its consensus protocol is based on a Raft protocol that works apart from the nodes, Ethereum is the most favorable choice of Blockchain platform, not only because it is made for public networks, but also because it is possible to compare the proposed approaches to an already existing and functioning PoW-based consensus protocol, Ethash [92].

In terms of network simulators, NS-3 and Veins have both been reported to be compatible with Hyperledger and Ethereum. Veins is a simulator exclusive to vehicular networks, while NS-3 is much more generic, which places Veins at an advantage for evaluation since it focuses on simulating the needed IoV scenario and nothing more.

2.5 Criptographic primitives

Devices in IoV networks are usually small and cheap and, as such, have constraints in terms of computational power and energy capacity. Given these characteristics and aiming at maximizing the lifespan of these devices, implemented algorithms and programs avoid high complexity, to avoid low throughput and fast battery drainage.

Blockchain consensus algorithms propose hard-to-solve puzzles that require high computational power and energy, which clashes against the resource constraints of IoV devices. As such, research on lightweight consensus protocols for Blockchain is a growing topic.

As such, this section explores new approaches to possible Proof-of-Work mechanisms that can replace the resource-heavy algorithms that exist at the moment.

P. Golle *et al* [94] introduce new cryptographic primitives using heuristic constructions like CBC-RSA, Two-root RSA, and assumptions such as *n*-Power Computational Diffie-Hellman or *n*-Power Decisional Diffie-Hellman.

The work proposes two Communication Enforcing Schemes (CES): CBC-RSA and Two-root RSA. CBC-RSA scheme applies RSA signatures to messages and, to reduce the size of the output result, CBC-chains the hashed messages and the signature of this scheme is equal to the signature of the last block. However, this requires high computational power in the case of heavy files since it could require thousands of RSA operations to hash them.

Two-root RSA scheme searches for two roots ($x, y \in \mathbb{Z}_N$, with N being a product of two primes and $x - y$ is coprime with N) of a random polynomial function, based on the hashed values of the original message, which are the signature of the message. Since there's a $1/9$ and $1/4$ probability of finding the roots, it takes between 4 to 9 attempts before finding the two roots.

On the Storage Enforcing Scheme (SES), there's a verifier V, a prover P, and a message source S. Given a group G with prime order p and generator g, the *n*-Power Decisional Diffie-Hellman (*n*-PDDH) assumption states that no polynomial-time algorithms can differentiate between the following two distributions on a random guess:

$$Distribution\ P^n: (g^x, g^{x^2}, g^{x^3}, \dots, g^{x^n}), where\ x \overset{R}{\leftarrow} \mathbb{Z}_p, \tag{2.1}$$

$$Distribution\ R_n: (g_1,\ g_2, \dots, g_n), where\ g_1,\ g_2, \dots, g_n \overset{R}{\leftarrow} G. \tag{2.2}$$

The n-Power Computational Diffie-Hellman (n-PCDH) assumption states that no polynomial-time algorithm can compute g^{x^n} given $g^x, g^{x^2}, g^{x^3}, \dots, g^{x^{n-1}}$.

The designed scheme limits the size of messages to m blocks, where blocks are part of $\mathbb{Z}_p$. With $n = 2m + 1$ and assuming n-PDDH holds in G: the verifier V gets a private key x from $\mathbb{Z}_p$ and the corresponding public key is:

$$PK = (g^x, g^{x^2}, g^{x^3}, \dots, g^{x^n}) = (g_1, g_2, \dots, g_n) \tag{2.3}$$

Through the DH assumptions, V can prove that the key is correct by having P compute a DH-tuple of values. S then divides the message into m blocks and places them in a tuple, where it appends a random element to the tuple from $\mathbb{Z}_p$ (M_{m+1}). S then computes $f_0 = \prod_{i=1}^{m+1} g_i^{M_i}$ and sends the result to V.

These algorithms can be integrated into PoW-based protocols as a substitute for the puzzle, for a lightweight procedure that offers similar security results.

Capítulo 3 Design and Implementation

This work revolved around the development and implementation of two lightweight and mathematically proven communication enforcing schemes, built with cryptographic primitives. These schemes require some computational power to find a given solution to a problem, but without the heavy computations required by regular Proof-of-Work protocols, like the one used on Bitcoin or Ethereum 1.0.

This way, the solutions explored in this chapter are expected to be viable replacements to the algorithms used in consensus protocols on the Blockchain platforms that still use Proof-of-Work-based mechanisms, in order to ease their implementations of applications on IoV networks without developers having to worry with resource constraints.

The chapter starts by defining the design of the base cryptographic schemes, followed by the their implementation on Ethereum and the definition of the network architecture in an IoV scenario.

3.1 Design of schemes

Following the published work of P. Golle *et al* [94], the two proposed communication enforcing schemes (CES), CBC-RSA and Two-root RSA can be used to enforce linear lower-grounds for communication and/or storage resources, helping prevent the "free rider" problem that affects P2P networks, where nodes act selfishly and do not help maintain the network by actively working on it.

Theoretically, CES schemes, while working just like any other digital signature protocol, ensure that a signer can only sign a message after exchanging at least as much data with the source of the message as would be required to send the message. This is enforced because a signer has to allow the source to either sign the message on its behalf or engage in a communication process with the same complexity of the message that is being transmitted.

The authors define a CES scheme as a triple of probabilistic polynomial-time algorithms:

- A key-generation algorithm G which outputs a private/public key pair (d, e) for every security parameter k;
- A signing algorithm Sig which given a message M and a private key d computes a signature

$$sig = Sig(M, d) \tag{3.4}$$

- A verification algorithm Ver s.t. if

$$(e,d) = G(k) \wedge sig = Sig(M,d) \Rightarrow Ver(M,sig,e) = 1 \tag{3.5}$$

such that for every probabilistic polynomial-time interactive algorithms of a signer P and a message source S,

- If for every private key d and message M, the protocol $(S(M), P(d))$ outputs a signature sig (3.4),
- and if, after a repeated interaction with P on polynomial-many messages $M_1, \dots, M_n$ of its choice, S cannot forge a signature on some $M \neq M_1, \dots, M_n$, except for probability negligible in k
- then the communication complexity of the $(P(M), S(d))$ protocol for messages $M \in \{0;\ 1\}_n$ is at least n.

Each transmitted message can be divided into n-blocks of equal size, to sign and create the final hash.

3.1.1 CBC-RSA definition

The algorithm applies Cypher-block chaining (CBC) with RSA keys, to preserve the size of the cipher, instead of having it grow with each new cryptographic cycle. Given a message M divided into n blocks, a private/public key pair (d,e), a product of two primes N and a hash function H, the CBC-RSA signature algorithm develops as follows:

1. Calculate C_1:

$$C_1 = H(M_1)^d \; mod \; N \tag{3.6}$$

2. Calculate C_i:

$$C_i = (C_{i-1} \oplus H(M_i)^d \; mod \; N, with \; i = 2, \dots, n \tag{3.7}$$

3. The resulting C_n is the CES signature.

And the verification algorithm goes backward by decrypting the signature with the public key:

1. Calculate each C_{i-1}, where:

$$C_{i-1} = (C_i^e \; mod \; N) \oplus H(M_i), with \; i = n, \dots, 2 \tag{3.8}$$

2. When C_1 is obtained, compare:

$$C_1^e \; mod \; N = H(M_1) \tag{3.9}$$

3.1.2 Two-root RSA definition

The algorithm uses the received message to create an n-degree polynomial function and finds two distinct roots for the generated problem, which are used later for verification purposes. Given a message M divided into n blocks, a product of two primes $N = pq$ and a full-domain length-preserving collision-resistant function H, the Two root RSA signature algorithm is structured as follows:

1. Choose a random vector IV, such that:

$$IV \overset{R}{\leftarrow} \mathbb{Z}_N^* \tag{3.10}$$

2. Compute C, where:

$$C = H(M_1, H(M_2, \dots H(M_n, H(IV, 0)) \dots) \tag{3.11}$$

3. Compute C_i, where:

$$C_i = H(C, M_{i+1})\ 0 \leq i < n \tag{3.12}$$

4. Define $P(x)$ as:

$$P(x) = x^n + C_i\, x^{n-1} + \dots + C_1 x + C_0 \tag{3.13}$$

5. Find two distinct roots $t_1,\ t_2 \in \mathbb{Z}_p$, and two distinct roots $s_1, s_2 \in \mathbb{Z}_q\ of\ P(x)$
6. Find $x_1, x_2 \in \mathbb{Z}_N$ satisfying:

$$x_1 \equiv t_1\ (mod\ p) \wedge x_1 \equiv s_1\ (mod\ q) \tag{3.14}$$

$$x_2 \equiv t_2 (mod\ p) \wedge x_2 \equiv s_2\ (mod\ q) \tag{3.15}$$

7. Return the CES signature $< IV, x_1, x_2 >$.

And the verification algorithm does the same, except for the process of finding roots, only building the polynomial function to test the roots as zeroes:

1. Compute $C, C_1, \dots, C_{n-1}$, and $P(x)$ from M.
2. Signature is accepted if

$$P(x_1) = 0 \wedge P(x_2) = 0 \tag{4.16}$$

3.2 Implementation

These algorithms were implemented with Golang [95], a very recent programming language developed by Google, and the core language of Ethereum and Hyperledger. They are two of the biggest developer-friendly Blockchain platforms that have seen usage outside of the economic and cryptocurrency transaction environment. While it was rather easy to learn the language, it lacked some of the necessary functions and there is not much discussion about the language on online forums.

3.2.1 CBC-RSA implementation

Algorithm 1 and Algorithm 2, showcased below, provide pseudo-code implementations of the CBC-RSA algorithm, according to the reasoning provided in the paper.

Algorithm 1: Generation of a signature based on the described CBC-RSA scheme.

```
Input: Message m
Output: Signature s

1.  blocks := divideMessage(m)
2.  //equation (3.6)
3.  encrypted := encryptRSA(hash(blocks[0]))
4.  //equation (3.7)
5.  for i = 1 to blocks.length:
6.  |   h := hash(blocks[i])
7.  |   x := xor(encrypted, h)
8.  └   encrypted := encryptRSA(x)
9.
10. return encrypted
```

Algorithm 2: Verification of a previously generated CBC-RSA signature.

```
Input: Signature s, Message m
Output: True if verification finds the signature uncorrupted, false otherwise

1.  blocks := divideMessage(m)
2.  //equation (3.8)
3.  for i = blocks.length-1 to 1:
4.  |   c := decryptRSA(s)
5.  |   h := hash(blocks[i])
6.  └   s := xor(c, h)
7.  //equation (3.9)
8.  c1 := decryptRSA(s)
9.  hm1 := hash(blocks[0])
10.
11. return compare(c1, hm1)
```

Golang implementation

Signature algorithm:

```go
func cbcrsa(byteMessages [][]byte) []byte {
    var hash = sha256.New()
    var encryptedValues = make([][]byte, len(byteMessages))
    //equation(3.6)
    hash.Write(byteMessages[0])
    h := hash.Sum(nil)
    encryptedValues[0] = encrypt(new(big.Int),
                                 new(big.Int).SetBytes(h)).Bytes()
    hash.Reset()

    return cbcChain(encryptedValues, byteMessages, hash)
}
```

```
func cbcChain(encryptedValues [][]byte, values [][]byte, hash hash.Has
h) []byte {
    var x = encryptedValues[0]
    //equation (3.7)
    for i := 1; i < len(values); i++ {
        hash.Write(values[i])
        xor := XOR(encryptedValues[i-1], hash.Sum(nil))
        hash.Reset()
        x = encrypt(new(big.Int), new(big.Int).SetBytes(xor)).Bytes()
        encryptedValues[i] = x
    }

    return x
}
```

The *cbcrsa* function receives the message divided into blocks, which in turn are arrays of bytes. With those blocks, this function creates C_1 with the first block, as specified in equation (4.6). After that, it proceeds to CB-chain the rest of the blocks with the first, to create the signature, according to equation (4.7). It cycles all the blocks and, in each cycle, hashes the value, applies XOR with the last hashed value, encrypts it with the RSA private key, and repeats the cycle to the next message block. In the end, it returns the final signature, which is the hash for the Blockchain block. In these algorithms, SHA256 is used to hash the blocks, since the block hash size for Ethereum is, usually, 32 bytes (i.e., 256 bits).

Verification algorithm:

```
func verifyCBCRSA(hashedValue []byte, messages [][]byte) bool {
    var x = hashedValue
    var hash = sha256.New()

    for i := 0; i < len(messages)-1; i++ {
        x = decrypt(new(big.Int), new(big.Int).SetBytes(x)).Bytes()
        hash.Write(messages[len(messages)-(i+1)])
        x = XOR(x, hash.Sum(nil))
        hash.Reset()
    }

    cmp := decrypt(new(big.Int), new(big.Int).SetBytes(x)).Bytes()
    hash.Write(messages[0])
    h := hash.Sum(nil)
    hash.Reset()

    return bytes.Compare(cmp, h) == 0
}
```

To verify the signature the verification algorithm requires the signature and the original message, divided in the same number of blocks as when it was used to create the signature. With the public key of the signer, the verification process starts by decrypting the signature, hashing the corresponding message to that segment, and applying an XOR operation between the decrypted data and that hashed message, to obtain the previous encrypted segment. This is repeated until it reaches C_1, where it decrypts it and compares the final result with the first message, hashed.

The encrypt and decrypt functions use the public and private keys of the miner to get the hashed data as follows:

```go
func encrypt(c *big.Int, m *big.Int) *big.Int {
    c.Exp(m, privateKey.D, publicKey.N)
    return c
}

func decrypt(c *big.Int, m *big.Int) *big.Int {
    e := big.NewInt(int64(publicKey.E))
    c.Exp(m, e, publicKey.N)
    return c
}
```

Problems and limitations

When working with the Golang programming language, Cipher-block chaining (CBC) algorithms can only be utilized with AES cipher keys, invoking the need to create a personal version of CBC that worked with RSA keys. This CBC mechanism was implemented by applying an XOR operation between the new hashed value and what had been hashed before, as seen in line 7 of both Algorithm 1 and Algorithm 2.

In the implementation, the algorithm was written exactly as described by the authors of the paper, without any more incidents or missing base code. However during the verification of the output, following the proposed verification algorithm and without any change to the original data or the signature, the decryption would output a value that was different from expected, at random. This only happened when the message was divided into more than 2 blocks and never for 2 or less. As such, the message was limited to a 2-block division, to test the algorithm without issue. This choice, however, still allows for a measure of the performance of the algorithm, but there were no tests on the influence of larger blocks of data on this division.

3.2.2 Two-root RSA implementation

Algorithm 3 and Algorithm 4 present the pseudo-code implementations of the algorithms for signing and verifying messages, respectively, with Two-root RSA.

Algorithm 3: Generation of a signature based on the described Two-root RSA scheme.

```
Input: Message m
Output: Signature <IV, zero_1, zero_2>

1. blocks := divideMessage(m)
2. while(true):
3. |  iv := randomVector() //equation (3.10)
```

```
4.      b := append(blocks, iv)
5.      c := hash(b) //equation (3.11)
6.      coeff := []
7.      //equation (3.12)
8.      for i = 0 to blocks.length:
9.          x := xor(c, blocks[i])
10.         h := hash(x)
11.         coeff[i] := float(h)
12.     coeff := coeff.append(1.0)
13.     //verify if $b^2-4ac > 0$, which is when it has 2 roots
14.     if(discriminant(coeff) > 0):
15.         zeroes := quadraticFormula(coeff)
16.         if(compare(zeroes[0], zeroes[1]) != 0):
17.             return iv, zeroes
```

Algorithm 4: Verification of a previously generated Two-root RSA signature.

Input: InitialVector IV, Float $zero_1$, Float $zero_2$, Message m
Output: True if verification finds that the zeroes are still the solution to the polynomial function, false otherwise

```
1.  blocks := divideMessage(m)
2.  b := append(blocks, iv)
3.  c := hash(b)
4.  coeff := []
5.
6.  for i = 0 to blocks.length:
7.      x := xor(c, blocks[i])
8.      h := hash(x)
9.      coeff[i] := float(h)
10.
11. coeff := coeff.append(1.0)
12. return isRoot(coeff, $zero_1$) and isRoot(coeff, $zero_2$)
```

Golang implementation

Signature algorithm:

```go
func tworootrsa(byteMessages [][]byte) ([]byte, []*big.Float) {
    var zeroes []*big.Float
    var initV []byte
    var hash = sha256.New()

    for len(zeroes) < 2 {
        initV = make([]byte, len(byteMessages[0]))
        //equation (3.10)
        inte, _ := rand.Int(rand.Reader, publicKey.N)
        initV = inte.Bytes()
        byteMessagesWIV := append(byteMessages, initV)
        //equation (3.11)
        result := hashStrings(byteMessagesWIV, hash)
        coefficients := make([]*big.Float, len(byteMessages)+1)
        //equation (3.12)
        for i := 0; i < len(byteMessages); i++ {
            hash.Write(bitwiseOperation(byteMessages[i], result))//XOR
            x := hash.Sum(nil)
            hash.Reset()
            coefficients[i] = new(big.Float).SetInt(big
                           .NewInt(int64(binary.BigEndian.Uint64(x))))
        }
        //equation (3.13)
        coefficients[len(byteMessages)] = New(1.0)
        //discriminant b²-ac
        quad_b := Pow(coefficients[1], 2.0)
        ac := Mul(coefficients[2], coefficients[0])
        timesfour := Mul(ac, big.NewFloat(4.0))
        b := Sub(quad_b, timesfour)

        if Lesser(New(0.0), b) {
            //Find roots
            var zero1, zero2 = getZeroes2ndDegree(coefficients[2],
                                     coefficients[1], coefficients[0])
            if zero1.Cmp(zero2) != 0 {
                zeroes = append(zeroes, zero1)
                zeroes = append(zeroes, zero2)
            }
        }
    }

    return initV, zeroes
}
```

Similar to *cbcrsa*, when calling *tworootrsa*, one has to pass the original message divided into blocks. Then, a random value byte vector (*initial vector*) is constructed, in order to create the value for C, as explained in (3.11), using the *hashStrings* function, which is a recursive function that hashes all of the blocks. C will be used to calculate $C_0, \dots, C_{n-1}$, which are the coefficients for the polynomial function. After that, since the function is a 2nd-degree function, it is possible to use the discriminant factor (3.20-22) to find if the function has two roots. If this is confirmed, the quadratic formula is applied to find the roots of the equation, which are saved, along with the *initial vector*, to be saved in the Blockchain block structure.

When implementing (3.12), because hash functions in Golang do not accept two different inputs, a bitwise operation (XOR) had to be applied to unify both values into one and to create a hash-able singular vector.

```go
func hashStrings(messages [][]byte, hash hash.Hash) []byte {

    return hashStringsAux(messages[0], messages[1:], hash)
}
//Recursive auxiliar function
func hashStringsAux(message []byte, messages [][]byte, hash hash.Hash)
                                                                 []byte {
    var length = len(messages)

    if length == 0 {
        hash.Write(message)
        shaValue := hash.Sum(nil)
        hash.Reset()

        return shaValue
    } else {
        var hashValue = hashStringsAux(messages[0], messages[1:], hash)

        hash.Write(bitwiseOperation(message, hashValue)) //XOR operation
        shaValue := hash.Sum(nil)
        hash.Reset()

        return shaValue
    }
}
```

This function hashes every message recursively intending to obtain C, which is needed to calculate the coefficients of the polynomial function.

Verification algorithm:

```go
func verifyTwoRoot(iv []byte, z1 *big.Float, z2 *big.Float,
                                                messages [][]byte) bool {
    hash := sha256.New()
    byteMessagesWIV := append(messages, iv)
    //equation (3.11)
    result := hashStrings(byteMessagesWIV, hash)
    coefficients := make([]*big.Float, len(messages)+1)
    //equation (3.12)
    for i := 0; i < len(messages); i++ {
        hash.Write(bitwiseOperation(messages[i], result)) //XOR operation
        x := hash.Sum(nil)
        hash.Reset()
        coefficients[i] = new(big.Float).SetInt(big
                                .NewInt(int64(binary.BigEndian.Uint64(x))))
    }
    //equation (3.13)
    coefficients[len(messages)] = big.NewFloat(1.0)
    //equation (3.16)
    b1 := checkZero(coefficients, z1)
    b2 := checkZero(coefficients, z2)

    return b1 && b2
}
```

The process to verify a Two-root RSA signature is quite simple, with some of the calculations having to be repeated from the signing process. This includes finding C and $C_0, \dots, C_{n-1}$. After this, the zeroes are used to calculate the value of the equation $P(x)$, with $x \in \{z_1, z_2\}$, to know if they are the zeroes of the equation.

Problems and limitations

There are no programmed functions to get the roots of n-degree polynomial functions with Golang. As such, a "man-made" solution was in need to be programmed to help find the roots of a function.

Given the theoretical definition of the roots in step 5, the algorithm needed to find $t_1, t_2 \in [0, p]$ and $s_1, s_2 \in [0, q]$. Since p, q and N are, practically, defined by the RSA key generator (see RSA private/public key structures), the size of these numbers is based on the size of the keys, which is set for 256 bits or 32 bytes. With numbers this big, the roots in Z_p and Z_q are, basically, the same, which means that it is only necessary to find x_1 and x_2, skipping step 5.

The first practical implementation of this algorithm first found possible roots by applying the Rational Zero Theorem [96], where an n-degree polynomial function

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 = 0 \tag{4.17}$$

with integer coefficients $a_0, a_1, \dots, a_n$.

If a_n and a_0 are not zero, then each rational solution

$$x = \frac{t}{v} \tag{4.18}$$

with t and v being coprime integers, satisfies the conditions that v a divisor of a_n and t a divisor or a_0.

However, this immediately brought up two problems: first, the time to find all the possible divisors of a_0 lasted minutes, even when the algorithm was optimized by only searching up to the root of a_0, by verifying both the current number and its square, as shown in Algorithm 5. This happens because a_0 is a really big number and there is no known way to find the divisors of a big number in Golang other than recurring to a brute strength approach.

Algorithm 5: Finder of all divisors of a given number

Input: Integer a
Output: List of all divisors of a

```
1. divisors := []
2. for i = 0 to sqrt(a):
3. |  if(mod(sqrt(a), i) == 0 and i*i != a):
    |   |   divisors := append(divisors, i)
    |   └   divisors := append(divisors, div(sqrt(a_0), i))
    |
    |   if(mod(sqrt(a), i) == 0 and i*i == a):
    └   └   divisors := append(divisors, i)

return divisors
```

The second problem was because a_n is always 1 and, as such, it only has itself as a divisor. Given that 1 is a universal divisor (every natural number can be, at least, divided by itself and 1), t and v can never be coprimes.

To tackle this problem while respecting what had been specified prior by the algorithm definition, the degree of the function was set at 2. This means the message is to be divided into two blocks and the calculations are to be made under a quadratic function. To get the solution to these polynomial functions, a quadratic formula implementation was necessary, since it had not been implemented in Golang for *big.Int* type structures. Using quadratic functions also helps to ease the search for functions that have enough roots or any root at all, due to the discriminant factor.

With $P(x)$ defined as:

$$P(x) = ax^2 + bx + c \tag{3.19}$$

and with $\{a, b, c\}$ being the coefficients of the quadratic function, if:

$$b^2 - 4ac > 0, P(x) \text{ has at two real solutions} \tag{3.20}$$

$$b^2 - 4ac = 0, P(x) \text{ has at one real solution} \tag{3.21}$$

$$b^2 - 4ac < 0, P(x) \text{ has no solution} \tag{3.22}$$

The two roots obtained on the signature are stored on a *big.Float* type structures. However, this type does not provide required mathematical functions, like power or root, which had to be implemented to obtain the signature and verify the results.

3.3 Ethereum

Ethereum is developed in a modular fashion, with each component design and implementation being the most independent from each other as possible. While this eases the introduction and implementation of new mechanisms for the Blockchain system, Ethereum was not made with the intent to create and reinvent core functionalities, such as consensus protocols, focusing instead on being a user-friendly smart contract development and insertion platform. Therefore, it lacks documentation on how to develop other solutions to what is already implemented by the community, making it hard to look up what has to be changed and/or updated to functionally insert a new working consensus protocol.

Figure 3.7 represents interactions between nodes in a network while they try to achieve consensus over a list of transactions. The implementations referred above are to be deployed on the mine and verifySeal functions presented in the scheme, as will be showcased below in the code snippets. These algorithms will substitute Ethash, the Proof-of-Work algorithm, utilized to achieve consensus, that is used at the moment in the main Ethereum network.

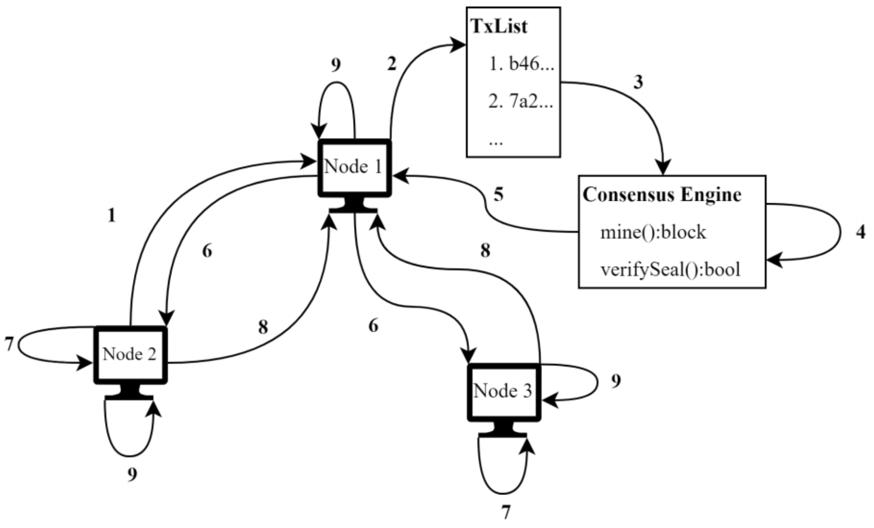


Figure 3.7 - Network consensus interactions with Ethereum.

1. *Node 2 sends a message to Node 1.*
2. *Node 1 adds the new message to its transaction list (TxList).*
3. *Node 1's Consensus Engine reads that transaction list.*
4. *Consensus Engine mines a new solution, either with CBC-RSA or Two-root RSA mechanisms.*
5. *Consensus Engine returns a sealed block.*
6. *Node 1 broadcasts the new solution to the network.*
7. *Nodes 2 and 3 verify the solution with their Consensus Engines.*
8. *Nodes 2 and 3 return the response to Node 1.*
9. *The block is added if the majority agrees, otherwise, it is discarded.*

As seen in Ethereum's Git repository [97], there is a large number of folders, each for each module of the Blockchain platform. However, for the implementation of a new consensus protocol, a programmer needs only to worry about the *consensus*, *eth*, and *core* folders and some of the files, as represented in Figure 3.8.

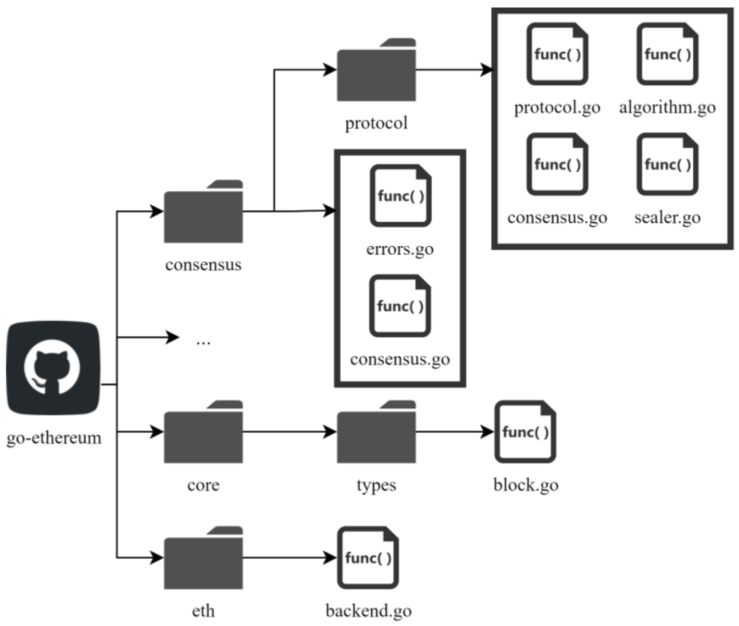


Figure 3.8 - Ethereum files necessary for consensus implementation.
While there are many other files inside Ethereum, only these matter when implementing a new consensus solution.

To implement a new consensus protocol, one must create four new files with all the functions of the given consensus '*Engine*' interface, on the '*consensus.go*' file. The file '*protocol.go*' holds the protocol structure, which is used to call the consensus methods and other general functions, such as getters. '*algorithm.go*' holds the core code for the protocol, the hashing, and verification functions. The '*sealer.go*' and '*consensus.go*' files hold the seal (hashing) and verification processes, respectively. Most of the functions on these files can remain the same as the current ones (e.g., in Ethash) unless the consensus protocol specifically needs to change those parts. In this case, it was only necessary to change the *mine* and *verifySeal* functions.

```go
// Engine is an algorithm agnostic consensus engine.
type Engine interface {
    // Author retrieves the Ethereum address of the account that minted
    // the given block, which may be different from the header's coinbase
    // if a consensus engine is based on signatures.
    Author(header *types.Header) (common.Address, error)

    // VerifyHeader checks whether a header conforms to the consensus
    // rules of a given engine. Verifying the seal may be done
    // optionally here, or explicitly via the VerifySeal method.
    VerifyHeader(chain ChainHeaderReader, header *types.Header,
      seal bool) error

    // VerifyHeaders is similar to VerifyHeader, but verifies a batch
    // of headers concurrently. The method returns a quit channel to
    // abort the operations and a results channel to retrieve the
    // async verifications (the order is that of the input slice).
    VerifyHeaders(chain ChainHeaderReader, headers []*types.Header,
      seals []bool) (chan<- struct{}, <-chan error)

    // VerifyUncles verifies that the given block's uncles conform to
    // the consensus rules of a given engine.
    VerifyUncles(chain ChainReader, block *types.Block) error

    // VerifySeal checks whether the crypto seal on a header is valid
    // according to the consensus rules of the given engine.
    VerifySeal(chain ChainHeaderReader, header *types.Header) error

    // Prepare initializes the consensus fields of a block header
    // according to the rules of a particular engine.
    Prepare(chain ChainHeaderReader, header *types.Header) error

    // Finalize runs any post-transaction state modifications
    // (e.g. block rewards)  but does not assemble the block.
    //
    Finalize(chain ChainHeaderReader, header *types.Header,
      state *state.StateDB, txs []*types.Transaction,
      uncles []*types.Header)
    // FinalizeAndAssemble runs any post-transaction state modifications
    // (e.g. block rewards) and assembles the final block.
    FinalizeAndAssemble(chain ChainHeaderReader, header *types.Header,
      state *state.StateDB, txs []*types.Transaction,
      uncles []*types.Header, receipts []*types.Receipt)
      (*types.Block, error)
    // Seal generates a new sealing request for the given input block and
    // pushes the result into the given channel.
    Seal(chain ChainHeaderReader, block *types.Block,
      results chan<- *types.Block, stop <-chan struct{}) error

    // SealHash returns the hash of a block prior to it being sealed.
    SealHash(header *types.Header) common.Hash

    // CalcDifficulty is the difficulty adjustment algorithm.
    // It returns the difficulty that a new block should have.
    CalcDifficulty(chain ChainHeaderReader, time uint64,
      parent *types.Header) *big.Int

    // APIs returns the RPC APIs this consensus engine provides.
    APIs(chain ChainHeaderReader) []rpc.API

    // Close terminates any background threads maintained by the
    // consensus engine.
    Close() error
}
```

mine function

The *mine* function receives a *block structure*, an *id*, and a *seed*. This function will create the signatures by following the signing algorithms of the consensus engine. For Ethash, it will use the built dataset to find a <*hash*, *nonce*> pair that corresponds to a certain rule, based on the set difficulty.

Ethash *mine* function core:
Ethash will compute the PoW value of a given nonce. The PoW is compared to a given rule pattern for a target value and, if it matches, the value is stored in the block header, along with the nonce. If the value does not match the rule pattern, the nonce is incremented and the process repeats until a valid PoW value is found.

```
// We don't have to update hash rate on every nonce, so update after
// 2^X nonces
attempts++
if (attempts % (1 << 15)) == 0 {
    ethash.hashrate.Mark(attempts)
    attempts = 0
}
```

```
// Compute the PoW value of this nonce
digest, result := hashimotoFull(dataset.dataset, hash, nonce)
if new(big.Int).SetBytes(result).Cmp(target) <= 0 {
    // Correct nonce found, create a new header with it
    header = types.CopyHeader(header)
    header.Nonce = types.EncodeNonce(nonce)
    header.MixDigest = common.BytesToHash(digest)
    // Seal and return a block (if still needed)
    select {
    case found <- block.WithSeal(header):
        logger.Trace("Ethash nonce found and reported",
        "attempts", nonce-seed, "nonce", nonce)
    case <-abort:
        logger.Trace("Ethash nonce found but discarded",
            "attempts", nonce-seed, "nonce", nonce)
    }
}
nonce++
```

CBC-RSA *mine* function core:
CBC-RSA algorithm will hash the header information and place said hash on the header, along with the public key information, for verification. The public key information has to be inserted in the block since there is no other known way to store RSA keys and fetch them later for verification on Ethereum. While this reduces privacy, it also reduces the time required for verification, since the verifier node does not need to fetch this data somewhere else.

```
// Compute cbc-rsa hash
digest := cbcrsa.cbcrsa(splitArrayNParts(hash,
    cbcrsa.conf.Divisions))
// Found hash, create a new header with it
header = types.CopyHeader(header)
header.MixDigest = common.BytesToHash(digest)
header.PubKeyN = cbcrsa.getPubKey().N
header.PubKeyE = big.NewInt(int64(cbcrsa.getPubKey().E))

// Seal and return a block (if still needed)
select {
case found <- block.WithSeal(header):
    logger.Trace("CBCRSA hash found and reported")
case <-abort:
    logger.Trace("CBCRSA hash found but discarded")
}
```

Two-root *mine* function core:
Two-root RSA will create a polynomial equation based on the information stored in the given header and a random *initial vector*. The algorithm will then try to find two valid roots for this

equation, which, when found, are stored in the header along with the aforementioned *initial vector*.

```go
// Compute two-root rsa roots
digest, roots := tworoot.tworootrsa(splitArrayNParts(hash,
    tworoot.conf.Degree))
// Two roots found, create a new header with them and IV
header = types.CopyHeader(header)
header.MixDigest = common.BytesToHash(digest)
header.Roots = roots

// Seal and return a block (if still needed)
select {
case found <- block.WithSeal(header):
    logger.Trace("Two-root RSA solution found and reported")
case <-abort:
    logger.Trace("Two-root RSA solution found but discarded")
}
```

For the Ethereum implementation, the message *M* that is used in the algorithms is a hash of the block header obtained with the *SealHash* function presented below.

```go
// SealHash returns the hash of a block prior to it being sealed.
func (tworoot *TWOROOT) SealHash(header *types.Header) (hash common.Hash)
 {
    hasher := sha3.NewLegacyKeccak256()

    rlp.Encode(hasher, []interface{}{
        header.ParentHash,
        header.UncleHash,
        header.Coinbase,
        header.Root,
        header.TxHash,
        header.ReceiptHash,
        header.Bloom,
        header.Difficulty,
        header.Number,
        header.GasLimit,
        header.GasUsed,
        header.Time,
        header.Extra,
    })
    hasher.Sum(hash[:0])
    return hash
}
```

verifySeal function

The *verifySeal* function receives a header and verifies if the hash it holds is correct for the information it has in it. For Ethash, it, again, recurs to the dataset, which is obtained with the block number and uses the hash and nonce to verify if the obtained <*hash, nonce*> pair corresponds to the <*hash, nonce*> pair received through the given header. As for the lightweight approaches, the process calls the verification algorithms, passing a header with the *mixHash* being the chained signature and the *pubKey* values being the public key of the signer or a header with the initial vector on the *mixHash* slot and the roots for the solution.

Ethash *verifySeal* function core:
The ethash algorithm fetches the dataset used when the nonce was found and calculates the PoW value, compares it to the current value on the header, to check for tampering, and then compares it to the rule pattern used when it was calculated the first time.

```
digest, result = hashimotoFull(dataset.dataset, Ethash
    .SealHash(header).Bytes(), header.Nonce.Uint64())

// Verify the calculated values against the ones provided in the header
if !bytes.Equal(header.MixDigest[:], digest) {
    return errInvalidMixDigest
}

target := new(big.Int).Div(two256, header.Difficulty)
if new(big.Int).SetBytes(result).Cmp(target) > 0 {
    return errInvalidPoW
}
return nil
```

CBC-RSA *verifySeal* function core:
CBC-RSA verification under the *verifySeal* function is pretty simple, since the verification algorithm, showcased in Chapter 3.2.1., already returns a Boolean value that is true if the value is correct and false if it is not.

```
digest = cbcrsa.verifyCBCRSA(header.MixDigest[:], splitArrayNParts(hash,
    cbcrsa.conf.Divisions), header.PubKeyN, header.PubKeyE)

// Verify the calculated values against the ones provided in the header
if !digest {
    return errInvalidMixDigest
}
```

Two-root RSA *verifySeal* function core:
The verification process is also simple, following the same reasoning that was explained with CBC-RSA, with the verification algorithm returning only a Boolean value that is true if the given numbers are roots of the polynomial equation created from the hashed header + the IV vector.

```
digest = verifyTwoRoot(header.MixDigest.Bytes(), header.Roots[0],
    header.Roots[1], splitArrayNParts(hash, tworoot.conf.Degree))

// Verify the calculated values against the ones provided in the header
if !digest {
    return errInvalidMixDigest
}
```

If the new protocol needs to change the block structure (like Two-root RSA needed, due to the output being a trio of values), this must be changed in the '*core/blocks.go*' file.

```
// Header represents a block header in the Ethereum blockchain.
type Header struct {
    ParentHash  common.Hash
    UncleHash   common.Hash
    Coinbase    common.Address
    Root        common.Hash
    TxHash      common.Hash
    ReceiptHash common.Hash
    Bloom       Bloom
    Difficulty  *big.Int
    Number      *big.Int
    GasLimit    uint64
    GasUsed     uint64
    Time        uint64
    Extra       []byte
    MixDigest   common.Hash
    Nonce       BlockNonce
    Roots       []*big.Float
    PubKeyN     *big.Int
    PubKeyE     *big.Int
}
```

For CBC-RSA, the *PubKeyN* and *PubKeyE* values are necessary for verification purposes, with the signature is stored in the *mixDigest* value, while for Two-root RSA the IV is stored in the *mixDigest* and the roots are stored as an array of **big.Float*.

Finally, in the file '*backend.go*', in the *eth* folder, the function '*CreateConsensusEngine*' is required to include a new conditional variable for the new consensus protocol, to create it during runtime.

```
// CreateConsensusEngine creates the required type of consensus engine
// instance for an Ethereum service
func CreateConsensusEngine(stack *node.Node,
      chainConfig *params.ChainConfig, config *ethash.Config,
      notify []string, noverify bool, db ethdb.Database)
      consensus.Engine {
    // My Consensus algorithms
    if chainConfig.CBCRSA != nil {
        log.Warn("CBC-RSA is configured as consensus engine")
        return cbcrsa.New(chainConfig.CBCRSA, db)
    }

    if chainConfig.TwoRootRSA != nil {
        log.Warn("Two-root RSA is configured as consensus engine")
        return tworoot.New(chainConfig.TwoRootRSA, db)
    }
    // If proof-of-authority is requested, set it up
    if chainConfig.Clique != nil {
        return clique.New(chainConfig.Clique, db)
    }
    // Otherwise assume regular proof-of-work
    switch config.PowMode {
    case ethash.ModeFake:
        log.Warn("Ethash used in fake mode")
        return ethash.NewFaker()
    case ethash.ModeTest:
        log.Warn("Ethash used in test mode")
        return ethash.NewTester(nil, noverify)
    case ethash.ModeShared:
        log.Warn("Ethash used in shared mode")
        return ethash.NewShared()
    default:
        engine := ethash.New(ethash.Config{...}, notify, noverify)
        engine.SetThreads(-1) // Disable CPU mining
        return engine
    }
}
```

This creates the protocol object that is going to be used by the *geth* process (Ethereum's main CLI client, used to initiate a new node and the network) to call the consensus functions during runtime.

Ethereum's block structure is showcased in Figure 3.9, but it can be subject to change if the consensus protocol requires a new field to be added, as is the case for the new lightweight approaches. In this implementation of the algorithms, a hashed version of the header's block is passed as the input Message M, which is then divided into a given number of blocks, as specified before, and then processed following the algorithm's procedures.

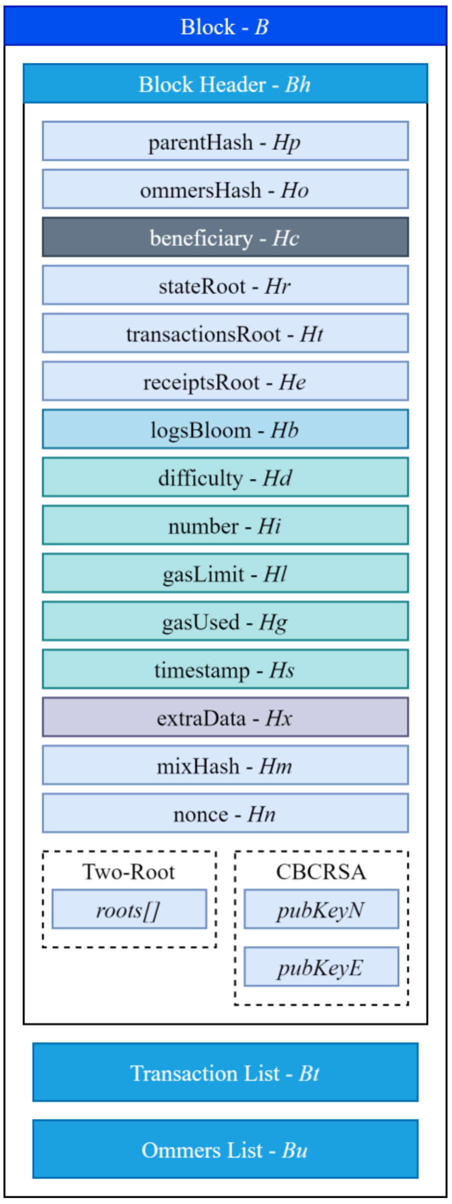


Figure 3.9 - Ethereum block structure.
With the implementation of CBC-RSA and Two-root RSA, two new fields had to be added: Public Key Pk and Zeroes Z_0. The Initial Vector Iv from Two-Root was placed on the mixHash field of the block.

Ethereum also holds *puppeth*, a CLI wizard that is used mainly to generate genesis configuration files, making it a lot easier to create such files, to later start the P2P network. Under the *cmd/puppeth* folder, in the '*wizard_genesis.go*' file, a developer can add their protocols to the already existing ones by adding more options for the creator to choose for their genesis.

```
// makeGenesis creates a new genesis struct based on some user input.
func (w *wizard) makeGenesis() {
    // Construct a default genesis block
    genesis := &core.Genesis{
        ...
    }
    // Figure out which consensus engine to choose
    fmt.Println()
    fmt.Println("Which consensus engine to use? (default = clique)")
    fmt.Println(" 1. Ethash - proof-of-work")
    fmt.Println(" 2. Clique - proof-of-authority")
    fmt.Println(" 3. CBC-RSA - lightweight proof-of-work")
    fmt.Println(" 4. Two-root RSA - lightweight proof-of-work")

    choice := w.read()
    switch {
    case choice == "1":
        // In case of ethash, we're pretty much done
         ...
    case choice == "2":
        // In the case of clique, configure the consensus parameters
         ...         }
    case choice == "3":
        // In case of CBC-RSA, set the number of divisions
        genesis.Config.CBCRSA = &params.CBCRSAConfig{Divisions: 2}
    case choice == "4":
        // In case of Two-root RSA, set the degree of the polynomial
        // function
        genesis.Config.TwoRootRSA = &params.TwoRootRSAConfig{Degree: 2}
    default:
        log.Crit("Invalid consensus engine choice", "choice", choice)
    }
    ...
}
```

Ethereum's fork policy allows for more than one chain to be active at a time, but, over time, the longest one (with more summed "work" overall) is selected unilaterally as the main chain. There is also the definition of uncle blocks (blocks that were created and proposed at the same time as the chosen block for the main chain) and their creators also receive some rewards for the work done.

3.4 Network Architecture

In the context of IoV, the network nodes are all the devices that participate in the exchange of data, like on-board computers on cars, traffic light sensors, speed radars or other sensors present in an area. This communication can happen due to a variety of factors (e.g., video or audio streaming, file transfer, information sharing, …) and in various fashions (e.g., continuous, event-based, time-based, …). In these networks, RSUs could be used as the main minters for blocks, while the resource-constrained devices would carry only the responsibilities that come with the communication itself. However, this also brings out some discussion on the benefits and problems with centralizing this responsibility to RSUs. In a fully decentralized scenario, the devices themselves are the miners and, in those cases, the lightweight algorithms are mandatory for a fully functionable and efficient Blockhain network.

When an event is detected (such as an accident or heavy traffic congestion), the vehicle that detected such event will broadcast a message to the nearby vehicles, informing them. Those receiving the message process it and add it to the block that is currently being mined, which is then hashed according to the specified CES protocols. The receiver asks the nearby vehicles for validation and, if confirmed by a majority, the block is added to the local chain and all the chains of the nearby vehicles. In an idyllic scenario, the validation process would be done by the closest RSU unit, and the chain would be built in a cloud service, in order to reduce the weight placed upon the resource-restrained devices. This last network topography and respective interactions are represented in Figure 3.10.

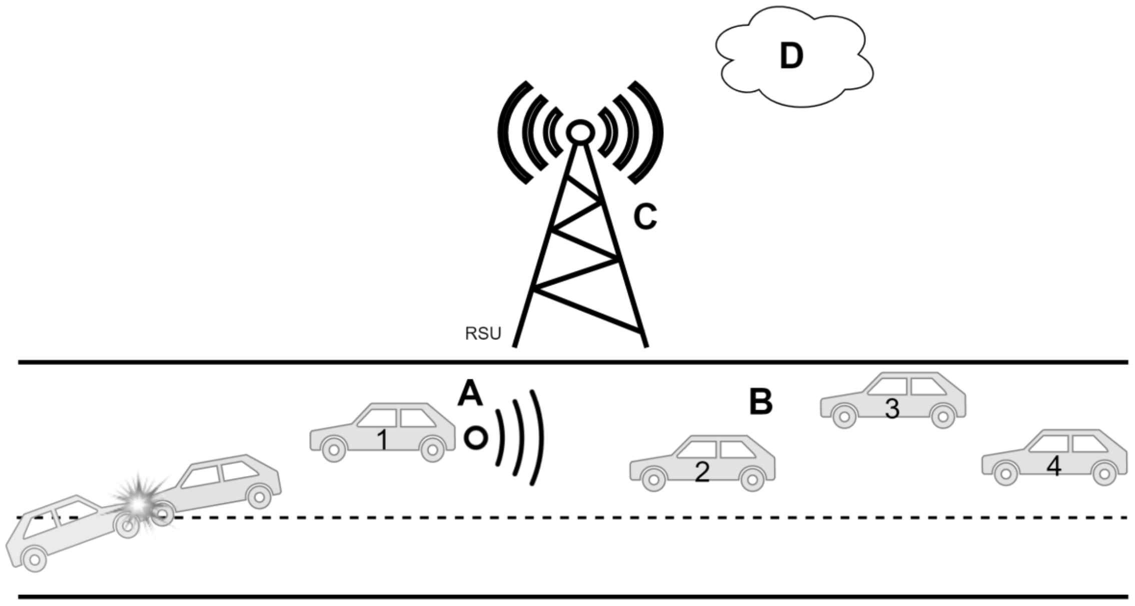


Figure 3.10 - Ideal network scenario.
A: Vehicle 1 detects the accident and broadcasts the event to the nearby nodes. B: The receiving nodes receive the information, create and hash the block, and send it to the nearest RSU. C: The RSU sends the block to all nearby nodes for verification, waits for >50% confirmations. D: After the block is verified and confirmed by a majority of the nodes, the block is sent to the chain, in the cloud. This format decouples the chain storage to the cloud, which lowers the requirements for the devices, reduces energy consumption and device complexity.

Capítulo 4   Performance Evaluation

Following the implementation and integration of the solutions on Ethereum, and the design of the network architecture, it was only necessary to test these solutions on a simulation of said presented network.

4.1 Simulation Model

This section presents the utility and configurations of the tools required for the simulations that were involved with testing and evaluating the performance of the designed algorithms in a real-case scenario.

4.1.1 Network Simulator

For the evaluation, a scenario was simulated in the Veins network simulator, which utilizes SUMO for traffic generation and OMNET++ for communication pattern simulation. In this scenario, vehicles are continuously introduced on the map and follow a certain path. When an event is detected (e.g., an accident, that makes the vehicle pause for some time), a message is broadcasted to all nearby nodes warning of such event and they change their route, if possible. After the retrieval of the messages sent (i.e., the sender and the receiver), this information was then applied onto the Blockchain to simulate close-to-real-life behavior, since there is not a known way to automate the requests from the simulation to the chain.

The simulation time on Veins was 200s with an update interval of 1s. The 13 simulated vehicles drove in an map of Erlangen with an area of 2.5 x 2.5 km. Communications were set to have a transmission rate of 6Mbps and a transmission power of 20mW, with 80bit packet header length. Every 10th node, starting from node 0, would detect one accident 73 seconds after being spawned in the map. Node mobility was traced by SUMO. The configuration parameters for the simulation on Veins is presented in Table 4.5.

Table 4.5 - Veins simulation configuration parameters

Parameter	Value
sim-time-limit	200s

playgroundSizeX	2500m
playgroundSizeY	2500m
playgroundSizeZ	50m
manager.updateInterval	1s
manager.numVehicles	13
nic.mac1609_4.txPower	20mW
nic.mac1609_4.bitrate	6Mbps
nic.phy80211p.minPowerLevel	-110dBm
node[*].nic.phy80211p.antennaOffsetY	0 m
node[*].nic.phy80211p.antennaOffsetZ	1.895 m
node[*].appl.headerLength	"80 bit"
node[*].appl.beaconInterval	1s
*node[*0].veinsmobility.accidentCount*	1
*node[*0].veinsmobility.accidentStart*	73s

4.1.2 Ethereum

To run this scenario, Ethereum accounts were created equal to the number of vehicles of the simulation (in this case, 13) that had sent some messages in the simulation. Each message exchange recorded in the network simulation was transformed into a 1 Wei (Ethereum’s lowest currency value) transaction from the sender account (the vehicle that sent the message) to the receiver account (the vehicle that received said message).

Each node was initialized with the respective consensus protocol, account, and some starting currency (e.g., 0xffffff, which would be enough for the required transactions gas costs) and connections were established between them, using their network IDs and the *addPeer(nodeID []byte)* function provided by the *admin* API. CBC-RSA and Two-root RSA only require the networkId, in this case 4000, and their parameter on the configuration file, for the simulation. Ethash also requires gasLimit and difficulty, for later calculations, which where, respectively, "0x47b760" and "0x80000". The consensus protocol configuration (which is defined in the genesis block file) is represented in Table 4.6.

Table 4.6 - Ethereum genesis block configuration file parameters

Parameter	**Value**
chainId	4000

ethash OR cbcrsa OR tworoot	{*specific config params*}
gasLimit	"0x47b760"
difficulty	"0x80000"

To measure the quality of algorithms, through their performance, Ethereum was made to register the time it took to hash and the time it took to verify that hash on a block, in microseconds. These values were then processed to calculate the average, in microseconds per block.

4.2 Results

The following data was gathered in two devices: a regular computer running Linux, with 8GB of RAM and two cores, and a Raspberry Pi, running a 64-bit ARM OS, with 4GB of RAM. The real-life scenario with simulated requests ran on the regular computer, due to the number of accounts that needed to run simultaneously, using 13 mining nodes exchanging 200 transactions, based on the data acquired on Veins.

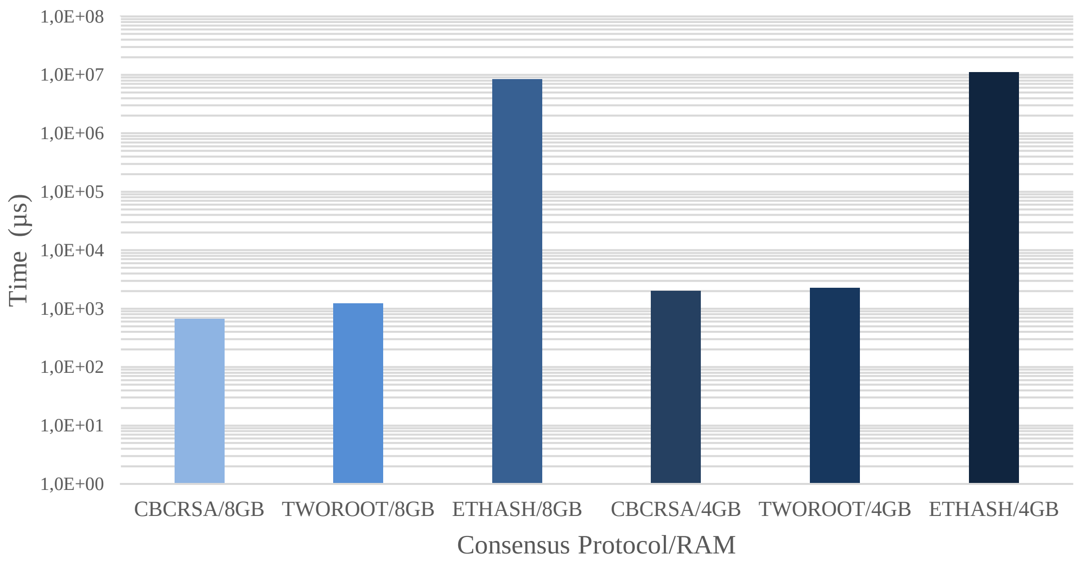


Figure 4.11 - Comparison of average times for the generation of block hashes for the various algorithms. Ethash is the main PoW-based consensus protocol used by Ethereum. The time axis had to be scaled to log_{10} to fit all values, or the values for CBC-RSA and Two-root RSA would be two fine lines on the bottom of the graph.

In Figure 4.11, the data shows that the proposed lightweight algorithms are up to 12000x times faster than Ethash in a regular computer and up to 5000x times faster in a device with lower computational power. Ethash also has a considerably slower mining initiation process, since the miner nodes need to first build the DAG in memory which takes up to 5 and a half minutes with 4GB of RAM. The DAG also has to be updated every 3840000 blocks, being calculated while the blocks are mined, which reduces the computational capacity for the mining itself. The DAG also increases its size when it updates, which creates another barrier for Ethash in these limited devices due to storage shortages.

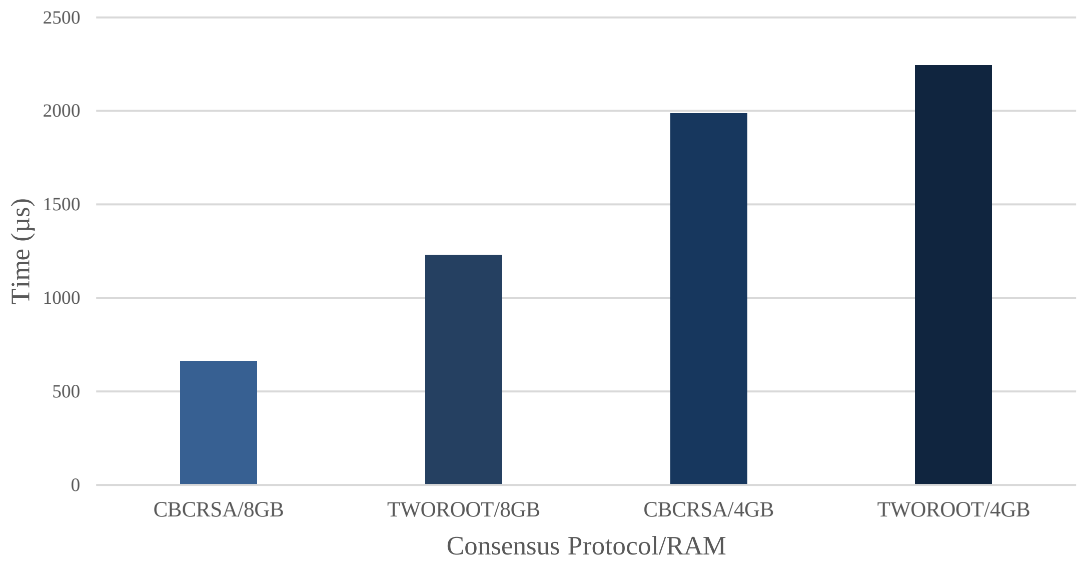


Figure 4.12 - Comparison of average times for the generation of block hashes for the lightweight algorithms. CBC-RSA is faster than Two-root RSA, but the difference is not huge, as seen in Fig. 5. However, this difference could deepen with proper optimization.

The data in Figure 4.12 shows that CBC-RSA is somewhat faster than the Two-root RSA approach. This happens because CBC-RSA only hashes and chains the byte values, while Two-root requires more complex mathematical calculations and root finding, which takes longer to process. This difference could, eventually and theoretically, increase with the degree of the polynomial function, if the algorithm was be optimized to deal with such constructions. A time increase with lower RAM is expected since the latter reduces the computational power of the device, increasing hash production times, but the average times still do not surpass 2500µs, a really small time value, which results in an enormous increase in output of block hashes when compared to the Ethash variation.

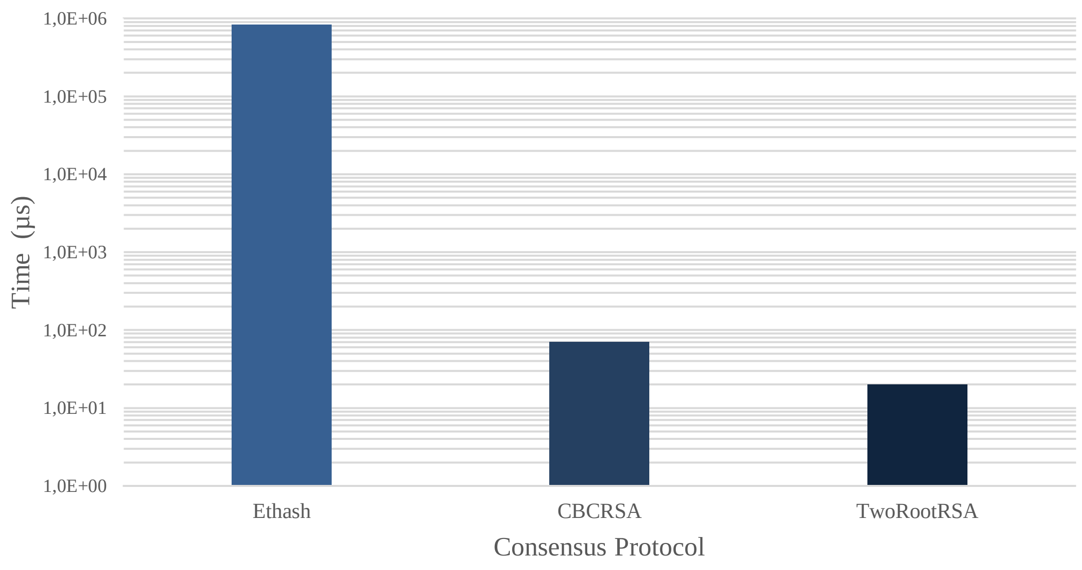


Figure 4.13 - Comparison of average times for the verification of created block hashes for the various algorithms. Ethash takes several thousands of times longer to verify a hash than the proposed lightweight algorithms and a scale of log$_{10}$ had to be used, again, to be able to compare the performance of the algorithms.

Figure 4.13 shows that the Ethash verification protocol, while faster than the hash creation mechanism, still takes longer than the proposed lightweight mechanisms. This is an important measure since it fastens the communications and processing of the blocks to be added to the chain. The proposed verification mechanisms are, on average, 12000x (CBC-RSA) and 40000x (Two-root) faster than the Ethash approach. Two-root RSA is faster than CBC-RSA because the verification mechanism does not require any type of complex calculations or hashing processes, being an easy math problem to check the truth of the given solutions, while CBC-RSA has to de-hash the given *mixHash* in order to verify it.

4.3 Technological limitations

While working with these technologies to obtain results and have a measure of the quality of the work done, some limitations were encountered due to a lack of information on the technology or problems that appeared due to the behavior of the technology itself not being the most ideal for the problem in question. These limitations slowed down the process of testing and quality data generation significantly and are described in the sections below.

4.3.1 Network simulator

Veins, while there is some documentation on its website, is not used but by a specific niche of researchers and, therefore, there is not much discussion online about how to use its tools, including SUMO. The main source of help is video tutorials on the internet. However, most of these are specific on their objective, which is not always the same as the one necessary at a given time. Limiting the number of vehicles on Veins is also very confusing, since the given value does not limit exactly the number of vehicles being simulated at a given time in the simulation, but the rate of creation of new vehicles in the simulation, i.e., in SUMO.

The code for Veins, OMNET++, and SUMO is also very difficult to understand, due to the lack of documentation and comments that explain how every module works and its usage in the simulator.

4.3.2 Ethereum

There were some problems with Ethereum, the biggest being the lack of documentation on how to implement new protocols, which required some trial and error. Secondly, testing Ethereum on the Raspberry Pi required the installation of a beta 64-bit OS, which required some investigation on the software offered by the Raspberry Pi organization.

During testing, the nodes would have memory problems and crash and had to be brought up again manually, in a quick fashion, to participate in the network and be up-to-date with all the exchanged data While gathering data for Ethash, due to the number of accounts running at the same time, the DAG construction, necessary for the nodes to start the mining process, took well over 40minutes, in good runs.

For CBC-RSA and Two-root RSA hashing tests, the fast sync option was making nodes drop the very first few blocks they got, which lead to problems later on because they would receive the "children" of those blocks and when verifying the header history, they would not find the parent hash. That created a ripple effect which led to errors and nodes dropping other nodes as trustful, cutting communications, that had to be brought up again.

Due to how fast the hashing process is, the number of blocks send through the network was so high that the nodes could not verify them correctly. Block creation had to be slowed down to 1 second per block, to obtain verification times. While this would not be a problem, if the mining only occurred, when necessary, which was the ideal scenario, with the existing Ethereum architecture and functionality, the usage of algorithms purely leads to these blockades.

Lastly, the transactions had to be inserted manually through *curl*, since there is no automatic way to insert transactions on an Ethereum network. In addition, the transactions had to be sent with some interval of time between them, or they would all be minted onto the same block, which could negate the communication enforcing property. There is also a problem with how transactions are shared in the network: they are broadcasted to all nodes, so all nodes mint their blocks with transactions that they might not participate in. This also entails that the block that is chosen for the global shared chain is not the one that has the transaction that would affirm the communication enforcement property.

Capítulo 5 Conclusion

This work presents two new lightweight solutions to tackle the energy and computation limitations of the devices that proliferate IoV networks. These approaches follow mathematically-proven cryptographic primitives-based schemes that ensure the needed security properties for communication while preventing free-riding with a communication enforcing property linked to them.

Chapter 2 summarizes the state-of-the-art for IoV and Blockchain technologies, including possible application areas and underlying technologies, with conclusions on why the latter's integration is so desired on the former due to the properties of both. Then, a listing of applications of Blockchain-based solutions in IoV is explored, where Blockchain is used to solve security problems in data sharing and/or storage in the context of IoV. Afterward, we review some of the most popular network simulators for IoV scenarios and some of the most popular Blockchain platforms available. The Internet of Vehicles has a lot of potential in improving driving experiences and infrastructure in the foreseeable future. However, it also requires development in ensuring security for its users and service providers. Blockchain has been proven to be a viable solution to cater to the security needs of IoV systems. Even so, Blockchain still needs improvements to excel in these environments.

In Chapter 3, the design and implementation of the two lightweight approaches are explained in detail. Firstly, the definition of a CES is explained and then, mathematical proof for the two lightweight CES algorithms (i.e., CBC-RSA and Two-root RSA) is given, which also provides instructions on how to build said schemes for general cases. Later on in the Chapter, the implementation with Golang and the integration of the algorithms on Ethereum, including limitations encountered during these processes. This also includes commentary on how the Ethereum project's modules are structured, which require changes when integrating a new consensus protocol, and what changes were necessary to implement the two lightweight approaches, with code samples that showcase the actual practical changes and what are they for.

Given the results gathered, exposed, and explained in Chapter 4, it is possible to conclude the proposed work is a viable solution to the resource problems that affect traditional Blockchain-based solutions for IoV networks and can be applied in every type of Blockchain for this purpose since the algorithms are decoupled from the development environment. The protocols can be implemented in any Blockchain system that supports Proof-of-Work protocols and is built in a programming language that provides the same cryptographic functions and similar data types. The proposed algorithms have also proven to be faster and consume less time and less energy to provide the necessary block hashes to achieve distributed consensus, along with the aforementioned free-riding prevention.

5.1 Future work

Regarding future work for the algorithms, a generalization process is one of the paths to follow, for CBC-RSA to tolerate more than 2 message block divisions and for Two-root RSA to be able to handle *n*-degree polynomial functions. For CBC-RSA, an in-depth investigation on how Golang does bytes calculations would be interesting, focusing on how those could lead to the verification errors found when dividing the message into more than 2 blocks. Two-root RSA could be implemented with the full-force method to find roots presented in Algorithm 5, utilizing a multithreaded approach that could ease the power consumption and duration of the root-finding algorithm. It would also be interesting to investigate how the increase in divisions of the original message affects the efficiency and security of the protocols, either positively or negatively, or even at all.

Two-root RSA integration also had some problems in the mining process, where it would stop after mining a random number of blocks and the process had to be restarted in order to mine more blocks, requiring some attention on this matter, since the source of this problem was not found.

With the goal to further reduce the energy and power consumption, Ethereum could be changed to drop the continuous mining process, only mining when a transaction or a list of transactions arrives at the device, or even periodically. This approach also helps with possible storage problems, since the chain grows larger even thousands of times faster with the lightweight approaches and while the DAG is not a problem, a large blockchain structure stored in the memory of limited devices could be problematic.

Another possible change for Ethereum would be the removal of the property of global knowledge of transactions that are exchanged in the network between devices, making the details only known by the devices that are required to have knowledge of those transactions, which could possibly be achieved by hashing and signing the transactions before adding them to the block. A transformation for cloud-based storage and RSU verification, to remove those responsibilities from the devices is an interesting path to follow and while there is some research about the subject, there is nothing for Ethereum.

Bibliography

[1] E. C. Eze, S. Zhang, and E. Liu, “Vehicular ad hoc networks (VANETs): Current state, challenges, potentials and way forward,” 2014, doi: 10.1109/IConAC.2014.6935482.

[2] A. Singh, L. Gaba, and A. Sharma, “Internet of Vehicles: Proposed Architecture, Network Models, Open Issues and Challenges,” *Proc. - 2019 Amity Int. Conf. Artif. Intell. AICAI 2019*, pp. 632–636, 2019, doi: 10.1109/AICAI.2019.8701312.

[3] E. Benalia, S. Bitam, and A. Mellouk, “Data dissemination for Internet of vehicle based on 5G communications: A survey,” *Trans. Emerg. Telecommun. Technol.*, vol. 31, no. 5, 2020, doi: 10.1002/ett.3881.

[4] T. Zhang, “Cisco Confidential Challenges and Opportunities.” 2015, [Online]. Available: https://site.ieee.org/denver-com/files/2016/02/IoV-Security-Challenges-and-Opportunities-zhang.pdf.

[5] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System | Satoshi Nakamoto Institute,” 2008.

[6] P. Álvares, L. Silva, and N. Magaia, “Blockchain-Based Solutions for UAV-Assisted Connected Vehicle Networks in Smart Cities: A Review, Open Issues, and Future Perspectives,” *Telecom*, vol. 2, no. 1, pp. 108–140, 2021, doi: 10.3390/telecom2010008.

[7] D. J. Skiba, R. James, and others, “The Internet of Things: A study in Hype, Reality, Disruption, and Growth,” *Raymond James US Research, Technology & Communications, Industry Report*, vol. 34, no. 1. pp. 63–4, 2014, [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/23586210.

[8] G. Davis, “2020: Life with 50 billion connected devices,” pp. 1–1, 2018, doi: 10.1109/icce.2018.8326056.

[9] O. Kaiwartya *et al.*, “Internet of Vehicles: Motivation, Layered Architecture, Network Model, Challenges, and Future Aspects,” *IEEE Access*, vol. 4, pp. 5356–5373, 2016, doi: 10.1109/ACCESS.2016.2603219.

[10] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibanez, “Internet of Vehicles: Architecture, Protocols, and Security,” *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3701–3709, 2018, doi: 10.1109/JIOT.2017.2690902.

[11] A. Islam, M. T. Hossan, and Y. M. Jang, “Introduction of optical camera communication for Internet of vehicles (IoV),” *Int. Conf. Ubiquitous Futur. Networks, ICUFN*, pp. 122–125, 2017, doi: 10.1109/ICUFN.2017.7993760.

[12] F. Yang, S. Wang, J. Li, Z. Liu, and Q. Sun, “An overview of Internet of Vehicles,” *China Commun.*, vol. 11, no. 10, pp. 1–15, 2014, doi: 10.1109/CC.2014.6969789.

[13] W. G. Meeting, “White Paper of Internet of Vehicles (IoV),” *50th Telecommun. Inf. Work. Gr.*, no. October, 2014.

[14] L. Nanjie, “Internet of Vehicles: Your next connection,” *Huawei WinWin*, pp. 23–28, 2011, [Online]. Available: http://www.huawei.com/en/about-huawei/publications/winwin-magazine/hw-110836.htm.

[15] F. Bonomi, C. Fellow, V. President, M. Others, A. Architecture, and C. Systems, “The Smart and Connected Vehicle and the Internet of Things,” 2013.

[16] W. Feng, Y. Li, D. Jin, L. Su, and S. Chen, “Millimetre-wave backhaul for 5G networks: Challenges and solutions,” *Sensors (Switzerland)*, 2016, doi: 10.3390/s16060892.

[17] W. Wu, Z. Yang, and K. Li, “Internet of Vehicles and applications,” in *Internet of Things: Principles and Paradigms*, 2016.

[18] C. H. Lee, C. M. Huang, C. C. Yang, and T. H. Wang, “A cooperative video streaming system over the integrated cellular and DSRC networks,” *IEEE Veh. Technol. Conf.*, pp. 0–4, 2011, doi: 10.1109/VETECF.2011.6093099.

[19] M. Abdelsalam and T. Bonny, “IoV Road Safety: Vehicle Speed Limiting System,” *2019 3rd Int. Conf. Commun. Signal Process. their Appl. ICCSPA 2019*, pp. 1–6, 2019, doi:

10.1109/ICCSPA.2019.8713713.

[20] D. Reichardt, M. Miglietta, L. Moretti, P. Morsink, and W. Schulz, "CarTALK 2000: safe and comfortable driving based upon inter-vehicle-communication," pp. 545–550, 2003, doi: 10.1109/ivs.2002.1188007.

[21] R. H. Huang, B. J. Chang, Y. L. Tsai, and Y. H. Liang, "Mobile Edge Computing-Based Vehicular Cloud of Cooperative Adaptive Driving for Platooning Autonomous Self Driving," *Proc. - 2017 IEEE 7th Int. Symp. Cloud Serv. Comput. SC2 2017*, vol. 2018-Janua, pp. 32–39, 2018, doi: 10.1109/SC2.2017.13.

[22] H. Kowshik, D. Caveney, and P. R. Kumar, "Provable systemwide safety in intelligent intersections," *IEEE Trans. Veh. Technol.*, vol. 60, no. 3, pp. 804–818, 2011, doi: 10.1109/TVT.2011.2107584.

[23] W. Wu, J. Zhang, A. Luo, J. Cao, and S. Member, "for Intersection Traffic Control," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 1, pp. 65–74, 2015, [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84919485495&partnerID=tZOtx3y1.

[24] P. Y. Chen, Y. M. Guo, and W. T. Chen, "Fuel-saving navigation system in VANETs," *IEEE Veh. Technol. Conf.*, 2010, doi: 10.1109/VETECF.2010.5594424.

[25] K. Collins and G. M. Muntean, "Route-based vehicular traffic management for wireless access in vehicular environments," *IEEE Veh. Technol. Conf.*, pp. 3–7, 2008, doi: 10.1109/VETECF.2008.261.

[26] K. M. Alam, M. Saini, and A. El Saddik, "Toward social internet of vehicles: Concept, architecture, and applications," *IEEE Access*, vol. 3, pp. 343–357, 2015, doi: 10.1109/ACCESS.2015.2416657.

[27] T. A. Butt, R. Iqbal, S. C. Shah, and T. Umar, "Social Internet of Vehicles: Architecture and enabling technologies," *Comput. Electr. Eng.*, vol. 69, no. May, pp. 68–84, 2018, doi: 10.1016/j.compeleceng.2018.05.023.

[28] N. Magaia *et al.*, "Industrial Internet of Things Security enhanced with Deep Learning Approaches for Smart Cities," pp. 1–14.

[29] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Commun. Surv. Tutorials*, 2015, doi: 10.1109/COMST.2015.2444095.

[30] J. Cheng, J. Cheng, M. Zhou, F. Liu, S. Gao, and C. Liu, "Routing in internet of vehicles: A review," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 5, pp. 2339–2352, 2015, doi: 10.1109/TITS.2015.2423667.

[31] B. Mokhtar and M. Azab, "Survey on Security Issues in Vehicular Ad Hoc Networks," *Alexandria Eng. J.*, vol. 54, no. 4, pp. 1115–1126, 2015, doi: 10.1016/j.aej.2015.07.011.

[32] S. Sharma and B. Kaushik, "A survey on internet of vehicles: Applications, security issues & solutions," *Veh. Commun.*, vol. 20, p. 100182, 2019, doi: 10.1016/j.vehcom.2019.100182.

[33] H. Wang, Z. Zheng, S. Xie, H. N. Dai, and X. Chen, "Blockchain challenges and opportunities: a survey," *Int. J. Web Grid Serv.*, vol. 14, no. 4, p. 352, 2018, doi: 10.1504/ijwgs.2018.10016848.

[34] H. N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for Internet of Things: A Survey," *IEEE Internet Things J.*, 2019, doi: 10.1109/JIOT.2019.2920987.

[35] M. B. Mollah *et al.*, "Blockchain for the Internet of Vehicles towards Intelligent Transportation Systems: A Survey," *IEEE Internet Things J.*, vol. 4662, no. JULY, pp. 1–1, 2020, doi: 10.1109/jiot.2020.3028368.

[36] A. Kiayias *et al.*, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake," *Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS'16*, 2017.

[37] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, "Blockchain technologies for the internet of things: Research issues and challenges," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2188–2204, 2019, doi: 10.1109/JIOT.2018.2882794.

[38] A. Dorri, M. Steger, S. S. Kanhere, and R. Jurdak, "BlockChain: A Distributed Solution to Automotive Security and Privacy," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 119–125,

2017, doi: 10.1109/MCOM.2017.1700879.

[39] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, “Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities,” *IEEE Access*, vol. 7, pp. 85727–85745, 2019, doi: 10.1109/ACCESS.2019.2925010.

[40] L. M. Bach, B. Mihaljevic, and M. Zagar, “Comparative analysis of blockchain consensus algorithms,” 2018, doi: 10.23919/MIPRO.2018.8400278.

[41] “Leased Proof of Stake | Waves documentation.” https://docs.waves.tech/en/blockchain/leasing#leasing-benefits-for-the-node-owner (accessed Nov. 11, 2020).

[42] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” *Proc. Symp. Oper. Syst. Des. Implement.*, 1999, doi: 10.1145/571637.571640.

[43] E. Buchman, “Tendermint: Byzantine Fault Tolerance in the Age of Blockchains,” 2016.

[44] “dBFT 2.0 Algorithm.” https://docs.neo.org/docs/en-us/tooldev/consensus/consensus_algorithm.html (accessed Nov. 11, 2020).

[45] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, “Proof of activity: Extending bitcoin’s proof of work via proof of stake [extended abstract],” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 3, pp. 34–37, 2014, doi: 10.1145/2695533.2695545.

[46] “Proof of burn - Bitcoin Wiki.” https://en.bitcoin.it/wiki/Proof_of_burn (accessed Nov. 11, 2020).

[47] P4Titan, “Slimcoin: A Peer-to-Peer Crypto-Currency with Proof-of-Burn,” *Whitepaper*, 2014.

[48] K. Karantias, A. Kiayias, and D. Zindros, “Proof-of-Burn,” 2020, doi: 10.1007/978-3-030-51280-4_28.

[49] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, “On security analysis of proof-of-elapsed-time (PoET),” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10616 LNCS, no. May 2019, pp. 282–297, 2017, doi: 10.1007/978-3-319-69084-1_19.

[50] F. Tschorsch and B. Scheuermann, “Bitcoin and beyond: A technical survey on decentralized digital currencies,” *IEEE Commun. Surv. Tutorials*, 2016, doi: 10.1109/COMST.2016.2535718.

[51] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, “PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain,” *CEUR Workshop Proc.*, vol. 2058, pp. 1–11, 2018.

[52] “What is POI | NEM Documentation.” https://docs.nem.io/en/gen-info/what-is-poi (accessed Nov. 12, 2020).

[53] M. Milutinovic, W. He, H. Wu, and M. Kanwal, “Proof of luck: An efficient blockchain consensus protocol,” *arXiv*, pp. 2–7, 2017.

[54] A. Shoker, “Sustainable blockchain through proof of exercise,” *2017 IEEE 16th Int. Symp. Netw. Comput. Appl. NCA 2017*, vol. 2017-Janua, pp. 1–9, 2017, doi: 10.1109/NCA.2017.8171383.

[55] K. Christidis and M. Devetsikiotis, “Blockchains and Smart Contracts for the Internet of Things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016, doi: 10.1109/ACCESS.2016.2566339.

[56] K. Croman *et al.*, “On Scaling Decentralized Blockchains Initiative for CryptoCurrencies and Contracts (IC3),” *Int. Conf. Financ. Cryptogr. Data Secur.*, pp. 106–125, 2016, [Online]. Available: http://fc16.ifca.ai/bitcoin/papers/CDE+16.pdf.

[57] F. Casino, T. K. Dasaklis, and C. Patsakis, “A systematic literature review of blockchain-based applications: Current status, classification and open issues,” *Telemat. Informatics*, vol. 36, no. May 2018, pp. 55–81, 2019, doi: 10.1016/j.tele.2018.11.006.

[58] B. A. Tama, B. J. Kweka, Y. Park, and K. H. Rhee, “A critical review of blockchain and its current applications,” *ICECOS 2017 - Proceeding 2017 Int. Conf. Electr. Eng. Comput. Sci. Sustain. Cult. Herit. Towar. Smart Environ. Better Futur.*, pp. 109–113, 2017, doi: 10.1109/ICECOS.2017.8167115.

[59] D. Di Francesco Maesa and P. Mori, “Blockchain 3.0 applications survey,” *J. Parallel Distrib. Comput.*, vol. 138, pp. 99–114, 2020, doi: 10.1016/j.jpdc.2019.12.019.

[60] Q. K. Nguyen, “Blockchain-A Financial Technology for Future Sustainable Development,” 2016, doi: 10.1109/GTSD.2016.22.

[61] M. Mettler, “Blockchain technology in healthcare: The revolution starts here,” 2016, doi: 10.1109/HealthCom.2016.7749510.

[62] A. Ekblaw, A. Azaria, J. D. Halamka, A. Lippman, and T. Vieira, “A Case Study for Blockchain in Healthcare: ‘MedRec’ prototype for electronic health records and medical research data White Paper MedRec: Using Blockchain for Medical Data Access and Permission Management IEEE Original Authors,” 2016.

[63] “The Secure Mobile Voting Platform Of The Future - Follow My Vote.” https://followmyvote.com/ (accessed Nov. 16, 2020).

[64] B. Vote, S. Asset, S. Contract, B. Count, and T. Borda, “BitCongress Whitepaper,” p. 10, 2015, [Online]. Available: http://www.bitcongress.org/BitCongressWhitepaper.pdf.

[65] “Walmart Blockchain Pilot Aims to Make China’s Pork Market Safer - CoinDesk.” https://www.coindesk.com/walmart-blockchain-pilot-china-pork-market (accessed Nov. 16, 2020).

[66] E. Mengelkamp, B. Notheisen, C. Beer, D. Dauer, and C. Weinhardt, “A blockchain-based smart grid: towards sustainable local energy markets,” in *Computer Science - Research and Development*, Feb. 2018, vol. 33, no. 1–2, pp. 207–214, doi: 10.1007/s00450-017-0360-9.

[67] F. Lin *et al.*, “Survey on blockchain for internet of things,” *J. Internet Serv. Inf. Secur.*, vol. 9, no. 2, pp. 1–30, 2019, doi: 10.22667/JISIS.2019.05.31.001.

[68] G. Rathee, A. Sharma, R. Iqbal, M. Aloqaily, N. Jaglan, and R. Kumar, “A blockchain framework for securing connected and autonomous vehicles,” *Sensors (Switzerland)*, vol. 19, no. 14, pp. 1–15, 2019, doi: 10.3390/s19143165.

[69] X. Wang, P. Zeng, N. Patterson, F. Jiang, and R. Doss, “An improved authentication scheme for internet of vehicles based on blockchain technology,” *IEEE Access*, vol. 7, pp. 45061–45072, 2019, doi: 10.1109/ACCESS.2019.2909004.

[70] J. Gao *et al.*, “A Blockchain-SDN-Enabled Internet of Vehicles Environment for Fog Computing and 5G Networks,” *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4278–4291, 2020, doi: 10.1109/JIOT.2019.2956241.

[71] M. Kamal, G. Srivastava, and M. Tariq, “Blockchain-Based Lightweight and Secured V2V Communication in the Internet of Vehicles,” *IEEE Trans. Intell. Transp. Syst.*, pp. 1–8, 2020, doi: 10.1109/tits.2020.3002462.

[72] C. Technology, “MICAz: Wireless Measurement System,” *Prod. Datasheet*, pp. 4–5, 2008, [Online]. Available: http://courses.ece.ubc.ca/494/files/MICAz_Datasheet.pdf.

[73] “VANET Toolbox: A Vehicular Network Simulator based on DES - File Exchange - MATLAB Central.” https://www.mathworks.com/matlabcentral/fileexchange/68437-vanet-toolbox-a-vehicular-network-simulator-based-on-des?s_tid=srchtitle (accessed Nov. 19, 2020).

[74] “5G Toolbox Documentation.” https://www.mathworks.com/help/5g/index.html (accessed Nov. 19, 2020).

[75] “LTE Toolbox Documentation.” https://www.mathworks.com/help/lte/index.html (accessed Nov. 19, 2020).

[76] “WLAN Toolbox Documentation.” https://www.mathworks.com/help/wlan/index.html (accessed Nov. 19, 2020).

[77] “The Network Simulator - ns-2.” https://www.isi.edu/nsnam/ns/ (accessed Nov. 18, 2020).

[78] “ns-3 | a discrete-event network simulator for internet systems.” https://www.nsnam.org/ (accessed Nov. 18, 2020).

[79] “EstiNet Network Simulator and Emulator (NCTUns).” http://nsl.cs.nctu.edu.tw/NSL/nctuns.html (accessed Nov. 18, 2020).

[80] “Veins.” https://veins.car2x.org/ (accessed Nov. 18, 2020).

[81] “OMNeT++ Discrete Event Simulator.” https://omnetpp.org/ (accessed Nov. 18, 2020).

[82] “SUMO Documentation.” https://sumo.dlr.de/docs/ (accessed Nov. 18, 2020).
[83] “Announcing Hyperledger Besu – Hyperledger.” https://www.hyperledger.org/blog/2019/08/29/announcing-hyperledger-besu (accessed Nov. 18, 2020).
[84] “Hyperledger Burrow – Hyperledger.” https://www.hyperledger.org/use/hyperledger-burrow (accessed Nov. 18, 2020).
[85] “Hyperledger Fabric – Hyperledger.” https://www.hyperledger.org/use/fabric (accessed Nov. 18, 2020).
[86] “Hyperledger Iroha – Hyperledger.” https://www.hyperledger.org/use/iroha (accessed Nov. 18, 2020).
[87] “Hyperledger Sawtooth – Hyperledger.” https://www.hyperledger.org/use/sawtooth (accessed Nov. 18, 2020).
[88] “Ethereum Whitepaper | ethereum.org.” https://ethereum.org/en/whitepaper/#ethereum (accessed Nov. 18, 2020).
[89] “How does Ethereum work, anyway?. Introduction | by Preethi Kasireddy | Medium.” https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369 (accessed Nov. 18, 2020).
[90] “Home | Corda Documentation.” https://docs.corda.net/ (accessed Nov. 18, 2020).
[91] “Tezos | Get Started.” https://tezos.com/get-started/ (accessed Nov. 18, 2020).
[92] “ethash | Ethereum Wiki.” https://eth.wiki/en/concepts/ethash/ethash (accessed Sep. 07, 2021).
[93] A. Dorri, S. S. Kanhere, and R. Jurdak, “Towards an optimized blockchain for IoT,” *Proc. - 2017 IEEE/ACM 2nd Int. Conf. Internet-of-Things Des. Implementation, IoTDI 2017 (part CPS Week)*, pp. 173–178, 2017, doi: 10.1145/3054977.3055003.
[94] P. Golle, S. Jarecki, and I. Mironov, “Cryptographic primitives enforcing communication and storage complexity,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 2357, pp. 120–135, 2003, doi: 10.1007/3-540-36504-4_9.
[95] “Documentation - The Go Programming Language.” https://golang.org/doc/ (accessed Sep. 15, 2021).
[96] “Zeroes of Polynomial Functions | Boundless Algebra.” https://courses.lumenlearning.com/boundless-algebra/chapter/zeroes-of-polynomial-functions/ (accessed Jul. 23, 2021).
[97] “ethereum/go-ethereum: Official Go implementation of the Ethereum protocol.” https://github.com/ethereum/go-ethereum (accessed Sep. 17, 2021).